Etude d'un langage intermédiaire pour la compilation d'Algol 60 -Application à un calculateur de type microprogrammé : CAE 510

Jean Le Palmec

N° D'ORDRE

THÈSES

Présentées à la Faculté des Sciences

de l'Université de Grenoble

pour obtenir le grade de

DOCTEUR - INGENIEUR

Par

Jean Le Palmec

INGENIEUR DES ARTS ET METIERS

INGENIEUR I. M. A. G.

PREMIERE THESE

Etude d'un langage intermédiaire pour la compilation d'algol 60

APPLICATION A UN CALCULATEUR DE TYPE MICROPROGRAMME : CAE 510

DEUXIEME THESE

PROPOSITIONS DONNEES PAR LA FACULTE

Thèse soutenue le 1966 devant la commission d'examen :

Monsieur J. KUNTZMANN Président

Messieurs B. VAUQUOIS

N. GASTINEL **Examinateurs**

L. BOLLIET

FACULTE DES SCIENCES

LISTE DES PROFESSEURS

DOYENS HONORAIRES :

M. FORTRAT P.
M. MORET L.

DOYEN :

M. WEIL L.

PROFESSEURS TITULAIRES :

MM.	NEEL L.	Magnetisme
	HEILMANN R.	Chimie Organique
	KRAVTCHENKO J.	Mecanique Rationnelle
	CHABAUTY C.	Mathematiques Pures
	PARDE M.	Potamologie
	BENOIT J.	Radioelectricité
	CHENE M.	Chimie Papetière
	BESSON J.	Electrochimie
	WEIL L.	Thermodynamique
	FELICI N.	Electrostatique
	KUNTZMANN J.	Mathematiques Appliquées
	BARBIER R.	Géologie Appliquée
	SANTON L.	Mécanique des Fluides
	OZENDA P.	Botanique
	FALLOT M.	Physique Industrielle
	GALVANI O.	Mathématiques
	MOUSSA A.	Chimie Nucléaire et Radioactivité

	TRAYNARD P.	Chimie Générale
	SOUTIF M.	Physique Générale
	CRAYA A.	Hydrodynamique
	REULOS R.	Théorie des Champs
	AYANT Y.	Physique Approfondie
	GALISSOT F.	Mathématiques Pures
Melle	LUTZ E.	Mathématiques Générales
MM.	BLAMBERT M.	Mathématiques
	BOUCHEZ	Physique Nucléaire
	LLIBOUTRY L.	Géophysique
	MICHEL R.	Géologie et Minéralogie
	BONNIER E.	Métallurgie
	DESSAUX G.	Physiologie Animale
	PILLET E.	Electrotechnique
	DEBELMAS J.	Géologie Générale
	GERBER R.	Mathématiques Pures
	PAUTHENET R.	Electrotechnique
	VAUQUOIS B.	Calcul Electronique
	SILBER R.	Mécanique des Fluides
	MOUSSIEGT J.	Electronique
	BARBIER J.C.	Physique Expérimentale
	BUYLE-BODIN M.	Electronique
	KOSZUL J.L.	Mathématiques
	DREYFUS B.	Thermodynamique
	VAILLANT F.	Zoologie
	KLEIN J.	Mathématiques Pures
	SENGEL P.	Zoologie
	ARNAUD P.	Chimie
	BARJON R.	Physique Nucléaire
	BARNOUD F.	Biosynthese de la Cellulose

PROFESSEURS ASSOCIES :

MM.	WAGNER H.	Botanique
	NAPP-ZINN K.	Botanique

PROFESSEURS SANS CHAIRE :

Mme	KOFLER L.	Botanique
	DEPASSEL R.	Mécanique
	PERRET R.	Servomecanisme
Mme	BARBIER M.J.	Electrochimie
MM.	COHEN J.	Physique
	GIDON P.	Géologie
Mme	SOUTIF J.	Physique Générale
MM.	GIRAUD P.	Géologie
	GASTINEL N.	Mathématiques Appliquées
	LACAZE A.	Thermodynamique
	GLENAT R.	Chimie Organique
	BRISSONNEAU P.	Physique Générale
	DUCROS P.	Minéralogie
	ANGLES D'AURIAC	Mécanique des Fluides
	ROBERT A.	Chimie Papetière
	COUMES A.	Electronique
	PEBAY-PEROULA	Physique
	DEGRANGE C.	Zoologie
	GAGNAIRE D.	Chimie Papetière
	RASSAT A.	Chimie
	PERRIAUX J.	Géologie
	BARRA J.	Mathématiques Appliquées

PROFESSEURS HONORAIRES

MM.	FORTIER A.	Mécanique des fluides
	BRELOT M.	Mathematiques
	WOLFERS F.	Physique
	DORIER A.	Zoologie

MAITRES DE CONFERENCES

MM.	BIAREZ J.	Mécanique des fluides
	DODU J.	Mécanique des fluides
	DOLIQUE J.M.	Electronique
	HACQUES G.	Mathématiques Appliquées
	LANCIA R.	Physique Automatique
	POULOUJADOFF M.	Electrotechnique
	KAHANE A.	Physique
Mme	BONNIER J.	Chimie
Mme	KAHANE J.	Physique
MM.	DEPORTES C.	Chimie Minerale
	DEPOMMIER P.	Physique Nucléaire
	CAUQUIS G.	Chimie Générale
	BONNET G.	Physique
Mme	BOUCHE L.	Mathématiques
MM.	COLOBERT L.	Physiologie Animale
	PAYANT J.J.	Mathématiques
	CAUBET J.P.	Mathématiques
	LAURENT P.J.	Mathématiques Appliquées
	BERTRANDIAS J.P.	Mathématiques Appliquées
	BRIERE G.	Physique
	LAJZEROWICZ J.	Physique
	VALENTIN J.	Physique
	DESPRE P.	Metallurgie
	BONNETAIN L.	Chimie Minérale

MAITRES DE CONFERENCE ASSOCIE :

M. RADELLI L. Géologie

MAITRE DE CONFERENCE HONORAIRE :

M. GASTEX A. Essais électriques

Ce travail a été réalisé au Laboratoire de Calcul de l'Université de Grenoble dans le cadre d'un contrat avec la Compagnie Européenne d'Automatisme Electronique.

Je tiens à exprimer ma profonde reconnaissance

A Monsieur le Professeur KUNTZMANN,
Directeur de l'Institut de Mathématiques Appliquées de Grenoble, qui a bien voulu me faire l'honneur de présider le Jury de thèse,

A Monsieur le Professeur VAUQUOIS,
Directeur du Centre d'Etudes pour la Traduction Automatique, qui a bien voulu faire partie du Jury de thèse,

A Monsieur le Professeur GASTINEL
Directeur de l'Institut de Programmation

et à Monsieur BOLLIET qui ont dirigé ce travail et dont les conseils et les encouragements m'ont permis de le mener à bien.

Que la Compagnie Européenne d'Automatisme Electronique qui a permis la réalisation de ce travail, trouve ici l'expression de ma reconnaissance. Mes remerciements vont particulièrement à Monsieur M. CONSTANTY pour sa précieuse collaboration pratique.

Je remercie également les membres du Laboratoire de Calcul, en particulier Monsieur VANDORPE, Mademoiselle BICAIS et Monsieur MOUNET qui ont permis la réalisation matérielle de cet ouvrage.

TABLE DES MATIERES

ANNEXES

BIBLIOGRAPHIE

C H A P I T R E I

INTRODUCTION

Les langages de programmation se sont éloignés de plus en plus des langages machine des calculateurs. La facilité de rédaction, de compréhension et de mise au point des programmes écrits dans ces langages évolués, permet à l'utilisateur de résoudre aisément ses problèmes sans recourir à un spécialiste. Naturellement ces programmes ne peuvent être exécutés directement, et l'intermédiaire indispensable entre le programmeur et la machine est assuré par un programme spécial, le compilateur.

Les premiers langages étaient étudiés en fonction d'un calculateur déterminé. Par la suite on les a définis indépendamment des machines, pour qu'ils deviennent aussi de véritables moyens d'échange et de communication. On a cherché, en leur donnant la souplesse et la puissance nécessaire, un langage à vocation universelle.

Algol défini par le Revised Report on Algol 60 [1] joue ce rôle dans le domaine du calcul scientifique.

Le développement d'Algol a entraîné de nombreuses recherches théoriques sur les langages et leur compilation. Ces recherches et l'expérience acquise avec Algol influeront sur les langages futurs et probablement sur la structure des machines. Nous n'avons pas l'intention d'examiner l'étendue de ces problèmes, mais d'exposer de manière assez détaillée le compilateur ALGOL 60 des calculateurs CAE 510 (ou 530), et le langage intermédiaire qu'il utilise.

1.1. Généralités

Il serait logique d'appeler traducteur ou générateur, le programme qui transforme le programme source, c'est-à-dire le texte écrit par le programmeur en ALGOL, en un programme résultat ou programme objet qui sera plus directement l'origine de l'éxécution.

En effet il est écrit dans un langage objet (ou généré) qui est soit

a) le langage machine : le programme objet est directement exécutable.

b) un langage intermédiaire :

1) un langage d'assembleur qu'il suffit alors d'assembler pour se ramener au cas précédent.

ou 2) un langage dit d'Interpréteur : un ensemble de programmes permettent d'exécuter le programme objet sans le transformer, de manière interprétative conforme aux exigences du texte source ALGOL. Ces programmes nécessaires à l'exécution du programme objet constituent l'Interpréteur.

Même lorsque le langage objet est le langage machine, la nature d'Algol impose de recourir à des programmes spéciaux pour résoudre certains problèmes à l'exécution. Aussi devrait-on réserver le mot compilateur à l'ensemble traducteur - sous programmesnécessaires à l'exécution du programme objet, ou plus généralement au système qui permet d'évaluer le texte Algol.

En fait les sous programmes interprétatifs sont relativement peu importants lorsque le langage objet n'est pas un langage d'Interpréteur, et l'on confond assez naturellement compilateur et traducteur ; nous ferons souvent la même confusion dans le cas d'un langage objet d'interpréteur, car le travail du traducteur est fréquemment assez voisin.

En effet on ne peut pas considérer de limite très nette entre les deux méthodes. Dans les méthodes interprétatives on résoud généralement pour des raisons d'efficacité, le maximum de problèmes par une traduction, et l'exécution du programme objet est par certains aspects, la simulation d'une machine de structure nouvelle.

1.1.1. Quelques aspects interprétatifs d'ALGOL

Nous rappellerons certains des problèmes posés par Algol, et qui ne peuvent être résolus qu'à l'exécution. Le plus important résulte de la notion de portée des identificateurs, c'est-à-dire de la localisation des quantités, et de la récursivité. L'allocation de mémoires ne peut être traitée de façon statique que si l'on rejette et les tableaux à bornes variables, et les appels récursifs de procédure : le compilateur traducteur peut alors attribuer effectivement des adresses d'implantation avant l'exécution. On notera que la localisation des quantités permet une optimisation statique des mémoires allouées.

Dans les procédures, le remplacement des formels par les paramètres effectifs fait intervenir l'exécution de 2 façons

1) il est lié à la récursivité

2) même si le formel est spécifié, ce qui permet une traduction ou une prétraduction, il est souhaitable de vérifier la correspondance dans les appels ; en outre on pourra déterminer à ce moment les transferts de type Réel-Entier.

Ainsi lorsque la production du programme objet est caractérisée par la réservation effective des mémoires nécessaires à l'évaluation, et c'est le cas des compilateurs actuels générant en langage machine, il est nécessaire de faire appel à des sous programmes interprétatifs d'exécution si l'on ne veut pas réduire le langage source à un sous ensemble trop limité d'Algol. En particulier ils résolvent de cette manière l'implantation des tableaux à bornes variables, le calcul de l'adresse d'un élément de tableau (en vérifiant éventuellement le nombre et la valeur des indices). Il faudrait procéder ainsi pour les problèmes de correspondance de paramètres effectifs cités précédemment.

En outre ces compilateurs ne permettent pas une véritable récursivité. Elle est totalement interdite, s'il n'y a pas de sous programme interprétatif d'appel de procédure, et reste limitée dans le cas contraire ; c'est une méthode d'empilage-désempilage qui peut alors leur permettre une solution assez satisfaisante.

1.1.2. Problèmes techniques

La traduction du programme source en programme objet implique toute une série de transformations effectuées en 1 seul ou plusieurs passages. La production d'un nouveau programme, par application de 1 ou plusieurs transformations successives sur des parties du programme source que l'on traite progressivement dans sa totalité, constitue un passage de traduction.

Ce nouveau programme sera le programme source pour le passage ultérieur. Il peut être écrit dans un véritable langage intermédiaire, ou ne différer du précédent que par une optimisation. En effet on peut distinguer 2 tâches

1) faciliter et simplifier le traitement suivant
2) apporter des informations particulières au programme traité, obtenir un programme objet plus efficace.

Outre ces avantages, latraduction en plusieurs passages permet de scinder le traducteur en sous programmes indépendants ; ils n'ont donc pas besoin d'être simultanément présents en machine. Remarquons que s'ils ne donnent pas lieu à des sorties sur support extérieur, les passages successifs restent pratiquement inaperçus de l'utilisateur.

La traduction nécessite une analyse de la structure du programme Algol, et cette analyse syntaxique constitue le problème majeur de la Compilation. Nous ne ferons qu'évoquer certaines distinctions.

1) Les méthodes élémentaires travaillent au niveau des symboles de base du programme Algol ;ainsi :

a) la méthode des paires de délimiteurs ou de Samelson et Bauer. Une pile permet l'analyse de la structure de parenthèses d'Algol, et retient les informations non encore traduites. Une matrice de transition contrôle la traduction : le délimiteur au sommet de la pile et le délimiteur courant pointé dans le texte source déterminentle sous programme à exécuter.

b) les méthodes des priorités. A chaque délimiteur est attachée une priorité. Cette méthode est déjà plus naturelle (Algol précise les niveaux de priorités pour les opérateurs arithmétiques), et donne un traitement simple des expressions. On utilise un pile pour les délimiteurs ; la comparaison des priorités du délimiteur en pile et du délimiteur pointé dans le programme source commande l'empilage ou le désempilage. Si le traitement des expressions est ainsi automatique, une partie de l'analyse de la structure du programme peut se faire de manières assez analogues à la méthode précédente, par association des délimiteurs correspondants.

Dijkstra a utilisé cette technique dans le compilateur Algol pour le X1 [3].

On notera que la priorité des délimiteurs, si elle dépend de la syntaxe, variera légèrement pour certains, selon les options de construction du compilateur. De même on peut utiliser une priorité différente pour un délimiteur en pile et dans le programme (double priorité de certains délimiteurs), ce qui facilite le traitement.

2) Méthodes syntaxiques. La syntaxe d'Algol apparaît plus nettement décrite par le compilateur ; il est composé de programmes qui correspondent approximativement aux différentes unités syntaxiques. C'est le cas de la méthode de Grau, ou méthode des états syntaxiques, où une pile retient les "états" des sous programmes successivement activés. Un "état" caractérise l'état du compilateur à l'entrée d'un sous programme, et permet de poursuivre le traitement lorsque l'on sort de ce sous programme. Le test du délimiteur courant, pointé dans le texte source, et de l'"état" au sommet de la pile, détermine, par l'intermédiaire d'un élément d'une table de transition, le sous programme à exécuter. Pour l'essentiel, ce dernier met à jour la pile et génère la partie deprogramme objet, correspondant à l'unité syntaxique traitée.

La méthode d'Evans, Perlis et Van Zoeren, très différente est basée sur la transformation du programme source en "threaded-list" grâce à des sous programmes appelés "recognizers" qui correspondent généralement aux unités syntaxiques.

3) Les méthodes"syntax-directed"

La compilation traduit le programme source en terme des unités syntaxiques qui le composent, et travaille sur des représentations de ces unités syntaxiques. Ceci conduit à des compilateurs indépendants de la syntaxe, en ce sens que la définition du langage peut être donnée au compilateur.

Le plus connu, celui d'Irons, reçoit en entrée le programme source, et les règles de traduction. Chaque règle comporte une formule syntaxique (la définition) et une description sémantique (instructions à générer, autres actions..). Le traducteur utilise les formules syntaxiques pour analyser le texte source, et produit une structure de liste indiquant les formules utilisées. Ce résultat permet de générer, grâce à l'information sémantique correspondant à chaque formule, un programme en langage d'assembleur.

1.2. Le Compilateur Algol sur CAE 510-530

La construction de 3 compilateurs Algols'achevait à l'Université de Grenoble, lorsque l'étude du compilateur Algol sur CAE 510 fut commencé en février 1963. Dans l'ordre chronologique :

. le compilateur CAB 500 de G. Werner [5] : il génère en un seul passage directement en langage machine, et utilise une méthode d'états syntaxiques (opérationnel en janvier 1964).

. Le compilateur IBM 7090-7040 de J.C. Boussard [6.], (T.A. Dolotta et M. Berthaud). La génération du programme objet en langage d'assembleur est traitée en 2 passages par l'Editeur [4] et le générateur ; celui-ci utilise une méthode de priorité (opérationnel sur 7090/44 en octobre 1963, sur 7040/44 en janvier 1964).

. Le compilateur GAMMA 60 de C. Gerbaux, Y. Siret et G. Durand assez proche du précédent (opérationnel en janvier 64).

La génération du programme objet en langage machine caractérise ces 3 compilateurs, d'où des restrictions de langage (1.1.1.)

Par ailleurs le passage unique du compilateur petite machine (CAB 500) amène quelques restrictions (comme la nécessité de déclarer un identificateur avant de l'utiliser dans une expression), et les compilateurs grosse machine (IBM 7090 - GAMMA 60) conduisent à des encombrements importants, dus à la méthode statique, à la structure des ordres générés, tout en nécessitant l'assemblage ultérieur des programmes ; pratiquement il faut un calculateur de dimension suffisante pour intégrer compilateur et assembleur dans un système d'exploitation.

Il fallait implanter un calculateur sur une petite machine (mots de 18 bits, version minimum 8 K), les CAE 510-530 dont la logique d'utilisation est la microprogrammation : on ferait nécessairement appel à des microprogrammes lors de l'exécution, en particulier pour les opérations.

Aussi fut il naturel de chercher une autre solution, et de développer un compilateur dans le sens de la méthode de Dijkstra dont nous connaissions les principes. Cela permit :

1°) un langage source très complet, où les restrictions sont peu nombreuses.

2°) l'implantation du compilateur sur la version minimum du CAE 510/530 (8192 mots, perforateur et lecteur rapide, flexowriter ou télescriptrice)

En effet la méthode de Dijkstra diminue l'encombrement du traducteur en reportant certaines taches à l'exécution, et de ce côté n'accroît que la taille de bibliothèque que nous avons de toute façon. Indépendamment de la technique interprétative qui a diminué la dimension du traducteur, celui ci est scindé en 2 programmes indépendants : l'Editeur et le Générateur ; reprise dans une version où le travail de codification est plus important, la technique du premier passage d'édition qu'avait définie T.A. Dollota [4] donne des dimensions équivalentes aux deux parties du traducteur. On a donc obtenu un compilateur constitué de 3 programmes indépendants se succèdant en mémoire : Editeur, Générateur, Interpréteur. La segmentation a non seulement facilité l'implantation du compilateur sur petite machine, mais rendu possible une exploitation commode du type "load and go" (ch. 14) grâce à l'équilibre d'encombrement que l'on est parvenu à réaliser entre les 3 parties (2.8.3., ch. 12). La méthode apportait en outre la généralité du langage. Nous espérions que la rançon de ce choix ne serait pas une augmentation exagérée du temps d'exécution, puisque la microprogrammation est l'utilisation habituelle du calculateur ; l'expérience l'a sensiblement confirmé.

Nous indiquerons rapidement que le constructeur d'un compilateur voit un très grand intérêt à la méthode interprétative. En effet le problème de compilation ne repose plus entièrement sur le traitement du traducteur avant l'exécution, comme c'est pratiquement le cas pour un traducteur statique classique. L'évaluation du programme, scindée en 2 phases, est prise en charge par 2 programmes. Le traducteur transforme essentiellement le texte source en notation post fixée de forme adaptée, que l'Interpréteur évalue grâce à une pile.

Le travail du générateur s'il reste par ailleurs le même que celui d'un générateur classique, est facilité par la structure simple des ordres du langage objet, et est soulagé de la gestion des mémoires de manoeuvre (et du type) prise en charge par l'Interpréteur, et donc aussi de la génération des ordres correspondants. Les tâches de compilation ne sont plus imbriquées mais réparties ici sur 3 programmes : la mise au point de l'ensemble est plus facile. En outre, le fait même que l'Interpréteur doit gérer dynamiquement les mémoires de manoeuvre, conduit à lui faire gérer dynamiquement toute la mémoire, c'est-à-dire à traiter la récursivité : dans l'optique du constructeur ce traitement de lui-même, sans augmentation sensible de son travail.

Nous pensons que ces caractéristiques nous ont facilité la réalisation d'un compilateur, opérationnel dans des délais raisonnables, depuis sa conception jusqu'à l'achèvement (février 63 - avril 64).

C'est à notre connaissance le premier compilateur de ce type construit en France. Nous ne disposions que des principes exposés par Dijkstra ([2] et [3]) sur sa méthode interprétative, et il fallait élucider très en détail et complètement cette technique ; à ce problème, la définition du langage objet définitif ajoutait la recherche d'un compromis entre efficacité d'exécution et l'équilibre souhaité des 3 parties du compilateur : cela ne rendait pas leur conception indépendante. Aussi il nous parut préférable de ne rien arrêter de vraiment définitif, au delà des premières lignes directrices adoptées pour le programme objet (production et exécution), mais de prolonger la conception pendant la réalisation pratique menée sur 2 parties, au moins, à la fois : Editeur- Générateur, puis Générateur-Interpréteur. Cela nous a donné la souplesse nécessitée par le manque d'expérience en ce domaine, alors que maintenant on définirait sans doute totalement les caractéristiques du langage objet. Aussi la construction de ce compilateur a été dans toutes ses phases de conception, programmation, mise au point, un travail très intéressant.

La programmation, aspect pratique, représente environ 9000 mots (procédures standard exclues), et la CAE a bien voulu se charger d'adapter elle-même sa bibliothèque de microprogrammes dans le cadre précis du compilateur (entrée-sortie, opérations arithmétiques et logiques, procédures standard) (cf. ch. 14).

Depuis l'intérêt de la méthode de Dijkstra n'a cessé d'être démontré par les nombreux travaux qu'elle à suscités. Elle a été très développée et est maintenant largement connue. Citons simplement parmi les seuls travaux de l'Université de Grenoble :

- l'intérêt pédagogique du Compilateur Algol simplifié écrit en Algol de J. Cohen et M. Brasseur [9]
- le compilateur Algol IBM 1130 de O. Lecarme et C. Belissant, en cours d'achèvement ; son langage objet et l'interpréteur sont inspirés du compilateur CAE.
- le compilateur Algol Phillips PR8000 de J. du Masle ; opérationnel depuis mars 1966, il est proche du Whestone Compiler de Randell B. et Russel L.J. [8].

Nous donnerons quelques indications sur le Whestone Compiler, car c'est une expérience contemporaine du compilateur CAE, et décrite dans le seul ouvrage exhaustif, à notre connaissance, sur un compilateur dérivé de la méthode de Dijkstra. Lors de sa parution en Angleterre (1964) le compilateur CAE était opérationnel, et la comparaison de ces deux expériences séparées serait intéressante. Le KDF9 possède 2 compilateurs Algol : le Kidgsgrove Compiler est multipassage, optimisé et génère en langage machine ; le Whestone Compiler utilisé surtout à la mise au point de programme, ce qui a influé sur sa conception, est dérivé de la méthode de Dijkstra. Son langage objet est proche de celui que nous avons sur le CAE 510 : cela montre bien que les conclusions faites à partir du même document [2] sont dans les deux cas assez voisines. Le Whestone Compiler permet une segmentation : les segments doivent

être des procédures où tous les identificateurs utilisés sont soit des formels, soit locaux au corps de procédure. Le traitement des étiquettes est au niveau de l'exécution plus facile à comprendre, mais peut être moins efficace en moyenne que celui du Compilateur CAE (cela est certain dans le cas usuel).

Le traitement des instructions POUR n'est pas très simple et est moins efficace ; ainsi, chaque instruction contrôlée par un Pour devient un bloc et aucun cas d'optimisation n'est reconnu (le cas le plus fréquent est optimisé dans le compilateur CAE). Par ailleurs le rôle de l'Interpréteur semble plus important ; il doit par exemple vérifier le type dans les expressions, et de celles ci dans les instruction. En effet les traducteurs Whestone et CAE sont différents ; le traducteur Whestone est en un seul passage. Le test du bon usage des identificateurs demande un traitement complexe (aucune hiérarchie n'est imposé dans les déclarations) et ne décèle complètement que les confusions Variable simple-Tableau-Procédure. Il est assez directement inspiré du traducteur décrit par Dijkstra [3] et fonctionne par priorité et "stack" ; c'est en majeure partie un ensemble de sous programmes correspondants aux différents délimiteurs, et qui utilisent le "stack" comme pile d'opérateurs et de variables d'états ; celles-ci enregistrent des informations sur les parties déjà examinées du texte source.

Nous regrettons de n'avoir pu présenter une rédaction aussi complète sur le compilateur ALGOL CAE à la même époque.

Seuls deux chapitres concerneront ici l'Editeur et le Générateur ; deux notes techniques leur ont été déjà consacrées [7].

C H A P I T R E II

MECANISME D'EXECUTION DES PROGRAMMES OBJETS :

ADRESSAGE DYNAMIQUE

Le mécanisme d'exécution du programme et la définition du langage objet, sont une adaptation de la méthode de Dijkstra([2], [3])

Elle peut se caractériser par l'utilisation optimum de la mémoire de travail pendant l'exécution. Cette mémoire de travail appelée "stack" permet :

1) Les réservations dynamiques : chaque fois que le déroulement du programme conduit à pénétrer dans un nouveau niveau de nomenclature, les mémoires qui seront affectées aux valeurs des variables locales de ce niveau, sont réservées par le programme à cet instant précis de l'exécution.

2) Au-delà des réservations d'un niveau, la mémoire est toujours utilisée en pile ordinaire pour les évaluations que nécessitent les instructions du niveau. On dira que cette zone de travail, où se déroulent les calculs intermédiaires, est composée d'accumulateurs. Un accumulateur est un groupe de 3 mémoires (dans notre cas) pouvant contenir un résultat et les informations liées à celui-ci (valeur, Adresse...)C'est le mode normal d'utilisation de la zone de travail.

La gestion dynamique des mémoires de manoeuvre entraîne le fait suivant : si n opérandes d'une opération ne sont pas définis explicitement, ils sont implicitement disponibles au sommet du "stack" dans les n derniers accumulateurs occupés ; l'opération libère ces accumulateurs et place le résultat, s'il existe, au sommet du stack.

Nous allons donner quelques précisions sur les points principaux.

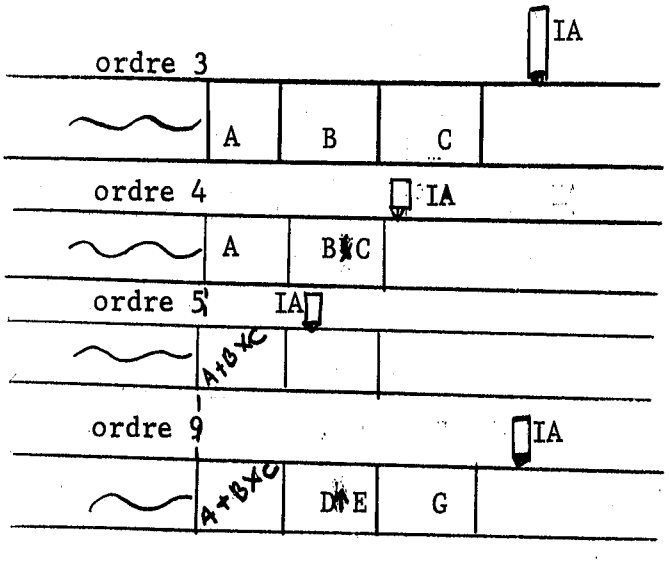
2.1. Expressions et Instruction d'Affectation

Toutes les quantités sont évaluées dans l'ordre du Texte Source, le traitement des effets de bord dans le cas des fonctions est alors bien défini (cf. 2.4.5.)

En effet l'évaluation d'une expression est programmée selon son écriture en notation post-fixée. Le fonctionnement du stack à l'exécution est le reflet de cette écriture.

Exemple : A + B x C - D↑E/G sera écrite ABC x + DE↑ G/-
et le programme objet sera

1	Charger	(en stack) la valeur de	A
2	"	" "	B
3	"	" "	C
4		Multiplier	
5		Additionner	
6	Charger	valeur de	D
7	"	"	E
8		Exponentier	
9	Charger	valeur de	G
10		Diviser	
11		Soustraire	



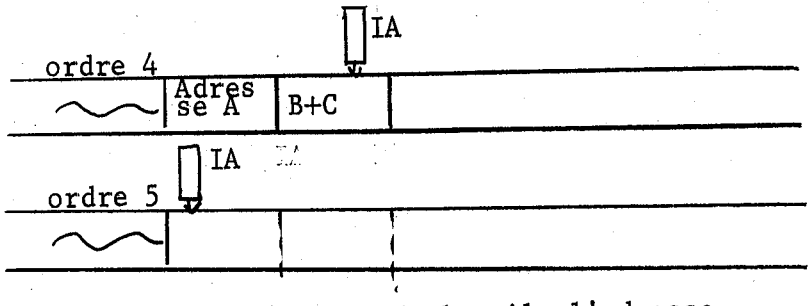
Le sommet de la pile est contrôlé par l'index IA (Index d'Accumulateur) pointant le premier accumulateur libre.

L'ordre "Charger" consiste à mettre une valeur au sommet de la pile ; IA progresse de 1. Les ordres "Multiplier",.., "Soustraire" consistent à faire ces opérations sur les opérandes contenus dans les accumulateurs repérés par IA-2, IA-1, et mettre le résultat en IA-2. L'accumulateur IA-1 est alors

libre ; le sommet de la pile s'est déplacé et IA est regréssé de 1. Les figures indiquent l'état du stack après les différents ordres.

L'instruction d'affectation se traduit de façon analogue. Elle peut être considérée en effet, comme une opération sur 2 opérandes, avec un résultat implicite . Le programme suivant correspondra à l'instruction A := B+C, soit en post-fixé ABC + :=

1	Stacker Adresse	A
2	Charger	B
3	Charger	C
4	Additionner	
5	Ranger	



L'ordre "Stacker Adresse A" signifie, mettre au sommet de la pile l'adresse physique (réelle en machine) de la mémoire attribuée à A.

L'ordre "Ranger" s'exécute en rangeant la valeur contenue dans l'accumulateur IA-1, à l'adresse donnée dans IA-2, puis libère ces 2 accumulateurs en régressant IA de 2.

Nous examinerons ultérieurement (ch. 5) les détails du problème d'affectation (affectation multiple).

2.2. Réservation dynamique.

Les déclarations localisent des quantités dans un bloc : en dehors elles n'ont pas d'existence, elles sont inaccessibles. C'est en ce sens que le bloc introduit un nouveau niveau de nomenclature (4.1.3. de [1]). Si des mémoires doivent être attribuées à ces quantités, il est intéressant de ne les affecter que pour la durée du bloc. Le compilateur peut calculer et attribuer les adresses pour qu'il en soit ainsi : la réservation est dite statique.

Une réservation dynamique, c'est-à-dire faite à l'exécution lorsqu'on pénètre dans un bloc, permettra déjà de rendre indépendantes les réservations de chaque bloc ; l'inexécution d'un bloc n'alourdit pas les réservations nécessaires et l'utilisation générale de la mémoire. Mais l'avantage le plus important réside dans la solution des problèmes de récursivité.

Réservations et calculs intermédiaires se font dans la mémoire organisée en pile. Aussi lorsque un bloc est appelé récursivement (par l'intermédiaire d'une procédure récursive), on créera dans la pile de nouvelles mémoires pour ses variables locales, et de nouvelles mémoires de manoeuvre pour les calculs intermédiaires, à chaque appel. La séparation nécessaire des réservations de mémoires qui correspondent à des blocs dynamiquement différents lors des activations récursives, est automatiquement assurée.

Les procédures comme les blocs introduisent un niveau. Nous verrons qu'elles sont traitées de façon identique. On a des ordres d'entrée de niveau qui comportent en opérande (direct ou non) l'encombrement des mémoires à affecter aux variables locales de ce niveau. L'index IA repérant le sommet de la pile est déplacé pour en tenir compte, et son ancienne valeur est affectée à l'Index DP (Début Paramètres). Ainsi les réservations se font à l'exécution à partir de la position couramment libre du stack de travail ; l'origine de cette implantation est conservée dans DP pour les raisons que nous verrons plus loin (2.3.2.).

2.2.1. Données de liaisons.

Les données de liaisons doivent être considérées comme des variables locales implicites de chaque niveau de nomenclature. Au nombre de 5 elles permettent d'assurer de façon générale les communications entre niveaux, et le déroulement du programme.

Une mémoire est réservée à chacune et ce sont les 5 premières mémoires d'un niveau. Nous examinerons leur nature à mesure que leur rôle apparaîtra nécessaire.

2.3. Niveau de Nomenclature et Variables locales.

Nous nous bornons pour l'instant, au cas élémentaire où les variables locales sont les variables simples déclarées dans un bloc.

Un numéro de bloc (ou numéro lexicographique) est attaché à chaque niveau de nomenclature et dans l'ordre lexicographique. Il augmente de 1 à chaque entrée de bloc, et diminue de 1 à chaque sortie. Le programme a un numéro de bloc égal à 0. Pour éviter toute particularisation de traitement c'est toujours pour nous un niveau de nomenclature ; s'il est une instruction composée dans le texte source il est transformé en bloc (sans déclaration).

Deux blocs disjoints peuvent avoir même numéro de bloc. Notons que nous interdisons d'étiqueter le Programme (restriction d'Algol).

Si le bloc B_N est lexicographiquement inclus dans le bloc B_{N-1}, la portée des variables locales de B_{N-1} s'étend à B_N où elles sont globales. Au cours de l'exécution de B_N, les variables locales de tous les blocs incluant B_N, doivent être accessibles.

2.3.1. Adressage dynamique des variables.

Dans le programme objet une variable sera référée :

a) par le numéro de bloc caractérisant le niveau où elle est locale.

b) par une adresse relative à l'intérieur de ce niveau, par rapport à l'origine des mémoires locales à ce niveau.

Numéro de bloc et adresse ralative constituent l'adresse dynamique. Les premières mémoires locales étant réservées pour les 5 données de liaison (2.2.1.), les adresses relatives des variables sont attribuées à partir de l'adresse 5.

Exemple

```
PROG : DEBUT ENTIER A,B,C ;
|          BA : DEBUT ENTIER D,E ;
|          |    BB : DEBUT ENTIER F ;
|          |    |    BC : DEBUT ENTIER G,H ;
|          |    |    |       Ik ;
|          |    |    |    FIN ;
|          |    |    FIN ;
|          |    FIN ;
|  BD : DEBUT ENTIER L ;
|  |       BE : DEBUT ENTIER M ;
|  |       |       I1 ;
|  |       FIN ;
|  FIN ;
|  FIN ;
```

Numéros de blocs

PROG	0
BA	1
BB	2
BC	3
BD	1
BE	2

Adresses dynamiques

	Numéro de bloc	Adresse Relative
A	0	5
B	0	6
C	0	7
D	1	5
E	1	6
F	2	5
G	3	5
H	3	6
L	1	5
M	2	5

2.3.2. Problèmes d'Adressage.

Avoir accès aux variables dont la portée s'étend à l'instruction I, c'est connaître l'origine d'implantation des mémoires locales de chacun des niveaux que l'on a successivement pénétrés, pour parvenir à cette exécution particulière de I. Nous dirons qu'il faut connaître l'implantation actuelle de la structure de blocs à laquelle appartient I.

Remarquons que cette structure est parfaitement définie d'après le texte source.

Exemple ; dans l'exemple précédent

Ik est dans la structure de blocs PROG, BA, BB, BC

Il " " PROG, BD, BE

Cette notion diffère de l'enchevêtrement dynamique des structures qui s'introduit du fait des procédures.

Nous dirons maintenant "Structure de blocs" pour l'implantation de la structure de blocs.

La valeur affectée à la 1ere donnée de liaison en pénétrant dans un niveau (numéro de bloc) N, est l'adresse de l'origine d'implantation des mémoires locales du niveau de numéro N-1.

Pour le dernier niveau ouvert, les index NB et DP contiennent respectivement le numéro de bloc et l'adresse origine des mémoires locales.

Aussi dans le cas des blocs le traitement est simple : entrer dans un bloc, c'est passer du niveau N-1, au niveau N. (On rappelle que, d'où on ne peut entrer dans un bloc que par le DEBUT, b° toute instruction Allera conduisant à une étiquette hors d'un bloc implique l'équivalent d'une FIN). Au moment où l'on exécute l'entrée bloc, l'Index DP contient la valeur DP_{N-1} que l'on affecte à la première des mémoires localescréées pour le bloc. Cette mémoire est alors pointée par IA, index dont la valeur est ensuite consignée dans DP en tant que DP_N. NB est progressé de 1.

. La première donnée de liaison et l'Index DP permettent donc de retrouver dans le stack, par une chaîne de reprise, les origines d'implantations DP_{N-1}, DP_{N-2},... DP_0 représentatives de la structure de blocs : l'index DP pointe la mémoire donnant l'adresse DP_{N-1}, qui donnera DP_{N-2}...etc. (Voir fig. 2.3.4.).

2.3.3. Display.

C'est la table des valeurs successives prises par DP pour la structure de blocs. Soit $DP_0,\ldots,DP_N$ précédemment décrite. Elle simplifie les calculs fréquents d'adresse réelle d'une variable. L'exploitation des adresses dynamiques se fait à partir du display : le numéro de bloc K permet d'obtenir directement dans cette table, DP_K, sans avoir à remonter la chaîne de reprise dans le stack ; à cette origine d'implantation du niveau K, on ajoutera l'adresse relative.

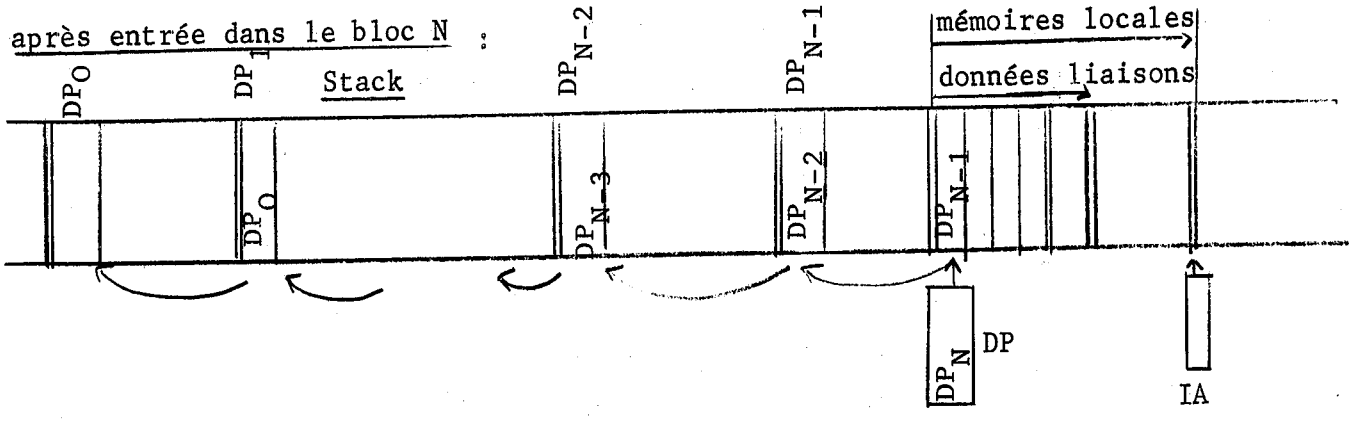
L'index NB pointe (à une constante près d'implantation) la dernière valeur entrée dans le Display.

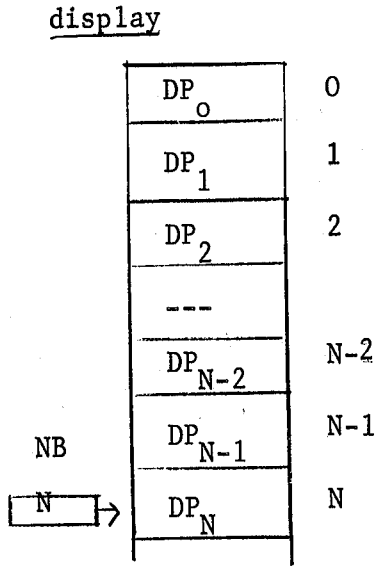
La duplication de la chaîne $DP_0\ldots.DP_N$ est automatique pour les blocs en mémorisant à chaque entrée, après progression de NB, la nouvelle valeur de DP à l'adresse pointée par NB.

2.3.4. Remise à jour du Display.

La donnée des valeurs de NB et DP pour les niveaux K, soient $NB_K(=K)$ et DP_K, caractérise sa structure de blocs : en effet on peut retrouver facilement la configuration correspondante du display. La valeur actuelle DP_K de DP est affectée à la mémoire du display que refère NB ; les valeurs DP_{K-1}, $DP_{K-2},\ldots,DP_0$ constituant la chaîne pointée par DP dans le stack de travail (2.3.2.) sont successivement recopiées en remontant dans le display. Le processus est arrêté lorsque K valeurs sont entrées.

La remise à jour du Display est nécessaire, comme nous le verrons, au sortir d'une procédure ou pour l'exploitation d'un paramètre effectif.





2.3.5. Traitement dynamique et procédure.

Nous avons vu en partie les principes et le mécanisme d'utilisation des blocs et des réservations dynamiques. Abordons plus complètement les raisons du traitement. Il est commode de distinguer deux aspects dans la notion de niveau de nomenclature, introduite par un bloc source algol, c'est-à-dire dans la notion de signification locale d'un identificateur :

- aspect statique
- aspect dynamique.

2.3.5.1. L'aspect statique concerne la reconnaissance de la structure lexicographique de bloc du programme source, faite dès la compilation (ch. 12). C'est déterminer la portée des identificateurs et y effectuer le remplacement par la représentation accordée parallèlement selon les déclarations. Nous dirons que l'on assure ainsi une protection statique des identificateurs.

Les représentations seront de la forme B,N où N particularise la quantité dans la base B, et sera un numéro (tout du moins nous le verrons comme tel). La base peut être unique pour tout le programme et pour toutes les catégories de quantités.

Lors de l'exécution la représentation donnera accès à la valeur associée à la signification locale de la quantité. On peut dire en effet que les étiquettes, aiguillages et procédures désignent une valeur qui est une adresse de programme objet ; la sémantique indique bien qu'il s'agit de la désignation explicite du successeur d'une instruction, de la réécriture d'une instruction.

Nous considérons donc, dans un premier temps, que la valeur associée dans tous le cas à une quantité, est contenue dans la mémoire d'adresse machine F(B)+N (F fonction d'adresse à préciser). C'est pourquoi on dit souvent au lieu de représentation, adresse ou même mémoire.

On notera que la protection statique des identificateurs peut se faire en minimisant le nombre de représentations dans le programme, donc le nombre de mémoires nécessaires, si l'on peut redéfinir la signification, c'est-à-dire affecter la valeur de la quantité locale correspondante, au plus tard avant l'exécution de la première instruction du bloc. A la compilation il suffit de réutiliser les représentationslibérées à la sortie du bloc. A ce point du traitement il faut différencier les variables et les autres quantités. En effet :

- la valeur est connue pour les étiquettes, aiguillages et procédures dès la compilation et ne changera pas ; c'est du moins le cas pour une traduction sans segmentation. Optimiser les représentations ne présente aucun intérêt. Elles désigneraient des mémoires dont le nombre serait sans doute optimisé, mais auxquelles il faudrait affecter une valeur déjà détenue dans le programme. Ce ne serait efficace ni en temps ni en place. Aussi la représentation sera la valeur ou son adresse dans le programme.

- La valeur d'une variable n'est pas définie à l'entrée d'un bloc ; la redéfinition, c'est-à-dire l'affectation de valeur ne se pose pas. Il suffit d'être certain que les mémoires désignées par les représentations soient disponibles. Si la valeur était implicitement définie de façon simple et uniforme à l'entrée d'un bloc (par exemple 0), le problème serait d'ailleurs analogue. On optimisera donc les mémoires des variables, et cela peut se traduire de diverses manières dans les représentations. Dans notre cas, on a vu que chaque bloc s'associait à la base B indiquée par le numéro lexicographique du bloc ; l'optimisation est réalisée en affectant les mêmes numéros de blocs, donc les mêmes bases à des blocs disjoints. Mais on peut procéder différemment et rapporter tous les blocs inclus dans le bloc BL à la base utilisée par BL ; par exemple on choisira des blocs-bases qui seront associés à la base indiquée par leur numéro lexicographique de bloc-base défini comme suit : le programme a le numéro 0, le numéro augmente de 1 et diminue de 1 en entrant et en sortant d'un bloc-base. Dans l'exemple 2.3.1. en choisissant BA et BD comme blocs-bases on aurait les représentations suivantes :

PROG	A ⟶ 0	5
	B ⟶ 0	6
	C ⟶ 0	7
BD	L ⟶ 1	5
BE	M ⟶ 1	6

BA	D ⟶ 1	5
	E ⟶ 1	6
	F ⟶ 1	7
BB	G ⟶ 1	8
BC	H ⟶ 1	9

Tous ces systèmes sont équivalents d'un point de vue statique : si les réservations et la correspondance représentation-mémoire machine sont faites avant l'exécution, ils donnent le même nombre de mémoires. Les mémoires sont réservées par base. Lorsque ce traitement précède l'exécution, il faut connaître le maximum atteint dans une base pour toutes ses utilisations, d'où l'intérêt d'une base unique pour le programme ; c'est ainsi que procèdent les générateurs statiques (sinon ils devraient utiliser un assembleur, et chaque représentation différente serait une réservation individualisée).

Traité à l'exécution le problème est différent ; il suffit de connaître et de réserver le maximum lié à l'utilisation particulière d'une base, au moment où l'on pénètre dans un bloc-base. Dans notre cas il y aura effectivement le minimum de mémoires réservées à un moment donné. Tout autre système réserverait en entrant dans un bloc-base des mémoires qui ne seront utiles qu'aux blocs intérieurs ; mais il permettrait un gain de temps d'exécution : opérations de réservation moins fréquentes, calcul des adresses réelles plus rapides si moins nombreuses, les origines d'implantation des bases peuvent être contenues dans des index machine. De ce point de vue une base unique pour tout le programme, en dehors des corps de procédure (nous verrons plus loin la raison), est encore plus favorable : la représentation est une adresse statique. Van der Poel a réalisé un système de ce type [10].

Les procédures introduisent un problème nouveau.

Nous rappellerons qu'activer une Procédure c'est insérer le corps de procédure à la place del'appel de procédure et l'exécuter, après avoir fait les remplacements impliqués par la règle de réécriture (4.7.3. du Rapport Algol [1])

Règle de Réécriture.

Point 1 Affectation d'une valeur (appel par valeur). Tous les formels figurant en partie valeur reçoivent la valeur du paramètre effectif ; l'affectation doit être considérée comme effectuée explicitement avant d'entrer dans le corps de procédure. Tout se passe comme si l'on créait un bloc fictif incluant le corps de procédure, et que l'on fasse les affectations à des variables locales de ce bloc fictif. En conséquence les variables appelées par valeur doivent être considérées comme locales au bloc fictif et non au corps de procédure.

Point 2 Remplacement du nom (appel par nom)

Le formel est remplacé dans tout le corps de procédure par le paramètre effectif correspondant, après avoir mis entre parenthèses ce dernier partout où cela est syntaxiquement possible. Des conflits possibles entre identificateurs introduits par ce remplacement et d'autres identificateurs déjà présents dans le corps de procédure seront systématiquement évités par des changements convenables des identificateurs formels ou locaux en litige.

Point 3 Enfin le corps de procédure ainsi modifié est inséré à la place de l'instruction procédure. Si la procédure est appelée d'un endroit en dehors de la portée d'une quantité non locale du corps de procédure, les conflits entre identificateurs introduits par cette insertion du corps de procédure, et les identificateurs dont la déclaration est valable à cet endroit seront systématiquement évités pas des changements convenables de ces derniers.

Point 4 En outre le corps de procédure lui-même "se conduit toujours comme un bloc qu'il en ait la forme ou non" (5.4.3. de [1]). D'où :

1°) la déclaration de procédure introduit un niveau de nomenclature et il faut la compiler en tant que bloc en vue de l'exécution. Le bloc fictif du point 1 est effectivement crée et contient comme variables locales les paramètres appelés par valeur ; nous l'appellerons bloc-procédure. Les symboles PROCEDURE et ";" de fin de déclarationde procédure jouent le rôle de DEBUT et FIN. Le point 1 l'impose bien : si le bloc-procédure n'était pas créé mais confondu avec le niveau corps de procédure, le résultat différerait lorsque le corps étiqueté se rappelle.

Exemple :

```
PROCEDURE F(X) ; VALEUR X ; REEL X ;
E : DEBUT ENTIER I,J ;.....
      SI B ALORS ALLERA E ;
    FIN DE F ;
```

Le niveau de nomenclature du bloc-procédure est nécessaire aussi pour une autre raison : il faut y attacher tous les formels ce qui protège les identificateurs locaux du corps de procédure (point 4).

D'autre part le formel n'est pas remplacé (appel par nom) à l'exécution, par le paramètre effectif ; on accède à celui-ci, par l'intermédiaire de la représentation du formel : elle désigne une mémoire où l'appel considéré aura placé la définition du paramètre effectif. Le point 2 est satisfait, et les conflits d'identificateurs sont évités très simplement. Comme dans le cas des paramètres appelés par valeur, la représentation désigne en fait une mémoire locale (ou plutôt un groupe) du bloc-procédure. Les formels se conduisent alors pour la compilation de façon analogue à des variables locales.

2°) Le corps de procédure introduit un niveau de nomenclature ; le considérer comme un bloc revient ici à protéger, vis à vis des seuls formels (du fait de la présence du bloc-procédure), les étiquettes étiquetant le corps de

procédure, et s'il n'est pas un bloc, les étiquettes internes au corps. Il suffit de traiter un bloc fictif incluant le corps procédure, sans en faire un bloc-base quelque soit le système adopté : le bloc fictif utilise la même base que le bloc procédure.

En outre si le corps de procédure est un bloc non étiqueté, son exécution nepeut se rappeler sans qu'il ne s'agisse de récursivité : la nouvelle exécution fait alors partie d'une nouvelle activation du bloc-Procédure. Il n'est donc pas intéressant de considérer ce bloc corps de procédure comme un bloc-base : on utilisera la même base que pour le bloc-Procédure. Dans une méthode dynamique on fera l'économie d'opérations (réservations et libérations à l'exécution) systématiquement entraînées par celles du bloc-Procédure.

3°) Le problème de représentations introduit par les déclarations de procédure est différent de celui des blocs : le corps de procédure (inclus dans le bloc-procédure créé) peut être inséré et exécuté en tout point du bloc B où la procédure est déclarée ; la portée de la déclaration peut cependant ne pas s'étendre entièrement à B. Tout identificateur présent dans le corps lors de la déclaration, représente soit un formel ou une quantité locale de corps, soit une quantité dont la portée s'étend à cet endroit, c'est-à-dire déclarée dans le bloc B ou un bloc incluant B (cf. 54.3. de [1]). L'appel de la procédure en dehors de la portée d'une quantité non locale A du corps, est un appel en un endroit d'un bloc BB où une nouvelle déclaration de A lui a donné une nouvelle signification. Le bloc BB est naturellement inclus dans le bloc B, aussi les conflits évoqués au point 3 n'existent pas : les deux A ont des représentations différentes grâce au traitement des blocs, et dans le bloc-procédure la représentation de A signifie bien la quantité dont la portée s'étend en tête de B.

Lorsqu'on exécute le bloc-procédure, la représentation d'une quantité locale du corps, de même celle d'un formel, doit avoir la signification précisée dans la procédure et sans confusion avec une quantité déclarée dans BB, si l'on insère le bloc-procédure dans le bloc BB inclus dans B. C'est là le problème nouveau ; sa solution est différente suivant la méthode employée, dynamique ou statique, c'est-à-dire selon que la signification peut être redéfinie ou non au niveau des bases à l'exécution.

D'un point de vue statique, il est nécessaire que les représentations accordées dans le bloc-procédure aient une signification permanente dans le bloc B de déclaration de la procédure. Si l'on choisit le bloc-procédure comme bloc-base, la base qu'il utilise ne pourra être réaffectée qu'à la sortie de B.

Au contraire avec une méthode dynamique, on peut réaffecter la base utilisée dès la sortie du bloc-procédure. A la compilation il se conduit alors comme un bloc ordinaire choisi comme base, mais son exécution diffèrera par les problèmes d'enchaînement : la signification de la représentation de ses quantités locales vient cacher des définitions données dans des blocs inclus dans B et incluant l'endroit où l'on insère ce bloc-procédure ; à sa sortie ces dernières redeviennent valables. Il y a donc ici un problème particulier de redéfinition : dans un bloc ordinaire, elle n'intervient que lorsque toute signification antérieure est définitivement perdue.

Nous avons vu l'influence sur le mode de représentation, de la liaison statique ou dynamique établie entre le représentation donnée à la compilation et la signification à l'exécution. Pour une raison déjà indiquée cela ne concerne que les variables simples dans lesquelles on comprendra les formels et les variables associées aux tableaux (cf. ch. 6). Nous verrons maintenant qu'un bloc-procédure doit être un bloc base, et qu'il faut donner un sens dynamique à la base utilisée.

2.3.5.2. Aspect dynamique de la signification locale d'un identificateur

La récursivité donne un sens dynamique à la règle de réécriture du corps de procédure et à son exécution. Si une première activation de P provoque elle même une seconde activation de P, cela implique la réécriture du bloc-procédure à l'endroit où on donne la seconde activation, c'est-à-dire à l'intérieur du même bloc-procédure inséré précédemment dans le programme pour le premier appel. Les quantités locales de cette première activation, dont le déroulement est suspendu par la seconde, sont des quantités dont la déclaration n'est pas valable dans le nouveau bloc-procédure exécuté. On est ramené au problème analogue déjà évoqué : l'utilisation de la même base dans le bloc-procédure et à l'endroit où on l'insère ; cela concernait alors des blocs inclus dans le bloc de déclaration, et la compilation pouvait éviter ce problème en donnant des représentations différentes. Cette possibilité n'existe plus maintenant où l'on insère le bloc-procédure dans le bloc procédure : la représentation d'un formel ou d'une de ses quantités locales est unique. Il est donc nécessaire de traiter la base utilisée par le bloc-procédure en dynamique ; par la même occasion on pourra utiliser cette même base dans un bloc choisi comme bloc-base et inclus dans le bloc B de déclaration, en particulier dans un bloc de procédure différente déclarée en tête de B. La mise en oeuvre est facilitée par le traitement général.

La récursivité justifie l'adressage dynamique où la représentation (B,N) désigne la variable locale associée à la mémoire d'adresse machine F(B) + N, F(B) étant définie comme la dernière implantation de la base B dans une pile : chaque fois que l'on pénètre dans un niveau réutilisant la base B, elle est implantée à partir de la première position libre dont l'adresse est la valeur F(B) considérée par la suite. Le compilateur attribue les bases et les représentations en respectant la structure de blocs, de telle manière qu'à l'exécution le traitement dynamique est simple ; le display peut constituer

un enregistrement commode des définitions ; seul subsiste un problème particulier d'enchaînement (redéfinition temporaire) pour les blocs-procédure créés et nécessairement choisis comme blocs bases, qui sera résolu par les données de liaison.

2.3.5.3. Quelques remarques permettront d'indiquer avant d'étudier la mise en oeuvre, que l'évaluation des expressions dans une pile, l'utilisation d'une pile unique et le choix des bases y sont liés pour donner une méthode cohérente.

Les variables locales et les mémoires de manoeuvre posent des problèmes proches ; les mémoires nécessitées par les calculs intermédiaires d'évaluation d'une expression, pourraient être considérées comme celles de variables locales déclarées en tête d'un bloc incluant l'expression qui les implique. Le minimum nécessaire dans une expression donnée, dépend de la méthode d'évaluation ; on minimisera sur l'ensemble des expressions le nombre de ces variables de manoeuvre : leur valeur n'est définie que momentanément durant l'évaluation de l'expression et perdue ensuite. Aussi peut on les voir d'un point de vue statique, comme déclarées en tête du bloc le plus extérieur ; l'utilisation est libre après chaque expression, mais les variables de manoeuvres utilisées dans un bloc procédure ne redeviennent libres qu'à la sortie du bloc B de déclaration de la procédure : l'activation de ce bloc-procédure peut s'intercaler dans l'évaluation de toute expression de B, qu'il soit directement ou non l'évaluation d'un indicateur de fonction ; le problème était voisin pour ses variables locales. Nous ne considèrerons que les algorithmes caractérisés par l'utilisation des mémoires de manoeuvre, en pile : le dernier résultat enregistré dans une mémoire de manoeuvre, est toujours le premier des résultats contenus en mémoires de manoeuvre à être utilisé par la suite. C'est le cas lorsque la traduction dérive directement de la notation post (ou pré)-fixée, en générant chaque opérateur dès qu'il est rencontré dans l'examen séquentiel de cette notation. Ceci permet une optimisation facile des mémoires de manoeuvre, et dans une méthode statique où le compilateur explicite les variables de manoeuvre, il gère une pile. Son sommet indique au cours d'une expression la première

mémoire de manoeuvre disponible, et en fin d'expression il est revenu à sa position initiale ; celle ci est enregistrée à la sortie d'un bloc procédure, et on affecte le maximum atteint sur le bloc-procédure comme nouvelle position initiale : l'ancienne position initiale n'est recouvrée qu'à la sortie du bloc de déclaration de procédure.

Mais la récursivité impose de traiter les variables de manoeuvre nécessaires au bloc-procédure, comme locales à ce bloc. Si le compilateur les explicitent (en appliquant le même principe de pile sur le corps) il doit les classer avec les variables locales du bloc procédure ; de ce point de vue elles auraient des représentations de même forme que les variables locales, et le traitement serait identique à l'exécution. En dehors des procédures les variables de manoeuvre seraient naturellement rapportées au bloc-programme. Ce traitement serait peu efficace en temps et en place : on atteindrait des mémoires de manoeuvre par une adresse dynamique alors que l'on sait qu'elles sont utilisées en pile (ceci découle du type de compilation considéré) et réserverait des mémoires inutiles à un instant donné ; ceci est important lorsque plusieurs activations de procédure s'enchaînent. Aussi n'est-il pas intéressant de les expliciter, mais d'utiliser le mécanisme de pile directement à l'exécution, tel que nous l'avons vu. La mémoire est utilisée en pile unique pour les réservations dynamiques des blocs-bases et l'évaluation des expressions : les mémoires de manoeuvre correspondent donc bien à des variables de manoeuvre locales à l'activation du bloc base au cours de laquelle on les crée. De plus il n'existe effectivement des mémoires locales attachées à une implantation d'un bloc-base, que si l'exécution correspondante est momentanément suspendue par l'évaluation d'un indicateur de fonction. Dans notre cas le programme objet est le reflet exact de la notation post-fixée, et l'évaluation d'une expression effectuée entièrement en mémoire de manoeuvre n'est guère dissociable du traitement dynamique décrit. Ce dernier est aussi très intéressant lorsque les registres de la machine jouent le rôle de mémoires de manoeuvre et que les valeurs des variables ne sont plus toujours prises dans l'ordre d'écriture, si l'on reste dans les conditions fixées ; on simule l'évaluation d'une expression dans une machine où les opérateurs (opérations, transferts) travaillent sur des registres adressables et sur une pile de registres (ou mieux de mémoires).

Dans la méthode suivie chaque bloc source est choisi comme bloc-base en plus des blocs-procédure. On peut y voir plusieurs raisons. Les tableaux à bornes dynamiques ne peuvent être créés qu'en pénétrant dans un bloc à l'exécution, et il faut libérer la place qu'ils occupent en sortant du bloc; ceci nécessite une variable locale implicite du bloc, l'encombrement des tableaux. L'utilisation d'une pile unique où les tableaux sont naturellement implantés, demande que la libération soit immédiate lorsqu'une instruction ALLERA conduit hors du bloc : les expressions sont évaluées dans la pile ; cette situation est reconnue grâce au numéro de bloc, en conservant les blocs où sont déclarés des tableaux comme blocs-base (cf. ch. 7). Le problème serait d'ailleurs le même avec une pile séparée pour les tableaux ; la pile unique présente plus d'avantages, quand pour faciliter la compilation et gagner le maximum de place, tous les blocs du programme sont des blocs-bases. En effet les variables simples et les tableaux à bornes fixes (nombres) pourraient être rapportés dans les procédures aux blocs-procédures et en dehors au bloc-programme. S'il vaut mieux éviter du fait de la récursivité, l'encombrement du niveau bloc-procédure par des réservations utiles aux blocs intérieurs, le traitement adopté présente un inconvénient en dehors des procédures où il n'est pas indispensable : la perte de temps dû à l'adressage dynamique n'est pas toujours compensé par un gain de place très appréciable sur l'adressage statique ; en effet, soient R les réservations dues à un instant donné de l'exécution aux seuls blocs B en dehors des procédures et M le maximum atteint par R : les réservations statiques réserveraient M (on écarte toute condition d'exécution sur un bloc B), et le maximum dynamique des réservations est souvent atteint par des activations de procédure dans un bloc B tel que la différence (M-R) est négligeable devant le maximum dynamique.

La méthode de Dijkstra est donc susceptible de nombreuses variantes. Revenons au traitement suivi.

2.4. Procédures

Chaque déclaration de procédure introduit donc un niveau de nomenclature que le compilateur traduit en créant un bloc.: le bloc-procédure, Son numéro de bloc est celui du bloc de déclaration de la procédure augmenté de 1. Les paramètres formels éventuels de la procédure sont traduits comme des variables locales de ce bloc.

2.4.1. Caractérisation dynamique.

Les formels se comportent dans le programme objet comme des quantités déclarées dans le bloc (-procédure) ; les ordres référant les formels indiquent qu'il s'agit d'un paramètre, et portent sur une adresse dynamique de variable locale de ce bloc ; cette variable locale particulière attachée à chaque paramètre, s'appelle Caractérisation dynamique (Cdy) et est constituée de 3 mémoires dans notre cas. Sa construction qui définit une signification particulière de la représentation d'un paramètre, est effectuée par le mécanisme d'activation de procédure à l'exécution (ch. 9.)

Dans le cas d'un appel par valeur, la Cdy contient la valeur et indique cette situation à l'ordre formel. Elle lui fournit dans le cas d'un appel par nom, les informations d'accès au paramètre effectif : adresse de variable, adresse de sous programme objet (calcul de valeur, d'adresse, d'étiquette), représentation de quantités, et les valeur NB_1 et DP_1 prises par les index NB et DP dans le bloc origine du paramètre effectif ; ces valeurs d'index sont en effet indispensables pour redéfinir la signification des représentations dans un sous programme objet de paramètre effectif par exemple.

L'ordre sur paramètre donnera toujours le même résultat que l'ordre analogue sur une quantité déclarée.

2.4.2. Corps de procédure.

Les règles de compilation habituelles des blocs (numéro et adressage dynamique) s'appliquent aux blocs intérieurs au bloc créé (bloc-procédure) comme à celui-ci.

On a vu que le corps de procédure doit être traité comme un bloc qu'il en ait la forme ou non : il s'agit de la protection des étiquettes locales du corps vis à vis des identificateurs de formels, si ces étiquettes n'appartiennent pas à un bloc du corps. Pour simplifier le traitement dans ce cas, nous avons admis de ne faire de protection que si le corps de procédure est enclos par DEBUT FIN. Il est donc interdit d'utiliser comme étiquette un identificateur de formel, pour étiqueter le corps de procédure, et si celui-ci n'est ni un bloc ni une instruction composée non étiquetée, pour étiqueter une instruction du corps non incluse dans un bloc (une instruction composée contrôlée par une instruction POUR, se comporte à cet égard comme un bloc cf. Ch. 8).

Lorsque le corps de procédure est un bloc non étiqueté, il n'est pas choisi comme bloc base ; ses quantités locales sont rapportées au niveau du bloc-procédure (cf. 2.3.5.1.). Dans le programme objet il n'apparait pas en tant que bloc, mais comme la traduction d'une instruction composée ; elle peut comporter en tête des ordres de création de tableaux, seules traces de déclaration. A la compilation il a bien joué le rôle d'un bloc introduisant un niveau de nomenclature, mais il n'y a pas eu augmentation du numéro de bloc, et ses réservations ont été incluses dans celles du bloc-procédure. C'est le même traitement que l'on fait pour protéger les étiquettes internes d'un corps instruction composée non étiquetée.

2.4.3. Particularités d'activation des procédures

Nous verrons comment est assurée la définition de la signification des représentations des identificateurs locaux du bloc procédure, définition temporaire pour la durée de son exécution, ce que nous appellerons : changer de structure de blocs.

Si les procédures demandent le même traitement d'entrée que les blocs, en particulier la réservation des mémoires locales, la tenue à jour du display et des index IA, NB, DP, les opérations diffèrent légèrement : elles ont été décrites pour les blocs (2.3.2. et 2.3.3.) ; l'ordre d'activation comporte alors directement en opérande les réservations à faire, et le numéro de bloc n'est pas indiqué : il suffit d'ajouter 1 à NB. L'ordre d'activation d'une procédure ne donne pas les réservations à faire : cette information figure en tête du programme objet correspondant à laprocédure (bloc-procédure).
Si N est le niveau du bloc-procédure activé, l'ordre d'activation indique, outre l'adresse du programme objet (adresse d'entrée de la procédure), le numéro de bloc N-1 du niveau de déclaration de la procédure ;il faut en effet le connaître puisque, et c'est le problème particulier d'exécution des blocs-procédure, on passe du niveau K au niveau N, et non K+1 comme pour un bloc ordinaire : on change de structure de blocs ; elles ont évidemment une partie commune, les niveaux 0 à N-1 ($K \geq N$, on est revenu à la structure de blocs du paramètre effectif pour activer une procédure paramètre - cf. ch. 9).

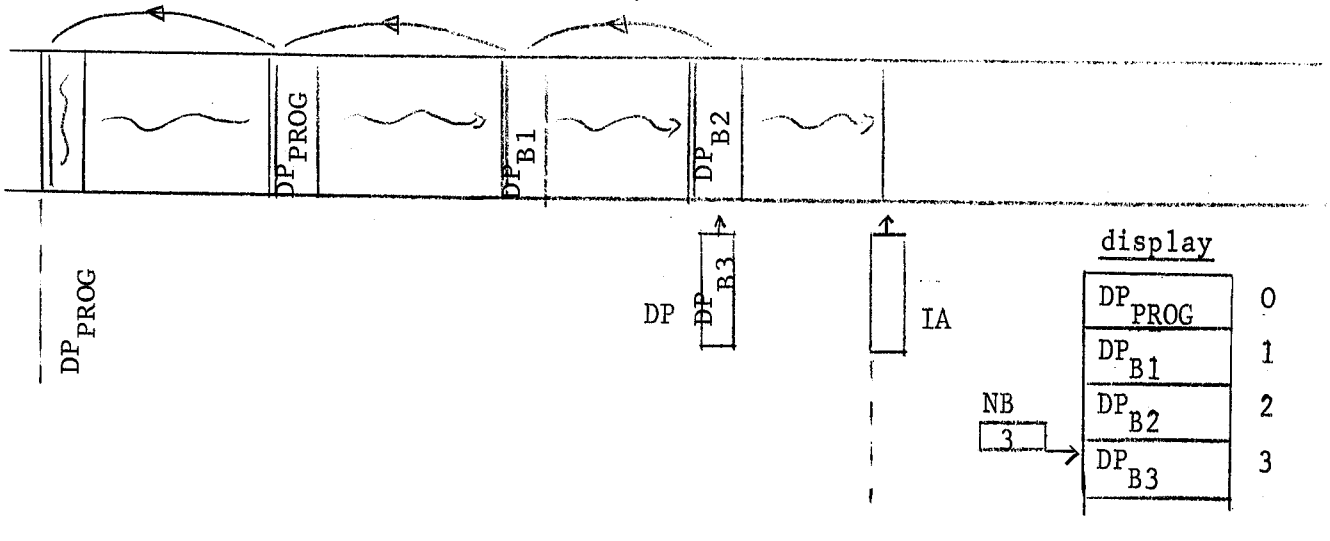
N-1 permet d'obtenir dans le display DP_{N-1} valeur de la première donnée de liaison et de mettre à jour NB (:= N). La mise à jour de DP, du display et de IA, est alors semblable à celles des blocs.

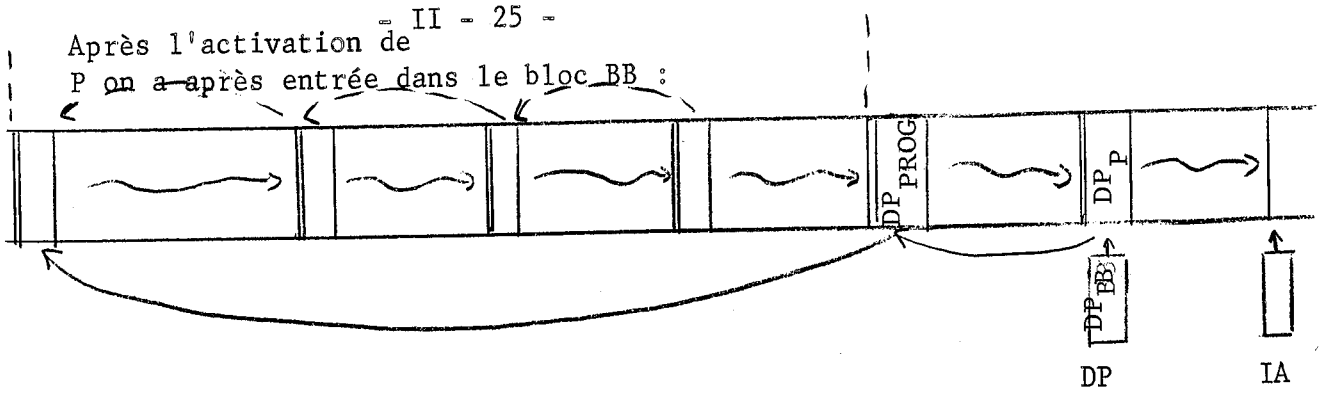
Le display donne la signification des représentations utilisées pendant l'exécution de la procédure ; il reflète ce qu'on pourrait appeler la structure dynamique : elle vient remplacer en partie la structure précédente, et à des endroits variables de l'exécution du programme ; on redéfinit généralement des significations de représentations, au moins pour les formels, puis à mesure que la structure se développe quand on pénètre dans les blocs intérieurs au bloc-procédure. Nous parlerons plutôt de changement de structures de blocs et verrons comment on revient à la structure initiale en fin d'activation (2.5.)

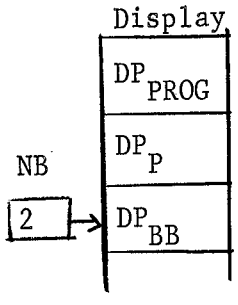
Exemple :

```
PROG : DEBUT ENTIER A ;   PROCEDURE P(X) ; ENTIER X ;
                          DEBUT ....
                          BB : DEBUT ENTIER B ;
                               -------
                               FIN ;
                          FIN ;
       B1 : DEBUT ENTIER B,C ;
            B2 : DEBUT ENTIER D ;
                 B3 : DEBUT ENTIER F,G ;
                      FIN ;   E : P(B) ;
                 FIN ;
       FIN ;
  FIN
```

Au moment où l'on va exécuter E : P(B), la situation est







2.4.4. Indicateur de fonction-Première opération d'une activation.

Lors d'une activation de procédure et avant toute autre opération, on réserve dans le stack :

a) un accumulateur s'il s'agit d'évaluer un indicateur de fonction, et uniquement dans ce cas. Sa valeur y sera rangée et à sa place normale pour poursuive le calcul de l'expression dont elle fait partie (2.1.).

b) une mémoire d'indication de résultat contenant : 0 pour une Instruction procédure, 1 pour une évaluation d'indicateur qui dénote donc la réservation de l'accumulateur précédent.

Lorsque IA reflète cette situation, les opérations d'activation (de bloc ou de procédure) que nous avons précédemment décrites, se déroulent.

Une procédure déclarée <TYPE> PROCEDURE peut être activée par une évaluation d'indicateur de fonction, ou par une instruction Procédure : on ne s'intéresse pas alors à la valeur de l'indicateur. L'alternative est traitée, dans cette méthode, grâce à la mémoire d'indication de résultat. Selon sa

valeur l'ordre donné dans le corps de procédure d'affectation à un identificateur de procédure, décide s'il faut ranger la valeur calculée (2.4.5.)

Remarquons que cette mémoire existe pour toutes les activations (de bloc ou de procédure) : on utilise ainsi le même ordre fin d'activation pour les blocs et procédures, et 2 ordres seulement d'activation pour les procédures ; Instruction Procédure et Evaluation Indicateur de fonction, ne diffèrent d'ailleurs que par les premières opérations.

2.4.5. Affectation de valeur à un Identificateur de Fonction.

L'instruction d'affectation à un identificateur de procédure, utilise un ordre spécial "Ranger Procedure" avec pour opérande le numéro de bloc de la procédure. Grace à ce numéro de bloc, on retrouve dans le display la valeur correspondante de DP, d'où l'adresse (inférieure de 1) de la mémoire indicatrice de résultat. On range alors, ou non, la valeur dans l'accumulateur précédent. IA est régressé et libère l'accumulateur au sommet du stack qui contenait la valeur.

Exemple Si dans le corps de procédure de F, déclarée ENTIER PROCEDURE au niveau 3, on a l'instruction F := A+B elle sera traduite dans le programme objet

Charger	A
Charger	B
Additionner	
Ranger Procédure	(4)

2.5. Fin d'activation de Procédure.

Nous verrons à propos de ce problème, les 3ème 4ème et 5ème données de liaison mises en place à l'activation d'un niveau.

2.5.1. Retour dans le programme:

La procédure d'une part, les ordres d'activation d'autre part, sont à des endroits différents du programme objet. En fin d'exécution de la procédure, il faut donc disposer de l'adresse de retour dans le programme.

Cette adresse de retour après l'ordre d'activation correspondant constitue la 3ème donnée de liaison.

2.5.2. Mise à jour de l'index IA : libération des réservations.

Terminer l'activation d'un niveau, c'est en particulier libérer le stack de travail, de ses mémoires locales.

IA, qui repère le sommet du stack de travail, recouvre sa valeur avant activation : soit DP-1. En effet la valeur de DP, origine des mémoires locales, doit être régressée pour libérer la mémoire indicatrice de résultat (2.4.4.).

2.5.3. Retour dans le bloc appelant.

Activer une procédure c'est changer de structure de blocs. Dans l'exemple de 2.4.3. nous sommes dans la structure PROG, B1, B2, B3, puis nous passons à la structure PROG, P...

Le problème est de revenir à la structure PROG, B1, B2, B3 en fin d'activation de P. C'est-à-dire de mettre à jour :

a) les index NB et DP qui la caractérisent pour la suite de l'exécution.

b) le display puisque l'exploitation de la structure de blocs, pour les adresses dynamiques, est faite à partir de lui.

Aussi les 4ème et 5ème données de liaison mises en place à l'activation d'un niveau sont DP_K et NB_K, origine d'implantation des mémoires locales et numéro de bloc (=K) du bloc qui a appelé. En fin d'activation elles servent à restaurer les Index DP et NB. Le display est alors mis à jour (2.3.4.).

2.5.4. Remarques sur les blocs-Fin d'activation.

Notons que la fin d'un bloc n'interrompt pas le déroulement séquentiel du programme. Aussi la 3ème donnée de liaison (adresse de retour) est fictive et indiquée 0 pour les entrées blocs.

La 1ère donnée de liaison établit par une chaîne, la structure de bloc (notion lexicographique). Au contraire les 3ème et 4ème données de liaison, indiquent des relations fonctions de l'exécution, de passage d'une structure à une autre (2.5.3.). Cependant ces 2 notions sont pratiquement équivalentes pour les blocs : en entrant dans un bloc, ou en sortant, la nouvelle structure de blocs ne diffère de la précédente que par adjonction ou suppression d'un niveau. Aussi les 4ème et 5ème données de liaisons sont en fait inutiles dans les entrées de bloc. En fin d'activation correspondante NB sera régressé de 1 et DP restauré par la 1ère donnée de liaison qu'il référait [Valeur pour le niveau de nomenclature précédent qui est bien celui qui a appelé].

Ceci est réalisable avec un seul ordre Fin d'activation valable pour les procédures et les blocs, puisque de toute façon il distingue ces 2 cas, pour la remise à jour du compteur de programme : il n'est pas modifié si la 3ème donnée de liaison est nulle.

Il est encore plus intéressant, au point de vue efficacité d'exécution, d'avoir 2 ordres : 1 ordre Fin d'activation Procédure et 1 ordre Fin d'activation Bloc ; les entrées correspondantes auront toujours 5 données de liaison pour les procédures, mais seulement les 2 premières données de liaison pour les blocs (2^{de} donnée de liaison cf. 2.6.).

En fait nous utilisons des entrées identiques pour les blocs et les procédures, traitées de façon identique (exceptée l'adresse de retour) par un seul ordre Fin d'activation (2.5.3.). Les variantes citées s'introduisent facilement dans le compilateur (voir par exemple au niveau de l'Interpréteur ch. 11).

2.6. Problème annexe de Remise à jour de IA - Seconde donnée de liaison.

Les progressions ou les régressions de IA, nous l'avons vu, sont systématiques et simples ; à la fin de chaque instruction Algol, IA revient sur lapremière mémoire après les implantations de mémoires locales. En ce sens il est tenu à jour automatiquement.

Mais les instructions ALLERA posent un problème de mise à jour de IA, si elles conduisent en dehors du niveau N, où elles sont données, au niveau P ($P < N^*$). IA doit alors prendre la valeur ZT_P, adresse de la Zone de Travail du niveau P, ou adresse de la première mémoire après les réservations locales du niveau P.

On pourrait y parvenir par la chaîne des DP, avec quelques problèmes. Aussi préfère-t'on ajouter une donnée de liaison. La 2^{de} donnée de liaison du niveau P est l'adresse de sa zone de travail ZT_P.

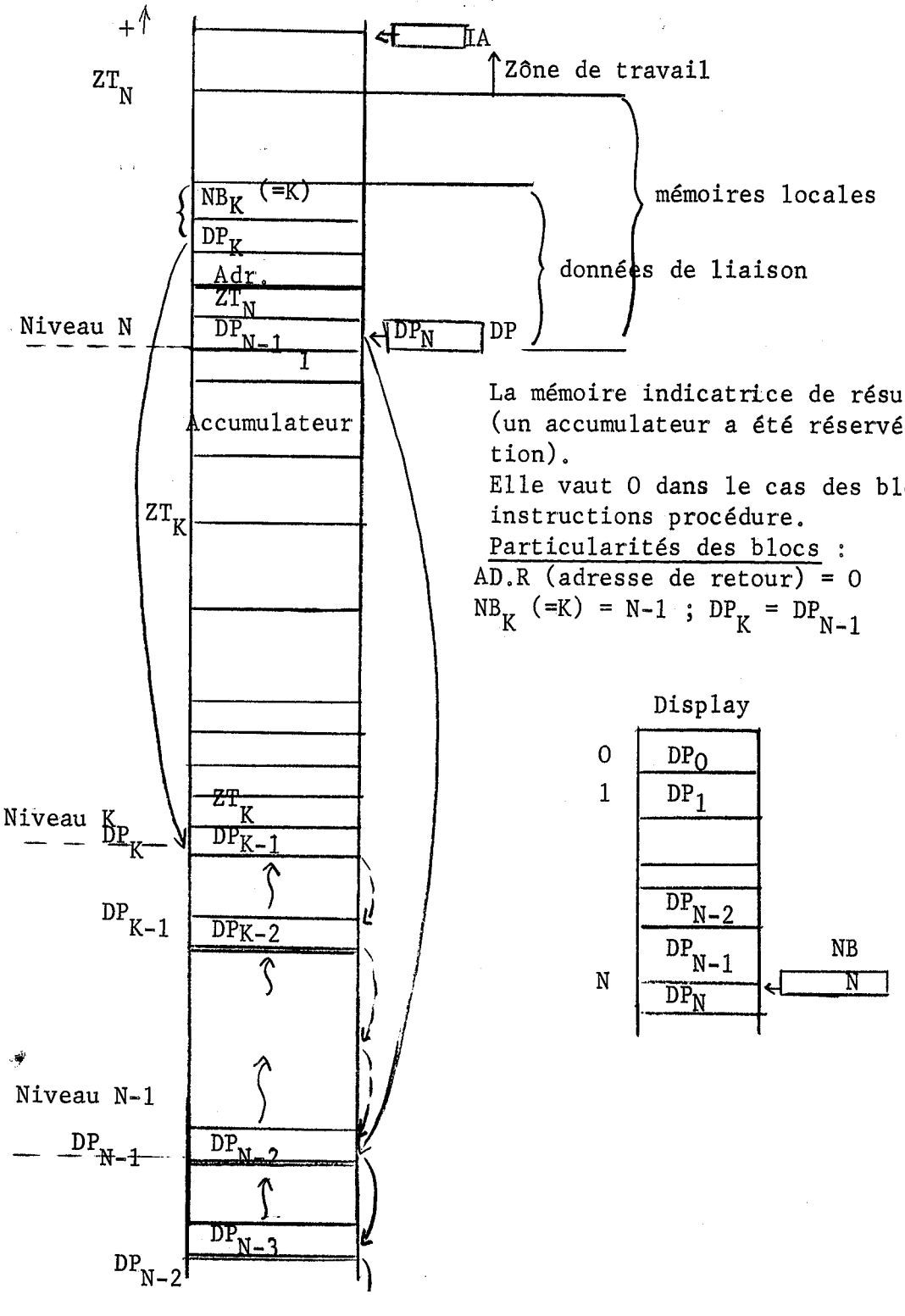
Le numéro de bloc qui accompagne une étiquette, permet de remettre à jour NB, puis grace à la valeur correspondante du display l'index DP. Le contenu de la mémoire d'adresse relative 1 est affecté à IA.

* Compte tenu de ce qui est dit pour les paramètres.

Ces opérations n'ont lieu que si le numéro de bloc de l'étiquette est différent de la valeur de NB ; s'il s'agit d'une étiquette formelle, on est revenu auparavant dans la structure de bloc du paramètre effectif (cf. ch. 7 et 9).

2.7. Récapitulation.

Soit le cas général, au niveau K l'évaluation d'un identificateur de fonction, la procédure étant déclarée au niveau N-1. Pendant l'activation nous aurons la situation suivante :



La mémoire indicatrice de résultat vaut ici 1 (un accumulateur a été réservé avant l'activation).
Elle vaut 0 dans le cas des blocs et des instructions procédure.

Particularités des blocs :

AD.R (adresse de retour) = 0

NB_K (=K) = N-1 ; $DP_K = DP_{N-1}$

2.8. Remarques

2.8.1 Méthode de Dijkstra et interpréteur.

Outre les problèmes déjà cités (1.1.2.), la technique d'adressage dynamique de Dijkstra est interprétative par plusieurs aspects.

a) Le mode d'utilisation de la mémoire impose le recours à des sous programmes de gestion supplémentaires chaque fois que l'on change de niveau de nomenclature (activation, calcul d'un paramètre effectif).

b) La gestion dynamique implique le calcul d'adresse réelle d'une variable à chaque occurence de celle-ci ; d'où un sous programme de calcul d'adresse machine, si le calculateur ne possède pas suffisamment de registres d'index.

Les calculs intermédiaires considèrent la mémoire comme une pile, ce qu'il faut habituellement simuler.

c) Puisque les opérations arithmétiques et les relations feront appel à des sous programmes, il est intéressant d'y inclure un traitement dynamique des conversions de type (test du type des opérandes et conversion éventuelle avant chaque opération à l'exécution).

L'importance acquise par la partie inteprétative, conduit à conserver cette optique pour résoudre un certain nombre de problèmes.

2.8.2 . Conclusions.

Les éléments du langage objet que la méthode de Dijkstra conduit à adopter, seront donc pour la plupart équivalents à des appels de sous programmes.

Il en résulte un accroissement du temps d'exécution, mais ce principal inconvénient de la méthode reste dû essentiellement à la gestion dynamique de la mémoire.

Elle présente plusieurs avantages : l'optimisation des mémoires (réelle puisque faite à l'exécution), la solution satisfaisante et naturelle des problèmes de récursivité. Elle permet aussi, à la compilation de réduire l'encombrement du générateur, et de produire un programme objet d'autant moins long que son langage est plus synthétique.

Ces 2 caractéristiques, très intéressantes pour les petites machines, permettent de construire directement en machine le programme objet complet pour un texte source de dimension raisonnable. On évite les inconvénients d'une sortie progressive sur support intermédiaire, en cours de génération.

Cependant l'importance des sous programmes interprétatifs à introduire lors de l'exécution, ne doit pas rendre illusoire le gain sur la longueur du texte source susceptible d'être généré. En décomposant une instruction en appels de sous programmes bien adaptés, et non en appels d'un plus grand nombre de sous programmes élémentaires, on diminuera la longueur du programme objet et son temps d'exécution, mais on augmentera et le nombre de SSP et leur encombrement total. La recherche du compromis qui est l'établissement du langage objet dépend largement de la machine (nombre de digits du mot mémoire, dimension de celle-ci, logique du calculateur).

L'encombrement équilibré entre compilation et exécution, se double d'une adaptation très naturelle lorsque l'utilisation logique de la machine est la microprogrammation, comme dans le cas des CAE 500.

CHAPITRE III

LANGAGE ET PROGRAMME OBJETS.

On utilisera tout au cours de cette note des exemples de Programmes PI à PXVI donnés en Annexe. Ces programmes n'ont aucune valeur sémantique et serviront uniquement à illustrer la forme du langage et du programme objets.

On rappellera parfois et très schématiquement, la structure de ces programmes dans le texte même, en souhaitant qu'il y gagne en clarté.

3.1. Mémoires locales. Variables locales d'un bloc.

Nous utiliserons le mot bloc pour niveau, et le vocabulaire suivant.

- a) Les variables locales au sens large, d'un bloc, sont :

1) Les variables simples $\mathcal{V}$ déclarées dans ce bloc
2) Les variables $\mathcal{C}$ attachées au tableau (à chaque tableau T déclaré correspond une variable $\mathcal{C}$)
3) Les variables $\mathcal{P}$ attachées aux instructions pour (cf ch 8)
4) Les variables Cdy (caractérisation dynamique) attachées à chaque paramètre [lorsque le bloc est une procédure]
5) Les variables ou données de liaison

Le nombre de ces variables, et par conséquent les réservations à faire à l'exécution sont connus.

Les variables $\mathcal{V}$, $\mathcal{C}$ et Cdy sont représentées dans le programme par une adresse dynamique : adresse relative d'implantation, et numéro de bloc.

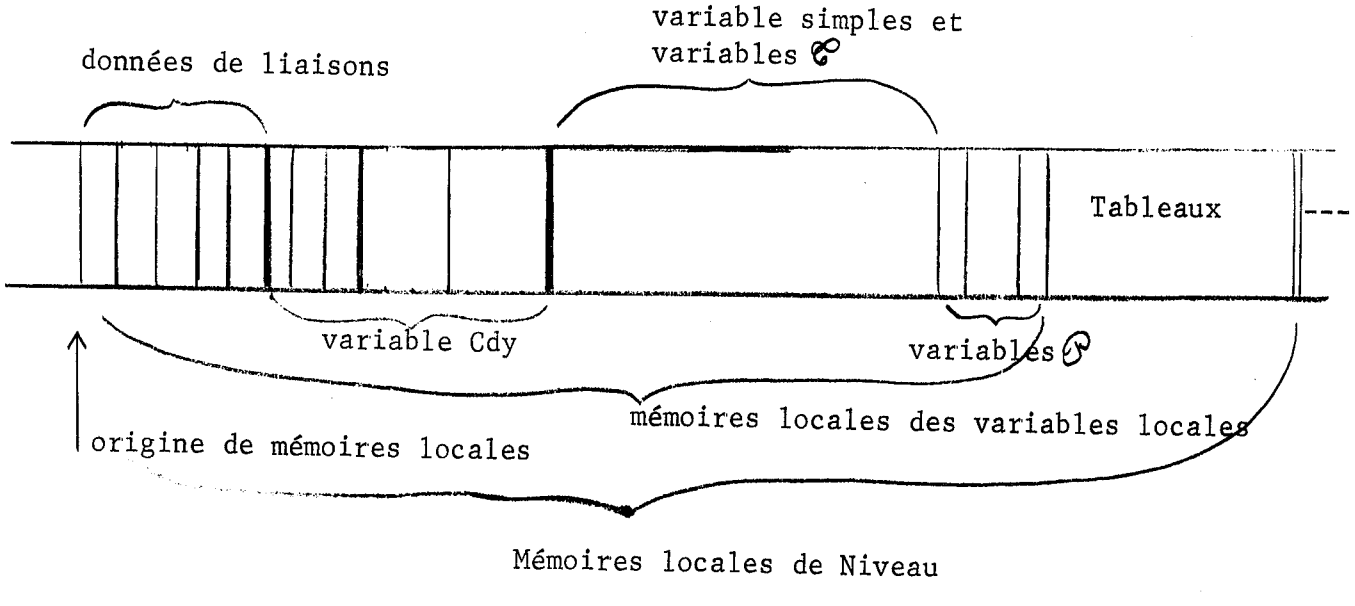
- b) Les mémoires locales d'un bloc, sont à l'exécution, les mémoires réservées, pour les variables locales, et les tableaux, par les ordres appropriés et respectivement dans cet ordre.

Aucune hiérarchie n'est imposée dans les déclarations (une déclaration de tableau peut suivre une déclaration de procédure ...).

L'organisation des mémoires locales est résumé dans le schéma suivant.

Variables simples et variables $\mathcal{C}$ se succédent suivant l'ordre des déclarations, et les variables $\mathcal{P}$ sont toujours les dernières variables locales.

Les variables Cdy n'existent que si le niveau correspond à une déclaration de procédure, et suivent immédiatement les données de liaison. Dans ce cas on pourra trouver à la suite ce qui correspond au corps de procédure s'il a été supprimé en tant que niveau d'exécution [les tableaux appelés par valeur précéderaient immédiatement les tableaux du corps](cf ch 2).



Mémoires locales de Niveau

3.2. Représentation, dans le programme objet, équivalente à un identificateur du programme source Algol.

On signifie en disant que le type est réel ou entier, que la valeur de la variable est contenue dans 2 ou 1 mémoires locales.

La distinction entre booléen et entier a disparu ['VRAI'et'FAUX' sont représentés par les entiers 0 et 1, d'autre part le compilateur a vérifié la validité ou homogénéité des expressions et instructions d'affectations qui ne peuvent être qu'arithmétiques ou booléennes].

Ce sont les ordres qui portent cette indication.

Les exemples se rapportent aux programmes donnés en annexe.

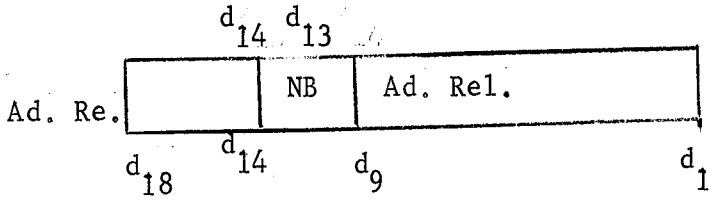
1) Variable simple :

Elle est représentée par l'adresse dynamique de la mémoire (ou de la première mémoire), locale affectée à la valeur de la variable.

Cette adresse dynamique de 13 digits comprend :

4 digits du numéro de bloc, du niveau de nomenclature auquel appartient la variable.

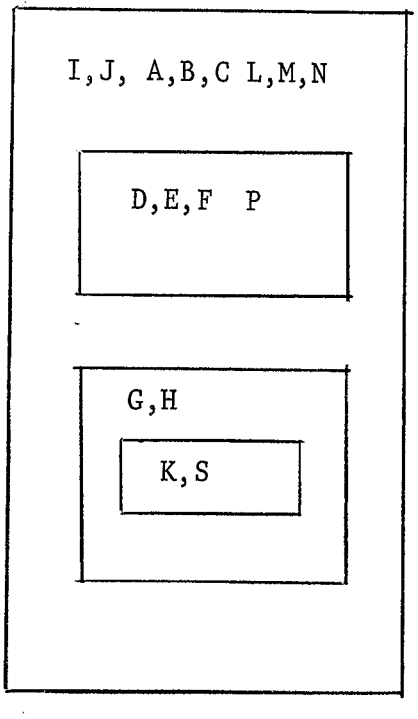
9 digits d'adresse relative par rapport à l'origine des mémoires locales du niveau de nomenclature.



Les adresses relatives sont attribuées selon l'ordre lexicographique des déclarations, à partir de la mémoire d'adresse relative 5 ; les 5 premières mémoires locales sont réservées aux données d'enchainement.

Les types réel et entier correspondent respectivement à la réservation de 2 et 1 mémoires locales.

Exemple : programme P_I



La représentation des différentes variables sera la suivante

I ⟷	0	5
J ⟷	0	6
A ⟷	0	7
B ⟷	0	9
C ⟷	0	11
L ⟷	0	13
M ⟷	0	14
N ⟷	0	15

D ⟷	1	5
E ⟷	1	7
F ⟷	1	9
P ⟷	1	11
G ⟷	1	5
H ⟷	1	7
K ⟷	2	5
S ⟷	2	6

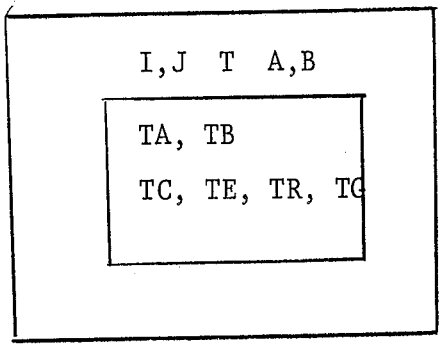
N° de bloc
Ad. Relative

2) Tableau :

A chaque tableau T déclaré dans le programme source est attachée une variable $\mathcal{C}$ de type entier, dont la valeur sera à l'exécution l'adresse du tableau.

L'identificateur de tableau T est remplacé dans le programme généré par la variable $\mathscr{C}$ dont le mode de représentation est celui déjà décrit au 1°).

Exemple : programme P_{IV}



Représentation

I	0	5
J	0	6
T	0	7
A	0	8
B	0	10

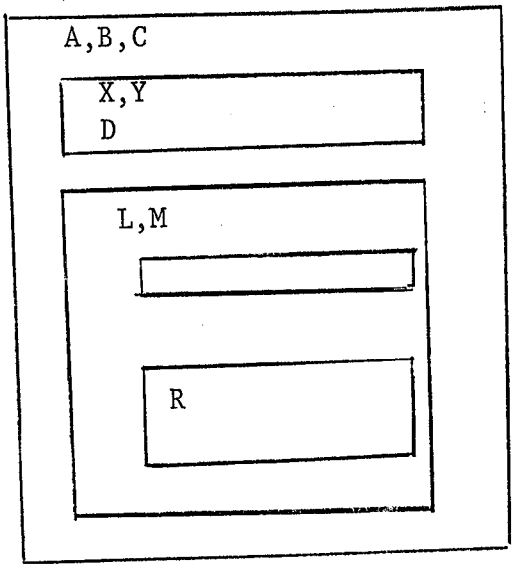
TA	1	5
TB	1	6
TC	1	7
TE	1	8
TF	1	9
TG	1	10

3) Paramètre :

A chaque paramètre formel correspond une "variable" Cdy ;en fait il ne s'agit pas d'une véritable variable (2.4.1.).

L'adresse dynamique, de Cdy, comme décrite au 1° représente le paramètre formel dans le programme généré. On notera que 3 mots sont réservés par Cdy, en séquence des données de liaison.

Exemple : programme P_{XIV}



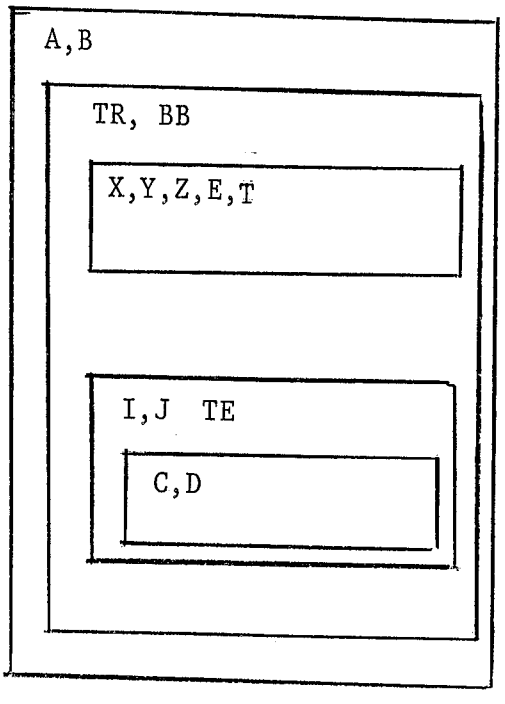
A	0	5
B	0	7
C	0	9

X	1	5
Y	1	8
D	1	11

L	1	5
M	1	6

R	2	5

Exemple : programme P_{XIII}



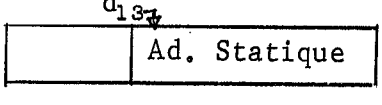
A	0	5
B	0	7
TR	1	5
BB	1	6

X	2	5
Y	2	8
Z	2	11
E	2	14
T	2	17
I	2	5
J	2	6
TE	2	7
C	3	5
D	3	7

4) Nombre sans signe. Valeur logique :

Chaque nombre sans signe du programme source se comporte comme un identificateur de variable simple très particulière : sa valeur affectée à la compilation, est contenue dans une table placée en tête du programme objet.

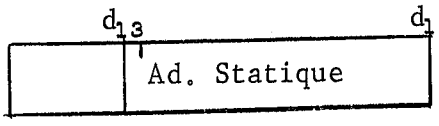
L'adresse (13 digits) dans cette table de la mémoire (ou des 2 mémoires cf 3.3) contenant la valeur du nombre, le représente dans le programme objet ; elle est dite statique par rapport à l'adresse dynamique décrite au 1°).



Les valeurs logiques VRAI et FAUX sont représentées par les adresses (27 et 25 octales) des mémoires contenant les constantes 0 et 1.

5) Etiquette - Aiguillage - Procédure :

ces quantités sont aussi représentées par l'adresse statique (13 digits) d'une mémoire d'une table placée en tête de programme objet.



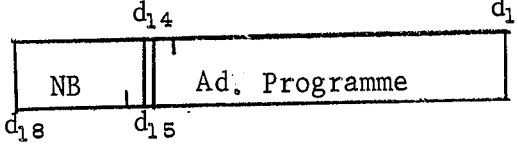
On dira : "l'adresse table"

Tables Etiquette - Aiguillage - Procédure :

La mémoire adressée en table contient ce qu'on pourrait appeler la valeur de l'identificateur correspondant, composée comme suit :

α) NB numéro de bloc du niveau de nomenclature auquel appartient l'identificateur : 4 digits (0 à 15)

β) Adresse de programme objet : adresse statique de 14 digits, soit 0 à 16 384



Cette adresse statique est celle dans le programme objet :

a) pour une étiquette : du 1er ordre généré correspondant à l'instruction source étiquetée

b) pour un aiguillage : de la mémoire caractéristique (nombre d'éléments cf ch 7)

c) pour une procédure : de l'entrée (nombre de paramètres, cf ch 4)

Remarque : procédures standard : elles sont considérées et se comportent comme des procédures déclarées au niveau 0.

3.3 Programme objet : présentation.

Il est généré dans un langage objet, immédiatement exécutable dès que l'Interpréteur est en mémoire. Le programme est accompagné des seules procédures standard qu'il utilise, (cf. ch 13 et ch 14).

A partir de son origine d'implantation fixe, et connue de l'Interpréteur, on trouve (cf. schéma 3.5.1.) : l'adresse de lancement (adresse du 1er ordre exécutable) et successivement les tables :

nombres entiers du programme
nombres réels
procédures standard
procédures
étiquettes
aiguillages

} elles n'existent éventuellement pas

puis immédiatement : le programme lui même (à partir de l'adresse de lancement).

L'index IA est initialisé sur la première mémoire libre après le programme et pourra se déplacer dans la zone de travail, libre jusqu'aux procédures standard implantées avant l'Interpréteur.

3.3.1. Tables des nombres.

Deux tables (cf 3.2.4) la table des entiers et la table des réels contiennent les valeurs des nombres sans signe du programme, selon le format décrit respectivement pour les valeurs de variables entières et réelles (3.5) : 1 ou 2 mémoires.

Un entier sans signe cité plusieurs fois dans le programme, n'existe qu'une seule fois dans la table des entiers. Au contraire dans la table des réels on trouvera les valeurs de tous les nombres sans signe non entiers, dans l'ordre où on les rencontre dans le programme : nous n'avons pas voulu écrire de programme de recherche et de comparaison en table de réels (on a moins fréquemment des nombres flottants identiques). On aurait pu aussi, placer les nombres réels directement dans le programme objet : les avoir mis en table ne prend en fait pas plus de place et permettrait éventuellement d'introduire le programme de comparaison cité.

Ces 2 tables pour lesquelles on a prévu une longueur fixe, sont construites au premier passage de compilation. Au second passage on génére le programme objet à partir du premier emplacement libre de la table des réels. Mais si la table des entiers, placée en tête, n'est pas pleine, les mémoires correspondantes sont perdues. Aussi a t-on une :

Variante d'implantation :

Selon leur encombrement les tables procédures étiquettes, aiguillages pourront être implantées dans les mémoires libres de la table des entiers.

3.3.2. Autres tables.

Elles sont toutes analogues ; une mémoire correspond à un identificateur comme décrit en 3.2.5°) [d_1 à d_{14} = Adresse programme, d_{15} à d_{18} = NB]
La table des procédures standard suit toujours celle des "réels".

3.4. Langage objet.

Il comporte 68 ordres, la plupart n'occupant qu'un mot.

L'interpréteur possède dans le compteur XCP l'adresse de l'ordre à exécuter. Après prélèvement de cet ordre, et progression de XCP, il le décode et se renvoie à l'exécution du sous programme correspondant par l'intermédiaire d'une table : le code de l'ordre n'est qu'un numéro. Avant le renvoi proprement dit, l'interpréteur fera éventuellement, certaines opérations communes à un ensemble d'ordres (calcul d'adresse physique à partir de l'adresse dynamique ...).

Les codes des ordres ont été attribués de façon à faciliter ces opérations. Les mots supplémentaires opérandes, sont prélevés par le sous-programme de l'ordre. Nous examinerons le langage objet, d'après le 1er mot, caractéristique, et en général le seul.

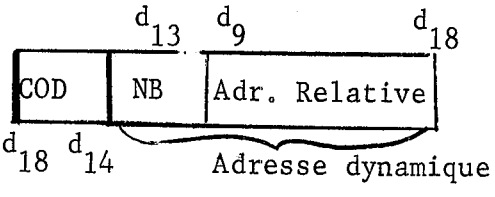
On y distingue 2 zones : zone adresse (d_1 à d_{13}), zone opération (d_{14} à d_{18}). Le code définissant l'ordre, est soit en zone opération, soit, si cette zone est nulle, en zone adresse : on a alors un ordre sans adresse. Nous donnons avec les codes leur appelation et représentation symbolique.

3.4.1. Ordre avec adresse.

d_{14} à d_{18} = code (1 à 31)
d_1 à d_{13} = adresse qui est une adresse dynamique (forme 1-a)
" physique (forme 1-b)
une opérande (forme 1-c)

La partie adresse est en général la représentation d'un identificateur (voir 3.2).

3.4.1.1. Forme 1-a



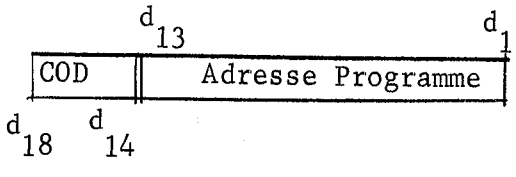
Les valeurs octales des codes sont données cadrées à droite sur d_{13}

Numéro	Octal	Code		cf. chapitre
1	02	CRN	Charger la valeur de la variable du type Réel	V
2	04	CEN	" " " Entier	id.
3	06	CRF	Charger la valeur du paramètre Formel Spécifié Réel	V et IX
4	10	CEF	" " " Entier	id.
5	12	ARN	Empiler Adresse de la variable type Réel	V
6	14	AEN	" " Entier	id.
7	16	ARF	Empiler Adresse du paramètre Formel spécifié Réel	V et IX
8	20	AEF	" " " Entier	id.
9	22	ETF	ou EAF Etiquette Aiguillage Formel	VII et IX
10	24	ADR	Additionner la variable de Type Réel	V
11	26	SBR	Soustraire " "	
12	30	MLR	Multiplier par " "	
13	32	DVR	Diviser par "	id.
14	34	EXR	Elever à la puissance de "	
15	36	ADE	Additionner la variable de type Entier	
16	40	SBE	Soustraire " "	
17	42	MLE	Multiplier par " "	
18	44	DVE	Diviser par " "	
19	46	EXE	Elever à la puissance de "	

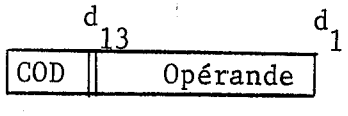
Lors des opérations arithmétiques (ordres 10 à 19), le premier opérande est dans le stack (accumulateur IA-1), le second est défini par l'adresse dynamique de la variable indiquée dans l'ordre.

3.4.1.2. Forme 1-b

Numéro	Octal	Code		V Cha.
21	52	CRS	Charger valeur du nombre Réel } l'adresse indiquée est celle de la constante en table	V
22	54	CES	" " " Entier }	id.
25	62	TST	Sauter, si la valeur booléenne en stack (accumulateur IA-1) est faux à l'adresse programme indiquée	id.
23	56	SAU	Saut inconditionnel à l'adresse programme indiquée	IV
20	50	TFA	Saut à l'adresse programme indiquée, sous-condition d'épuisement de l'élément de liste POUR 3	VIII
31	76	SSP	Activer le sous-programme dont l'adresse est indiquée (Calcul { adresse variable contrôlée / ou pas de variation d'un Pour })	VIII



3.4.1.3. Forme 1.c



Numéro	Octal	Code		V. Chap.
27	66	APN	Appel Procédure Normal (instruction procédure)(1)	IV
28	70	AFN	Appel Fonction Normal (évaluation indicateur de Fonction) (1)	"
29	72	APF	Appel Procédure Formel (1)	IV et IX
30	74	AFF	Appel Fonction Formel (1)	"
24	60	STA	Empiler la partie adresse (est l'adresse table d'une étiquette ou aiguillage)	VII
26	64	*L*	Mise à jour du compteur de ligne par la partie adresse	

(1) La partie adresse est le nombre de paramètres effectifs de l'appel.
Un second mot opérande suit ces ordres (adresse table de la procédure ou adresse dynamique si formel).

L'ordre *L* n'a pas d'influence sur l'exécution du programme. Ces ordres reflètent aussi exactement que possible dans le programme objet, la présentation par lignes (ou par cartes) du texte Algol Source. La tenue à jour d'un compteur de ligne permet en particulier à l'exécution, lors d'une erreur (débordement d'une opération, dépassement de la capacité de la mémoire...) de repérer son emplacement dans le texte source. On se reportera au chapitre 14.

3.4.2. Ordre sans adresse.

d_{14} à d_{18} = 0
d_1 à d_{13} = Code (1 à 37)

Ces ordres sont classés d'après leur fonction. Dans la représentation symbolique, nous noterons que la zone opération est nulle par Ø. La valeur octale donnée (cadrée sur d_1) est en fait l'ordre lui-même.

3.4.2.1. Opérations.

Ces opérations ont 2 opérandes (1 pour les ordres 3 et 14 qui sont unitaires) contenus dans les 2 derniers accumulateurs du stack (le premier en IA-2, le second en IA-1). Ces opérandes sont arithmétiques pour les ordres 1 à 13, booléens de 14 à 18. Aucune vérification n'est à faire à ce sujet (faite à la génération). Seul un traitement dynamique de type a lieu pour les opérandes arithmétiques (5.1.).

Le résultat arithmétique (ordres 1 à 7) ou booléen (ordres 7 à 18) est placé dans l'accumulateur IA-2, puis l'accumulateur IA-1 est libéré.

Les opérations unitaires sont une modification du dernier accumulateur (IA-1)

Numéro	Octal	Code		Opérateur Algol correspondant	
1	01	Ø	ADD	+	
2	02	Ø	SUB	-	
3	03	Ø	ISI		Inversion de signe
4	04	Ø	MUL	x	
5	05	Ø	DIV	/	
6	06	Ø	DEV	÷	
7	07	Ø	EXP	↑	
					Relations
8	10	Ø	INF	<	
9	11	Ø	ING	≤	
10	12	Ø	EGA	=	
11	13	Ø	SUG	≥	
12	14	Ø	SUP	>	
13	15	Ø	DIF	≠	
					Opérations logiques.

Numéro	Octal	Code		Opérateur Algol correspondant
14	16	Ø	NON	¬
15	17	Ø	ET	∧
16	20	Ø	OU	V
17	21	Ø	IMP	⊃
18	22	Ø	IDE	≡

3.4.2.2. Rangement - Extraction.

Numéro	Octal	Code			v. Chap.
19	23	Ø	RAN	Rangement simple valeur d'une variable	V
20	24	Ø	RAM	" multiple "	id.
30	36	Ø	RAP	Rangement simple valeur d'une fonction (1)	V
31	37	Ø	RAQ	" multiple " (1)	id.
25	31	Ø	PVA	Obtenir la valeur d'une variable Indicée	VI
26	32	Ø	IND	" l'adresse "	id.
36	44	Ø	VAL	Obtenir la valeur d'une variable dont on a l'adresse, avec conservation de celle-ci (Instruction POUR)	VIII

(1) Ces ordres relatifs à l'affectation de valeur à un identificateur de fonction ; comporte 1 mot opérande supplémentaire (type et numéro de bloc de la procédure).

3.4.2.3. Ordres d'Instruction.

Numéro	Octal	Code			V. Chap.
23	27	Ø	EBL	Entrée Bloc (1)	IV
24	30	Ø	SBL	Sortie Bloc	IV
27	33	Ø	AIG	Aiguillage (évaluation d'un indicateur)	VII
28	34	Ø	ALL	Aller a (l'étiquette est en stack)	VII
33	41	Ø	FAI	Faire simplifié (2)	VIII
35	43	Ø	DO	Faire normal (3)	VIII
37	45	Ø	RSP	Retour sous programme (répondant de SSP***)	VIII
22	26	Ø	RTP	Retour Faire (Pour) (1)	VIII
21	25	Ø	RTN	Retour sous programme, fin de calcul de paramètre effectif	VII
29	35	Ø	TAB	Création de tableaux (4)	VI
32	40	Ø	STP	Fin d'exécution du programme (STOP)	
34	42	Ø	COD	Passage au langage "interprétatif RW" ou "machine".	

(1), (2), (3), (4) comportent respectivement 1,4,2,3 mots opérandes supplémentaires. Le code Ø STP n'est utilisé qu'une seule fois en fin de programme, et Ø COD ne l'est que dans l'écriture des procédures standard (10.1)

3.4.3. Listing.

Nous ne donnons ces explications que pour suivre les exemples donnés en annexe.

Toutes les tables en têtes de programme, puis dans le programme les chaines, et les opérandes des ordres qui comportent des mots supplémentaires, sont sortis en octal.

Les ordres sont listés avec le code symbolique défini en 3.4. Cependant les ordres sans adresse ne sont pas précédés de Ø, mais imprimés au milieu.

Les adresses sont sorties en octal, ou dans le cas des adresses dynamiques, en décimal avec séparation du numéro de bloc.

Les autres mots du programme qui ne sont pas des ordres, seront imprimés comme les ordres qu'ils représenteraient, ou en octal s'ils ne correspondent à aucun.

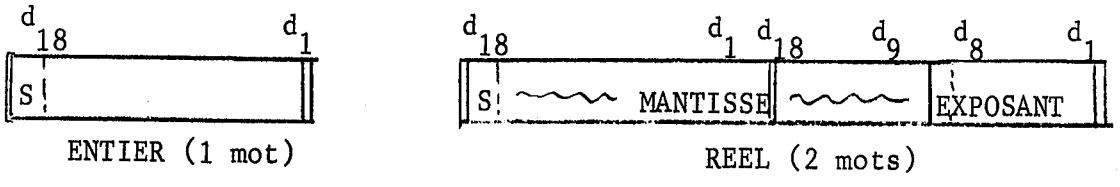
Remarque.

Le listing est généralement clair. Cependant les têtes de procédures (nombre de paramètres, modèles, indicateur de réservation), les caractérisations statiques avant les procédures sont sorties comme des ordres. Par ailleurs une caractérisation statique décodée comme APF*, provoque la sortie du mot suivant en octal et ses conséquences ; la lisibilité du listing en souffre un peu.

3.5. Format et Accumulateurs

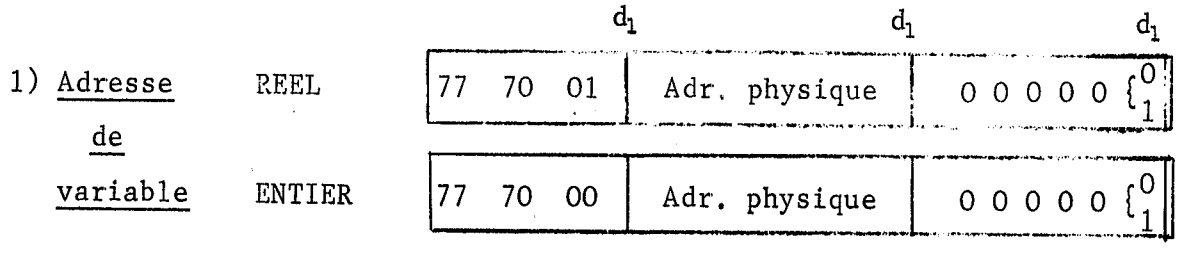
Nous allons rappeler ou donner quelques informations générales de l'exécution.

- Une mémoire locale est affectée à une variable de type ENTIER, deux à une variable de type REEL (3.2.). La valeur est contenue sous le format suivant



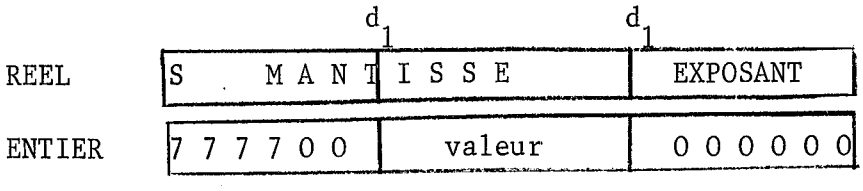
- La zone de travail, mémoire au dela des réservations locales, est utilisée normalement par accumulateur, ou groupe de 3 mémoires, contenant un "résultat". La première contient une marque indiquant le contenu et le type s'il s'agit d'une variable.

Les configurations possibles d'un accumulateur, sont les suivantes :



Dans le cas d'une adresse de variable, il s'agit de l'adresse réelle d'implantation, contenue dans la seconde mémoire. La troisième mémoire est nulle sauf s'il s'agit d'un paramètre effectif dont le type arithmétique ne correspond pas à la spécification du paramètre formel (1 = indice de conversion).

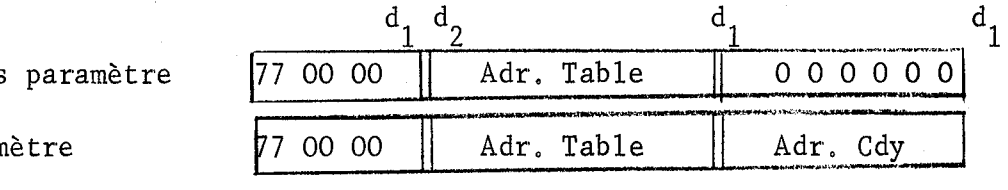
2) Valeur d'un résultat intermédiaire.



une valeur entière est contenue dans 1 seule mémoire (la seconde). Pour une valeur réelle la marque dans la 1ère mémoire est inutile (une mantisse normalisée ne commence pas par 77, retenu pour les autres marques). La mantisse occupe 2 mémoires, l'exposant une.

On notera que les calculs intermédiaires de Type Réel se font avec une précision plus grande que la précision donnée à la valeur d'une variable de type réel. En outre dans l'évaluation d'une telle expression, les conditions de débordement sont beaucoup plus larges que pour la valeur d'une variable réel. Ceci est intéressant : un débordement peut être compensé. Par exemple dans X := A x B - C x D où les calculs de A x B et C x D ne débordent pas, mais où les valeurs de A x B et C x D provoqueraient un débordement au niveau de l'affectation dans X := A x B et ou Y := C x D.

3) Etiquette et Aiguillage.



La marque de la 1ère mémoire n'a pas d'importance (uniquement pour la mise au point ; on peut supprimer cette mémoire).

La seconde mémoire contient l'adresse table de l'étiquette ou de l'aiguillage comme décrite en 3.2.5). La 3ème mémoire contiendra l'adresse physique de la caractérisation dynamique du paramètre effectif si c'en est un : correspond à l'ordre sur paramètre formel ETF (76 ou EAF) ; dans le cas contraire elle est nulle (ordre STA).

3.5.1. Nous rappellerons que l'exécution est contrôlée par

DISPLAY (longueur maximum 16 mots) et les index suivants :

XCP Compteur programme : adresse du prochain ordre à exécuter.

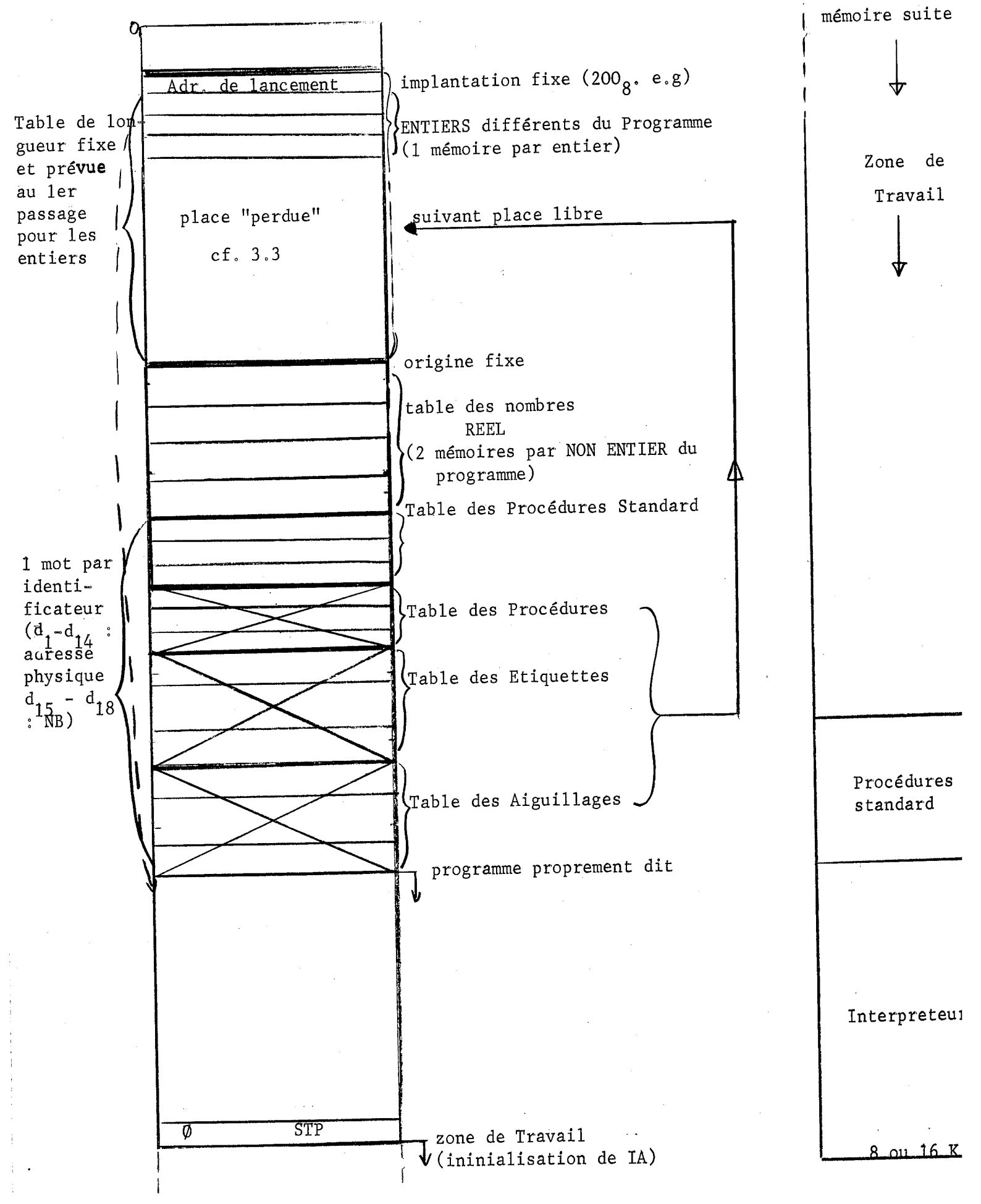
NB Numéro de bloc courant (maxi 16), repérant la dernière valeur entrée dans le display.

DP Index Debut Paramètre : origine d'implantation du niveau courant.

IA Index d'accumulateur : pointe la première mémoire libre du stack de travail, c'est-à-dire le 1er accumulateur libre.

{Nous avons mentionné les 2 derniers accumulateurs occupés comme (IA-2) et (IA-1) ils sont en fait repérés par les adresses IA-3 et IA-6}.

A l'exécution nous avons les implantations suivantes:


0
Adr. de lancement
implantation fixe (200_8. e.g)
ENTIERS différents du Programme
(1 mémoire par entier)
Table de longueur fixe et prévue au 1er passage pour les entiers
place "perdue"
cf. 3.3
suivant place libre
origine fixe
table des nombres
REEL
(2 mémoires par NON ENTIER du programme)
Table des Procédures Standard
1 mot par identificateur (d_1-d_{14} : adresse physique d_{15} - d_{18} : NB)
Table des Procédures
Table des Etiquettes
Table des Aiguillages
programme proprement dit
Ø
STP
zone de Travail
(ininialisation de IA)
mémoire suite
Zone de Travail
Procédures standard
Interpreteu
8 ou 16 K

C H A P I T R E IV

BLOCS ET PROCEDURES

L'essentiel a été vu au chapitre 2. Nous examinerons que les ordres et les particularités du programme objet.

4.1. Bloc - Ordre Ø EBL

Tout bloc source (conservé comme niveau d'exécution cf. 2.4. ou 4. 2) se traduit par une "Entrée Bloc" : ordre Ø EBL suivi d'un mot opérande, l'Indicateur de Réservation, ou le nombre de mémoires locales que réservera l'entrée bloc.

Rappelons qu'il faut compter 5 pour les données de liaison, ajouter 1 par tableau, 1 et 2 respectivement pour les variables entier et réel, 1 par instruction POUR.

Exemples P.I. en 230 Ø EBL ; P.VII en 266 Ø EBL
20 octal 22

Les programmes P.I à P.XV sont donnés en annexe.
L'interprêteur (exécution des ordres) est décrit au ch. 11.

4.1.1. Programme objet d'un bloc.

Après l'ordre EBL, il reste trace dans le programme objet des déclarations de tableau, d'aiguillage, de procédure (Celles de variables simples ont disparues), dans l'ordre du texte source.

L'ordre SAU** qui précède les parties correspondant aux déclarations d'aiguillage et de procédure permet de les enjamber.

Les premiers ordres exécutables seront donc éventuellement les ordres d'évaluation de bornes avant une création de tableau (ordre Ø TAB).

- Le bloc se termine par l'ordre sortie bloc Ø SBL. (4.3.)
  ex : P IV, P XI, P XIII.

4.2. Procédure.

Le programme objet d'une déclaration de procédure se compose

a) d'une tête de procédure : informations (non exécutables) utilisées par les ordres d'activation de ce niveau. (ch. 9)

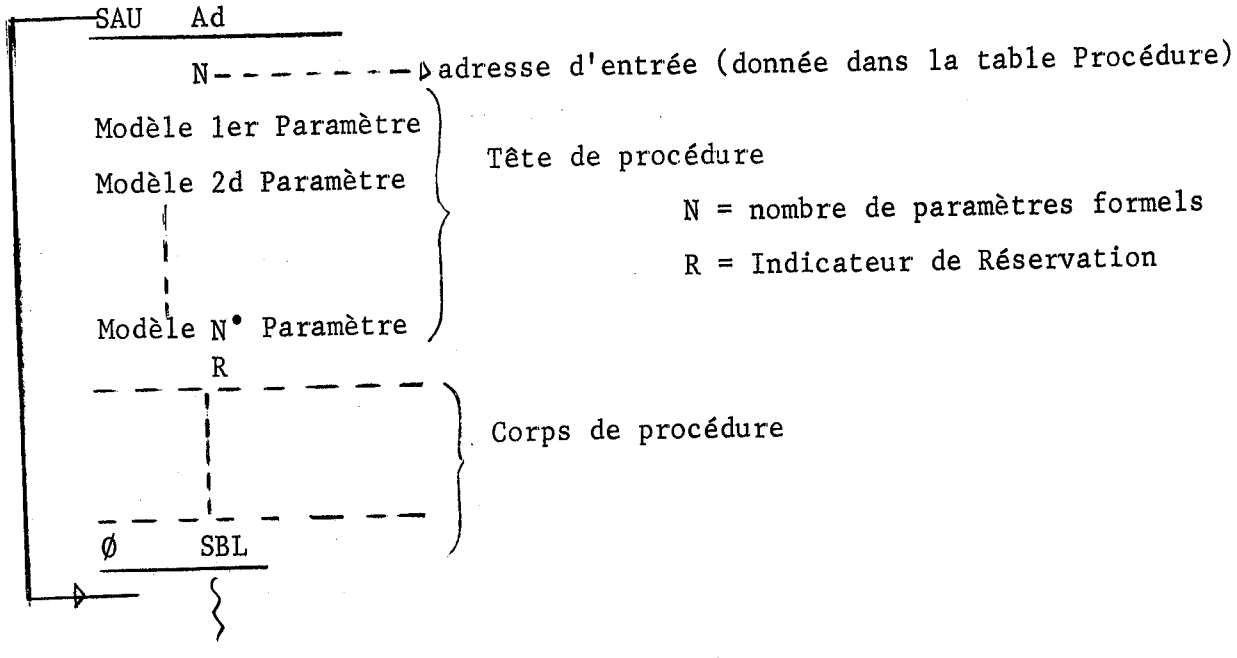
b) d'un corps de procédure : compilé d'après le corps de procédure source, suivant les règles habituelles du langage objet, avec la particularité suivante : s'il s'agit d'un bloc non étiqueté dansle texte source, il disparait en tant que niveau d'exécution (cf. 2.4.). Dans ce cas il n'est donc pas introduit et terminé par les ordres Ø EBL et Ø SBL ; ses réservations sont incluses dans celles de la procédure (Indicateur), mais il subsiste évidemment le programme généré des déclarations, mentionné en 4.1.1.

c) d'une fin bloc ordre Ø SBL
   Ex : P.XIII et P. XIX (représentations données en 3.2.5) P. XV

- La tête de procédure comprend successivement :

  - le nombre de paramètres formels (1 mot),
  - 1 modèle par paramètre (1 mot) dans l'ordre de la liste source,
  - l'indicateur de réservation (1 mot).

Rappelons que l'indicateur de réservation s'obtient en comptant 5 pour les données de liaison, puis en ajoutant 3 par paramètre et, si le corps de procédure source est un bloc non étiqueté, ce qu'on aurait ajouter à ses données de liaison (4.1.).



4.2.1. Activation d'une procédure.

Nous avons 4 ordres suivant qu'il s'agit d'une instruction procédure ou d'un indicateur de fonction, que la procédure est formelle ou non.

Leur format est le même ; le second mot contient "l'adresse" de la procédure, qui est l'adresse table (cf. 3.2.5.), ou l'adresse (dynamique) de la caractérisation dynamique dans le cas d'une procédure formelle.

Les ordres formels diffèrent uniquement par l'accès à la caractéri - sation dynamique qui

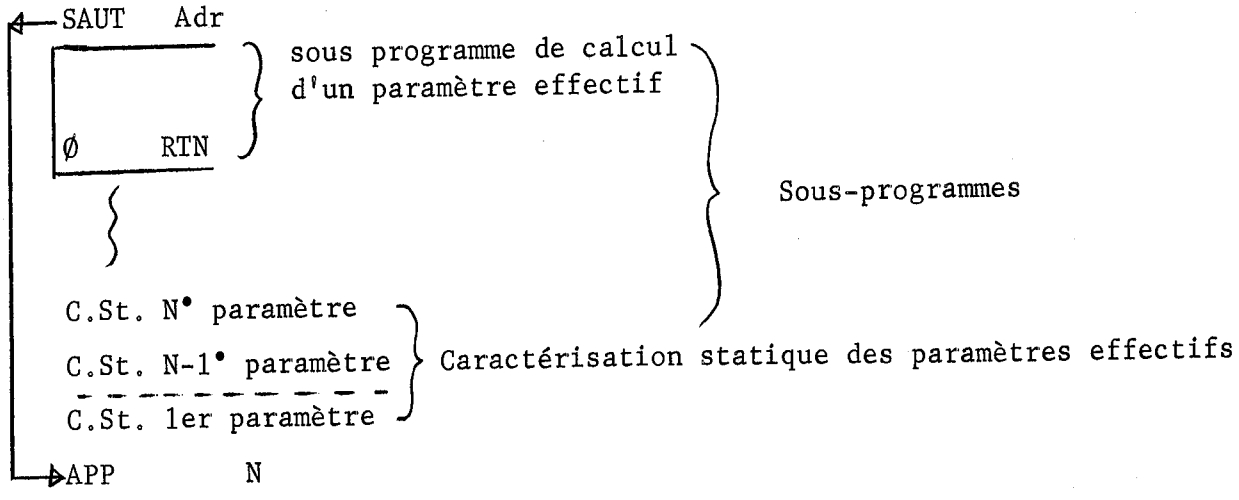
1°) donne l'adresse table de la procédure effective et,
2°) permet de repositionner le display, pour la structure de bloc origine de ce paramètre effectif, avant de mettre à jour NB et d'entrer la 1ere donnée de liaison (fin du mécanisme d'activation). Nous verrons ces particularités au chapitre 9 ; leur utilisation et traitement sont par ailleurs identiques à ceux des ordres non formels.

L'ordre d'évaluation d'un indicateur de fonction ne se distingue de l'ordre instruction Procédure que par la réservation préalable d'un accumulateur ; la mémoire indicatrice, réservée en stack, vaut alors 1 au lieu de 0.

Ordres : 2 mots

Evaluation Indicateur de fonction	AFN N Adr. Table	Formel :	AFF N Adr. Dynamique
Instruction Procédure	APN N Adr. Table	Formel :	APF N Adr. Dynamique

- La structure du programme généré pour ces activations est la suivante :



Un ordre SAU envoie à l'ordre d'activation : ce dernier est en effet précédé des caractérisations statiques Cst des paramètres effectifs dans l'ordre inverse de la liste source, et des éventuels sous-programmes relatifs aux paramètres (cf.9.2.). Si la procédure n'a pas de paramètres, on a directement l'ordre d'appel.

Les caractérisations statiques définissent les paramètres effectifs (1 mot par paramètre). Le mécanisme d'activation les utilise, conjointement aux modèles de la tête de procédure pour créer les caractérisations dynamiques (Cdy), phase que nous examinerons au Chapitre 9.

L'autre partie du traitement d'activation (non formel), est décrite au chapitre 2 auquel nous renvoyons. Nous rappellerons que l'adresse table (second mot pour non formel) est l'adresse de la mémoire qui contient le Numéro de bloc de déclaration de la procédure et l'adresse de la tête de

procédure. Il faut éliminer les "numéros de lignes", qui précèdent la tête de procédure elle-même, avec remise à jour du compteur de ligne. On a alors le nombre de paramètres formels qui est comparé au nombre de paramètres effectifs de l'appel.

Les 3,4 et 5ème données de liaisons (adresse de retour dans le programme, DPK, NBk) ont été mises en places dès l'entrée. Les 1ère et 2de données de liaisons ne le seront qu'après la phase de construction des Cdy en appel par nom. (9.4.2.) En particulier le numéro de bloc du niveau de déclaration de la procédure permet à ce moment d'obtenir la 1ère donnée de liaison et la nouvelle valeur NB ; l'indicateur de réservation venant après les modèles sert à la mise à jour de IA, d'où aussi la 2de donnée de liaison. On verra la récapitulation 2.7.

4.3. Fin d'activation - Ordre Ø SBL

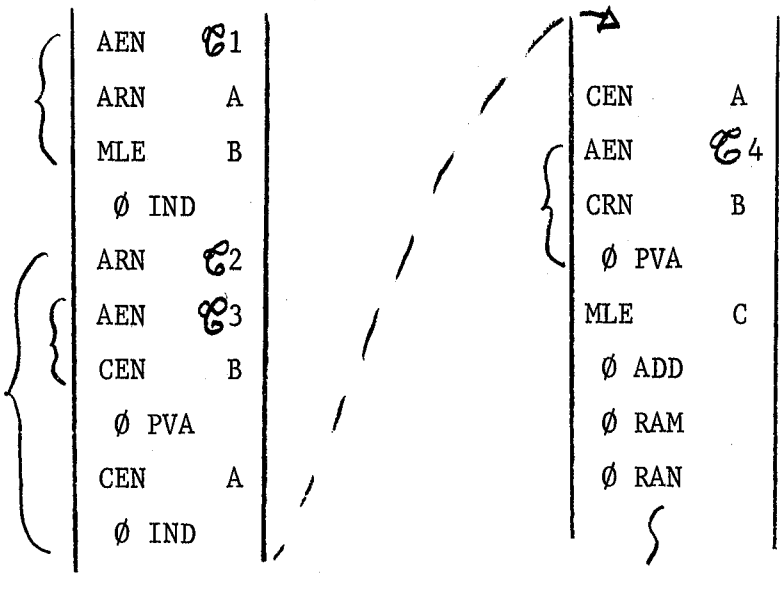
Un seul ordre dit "Sortie bloc" termine les activations de niveau provoquées soit par une entrée bloc, soit par un appel de procédure. (il correspond au symbole FIN de bloc ou ; qui suit la déclaration de procédure, dans le texte source). Il distingue les 2 cas dans la remise à jour du compteur programme XCP (inéffectuée pour les blocs) ; le traitement pouvant être par ailleurs, le même. Nous renvoyons au chapitre 2.

Le symbole ↓ d'indexation prend 2 significations, de prise d'adresse ou de valeur, suivant qu'il concerne une affectation ou une expression : c'est le principe général de traduction de l'écriture post fixée en programme objet. En partie droite d'une instruction d'affectation nous sommes en expression : les ordres correspondant aux identificateurs de variables simples sont des "Prendre Valeur", et de même l'opérateur d'indexation est $\downarrow^{V}$ (traduit par Ø PVA).

En partie gauche, si nous ne sommes pas dans une expression c'est à dire dans la portée d'un opérateur ↓, les ordres correspondant aux identificateurs de variables simples sont alors des "Prendre adresse" de même l'opérateur d'indexation est $\downarrow^{A}$ (traduit par Ø IND)

Notons que le compilateur ne reconnait pas la portée variable de l'opérateur ↓. Elle est déterminée lors de l'exécution comme on l'a vu ; d'autre part les identificateurs de tableau sont toujours traduits comme des ordres "Prendre Adresse".

En supposant des déclarations convenables, le programme objet de l'exemple sera symboliquement :



C H A P I T R E V

EXPRESSION - INSTRUCTION D'AFFECTATION INSTRUCTION CONDITIONNELLE - INSTRUCTION COMPOSEE.

5.1. Expression.

Le programme objet est la transcription des expressions en notation post fixée, et le principe de son exécution dans le stack de travail a été vu au chapitre 2.

Le générateur à vérifié, grâce aux déclarations et spécifications, que les expressions étaient correctes, c'est-à-dire soit arithmétique soit booléennes. On est assuré, à l'exécution, que les opérandes sont booléens pour un opérateur logique, arithmétiques pour les opérateurs arithmétiques et de relations.

Subsiste seul, un problème de transfert de type (Réel$\longleftrightarrow$Entier) pour les valeurs arithmétiques à l'exécution. Aussi tout accumulateur contient une marque distinctive du format de la valeur contenue (3.5.2.). Elle est crée par tous les ordres plaçant un résultat en stack ; elle est en fait implicite, pour le format flottant. Ces ordres sont les opérations, les affectations de valeur à un identificateur de procédure, les chargements de valeur d'un paramètre formel, les indexations (Ø PVA valeur variable indiciée, Ø VAL valeur variable contrôlée) et les chargements de valeur de variable simple ou de constante.

Dans ces 2 derniers cas on change, pour les variables (ou constantes) Réel, le format flottant 2 mots, en mémoires locales (ou table), en format flottant 3 mots en accumulateur. Rappelons que l'évaluation d'une expression arithmétique de type Réel se fait avec une meilleure précision, et des conditions de débordement plus larges, que la précision et les conditions de débordement de la valeur d'une variable Réel (3.5.). Ceci est en particulier vrai pour la valeur d'un indicateur de fonction.

- Les ordres de chargement de valeur de variable simple sont :
CRN Adr. Dynamique, et CEN Adr. Dynamique suivant que la variable dont on donne l'adresse dynamique est de type réel (CRN) ou entier (CEN).

- Pour les constantes, on a les ordres analogues avec une adresse statique, celle du nombre sans signe en table : CRS Adr. Table pour les réels
CES Adr. Table pour les entiers. Les nombres se comportent par ailleurs comme des variables dans l'évaluation des expressions.

Indiquons, au sujet des paramètres formels spécifiés Réel et Entier, les ordres de chargement de valeur en stack, respectivement :
CRF Adr. Dynamique et CEF Adr. Dynamique. Ils placent la valeur du paramètre effectif avec le format spécifié, et ont le même résultat, que CRN et CEN quant à l'évaluation d'une expression.

Les opérations arithmétiques (sans adresse) et les relations (ordres Ø 1 à 13) sont précédées d'un sous-programme commun, de test de type et de conversion éventuelle, avant de s'orienter vers l'un des 2 sous-programmes spécifiques de l'ordre, relatifs le premier aux opérandes entiers, le second aux opérandes réels. Ce sous programme commun teste le contenu des 2 derniers accumulateurs au sommet du stack ; sans effet s'ils sont de même type,

il convertit dans le cas contraire en réel celui des 2 qui est entier.
Une exception : il reconnait la situation accumulateur (IA-2) réel et accumulateur (IA-1) entier, pour une exponientiation, et renvoie directement au sous programme spécifique correspondant. Par ailleurs le sous programme spécifique de l'opération division entière sur réels est un déclenchement d'erreur, et celui de la division sur entier renvoie à la conversion.

Le type du résultat des opérations arithmétiques est conforme à la définition du rapport Algol (3.3.4.).

5.1.1 Expression arithmétique simple.

Le programme généré utilise les ordres sans adresse Ø 1 à 7. Les ordres avec adresse 10 à 19 permettent de condenser l'écriture (et de gagner du temps d'exécution) dans certain cas.

Exemple : P.I. l'expression (ligne 17) G + (A-BxC)/(A+B) s'écrit en post fixé : G ABC✗ - AB+/+. Avec des opérations sans adresse le programme serait (voir représentation en 3.2.1)

CRN	1	5	G
CRN	0	7	A
CRN	0	9	B
CRN	0	11	C
MUL			✗
SUB			-
CRN	0	7	A
CRN	0	9	B
ADD			+
DIV			/
ADD			+

En fait on voit dans le listing une autre écriture, utilisant des opérations avec adresse.

MLR 0 11, soit ✗ C

ADR 0 9 soit + B

- La condensation d'écriture se produit chaque fois qu'un ordre d'opération arithmétique est précédé d'un ordre de chargement en stack, de la valeur d'une variable simple, et uniquement dans ce cas : elle n'a pas lieu pour les paramètres formels et les constantes.

L'opération avec adresse indique le type de la variable simple opérande, comme l'ordre de chargement l'aurait indiquée.
L'ordre ADE Adr. Dyn. est équivalent à CEN Adr. Dyn.
ADD
Il s'exécute par ailleurs comme l'enchainement de ces 2 ordres ; mais connaissant le type du dernier accumulateur chargé, on s'oriente directement dans le sous programme de test et conversion, sur le test du second accumulateur (IA-2) : d'où un gain de temps.

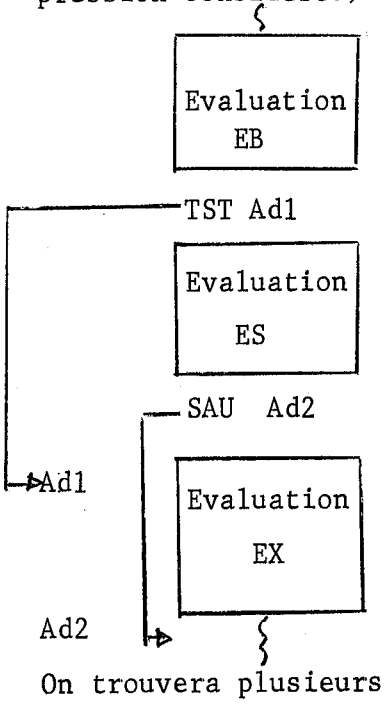
5.1.2. Expression Booléenne Simple.

Les valeurs logiques VRAI et FAUX sont représentées par les valeurs 0 et 1 de type entier, et les valeurs booléennes se comportent comme des entiers pour la mémorisation et le format en stack. Ceci ne pose aucun problème (5.1.). Les ordres utilisés sont les ordres sans adresse Ø 8 à 18. L'évaluation est très parallèle au traitement d'une expression arithmétique. Remarquons que le sous programme commun de test et conversion, a lieu avant les opérations de relation proprement dite.

Exemple P.I. L'expression (ligne 12) C < B ∧ F × D > A↑E s'écrit en notation post fixée CB < FD× AE↑>∧ d'où le programme généré que l'on verra sur le listing.

5.1.3. Expression

Soit une expression de forme générale SI EB ALORS ES SINON EX, où EB est une expression booléenne, ES une expression simple et EX une expression de même catégorie que ES (arithmétique ou booléen, soit la catégorie de l'expression considérée) ce que le générateur ă vérifié. Le programme généré a la structure ci-contre et utilise, outre l'instruction de saut inconditionnel déjà vue, un ordre nouveau, TST Adr. Programme, qui signifie : Si la valeur logique contenue dans l'accumulateur au sommet du stack est FAUX, sauter à l'adresse programme indiquée, sinon continuer en séquence. L'accumulateur est libéré dans les deux cas (regression de IA).



On trouvera plusieurs exemples dans le programme P.I.

- Nous avons vu des expressions où les primaires sont des variables simples (ou des nombres), et indiqué qu'il n'y avait pas de différence s'ils sont des parmètres formels spécifiés Réel ou Entier (sauf naturellement pour les ordres eux-mêmes, de chargement de valeur) ; exemple P.XIV ligne 3.

Rappelons que s'il s'agit d'un indicateur de fonction, 1 accumulateur est réservé au sommet du stack avant 1' activation de la procédure correspondante. Il contient le résultat quand on revient à la suite de l'évaluation de l'expression, après la fin d'activation de la procédure (5.2.2 et Ch 4) - Exemple : P.XIV ligne 6, où l'expression est réduite à un primaire.

Nous traitons au chapitre suivant les primaires variables indicées.

5.2. Instructions d'Affectation

Comme dans une expression, le générateur à vérifié que toute instruction d'affectation était homogène (liste de parties gauches et expression de même type, c'est-à-dire arithmétique ou booléen).

5.2.1. Affectation à une variable

On sait que le programme généré reflète la notation post fixée, où le symbole := est considéré comme un opérateur (2.1.). Il faut distinguer 2 opérateurs. L'opérateur d'affectation multiple := M permet de noter commodément que l'instruction d'affectation n'est pas terminée (c'est-à-dire qu'il n'est pas le dernier opérateur d'affectation) : la valeur à affecter ne doit pas être perdue. Elle ne pourra l'être qu'avec le dernier opérateur d'affectation, l'opérateur d'affectation simple :=. Dans cette méthode := M et := se traduisent par 2 ordres distincts comme nous le voyons plus bas.

Notons que les affectations se font de droite à gauche dans la liste de parties gauches, mais que les adresses ont bien été prises de gauche à droite ce qui est conforme au Rapport ALGOL [1].

Exemple : P.I. a) B := A + 2 s'écrit B A2 + :=

b) I := J := A + B I J AB+ $:=_{M}$:=

Les 2 ordres concernant les variables simples auxquelles on veut affecter la valeur d'une expression, sont des ordres "Stacker adresse physique de la variable" : l'interpréteur calcule l'adresse d'implantation à partir de l'adresse dynamique, l'empile dans le premier accumulateur libre avec la marque correspondante, et progresse IA. Ce sont respectivement pour les réels et les entiers :

ARN Adr. Dynamique et AEN Adr. Dynamique.

Nous avons les ordres analogues pour les paramètres formels spécifiés réels et entiers :

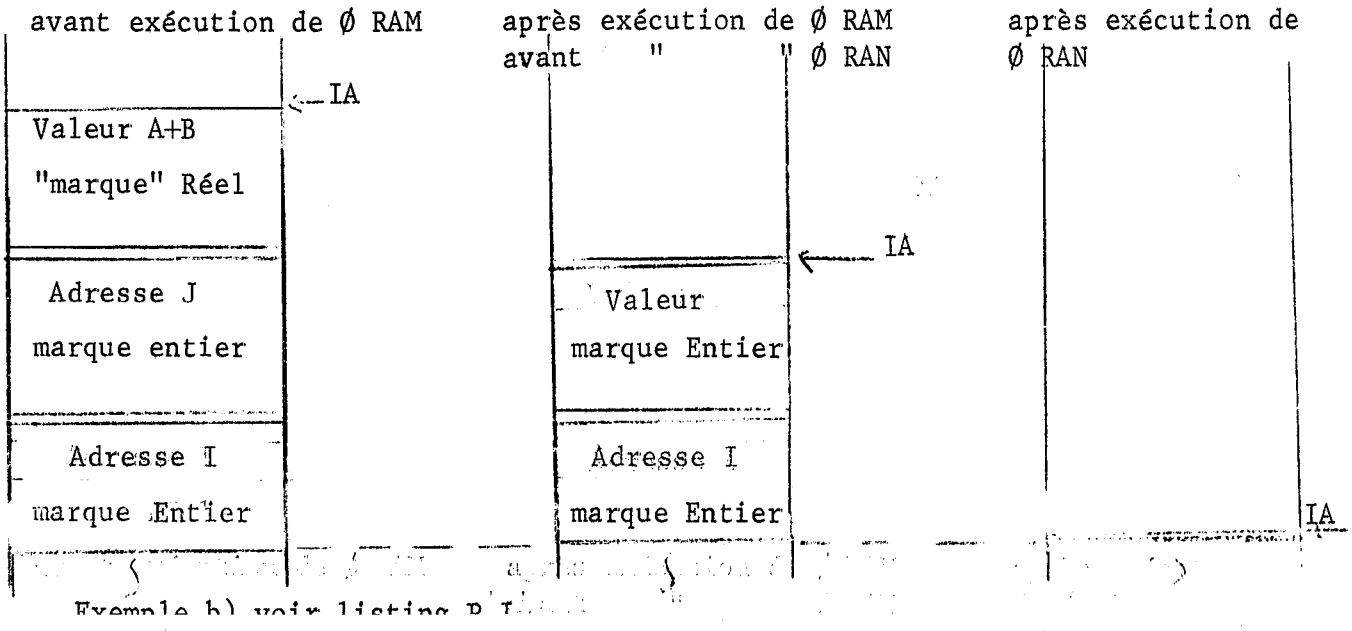
ARF Adr. Dynamique et AEF Adr. Dynamique.

Leur résultat est identique : l'adresse d'implantation du paramètre effectif avec la marque de type. Cependant ils ajoutent pour les paramètres arithmétiques un indice de conversion (1 dans 3ème mot de l'accumulateur), lorsque le type effectif n'est pas celui de la spécification. Cet indice n'est pas utilisé dans les opérations d'affectation.

La marque de type (3.5.) accompagnant l'adresse d'implantation de la variable, permet lors de l'affectation, de déterminer le format de la valeur à mémoriser : flottant 2 mots ou virgule fixe 1 mot. (on voit qu'on exécutera un Test inutile dans le cas d'une affectation booléenne).

Deux ordres Ø RAN et Ø RAM traitent respectivement les affectations simples (opérateur :=) et multiples (opérateur := M). Au moment de ces opérations l'accumulateur (IA-1) contient la valeur à ranger, l'accumulateur (IA-2) l'adresse de rangement. Dans le premier cas après exécution de Ø RAN les 2 accumulateurs sont libérés ; dans le second après exécution de Ø RAM la valeur en (IA-1) est recopiée en (IA-2), et l'accumulateur (IA-1) libéré : en effet la valeur doit être encore affectée à une variable, au moins, dont l'adresse est en stack.

Une succession d'ordres Ø RAM se termine toujours par un Ø RAN qui est le dernier ordre d'une instruction d'affectation compilée.



Exemple b) voir listing P.I.

Les ordres d'affectation convertissent éventuellement sur place la valeur (arithmétique) à ranger, pour l'accorder au type indiqué avec l'adresse, dans l'accumulateur précédent. Ils s'orientent alors vers le sous programme de mémorisation correspondant, puis déplacement de la valeur pour Ø RAM.

Pour suivre exactement le Rapport Algol [1], le type de toutes les variables (ou identificateurs de Procédure) en partie gauche doit être le même : dans une instruction d'affectation arithmétique, seul le 1er ordre d'affectation devrait pouvoir convertir le type de la valeur ; ensuite tout test indiquant la nécessité de conversion dénoterait une erreur. Ceci n'est pas réalisé dans l'interpréteur qui admet, en partie gauche arithmétique, les types réel et entier mélangés. La commodité d'écriture offerte à l'utilisateur en est la raison, de la même manière qu'on lui a permis d'écrire des paramètres arithmétiques dontle type effectif n'est pas le type spécifié. Ceci conduit d'ailleurs au traitement adopté pour les Affectations. L'utilisateur se rappellera pour les conversions, que les affectations ont lieu de droite à gauche. Le traitement exact d'Algol peut se faire avec les ordres actuels au niveau de l'Interpréteur, et avec plus d'efficacité avec 2 nouveaux ordres.

Nous verrons au chapitre suivant que le traitement est identique pour une instruction d'affectation avec variable indicée : son adresse est calculée et mise en stack avant de passer à l'évaluation de l'expression.

5.2.2. Affectation à un identificateur de procédure.

L'activation d'une procédure déclarée <TYPE> PROCEDURE a réservé avant toute autre opération, l'accumulateur alors libre au sommet du stack pour ranger la valeur uniquement s'il s'agit de l'évaluation de l'indicateur de fonction, puis comme toute activation de procédure, une mémoire indicatrice de résultat, vient ensuite l'implantation proprement dite de l'activation.

La mémoire indicatrice est nulle pour les instructions procédure (ordres APN et APF) ; elle vaut 1 pour les évaluations d'indicateur (ordres AFN et AFF), excepté, et nous ne l'avions pas encore indiqué, lorsque le type de la fonction paramètre effectif n'est pas le même que le type (arithmétique) spécifié pour le formel, dans une évaluation d'indicateur de fonction formel (ordre AFF) : elle vaut alors -1 (9.5.4.).

Ceci permet de satisfaire la règle suivante (moins stricte qu'Algol) :

Dans le corps de procédure, chaque fois qu'est utilisée la valeur d'un paramètre arithmétique, la valeur du paramètre effectif origine est éventuellement convertie pour s'accorder au type spécifié du paramètre formel.

- Dans les instructions d'affectation, on empile l'adresse d'une variable avant l'évaluation de l'expression à affecter : s'il est naturel, au point de vue compilation séquentielle de procéder au calcul de l'adresse d'une variable indicée dès qu'elle se présente, c'est aussi la règle donnée dans le Rapport Algol (traitement correct des effets de bord). De même il faut donner immédiatement l'ordre "Prendre Adresse formel" : les ordres ARF et AEF peuvent être l'excitation implicite d'un sous programme de calcul

d'adresse de paramètre effectif (alors variable indicée). Pour les variables simples l'utilisation d'un ordre spécial d'affectation (avec leur adresse dynamique en opérande) éviterait d'empiler leur adresse, avant l'évaluation de l'expression. On ne l'a pas fait par souci de simplicité et d'homogéneité de méthode.

Mais les particularités de l'instruction d'affectation à un identificateur de procédure nécessiteront un ordre spécial : aussi l'adresse est donnée, par la même occasion, avec l'ordre de rangement, et non empilée avant l'évaluation de l'expression. Elle concerne en effet un accumulateur bien défini en le considèrant comme mémoire locale, d'adresse relative -1, du niveau procédure. On réduit l'adresse dynamique à l'indication du numéro de bloc. On doit alors distinguer comme en 5.2.1., deux opérateurs d'affectation procédure, respectivement simple et multiple $:=_P$ et $:=_{PM}$. Le programme objet n'est plus la traduction de la notation post fixée exacte mais modifiée : l'identificateur de procédure précède l'opérateur :=

Exemples - Programme P XIV

a) ligne 3 F := X/D

b) ligne 5 L := SANPAR := L + 1

notation post fixée	notation utilisée
a) F X D/ :=	X D /F $:=_P$
b) L SANPAR L 1 + $:=_M$:=	L L 1 + SANPAR $:=_{PM}$:=

Deux ordres d"affectation procédure "simple" et "multiple" suivies d'un mot opérande, traduisent respectivement les opérateurs $:=_p$ et $:=_{PM}$ (l'opérande est la traduction de l'identificateur de procédure).

Ø	RAP	et	Ø	RAQ
±	N		±	N

N est le numéro de bloc de la procédure. Le symbole ± indique que d_{18} du second mot de l'ordre vaut 0 si la procédure est de type réel, 1 s'il est entier.

Traduction dans l'exemple précédent.

F estune procédure Réel de numéro de bloc 1, SANPAR une procédure Entier de numéro de bloc 2. Les ordres d'affectation correspondant sont :

a)

Ø	RAP
000001	

octal

b)

Ø	RAQ
400002	

Le traitement des ordres Ø RAP et Ø RAM se fait de la façon suivante;

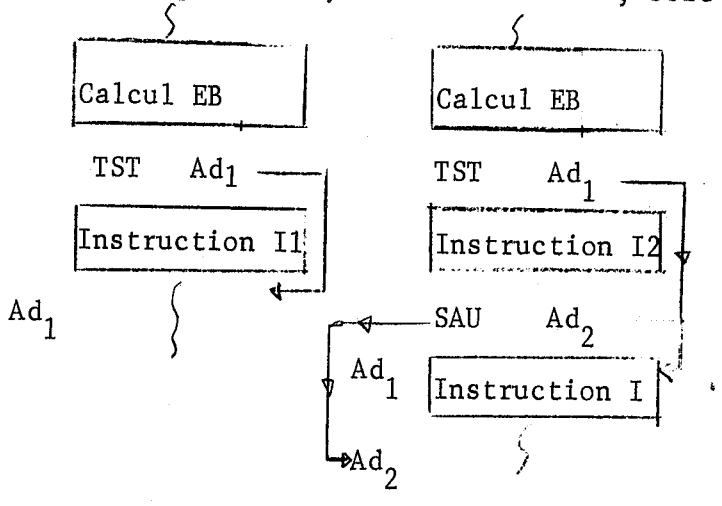
1) Conversion éventuelle, sur place, de la valeur contenue dans le dernier accumulateur pour s'accorder au type de la procédure. (signe du second mot de l'ordre).

2) Calcul de l'adresse de la mémoire indicatrice de résultat DISPLAY [N]-1. Si elle n'est pas nulle (2.4.4.) recopie de l'accumulateur au sommet du stack, dans l'accumulateur qui la précède, suivi dela conversion de ce dernier accumulateur si la mémoire indicatrice vaut -1. (cf. début de ce paragraphe).

3) Libérer l'accumulateur au sommet du stack pour l'affectation simple (regression de IA pour Ø RAP).

5.3. Instruction conditionnelle - Instruction composée

Les instructions conditionnelles sont générées, en suivant un schéma analogue aux expressions, avec l'ordre TST, soit :



où EB est une expression booléenne

I2 une instruction inconditionnelle

I1 une instruction POUR ou I2

I une instruction

ce que le générateur a vérifié.

Dans une instruction composée, les symboles DEBUT et FIN du texte source, n'ont qu'un rôle de parenthèses pour les instructions. Elles ne se traduisent par aucun ordre dans le programme objet.

Exemple Programme P.III

CHAPITRE VI

TABLEAUX ET VARIABLES INDICEES

6.1. Ordre de création de Tableau Ø TAB

A chaque tableau T, est attachée une variable $\mathcal{C}$ locale au niveau de déclaration de T. $\mathcal{C}$ est de type entier, c'est-à-dire que sa valeur est contenue dans une seule mémoire locale. Son adresse relative correspond à l'ordre lexicographique des déclarations ; puisque aucun ordre de déclaration n'est imposé, les mémoires locales des variables $\mathcal{C}$ et des variables simples sont mélées d'une façon générale.

$\mathcal{C}$ vaut à l'exécution l'adresse d'implantation du tableau correspondant, et nous verrons qu'on ne se refère dans le programme objet à une variable indicée que par l'intermédiaire de $\mathcal{C}$, d'ailleurs citée avec le type de tableau. On ne s'intéresse pas au type de $\mathcal{C}$, bien défini implicitement, mais à celui du tableau auquel elle donne accès.

Chaque déclaration de tableau du programme source, et plus exactement chaque section de tableau, se comporte comme une instruction de création de tableau ; dans le programme objet on a :

a) l'évaluation des bornes dans l'ordre de la liste de bornes source ; les valeurs sont laissées successivement en stack de travail.

b) l'ordre decréation de tableau Ø TAB ; il créera les n tableaux correspondants à la section de tableau source, à la suite des mémoires locales déjà implantées.

6.1.1. Structure d'un tableau en stack

L'organisation d'un tableau en mémoire est la suivante :

a) Tête de tableau

Groupe les informations qui permettent l'exploitation du tableau. De type entier (1 mot mémoire) elles sont successivement

E_t : encombrement total du tableau (adresse physique de la première mémoire après le tableau).

D : dimension du tableau

- m_D opposé de la valeur de la borne inférieure de la D° dimension

N_D nombre d'éléments sur le D° dimension

- m_{D-1} opposé de la valeur de la borne inférieure de la D-1° dimension

N_{D-1} nombre d'éléments sur la D-1° dimension

- m_1 opposé de la valeur de la borne inférieure de la 1ère dimension

N_1 nombre d'éléments sur la 1ere dimension.

Les bornes ont des valeurs entières (il y a eu éventuellement, transfert de type de la valeur de l'expression arithmétique).

b) Les N éléments

constituant le tableau proprement dit. Chacun occupe 1 ou 2 mémoires suivant le type entier ou réel du tableau. Ils sont rangés "par ligne" : le dernier indice varie le premier, puis le second...

L'élément correspondant à la valeur de la variable indicée $T[I_1, I_2, \ldots I_D]$ sera le $S^{\bullet}$, où

$$S := \sum_{k=D}^{k=1} (I_k - m_k) \prod_{j=k+1}^{j=D+1} N_j$$

$$N_{D+1} := 1 \qquad \text{(numérotation à partir de 0)}$$

6.1.2. Ordre Ø TAB

Exemples : Programme P IV. On voit pour chaque section de tableau d'une déclaration, l'évaluation des bornes laissant leur valeur en stack, puis l'ordre Ø TAB. Il est suivi de 3 mots opérandes :

Ø TAB
N
Type ‖ Adr. Dyn.
NT

. N : Nombre de bornes (2xD)

. NT : Nombre de tableaux à créer.

. Le second mot opérande contient :

a) Adr. Dyn. : Adresse dynamique, de la variable ℭ attachée au premier tableau à créer, donnée sous la forme habituelle [d_{13} à d_{10} = numéro de bloc, d_9 à d_1 adresse relative]

b) le type du (des) tableau à créer : réel ou entier, selon que d_{16} vaut 0 ou 1 ; on a en fait l'information, provenant de la codification du type algol de déclaration (d_{18} à d_{14} = 06 réel, 12 entier, 16 booléen) surabondante ici.

Les mémoires ℭ attachées aux tableaux déclarés dans la même section, se suivent : le compilateur procède à l'attribution suivant l'ordre lexicographique

Il suffit donc de connaître l'adresse de $\mathcal{C}$ attachée au 1er tableau de la section ; on peut utiliser seulement son adresse relative : il s'agit d'une mémoire locale du niveau courant.

Le principe d'exécution (ch. 11) de cet ordre est le suivant :

a) Création du 1er Tableau.

Les N bornes contenues dans les N derniers accumulateurs sont empilées après transfert éventuel en valeurs de type entier, au dessus de ces accumulateurs. Ils sont ainsi libérés pour créer la tête du tableau, qui commencera sur la dernière mémoire locale précédemment implantée du niveau où IA vient d'être repositionné.

Lors de la création de la tête de tableau, suivant la structure 6.1.1., on vérifie que $N_j - 1 = M_j - m_j \geq 0$ où M_j et m_j sont respectivement les bornes supérieure et inférieure de la j^e dimension. Parallèlement au calcul de N_j on évalue πN_j qui fournit le nombre d'éléments du tableau.

Après mise en place de la tête de tableau, IA, qui réfère la mémoire suivante, est déplacé pour tenir compte de l'encombrement des éléments du tableau : de πN_j ou $2 \times \pi N_j$ selon son type entier ou réel (2ème mot opérande). Cette valeur de IA est mémorisée dans la première mémoire de la tête de tableau. L'adresse de celle-ci, ou adresse d'implantation du tableau, est alors indiquée dans la mémoire de la variable locale attachée au tableau (adresse donnée dans le second mot opérande).

b) Création des autres tableaux.

Les NT-1 tableaux (NT : 3ème mot opérande) restant à créer, sont implantés successivement dans le stack, à partir de la position couramment

repérée par IA. Il y a en particulier recopie de la tête du tableau précédent ; l'adresse de chaque variable $\mathscr{C}$ se déduit de la précédente par progression de 1.

c) Mise à jour de la seconde donnée de liaison du niveau (soit S [DP+1]) avec la valeur atteinte par IA : on rappelle qu'elle indique la fin des mémoires locales (début zone de travail).

- On voit que les réservations d'un niveau comportant 2 phases.

1) réservation des mémoires des variables locales (au sens large) par l'ordre d'activation de niveau.

2) implantation des tableaux par les ordres Ø TAB.
L'ensemble constitue les mémoires locales.

6.2. Variable Indicée - Ordres Ø IND et Ø PVA.

Une variable indicée désigne une valeur qui est un élément de tableau. On aura donc besoin de l'adresse d'implantation correspondante, soit pour elle-même (instruction d'affectation par exemple), soit pour la valeur (expression). L'accès au tableau se fait par l'intermédiaire de la variable locale $\mathscr{C}$ attachée au tableau ; une variable indicée est définie par l'adresse physique de $\mathscr{C}$, et les valeurs de ses indices.

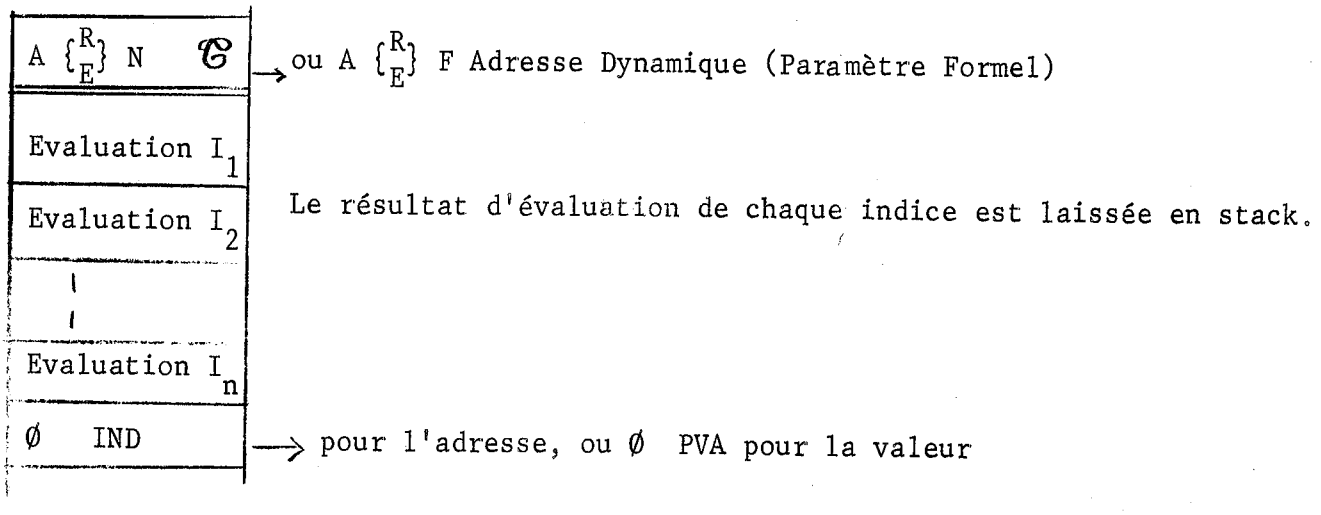
Les ordres Ø IND et Ø PVA exploitent ces informations, qu'ils trouvent dans les accumulateurs au sommet du stack, pour donner respectivement l'adresse et la valeur de la variable indicée. Ces 2 ordres diffèrent peu, et s'exécutent pratiquement par le même sous programme.

Le résultat, après libération des accumulateurs précités, est empilé.

- L'adresse de $\mathscr{B}$ est mise en stack par l'un des ordres déjà utilisé à cet effet pour les variables ARN, AEN ou s'il s'agit d'un formel ARF, AEF . La marque qui l'accompagne en accumulateur, désigne le type du tableau (ou de la variable indicée). C'est en outre une marque d'adresse (3.5.) : ceci permet, à l'exécution des ordres Ø IND et Ø PVA, de reconnaitre l'accumulateur adresse de $\mathscr{C}$, de ceux qui le précèdent au sommet du stack, et contiennent les valeurs des indices.

On rappelle que s'il s'agit d'un formel, l'adresse indiquée dans les ordres ARF et AEF n'est pas celle de $\mathscr{C}$, mais de la caractérisation dynamique ; l'adresse $\mathscr{C}$ obtenue par cet intermédiaire, est éventuellement accompagnée d'un indice de conversion (spécification différente du type du paramètre effectif). L'ordre Ø PVA en tient compte pour observer la règle rappelée en 5.2.2.

Le programme objet correspondant à T[I1, I2, IN] sera :



on en verra plusieurs exemples dans les programmes PI et PXIII.

Le principe d'exécution (Ch. 11) des ordres Ø IND et Ø PVA est le suivant ([illegible])

a) On remonte en les comptant, les accumulateurs ausommet du stack qui contiennent les valeurs, jusqu'à trouver un accumulateur d'adresse ; d'où le nombre N d'indices. La valeur de $\mathcal{C}$, dont on a l'adresse, est l'adresse du tableau. Une première vérification permet de s'assurer que N et D (dimension du tableau) sont égaux.

b) Pour chaque indice après test de type (et transfert éventuel en entier) on a calcul de :

$I_k - m_k$ qui doit être ≥ 0 (vérification indice ≥ borne inférieure)

$S := (I_k - m_k) \times L + S$

$L := L \times N_K$

$N_K - (I_k - m_k)$ qui doit être ≥ 1 (vérification indice ≤ borne supérieure)

Ceci pour k variant de N à 1, S initial := 0 et L initial := 1
S donne le rang de l'élément de tableau cherché, numérotation à partir de 0, comme indiqué en 6.1.1.

L'index HA repère la valeur du k° indice en accumulateur, et HT les informations correspondantes de la k° dimension dans la tête de tableau (initialisations sur la N° dimension : HA à la valeur initiale de IA et régressant en stack, HT en début de tête de tableau et progressant normalement).

En fin de calcul HT pointe le 1er élément du tableau, on lui ajoute S ou 2S suivant le type du tableau (marque accompagnant $\mathscr{C}$), pour obtenir l'élément cherché.

c) Le traitement se différencie alors, selon l'ordre

pour Ø IND l'adresse de l'élément remplace l'adresse de $\mathscr{C}$ en accumulateur ; la marque d'adresse n'est pas à modifier, et il ne faut pas toucher à l'indice éventuel de conversion (utilisation ultérieure possible).

pour Ø PVA la valeur de l'élément remplace l'adresse de $\mathscr{C}$ en accumulateur. Cette opération est suivie d'un transfert de type de la valeur, si l'adresse de $\mathscr{C}$ était accompagnée d'un indice de conversion (observation de la règle 5.2.2.)

IA indique alors le 1er accumulateur, libre après l'adresse, ou la valeur, de la variable indicée.

6.2.1. Remarques - Instructions d'Affectation et Expressions.

Le traitement des affectations à une variable est général ; l'adresse est empilée avant que l'on passe à l'évaluation de la partie droite (5.2.2.).

Le programme objet reflète toujours exactement la notation post fixée.

Exemple T1 [A x B] := T2 [T3 [B], A] := A + T4 [B] x C

s'écrit T1 AB x$\downarrow^{A}$ T2 T3 B$\downarrow^{V}$ A$\downarrow^{A}$ A T4 B$\downarrow^{V}$ C x + :=$_{M}$:=

Exemples : Programme PI.

On verra (ch. 9) qu'une variable indicée constituant un paramètre effectif donne lieu à un calcul d'adresse (Ø IND).

6.3. Tableau Formel.

Les ordres AEN ou ARN portant sur $\mathfrak{C}$, sont remplacés par les ordres "Prendre adresse de formel" AEF ou ARF refèrant la caractérisation dynamique du formel. On a vu que ces ordres, utilisés pour les formels spécifiés <TYPE> placent l'adresse de la variable simple paramètre effectif, en stack, de la même manière que AEN ou ARN. Pour un tableau, la variable paramètre effectif est la variable simple $\mathfrak{B}$. Le traitement reste le même (ch. 9).

L'interpréteur ne comporte pas le traitement des tableaux appelés par valeur : l'utilisation de ce mode d'appel est interdit.

L'introduction des sous-programmes correspondant ne poserait pas de difficultés. Le mécanisme d'activation de procédure comporterait des opérations analogues à l'exécution de Ø TAB : recopié (avec éventuellement transfert de type cf. règle de 5.2.2.) du tableau paramètre effectif à la suite des mémoires locales du niveau ; indication de l'adresse d'implantation du tableau dans la caractérisation dynamique ; mise à jour de la 2de donnée de liaison.

Avec l'appel par nom, le mécanisme d'activation se contente de mettre dans la caractérisation dynamique, l'adresse physique de $\mathfrak{T}$ relative au tableau paramètre effectif.

Remarque.

L'encombrement du tableau (adresse fin de tableau +1) existe en tête du tableau pour faciliter l'adaptation éventuelle de l'appel des tableaux par valeur, et surtout les procédures standard d'entrée-sortie de tableau.

C H A P I T R E VII

ETIQUETTE - INSTRUCTION ALLER A - AIGUILLAGE - ACTIVATION D'UN SOUS PROGRAMME

7.1. Etiquette - Ordres STA et ÆAF

A l'exécution, une instruction ALLER A doit connaitre pour l'étiquette qu'elle refère :

a) l'adresse dans leprogramme objet de l'ordre correspondant

b) son numéro de bloc ; en effet on va continuer l'exécution à ce niveau et terminer par là même, les activation enchainées, jusqu'à l'instruction ALLER A.

c) l'implantation particulière de ce bloc (et de sa structure de blocs). Le problème est le même qu'avec une variable locale : la récursivité conduit à avoir dans le stack de travail, plusieurs implantations qui correspondent au même bloc du programme source. Mais ce sont des blocs différents pour l'exécution ; et lorsqu'on se réfère à une variable locale elle est relative à la dernière activation du bloc.

L'étiquette se conduit ainsi : elle est locale et attachée à une activation particulière. Le display donne la stucture de blocs pour résoudre ce problème.

Si l'on veut exploiter une étiquette (non un paramètre), il permet de déterminer l'implantation du bloc où elle est locale, connaissant son numéro. Dans ce cas, le display est à jour lorsqu'on reprend l'exécution du bloc (les derniers niveaux de la structure précédente disparaissent, mais il n'est pas besoin de les effacer du display).

d) la première mémoire de la zone de travail du bloc considéré. Il faut en effet remettre à jour l'index IA, à la suite de la dernière mémoire locale, on l'obtient par la 2de donnée de la liaison du bloc (2.6. - on peut traiter séparément le cas où l'on reste dans le même bloc).

Un exemple rendra plus sensible le problème de traitement des étiquettes. Le programme suivant n'a pas d'autre signification.

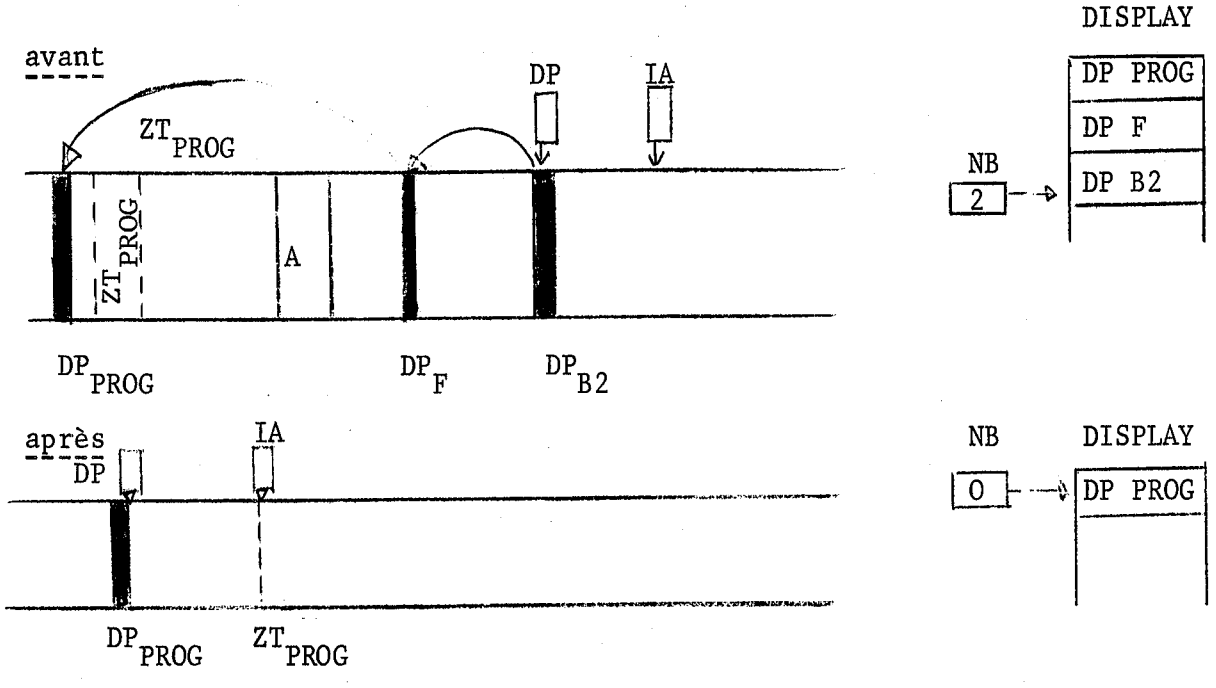
```
PROG : DEBUT ENTIER N,A,B ;.....
       ENTIER PROCEDURE F (X,EF) ; ENTIER X ; ETIQUETTE EF ;
        DEBUT ...
        B2 : DEBUT ENTIER C,D ;....
             SI X ≥ 6 ALORS ALLERA E1 ; SI X ≤ 0 ALORS ALLERA E2 ;
             SI X = 4 ALORS ALLERA EF SINON F := A x F(X-1, E3)+A ;
             ......
           FIN ;
           ........
           ALLERA FINI ; E3 : F := 2 ; ALLERA FINI ; E2 : F := 1 ;
      FINI : FIN DE F ;
      ............
     LIRE (N,A) ; B := A+F(N, E1) ; ALLERA SORTIE ;
   E1 : IMPRIMER ('INDEFINI') ; .........
SORTIE : FIN DE PROG
```

Examinons les conséquences de l'activation de F par B := A + F (N,E1) pour quelques valeurs de N.

1°) N égal à 6.

L'activation est aussitôt interrompue, sans que l'on ait atteint la fin "normale" d'activation du niveau, pour aller à E1. E1 est une quantité locale de PROG auquel on accède par le Display.
On notera que la remise à jour de IA libère les accumulateurs occupés dans l'implantation de PROG par le calcul de B.

Les situations avant et après l'exécution de ALLERA E1 donnée dans B2 sont les suivantes

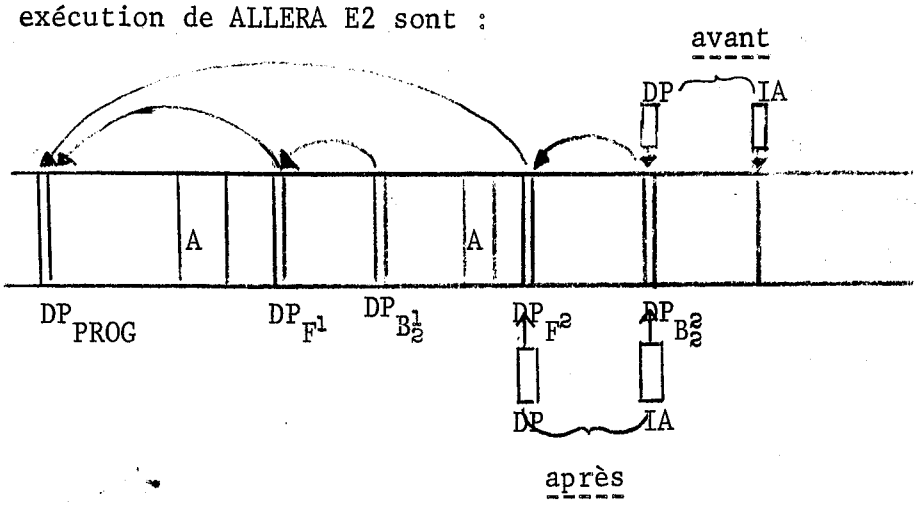


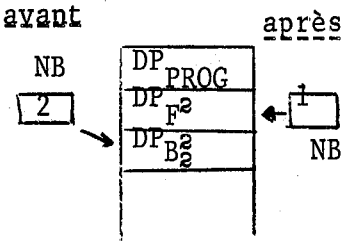
2°) N égal à 1

Il y a appel récursif de F par F := AxF(X-1, E3) + A. Lors de cette seconde activation on exécute ALLERA E2. E2 est une quantité locale du bloc

F actuellement exécuté. Le problème est le même qu'au 1°), le display y donne accès, le distinguant de la 1ère activation suspendue pour le moment.

En notant les activations en indice les situations avant et après exécution de ALLERA E2 sont :





La seconde activation de F se termine alors normalement en donnant la valeur 1 et l'on revient à l'exécution du bloc qui l'a appelée, c'est-à-dire dans B_2^1 pour poursuivre l'exécution de F := A x 1 + A.

3°) N égal à 5

Il y a aussi appel récursif de F par A x F(X-1, E3)+A mais lors de cette activation on va exécuter ALLERA EF. Nous utiliserons les mêmes notations que sur la figure précédente. Le paramètre effectif E3 n'appartient pas à la structure de blocs DP_{PROG}, DP_{F^2}, $DP_{B_2^2}$ décrite par le display : E3 est locale à un bloc de la structure appelant cette seconde activation de F, soit DP_{PROG}, DP_{F^1}, $DP_{B_2^1}$ qu'il faut retrouver pour exploiter l'étiquette. Dans le cas présent E3 est locale à F^1, et la seconde activation est interrompue de façon tout à fait analogue à ce qui a été examiné au 1°) : on revient à la première activation en abandonnant le calcul commencé de F := A x F(X-1, E3)+A, pour exécuter E3 : F := 2.
(la valeur de B sera A+2).

A chaque étiquette correspond, dans une table, une mémoire contenant son numéro de bloc (niveau de nomenclature auquel elle appartient), et l'adresse dans le programme objet du 1er ordre de l'instruction étiquetée. L'adresse de cette mémoire représente l'étiquette dansle programme objet 3.2.5.).

L'évaluation d'une expression de désignation, se fait comme pour une expression arithmétique ou booléenne. Le résultat étiquette, ou plutôt les informations équivalentes quant à l'exécution, est laissé en stack dans un accumulateur (dit d'étiquette).

7.1.1. Ordre STA

L'expression de désignation la plus simple est l'étiquette ; il lui correspond l'ordre d'évaluation STA Adresse, dans le programme objet. (Le programme source ne cite pas un formel ; ce que nous examinons ensuite, mais bien une étiquette dont la portée s'étend à cet endroit).

L'Adresse indiquée avec l'ordre STA est naturellement celle de la mémoire en table. On a montré précédemment que cette mémoire fournit les informations suffisantes, pour qu'une instruction ALLER A, donnée dans un bloc où s'étend la portée de l'étiquette, puisse l'exploiter. C'est l'utilisation la plus habituelle d'une expression de désignation ; aussi limite t-on l'ordre STA à empiler dans le 1er accumulateur libre, l'adresse de la mémoire en table étiquette (partie adresse de l'ordre).

7.1.2. Ordre EAF.

Si le programme source cite, dans une expression de désignation un formel spécifié Etiquette, il lui correspond dans le programme objet l'ordre

d'Evaluation:EAF Adresse Dynamique.

L'adresse dynamique est celle de la caractérisation dynamique Cdy, que crée le mécanisme d'activation de la procédure, pour le paramètre effectif.

Comme tous les ordres sur les formels, il correspond à un ordre sur quantité déclarée, et donne un résultat équivalent à l'exécution . C'est ici une étiquette ; mais une instruction ALLER A ne peut plus l'exploiter, à partir des seules informations de la mémoire en table.En effet le display ne reflète plus une structure de blocs comprenant le bloc auquel est attaché l'étiquette, mais la structure de blocs de l'activation de procédure exécutée (cf. Ex7.1.3.) On se ramènera au cas précédent, par la connaissance du bloc BK où est activée la procédure P, qui détermine l'étiquette remplaçant réellement le formel auquel on s'intéresse. P n'est pas forcément la procédure que l'on exécute (activation courante) : on peut écrire un paramètre formel comme paramètre effectif d'une instruction procédure ; les aiguillages ajoutent par ailleurs dans le cas des expressions de désignation, un relais supplémentairede communication de paramètres effectifs. Nous préciserons bientôt ces problèmes.

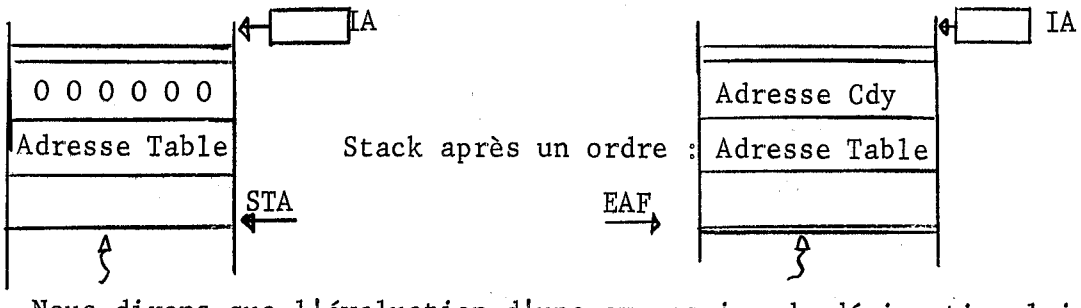
B_k est défini par son numéro de bloc NB_K et son implantation DP_K : ceci permet de repositionner NB, DP puis le Display. Nous appellerons cette structure de blocs de B_K, la structure de blocs origine de l'étiquette résultat. Les caractérisations dynamiques Cdy contiennent des informations du type NB_K, DP_K ; aussi a t-on choisi d'indiquer la structure de blocs origine d'une étiquette, résultat d'évaluation d'un paramètre effectif, par l'adresse physique d'une Cdy.

- En résumé l'ordre EAF, détermine grace à la caractérisation dynamique dont il a l'adresse

a) l'adresse de la mémoire table de l'étiquette effective.

b) l'adresse physique de la Cdy qui définit la structure de blocs origine de l'étiquette (peut être différente de la Cdy référée).

Ce résultat est naturellement laissé dans l'accumulateur au sommet du stack, comme pour l'ordre STA ; le 3ème mot contient maintenant une adresse au lieu de 0.



Nous dirons que l'évaluation d'une expression de désignation laisse en stack : une étiquette paramètre si le 3ème mot de l'accumulateur est une adresse de Cdy (définissant donc la structure origine où il faut l'exploiter), une étiquette non paramètre si ce 3ème mot est nul (que l'on peut donc exploiter dans la structure actuelle). Cela s'applique évidemment à l'évaluation d'un indicateur d'aiguillage.

7.1.2.1. Caractérisations dynamiques et ordre EAF.

Le problème général des caractérisations dynamiques de paramètres effectifs, et des ordres sur les formels, est étudié au ch. 9. Nous ne l'envisageons ici que pour préciser la définition de structure de blocs origine, et le rôle de l'ordre EAF utilisé aussi pour les formels spécifiés Aiguillage (*)

(*) l'ordre EAF est listé ETF sur les exemple donnés en annexe. La dénomination EAF (Etiquette-Aiguillage-Formel) est préférable.

Indiquons en se limitant aux appels par nom :

a) le paramètre effectif n'est pas un paramètre formel :

La caractérisation dynamique donne

1) NB_K et DP_K relatifs au bloc B_K, où est activée la procédure considérée.

2) l'adresse en table (étiquette ou aiguillage) si le paramètre effectif est un simple identificateur, ou s'il est une expression de désignation l'adresse, dans le programme objet, du sous programme d'évaluation.

b) le paramètre effectif est un paramètre formel X

Sa caractérisation dynamique est identique à celle que l'on a crée pour le paramètre effectif correspondant au formel X.

Une Cdy donnera toujours, comme dans ces 2 cas la structure de blocs origine du paramètre effectif, définie par le bloc B_K où il faut se replacer pour effectuer un calcul relatif au paramètre effectif. La raison du calcul peut être directe (évaluation d'une expression de désignation paramètre effectif) ou non (indicateur d'aiguillage où ce dernier est paramètre effectif). On notera que la structure de blocs origine d'une étiquette-résultat n'est confondue avec celle du paramètre effectif que si celui ci ne fait pas intervenir de formel spécifié étiquette, directement ou non.

Dans le cas simple d'un paramètre effectif aiguillage ou étiquette, l'ordre EAF obtient immédiatement dans la Cdy référée, l'adresse-table, qu'il stacke en l'accompagnant de l'adresse physique de cette même Cdy.

L'évaluation d'une expression de désignation par un ordre EAF, pose les mêmes problèmes pour l'étiquette résultat, que l'indicateur d'aiguillage formel examiné en 7.4.

7.2. Instruction ALLER A - Ordre Ø ALL

Le compilateur a vérifié qu'une instruction ALLER A porte sur une expression de désignation, et que celle-ci est homogène (le résultat sera surement une étiquette). Au moment de l'exécution on ne s'occupe plus de ces problèmes (comme dans les affectations). Le programme objet est encore le reflet de la notation post fixée étendue ; le symbole ALLER A se conduit de faion analogue à un := d'une instruction d'affectation, et se traduit par l'ordre Ø ALL.

Exemple : programme PII ; l'instruction

ALLERA SI B < A + Ī ALORS E1 SINON E3 ;

s'écrit

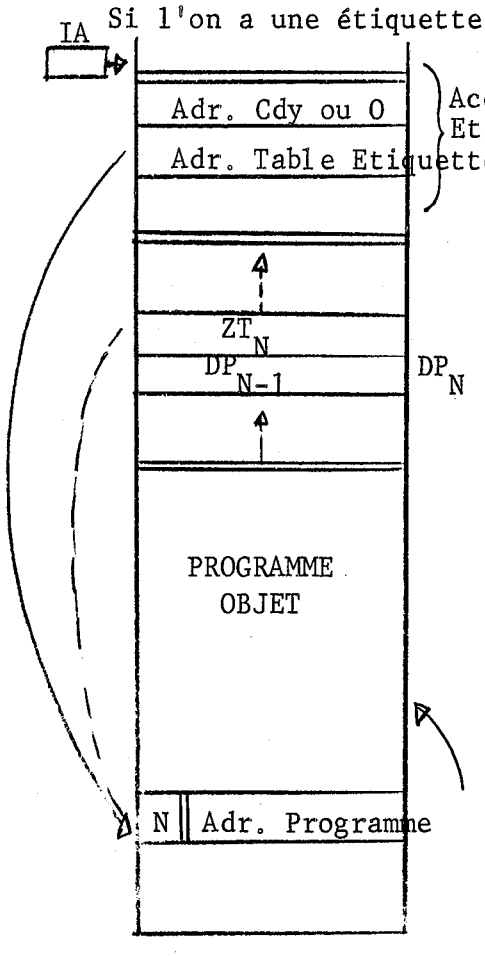
B AI + < SI E1 ALORS E3 SINON ALLERA

on verra le programme objet sur le listing.

Lorsqu'on va exécuter l'ordre Ø ALL le dernier accumulateur occupé contient l'étiquette à exploiter : étiquette paramètre ou non (7.1.2.). Après les indications déjà données, nous n'examinerons que rapidement cette exécution (cf. Ch. II) :

a) Si l'on a une étiquette fictive (2^d mot = -1 : indicateur d'aiguillage non défini - cf. 7.4.) l'accumulateur étiquette est libéré et l'ordre Ø ALL terminé. (L'exécution continuera donc en séquence dans le programme objet).

b) l'adresse table de l'étiquette permet d'obtenir l'adresse programme où l'on va sauter (remise à jour de XCP) et le numéro de bloc N de l'étiquette Si l'on a une étiquette non paramètre on saute directement à la phase C2.



Pour une étiquette paramètre, le 3ème mot non nul, donne l'adresse de la caractérisation dynamique, définissant la structure de blocs origine de l'étiquette. On s'y repositionne ; les valeurs NB_K et DP_K fournies par la Cdy permettent la remise à jour de NB, de DP, puis du display. Ce problème est analogue à une fin d'activation de procédure.

C1) le numéro de bloc N de l'étiquette permet de repositionner les index NB et DP (:= display [NB]) pour le bloc dont on va poursuivre l'exécution. Puis l'index IA est mis à jour (:= S [DP+1]) par la S^de donnée de liaison : adresse de la zone de travail du niveau N. L'exécution de l'ordre Ø ALL est alors achevée.

C2) Pour une étiquette non paramètre, si l'on ne sort pas du niveau courant (N = NB), on se contente de régresser IA pour libérer l'accumulateur étiquette ; l'ordre Ø ALL s'arrête là. Sinon on exécute la phase C1.

Remarques.

. La phase C2 permet de gagner un peu de temps pour le cas usuel. Cependant le test en fait perdre un peu pour les autres et ce n'est peut être pas très intéressant.

. La seule justification de considérer des étiquettes paramètres ou non, (soit de programmer STA et EAF à donner des formats différents), est d'essayer de gagner du temps pour les instructions ALLER A les plus courantes, quitte à en perdre sur les autres par ailleurs.

7.3. Déclaration d'Aiguillage.

La déclaration d'aiguillage peut être considérée comme une déclaration de procédure ; elle fournit un résultat étiquette, comme une procédure déclarée <TYPE> PROCEDURE donne un résultat valeur logique ou arithmétique, lorsqu'elle est activée.

De plus on peut voir le corps de procédure comme imposé : c'est l'appel d'une fonction standard, dont les paramètres effectifs sont les expressions de désignation de la liste d'aiguillage. Son nombre de paramètres est variable, et la valeur de cette fonction standard est celle de son Xème paramètre effectif. (X paramètre formel de la déclaration de procédure aiguillage, et paramètre effectif de la fonction standard, ce qui n'est pas formulé explicitement; à X correspond le paramètre effectif I dans l'activation A[I], ou évaluation d'un indicateur d'aiguillage). La fonction standard peut être utilisée récursivement.
Son appel, "corps" de la déclaration aiguillage, se présente de façon analogue à une instruction procédure classique (voir b, c, d ci-dessous). De plus compte tenu des conditions très particulières d'activation, la déclaration de procédure se réduit à ce corps. On aura donc :

a) un ordre de saut après le programme objet de la déclaration.

b) pour chaque expression de désignation, la séquence d'évaluation suivie de l'ordre Ø RTN. On a donc un véritable sous programme

de calcul de paramètre effectif (Ø RTN est l'ordre de retour cf. 7.5. et ch. 9).

c) la liste des adresses d'entrées de ces sous programmes, dans l'ordre inverse de la liste d'aiguillage ; c'est l'analogue des caractérisations statiques d'une instruction procédure.

c) le nombre d'entrées dans la liste. C'est en quelque sorte le nombre de paramètres effectifs dansl'appel de procédure (il n'y a pas lieu d'indiquer cet appel puisqu'il est bien déterminé implicitement).

On remarquera qu'il y a toujours un sous programme d'évaluation, même si l'expression de la liste est une simple étiquette. D'autre part, le compilateur a vérifié que les expressions était bien de désignation.

La mémoire en table aiguillage, correspondant à l'aiguillage déclaré (3.2.5.), donne l'adresse du nombre d'entrées. (dite aussi de la mémoire caractéristique).

Exemple : Soit au niveau N la déclaration AIGUILLAGE A := E1, E2, E3 ; où E1, E2, E3, sont des expressions de désignation (voir en 7.4. le programme objet).

On verra aussi les exemples des programmes P XI, PXII, PXVI.

7.4. Evaluation d'un Indicateur d'Aiguillage - Ordre Ø AIG

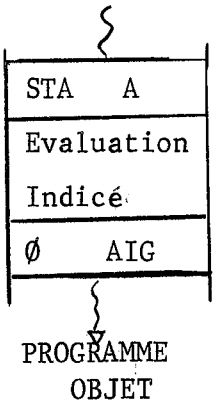
Le programme objet est établi suivant la notation post fixée, analogue à l'écriture pour une variable indicée.

Exemple A [B+C] s'écrit A B C +$\downarrow_{AIG}$

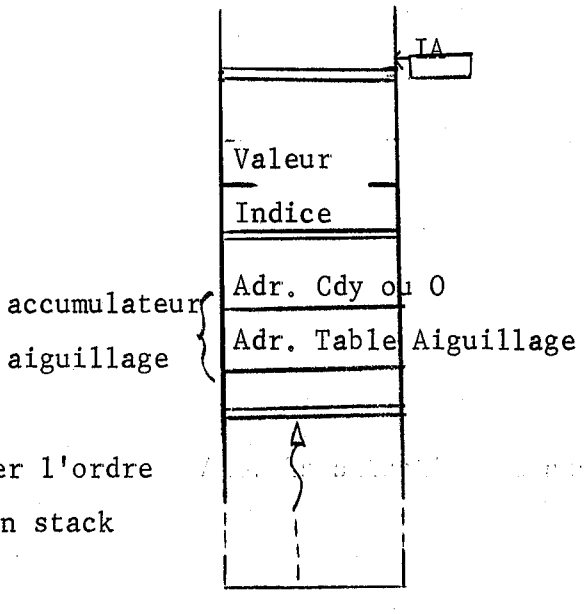
L'opérateur $\downarrow_{AIG}$ porte sur les 2 opérandes qui le précèdent, l'un arithmétique l'autre aiguillage (l'opérateur $\downarrow$ portait sur un nombre variable d'opérandes). Le compilateur a vérifié qu'un indicateur d'aiguillage n'avait qu'un indice unique et de type arithmétique.

Pour les aiguillages on utilise les 2 mêmes ordres que pour les étiquettes : STA et pour les formels EAF. Rien ne diffère si ce n'est que l'adresse table est maintenant celle d'un aiguillage, et que l'adresse Cdy indiquée pour un formel, définie la structure origine du paramètre effectif (7.1.2.1.) ; il n'est plus question de résultat ici.

L'opérateur $\downarrow_{AIG}$ se traduit par l'ordre Ø AIG et le programme objet est alors symboliquement :



ou EAF si formel



Lorsqu'on va exécuter l'ordre Ø AIG la situation en stack est la suivante :

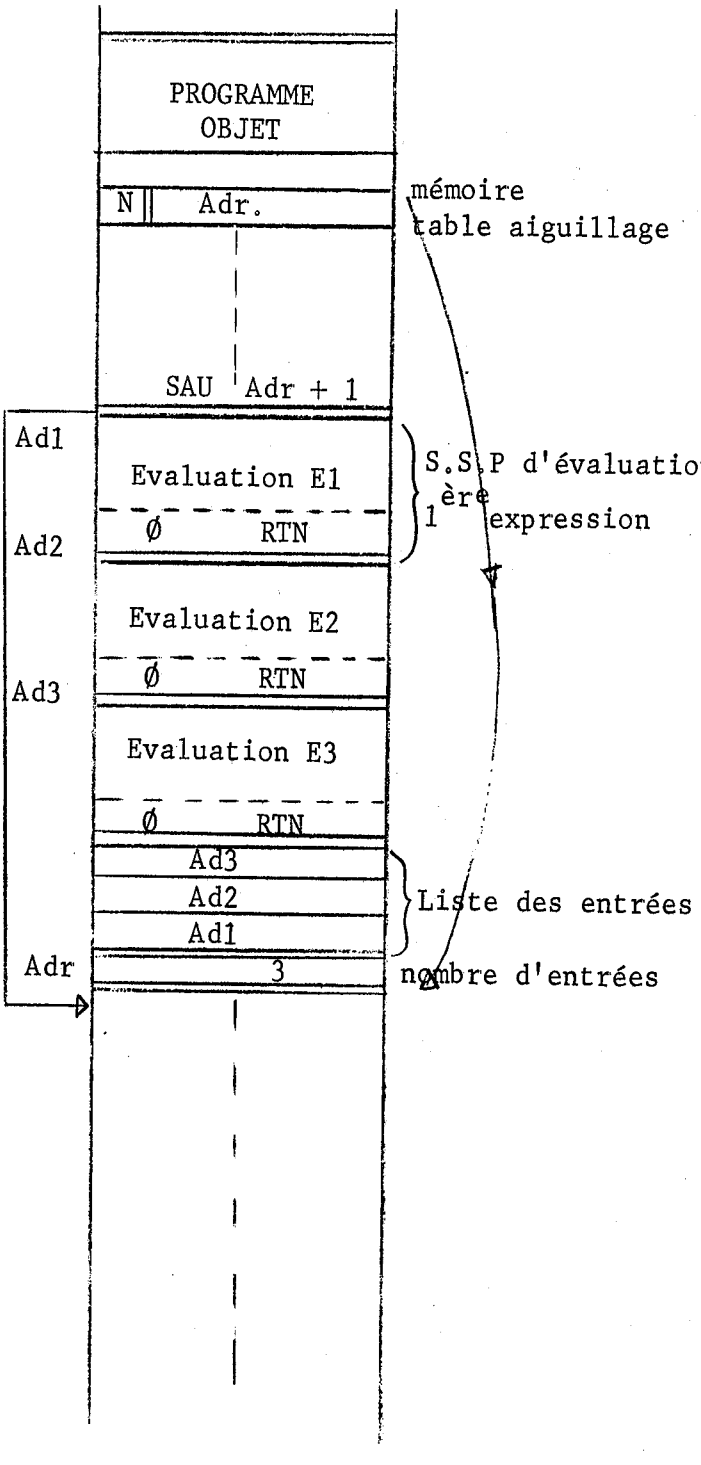
On a indiqué que l'activation de la "procédure" Aiguillage (et donc la "fonction standard" appelée) devait permettre la récursivité, puisque la déclaration d'aiguillage source peut faire intervenir des indicateurs d'aiguillage ou des procédures par exemple. Comme le travail se limite au calcul d'un paramètre effectif, on n'aura pas de problème particulier : un tel sous programme d'évaluation traite naturellement la récursivité. Le paramètre effectif de l'appel "procédure" aiguillage (l'indice) n'est utile qu'une seule fois au début, et il est inutile de faire des entrées de procédures.

Le seul problème est celui où l'aiguillage "activé", est un aiguillage paramètre : l'analogue de l'activation d'une procédure formel, pour les mêmes raisons, il suffit de se replacer avant l'exécution dans sa structure de blocs origine ; c'est d'ailleurs le procédé standard d'activation des sous programmes d'évaluation. Mais il faut conserver dans ce cas, la structrue de blocs origine de l'aiguillage, pour l'indiquer avec l'étiquette résultat, si elle lui appartient : on laisse alors l'accumulateur correspondant en stack.

Le_principe_d'exécution_de_Ø_AIG_est_le_suivant :

a) La valeur I de l'indice est tranférée en valeur de type entier, si elle ne l'est pas (1er accumulateur).

Si $I \leq 0$ la valeur de l'indicateur n'est pas définie (on peut déclencher un constat d'"erreur"). L'adresse-table d'aiguillage permet d'obtenir l'adresse de la mémoire caractéristique, d'où le nombre L d'entrées.
Si $I > L$ l'indicateur d'aiguillage n'est pas défini. Or une instruction ALLERA est équivalente à une instruction vide lorsque l'expression de



désignation est un indicateur d'aiguillage dont la valeur n'est pas défini". [1] Dans ce cas on laisse comme résultat d'évaluation une étiquette fictive (-1 dans le 2^d mot du 2^d accumulateur et l'ordre Ø AIG terminé).

b) On va maintenant libérer les accumulateurs Indice et Aiguillage (ou seul le premier) et préparer l'évaluation de l'expression de désignation sélectée par I. L'ordre Ø AIG activera le sous-programme objet correspondant grace au mécanisme de l'interpréteur décrit en 7.5., et les informations communiquées à ce mécanisme sont repérées par le nom de ses phases.

. I détermine dans la liste d'entrée, l'adresse du sous programme à activer (phase 7.5.c).

. Si l'aiguillage est un paramètre effectif on ne libère que le 1er accumulateur, conservant celui de l'aiguillage. On retient alors l'adresse du sous programme de l'interpréteur qu'il faudra exécuter après évaluation de l'expression, et l'adresse de Cdy (3ème mot de l'accumulateur aiguillage) qui définit la structure pour l'activation (phase 7.5.a-b)

. Si l'aiguillage n'est pas un paramètre, c'est - à -dire qu'il est activé dans un bloc appartenant à sa portée, on retient qu'il est inutile de faire la remise à jour avant activation (phase 7.5.b.). En outre au retour de l'évaluation l'étiquette-résultat indiquera bien sa structure ; le 3ème mot de l'accumulateur :

α) sera nul : l'étiquette appartient donc à la structure de l'aiguillage, et est en effet exploitable directement puisque nous y sommes.

ou

β) donnera l'adresse Cdy définissant la structure de l'étiquette.

On libère donc les 2 accumulateurs au sommet du stack (indice et aiguillage), et on retient que l'ordre Ø AIG ne nécessitera aucune opération particulière au retour du sous programme activé (phase 7.5.a). Ce traitement de l'ordre Ø AIG doit faire gagner du temps dans les cas usuels.

c) l'ordre Ø AIG active par le mécanisme 7.5., le sous programme objet expression de désignation. Au retour de son exécution les opérations interprèteur précisées à l'activation, achèvent l'ordre Ø AIG. Elles sont examinées en 7.6.

7.5. Activation d'un sous programme de calcul de paramètre effectif.

Un sous programme paramètre effectif est constitué de la séquence objet d'évaluation d'une expression, ou de calcul de l'adresse d'une variable indicée, et est terminé par l'ordre Ø RTN (9.2.). L'évaluation d'un indicateur fait intervenir l'activation d'un tel sous programme, aussi l'examinons nous ici.

Il n'existe pas d'ordre objet d'activation d'un sous programme de paramètre effectif ; mais les ordres sur les formels utilisent un sous

programme-interprèteur commun d'activation, s'ils reconnaissent que le paramètre effectif de l'appel de procédure est donné par un sous programme objet. Son exécution ne suffit généralement pas à fournir le résultat désiré par l'ordre formel. Supposons que l'on veuille par exemple la valeur d'un paramètre, pour un formel spécifié <TYPE>. Le sous programme exécuté a pu fournir en stack, soit la valeur d'une expression, soit l'adresse d'une variable indicée. Dans ce dernier cas il faut d'abord obtenir la valeur. Puis pour un paramètre arithmétique, on accordera le type effectif au type spécifié si nécessaire.

On pourrait scinder les ordres sur formels en 2 ordres objets: le second se chargerait en particulier des opérations nécessaires lorsqu'il y a en activation de sous programme (le premier n'aurait alors fait que l'activation).

Nous ne l'avons pas fait car ce serait moins efficace à l'exécution et allongerait le programme objet. L'ordre sur un formel indique alors, au moment d'activer le sous programme de paramètre effectif, l'adresse du sous programme-interpréteur à exécuter au retour.

Ces quelques indications ont permis de replacer dans l'exécution d'un ordre formel (on se reportera au ch. 9), l'activation et le retour d'un sous programme de paramètre effectif. Nous verrons ces mécanismes maintenant.

Mécanisme - Interpréteur d'activation. Le problème est proche d'une activation de procédure : on quitte la structure de blocs où l'on est, pour évaluer le paramètre effectif dans sa structure origine. Le sous programme interpréteur d'activation crée l'équivalent d'une entrée de procédure dans le stack. Le S.S.P. d'évaluation se conduit comme un "bloc", où l'ordre Ø RTN qui le

termine joue le rôle de l'ordre Ø SBL achevant l'exécution d'une procédure.

a) Comme le problème est plus simple l'entrée créée dans le stack se réduit à 4 informations dites données de retour. Ce sont : les valeurs des 3 index :

XCP → Adresse de retour dans le programme objet
DP → DP_N
NB → NB_N
caractérisant le bloc appelant et sa structure

} Identique à une entrée de procédure

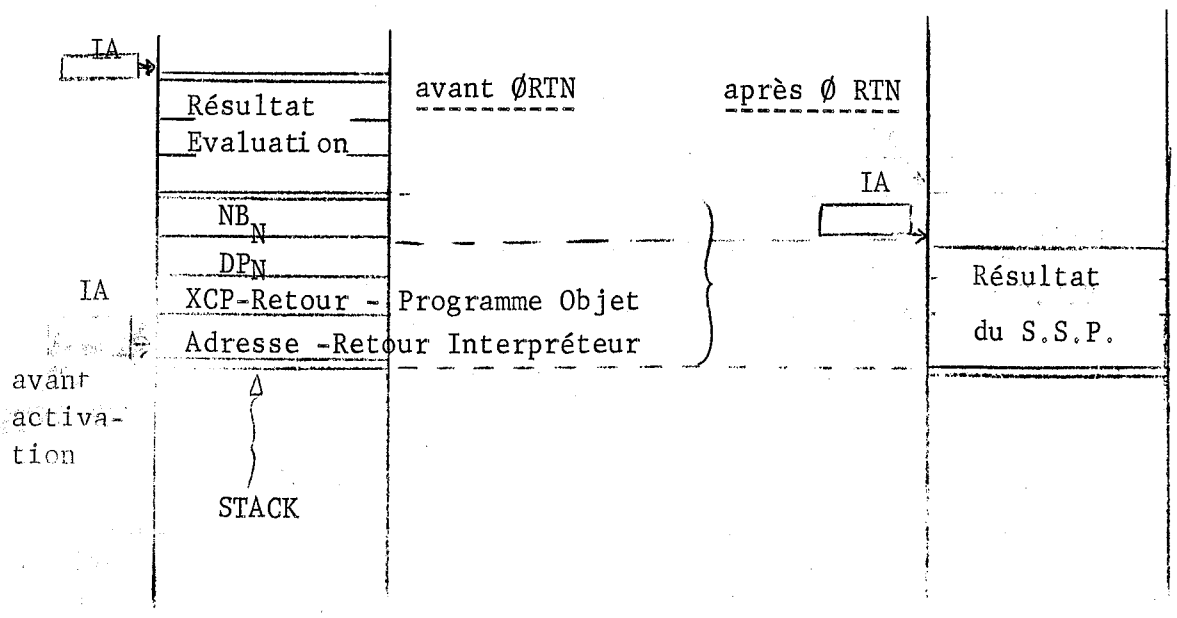
et l'adresse de retour à un sous programme de l'interpréteur fournie par l'ordre formel qui a provoqué l'activation. La récursivité conduit en effet à la placer en stack : elle est différente selon l'ordre formel (et diffère même pour un même ordre selon le paramètre effectif).

b) On se repositionne dans le bloc et la structure origines du paramètre effectif [La Cdy donne NB_K, DP_K et le sous programme interpréteur standard met à jour NB, DP et le display].

c) On passe à l'activation proprement dite : mise à jour de XCP avec l'adresse d'entrée du sous programme objet, et lancement de son exécution (par un saut à l'entrée de l'interpréteur).

7.5.1. Fin d'activation d'un sous-programme - Ordre Ø RTN

Le résultat du sous programme est dans l'accumulateur au sommet du stack, au dessus des données de retour mises en place avant l'activation. On va alors exécuter l'ordre Ø RTN.



Son principe d'exécution est le suivant :

. Les données de retour NB_N et DP_N servent à la mise à jour de NB et DP puis du Display. On est donc revenu au bloc qui a provoqué l'activation, et à sa structure de blocs.

. Le compteur programme XCP prend la valeur XCP-Retour, adresse du prochain ordre objet à exécuter (1ere donnée de Retour).
Le traitement est jusqu'ici identique à une fin d'activation de procédure.

. L'adresse de retour à l'interpréteur est prélevée ; mais avant d'y sauter, le résultat d'activation du sous programme, est régressé (de 4 mots), pour être implanté normalement : c'est à dire à l'emplacement du premier accumulateur libre au sommet du stack avant l'activation.

L'ordre Ø RTN est terminé ; on passe alors à l'exécution du sous programme de l'interpréteur, dont on a obtenu l'adresse. C'est ce sous programme interpréteur qui achève l'exécution de l'ordre objet ayant provoqué l'activation du sous programme de paramètre effectif : il se terminera par un saut à l'entrée de l'interpréteur. On exécutera alors l'ordre objet pointé par XCP.
L'ordre Ø RTN lui-même est donc un peu spécial, puisque c'est le seul pour lequel la fin d'exécution n'est pas un saut à l'entrée de l'interpréteur.

7.6. Achèvement de l'exécution de l'ordre Ø AIG

. Si l'aiguillage n'était pas formel, il n'y avait rien à faire, et l'adresse de retour dans l'interpréteur, indique l'entrée de l'interpréteur : l'ordre Ø AIG s'achève donc automatiquement avec l'exécution de Ø RTN.

. Au contraire pour un aiguillage formel, on effectue au retour un sous programme spécial de l'interpréteur.

On a conservé l'accumulateur de l'adresse d'aiguillage, car il donne l'adresse de la caractérisation dynamique de l'aiguillage paramètre effectif. (Cdy).

Au dessus, le dernier accumulateur occupé, contient l'étiquette résultat d'évaluation de l'indicateur. Deux cas peuvent se produire :

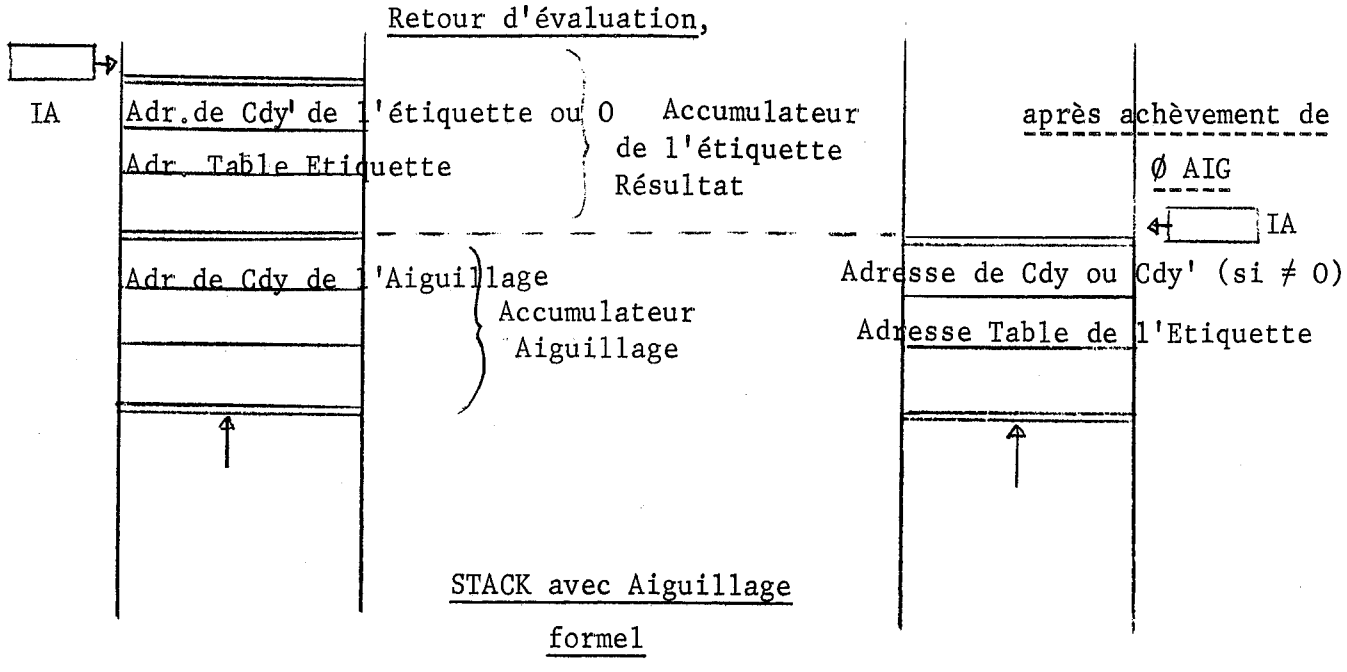
a) Cette étiquette n'est pas un paramètre effectif, elle appartient donc à la structure de blocs de l'aiguillage. (3ème mot de l'accumulateur de l'étiquette nul).

b) L'étiquette est un paramètre effectif ; sa propre structure de blocs est indiquée par l'adresse physique de Cdy (caractérisation dynamique) qui lui est adjointe en accumulateur.

Ceci provient simplement que dans la déclaration d'aiguillage du texte source, l'expression de désignation considérée, revenait à indiquer un paramètre formel (b) ou non (a).

Aussi, on se contente dans le cas (a), de placer l'étiquette résultat, dans l'accumulateur précédent, sans toucher le 3ème mot qui caractérise bien la structure de blocs de l'étiquette ; au contraire dans le cas

(b), ce 3ème mot est remplacé par celui de l'accumulateur étiquette regressé en bloc. L'ordre Ø AIG est alors achevé.



C H A P I T R E VIII

INSTRUCTION POUR

8.1. Généralités

8.1.1. Sous Programme d'Instruction - Ordres Ø DO et Ø RTP

L'instruction S, sur laquelle porte la proposition POUR, sera constituée comme un sous-programme. Ceci est tout à fait naturel puisque le compilateur ne génère qu'une seule fois l'instruction S, au moment ou elle se présente ; mais elle peut être exécutée plusieurs fois, selon les éléments de la liste de POUR. On doit donc pouvoir l'activer de plusieurs endroits différents. Le seul problème est celui de l'adresse de retour R_p dans le programme objet après exécution de S.

Le traitement doit permettre la récursivité : une instruction POUR peut l'introduire si elle fait intervenir des procédures. Une façon simple d'y parvenir est d'attacher à chaque instruction POUR une variable locale P du niveau où elle est donnée. (Les variables P entrent dans l'indicateur de réservation du bloc, et sont les dernières variables locales du bloc, après les variables simples et des tableaux déclarés - voir 3.1.).

La valeur de P sera l'adresse de retour R_p lors de l'exécution. Il est en effet inutile de constituer S en un véritable bloc, avec les ordres d'entrée et de sortie correspondants ; on ne peut, non plus, se contenter de laisser R_p dans le stack de travail : les instructions ALLERA contenues dans S provoqueraient des difficultés, si elles conduisent en dehors de S, spécialement dans le cas d'instruction POUR imbriquées.

Le sous programme de l'instruction S a donc la constitution suivante (fig. 8.1.). :

a) adresse de la variable locale P attachée à l'instruction POUR
b) le programme objet de S
c) l'ordre de Retour-Pour : Ø RTP ; le second mot opérande indique l'adresse de la variable P.

L'adresse dynamique de P se réduit à une adresse relative : sachant qu'elle appartient au niveau courant, on ne note pas le numéro de bloc.

Le programme objet de l'instruction POUR commence par un ordre SAU au premier élément de la liste de POUR (8.2.). Il enjambe le mot en séquence (mémoire M_P) qui contient l'adresse d'entrée (Adr. S), de sous programme-Instruction S. Tous les ordres d'activation réfèrent M_P, et la seule justification de cet adressage indirect est de simplifier la génération.

L'ordre d'activation de S est l'ordre Ø DO ; il est suivi de 2 mots opérandes 1) l'adresse de la mémoire M_P
2) l'adresse R_P de l'ordre objet à exécuter après le sous programme.

Son exécution se limite à placer l'adresse de retour R_P, dans la mémoire locale de P, et à mettre à jour le compteur programme XCP (:= Adr. S + 1).

On exécutera donc ensuite le 1er ordre du programme objet de S. L'ordre Ø RTP qui exécute, on l'a vu, la fin d'activation de ce sous programme, se borne à repositionner XCP grace à la valeur de P (égale à R_P).

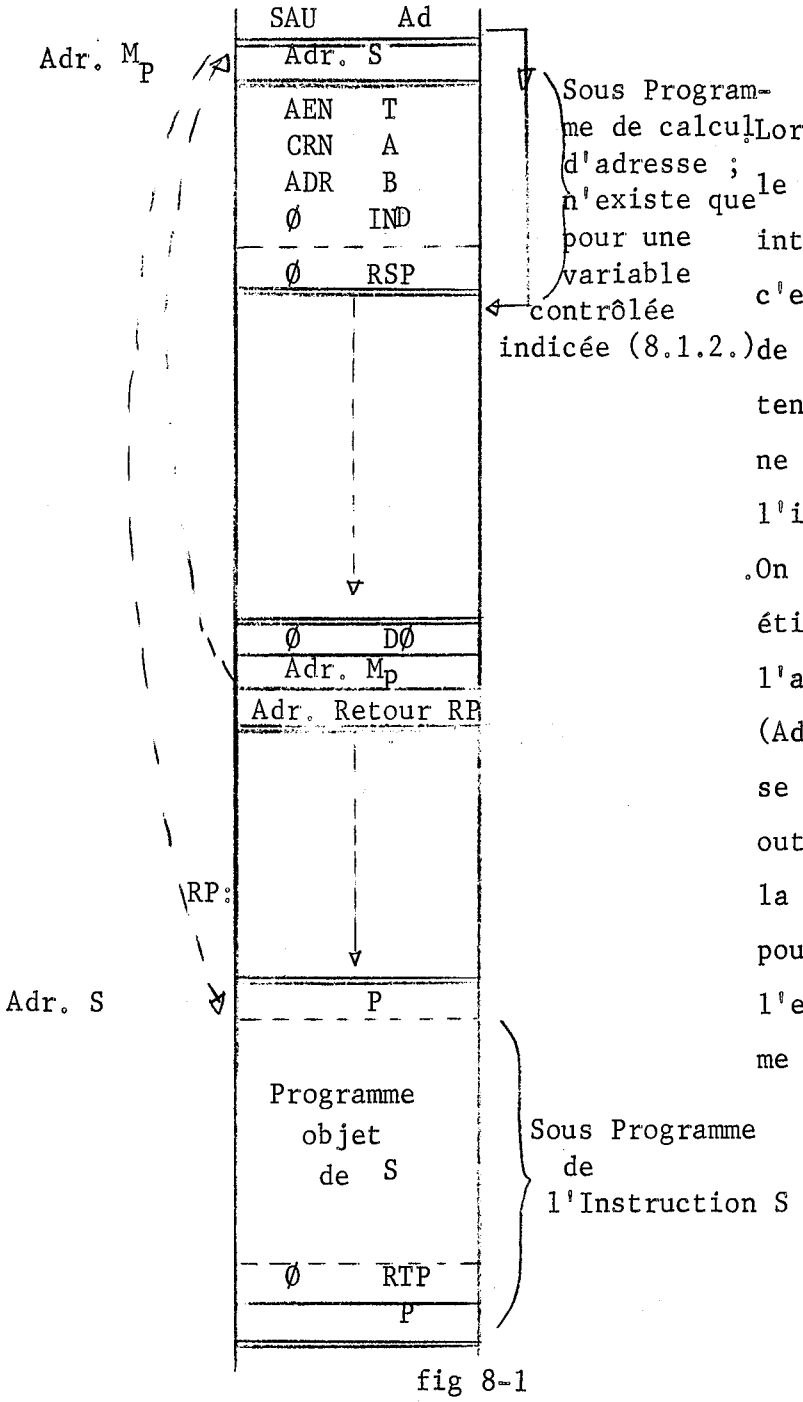


fig 8-1

Lorsque S est instruction composée le compilateur localise les étiquettes intérieures à S dans cette instruction : c'est-à-dire que tout en ayant le numéro de bloc courant, elles n'ont pas d'existence en dehors de S. Aucun saut erronné ne peut donc se produire, du dehors de l'instruction POUR, dans S.

.On remarquera que si S est elle même étiquetée, l'étiquette correspond à l'adresse du 1er ordre généré pour S (Adr. S+1). L'instruction peut donc se "rappeler" sans inconvénient. En outre cet étiquetage est signalé dès la phase codification en compilation pour éviter un saut erronné à S de l'extérieur du POUR. (Exemple : programme P IX).

8.1.2. Adresse de la variable contrôlée - Ordres SSP et Ø RSP

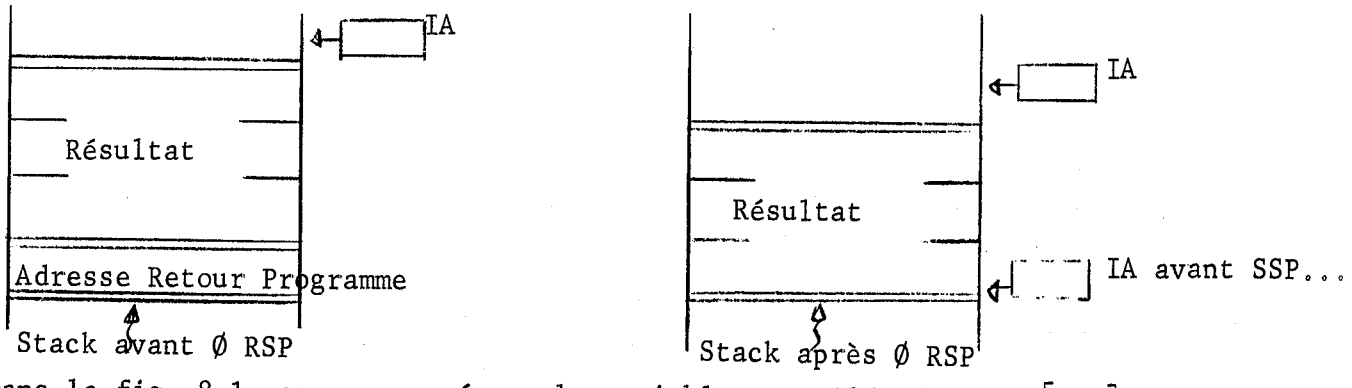
L'instruction S peut modifier la variable contrôlée par effet de bord, si celle-ci est une variable indicée ou un paramètre formel appelé par nom. On recalculera dons systématiquement l'adresse de la variable contrôlée quelle qu'elle soit, à chaque utilisation.

Dans le cas d'une variable indicée il faut constituer un sous programme de calcul de son adresse ; (il est naturellement en tête de l'instruction POUR, après la mémoire MP). S'agissant d'un simple calcul, dans le bloc courant, il suffit de placer l'adresse de retour, dans le stack de travail, pour traiter la récursivité.

L'ordre d'activation est SSP Adr. Programme . Il empile dans le stack la valeur du compteur programme XCP (adresse retour := celle de l'ordre en séquence), puis il positionne XCP sur l'entrée du sous programme (partie adresse de l'ordre).

Il y a alors l'exécution du sous programme constitué par une séquence classique de calcul d'adresse de variable indicée, et terminé par l'ordre de retour Ø RSP (fig. 8.1.)

Au moment de l'exécution de Ø RSP , l'accumulateur au sommet du stack contient le résultat et surmonte l'adresse de retour : il suffit de la prélever pour mettre à jour XCP, et régresser le résultat de 1 mémoire.



Dans la fig. 8.1. on a supposé que la variable contrôlée était T [A+B] avec des déclarations convenables.

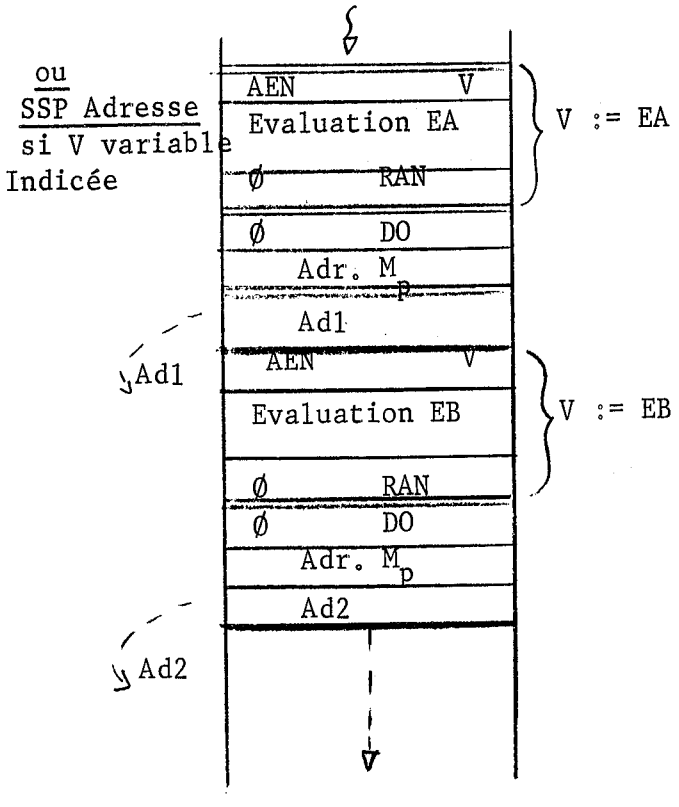
Nous allons maintenant examiner les caractéristiques du programme objet suivant les différents éléments de liste de POUR. Il reflète d'ailleurs exactement les règles données en 4.6.4. du Revised Report. [1]

Le compilateur a vérifié que la variable contrôlée est arithmétique et que les éléments de la liste font intervenir des expressions de type correct.

On appellera successeur d'un élément de liste, l'élément de liste suivant s'il existe, sinon l'instruction suivant l'instruction POUR.

8.2. Elément de Liste POUR 1 (Expression arithmétique)

Exemple POUR V := EA, EB,..... FAIRE S ;

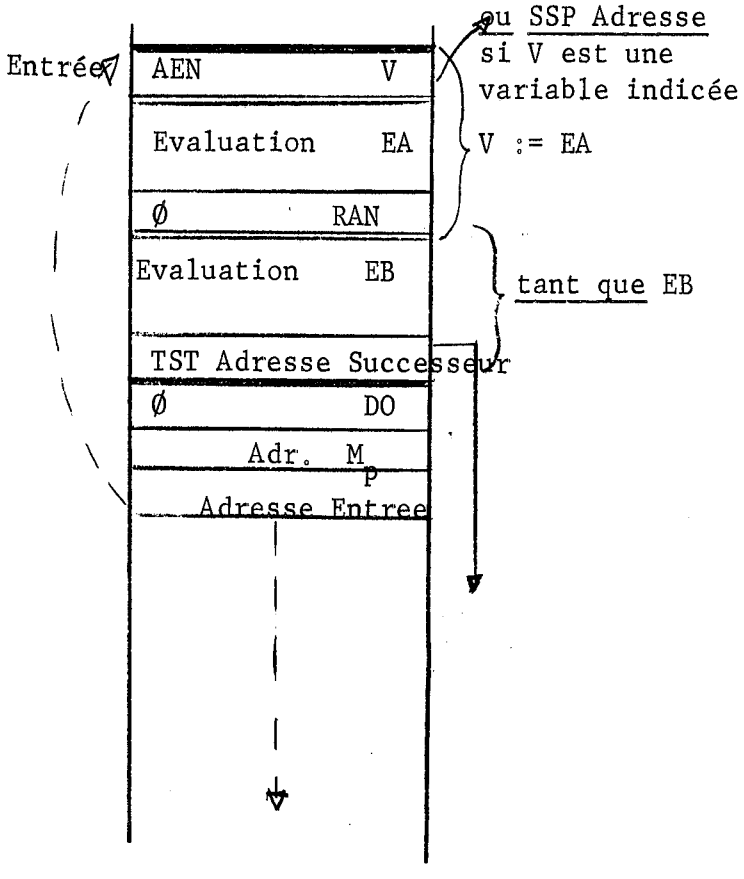


Le programme objet est une simple affectation suivie de l'ordre Ø DØ d'activation de S.
L'adresse de retour indiquée est celle du successeur ; si c'est un élément de liste il est en séquence.
Dans l'exemple ci-contre, on a supposé que V est une variable simple entière. Ce peut être un formel, on aurait alors un ordre AEF ou ARF. Si c'est une variable indicée on a SSP Adr. MP+1 que nous indiquerons SSP Adresse.

Exemples : programmes PV et PVI

8.3. Elément de Liste POUR 2 (Tant que)

Exemple POUR V := EA TANT QUE EB ,.....,FAIRE S ;



Le programme généré est encore une affectation simple suivie d'un test sur la valeur de l'expression booléenne, avant de donner l'ordre d'activation Ø DO.

L'ordre TST a déjà été employé pour les expressions ; l'adresse de saut conditionnel (si FAUX) est celle du successeur.

L'adresse de retour après exécution de S, indiquée dans l'ordre Ø DO est celle d'entrée dans l'élément de liste POUR 2, soit de l'affectation.

Exemples : Programmes PV et PVI

8.4. Elément de liste POUR 3 (PAS JUSQUA)

8.4.1. Cas général - Ordres Ø VAL et TFA.

Exemple POUR V := EA PAS EB JUSQUA EC,.....FAIRE S ;

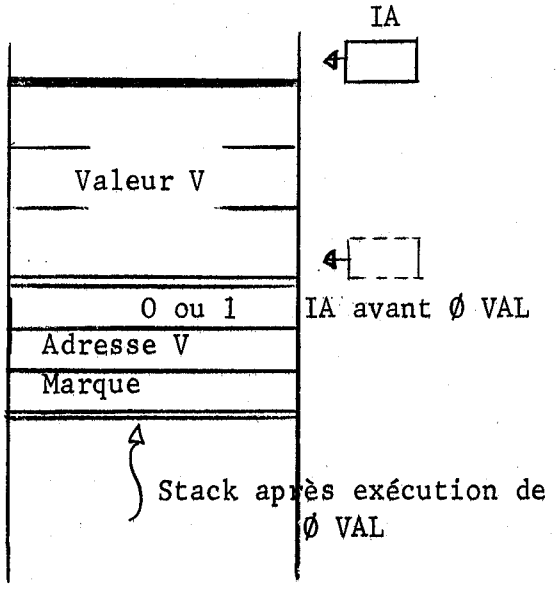
Le programme objet correspond à l'écriture

```
        V := EA ; ALLERA L2 ;
L1 : V := V + EB ; L2 : SI (V-EC) x SIGN (EB) > 0 ALORS ALLERA SUCCESSEUR ;
        S ;    ALLERA L1 ;
```

Ceci est équivalent à la règle du Revised Report on ALGOL 60 ([I] 4.6.4.2.), et on peut remarquer qu'elle conduit à évaluer 2 fois l'expression EB.

Lorsque le programme objet d'évaluation du "pas" EB comporte plus d'un ordre, on en fait un sous programme de calcul ; ce dernier est constitué comme celui du calcul d'adresse de variable contrôlée indicée, et s'active de la même manière. On n'aura pas de sous programme, si le pas se réduit à une variable simple, un nombre sans signe (ou signe +), un formel spécifié REEL ou ENTIER (sous réserve que le délimiteur JUSQUA soit sur la même ligne)

On devra exécuter V := V + EB ; d'où l'intérêt d'un ordre, qui à partir de l'adresse de la variable contenue dans le dernier accumulateur en stack, prend la valeur de cet opérande, et l'empile au dessus de l'adresse dans le 1er accumulateur libre.

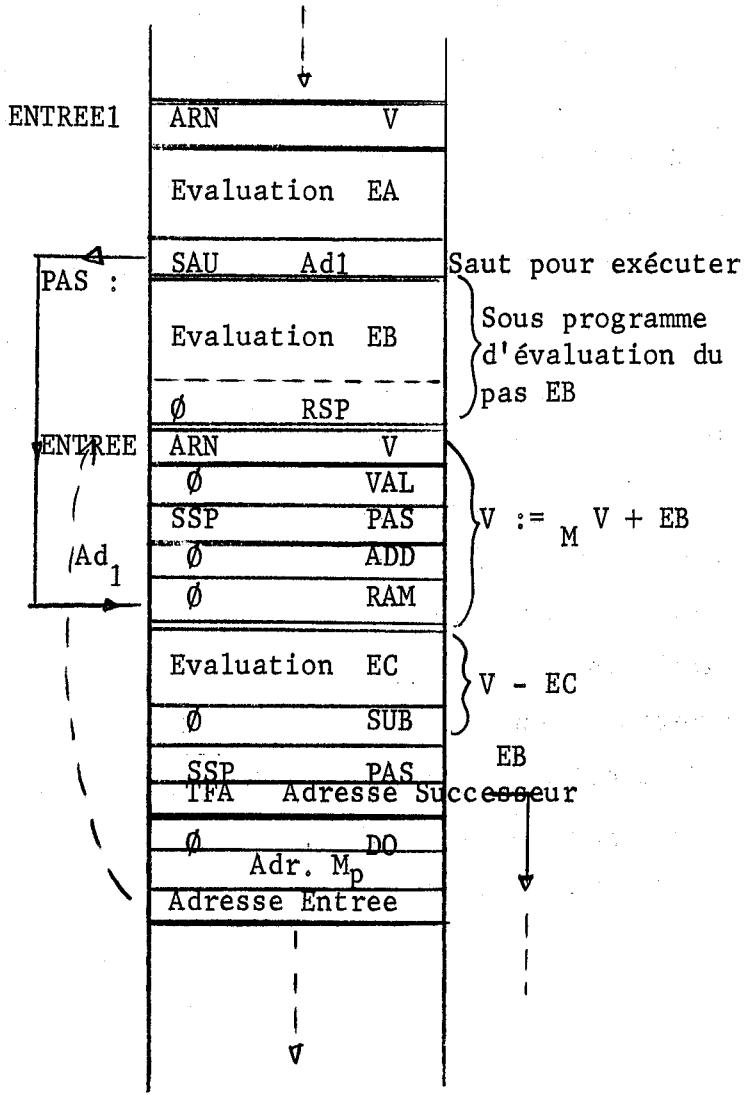


C'est le rôle de l'ordre Ø VAL.

Il s'exécute de façon analogue au sous programme interpréteur terminant l'ordre Ø PVA, après calcul de l'adresse d'une variable indicée. Il traite aussi la conversion éventuelle de la valeur, si c'est celle d'un paramètre effectif dont le type diffère du type spécifié pour le formel (6.2.c).

Pour affecter la valeur à V on utilise l'ordre d'affectation multiple Ø RAM, et non celui de l'affectation simple : cela permet de laisser la valeur de V en stack pour le calcul qui suit de V-EC.

Le test de la condition d'épuisement est effectué par un seul ordre : TFA Adresse Programme. Au moment de son exécution les 2 accumulateurs au sommet du stack contiennent les valeurs V-EC et EB. Le rôle de TFA est de libérer ces 2 accumulateurs, et si la condition d'épuisement (V-EC) x SIGN(EB)>0 {soit '(IA-2)' x SIGN ('(IA-1)'>0} est VRAI de sauter au successeur : mise à jour du compteur programme XCP avec la partie adresse de l'ordre. Sinon on exécutera normalement l'ordre en séquence.



Structure du programme objet.

Les ordres ARN V sont remplacés par des ordres SSP Adresse . Si V est une variable indicée.

Ce sous programme peut ne pas exister ; les ordres SSP PAS sont remplacés par les ordres ARN, ARF, CRS...

L'adresse d"Entrée 1" qui correspond à la première affectation est prise comme adresse de successeur par l'élément précédent, ou par l'ordre SAU introduisant l'instruction POUR.

Exemples Programmes P VII, P VIII, PX

8.4.2. Cas particulier optimisé : Ordre Ø FAI

Le compilateur reconnait un cas d'optimisation de l'élément de liste POUR 3, et produit un ordre condensé Ø FAI lorsque :

a) la variable contrôlée est une variable simple de type entier (non un formel)
b) le pas (EB) est un nombre entier ≥ 0
c) la valeur finale (EC) est un nombre entier ≥ 0 (sous réserve que le délimiteur FAIRE ou , soit sur la même ligne).
Exemple POUR I := EA PAS 10 JUSQUA 100,.....FAIRE S ;

La première affectation est effectuée séparément, et donne lieu à son propre ordre d'activation Ø DO. Comme le signe du pas est connu, on la traduit sensiblement par I := EA ; SI I ≤ VALEUR FINALE ALORS S SINON ALLERA L2 ; L'ordre Ø FAI effectue :

L1 : I := I + PAS ; SI I ≤ VALEUR FINALE ALORS
DEBUT S ; ALLERA L1 FIN ;

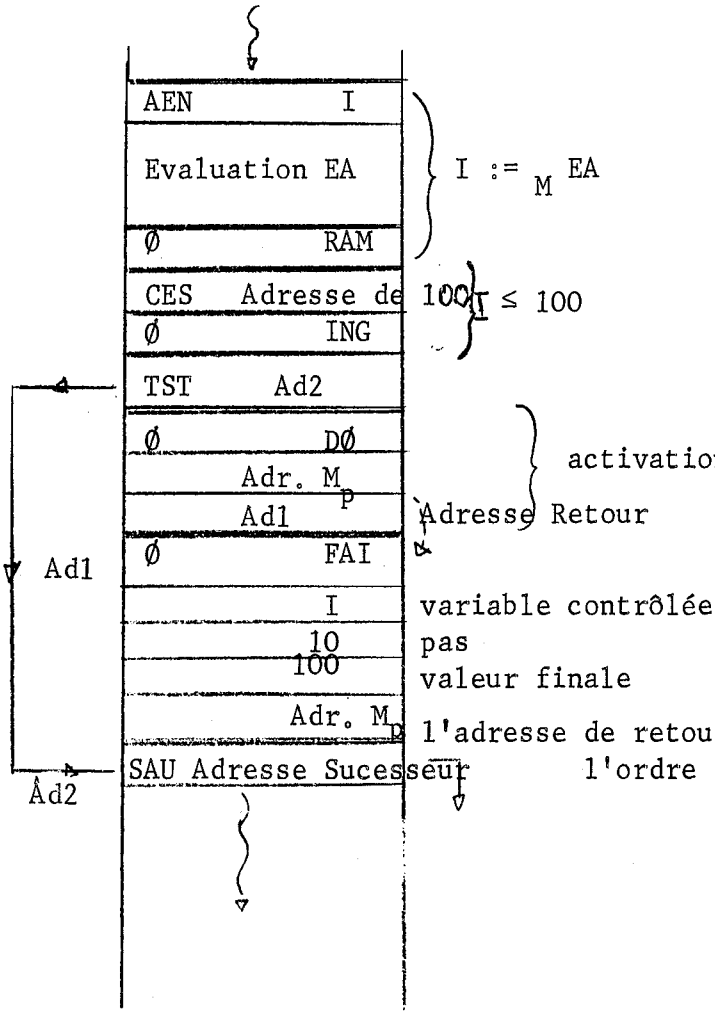
Et l'instruction suivante est équivalente à L2 : ALLERA SUCCESSEUR

L'ordre Ø FAI est suivi de 4 mots opérandes ; ce sont
a) la variable contrôlée: adresse dynamique
b) la valeur du pas
c) la valeur de la valeur finale
d) l'adresse de la mémoire M_p du programme (8.1.1.).

Cet ordre exécute à la fois la progression de la variable, le test d'épuisement, et l'activation éventuelle du sous programme de S comme un ordre Ø DO :

Si la condition d'épuisement n'est pas remplie, on active le SSP de S avec comme adresse de retour, l'adresse de l'ordre Ø FAI lui-même. Sinon on passe à l'exécution de l'ordre en séquence du Ø FAI.

Ce dernier est donc un ordre de saut au successeur.



Forme de programme objet

Exemples Programmes P IX, P XIII

Remarques

- on est obligé de calculer à chaque fois l'adresse physique de la variable contrôlée.
- le pas peut être nul. L'élément de liste POUR 3 optimisé est immédiatement épuisé si la valeur initiale EA est supérieure à la valeur finale ; dans le cas contraire le résultat est conforme au Rapport Algol (élément jamais épuisé).

C H A P I T R E IX

PARAMETRES

Les paramètres d'une procédure permettent la communication entre le corps de procédure, et l'endroit du programme où elle est activée. D'un point de vue général ils font intervenir, à l'exécution du corps de procédure, des quantités qui n'y sont pas normalement accessibles : leur portée ne s'y étend pas, sauf cas particulier d'activation.

Le Rapport Algol précise (§ 4.7.3.) que l'exécution d'une procédure dont le corps est une instruction écrite en Algol, équivaut à la règle de recopie suivante :

a) modifier l'écriture du corps de procédure en remplaçant les paramètres formels par les paramètres effectifs selon des règles indiquées.

b) Insérer le corps de procédure ainsi modifié à la place de l'instruction procédure et l'exécuter.

Les conflits entre identificateurs ne se posent pas, ou tout au moins le traitement le résout, comme nous le verrons. Seuls les paramètres effectifs changent d'une activation à l'autre, et donneront lieu à une recopie ; il n'y a qu'un seul corps de procédure dans le programme objet, ceci est naturel et constitue l'unique possibilité avant l'exécution.

Sa génération est limitée, justement par ignorance des paramètres effectifs. On impose la spécification de tous les paramètres formels, restriction importante mais usuelle du langage Algol, pour 2 raisons très liées :

on s'assure à la compilation que le corps de procédure conduira certainement à l'exécution d'une instruction Algol correcte (si les règles de correspondance paramètres effectifs - formels sont respectées). On facilite la génération en pouvant produire des ordres sur formels qui précisent ce qui est attendu du paramètre effectif ; ce ne serait pas aisément soluble en l'absence de spécification. On doit considérer ces ordres formels comme des ordres précompilés, qui permettent de traiter les formels, dans le programme objet, de façon analogue aux quantités déclarées.

Par ailleurs on a généré l'appel d'une procédure en précisant chacun de ses paramètres effectifs par une caractérisation statique Cst ; elle donne, dans le programme objet, l'adresse de la séquence d'évaluation si c'est un paramètre effectif expression, sinon l'identificateur.

A l'exécution du corps de procédure il suffirait alors de procéder de la manière suivante, pour obtenir l'équivalent de la règle de recopie d'un point de vue statique : on a recopié au moment de l'activation (appel par nom des paramètres) les caractérisations statiques pour l'appel considéré dans un emplacement E_P du programme objet du corps de procédure, emplacement réservé à cet effet par le compilateur. L'exécution ne présente de particularités que lorsque l'on rencontre un ordre sur formel : l'accès à EP permet de prélever le nom, ou de sauter dansle programme objet et exécuter la séquence de calcul relative au paramètre effectif. Après le traitement impliqué par l'ordre formel, on revient à nouveau dans le programme objet du corps de procédure pour continuer son exécution.

Mais cette optique est insuffisante; la règle de recopie doit être considérée du manière dynamique : une première activation de la procédure P, peut provoquer d'autres appels de P ; sans que la première soit achevée (récursivité). Ce qui doit être recopié, les caractérisations de

paramètres effectifs, ne peut donc l'être dans le programme objet : on les implantera dans le stack de travail à la suite des données de liaison d'entrée dans ce niveau. Ce sont les caractérisations dynamiques Cdy que l'on a déjà mentionnés (2. et 3.2.).

Le problème de la récursivité est ainsi traité de la même manière que pour les variables locales ; un ordre sur formel considère bien le paramètre effectif de la dernière activation. Il y accède par la Cdy qu'il réfère ou groupe de mémoires locales dont la "valeur" affectée à l'activation de la procédure caractérise le paramètre effectif. (aussi à t'on qualifier les Cdy de variables locales au sens large).

Si la Cdy est une copie dans le stack, de la caractérisation statique donnée dans le programme objet, il faut cependant lui adjoindre (appel par nom) la caractérisation de la structure de blocs à laquelle appartient le paramètre effectif : il fait intervenir en effet, des quantités dont la portée s'étend au niveau où il est réellement donné, et attachées à l'implantation d'exécution de ce niveau. C'est en ce sens que sont résolus les conflits d'identificateurs mentionnés par 4.7.3. du Rapport Algol.

Plus simplement, soit un paramètre effectif (appel par nom) qui est une expression ; en exécutant le corps de procédure on désire en obtenir la valeur : l'évaluation de la séquence correspondante ne peut se faire qu'en se plaçant dans le bloc où est donnée l'activation de procédure (règle de recopie). On sort donc de la structure de blocs actuelle, d'exécution de la procédure ; il faut repositionner correctement les index et le display. Dans le cas présent il suffirait de se référer aux 3ème et 4ème données de liaison du niveau procédure (elles indiquent bien la structure où l'on revient). Mais ce ne serait pas commode, et ne serait pas valable lorsque le paramètre effectif est déjà formel.

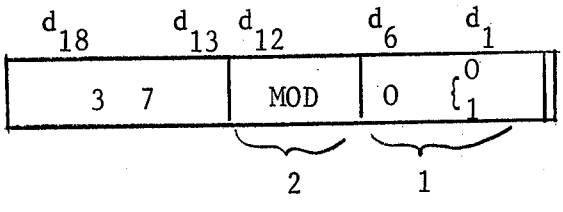
Les appels de procédure sont générés à la compilation sans vérifier que les paramètres effectifs correspondent aux spécifications des formels de la procédure à activer. Aussi un des rôles des modèles de formels, qui existent en tête du programme objet de la procédure, est de permettre cette vérification à l'exécution. Le mécanisme d'activation l'effectue grace à l'utilisation conjointe des Cst, et construit simultanément les Cdy.

9.1. Modèle de paramètre formel

La tête de procédure (.4.2.) est constituée dès la première phase de la compilation, la codification. A chaque formel correspond un modèle (1 mot) ; la liste de ceux-ci respecte l'ordre de la liste de paramètres formels de la déclaration.

Le modèle reflète la spécification et le mode d'appel.

Format



1) Mode d'appel d_1-d_6 ; bit significatif d_1

appel par nom d_1-d_6 nuls (octal 00)

appel par valeur d_1=1, d_2-d_5 nuls (octal 01)

2) Spécification d_7-d_{12} ; bits significatifs d_9-d_{12}

$d_7 = d_8 = 0$

$d_9 - d_{12}$ = spécification (identique aux bits $d_{14} - d_{17}$ de la codification des identificateurs de quantités déclarées - sauf pour les chaînes).

3) Une partie non significative d_{13} - d_{18} = 37 octal provenant d'un indicatif de codification.

9.2. Caractérisation Statique de paramètre effectif : Cst

On a vu en 4.2.1. la structure du programme objet d'appel de procédure.

On a constitué un sous programme-paramètre effectif chaque fois qu'un paramètre effectif est

1°) une expression (arithmétique , booléenne ou de désignation)

2°) une variable indicée.

Ce sous programme est terminé par l'ordre de retour Ø RTN. Il se réduit par ailleurs à la séquence objet habituelle, dans le 1er cas : d'évaluation de l'expression puisque ce sera certainement à cette valeur que l'on s'intéressera dans laprocédure ;
. dans le 2^{d} cas : de calcul d'adresse de la variable indicée ; on ne peut en effet savoir, comme pour toute variable, si l'on s'intéressera dans la procédure à l'adresse, à la valeur ou aux deux.

Les sous programmes éventuels sont donnés dans l'ordre de la liste de paramètres.

Si le paramètre effectif se réduit à un identificateur, son équivalent, adresse statique de table dans le programme objet ou adresse dynamique de variable locale, détermine parfaitement la quantité correspondante ; de même pour un paramètre formel. Il n'y a pas besoin de sous programme.

Aussi la liste des caractérisations statiques de paramètres effectifs, donne-t-elle pour chacun de ceux ci, soit l'adresse du sous programme de calcul s'il y en a un, soit l'adresse équivalente à l'identificateur.

La liste de Cst placée juste avant l'ordre d'appel, est dans l'ordre inverse de la liste de paramètres effectifs.

Chaque Cst occupe un mot et comprend une partie qualitative Cs indiquant la catégorie du paramètre effectif. On reconnait le maximum de catégories de paramètre effectif (.2.3), soit par famille :

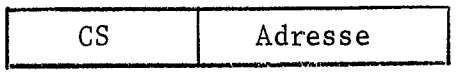
. identificateur de quantité : toutes les catégories de quantités subdivisées selon le type Algol (s'il existe) et de même pour les nombres sans signe.

. variable indicée (sous programme-adresse) : 3 catégories selon le type Algol.

. expression (sous programme-valeur) : 3 catégories selon le type arithmétique, booléen ou de désignation.

. puis formel, chaîne.

Format



$d_1 - d_{13}$ = Adresse { statique { de sous programme / de table ; dynamique de variable (selon le format habituel 3.2.)

On verra en annexe de ce chapitre les différentes Cst (appellation symbolique et Codage).

Remarques.

- La Cst d'un paramètre effectif, qui est un formel n'indique pas la spécification de ce formel, mais uniquement cette situation ; ceci est assez naturel. Par ailleurs l'adresse est celle de la Cdy correspondante.

- Un paramètre effectif nombre sans signe (ou positif) est considéré de manière analogue à une variable simple mais avec une adresse statique.

- Le type d'une variable indicée, d'un tableau formel, est celui qui est spécifié pour ce formel.

9.3. Caractérisation dynamique de paramètre effectif : Cdy

La caractérisation dynamique construite par le mécanisme d'activation de la procédure, grace à la Cst du paramètre effectif et au modèle du formel correspondant, occupe 3 mémoires locales en stack.

Les informations qu'elles contiennent sont les suivantes (fig. P. IX 1er mot.

a) Mod : (d_7-d_{12}). C'est la spécification du formel telle qu'elle est indiquée dans le modèle correspondant. Ceci est utilisé lorsque le paramètre formel correspondant est cité comme paramètre effectif dans un appel de procédure.

b) ORFS : (d_1-d_6) : ordre formel secondaire précise à l'ordre formel qui se réfère à la Cdy, les opérations à entreprendre.
Il se divise en ORFS 1 (d_4 d_3 d_2) indicatif, dont nous examinons plus loin la signification détaillée, et un indice de conversion d_1.

Cet indice n'a de sens que si l'on s'intéresse à un paramètre effectif susceptible de produire une valeur arithmétique : le formel a été spécifié

REEL ou REEL } TABLEAU
ENTIER ENTIER } PROCEDURE ;

il vaut alors 1 s'il n'y a pas identité entre le type spécifié pour le formel et celui du paramètre effectif, ou que ce dernier est inconnu (expression). Dans tous les autres cas il vaut 0.
L'indice permet d'imposer à la valeur du paramètre effectif, prise dans la procédure, le type spécifié pour le formel, et de traiter correctement les adresse dans une affectation à un formel.

C) NB_K : (d_{13} - d_{18}) voir 3ème mot.

. Pour les paramètres appelés par valeur, si les formels correspondants sont spécifiés <TYPE>, le 2^d mot et le 3ème mot contiennent la valeur sous le format correspondant de variable locale (1 mot entier, 2 mots flottant pour un réel). Cette situation est indiquée par ORFS.

Dans les autres cas on aura :

2^d Mot Une adresse

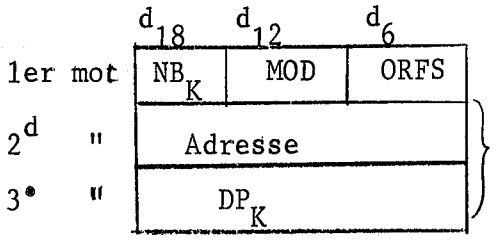
. dans le programme objet d'un sous programme de paramètre effectif.
. statique d'une mémoire-table étiquette, aiguillage, procédure.
. physique d'implantation d'une variable locale.

Ce S^d mot provient de la partie adresse de la Cst, directement sauf pour une variable simple, ou variable $\mathfrak{G}$: on a calculé l'adresse physique à partir de l'adresse dynamique donnée dans la Cst, car il est inutile de le refaire à chaque référence du paramètre (il s'agit toujours de la même variable).

3ème Mot : DP_K

qui constitue avec NB_K (d_{13} - d_{18} du 1er mot) la caractérisation de la structure de blocs origine du paramètre effectif : valeurs des index

DP et NB pour le bloc dit origine du paramètre effectif, où est donnée l'activation de procédure qui définit le paramètre effectif à utiliser réellement dans l'activation que nous considérons ; ces 2 activations ne sont pas forcément confondues (utilisation de paramètre formel en paramètre effectif). Cette notion déjà mentionnée sera bien définie en étudiant la création des Cdy.



ou valeur si paramètre spécifié <TYPE> appelé par valeur

Remarques

. Digits significatifs (les autres sont nuls) du 1er mot

NB_K : d_{13}-d_{16} (4)
MOD : d_9 - d_{12}(4)
ORFS : d_1 - d_4 (4)

. Pour un paramètre appelé par valeur et spécifié ETIQUETTE, la Cdy est identique à une Cdy de paramètre effectif réduit à une étiquette dans l'instruction source d'activation (le mode d'appel n'influe pas dans ce dernier cas).

. NB_K et DP_K sont en fait inutiles lorsque le paramètre effectif est une variable simple ou une variable (tableau), puisqu'on a déjà pris et placée l'adresse physique dans Cdy. Leur présence vient de la généralité du mécanisme de construction.

9.3.1. Caractérisation dynamique et ORFS-Résumé.

ORFS présent dans la Cdy, précise à l'ordre formel qui réfère la Cdy, la signification des informations contenues dans celle ci, donc le traitement à faire.

Sans revenir sur l'indice de conversion, on reconnait 5 situations explicitées dans le tableau ci-dessous.

Les vérifications que l'on a faites par le mécanisme d'activation sur la correspondance formel-effectif, permettent de ne distinguer qu'un minimum de cas. Variable et procédure sont séparées : l'ordre demandant la valeur d'un paramètre effectif (pour un formel spécifié <TYPE>) doit pouvoir reconnaitre une variable simple, d'un indicateur de fonction sans paramètre, sens ici de "Nom de Procédure".

Codage ORFS 1er mot de Cdy d_4 d_3 d_2 d_1	Nom Symbolique de ORFS	Signification	Indice de Conversion d_1 peut être ≠ 0 si	Contenu 2d mot de Cdy
0 0 0 $\{^0_1$	NOMP	.Nom de procédure, Etiquette, Aiguillage .Chaîne	formel spécifié REEL ENTIER ou REEL ENTIER } PROCEDURE	.Adresse de la mémoire entable, Procédure, Etiquette, Aiguillage .Adresse de la chaîne dans le programme objet.
0 0 1 $\{^0_1$	EXCV	Activer le sous programme ; le résultat est une valeur d'expression arithmétique, booléenne ou de désignation	d_1 = 1 si le paramètre effectif est une expression arithmétique	} Adresse du sous programme objet
0 1 0 $\{^0_1$	EXCA	Activer le sous programme ; le résultat est l'adresse d'une variable indicée	formel spécifié REEL ENTIER	
0 1 1 $\{^0_1$	NOM	.Nom d'une variable simple ou variable [Tableau]	formel spécifié REEL ENTIER ou REEL ENTIER } TABLEAU	. Adresse physique d'implantation de la variable
1 0 0 0	VAL	valeur d'une variable		valeur de la variable (éventuellement 3ème mot de Cdy si réel).

9.4. Construction des Caractérisations Dynamiques

. Le mécanisme d'activation traite intitialement tous les paramètres effectifs dans l'ordre de la liste, en les appelant par nom.
Il constitue les Cdy correspondantes dans le stack, à la suite des données de liaison , ce qui réalise l'implantation prévue à la compilation(un index pointe la Cst du paramètre effectif, un autre le modèle du formel. Le premier régresse dans la liste qui précède l'ordre d'appel, le 2^d progresse dans la tête de procédure).

. Puis il reprend, dansl'ordre de la liste, l'examen de chaque paramètre. S'il doit être appelé par valeur, on exécute l'ordre d'évaluation correspondant, et l'on range le résultat obtenu dans Cdy. On modifie alors le 1er mot de celle-ci pour que ORFS dénote bien ce fait, avant d'examiner le paramètre suivant.

Tant que l'on n'a pas constitué les Cdy en appel par nom, les index et le display restent positionnés pour le bloc appelant. On ne les mettra à jour, qu'après cette phase préliminaire où l'on a besoin de NB et de DP du bloc appelant.

9.4.1. Phase "préliminaire" de constitution des Cdy en appel par nom.

9.4.1.1. Le paramètre effectif n'est pas un formel. (soit CS ≠ 40 voir 9.2.)

La structure de blocs origine du paramètre effectif est celle du bloc appelant : NB_K et DP_K sont les valeurs actuelles des index DP et NB.

La partie adresse de la Cst donne directement le 2^d mot de la Cdy, ou après transformation de l'adresse dynamique en adresse physique, pour une variable simple ou une variable ℭ [Tableau] paramètre effectif.

L'élaboration de la partie ORFS de la Cdy, est faite en même temps que l'on vérifie la correspondance paramètre effectif-formel. Nous examinerons en détail le traitement et ses raisons, tout en précisant par quelques indications la technique utilisée ici ; il faut noter que cet aspect du problème est lié au codage des modèles et des parties qualitatives des caractérisations statiques ; sa solution est assez facilitée dans notre cas.

A. La Cst distingue, dans les 23 catégories, 5 familles de paramètres effectifs, qui sont pratiquement celles que reconnaîtra ORFS (9.3.1.).

Famille	Paramètre effectif
0	Variable simple - Variable ℭ [Tableau]
1	Procédure-Etiquette-Aiguillage-Chaîne
2	Variable indicée
3	Expression (arithémtique, booléenne, de désignation)
4	Nombre

Le numéro de famille est en général donné directement par les 2 premiers digits d_{18}-d_{17} de Cst ; cependant il faut ramener les chaînes à la famille 1 et distinguer les nombres sans signe des variables indicées. Ces nombres seront considérés comme des variables simples appelées par valeur, mais on ne peut les classer dans la famille 0 à ce point du traitement : leur adresse est statique, et l'adresse dynamique des variables n'est pas encore transformée en adresse physique. On crée donc une famille spéciale pour les nombres sans signe.

B. La spécification du formel impose au paramètre effectif d'être quelque chose de très précis. S'en assurer, est ce que nous avons appelé vérification de la correspondance paramètre effectif-formel. Faite par le mécanisme d'activation de la procédure, elle permet d'éviter toute vérification ultérieure d'un ordre formel au sujet du paramètre effectif : on est en effet certain d'être conduit à l'exécution d'une instruction Algol correcte, car le traducteur à déjà vérifié l'utilisation du paramètre, grace à sa spécification.

Ceci n'est pas tout à fait exact quand le formel est spécifié <TYPE> : s'il apparait en partie gauche d'une instruction d'affectation, il ne peut être remplacé par un paramètre effectif "expression", et on ne s'en apercevra qu'en exécutant l'instruction d'affectation [les ordres ARF et AEF le testent 9.5.1.]. Il aurait fallu noter dans le modèle des formels spécifiés <TYPE> le fait qu'ils apparaissent en partie gauche, pour pouvoir le vérifier en entrant dans la procédure.

La spécification doit être respectée par le paramètre effectif (cf. 4.7.5. de [1] mais nous admettrons :

. que le type d'un paramètre effectif susceptible de produire une valeur arithémtique ne soit pas celui de la spécification. (L'indice de conversion note cette situation (9.3.). Son utilisation est examinéeen 9.5.)

Cette extension d'Algol est pratiquement rendue nécessaire par la restriction imposée ailleurs : tous les formels doivent être spécifiés.

Notons qu'un paramètre effectif déclaré <TYPE> PROCEDURE est permis pour la spécification PROCEDURE du formel.

Ce tableau résume les restrictions imposées aux paramètres par la spécification :

Formel spécifié	Paramètre effectif admis à l'entrée de la Procédure
REEL ou ENTIER	Variable (simple et Indicée Réel ou Entier, Nombre sans signe REEL ou ENTIER) Expression arithmétique Procédure déclarée { REEL / ENTIER } PROCEDURE (en tant qu'expression se réduisant à un indicateur de fonction supposé sans paramètre)
BOOLEEN	Idem avec le type BOOLEEN
{ REEL / ou / ENTIER } TABLEAU	Tableau déclaré { REEL / ou / ENTIER } TABLEAU
BOOLEEN TABLEAU	Tableau déclaré BOOLEEN TABLEAU
PROCEDURE	Procédure déclarée PROCEDURE ou <TYPE> PROCEDURE
{ REEL / ou / ENTIER } PROCEDURE	Procédure déclarée { REEL / ou / ENTIER } PROCEDURE
BOOLEEN PROCEDURE	Idem avec le type BOOLEEN
ETIQUETTE	Etiquette, Expression de désignation
AIGUILLAGE	Aiguillage
CHAINE	Chaîne

Le traitement est envisagé de la façon suivante : on vérifie que le paramètre effectif est

1) une quantité déclarée de façon analogue à la spécification du formel

ou 2) une quantité déclarée ou une expression autorisée pour le formel. Cette distinction ne reflète que l'aspect technique de la vérification et de la détermination de l'indice de conversion effectuée simultanément : le sous programme interprèteur comporte 2 séquences différentes attachées au 1°) et au 2°). Nous l'indiquerons brièvement (cf. ch. 11). Lorsque le paramètre effectif est un identificateur, la partie qualitative de sa Cst est au cadrage près identique au modèle (partie MOD) d'un formel spécifié de la même manière qu'est déclarée la quantité représentée par l'identificateur. (Une étiquette est considérée déclarée, de même une chaîne). Par ailleurs en distinguant les familles de paramètres effectifs "variable indicée" et "nombre sans signe", on a ramené leur Cst prélevée à celle de la variable simple correspondante (conservation de d_{16} d_{15} seuls qui indiquent le type) Aussi s'il y a identité entre la partie qualitative de la Cst et le modèle (partie MOD) la correspondance est correcte et l'indice de conversion nul. Sinon on recherche la combinaison Cst-modèle dans une table (18 mots) d'associations correctes ; l'indice de conversion sera nul ou 1 suivant que l'on trouvera l'association en début ou fin de table. Cette méthode s'applique assez bien car la machine possède une instruction de consultation de table .

C. La partie ORFS (9.3.2.) du 1er mot de la Cdy est donnée maintenant par l'indice de conversion et par la famille du paramètre effectif : les numéros de famille 0,1,2,3,4 (cf. point A) déterminent respectivement NOM, NOMP, EXCA, EXCV, NOM (appellation symbolique de ORFS)

Le 1er mot de la Cdy est "marqué" si le paramètre doit être appelé par valeur, en prévision de la seconde phase de constitution de Cdy. Cette indication donnée par le modèle du formel est reportée en forçant à 1 le bit de signe (d_{18}) du 1er mot de la Cdy. (NB_K ne s'y étend pas et ce mode de repérage est commode).

La construction de la Cdy, en appel par nom, est alors résolue Remarque un paramètre effectif "nombre sans signe" (famille 4) est devenu un paramètre "variable simple" appelé par valeur, même si le formel ne figurait pas en partie valeur (on a "marqué" la Cdy).

A l'exécution de la procédure, ce paramètre est une "variable locale" : l'accès sera plus rapide, mais évitera surtout que le programme ne modifie par erreur une constante ; on traite ici les nombres sans signe comme des variables simples ce qui est commode ; mais si on ne forçait pas le mode d'appel, les conséquences pourrait être "l'affectation d'une valeur à une constante" et ne seraient pas détectées.

9.4.1.2. Le paramètre effectif est un formel X

Au formel X correspond un paramètre effectif dont on a naturellement construit la caractérisation dynamique Cdy1. La Cst de X utilisé comme paramètre effectif, nous indique l'adresse dynamique de Cdy1 ; Nous la transformons en adresse physique (utilisation de l'état actuel du display) pour l'examiner : Cdy 1 donne l'accès au paramètre effectif que nous allons véritablement employer et dont nous voulons construire la caractérisation dynamique Cdy. De ce fait Cdy sera pratiquement la recopie de Cdy 1.

Cependant si X spécifié <TYPE> était appelé par valeur, Cdy_1 contient la valeur correspondante à X, devenu équivalent à une variable locale : on construira dans ce cas Cdy comme caractérisation d'une variable

locale appelée par nom, l'adresse physique d'implantation étant celle de Cdy_1 (S^d mot de Cdy_1).

Subsiste le problème de vérification de la correspondance paramètre effectif-formel, et éventuellement de l'indice de conversion.

On a vu que l'on considérait que la spécification impose une signification précise pour toute utilisation du paramètre effectif : c'est-à dire que si un formel X est utilisé en paramètre effectif et correspond au formel Y dans un appel de procédure, nous ne considérerons pas le paramètre effectif remplaçant X, mais une quantité fictive dont la déclaration serait analogue (correspondrait) à la spécification de X , pour le problème de correspondance paramètre effectif-formel Y.

[on verra la différence avec le Rapport Algol en Remarque 2°)]

Le traitement est effectué comme en 9.4.1.1. par le même sous programme interprèteur : la partie qualitative du paramètre effectif est remplacée ici par la partie MOD (spécification de X) prise dans Cdy_1, où on l'a conservée à cet effet.
La vérification ainsi faite nous conduira certainement à une instruction Algol correcte, dans les mêmes conditions que précédemment.

Indice de Conversion : Cette question pour les paramètres effectifs à type arithmétique, à toujours le même but d'imposer à une valeur de paramètre effectif, le type spécifié pour le formel.

La correspondance paramètre effectif (formel X)-formelY, nous donne comme en 9.4.1.1., un premier indice de conversion relatif au type du paramètre effectif pris conformément à X, et au type qu'impose maintenant Y.

Dans la caractérisation Cdy_1 du paramètre effectif pour X, nous avions noté un autre indice de conversion relatif au type de ce paramètre et au type imposé par X.

L'addition logique de ces 2 indices, donne l'indice de conversion à considérer maintenant, pour imposer le type spécifié par Y, au paramètre effectif véritable. Si le paramètre effectif est une expression, ce que l'on reconnait sur Cdy_1, on conserve l'indice 1 pusiqu'il dénote non un transfert de type mais un test.

En résumé lorsqu'un paramètre effectif est un formel X, si Cdy_1 caractérisation relative au paramètre effectif pour X,

1°) n'est pas celle d'un paramètre spécifié <TYPE> et appelé par valeur : on recopie Cdy_1 pour construire Cdy ; le premier mot est ainsi modifié : la partie MOD correspond à la nouvelle spécification, l'indice de conversion est celui que l'on a déterminé, les autres informations restent naturellement identiques.

2°) est celle d'un paramètre spécifié <TYPE> et appelé par valeur : on se ramène à la construction d'une Cdy de variable simple en appel par nom (9.4.1.1.). L'adresse d'implantation est celle du S^d mot de Cdy et l'indice de conversion a été déterminé. (On notera à ce propos qu'il était forcément nul dans Cdy_1).

Puis comme précédemment on marquera la Cdy si le paramètre doit être appelé par valeur.

Remarques.

1) Considérer que la spécification impose un type au paramètre effectif, conduirait logiquement dans le cas présent, à convertir la valeur du paramètre effectif au type imposé par X, puis cette valeur au type imposé par Y.
La structure du traitement est prévu pour cela, mais il faudrait prendre en compte l'arrondi à une valeur entière sous format flottant ; le même problème se pose d'ailleurs dans les affectations ; il ne parait pas intéressant de s'arrêter à ce point de vue. [Il serait assez facile de mettre en place le traitement éventuel, dans les 2 cas ; pour le problème présent il faudrait considérer un indice de conversion à 3 valeurs (la valeur supplémentaire indiquant l'arrondi à limité d'une valeur de type réel), qui ne serait plus déterminé à partir des 2 indices relatifs à X et Y par une simple addition logique].

On traite le type du paramètre effectif, conformément à la spécification, sans s'intéresser à la manière dont on l'a communiqué.

2) Considérer une quantité fictive de déclaration correspondant à la spécification de X, ou le paramètre effectif remplaçant X, ne diffère que dans 2 catégories de cas qu'il ne nous parait pas intéressant de traiter.
Soient X et Y formels des déclarations de procédure P1 et P2 ; l'activation de P2 (X,...) est donnée dans le corps de procédure de P1. Nous rejettons :

PARAMETRE EFFECTIF		FORMEL CORRESPONDANT
PB déclarée BOOLEEN PROCEDURE	niveau P1 →	X spécifié PROCEDURE
X	← niveau P2 →	Y spécifié BOOLEEN PROCEDURE
PB déclarée BOOLEEN PROCEDURE	procédure sans paramètre →	X spécifié BOOLEEN
X	→	Y spécifié BOOLEEN PROCEDURE

9.4.2. Seconde phase de constitution des Cdy. : Appel par valeur

Lorsque les caractérisations dynamiques de tous les paramètres considérés en appel par nom ont été construites, on se place dans les conditions d'activation de la procédure [2.7. et 4.2.1.]. Les index NB, DP et de display sont mis à jour, alors qu'ils reflétaient jusqu'alors la situation pour le bloc appelant ; les 2 premières données de liaison (DP_{N-1} et ZT_N) sont mises en place : IA est mis à jour et sa valeur mémorisée comme seconde donnée de liaison (ZT_N : encombrement local), d'après l'indicateur de réservation de la procédure, auquel on ajoute 3 mémoires locales. En effet on va maintenant appeler par valeur, les paramètres effectifs qui doivent l'être ; ceci va nous amener à faire des calculs en dehors de la procédure et par suite de la récursivité, nous devons empiler certaines informations. Dans ces 3 mémoires locales supplémentaires au sommet du stack, nous avons donc:

a) l'adresse du 1er ordre objet du corps de procédure pour pouvoir entreprendre l'exécution après le traitement des appels par valeur

b) l'adresse de la caractérisation dynamique du dernier paramètre pour savoir arrêter l'examen.

c) l'adresse de la Cdy couramment examinée, lorsque le paramètre doit être appelé par valeur : lorsqu'on aura obtenu cette valeur il faut savoir où la ranger et reprendre l'examen.

En effet on procède de la façon suivante : les caractérisations dynamiques sont examinées dans l'ordre à partir de la première. On a vu que si le paramètre doit être appelé par valeur, le 1er mot de Cdy est "négatif" :

on supprime alors le digit de signe : son rôle de "marque" est achevé.
On exécute l'ordre formel "Prendre Valeur" correspondant à la spécification du paramètre, après avoir indiqué dans XCP, compteur programme, une adresse interne de l'interpréteur. A cette adresse nous avons un ordre très particulier du langage objet : Ø COD qui permet de continuer l'exécution non plus en langage objet (généré par le compilateur) mais en langage machine (plus exactement de logrammes voir Ch. 10). La raison en est, qu'un ordre formel provoque naturellement en s'achevant, l'exécution de l'ordre suivant du programme objet que pointe XCP. Il faut procéder ici de la même manière pour pouvoir ranger dans la Cdy la valeur que l'ordre formel nous a laissé au sommet du stack. Le sous programme machine lancé par Ø COD dispose de l'adresse Cdy, dans la mémoire qui précède immédiatement l'accumulateur résultat. Après modification du premier mot de Cdy et rangement (libérant l'accumulateur) on reprend l'examen de la Cdy du paramètre suivant.

Lorsque tous les paramètres sont examinés, les 3 mémoires supplémentaires au sommet du stack sont libérées : IA et la seconde donnée de liaison sont mis à jour. Par ailleurs la valeur de la 3ème mémoire supplémentaire a réinitialisé le compteur programme XCP : on reprend l'exécution du programme objet, le 1er ordre du corps de procédure.

9.4.2.1. Appel par valeur d'un formel spécifié <TYPE>

On initialise le compteur programme XCP à l'adresse de la séquence objet de l'interpréteur : Ø COD

* ZRAV

où ZRAV est le sous programme interpréteur de rangement.
On active ensuite suivant la spécification du formel (partie MOD de la Cdy), le sous programme relatif à l'ordre CRF ou CEF (9.5.1.2.). Ces ordres achevés,

laissent la valeur du formel au sommet du stack avec le type imposé.

Le sous programme ZRAV sera exécuté ; sans modifier autrement le 1er mot de Cdy, il met en place l'indication VAL en partie ORFS. Après test du type de la valeur il utilise les sous programmes, déjà cités dans l'affectation (simple), de rangement Réel ou Entier, il leur précise comme adresse physique, le 2^d mot de Cdy.

9.4.2.2. Tableau formel appelé par valeur.

Non autorisé, les sous programmes de l'interpréteur n'existent pas (voir 6.3.) .

9.4.2.3. Appel par valeur d'un formel spécifié ETIQUETTE

La valeur est une étiquette. La caractérisation dynamique d'un paramètre effectif étiquette doit indiquer la structure de blocs origine de l'étiquette ; on en a examiné les raisons, et l'exploitation, en détail au chapitre 7.

. Si la caractérisation dynamique indique déjà une étiquette (ORFS vaut NOMP), cette condition est déjà vérifiée. En effet ou :

1) le paramètre effectif était une étiquette, et la Cdy construite est l'adresse de la mémoire en table relative à l'étiquette et l'indication de la structure de blocs pour le bloc appelant : il donne bien les conditions dans lesquelles il faut se replacer pour exploiter l'étiquette.

2) Le paramètre effectif était un formel, et la Cdy du paramètre effectif correspondant à ce formel était déjà une étiquette. Comme on a simplement recopié cette Cdy, il n'y a pas de problème.

. Au contraire si la caractérisation dynamique indique une expression de désignation (ORFS vaut EXCV), il faudra s'assurer quand on l'aura évaluée, que l'on précise bien avec l'étiquette résultat sa structure de blocs origine. Ce problème est parallèle à l'évaluation d'un indicateur d'aiguillage par l'ordre Ø AIG (ch. 7), et d'une expression de désignation par l'ordre EAF (9.5.3.). Si l'on exécute le programme objet de l'expression de désignation, le résultat considéré dans le bloc où l'on fait les calculs (repositionné par la structure de blocs indiquée par Cdy) est : une étiquette non paramètre attachée à la structure de ce bloc, ou une étiquette paramètre attachée à une structure de blocs précisée par l'adresse d'une Cdy dans l'accumulateur résultat (ch. 7).

Au retour d'évaluation, il suffirait de ranger une étiquette non paramètre dans la Cdy, sans modifier la structure de blocs qui y est précisée correctement. Au contraire, il faut recopier dans la Cdy, la structure de blocs à laquelle est attachée une étiquette paramètre,

Par commodité on utilise l'ordre EAF qui donne l'étiquette paramètre effectif avec l'adresse de la Cdy caractérisant sa structure. Le traitement est uniformisé et conduit éventuellement à recopier sur elle-même la Cdy.

<u>En résumé pour effectuer l'appel par valeur d'un formel ETIQUETTE</u>

1) on ne fait rien si le paramètre effectif est une étiquette.
2) s'il est en appel par nom, une expression de désignation, on initialise le compteur programme XCP à l'adresse de la séquence objet de l'interpréteur

Ø COD . on active ensuite le sous programme spécifique
* RGETV de l'ordre EAF (9.5.3.)

Lorsque cet ordre est achevé le résultat étiquette-paramètre est dans l'accumulateur au sommet du stack. Le sous programme RGETV est exécuté et recopié dans la Cdy du paramètre, la Cdy précisée par l'accumulateur résultat. Puis l'étiquette résultat (adresse table) est placée dans le Cdy (second mot), et celle ci est modifiée (1er mot) pour que ORFS indique NOMP.

9.5. Ordres formels - Exécution.

9.5.1. Formel spécifié <TYPE>

Le paramètre effectif correspondant peut être une variable (simple ou indicée), une expression, une procédure (en tant qu'indicateur de fonction sans paramètre). La caractérisation Cdy que refère l'ordre formel indique cette situation (9.3.)

Si le formel apparait en partie gauche d'une instruction d'affectation, ou comme variable contrôlée, d'une instruction POUR, ce ne peut être qu'une variable, et les ordres "Prendre Adresse de Formel" le contrôlent. D'une manière indirecte, les ordres "Prendre Valeur de Formel" vérifient que le paramètre effectif fonction est bien légitime, en évaluant un indicateur de fonction sans paramètre : si la fonction n'est pas sans paramètre, son activation déclenche une erreur. Ce sont les seules vérifications que l'on a préféré ne pas faire à l'entrée.

D'une façon générale, l'ordre formel trouvera dans Cdy les indications NB_K et DP_K lui permettant de se replacer dans la structure de bloc origine du paramètre effectif, lorsque ce dernier donne lieu à l'actication d'une procédure, on d'un sous programme (évaluation d'une expression calcul d'adresse de variable indicée). Nous avons vu en 7.5. comment sont

activés les sous programmes de paramètres effectifs et nous n'y reviendrons pas. On rappelle cependant qu'on note une adresse de retour à un sous programme interpréteur ; il permet après obtention du résultat du sous programme paramètre effectif, de prendre la valeur à partir du résultat adresse de variable indicée par exemple, et ou de traiter les conversions de type.

Les 4 ordres formels ARF, AEF, CRF, CEF concernant les formels spécifiés <TYPE> s'exécutent selon le même schéma, qui provoque des décisions différentes suivant l'indication ORFS de la caractérisation dynamique. Chaque ordre se contente de préciser une table des informations qui lui sont particulières, au noyau commun d'exécution. Ce sont des adresses de sous programmes interpréteurs.

Nous résumerons l'exécution sous forme de Tableau.

Cdy \ ordre Formel			ARF	AEF	CRF	CEF	Action prise par le noyau commun d'exécution.
0	NOMP	0	ER	ER	ORFSA	ORFSA	Saut au sous programme interpreteur dont on a l'adresse
1		1	ER	ER	ORFSA	ORFSA	
2	EXCV	0	ER	ER		ZENT	Activation du sous programme paramètre effectif dont l'adresse d'entrée est indiquée dans Cdy (2^d mot). L'adresse de retour sous programme interpréteur est donnée par la table de l'ordre.
3		1	ER	ER	EXCVTR	EXCVTE	
4	EXCA	0	ZENT	ZENT	EVALR	EVALE	
5		1	EXAC	EXAC	EVALRC	EVALEC	
6	NOM	0	ZARN	ZAEN	ZCRS	ZCES	Saut au sous programme interprèteur indiqué par la table de l'ordre avec pour adresse de l'opérande 1) adresse notée dans Cdy (2^d mot) lorsque NOM 2) adresse d'implantation de Cdy (2^d mot) lorsque VAL.
7		1	NOMRC	NOMEC	VNOMRC	VNOMEC	
10	VAL	0	ZARN	ZAEN	ZCRS	ZCES	
octal symbol.d_1 (ORFS)			adresse de sous programme Interpreteur				
d_1 indice de conversion inclus dans ORFS							

Remarques

. ORFS a été appelé ordre formel secondaire parce qu'il détermine un sous programme interpréteur. Par ailleurs on aurait pu faire correspondre à ces sous programmes, un ordre du langage objet, par scission des ordres formels en 2 ordres à la compilation ; en activant un sous programme de paramètre effectif, on n'aurait plus besoin d'une adresse de retour-interpréteur.

. Les sous programmes interpréteurs ZARN , ZAEN, ZCRS, ZCES sont ceux des ordres objet ARN, AEN, CRS, CES (après traitement initial donnant l'adresse physique de l'opérande). ZENT correspond à l'entrée de l'interpréteur : lancement de l'exécution de l'ordre objet pointé par le compteur programme, après son décodage. ER est le sous programme de constat d'erreur.

9.5.1.1. Ordres sur les adresses de formel ARF , AEF.

ARF Adresse Dynamique, signifie stacker l'adresse d'un formel spécifié REEL ; de même AEF pour un formel spécifié ENTIER,

L'adresse stackée sera accompagnée de la marque de type du paramètre effectif. Eventuellement l'accumulateur contiendra aussi (3ème mot) un indice de conversion valant 1, si la variable paramètre effectif n'a pas le type spécifié pour le formel (paramètre arithmétique). [3.5.]

La caractérisation dynamique est NOM ou VAL : l'adresse d'implantation de la variable est respectivement contenue dans la Cdy (2^d mot), ou est celle de Cdy (du second mot). Après l'avoir prise, on est ramené au cas des ordres ARN et AEN relatifs aux variables (déclarées). Rappelons que pour ceux-ci, l'adresse dynamique est transformée en adresse physique, au décodage

de l'ordre. Il suffit donc, si l'on avait l'ordre formel ARF

a) d'exécuter le sous programme interpréteur relatif à ARN, lorsque la variable effective a le type spécifié, soit ORFS indiquant VAL ou NOM_0(=6)

b) d'exécuter dans le cas contraire le sous programme relatif à AEN, puis une séquence plaçant l'indice de conversion dans l'accumulateur. Ceci est traité par NOMRC lorsque ORFS indique NOM_1 (=7).

Le traitement de l'ordre AEF est semblable.

La caractérisation dynamique est EXCA ; systématiquement, le noyau commun d'exécution active (755,)le sous programme objet-paramètre effectif, quand la Cdy en indique un. Après retour de ce sous programme, nous avons ici, l'adresse de la variable effective dans l'accumulateur au sommet du stack.

On avait admis dans les caractérisations (9.2. Remarques), que le type d'un paramètre effectif variable indicée est celui du tableau. Si le tableau nommé était formel, le type véritable sera différent, lorsque tableau effectif n'a pas le type spécifié. Dans ce cas l'accumulateur-adresse contient l'indice de conversion. Par ailleurs, la marque adresse de l'accumulateur indique naturellement, toujours le type effectif.

Quand le type prévu de la variable indicée était identique au type spécifié pour le formel, soit ORFS ≡ $EXCA_0$ (=4), l'accumulateur-adresse résultat est donc correct. L'exécution des ordres ARF et AEF s'achève là ; l'adresse de retour à l'interpréteur, après activation du sous programme paramètre effectif est alors celle de l'entrée interpréteur ; passer à l'exécution de l'ordre objet suivant.

Dans le cas contraire, soit ORFS ≡ $EXCA_1$ (=5), le sous programme interpréteur EXAC remplacera l'indice de conversion, dans l'accumulateur étiquette, par son complément et achève les ordres ARF, AEF. En effet le type prévu ne correspondant pas à la spécification du formel, il faut s'il est réalisé, indiquer la non correspondance par l'indice de conversion 1, et dans le cas contraire puisqu'il y a effectivement correspondance supprimer cet indice de conversion 1.

La caractérisation dynamique est NOMP ou EXCV : le paramètre effectif est une expression, et on déclenchera un constat d'erreur : saut direct au sous programme d'erreur dans le cas de NOMP, le même saut comme retour à l'interpréteur après évaluation de l'expression dans le cas EXCV.

9.5.1.2. Ordres sur les valeurs de Formels CRF et CEF

CRF Adresse Dynamique, signifie charger dans l'accumulateur au sommet du stack le formel spécifié REEL ; de même CEF pour un formel spécifié ENTIER.

La valeur stackée, l'est avec le type imposé par la spécification.

La caractérisation dynamique est NOM ou VAL.

Le même principe a déjà été utilisé pour les ordres ARF et AEF. Ayant pris l'adresse d'implantation de la variable, on est ramené pour prendre sa valeur avec le type spécifié, à exécuter dans le cas de l'ordre CRF :

a) le sous programme interpréteur de l'ordre CRS, lorsque son type est celui de la spécification : ORFS indiquant VAL ou NOM_0 (=6) ;

b) le sous programme de l'ordre CES, suivi de la séquence transfert de type ENTIER en REEL. L'enchainement de ces sous programmes écrits par ailleurs, est organisé par VNOMRC lorsque ORFS indique NOM_1 (=7) : type effectif ne correspondant pas à la spécification.

L'ordre CEF est traité de la même manière : soit CES soit VNOMEC (CRS suivi du transfert Réel en Entier).

La caractérisation dynamique est EXCA

Au retour du sous programme paramètre effectif, nous avons en stack l'adresse de la variable indicée ; après l'avoir prélevée et libéré l'accumulateur on est ramené au traitement vu précédemment.

Les sous programmes interpréteurs effectués, font appel à la même séquence pour ce travail, et déterminer selon le type effectif de la variable l'orientation du traitement. Soit
pour CRF : CRS ou VNOMRC
" CEF : CES ou VNOMEC

Cette séquence peut tester directement le type effectif sur la marque adresse de l'accumulateur ; on ne tiendra pas compte dans ce cas des prévisions de type effectif (EVALR et EVALRC confondus, de même EVALE et EVALEC).

On gagnera très légèrement sur le temps d'exécution en testant si le type effectif prévu est respecté ou non ; les enchainements sont alors organisés par des sous programmes distincts selon que le type prévu était celui de la spécification ou non (EVAL et EVALRC pour les formels Réel, EVALE et EVALEC pour les formels Entier).

La caractérisation dynamique est EXCV

Le paramètre effectif est une expression, et au retour de l'activation du sous programme correspondant, nous avons sa valeur en stack.

Le seul cas où Cdy n'a pas d'indice de conversion, ORFS ≡ $EXCV_0$(=2) correspond à une expression booléenne, et elle ne peut être référée en outre que par l'ordre CEF. Il n'y a naturellement rien à faire sur la valeur en stack et l'ordre est terminé (l'adresse de retour à l'interpréteur après activation du sous programme paramètre effectif, est celle d'entrée de l'interpréteur).

$EXCV_1$ (=3) correspond à une expression arithmétique, l'indice de conversion signifie simplement que le type du résultat en stack n'a pas été déterminé. On exécute alors pour l'ordre CRF le sousprogramme EXCVTR, qui teste le type de la valeur et effectue, si nécessaire, le transfert en Réel. L'ordre CRF est alors achevé.

EXCVTE fait un traitement parallèle pour l'ordre CEF.

La caractérisation dynamique est NOMP.

Le paramètre effectif est une procédure-fonction, avec la signification d'un indicateur de fonction sans paramètre. (pour compiler ce paramètre effectif en tant qu'expression, il aurait fallu connaître son utilisation dans la procédure).

On va donc se ramener à l'activation d'une procédure-fonction formelle (9.5.4.) : la séquence ORFSA indique simplement le nombre 0 de paramètres effectifs, et régresse le compteur programme, pour se placer dans les conditions particulières d'exécution du sous programme relatif à l'ordre AFF, qu'elle rejoint.

9.5.2. Formel spécifié TABLEAU , <TYPE> TABLEAU

Il ne peut être référé que par 2 ordres ARF et AEF : un calcul d'adresse ou de valeur d'une variable indicée dans le programme objet.

Ces ordres et leur exécution, ont déjà été vu pour les formels spécifiés <TYPE>.

La nécessité de noter un indice de conversion dans l'accumulateur adresse, est alors évidente : l'ordre Ø PVA (Ch. 6) ne pourrait sans cela imposer la valeur de la variable indicée, le type spécifié. Le problème est analogue pour l'ordre Ø VAL qui prend la valeur d'une variable contrôlée d'après son adresse en stack.

Affectation de valeur à un formel.

Ceci correspond à une affectation, dans le programme source à un formel spécifié <TYPE> ou à une variable indicée d'un tableau formel.

La valeur de l'expression telle qu'elle vient d'être évaluée, est éventuellement convertie pour respecter le type de la variable paramètre effectif, puis affectée.

Cela signifie que le type spécifié par le formel n'est pas imposé auparavant à la valeur. On en tiendra compte dans les affectations simples et multiples.

Les ordres Ø RAN et Ø RAM ne considèrent que le type de l'adresse de la variable, et non de l'indice de conversion.

On pourrait en tenir compte, et si sa présence est détectée, effectuer un arrondi à la valeur entière sous format flottant lorsque le type effectif et le type de la valeur sont Réel. En fait compte tenu de la remarque 9.4.1.2. la position adoptée nous semble justifiée.

9.5.3. Formel spécifié ETIQUETTE , AIGUILLAGE : Ordre EAF

EAF refère les formels étiquette et aiguillage ; il place dans l'accumulateur au sommet du stack, l'adresse de la mémoire en table attachée au paramètre effectif ; il l'accompagne en outre de l'adresse physique de la Cdy qui précise la structure de bloc origine de ce paramètre

Son rôle et l'essentiel de son exécution ont été vus au chapitre 7.

La caractérisation dynamique est NOMP :

Il suffit de prélever l'adresse table du paramètre effectif dans la Cdy (2^d mot) et de l'empiler, ainsi que l'adresse physique de cette même Cdy.

C'est toujours le cas pour un aiguillage formel.

La caractérisation dynamique est EXC

Le paramètre effectif est une expression de désignation ; on active le sous programme objet correspondant, selon le processus habituel.

Au retour d'évaluation le résultat en stack est une étiquette paramètre ou non paramètre, comme nous l'avons évoqué en 9.4.2.3. Lorsqu'on évalue un indicateur d'aiguillage formel, la situation est identique : il faut reporter dans l'accumulateur résultat, s'il contient une étiquette non paramètre, l'adresse de la Cdy du paramètre que l'on vient d'évaluer : elle précise pour l' utilisation ultérieure dans la procédure, la structure de blocs origine du résultat (Ch. 7).

Dans le cas présent, on peut donc empiler l'adresse de la Cdy (l'index IA progresse d'un accumulateur), puis activer le sous programme paramètre effectif. On note l'adresse du sous programme interpréteur achevant l'exécution de Ø AIG, comme adresse retour-interpréteur de l'activation. Ici ce sous programme effectuera la fin de traitement de l'ordre EAF.

On a programmé une seconde solution, un peu différente, qui évite d'empiler l'adresse de Cdy dans un accumulateur, mais moins intéressante par ailleurs. Au retour d'activation, l'accumulateur résultat est en place pour son exploitation puisqu'il ne surmonte pas un accumulateur adresse Cdy. S'il contient une étiquette paramètre (3ème mot non nul) l'ordre EAF est immédiatement achevé. Dans le cas contraire on le marque (3ème mot) avant de terminer EAF, avec l'adresse de Cdy.

Pour obtenir cette dernière, on remarque que si l'ordre EAF est utilisé pour l'appel par valeur (9.4.2.3.) le compteur programme XCP contient l'adresse bien déterminée de la séquence Ø COD : l'adresse de Cdy est alors en dessous
* RGETV
de l'accumulateur.

Dans le cas contraire XCP indique l'adresse en séquence de l'ordre EAF lui-même, d'où la détermination de l'adresse physique de Cdy.

Remarque.

On utilise pour les étiquettes paramètres en stack, l'adresse Cdy, plutôt que les informations NB_K, DP_K qui nous intéressent directement : on peut ainsi employer le sous programme standard, qui repositionne dans la structure de blocs du paramètre effectif - (cette méthode est pratiquement plus efficace).

9.5.4. Formel spécifié <TYPE> PROCEDURE , PROCEDURE
Activation d'une procédure formelle : Ordres APF et AFF

Les ordres APF et AFF correspondent respectivement à une instruction procédure et à l'évaluation d'un indicateur de fonction, donnés avec une procédure formelle.

Le programme objet est identique à l'activation normale d'une procédure seul le second mot de l'ordre lui-même n'indique plus l'adresse table de la

procédure, mais l'adresse dynamique de la Cdy de la procédure paramètre (4.2.1.)

Exemple

AFF	N	N nombre de paramètres effectifs.
Adresse	Dynamique	

La Cdy référée donne l'adresse table de la procédure, et la structure de blocs origine de la procédure effective.

Lorsqu'on activait une procédure normalement, elle était déclarée dans un niveau précédant le bloc où on donnait l'ordre d'activation, et appartenant à sa structure de blocs : il suffisait simplement de régresser NB pour indiquer le niveau de déclaration, sans modifier le display ; puis, viennent les opérations usuelles d'entrée bloc. Au contraire, si on active une procédure formelle, il faudra se repositionner dans le bloc où est activée la procédure qui définit le paramètre effectif procédure, avant d'effectuer les traitements rappelés. Ce sont les conditions d'utilisation générales des paramètres.

Ceci excepté, ainsi que quelques ordres préalables, on utilise qu'un même sous programme d'activation de procédure.

Le traitement des procédures formelles est alors le suivant :

1) prélèvement de l'adresse-table de la procédure effective et mémorisation de l'adresse de la Cdy. On réserve un accumulateur pour les évaluations d'indicateur de fonction. Puis on stacke comme valeur de la mémoire indicatrice de résultat : 0 pour les instructions procédure (ordre APF), et 1 ou -1 pour l'ordre AFF suivant que le type de la procédure-fonction effective est celui de la spécification ou non (ORFS indiquant dans Cdy $NOMP_0$ ou $NOMP_1$). Cela permettra à l'ordre d'affectation de valeur à un identificateur de procédure, d'imposer le type spécifié (ch. 5).

2) on exécute la suite du traitement d'activation déjà décrit, et en particulier la constitution en appel par nom des Cdy (9.4.1.).

3) A la fin de cette phase, on repositionne les index NB, DP et le Display grace à l'adresse mémorisée (aucun problème de récursivité à ce sujet) de la Cdy de la procédure formelle.

4) On peut alors exécuter, la 2^{de} phase décrite en 9.4.2. : fin d'appel par nom, nous mettant dans les conditions d'exécution de la procédure, appels par valeur des paramètres, achèvement du mécanisme d'entrée procédure.

9.5.5. Remarque : Chaîne.

Dans le corps d'une procédure déclarée, un formel spécifié chaîne ne peut intervenir que comme paramètre effectif d'une activation de procédure.

En effet le paramètre effectif chaîne, ne peut être traité que par une procédure standard dont le corps est écrit en code machine (ch. 10). La Cdy donne alors l'adresse de la chaîne dans le programme objet.

CHAPITRE X

PROCEDURES STANDARD

Les procédures standard sont écrites comme des procédures déclarées dans le programme source, au niveau 0. (Le programme lui-même qu'il soit un bloc ou une instruction composée est compilé comme bloc de numéro 0).

Aussi les procédures standard s'utilisent elles de la même manière que les procédures déclarées. Dans le programme objet, les ordres d'activation ne les distinguent pas ; la table des procédures standard en tête de programme, n'est pas confondue avec la table des procédures déclarées pour de simples raisons de compilation ; elle contient naturellement la même information (3.2.5.) se réduisant ici à l'adresse d'entrée puisque le niveau de déclaration est 0 .

On peut écrire des procédures standard, des 'fonctions' standard entier, des fonctions standard réel, qui correspondent aux procédures déclarées respectivement PROCEDURE, ENTIER PROCEDURE, REEL PROCEDURE, et qui sont soumises aux mêmes règles d'emploi.

La seule particularité est que l'on puisse autoriser un nombre de paramètres variables, par l'écriture même de la procédure standard ; si cette situation n'est pas indiquée, le nombre de paramètres effectifs dans une activation doit être respecté.

Cette méthode de traitement des procédures standard présente l'intérêt de s'insérer très aisément dans le cadre du compilateur, et de leur conserver une souplesse suffisante.

10.1 Ecriture des procédures standard. ordre Ø COD

L'écriture est exactement celle du programme objet produit par le compilateur pour une procédure déclarée. Cependant on peut introduire dans le corps de procédure des passages en "langage machine" grace à l'ordre Ø COD.

Rappelons l'écriture d'une procédure objet :

1) une tête de procédure comprenant le nombre de paramètres, les modèles de ces paramètres (9.1.), l'indicateur de réservation tenant compte des 5 données de liaison, de 3 mémoires de Cdy par paramètre, des éventuelles variables locales.

2) le corps de procédure

3) l'ordre sortie bloc Ø SBL. Le corps de procédure est écrit dans le langage objet du compilateur que nous avons étudié. Nous appellerons ici ses élements : ordres algol.

Nous considérerons comme "langage machine" le langage dit Interprétatif RW : ses élements sont des noms de "logrammes". Ces derniers sont des sous programmes écrits en véritable langage machine, de telle manière qu'un index, dit compteur d'instruction , permette d'enchaîner l'exécution des logrammes, grâce à 1 ou 2 ordres de contrôle du déroulement du programme. Le programme est constitué d'une suite de noms de logrammes, chaque nom étant suivi des opérandes éventuels. Les noms de logramme s'utilisent comme des codes d'opération d'une machine. Après assemblage du programme et de l'ensemble des logrammes, chaque nom de logramme est remplacé par l'adresse d'entrée de ce dernier. Nous expliquerons approximativement le principe de déroulement du programme. Supposons le processus initialisé : le compteur d'instruction IC pointe dans le programme l'adresse du logramme à exécuter. La séquence

de contrôle lance alors l'exécution de ce logramme ; celui-ci réfère dans le programme ses opérandes référées par IC, met à jour IC, en même temps qu'il s'exécute. Lorsqu'il s'achève, il saute à la séquence de contrôle et nous sommes replacés dansles mêmes conditions : IC pointe le logramme suivant à exécuter. En fait la souplesse du langage machine rend ces enchaînements naturels et simples ; on utilise 2 séquences de contrôle comportant seulement 1 et 2 ordres, selon les logrammes. Ces derniers peuvent être considérés comme des microprogrammes.

Dans le corps de procédure l'ordre Algol Ø COD permet de passer au langage RW ; suivra donc une séquence programmée avec des logrammes. Elle se termine par le logramme spécial * ALGOL qui assure le retour en langage objet Algol.

Exemple page X - 4

On dispose en outre de plusieurs logrammes qui permettent de condenser certaines séquences, se répétant dans de nombreuses procédures standard.

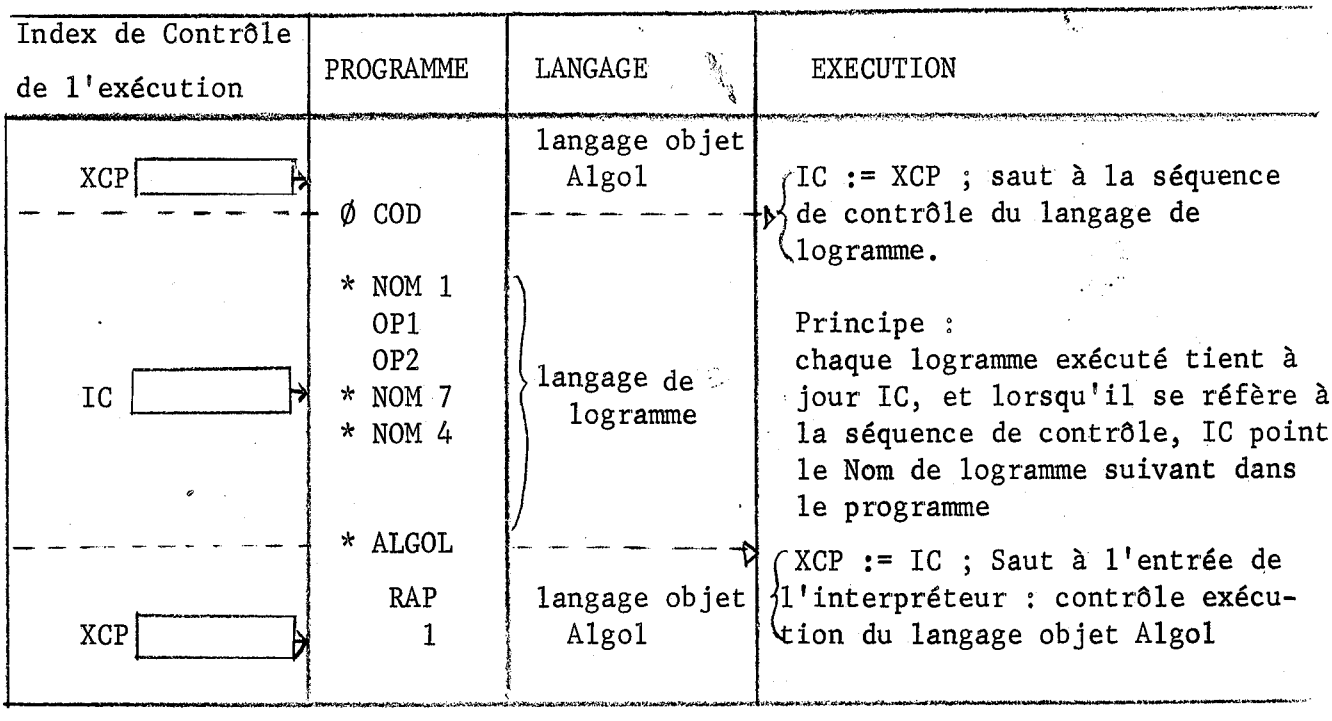
Exemple

logramme * RANSOR équivaut à (affectation valeur à identificateur de procédure et fin d'activation)

```
* ALGOL
Ø RAP
  1
Ø SBL
```

Au moment de l'exécution, on a évidemment en mémoire, les logrammes cités dans la procédure ; ils appartenaient déjà à l'interpréteur, ou ont été chargés en même temps que les procédures standard.

Remarque Indiquons sommairement l'exécution de Ø COD et * ALGOL, et résumons le passage "langage objet Algol - "langage machine".



10.1.1. Liste de paramètres variable

Exemple : procédure de lecture, impression.

L'écriture décrite précédemment, implique l'équivalent d'une procédure déclarée avec la liste de formels (X,Y,...Z,S). Dans un appel avec la liste effective (A,B,.....G,L), les paramètres effectifs doivent naturellement correspondre aux spécifications des formels (précisées par les modèles), et en respecter le nombre. La correspondance établie est A → X, B → Y,....,G → Z , L → S.

Mais en faisant précéder l'écriture de la procédure standard du mot 777777, sans modifier par ailleurs cette écriture, on autorise l'appel de cette procédure standard avec la liste effective (A,B,....,G, LISTE).

LISTE est une liste non vide L1, L2,...LN de paramètres effectifs correspondant à la spécification du même et dernier formel de la "déclaration".

La correspondance est établie, et doit être respectée, de la façon suivante :

1°) A → X, B → Y,....G → Z. comme précédemment ;

2°) puis le formel S étant considéré comme les formels S1, S2, ...SN appelés et spécifiés de façon identique.
L1 → S1, L2 → S2,...,LN → SN

Particularités des procédures standard à liste variable.

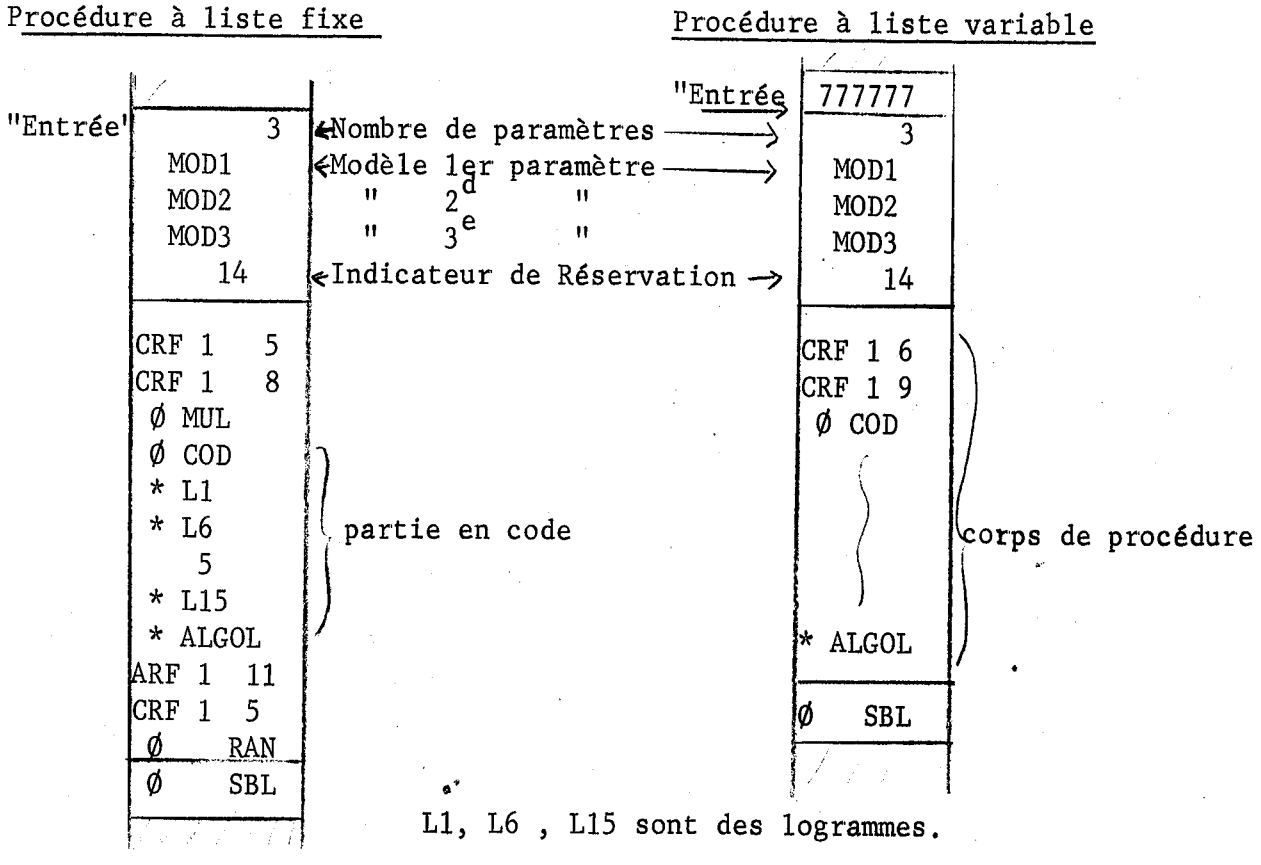
Une procédure standard à liste fixe, peut comporter des variables locales ; les caractérisations dynamiques des paramètres effectifs sont implantées aux adresses relatives 5,8,11,...exactement comme dans une procédure déclarée.

Au contraire dans une procédure standard à liste variable, il n'y aura pas de variables locales : on ne peut leur attribuer aucune adresse relative déterminée, à la suite des Cdy qui seront en nombre inconnu. Par ailleurs les Cdy sont implantées lors de l'exécution aux adresses relatives 6,9,12,....; la mémoire locale 5 contient alors la valeur N-1, où N est le nombre de paramètres effectifs de LISTE : L1, L2, ... LN.

Ceci permettra d'opérer plus facilement sur la partie variable de la liste.

Le corps de procédure doit tenir compte de ces particularités d'implantation à l'exécution.

Exemples d'écriture de Procédure standard



L1, L6 , L15 sont des logrammes.

Remarque pour les procédures à liste variable la valeur de l'indicateur de réservation n'est pas significative.

10.2. Activation des Procédures standard.

Le mécanisme d'entrée procédure décrit au chapitre 9, teste systématiquement la valeur du 1er mot de la tête de procédure ; en effet, on donne les mêmes ordres d'activation pour les procédures déclarées et standard, comme on l'a vu, dans le programme objet. Si cette valeur n'est pas 777777, on a donc une procédure déclarée ou une procédure standard à liste fixe : on effectue le traitement du chapitre 9.

Dans le cas contraire on a une procédure standard à liste variable, la mémoire locale 5 de l'entrée procédure est chargée avec le nombre de paramètres effectifs supplémentaires de l'appel (10.1.1.). Le sous programme de constitution des Cdy en appel par nom (9.4.1.) est exécuté, en considérant le modèle du dernier formel de la tête de procédure, lorsqu'on traite les paramètres effectifs supplémentaires. Comme la programmation impose de rétablir certains aiguillages à la fin de ce traitement, on tient compte à cette occasion que le niveau de "déclaration" de la procédure standard est 0 : ceci évite des remises à jour éventuelles, si elle est activée en tant que paramètre.

Puis on exécute normalement la suite du mécanisme d'entrée : appel par valeur des paramètres.(9.4.2.)

10.3.Restrictions liées au chargement des Procédures standard.

Les procédures standard sont chargées en fin de la génération (ch. 13). Seules les procédures standard citées dans le programme source compilé sont implantées, ainsi que les logrammes qu'elles utilisent.

Ces implantations variables selon le texte Algol source, sont effectuées par un programme de chargement travaillant à 2 niveaux (procédure puis logrammes) sur un ruban bibliothèque écrit en binaire. La structure de ce binaire est assez spéciale pour qu'il soit translatable de façon simple.

Il en résulte certaines restrictions.

Ainsi, dans les parties écrites en code, un opérande de logramme ne peut être les constantes représentées en octal par 400000 à 400077. (On peut toujours les utiliser par l'intermédiaire d'une mémoire qui les contient ailleurs). En outre on ne peut référer des adresses, que relativement à la position courante, ou absolues dans l'interpréteur.

C H A P I T R E XI

PROGRAMME INTERPRETEUR

Nous avons voulu dans ce chapitre exposer les principes, mais aussi expliciter très complètement le programme interpréteur en l'écrivant en Algol.

Sans atténuer la valeur et la portée générale de la description qui doit permettre d'adapter facilement cette méthode sur des calculateurs particuliers, nous avons suivi d'assez près l'organisation et la programmation de l'Interpréteur effectif, écrit directement en langage machine (CAE 510) ; nous espérons ainsi, le rendre aisément compréhensible aux utilisateurs qui l'examineraient.

Le programme interpréteur est un bloc de l'ensemble compilateur, où sont définis le programme objet désigné comme le tableau P, et le stack constitué par la mémoire machine et désigné comme tableau S. Les variables globales sont simplement indiquées par leur signification dans les initialisations.

Si la description est complète, les sous programmes relatifs aux opérations, aux relations et certaines procédures ne sont pas explicités. Cela n'ajouterait rien à la compréhension du programme, que nous nous sommes efforcés de rendre clair par de nombreux commentaires. En outre nous avons inséré des indications supplémentaires dans des pages intitulées "hors programme".

Nous avons du prendre quelques libertés avec le langage Algol dans l'écriture de l'interpréteur :

. L'opérateur Λ est défini pour 2 opérandes entiers (opération est l'intersection bit par bit), de même V.
De plus les constantes entières utilisées comme masques sont indiquées en octal (valeur des 6 chiffres octaux constituant la totalité du mot) précédé de la lettre H, qui les distingue des nombres entiers usuels. Exemple : H007700 signifie 007700 octal

. Des étiquettes sont parfois considérées comme des entiers (adresses absolues).

. Quelques procédures spécialisées sont placées à proximité des parties de programmes qui les utilisent.

Hors programme

Variables contrôlant l'exécution du programme objet.:

XCP Compteur programme;pointe l'ordre à exécuter. L'adresse qu'il contient est systématiquement progressée de 1 après le décodage de l'ordre objet.

IA Index d'Accumulateur;pointe l'accumulateur (1er mot) libre au sommet du stack.

NB Numéro de Bloc du niveau exécuté

DP Index Debut Paramètres : origine d'implantation des mémoires locales du Niveau exécuté.

DISPLAY Tableau des Valeurs prises par DP tel que décrit en 2.2.2.3.

Variables particulières :

XNL Compteur de ligne de programme objet

XOP Partie adresse d'un ordre ; adresse d'une opérande...

XORD Partie code d'un ordre

IAMAX Valeur maximum que ne doit pas dépasser l'index IA

IENTREE Indice contrôlant la fin d'exécution des sous programmes ARN AEN, CRS, CES. Associé à l'aiguillage AIGENTREE, sa valeur normale 1 provoque le bouclage immédiat surl'exécution de l'ordre objet suivant.

Mémoires de Travail.

RMA,...HCVTA,...FSTANDARD

Tableaux particuliers utilisés dans les ordres d'activation de procédure (leurs éléments seront indiqués à ce moment) : TBORF, TBMOCS.

Marques d'Accumulateur. (1er mot d'un accumulateur)

HADR	accumulateur contenant une	Adresse de Réel
HADE	" "	Adresse d'Entier
HVE	" "	Valeur d'Entier
HETI	" "	d'Etiquette ou Aiguillage

Aiguillage contrôlant l'exécution des ordres sur formel ARF, AEF, CRF, CEF :

TBFORMEL

Aiguillages répartissant l'exécution des ordres objet :

Ordres sans Adresse : AIGORD , AIGORDR
Ordres avec Adresse : AIGORDA

```
DEBUT COMMENTAIRE PROGRAMME INTEPRETEUR ;

ENTIER XCP,IA,NB,DP ; ENTIER TABLEAU DISPLAY [1:16] ;

ENTIER XNL,XOP,XORD,IENTREE,IAMAX ;

ENTIER RMA,RMD,RMP,A5,A7,T1,T2,T3,T6,T7,T9,T10,T11,T12,T13,WIAC,HND,HAD,HADF,
HA,WIA,HT,K,I,NBE,CSAD,HMOD,NMOD,HVAL,NCS,PA1,HCVT,HCVT1 ;
BOOLEEN FORMEL, FSTANDARD ;
```

```
ENTIER TABLEAU TBORF [0:4], TBMOCS [1:19] ;
ENTIER HADR,HADE,HVE,HETI ;

AIGUILLAGE TBFORMEL := ER, ER,ER,ER,ZENT,EXAC,ARN,NOMRC,ARN,ER,ER,ER,ER,
ZENT,EXAC,AEN,NOMEC,AEN,ORFSA,ORFSA,ZENT,EXCVTR,EVALR,EVALRC,CRS,VNOMRC,CRS,
ORFSA,ORFSA,ZENT,EXCVTE,EVALE,EVALEC,CES,VNOMEC,CES ;

AIGUILLAGE AIGORDA := CRS,CES,CRF,CEF,ARN,AEN,ARF,AEF,EAF,ADR,SBR,MLR,DVR,EXR,
ADE,SBE,MLE,DVE,EXE,TFA,CRS,CES,SAU,STA,TST,NL,APN,AFN,APF,AFF,SSP ;

AIGUILLAGE AIGORD := ADDE,SUBE,ISI,MULE,DIVE,DIVENT,EXPE,INFE,INGE,EGE,
SUGE,SUPE,DIFE,NON,ET,OU,IMP,IDE,RAN,RAM,RTN,RTP,EBL,SBL,PVA,IND,AIG,ALL,
TAB,RAP,RAQ,STP,FAI,COD,DO,VAL,RSP ;

AIGUILLAGE AIGORDR := ADDR,SUBR,ISI,MULR,DIVR,ER7,EXPR,INFR,INGR,EGR,SUGR,
SUPR,DIFR ;

PROCEDURE ENTREE ; ALLERA ZENT ; COMMENTAIRE Cette procédure achève l'exécu-
tion du programme interpréteur d'un ordre objet, en assurant le retour
au noyau de décodage de l'ordre suivant du programme objet ;

AIGUILLAGE AIGENTREE := ZENT,INVTYP,MVRES1,INVTYPER,INVTYPRE,EXAC1 ;

ENTIER PROCEDURE ADPHY(ADDYN) ; ENTIER ADDYN ;
        COMMENTAIRE Cette procédure donne l'adresse physique d'implantation
        correspondant à l'adresse dynamique ADDYN ;
        ADPHY := DISPLAY [(ADDYN ∧ H017000) x 2 ↑ (-9)] + (ADDYN ∧ H000777) ;
```

```
PROCEDURE TABLE (AMTABLE,ADRESSE,NBD) ; ENTIER AMTABLE,ADRESSE,NBD ;
    DEBUT COMMENTAIRE Cette procédure utilisée pour les procédures, étiquettes
    et aiguillages place en ADRESSE et NBD, respectivement l'adresse programme
    objet et le numéro de bloc de niveau de nomenclature de la quantité, dont
    l'adresse de mémoire en table est donnée par AMTABLE ;
       ADRESSE := P [AMTABLE]∧H037777 ;
       NBD     := (P [AMTABLE]∧H740000) x 2↑(-14) ;
    FIN ;

PROCEDURE REMAJ ;
    DEBUT COMMENTAIRE Remise à jour du Display lorsque les index
    NB et DP le sont ;
    RMD := DP ;
    POUR RMP := NB PAS -1 JUSQUA 1 FAIRE
    DEBUT DISPLAY [RMP] := RMD ; RMD := S [RMD] FIN
    FIN

PROCEDURE ERREUR (N) ; ENTIER N ;
    DEBUT COMMENTAIRE Cette procédure provoque le constat d'erreur (où N est
    le numéro d'erreur) sur l'imprimante : impression de N et de la valeur
    du compteur de ligne XNL. Puis elle arrête le calculateur ;
    ...
    FIN ;

PROCEDURE RANGE (INDEX) ; ENTIER INDEX ;
  COMMENTAIRE Cette procédure utilisée dans les ordres d'affectations, range
  dans la mémoire locale dont l'adresse est donnée par XOP, la valeur de l'entier
  contenu dans l'accumulateur pointé en stack de travail par INDEX ;
      S[XOP] := S[INDEX+1] ;
```

PROCEDURE RANGR (INDEX) ; ENTIER INDEX ;
DEBUT COMMENTAIRE Procédure analogue à RANGE, mais pour une valeur de type Réel. Son format flottant 3 mots en stack, est ramené au format flottant 2 mots (en particulier recadrage de l'exposant sur 128) des valeurs réel en mémoire locale ;
...
FIN ;

PROCEDURE CER (INDEX) ; ENTIER INDEX ;
DEBUT COMMENTAIRE Cette procédure effectue le transfert de type d'un accumulateur contenant une valeur ENTIER et pointé dans le stack de travail par INDEX, en valeur de type REEL ;
...
FIN ;

PROCEDURE CRE (INDEX) ; ENTIER INDEX ;
DEBUT COMMENTAIRE Analogue a CER. Transfert de type d'un accumulateur de valeur REEL en ENTIER (en particulier mise en place de la marque valeur ENTIER) ;
...
FIN ;

PROCEDURE CVENT ;
DEBUT COMMENTAIRE Utilisée dans les ordres d'opérations avec adresse, cette procédure charge l'accumulateur au sommet du stack de travail, avec la valeur de la variable locale de type ENTIER, dont l'adresse est donnée par XOP (travail identique à celui de l'ordre CES) ;
...
FIN ;

```
PROCEDURE CVRL ;
DEBUT COMMENTAIRE Analogue à CVENT, mais pour une valeur de variable locale
de type REEL ;
...
FIN ;

PROCEDURE COPIE (IR,IZ,N) ; ENTIER IR,IZ,N ;
DEBUT COMMENTAIRE Recopie des N mots d'une zone dont le début est referé
par l'index IZ, dans la zone indiquée par IR ;
 POUR I := 0 PAS 1 JUSQUA N-1 FAIRE S[IR+I] := S[IZ+I] ;
FIN ;

BOOLEEN PROCEDURE IMPAIR (A) ; ENTIER A ;
IMPAIR := SI (A∧H000001)=1 ALORS VRAI SINON FAUX ;
COMMENTAIRE (Correspond à une instruction machine sur le registre A du
calculateur) ;

PROCEDURE COMPEF ;
DEBUT COMMENTAIRE utilisée dans les ordres d'activation de procédure,
      voir page 33 ;
...
FIN ;

PROCEDURE ORFSC (RETOUR,SOUS PROGRAMME)
DEBUT COMMENTAIRE Activation d'un sous programme objet de calcul de paramètre
effectif - voir page 37 ;
...
FIN ;
```

```
PROCEDURE PFMAJ (ADCDY) ;
DEBUT COMMENTAIRE Remise à jour des index NB,DP et du Display avant utilisation
d'un paramètre effectif - voir page38 ;
...
FIN ;

COMMENTAIRE LANCEMENT DU PROGRAMME INTERPRETEUR ;
   XCP := ADRESSE LANCEMENT PROGRAMME OBJET ;
   IA  := ADRESSE FIN PROGRAMME OBJET ;
  IAMAX:= ADRESSE DE L EMPLACEMENT OCCUPE PAR LES PROCEDURES STANDARD ET L'INTER-
          PRETEUR EN HAUT DE MEMOIRE ;
   NB  := -1 ;
  IENTREE :=1 ;
  HADE := H777000 ; HADR := H777001 ; HETI := 770000 ; HVE := H777700 ;

ZENT : ; COMMENTAIRE NOYAU DE DECODAGE DES ORDRES OBJETS ET LANCEMENT DE LEUR
                    EXECUTION ;
        XORD := (P[XCP]Λ H760000)x2↑(-13) ; XOP := P[XCP]Λ HO17777 ;
        XCP  := XCP + 1 ;

SI XORD ≠ 0 ALORS DEBUT SI XORD ≤ 19 ALORS XOP := ADPHY (XOP) ;
                    ALLERA AIGORDA [XORD] ;
                FIN ORDRES AVEC ADRESSE - Calcul de l'adresse physique
                    pour les premiers ordres ;
```

```
SI XOP ≠ 3∧XOP ≤ 13 ALORS
   CVT : DEBUT COMMENTAIRE Test du type des 2 opérandes arithmétiques en
         accumulateurs et conversion éventuelle de l'un deux ;
         SI S[IA-3] = HVE ALORS
            CVTE : DEBUT SI S[IA-6] = HVE ALORS ALLERA ORDS ;
                         SI XOP = 7 ALORS ALLERA EXPRE SINON CER (IA-3) ;
                   FIN Premier opérande était entier ;
         SINON
            CVTR : SI S[IA-6] = HVE ALORS CER (IA-6) ;
         ALLERA AIGORDR [XOP] ; COMMENTAIRE ORDRES SANS ADRESSE - Opérations
         sur 2 opérandes arithémtiques Reel ;
        FIN de CVT ;

ORDS : ALLERA AIGORD [XOP] ;
       COMMENTAIRE ORDRES SANS ADRESSE - Opérations sur 2 opérandes arithmé-
       tiques Entier et autres ordres ;
```

Hors programme

AIGUILLAGE DES ORDRES VERS LES SOUS PROGRAMMES CORRESPONDANT APRES DECODAGE ET OPERATIONS PRELEMINAIRES.

Les étiquettes du programme interprèteur sont les noms symboliques des ordres objets. Lorsque ce n'est pas le cas on a indiqué ce dernier entre parenthèses.

ORDRES OBJET AVEC ADRESSE

Aiguillage AIGORDA

Etiquette (nom de l'ordre)	Voir page	Numéro de l'ordre
CRS	15	1 (CRN)
CES	15	2 (CEN)
CRF	35	3
CEF	35	4
ARN	15	5
AEN	15	6
ARF	35	7
AEF	35	8
EAF	36	9
ADR	12	10
SBR	12	11
MLR	12	12
DVR	12	13
EXR	12	14
ADE	12	15
SBE	12	16
MLE	12	17
DVE	12	18
EXE	12	19 (*)
TFA	23	20
CRS	15	21
CES	16	22
SAU	15	23
STA	21	24
TST	15	25
NL	15	26
APN	27	27
AFN	27	28
APF	27	29
AFF	27	30
SSP	22	31

* voir aussi EXPRE page 12

ORDRES OBJET SANS ADRESSE

Aiguillage AIGORD		ORDRE		Aiguillage AIGORDR	
Etiquette (nom de l'ordre)	voir page	Numéro	Nom si différent	Etiquette	voir page
ADDE	12	1	(ADD)	ADDR	12
SUBE	12	2	(SUB)	SUBR	12
ISI	12	3	(ISI)	ISI	12
MULE	12	4	(MUL)	MULR	12
DIVE	12	5	(DIV)	DIVR	12
DIVENT	12	6	(DEV)	ER7	12
EXPE	12	7	* (EXP)	EXPR	12
INFE	12	8	(INF)	INFR	12
INGE	12	9	(ING)	INGR	12
EGE	12	10	(EGA)	EGR	12
SUGE	12	11	(SUG)	SUGR	12
SUPE	12	12	(SUP)	SUPR	12
DIFE	12	13	(DIF)	DIFR	12
NON	13	14			
ET	13	15			
OU	13	16			
IMP	13	17			
IDE	13	18			
RAN	16	19			
RAM	16	20			
RTN	37	21			
RTP	22	22			
EBL	15	23			
SBL	15	24			
PVA	20	25			
IND	20	26			
AIG	22	27			
ALL	21	28			
TAB	12	29			
RAP	16	30			
RAQ	16	31			
STP	14	32			
FAI	23	33			
COD	14	34			
DO	22	35			
VAL	23	36			
RSP	22	37			

COMMENTAIRE Nous énumérerons simplement les noms des sous programmes spécifiques des opérations et relations. Les 2 opérandes sont alors de même type (traitement préliminaire). L'accumulateur au sommet du stack (1er opérande) est libéré, et le résultat placé dans l'accumulateur précédent (ex-2^{d} opérande). On revient alors à ZENT.

OPERATEUR ALGOL	SOUS PROGRAMMES CORRESPONDANTS les 2 opérandes (arithmétiques) sont	
	ENTIER	REEL
+	ADDE	ADDR
-	SUBE	SUBR
x	MULE	MULR
/	DIVE (voir	DIVR
÷	DIVENT plus bas)	ER7
↑	EXPE	EXPR
<	INFE	INFR
≤	INGE	INGR
=	EGE	EGR
≥	SUGE	SUGR
>	SUPE	SUPR
≠	DIFE	DIFR

Cas particulier : le sous programme EXPRE, vers lequel on s'oriente directement à partir du décodage des ordres et traitement préliminaire, correspond à r ↑i (où r est REEL et i ENTIER).

Le résultat des opérations arithmétiques est du même type que les opérandes sauf pour EXPE, EXPR, EXPRE (voir règles du Revised Report ALGOL 60 3.3.4.3.)

Le résultat des relations est une valeur logique, VRAI et FAUX étant respectivement représentés par les entiers 0 et 1.

Pour les opérations logiques, les sous programmes travaillent de façon analogue.

OPERATEUR ALGOL	SOUS PROGRAMME	
¬	NON	(unitaire)
Λ	ET	
V	OU	
⊃	IMP	
≡	IDE	

;

```
DIVE : CER (IA-3) ; CER (IA-6) ; ALLERA DIVR ;
ER7  : ERREUR (7) ;
ISI : SI S[IA-3] = HVE ALORS S[IA-2] := - S[IA-2]
                       SINON (S[IA-3], S[IA-2]) := -(S[IA-3], S[IA-2]) ;
      COMMENTAIRE opération - unitaire ;
ADE : CVENT ; COMMENTAIRE Chargement dans le 1er accumulateur libre de la
      valeur de la variable simple entière dont l'adresse est dans XOP ;
      XOP := 1 ;
      ALLERA CVTE ;

ADR ; CVRL ; COMMENTAIRE Même rôle que CVENT mais pour une variable simple
             réelle ; XOP := 1 ;
             ALLERA CVTR ;
```

COMMENTAIRE Les ordres d'opérations arithmétiques avec adresse suivant s'exécutent de la même manière que ADE et ADR. Ce sont pour les

Réels	entiers	opérateur Algol correspondant
SBR	SBE	-
MLR	MLE	x
DVR	DVE	/
EXR	EXE	↑

;

STP : ; COMMENTAIRE Dernier ordre du programme objet, provoque l'impression de PROGRAMME TERMINE sur l'imprimante et l'arrêt du calculateur ; DEBUT..... FIN ;

COD : ; COMMENTAIRE Cet ordre objet permet de quitter le mode l'exécution des ordres objets par le programme interpréteur. On poursuivra en séquence, l'exécution en langage machine. Le dernier ordre de cette séquence, relancera le mode d'exécution du programme interpreteur proprement dit (cf. 10.1.) ;

```
EBL : S[IA] := 0 ; IA := IA+1 ;
      S[IA] := DP ; S[IA+2] := 0 ; S[IA+3] := DP ; S[IA+4] := NB ;
      NB := NB+1 ; DISPLAY [NB] := DP := IA ;
      S[DP+1] := IA := IA+P[XCP] ; XCP := XCP + 1 ;
      ENTREE ;

SBL : IA := DP-1 ; DP := S[IA+4] ; NB := S[IA+5] ;
      SI S[IA+3] ≠ 0 ALORS XCP := S[IA+3] ;
      REMAJ ; ENTREE ;
```

```
        COMMENTAIRE Variante(non programmée) en séparant le traitement de la
        sortie de niveau des blocs et des procédures ;
EBL : S[IA] := DP ; S[IA+2] := 0 ;
      NB := NB+1 ; DISPLAY [NB] := DP := IA ;
      S[DP+1]* := IA := IA + P[XCP] ; XCP := XCP+1 ;
      ENTREE ;

SBL : IA := DP ;
      SI S[IA+2] = 0 ALORS
      DEBUT DP := S[IA] ; NB := NB-1 ; ALLERA ZENT FIN bloc source ;
      DP := S[IA+3] ; NB := S[IA+4] ; XCP := S[IA+2] ; IA := IA - 1 ;
      REMAJ ; ENTREE ; COMMENTAIRE Sortie Niveau Procédure ;
      COMMENTAIRE Si on a généré 2 ordres différents de Sortie Bloc et
      Sortie procédure il n'est plus besoin que 2 données de liaison dans
      l'entrée bloc, et le test sur la 3ème liaison qui distingue les 2
      sorties disparait ;

TST : IA := IA-3 ; SI S[IA+1] = 0 ALORS ALLERA ZENT ;
SAU : XCP := XOP ; ENTREE ;

NL : XNL := XOP ; COMMENTAIRE Remise à jour du Compteur de ligne ;
     ENTREE ;
ARN : S[IA] := HADR ; ALLERA ZAE1 ;
AEN : S[IA] := HADE ;
      ZAE1 : S[IA+1] := XOP ; S[IA+2] := 0 ; ALLERA TIAMAX ;

CRS : S[IA] := S[XOP] ; S[IA+1] := S[XOP+1]∧H777400 ;
      S[IA+2] := (S[XOP+1]∧H000377) - 128 ; ALLERA TIAMAX ;
      COMMENTAIRE L'exposant qui accompagne les faibles poids de la mantisse
      dans le 2^d mot en mémoire locale, est séparé et cadré sur 0 pour former
      le 3ème mot de l'accumulateur ;
```

```
CES : S[IA] := HVE ; S[IA+1] := S[XOP] ; S[IA+2] := 0 ;
      TIAMAX : IA := IA+3 ; SI IA > IAMAX ALORS ERREUR (5) ;
               ALLERA AIGENTREE (IENTREE) ;
      COMMENTAIRE Le retour se fait sur ZENT, sauf si l'exécution des 4 ordres
      précédents a été lancée indirectement par l'interpréteur : provoquée
      par un ordre d'indexation, un ordre sur formel. On s'orientera alors
      vers l'inversion de type ou la mise en place de l'indice de conversion.
          Pour les 2 séquences CRS et CES, utilisées par les ordres CRN, CEN,
      CES, CRS on a pris la liberté d'écrire que le programme P était inclus
      dans S ;

RAM : T5 := 0 ; ALLERA ZRAND1 ;
RAN : T5 := 1 ;
    ZRAND1 : IA := IA-6 ; XOP := S[IA+1] ; SI S[IA] = HADR ALORS ALLERA RANR ;
    RANE : SI S[IA+3] ≠ HVE ALORS CRE (IA+3) ;
           RANGE (IA+3) ; COMMENTAIRE Rangement d'un ENTIER ;
      RANS : SI T5 = 0 ALORS DEBUT COPIE (IA, IA+3,3) ; IA := IA+3
                                 FIN Affectation multiple ;
           ENTREE ;
    RANR : SI S[IA+3] = HVE ALORS CER (IA+3) ; RANGR (IA+3) ; ALLERA RANS ;
           COMMENTAIRE Rangement d'un REEL ;

RAQ : T5 := 0 ; ALLERA ZRAP1 ;
RAP : T5 := 1 ; ZRAP1 : XOP := P[XCP] ; XCP := XCP+1 ;
      SI XOP>0 ALORS DEBUT SI S[IA-3] = HVE ALORS CER (IA-3) FIN Fonction Réel
               SINON SI S[IA-3] ≠ HVE ALORS CRE (IA-3) ;
```

```
MVRES : XOP := DISPLAY [XOP∧H000037] -1 ;
      COMMENTAIRE XOP est l'adresse de la mémoire indicatrice de résultat ;
SI S[XOP] ≠ 0 ALORS DEBUT COPIE (XOP-3, IA-3, 3) ;
                          SI S[XOP] ≠ 1 ALORS IENTREE := 3
                    FIN Affectation de la valeur dans l'accumulateur réservé
                    à cet effet ;
SI T5 ≠0ALORS IA := IA-3 ; ALLERA AIGENTREE [IENTREE] ;
COMMENTAIRE Retour normal à ZENT, sauf si procédure formelle lorsque le type
effectif (arithmétique) n'est pas celui de la spécification. On revient
alors a MVRES1 ;
MVRES1 : WIAC := XOP-3 ; ALLERA INVTYPA ;

INVTYP : WIAC := IA-3 ;
INVTYPA : SI S[WIAC] = HVE ALORS INVTYPER : CER (WIAC)
                           SINON INVTYPRE : CRE (WIAC) ;
          IENTREE := 1 ; ENTREE ;
```

Hors programme

TAB ordre de création de tableau

variables particulières :

T6 nombre de bornes, HND Dimension

HAD adresse physique de la mémoire de la variable locale attachée au premier tableau à créer.

HADF Adresse en séquence de la mémoire locale attachée au dernier tableau à créer ;

T5 Clé nulle pour les tableaux de type entier

HA Index de stackage des valeurs (entières) des bornes et WIA sa valeur initiale

T7 Index origine d'implantation du tableau à créer

T9 Nombre d'éléments du tableau

Dans ZTAB3 : on implante successivement dans la tête de tableau

- m_j puis $N_j := M_j - m_j + 1$

et l'on calcule T9 := produit des N_j

Dans ZTAB4 :

T10 Index fin de tête de tableau.

On met à jour l'index IA après la dernière mémoire réservée pour le tableau, et cet encombrement est mis en place dans la tête du tableau.

Puis on affecte l'adresse d'implantation du tableau, à la mémoire locale qui lui est attachée.

ZTAB5 est la recopie des autres tableaux à créer.

IND et PVA ordres d'indexation.

T6 nombre d'Indices

T7 clé nulle pour les tableaux réels

HT index progressant dans la tête de tableau, pointe initialement la dimension

HA est l'index d'examen des indices et regresse à partir du dernier

T5 clé nulle pour l'ordre PVA. Fin page "Hors programme"

```
TAB : T6 := P[XCP] ; HND := T6+2 ;
      T5 := SI P[XCP+1] > H100000 ALORS 0 SINON 1 ;
      HAD := ADPHY (P[XCP+1]∧H017777) ; HADF := HAD + P[XCP+2] ;
      XCP := XCP+3 ;
      WIA := HA := IA ;
   ZTAB1 : POUR J := 1 PAS 1 JUSQUA T6 FAIRE
           DEBUT IA := IA-3 ; SI S[IA] ≠ HVE ALORS CRE (IA) ;
                 S[HA] := S[IA+1] ; HA := HA+1
           FIN Traitement des bornes ;
   T7 := IA ;
   S[IA+1] := HND ; IA := IA+2 ; COMMENTAIRE Implantation de la dimension
                                 dans la tête du tableau ;
   T9 := 1 ;
ZTAB3 : POUR RMD := WIA PAS 2 JUSQUA HA-2 FAIRE
      DEBUT SI S[RMD] <S[RMD+1] ALORS ERREUR (12) ;
            S[IA] := -S[RMD+1] ;
            RMA := S[IA+1] := S[RMD] - S[RMD+1]+1 ;
            T9 := T9 x RMA ; IA := IA+2
      FIN Implantation de la tête de tableau et calcul du nombre d'éléments ;
```

```
SI T5 ≠ 0 ALORS T9 := 2xT9 ; COMMENTAIRE Encombrement double d'un tableau réel ;
ZTAB4 : T10 := IA ;S[T7] := IA := IA+T9 ;
      SI IA > IAMAX ALORS ERREUR (5) ; S[HAD] := T7 ;
      COMMENTAIRE Fin de création d'un tableau ;
      HAD := HAD+1 ; RMD := T7 ; T7 := IA ;
   SI HAD = HADF ALORS DEBUT S[DP+1] := IA ; ENTREE
                       FIN de la création des tableaux
                       Mise à jour de la donnée de liaison-encombrement local ;
ZTAB5 : POUR RMD := RMD PAS 1 JUSQUA T10-1 FAIRE
        DEBUT S[IA] := S[RMD] ; IA := IA+1 FIN de la recopie de la tête de
        tableau ;
        ALLERA ZTAB4 ;

IND : T5 := 1 ; ALLERA ZPVA1 ;
PVA : T5 := 0 ;
      ZPVA1 : T6 := 0 ; HA := IA ;
      INPV1 : IA := IA-3 ;
              SI S[IA] = HADE ALORS ALLERA INPVE SINON
              SI S[IA] = HADR ALORS ALLERA INPVR ;
              T6 := T6+1 ; ALLERA INPV1 ;
      INPVR : T7 := 0 ; ALLERA INVP2 ; INPVE : T7 := 1 ;

      INPV2 : HT := S[S[IA+1]]+1 ; SI S[HT] ≠ T6 ALORS ERREUR (9) ;
                                   COMMENTAIRE Erreur nombre d'indices ;
              HT := HT+1 ; T11 := 0 ; T12 := 1 ;

      INPV4 : POUR K := 1 PAS 1 JUSQUA T6 FAIRE
```

```
DEBUT HA := HA-3 ; SI S[HA] ≠ HVE ALORS CRE (HA) ;
      T13 := S[HT] + S[HA+1] ; SI T13 < 0 ALORS ERREUR (10) ;
      T11 := T11 + T12 x T13 ; T12 := T12 x S[HT+1] ;
      SI S[HT+1] - T13 < 1 ALORS ERREUR (11) ;
      HT := HT+2
FIN T11 Rang de l'élément, T13 := IK-mK, T12 Produit de NK ;

    XOP := HT + (SI T7 ≠ 0 ALORS T11 SINON 2xT11) ; COMMENTAIRE Encom-
                                                    brement double des Tableaux Réel ;

    SI T5 ≠ 0 ALORS
       DEBUT S[IA+1] := XOP ; IA := IA+3 ; ENTREE FIN Stackage de
    l'adresse la variable indicée sans modifier la marque éventuelle
    du 3ème mot ;

    SI S[IA+2] = 1 ALORS IENTREE := 2 ;
    ALLERA SI S[IA] = HADR ALORS CRS SINON CES ;
    COMMENTAIRE Aller vers le chargement de la valeur en ayant éventuel-
    lement préparer l'inversion de Type ;

  NONDEFINI : XOP := -1 ; COMMENTAIRE voir Aig ;
STA : S[IA] := HETI ; S[IA+1] := XOP ; S[IA+2] := 0 ; ALLERA TIAMAX ;

ALL : SI S[IA+2] = -1 ALORS  ZALL1 : DEBUT IA := IA-3 ; ENTREE FIN ;
      TABLE (S[IA-2], XCP, NBE) ;
      SI S[IA-1] ≠ 0 ALORS PFMAJ (S[IA-1])
                     SINON SI NB = NBE ALORS ALLERA ZALL1 ;
      NB := NBE ; DP := DISPLAY[NB] ; IA := S[DP+1] ;
      ENTREE ;
```

```
AIG : IA := IA-6 ; SI S[IA+3] ≠ HVE ALORS CRE (IA+3) ;
      SI S[IA+4] ≤ 0 ALORS ALLERA NONDEFINI ;
      TABLE (S[IA+1], RMP, NBE) ;
      SI S[IA+4] > P [RMP] ALORS ALLERA NONDEFINI ;
      XOP := S[IA+2] ; SI XOP ≠ 0 ALORS IA := IA+3 ;
      ORFSC (SI XOP = 0 ALORS ZENT SINON RGETI , RMP - S[IA+4]) ;
      COMMENTAIRE Dans le cas d'un aiguillage paramètre XOP donne l'adresse
      de sa caractérisation dynamique requise par ORFSC. Dans ce cas après
      évaluation on reviendra à RGETI, on avait alors conservé l'accumulateur.
      RMP est l'adresse de la mémoire caractéristique de l'aiguillage ;
    RGETI : IA := IA-3 ; S[IA-2] := S[IA+1] ;
            SI S[IA+2] ≠ 0 ALORS S[IA-1] := S[IA+2] ;
            ENTREE ;

DO : RMP := P[XCP] ;
     RMA := DP+P[P[RMP]] ;
     S[RMA] := P[XCP+1] ; XCP := P[RMP]+1 ;
     ENTREE ; COMMENTAIRE RMP est l'adresse indirecte de l'Instruction contrôlée,
              RMA l'adresse de la variable locale attachée au POUR ;

RTP : XCP := S[DP+P[XCP]] ; ENTREE ;
     COMMENTAIRE La variable locale attachée au POUR, d'adresse DP+P[XCP]
     contient l'adresse de retour dans le Programme ;

SSP : S[IA] := XCP ; IA := IA+1 ; XCP := XOP ; ENTREE ;

RSP : XCP := S[IA-4] ; COPIE (IA-4, IA-3, 3) ; IA := IA-1 ; ENTREE ;
```

```
VAL : SI S[IA-1]≠0 ALORS IENTREE := 2 ;
      XOP := S[IA-2] ;
      ALLERA SI S[IA-3] = HADE ALORS CES SINON CRS ;
      COMMENTAIRE Après le chargement de la valeur, on ira sur l'inversion de
      type si le 3ème mot de l'accumulateur adresse indiquait la non correspon-
      dance ;

TFA : IA := IA-6 ; RMA := SI S[IA+3] ≠ HVE ALORS S[IA+3] SINON S[IA+4] ;
      SI RMA = 0 ALORS ALLERA ZENT ;
      T2 := RMA Λ H400000 ;
      RMA := SI S[IA] ≠ HVE ALORS S[IA] SINON S[IA+1] ;
      SI RMA = 0 ALORS ALLERA ZENT ;
      SI (RMA Λ H400000) = T2 ALORS ALLERA SAU ;
      ENTREE ;
      COMMENTAIRE On considère pour une valeur réel le 1er mot (poids fort
      mantisse) - On examine d'abord le pas et T2 est son signe, puis
      (variable contrôlée - valeur finale ) ;

FAI : RMP := ADPHY (P[XCP]) ; COMMENTAIRE Adresse de la variable contrôlée d'un
                              Faire simplifié ;
      S[RMP] := RMA := S[RMP] + P[XCP+1] ;
      SI RMA > P[XCP+2] ALORS XCP := XCP+4 ;
        SINON DEBUT RMP := P[XCP+3] ; RMA := DP+P[P[RMP]] ;
              S[RMA] := XCP-5 ; XCP := P[RMP]+1 ;
              FIN RMP adresse indirecte de l'Instruction Contrôlée
              RMA adresse de la variable locale attachée au POUR ;
        ENTREE ;
```

Hors programme

Ordres d'activation de procédure

1) formel APF,AFF

A7 Adresse de la caractérisation dynamique de la procédure paramètre effectif

RMP Adresse de la mémoire en table de la procédure.

Après réservation d'un accumulateur pour un indicateur de fonction la mémoire de résultat vaut 1 ou -1 selon que le type effectif est le type spécifié ou non.

2) activation Normale APN, AFN

DPO origine d'implantation du niveau que l'on active

T1 donne le numéro de bloc du niveau appelant, cadré comme il le sera dans le 1er mot de la caractérisation dynamique d'un paramètre effectif appelé par nom.

NB numéro de bloc de déclaration de la procédure (jusqu'à la phase APPE4)

HMOD Index pointant la tête de procédure objet (initialisé sur l'entrée et progressant)

Dans NLAP : mise à jour du compteur de ligne avec les éventuels numéros de ligne

Dans STANDARD : A5 est le nombre de paramètres effectifs supplémentaires d'une procédure standard à liste variable.

Dans APPEXO : Cas usuel ; on a pointé dans la tête de procédure objet le nombre de paramètres, et il doit être identique au nombre de paramètres effectifs de l'appel (XOP).

Dans APPE1 XCP : pointe la caractérisation statique du paramètre effectif

HMOD : pointe le modèle du formel correspondant.

HVAL marque d'appel par valeur pour le premier mot de la Cdy

NMOD partie spécification du formel

NCS partie qualitative de la caractérisation statique

CSAD partie quantitative (adresse) " "

PA1 déterminera dans le tableau TBORF (voir ci-dessous) la partie ORFS (ordre formel secondaire) sans l'indice de conversion du 1er mot de la Cdy.

HCVT indice de conversion

La procédure COMPEF vérifie la correspondance paramètre effectif-formel et détermine HCVT relatif à celle-ci.

Tableau TBORF
(voir aussi 9.3.1.
et 9.4.1.)

Il détermine la partie ORFS1 dans la phase d'Appel par nom.

Correspondance avec le nom symbolique de ORFS

0 NOMP

2 EXCV

4 EXCA

6 NOM

TBORF	PA1 = N° de famille du Paramètre effectif	
6	0	Variable Simple - Variable
0	1	Procédure - Etiquette - Aiguillage - Chaîne
4	2	Variable indicée
2	3	Expression (arithmétique, booléenne, de désignation
6	4	Nombre sans signe

Dans APPE8 Lorsque on atteint la partie supplémentaire d'une liste variable de procédure standard, on rétablit l'index HMOD sur le dernier modèle.

Dans DJPEF Le paramètre effectif de l'appel est un formel et la partie modèle de la Cdy du paramètre effectif correspondant à ce dernier, sert de partie qualitative de caractérisation statique actuelle.
HCVT1 Indice de conversion relatif au paramètre effectif origine.

Dans APPE4 Fin de la phase appel par nom des paramètres effectifs. Mise en place de la première donnée de liaison... On termine l'entrée du niveau procédure avant examen et appel par valeur des paramètres effectifs (indication par le bit de signe d_{18} du 1er mot de la Cdy).

Dans APP21,APPVAL

XOP Adresse physique de la Cdy du paramètre effectif examiné.
RMA partie modèle de la caractérisation dynamique permet de distinguer les différentes sortes de quantité à appeler par valeur.

```
APF :   T1 := 0 ; ALLERA ZAF1 ;
AFF :   T1 := 1 ;
    ZAF1 :  A7 := ADPHY (P[XCP]) ;
    ZAF2 : FORMEL := VRAI ;
         SI T1 = 0 ALORS S[IA] := 0
                   SINON DEBUT IA := IA+3 ;
                         S[IA] := SI IMPAIR (S[A7]) ALORS -1 SINON 1
                         FIN ;
                   RMP := S[A7+1] ; ALLERA ZAP2 ;

AFN : IA := IA+3 ; S[IA] := 1 ; ALLERA ZAP1 ;
APN : S[IA] := 0 ;
    ZAP1 : FORMEL := FAUX ; RMP := P[XCP] ;
    ZAP2 : DPO := IA+1 ;
          S[DPO+2] := XCP+1 ; S[DPO+3] := DP ; S[DPO+4] := NB ;
          COMMENTAIRE mise en place des 3ème, 4ème et 5ème données de liaison ;
          IA := DPO+5 ;
          T1 := NB x 2↑(-12) ; TABLE (RMP, HMOD, NB) ;

    APPEX : RMA := P[HMOD] ; HMOD := HMOD+1 ;
          SI RMA > 0 ALORS ALLERA APPEXO SINON
          NLAP : SI RMA ≠ H777777 ALORS DEBUT XNL := RMA ; ALLERA APPEX FIN
                 SINON
          STANDARD : DEBUT COMMENTAIRE Procédure standard a liste variable de
                     paramètres ;
                     SI P[HMOD] < XOP ALORS ERREUR (13) ;
                     A5 := S[IA] := P[HMOD] - XOP ; IA := IA+1 ;
                     FSTANDARD := VRAI ; ALLERA APPEX1
                     FIN ;
```

```
APPEXO : COMMENTAIRE Ce n'est pas une procédure standard à liste variable ;
         SI RMA ≠ XOP ALORS ERREUR (13) ; FSTANDARD := FAUX ;
APPEX1 : XCP := XCP-1 ;

APPE1 : ; COMMENTAIRE Constitution des caractérisations dynamiques en appel
          par nom ;
          SI XOP = 0 ALORS ALLERA SI FORMEL ALORS APPE6 SINON APPE4 ;
          HVAL := SI IMPAIR (P[HMOD]) ALORS H400000 SINON 0 ;
          NMOD := P[HMOD]∧HOO7700 ; HMOD := HMOD+1 ;
          CSAD := P[XCP]∧HO17777 ; NCS := P[XCP]∧H760000 ;
          SI NCS = H400000 ALORS ALLERA DJPEF ;
          SI NCS = HO20000 ALORS NCS := H300000 ; COMMENTAIRE Modification pour
                                                  une chaîne ;
          PA1 := (NCS x 2↑(-16))∧3 ; COMMENTAIRE PA1 numéro de famille du
                                     paramètre effectif ;

          SI PA1 ≠ 2 ALORS
          VALVI : DEBUT SI (NCS∧HO20000) = 0 ALORS DEBUT PA1 := 4 ; HVAL := H400000
                        NCS := NCS∧H400000 ;                          FIN ;
                  FIN de la préparation pour un paramètre effectif nombre pur
                  (PA1 vaut 4) ou variable indicée ;
          COMPEF ;
          SI PA1 = 0 ALORS CSAD := ADPHY (CSAD) ;
       CDYN : COMMENTAIRE Construction en stack de la caractérisation dynamique ;
              S[IA] := HVAL + T1 + NMOD + TBORF[PA1] + HCVT ;
              S[IA+1] := CSAD ; S[IA+2] := DP ;
       CDYN2 : IA := IA+3 ; SI IA > IAMAX ALORS ERREUR (5) ;
               XOP := XOP-1 ;
       CDYN21 : SI FSTANDARD ALORS
```

```
APPE8 : DEBUT SI XOP = 0 ALORS ALLERA APPE9 ;
            SI XOP - A5 < 1 ALORS HMOD := HMOD+1 ;
        FIN procédure standard à liste variable ;
        ALLERA APPE1 ; COMMENTAIRE Bouclage ;

DJPEF : ; COMMENTAIRE Le paramètre effectif est déjà un formel ;
        CSAD := ADPHY (CSAD) ;
        PRPFA : T2 := S[CSAD] ; NCS := (T2 ∧ H007700) x 2↑5 ;
                RMA := T2 ∧ H000077 ; HCVT1 := SI IMPAIR (RMA) ALORS 1 SINON 0 ;
        COMPEF ;
        SI RMA ≠ 8 ALORS DEBUT PA1 := 0 ; CSAD := CSAD+1 ; ALLERA CDYN
                         FIN C'est un formel appelé par valeur, devenu variable
                         locale ;
        CDYNF : HCVT := SI RMA = 3 ALORS 1 SINON (SI IMPAIR (HCVT+HCVT1)
                                                    ALORS 1 SINON 0) ;
                S[IA] := (T2 ∧ H770076) + HVAL + NMOD + HCVT ;
                S[IA+1] := S[CSAD+1] ; S[IA+2] := S[CSAD+2] ;
                ALLERA CDYN2 ;

APPE9 : ; COMMENTAIRE Fin de constitution des caractérisations dynamiques en
        appel par nom pour une liste variable de procédure standard ;
        NB := 1 ; DISPLAY[1] := DP := DPO ;
        S[DP] := DISPLAY[0] ; S[DP+1] := IA+2 ;
        SI IA+2 > IAMAX ALORS ERREUR (5) ; XOP := DP+6 ;
        S[IA] := HMOD+1 ; S[IA+1] := IA ; IA := IA+1 ; ALLERA APPE21 ;

APPE6 : ; COMMENTAIRE fin de constitution des caractérisations dynamiques en
        appel par nom. Remise à jour du Display pour un formel procédure ;
        A1 := NB ; PFMAJ(A7) ; NB := A1 ;
```

```
APPE4 : S[DPO] := DISPLAY[NB] ; NB := NB+1 ; DP := DISPLAY[NB] := DPO ;
        RMP := DP + P[HMOD] ; COMMENTAIRE P[HMOD] est l'indicateur de réservation ;
        SI RMP+2 > IAMAX ALORS ERREUR (5) ;
        S[RMP] := HMOD+1 ; S[RMP+1] := IA ; IA := RMP+1 ;
        S[DP+1] := IA+1 ; COMMENTAIRE Stacker l'adresse programme objet du corps
        de procédure et l'adresse terminale des caractérisations dynamiques ;
        XOP := XOP+2 ;

APPE2 : ; COMMENTAIRE Examen des caractérisations et appel par valeur des
          paramètres qui doivent l'être ;
          XOP := XOP+3 ;
APPE21 : SI XOP = S[IA] ALORS
          APPE3 : DEBUT COMMENTAIRE Toutes les caractérisations dynamiques ont
          été consultées. On met à jour le compteur programme et libère les
          2 mémoires locales supplémentaires ;
          IA := S[DP+1] := S[DP+1]-2 ; XCP := S[IA] ; ENTREE ;
          FIN DES ORDRES DACTIVATION DE PROCEDURE ;
          SI S[XOP] > 0 ALORS ALLERA APPE2 ; COMMENTAIRE Bouclage, le paramètre
          n'est pas appelé par valeur ;

APPVAL : ; COMMENTAIRE Appel par valeur d'un paramètre ;
          S[IA+1] := XOP ; IA := IA+2 ;
          T1 := S[XOP] := S[XOP]∧H377777 ;
          RMA := (T1∧H007700) x2 ↑(-8) ;
          SI RMA = H13 ALORS ALLERA APETV SINON SI IMPAIR (RMA) ALORS ALLERA
                                                                  APTBV ;

APTYV : ; COMMENTAIRE formel spécifié TYPE, appel par valeur ;
          XCP := RAVAP ; ALLERA SI RMA = 2 ALORS CRF SINON CEF ;
```

```
        RAVAP : HCODE ; COMMENTAIRE Retour après exécution de l'ordre formel
                stackant la valeur, en langage de programme objet à cet ordre.
                HCODE symbolise l'ordre objet Ø COD (cf. 10.1. et 9.4.2.).
                Il permet de reprende l'exécution en séquence en langage
                machine ;
                ZRAV : IA := IA-5 ; XOP := S[IA+1] ;
                       S[XOP] := (S[XOP] ∧ H377700) +8 ; XOP := XOP+1 ;
                       SI S[IA+2] = HVE ALORS RANGE (IA+2) SINON RANGR (IA+2) ;
                       XOP := XOP+2 ; ALLERA APP21 ;
                     COMMENTAIRE Après modification du 1er mot de la carac-
                     térisation dynamique, rangement de la valeur ;
APTBV : ERREUR (6) ; COMMENTAIRE Appel interdit d'un Tableau par valeur ;
APETV : ; COMMENTAIRE formel spécifié étiquette, appel par valeur ;
        SI (T1 ∧ H000077) = 0 ALORS
           RGETV2 : DEBUT IA = IA-2 ; ALLERA APPE2 FIN C'était un paramètre
                    effectif étiquette ;
           XCP := RETV ; ALLERA EAF ;

           RETV : HCODE ; COMMENTAIRE Retour après exécution de l'ordre formel
                          stackant la valeur de l'expression de désignation ;
                  RGETV : IA := IA-5 ; XOP := S[IA+1] ;
                          COPIE (XOP, S[IA+4], 3) ;
                          S[XOP] := S[XOP] ∧ H777700 ; S[XOP+1] := S[IA+3] ;
                          ALLERA APPE2 ;
                          COMMENTAIRE On recopie la caractérisation dyna-
                           mique puis on indique NOM (étiquette) dans le
                          1er mot, et place l'étiquette dans le second ;
```

Hors_programme

Procédure COMPEF (voir 9.4.1.1.) on vérifie l'identité (au cadrage près) de la partie qualitative de la caractérisation statique du paramètre effectif et de la spécification du formel correspondant, ou l'on recherche cette association dans le tableau TBMOCS. On détermine aussi l'indice de conversion HCVT.

Tableau TBMOCS

NCS et NMOD sont donnés par leurs représentations symboliques, conformes aux indications des tableaux, respectivement pour :
les caractérisations statiques
les modèles-(annexe ch. 9)

NCS $d_{18}....d_{13}$	NMOD $d_{12}....d_{7}$	Partie nulle $d_6 d_1$	
PR	PO		HCVT := 0
PE	PO		
PB	PO		
EXPD	ET		
EXPB	QB		
PB	QB		
PE	QE		
PR	QR		
QE	QR		HCVT := 1
PE	QR		
EXPA	QR		
QR	QE		
PR	QE		
EXPA	QE		
TE	TR		
TR	TE		
PE	PR		
PR	PE		

```
PROCEDURE COMPEF ;
   DEBUT COMMENTAIRE Cette procédure vérifie la correspondance paramètre
   effectif-formel, et détermine l'éventuel indice de conversion HCVT ;
         SI (NCS x 2 ↑(-5)) = NMOD ALORS ALLERA COMPF1 ;
         RMA := NMOD + NCS ;
         POUR I := 1 PAS 1  JUSQUA 18 FAIRE
         DEBUT COMMENTAIRE On recherche RMA association de la caractérisation
         statique du paramètre effectif et de la spécification du formel
         dans la table de correspondance TBMOCS ;
         SI RMA = TBMOCS[I] ALORS ALLERA SI I > 9 ALORS NON TYP SINON COMPF1 ;
         FIN ;
         ERREUR (14) ;
      NONTYP : HCVT := 1 ; ALLERA COMPF2 ;
      COMPF1 : HCVT := 0 ;
      COMPF2 : FIN de COMPEF ;
```

Hors programme

TB FORMEL

Aiguillage des Ordres formels selon la caractérisation dynamique (poids faible de premier mot voir 9.5.1.)

Indication de la caractérisation dynamique ↓		ARF	AEF	CRF	CEF	ORDRE OBJET sur formel ←
NOMP	0	ER	ER	ORFSA	ORFSA	
	1	ER	ER	ORFSA	ORFSA	
EXCV	2	ER	ER	ZENT	ZENT	
	3	ER	ER	EXCVTR	EXCVTE	
EXCA	4	ZENT	ZENT	EVALR	EVALE	
	5	EXAC	EXAC	EVALRC	EVALEC	
NOM	6	ARN	AEN	CRS	CES	
	7	NOMRC	NOMEC	VNOMRC	VNOMEC	
VAL	8	ARN	AEN	CRS	CES	
Symbole	Décimal					
ORFS						

```
ARF : I := 1 ; ALLERA ORFS ;
AEF : I := 10 ; ALLERA ORFS ;
CRF : I := 19 ; ALLERA ORFS ;
CEF : I := 28 ;
      ORFS : ; COMMENTAIRE Ordres sur formel spécifié TYPE. L'ordre formel
             secondaire indiqué dans la caractérisation dynamique et le type
             d'ordre donné par I, détermine une étiquette de l'aiguillage
             TBFORMEL. Voir page précédente et 5 ;
             T3 := S[XOP] ∧H0000077 ;
             SI T3 < 2 ALORS ALLERA TBFORMEL[I+T3] ; COMMENTAIRE paramètre
                                  effectif procédure (indicateur de fonction) ;
             SI T3 ≤ 5 ALORS ORFSC (TBFORMEL[I+T3], S[XOP+1]) ;
                       COMMENTAIRE Activation du sous programme objet du
                       paramètre effectif, retour dans l'interpréteur à l'adresse
                       donnée par TBFORMEL ;

             XOP := SI T3 = 8 ALORS XOP+1 SINON S[XOP+1] ;
             ALLERA TBFORMEL[I+T3] ; COMMENTAIRE paramètre effectif appelé
                    par valeur, ou variable simple ;

ORFSA : XCP := T1 := XCP-1 ; CSAD := XOP ; XOP := 0 ;
        ALLERA ZAF2 ;

EXCVTR : SI S[IA-3] = HVE ALORS CER (IA-3) ; ENTREE ;

EXCVTE : SI S[IA-3] ≠ HVE ALORS CRE (IA-3) ; ENTREE ;

EVALR : EVALRC ; IA := IA-3 ; XOP := S[IA+1] ;
                ALLERA SI S[IA] = HVE ALORS VNOMRC SINON CRS ;
```

```
EVALE : EVALEC : IA := IA-3 ; XOP := S[IA+1] ;
                 ALLERA SI S[IA] = HVE ALORS CES SINON VNOMEC ;

VNOMRC : WIAC := IA ; IENTREE := 4 ; ALLERA CES ; COMMENTAIRE On fera ensuite
                                    la conversion de Type Entier en Réel ;

VNOMEC : WIAC := IA ; IENTREE := 5 ; ALLERA CRS ;

EXAC : S[IA-1] := SI S[IA-1] = 0 ALORS 1 SINON 0 ; ENTREE ;

NOMRC : IENTREE := 6 ; ALLERA AEN ; COMMENTAIRE On revient sur EXAC1 ;

NOMEC : IENTREE := 6 ; ALLERA ARN ;

        EXAC1 : ; COMMENTAIRE Mise en place après ARN ou AEN provoquée par
                  NOMRC ou NOMEC de l'indice de conversion ;
                  S[IA-1] = 1 ; IENTREE := 1 ; ENTREE ;


EAF : ; COMMENTAIRE Ordre sur un formel spécifié ETIQUETTE ou AIGUILLAGE ;
      SI (S[XOP]∧H0000077) ≠ 2 ALORS
         DEBUT COMMENTAIRE paramètre effectif étiquette ou aiguillage ;
         S[IA] := HETI ; S[IA+1] := S[XOP+1] ; S[IA+2] := XOP ;
         ALLERA TIAMAX
         FIN
         SINON ORFSC (RETFX, S[XOP+1]) ; COMMENTAIRE Activation du sous
         programme objet d'évaluation de l'expression de désignation. Retour
         dans l'interpréteur à RETFX ;
```

```
        RETFX : XOP := IA-1 ;
                SI S[XOP] = 0 ALORS S[XOP] := SI XCP = RETV ALORS S[XOP-3]
                                                SINON ADPHY (P[XCP-1]∧HO17777) ;
                ENTREE ;

RTN : ; COMMENTAIRE Ordre de retour d'un sous programme objet, correspond à
        l'activation par la procédure ORFSC ;
        IA := IA-4 ; T2 := S[IA-3] ;
        XCP := S[IA-2] ; DP := S[IA-1] ; NB := S[IA] ;
        REMAJ ;
        ALLERA T2 ; COMMENTAIRE T2 est l'adresse de retour dans l'interpréteur
                    que l'on a stackée à l'activation du sous programme ;

PROCEDURE ORFSC (RETOUR, SOUSPROGRAMME) ;
          ETIQUETTE RETOUR, SOUSPROGRAMME ;
  DEBUT COMMENTAIRE RETOUR est l'adresse de retour dans le programme objet
  (ou d'un ordre du langage objet dans l'interpréteur), SOUS PROGRAMME l'adresse
  d'entrée du sous programme (de paramètre effectif) à activer ;
  S[IA] := RETOUR ; S[IA+1] := XCP ; S[IA+2] := DP ; S[IA+3] := NB ;
  IA := IA+4 ;
  XCP := SOUSPROGRAMME ;
  SI XOP ≠ 0 ALORS PFMAJ (XOP) ; COMMENTAIRE XOP est nul uniquement pour
  un aiguillage non paramètre, et donne dans les autres cas l'adresse de la
  caractérisation dynamique ;
  ALLERA ZENT ;
  FIN de ORFSC : ACTIVATION D UN SOUS PROGRAMME OBJET ;
```

```
PROCEDURE PFMAJ (ADCDY) ;
    DEBUT COMMENTAIRE Remise à jour des index DP, NB et du Display grace à
    l'adresse ADCDY, de la caractérisation dynamique de paramètre effectif,
    définissant la structure où l'on replace ;
    NB := (S[ADCDY]Λ H770000) x 2↑(-12) ;
    DP := S[ADCDY+2] ;
    REMAJ ;
    FIN DE PFMAJ ;
```

C H A P I T R E XII

EDITION ou CODIFICATION du TEXTE ALGOL SOURCE

La compilation comporte 2 phases distinctes; l'édition, puis la génération qu'effectuent respectivement les programmes éditeur (où codifieur) et générateur. Ces 2 programmes n'existent pas simultanément dans la mémoire de la machine ; le générateur sera implanté au même emplacement que l'éditeur, lorsque ce dernier aura achevé son traitement.

L'édition, phase initiale de la compilation, consiste à traduire le programme source Algol dans le langage de codification. Ce dernier est un intermédiaire entre le langage Algol et le langage objet du compilateur. Langage source pour le générateur, le langage de codification doit être bien adapté à la phase suivante de génération et résoudre plusieurs problèmes qui la faciliteront.

L'éditeur produit la chaîne codée, équivalente au programme source Algol. Nous reconnaîtrons en indiquant les caractéristiques essentielles du langage de codification, vis-à-vis du langage Algol, certains traits du langage objet :

1) Chaque élément est de longueur déterminée (fixe et 1 mot)
2) La représentation d'un identificateur résoud les problèmes de portée.
3) Une "préréservation" est faite pour chaque niveau de nomenclature (pris au sens du programme objet) ; elle est donnée par un "indicateur" de réservation.

4) Les déclarateurs de type n'existent pas. En outre :

. les déclarations de variables simples sont supprimées

. les têtes de déclaration de procédure sont traitées (au sens du programme objet, elles ne seront pas modifiées par la génération)

. certains délimiteurs sont éliminés ou modifiés dans les déclarations

5) Les commentaires sont supprimés et les effets de présentation typographiques éliminés (sous réserve des numéros de ligne examiné plus loin).

6) Certaines erreurs du texte Algol source sont détectées.

La chaîne codée produite est implantée directement en mémoire ; de cette manière la phase suivante de la compilation ne nécessitera que le chargement du générateur. En effet, sortir la chaîne codée au fur et à mesure de sa construction conduirait à des manipulations supplémentaires gênantes pour le calculateur minimum (entrée sortie par un lecteur-perforateur rapide de bandes). Ses caractéristiques seraient en outre moins souples (table d'étiquettes...).

Pour garder cette chaîne codée en mémoire, il faut disposer d'une place suffisante si l'on veut compiler des programmes de dimensions raisonnables. Ne pas entrer le texte Algol source permet d'y parvenir. Il est lu ligne par ligne (ou carte par carte) et chaque ligne est traitée avant lecture de la suivante : la mémoire du calculateur ne contient que la ligne en traitement.

En réalité, on travaille en bascule sur 2 lignes : 2 zones contiennent alternativement le texte source de la ligne traitée, et celui de la ligne suivante. Le traitement effectif a lieu sur une troisière zone qui donne le texte décodé de la ligne traitée : suppression des codes à signification typographique (rouge, noir, espace, tabulation) ou de codage (majus-

cule, minuscule) ;on n'aque des caractères significatifs ramenés à un code interne unique ; seul le microprogramme de décodage est à changer si l'on utilise un ensemble d'entrées différent (lecteur de cartes, flexo de code différent...).

Lorsque la ligne texte interne est traitée, on décode la ligne suivante dont le texte externe est en machine, puis on donne l'ordre de lecture d'une nouvelle ligne.

On a pratiquement levé l'inconvénient de la disparition du texte source lors de la détection d'erreur en utilisant un numéro de ligne (ou de carte). Chaque fois que l'on traite une nouvelle ligne(ou carte), on incorpore son numéro à la chaîne codée. A la génération, il permet de tenir à jour un compteur de ligne et d'identifier la ligne où s'est produit l'erreur détectée ; de plus, il est conservé dans le programme objet (ordre *L*) et on a vu qu'ilyjouait un rôle identique.

Pendant la codification, on imprime la ligne traitée : son texte externe est effectivement en machine ; on l'accompagne du numéro de ligne (ou de carte) et du numéro de sa première unité syntaxique (12.1.) Une erreur est indiquée par un numéro de type d'erreur et le numéro de l'unité syntaxique où on l'a reconnue.

On peut opter pour l'impression, soit systématique du programme, soit uniquement des lignes erronnées, ce qui accélère le traitement.

12.2 Langage de Codification et Chaîne Codée

On appellera <symbole de base> ::= <Delimiteur> définition restreinte par rapport à Algol

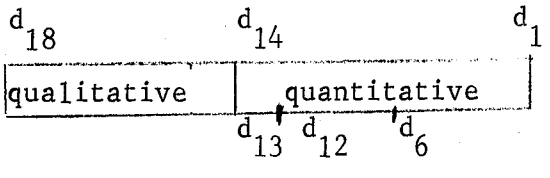
et <unité syntaxique> ::= <symbole de base>|<Identificateur>|<nombre dans signe>|<valeur logique>

Par ailleurs un élément de la chaîne codée occupe un mot et sera :

- une unité syntaxique
- un Indicateur de Réservation
- un modèle de paramètre formel
- un nombre de paramètres formels
- 3 caractères d'une chaîne
- un numéro de ligne (ou de carte)

Compte tenu des liens qu'il présente avec le langage objet, nous n'étudierons que rapidement le langage de codification.

Chaque élément qui est une unité syntaxique ou un numéro de ligne se décompose (1mot) en :



1°) partie qualitative 5 digits d_{18} - d_{14}

2°) partie quantitative 13 digits d_{13} - d_1

Le contexte qui précède les autres éléments en chaîne codée détermine la signification et rend inutile la distinction qualitatif-quantitatif de l'information. Cependant les modèles ont une partie qualitative pour plus de clarté (mise au point).

12.1.1. Codification des unités syntaxiques.

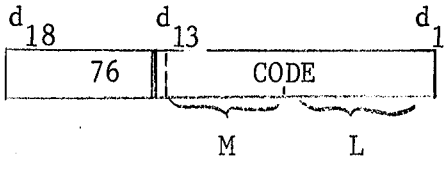
La partie qualitative permet de distinguer les différentes catégories d'unités syntaxiques et est particulièrement adaptée au problème de représentation des identificateurs ou nombre pur.

La partie qualitative d'un identificateur reflète le genre de la quantité déclarée ou la spécification du paramètre formel, cependant que la partie quantitative le particulise de manière commode pour la génération ou est même directement utilisable dans le langage objet.

12.1.1.1. Symbole de base

partie qualitative : 76

partie quantitative : le code du symbole de base qui se décomposera pour le générateur en priorité (caractère M) et numéro (caractère L)

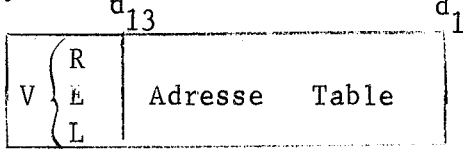


On trouvera en annexe A-XII les codifications.

12.1.1.2. Nombre Pur

Ce sont les nombres sans signe et les valeurs logiques VRAI et FAUX . Les tables des entiers et des réels du texte source, utilisées dans le programme objet sont construites à la codification. L'équivalent à l'exécution pour les valeurs logiques VRAI et FAUX est déjà connu : mémoires 27 et 25 (0 et 1).

La partie quantitative d'un nombre pur est très naturellement l'adresse physique de sa valeur en table : ce sera la représentation utilisée dans le programme objet. La partie qualitative distingue suivant le type VR, VE, VL (Symbolique).



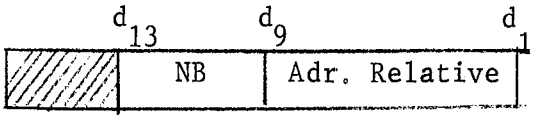
12.1.1.3. Identificateur.

On a vu dans le programme objet, que chaque identificateur est attaché à un niveau de nomenclature ; ce dernier est déterminé à la codification et son numéro de bloc figurera (d_{13}-d_{10}) dans la partie quantitative de l'identificateur. Comme on doit particulariser l'identificateur, on lui attribue selon l'ordre lexicographique des déclarations (ou spécification) un numéro noté dans l'autre section (d_9-d_1) de sa partie quantitative.

En fait, on en profite dans le cas d'identificateurs de variable simple, tableau et paramètre formel , pour donner de suite l'adresse relative de la (les)mémoire locale correspondante lors de l'exécution. Ce n'est pas un simple numéro d'occurence local. La partie quantitative est alors exactement l'adresse dynamique, telle qu'elle est utilisée dans le programme objet.

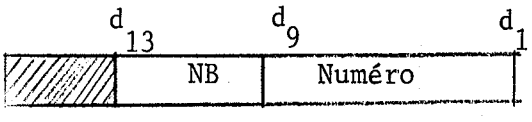
Dans les autres cas, il faut se contenter d'indiquer un numéro : la numérotation est globale pour l'ensemble du programme, mais distingue Etiquette, Procédure, Aiguillage (cette séparation apporte la certitude pratique de ne pas être limité par les 9 digits (512) de toute numérotation).

Partie quantitative des Identificateurs



a) Adresse Dynamique (cf. Programme objet)

pour les . Variables simples
. Tableaux
. Paramètres formels



b) Numéro de bloc et Numéro global

pour les . Procédures (déclarées)
. Etiquette
. Aiguillage

avec distinctions des 3 catégories.

Partie qualitative des identificateurs

Elle donne les informations suivantes :

a) Identificateur de paramètre formel ou de quantité déclarée

[Situation indiquée par le digit de signe d_{18} valant respectivement 1 ou 0]. Cette distinction établie, les autres informations sont identiques dans les 2 cas : déclarations et spécification jouent un rôle analogue pour la codification (12.1.2.3.)

b) Catégorie

Elle correspond aux différentes quantités reconnues par ALGOL en ajoutant la catégorie chaîne (n'existant que dans le cas d'un formel) et en séparant procédure-fonction et procédure (selon la déclaration, ou spécification, <TYPE> PROCEDURE et PROCEDURE). La numérotation de la partie qualitative leur est cependant commune.

On aura : Variable simple, Tableau, Procédure-fonction, Procédure, Etiquette, Aiguillage, Chaîne.

c) le type Algol

pour les variables simples, tableaux, fonctions, soit REEL, ENTIER, BOOLEEN. La codification condensée au maximum n'a pas permis de séparer les 2 informations qui seront effectivement intéressante à la génération ; le type (arithmétique ou booléen) et l'encombrement (1 pour les entiers et booléens, 2 pour Réel).

Remarque : Procédure Standard : La table des procédures standard citées dans le programme source, table que l'on utilise dans le programme objet (3.2.5.) est construite dès la codification. Un identificateur de procédure standard est représenté dans la chaîne codée par l'adresse physique de cette mémoire de table.

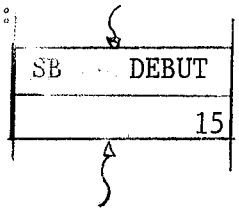
Cette mémoire contient outre l'adresse de chargement (ou d'entrée), l'information catégorie (d_{18} d_{17} d_6 : procédure standard, fonction standard réel, entier, booléen). Les digits de catégorie-type sont utilisés à la génération et effacés avant l'exécution.

12.1.2. Bloc - Déclaration - Niveau.

Les niveaux au sens du programme objet sont déterminés à la codification ; les déclarations (ou spécifications) permettent de construire la représentation décrite des identificateurs ; celles qui provoquent l'attribution d'adresse dynamique (spécification, déclarations de variables simples et tableaux) donnent en conséquence l'"Indicateur de Réservation" du niveau tel qu'il est utilisé dans le programme objet.

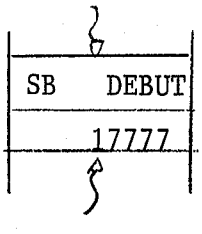
Dans un bloc l'indicateur de Réservation suit le SB DEBUT en chaîne codée, les instructions composées sont caractérisées simplement par un pseudo-indicateur de réservation (17777_8 valeur que ne peut atteindre un véritable indicateur limité à 777_8)

Eg :



Bloc

Indicateur de Réservation



Instruction Composée

Nous examinerons ce que deviennent les déclarations en chaîne codée ; nous avons déjà vu leur rôle en ce qui concerne la représentation des identificateurs.

Les déclarateurs de type n'existent pas en chaîne codée.

12.1.2.1. Déclarations de Type et de Tableau

Elles provoquent la "réservation" pour chaque quantité déclarée. 1 mémoire locale : variable simple de type ENTIER ou BOOLEEN, 2 mémoires locales : variable simple de type REEL, 1 mémoire locale : Tableau quel qu'en soit le type.

En outre 1) les déclarations de type sont supprimées (elles n'ont plus de rôle à jouer)

2) les déclarations de Tableau sont modifiées ; elles ont la construction suivante en chaîne codée :

<déclaration de tableau> ::= TABLEAU <liste de tableaux>

avec

<liste de bornes> ::= <expression>|<liste de bornes>, <expression>

<liste de tableaux> ::= <identificateur> [<liste de bornes>]|
<identificateur> <liste de tableaux>

On notera que chaque section de tableau source donne naissance en chaîne codée à une nouvelle déclaration de tableau : le délimiteur source "," est remplacé par ; TABLEAU.

Les délimiteurs "," séparant les identificateurs de tableau sont supprimés et ":" séparant bornes inférieures et supérieures est remplacé par "," (on peut écrire indifférement , ou : dans une liste de bornes source)

Exemple la déclaration suivante dans le programme source

ENTIER TABLEAU A, B, C [1 : 10, I x J : K+1], D[K : L]

12.1.2.2. Déclaration d'Aiguillage

Elle reste inchangée en Chaîne codée

Précisons cependant que les expressions de désignation de la liste d'aiguillage sont traitées comme toute expression : elles ne doivent faire intervenir que des identificateurs du niveau de déclaration ou d'un niveau supérieur ("imbriquant").

Les étiquettes sont soumises à la réserve de 12.1.2.4.

12.1.2.3. Déclaration de Procédure

La caractérisation essentielle est la suppression de la liste de paramètres formels, de la partie valeur et de la partie spécification. Elles sont remplacées par ce qu'on a appelé tête de procédure dans le programme objet, constituée sous sa forme définitive.

<tête-objet de procédure> ::= <Nombre de paramètres formels> <liste de modèles> <Indicateur de Réservation>

Le symbole ; qui termine la déclaration de procédure est remplacé par le symbole ;FP en chaîne où l'on a alors

<déclaration de Procédure> ::= PROCEDURE <Identificateur de Procédure> <tête objet de Procédure> ; <corps de Procédure> ;FP

Rappellons que pour la génération, la partie qualitative des identificateurs de procédure permettra de reconnaitre les procédures des procédures fonctions.

. Les spécifications obligatoires pour tous les formels, jouent un rôle analogues aux déclarations, pour définir la représentation des identificateurs : un identificateur de formel a même partie qualitative, que l'identificateur de la quantité dont la déclaration correspondrait à la spécification; le digit de signe d_{18} vaut alors 1 (soit -, au lieu de 0 soit +) et indique la situation formel.

Comme on l'a vu, la partie quantitative est l'adresse dynamique d'une mémoire locale du niveau, chaque formel "réservant" 3 mémoires locales prises en compte dans l'indicateur de Réservation du niveau.

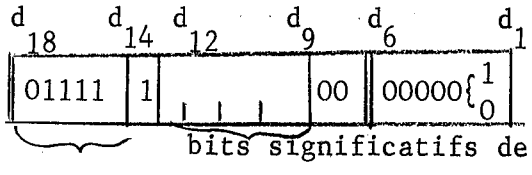
Le modèle d'un formel (1 mot) donne sa spécification et son mode d'appel par nom ou valeur (9.1.).

Remarque

On indiquera seulement que l'information spécification est obtenue en décalant de 5 digits vers la droite la partie qualitative du formel. Les digits significatifs de la spécification (d_{12}-d_9) sont alors identiques aux digits (d_{17}-d_{14}) de la partie qualitative d'un identificateur de déclaration correspondante (où $d_{18} = 0$) : ceci explique certaines particularités de la vérification correspondante paramètre effectif-formel à l'exécution (Ch. 9).

Rappel

Modèle



d_1 = mode d'appel { 0 → par NOM ; 1 → par valeur

Indicatif qualitatif "MOD"

On rappelle que le corps de procédure devient une instruction composée s'il est un bloc non étiqueté (4.2. et 12.2.3.1.).

12.1.2.4. Etiquetage.

Les étiquettes ont un caractère local et se comportent comme si elles étaient déclarées en tête du plus petit bloc comprenant l'instruction qu'elles étiquettent (cf. 4.1.3. du Revised Report on Algol 60). Aussi considère-t'on comme "déclaration" d'étiquette la situation suivante du texte source <Identificateur> : <Instruction>. On construit la représentation de l'identificateur (les étiquettes numériques sont interdites) comme dans le cas des véritables déclarations de quantité. A la différence de ces dernières

qui caractérisent l'ouverture d'un niveau (sous réserve possible pour les corps de procédure), l'étiquetage ne fait qu'utiliser le dernier niveau ouvert ou niveau courant. C'est en particulier pour cette raison, que le programme lui-même est en chaîne codée le bloc de numéro 0, même s'il est une instruction composée dans le texte source.

Par ailleurs la forme d'une instruction étiquetée est identique en chaîne codée et dans le programme source.

.Lorsqu'une proposition POUR porte sur une instruction composée, les étiquettes internes à cette instruction composée n'ont pas d'existence à l'extérieur (ch. 8) Tout en appartenant au niveau de nomenclature où est donnée l'instruction POUR, et elles sont en effet son numéro de bloc, elles sont localisées dans l'instruction composée (en particulier les mêmes identificateurs sont utilisables à l'extérieur dans le même niveau). Ceci est une option de vérification, aisément supprimable en forçant un aiguillage de l'éditeur, et n'apporte que des restrictions minimes. On détectera les erreurs dues à:"l'effet d'une instruction ALLERA, extérieure à une instruction POUR, et qui se réfère à une étiquette intérieure à l'instruction POUR, n'est pas défini " (4.5.6. Revised Report on Algol 60). On signale en outre comme cause possible d'erreur les instructions contrôlées par un POUR (exemple Programmes P-IV en annexe) qui sont étiquetées.

12.2. Editeur et exécution de la Codification.

Le travail essentiel de la codification est le traitement des identificateurs : construire leur représentation et l'utiliser dans la traduction du programme source en chaîne codée. Si l'on fait exception du rôle et de la forme des déclarations en chaîne codée, il est en effet simple.

Dans le cas d'une expression, d'une instruction, le principe de traitement est le suivant, lorsqu'on pointe en examinant séquentiellement le programme source :

1) Un symbole de base : Sitôt reconnu, sa représentation connue de ce fait est écrite en chaîne codée.

2) Un nombre sans signe : sa valeur (représentation machine) est créée, implantée en tables des nombres purs et l'adresse de cette implantation permet de construire la représentation du nombre ; elle est alors écrite en chaîne codée.

3) Un identificateur : on dispose d'une table où sont indiqués les identificateurs-source des quantités déclarées et leur représentation construites au moment de la déclaration correspondante. Dès que l'on a obtenu l'identificateur source pointé dans le programme, on le cherche dans cette table

   a) il est présent dans la table : on obtient donc sa représentation que l'on écrit en chaîne codée.

   b) il est absent de la table : ceci est du, par exemple, aux étiquettes, à l'ordre lexicographique des déclarations du niveau. Dans ce cas, on ne peut qu'écrire une représentation fictive en chaîne codée. On note alors dans la table l'identificateur source et l'adresse en chaîne codée de l'absence de représentation ; lorsqu'on l'aura obtenue on pourra donc la mettre en place.

On remarquera qu'en fait, la reconnaissance de certains symboles de base dans les expressions et instructions du programme source provoque, en plus du traitement habituel de passage en chaîne codée, des vérifications : on détecte ainsi des erreurs du texte source dès la codification.

D'une manière générale, les symboles de base DEBUT, FIN, les déclarateurs du programme source, liés aux problèmes de niveau et de représentation d'identificateurs déclenchent des programmes de traitement particuliers.

Nous n'examinerons que très rapidement le travail de l'Editeur ; on pourra se reporter à la note [7] (organigrammes de Codification).

12.2.1. Niveau et Stacks

Les identificateurs sont locaux à un niveau de nomenclature et on ne peut avoir une table globale pour le programme : on utilise une pile dite stack des identificateurs où l'on ouvre des "niveaux-stack". La table des identificateurs employée à un instant donné est celle du dernier niveau stack ouvert. On protège ainsi les identificateurs courants de ceux des niveaux précédents : leur signification est localisée. La régression de niveau -stack permet en particulier de recouvrer la signification antérieure à l'ouverture du niveau, en effaçant ce niveau du stack. Ces opérations dépendantes de la structure de blocs d'Algol, sont parallèles à son analyse, contrôlée par une autre pile, "le stack-Début".

Nous reviendrons sur le fonctionnement du stack-identificateur après avoir étudié le traitement des blocs et des déclarations. Indiquons cependant que l'on fait une ouverture de niveau-stack

a) à l'entrée dans un nouveau niveau de nomenclature de texte source (bloc, déclaration de procédure).

b) pour localiser des identificateurs dans un"faux bloc" (instruction composée dépendant d'une proposition POUR, corps de procédure bloc non étiqueté).

Le stack identificateur comporte 3 colonnes : dans une ligne, les 2 premiers mots donnent l'identificateur source (6 caractères alphanumériques), le 3ème son "profil". Les index STACK et STACK1 réfèrent respectivement le début du niveau ouvert etla première position libre. Lorsqu'on consulte le stack, l'identificateur source est d'abord recherché sur ses 3ers caractères grâce à une instruction machine spéciale de recherche en table, ce qui permetde gagner un peu de temps. Le "profil" relatif à un identificateur est soit :

1) sa représentation dans le langage de codification due à sa déclaration ou spécification.

2) "un profil de reprise" dénotant que l'on a déjà cité l'identificateur dans le programme source sans disposer à cet instant de sa représentation dans le niveau courant du stack. Le profil de reprise donne la dernière adresse de la chaîne indiquant où manque cette représentation en chaîne codée : en effet, on ne peut le rechercher dans les niveaux précédents puisque sa déclaration peut être donnée ultérieurement.

Cette méthode permet de n'utiliser qu'un seul stack identificateur et grâce auxchaînes de reprise, de n'avoir qu'une occurrence dans un niveau de stack pour le même identificateur cité plusieurs fois dans le niveau correspondant du texte source. On pourrait avoir 2 stacks où l'on ouvrirait simultanément des niveaux pour les identificateurs déclarés à ce niveau et ceux qui ne le sont pas ; en outre, en abandonnant les chaînes de reprises, la programmation serait plus simple, mais l'encombrementdes 2 stacks plus important.

12.2.1.1. Contrôle du Stack - Identificateurs - Stack-Début

Le contrôle du stack-Identificateurs, c' est-à-dire des localisations, est lié à l'analyse de la structure de parenthèses des symboles de base DEBUT et FIN, PROCEDURE et ; délimitant une déclaration de procédure. Comme les instructions composées ne donnent pas lieu à une localisation, on utilise une pile spéciale pour cette analyse : le"stack-Début".

Tout symbole DEBUT ou PROCEDURE y est empilé lorsqu'on le détecte dans le texte source et n'est effacé que lorsqu'on atteindra le délimiteur correspondant. Ces opérations contrôlent en particulier les ouvertures de niveau et les regréssions de niveau du stack identificateurs. Chaque délimiteur placé dans le stack-début est représenté (1 mot) par :

a) une marque caractéristique qui permettra de reconnaitre s'il correspondait à une localisation, c'est-à-dire à une ouverture dans le stack identificateur (bloc, faux bloc, procédure) ou non (instruction composée), puis si cette localisation était un niveau de nomenclature (bloc, procédure).

b) une information quantitative : adresse de l'emplacement retenu en chaîne codée pour l'indicateur de réservation (12.1.2. et 12.1.2.3.) ou réservation déjà faite au niveau de déclaration d'une procédure.

Nous allons étudier ces traitements.

12.2.2. Bloc et Instruction composée

On dispose d'un sous programme éliminant les commentaires qui détermine si l'unité syntaxique en séquence du symbole de base actuellement examiné dans le texte source, est un déclarateur ou non. Ce sous programme est utilisé lorsqu'on reconnait un délimiteur DEBUT pour distinguer les blocs

des instructions composées et plus généralement après chaque ;.

Chaque DEBUT détecté est d'abord systématiquement traduit en chaîne codée en le faisant suivre du pseudo-Indicateur d'Instruction composée (12.1.2.).

. Si c'est effectivement une instruction composée, on se contente de stacker la marque correspondante dans le stack-Debut ; par la suite la détection du symbole FIN dans le texte source, provoquant la consultation du sommet du stack-Debut, reconnait cette situation et régresse simplement le sommet du stack. Les délimiteurs DEBUT et FIN n'ont donc joué aucun rôle.

. Au contraire s'il s'agit d'un bloc la marque stackée dans le stack-Début précise l'adresse du pseudo-indicateur de réservation que l'on vient de placer en chaîne codée ; lorsqu'on aura traité la dernière déclaration du bloc, cette adresse permettra de sortir en chaîne codée le véritable indicateur de préservation, soit la valeur atteinte par le compteur de réservation CREB. Simultanément on initialise ce compteur à 5 (pour tenir compte des données de liaisons), on progresse de 1 le numéro niveau de nomenclature courant NB et on ouvre un niveau dans le stack-Identificateurs.
Puis on poursuit le traitement du texte-source.
Le premier symbole FIN détecté, reconnait dans le stack-Début la situation bloc et après régression de ce stack, provoque une régression de niveau dans le stack-identificateur et la regression de 1 du numéro de bloc NB ; on est donc bien revenu au niveau de nomenclature antérieur.

. Dans le cas des faux blocs (instruction composée contrôlée par une proposition POUR, bloc non étiqueté ; ou instruction composée corps de procédure) on veut simplement localiser les identificateurs sans changer de niveau de nomenclature.

Aussi le symbole DEBUT correspond-il au stackage de la marque caractéristique de cette situation dans le stack-Début et à l'ouverture d'un niveau dans le stack identificateurs, sans modification de NB. Le délimiteur FIN provoque le même traitement que pour un bloc, excepté que NB reste inchangé.

12.2.3. Déclarations.

Les déclarations permettent de construire les représentations des identificateurs et de les placer en stack (12.2.4.) ; ces traitements sont assurés par un certain nombre de sous programmes standard. Nous avons vu d'autre part comment elles se traduisent en chaîne codée (12.1.1.).

On veut déterminer quand s'achève la tête de bloc. Aussi utilise - t'on après chaque <;> délimitant une déclaration, le sous programme général déjà cité, qui indique si l'unité syntaxique suivante est un déclarateur ou non. Les déclarations de variables simples ne comprennent que les identificateurs à déclarer et lorsque le traitement correspondant est achevé le <;> terminal est lui-même atteint. Au contraire dans les autres déclarations après le traitement des identificateurs à déclarer, on doit s'intéresser à des expressions (liste de bornes, liste d'aiguillage), ou bien le processus est récursif pour les procédures. On retient alors pour les déclarations de tableau et d'aiguillage,"l'état déclaration" (variable d'état de l'éditeur DECLAR := 1). Par la suite, on sait en détectant un <;> dans le texte source qu'il terminait une déclaration, on supprime donc cet état déclaration (DECLAR := 0) et on fait le traitement habituel : si on suit une déclaration, on s'orientera vers son sous programme particulier ; sinon on a atteint la fin de tête de bloc et après avoir mis en place l'indicateur de réservation en chaîne codée (12.2.2.), on reprendra le traitement normal des instructions et expressions (12.2 et 12.2.4.)

Pour les déclarations de procédure on n'utilise pas la variable d'état déclaration et on reconnait d'une autre manière le <;> qui la termine. (Ceci est lié à la récursivité de sa définition - 12.2.3.1.)

Lorsqu'on a quitté la tête de bloc, on est en état instruction (DECLAR := 0) en reconnaissant un <;>. Le sous programme général de test de l'unité syntaxique suivante, toujours effectué parès un <;>, détecte alors une erreur si on suit une déclaration.

Remarque Déclaration de Tableau

La déclaration de tableau force un aiguillage du programme de traitement des identificateurs dans une expression (12.2.4.) ; il est rétabli systématiquement à chaque <;> délimitant une déclaration. Ceci permet de détecter l'erreur suivante : emploi dans une liste de bornes d'un identificateur déclaré au même niveau ; cette vérification n'a de valeur que si la déclaration de l'identificateur précède lexicographiquement la déclaration de tableau dans le texte source.

Une section de tableau est reconnue lorsque, examinant un délimiteur] dans le programme Algol, le forçage de l'aiguillage du programme traitement des identificateurs caractérise l'état déclaration de tableau et que le décompte des symboles [et], tenu par un compteur, est revenu à 0. On vérifie alors la présence du délimiteur <;>, traduit en chaîne codée comme <;> TABLEAU (12.1.2.2.), avant de reprendre le programme déclaration de tableau.

12.2.3.1. Déclaration de Procédure

Elle introduit un nouveau niveau de nomenclature, alors que l'on n'a pas achevé le traitement de toutes les déclarations de la tête de bloc où elle a lieu. Aussi préserve-t-on la valeur actuelle atteinte par le compteur de réservation CREB en l'empilant avec la marque caractéristique du symbole PROCEDURE dans le stack-Début.

Comme pour un bloc on ouvre un niveau dans le stack-Identificateur; le numéro de bloc est progressé de 1, et le compteur CREB réinitialisé à 5. Lorsque la tête de procédure est traitée (représentation des formels, tête de procédure en chaîne codée selon 12.1.2.3.), l'indicateur que l'on vient de placer en chaîne à l'adresse AD, reflète uniquement les réservations pour les formels.

Lorsque le corps de procédure commence par DEBUT, que ce soit dans le texte source, une instruction composée ou un bloc, on empile l'adresse AD de l'indicateur de réservation précédent accompagné de la marque "faux bloc" dans le stack-Début. Le traitement est en effet celui d'un faux bloc (12.2.2.). Le compteur CREB n'a pas été réinitialisé, et si c'est un bloc-source en atteignant la dernière déclaration, la valeur atteinte par CREB est placée en chaîne codée à l'adresse indiquée au sommet du stack-Début : le processus est le même que pour un bloc ; mais ici, c'est l'indicateur de réservation du niveau procédure qui est modifié pour tenir des réservation éventuelles dues au corps de procédure.

Pour déterminer le <;> délimitant la déclaration de procédure, on teste à chaque <;> d'Instruction si la marque au sommet du stack-Début est PROCEDURE. Dans ce cas, on régresse le dernier niveau du stack identificateur et le numéro de bloc de 1. Puis on restaure le compteur CREB à la valeur qu'il avait déjà atteinte pour le niveau de nomenclature auquel on vient de revenir ; elle est au sommet du stack-Début que l'on régresse. Avant de reprendre le traitement normal après une déclaration (12.2.3.), on traduit le <;> de fin de déclaration de procédure par un symbole spécial en chaîne codée <;FP>. La génération en sera facilitée.

12.2.4. Fonctionnement du Stack-Identificateurs (fig. p. XII-26).

Le traitement des identificateurs est le suivant dans une expression, une instruction, lorsqu'on a consulté le niveau courant :

a) l'identificateur source n'est pas déjà cité : on le place au sommet libre du stack, avec comme profil l'adresse de la représentation 0 qui lui est donnée provisoirement en chaîne codée. On a constitué la tête de la chaîne de reprise. L'index STACK1 progresse d'une ligne.

b) l'identificateur source est déjà cité en stack

. 1) son profil est un profil de reprise : c'est la représentation qu'on lui donne provisoirement en chaîne codée pendant que l'adresse de cette dernière est indiquée comme nouveau profil ; on prolonge ainsi la chaîne de reprise.

.2) son profil est sa représentation : identificateur de quantité déclarée, ou de formel spécifié au niveau courant ; elle est alors écrite en chaîne codée.

Les profils de représentation sont ou supérieurs à 40000_8 (déclaration) ou négatifs (formels 12.1.1.3.). Les profils de reprise sont donc caractérisés par une valeur positive inférieure à 40000_8, permettant d'adresser une chaîne codée de 16 384 mots.

A la regression de niveau, tous les identificateurs source du niveau regréssé sont examinés ; on consulte dans l'ordre le niveau courant. Si l'identificateur possède un profil de représentation, on passe simplement sans rien faire, au suivant : cela traduit l'abandon de la signification particulière de l'identificateur, ou localisation liée à l'ouverture de niveau. Au contraire s'il possède un profil de reprise, on recherche l'identificateur au niveau antérieur. Là soit :

a) l'identificateur source n'est pas cité : il est recopié avec son profil (de reprise) à la suite du niveau précédent : ce dernier progresse donc; la destruction du début du niveau régressé est en effet possible puisque déjà examiné.

Soit :

b) l'identificateur source y est cité et

1) son profil est sa représentation : il faut la mettre en place dans la chaîne codée, aux endroits indiqués par la chaîne de reprise ; l'adresse terminale de cette chaîne est donnée par le profil de reprise au niveau regressé. Le processus de remplacement s'arrête lorsque la tête de reprise (mémoire de valeur 0) est atteinte dans la chaîne codée.

2) son profil est de reprise : il faut constituer en une seule les 2 chaînes de reprise du niveau régréssé et du niveau précédent. La tête de chaîne de reprise en chaîne codée du niveau régressé est déterminée et reçoit le profil de reprise indiqué au niveau antérieur pour l'identificateur-source. Ce dernier profil est remplacé par le profil de reprise donné au niveau régressé.

Le nouveau niveau courant est le niveau antérieur au niveau régressé : l'index STACK est restauré à sa valeur empilée au moment de l'ouverture ; l'index STACK1 tient compte des recopies éventuelles (a) ;

Lorsque l'on crée la représentation d'un identificateur (déclaration ou spécification) il faut aussi consulter le niveau courant du stack.

a) l'identificateur-source est déjà présent : C'est certainement une erreur dans le cas de la construction de la représentation d'un formel : on en fait le constat sur la flexowriter. Pour une déclaration de quantité, c'est aussi une erreur si le profil du stack est une représentation.

Ce doit être un profil de reprise : la représentation que l'on vient de créer est mise en place selon la chaîne de reprise dans la chaîne codée (comme pour b)1°) de la régression de niveau) et devient le nouveau profil dans le stack.

b) l'identificateur-source est absent : il est alors placé ainsi que son profil représentation au sommet du stack identificateur. L'index STACK1 progresse d'une ligne.

Lorsqu'on atteint la fin du programme source, le stack-indicateur est réduit au niveau courant : niveau du programme. Il contiendra, s'il y a eu erreur, les identificateurs non déclarés ou cités en dehors de leur portée. La consultation du stack permet de le détecter et le constat d'erreur comporte l'impression deces identificateurs source, accompagnés du numéro de ligne de leur première occurence dans le texte source. (il est atteint par le biais de la chaîne de reprise).

Remarque. Procédure Standard

Le rapport Algol précise que certains identificateurs doivent être réservés aux procédures standard. On peut effectivement l'imposer, en vérifiant à chaque déclaration de quantité ou spécification de formel, que l'identificateur source correspondant n'est pas réservé à une procédure standard. Cependant la consultation systématique de leur table alourdirait le traitement et n'est pas faite. On a alors le choix entre 2 traitements; la programmation de l'éditeur permet de passer facilement de l'un à l'autre.

La déclaration d'une quantité, la spécification d'un formel, avec un identificateur de procédure standard, retire à cet identificateur la signification procédure standard pour lui accorder celle de la déclaration (spécification) soit dans :

a) la portée normale de la déclaration ou spécification du texte source. Lorsque la fin du programme atteinte, on examine les identificateurs non déclarés (voir précédemment) on déterminera les procédures standard.

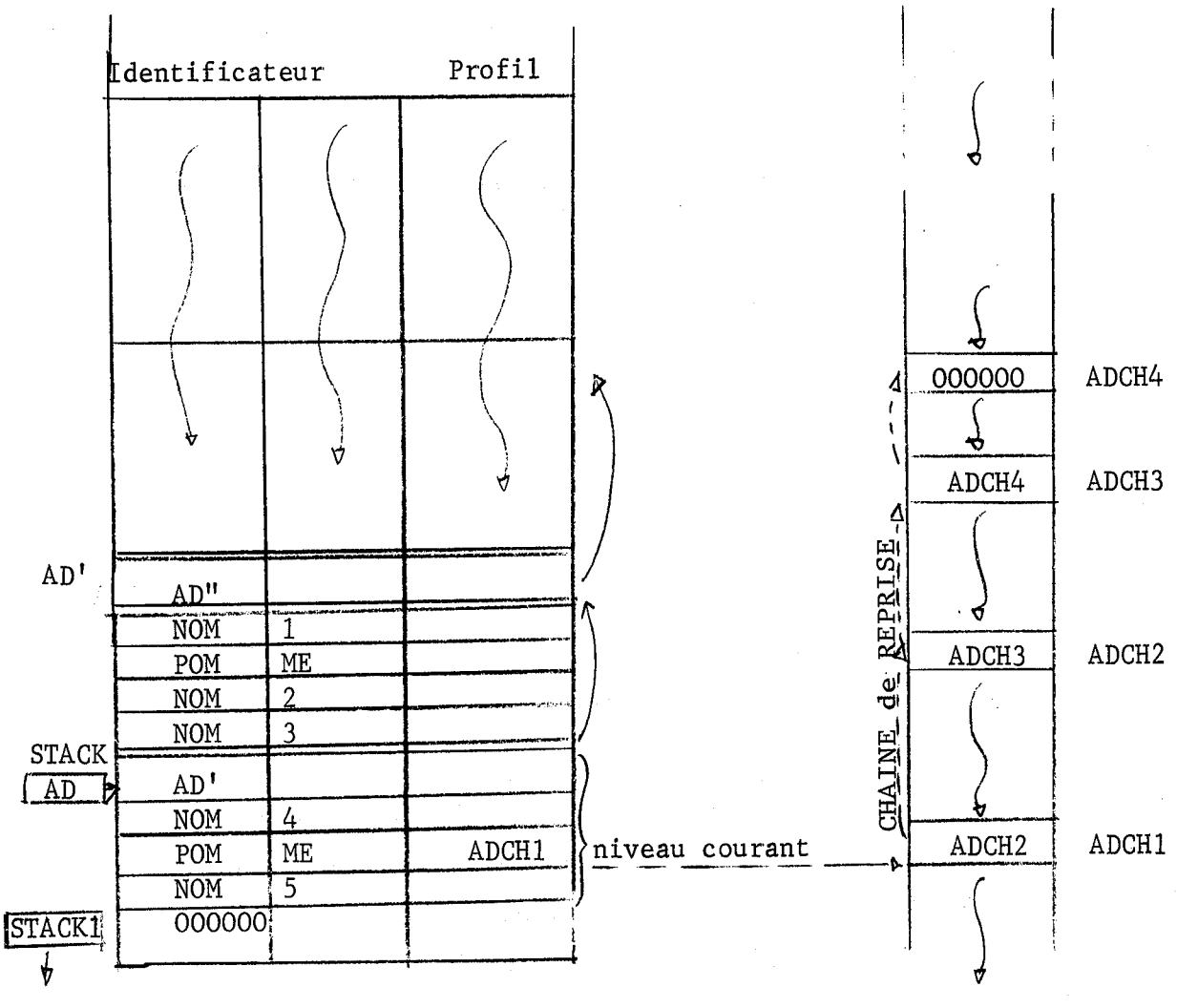
Soit dans :

b) la suite lexicographique du texte source, et uniquement pour le niveau ou localisation propre de la déclaration ou spécification : la signification procédure standard est conservée dans les parties écrites antérieurement à la déclaration dans le texte source. et dans les niveaux (ou localisation) imbriqués postérieurement.

S'il veut s'éviter l'observation attentive de la règle, le programmeur ne tentera pas d'utiliser les identificateurs de procédures standard dans une signification différente.

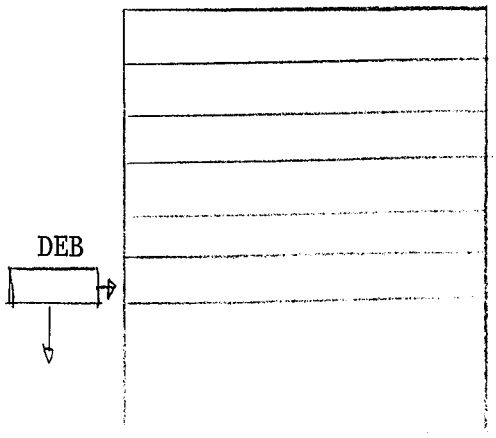
On a adopté la méthode b), car elle conduit à ne jamais encombrer le stack avec des identificateurs de procédures standard. Le traitement déjà décrit des identificateurs dans les expressions et instructions s'applique lorsqu'il n'a pas encore été cité au niveau courant (absence du stack), sous réserve qu'il ne figure pas dans la table PS des procédures standard. De même format que le stack identificateur elle s'utilise de manière analogue. Si on reconnait en la consultant pour un identificateur absent du stack identificateur, un identificateur de procédure standard, le profil trouvé en table est placé comme représentation provisoire en chaîne codée ; l'adresse de cette écriture devient le nouveau profil en table PS. Comme au départ de la codification la colonne des profils est initialisée à 0, on constitue ainsi une chaîne de reprise classique pour la procédure standard considérée. Parvenu à la fin du texte source, on examine cette table PS : un profil nul dénote que la procédure n'est pas citée ; dans le cas contraire on attribue la première mémoire libre de la table-procédure-standard de la chaîne codée à la procédure standard citée (12.1.1.3.).

Son adresse est alors mise en place, grâce à la chaîne de reprise, comme représentation en chaîne codée, puis comme nouveau profil en table PS (il sera utilisé dans le chargement des procédures). Une 4ème colonne adjointe à la table PS contient l'encombrement et l'information catégorie type de chaque procédure : on calcule l'adresse de chargement de la procédure qui, accompagnée de l'information catégorie, est affectée comme valeur de la mémoire correspondante dans la table-procédure-standard de la chaîne codée.



STACK IDENTIFICATEUR

CHAINE CODEE



STACK-DEBUT et Marques dans ce STACK

Valeur de : octale	d_{18} d_{17} d_{16}	d_{15} ..... d_1
PROCEDURE	4	Valeur CREB au moment de la déclaration de procédure
BLOC	2	Adresse chaîne codée de 1'Indicateur de réservation
FAUX-Bloc 1) (*) 2)	1	Adresse en chaine codée de 1'indicateur de réservation du niveau PROCEDURE --- 0 0 0 0 0
INSTRUCTION COMPOSEE	3	0 0 0 0 0

(*) 1) Corps de procédure dans le texte source (cf. 12.2.3.1.)

2) Instruction composée contrôlée par proposition POUR

C H A P I T R E XIII

PRINCIPES - PROGRAMME GENERATEUR

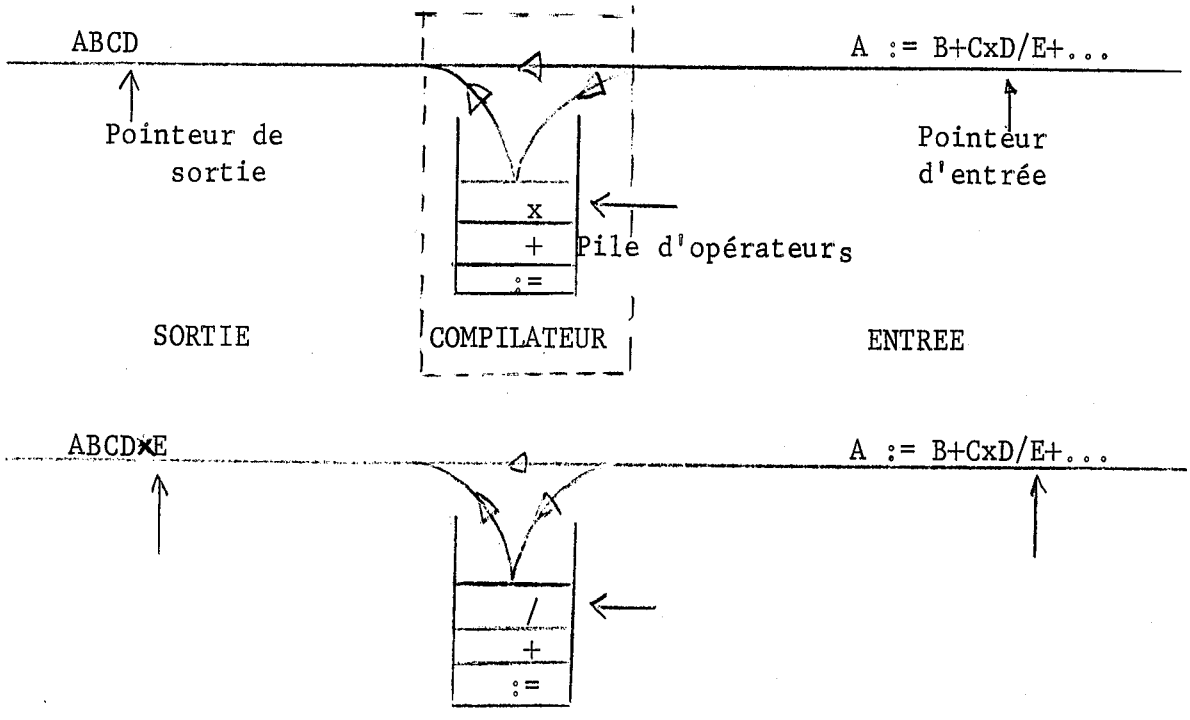
La génération du programme objet est pratiquement limitée à l'analyse syntaxique de la chaîne codée produite par l'édition : la structure du programme objet reflète la notation postfixée. Dijkstra a donné l'image suivante de la compilation conduisant à cette production :

Le texte source constitue une file d'entrée examinée séquentiellement par le générateur qui transfère immédiatement tout identificateur sur la file de sortie : le programme objet. Il n'y a pas ici de pile d'identificateurs, mais on retrouve l'équivalent à l'exécution comme on l'a vu.

Seuls les opérateurs donnent lieu à un traitement : ils transitent par une pile appelée stack opérateur qui permet l'analyse selon la méthode de priorité bien connue que nous considérons sous la forme suivante : lorsqu'on examine un opérateur en chaîne codée sa priorité est comparée à celle de l'opérateur au sommet de la pile,

1°) Si la priorité de l'opérateur en chaîne est supérieure à la priorité de l'opérateur en stack, l'opérande droite de ce dernier n'a pas été généré : aussi l'opérateur en chaîne est empilé et l'on progresse dans l'examen de la chaîne codée.

2°) Dans le cas contraire où la priorité de l'opérateur en chaîne est inférieure ou égale à celle de l'opérateur au sommet du stack, l'opérande droite de ce dernier a été traité ; aussi l'opérateur en stack est désempilé et transféré dans la file de sortie : il est généré. Du fait de cette génération on recommencera le test de priorité entre l'opérateur toujours examiné sur la file d'entrée et l'opérateur qui se trouve maintenant au sommet de la pile ; le résultat du test impliquera alors le traitement 1 ou 2.



On remarque que cette méthode est conçue pour traiter les expressions ; les niveaux de priorités des opérateurs arithmétiques, de relations, et logiques, sont donnés dans le rapport Algol [1]. Mais de plus en choisissant une priorité convenable des délimiteurs := et ALLERA on peut traiter les instructions d'affectation et ALLERA comme les expressions : Ces 2 symboles doivent avoir une priorité inférieure à celle des opérateurs d'expression considérés plus haut mais supérieure à celle du délimiteur fin d'instruction qui est aussi un délimiteur fin d'expression. On aborde là le traitement de la structure parenthésée, bien connu dans les expressions ; on accorde aux parenthèses une priorité inférieure à celle des opérateurs, la priorité de la parenthèse gauche étant par ailleurs inférieure à celle de la parenthèse droite. Cette méthode va de pair :

a) avec la reconnaissance dans l'examen séquentiel du texte source de la parenthèse gauche pour l'empiler immédiatement et progresser dans l'examen du texte sans faire de test de priorité.

b) la reconnaissance lorsqu'on va empiler un délimiteur que c'est une parenthèse droite : on doit vérifier dans ce cas que l'on a bien une parenthèse gauche au sommet de la pile : le traitement se limite à la désempiler et passer à l'examen de l'unité syntaxique suivante en considérant que l'on vient de générer un opérande.

On peut considérer un autre algorithme de la méthode de priorité qui en adoptant les mêmes niveaux de priorités, affranchira du point a) dans le traitement de la structure parenthésée : tout opérateur et délimiteur autre que) est immédiatement empilé. C'est lorsque l'on vient de traiter un opérande que l'on fait le test de priorité entre le délimiteur au sommet de la pile opérateur et le délimiteur qui suit l'unité syntaxique actuellement examinée dans le texte source : l'opérateur pile est généré si sa priorité est supérieure ou égale à celle de l'opérateur chaîne, et dans ce cas le test de priorité est renouvelé. On progresse dans l'examen du texte lorsqu'il y a arrêt de la génération au moment où l'on examine un opérande. Dans la 1ère méthode que nous avons choisie la génération a lieu en examinant un opérateur dans la file d'entrée d'où la nécessité du point a). Le problème des (et) d'expressions se généralise, aux autres délimiteurs parenthèsant les expressions et les instructions et l'analyse du texte source peut être entièrement fondée sur la priorité des délimiteurs. On choisit alors la 2de méthode de priorité (compilateur ALGOLIBM 7040 de J.C. Boussard [6])

Ici l'analyse est conduite de façon différente et la méthode de priorité n'est en fait utilisée que pour les expressions simples non parenthésées, en ce sens que ce mécanisme ne joue que pour analyser l'expression entre parenthèses ; la prise en compte de la parenthèse gauche et son empilage découlent de l'analyse de l'expression selon les règles syntaxiques de [1] ; en atteignant la parenthèse droite (alors que l'expression a été générée c'est-à-dire que l'on reconnait vouloir empiler une parenthèse droite) la parenthèse gauche empilée

qui caractérise l'appel du programme 'Expression Simple Non Parenthésée" provoque le retour à ce programme. Il faut de façon générale faire une vérification avant de poursuivre l'analyse : en effet on doit considérer plusieurs délimiteurs de type parenthèse droite comme les opérateurs séquentiels ALORS, SINON... les séparateurs PAS, JUSQUA.... les crochets], FIN. Ils parenthèsent les expressions, simultanément les instructions pour certains d'entre eux ; on leur accorde une priorité analogue à) d'expression pour qu'ils aient le même rôle que ce dernier ; le délimiteur de type parenthèse gauche correspondant a donc aussi une priorité analogue à (expression. La méthode est identique avec les délimiteurs parenthèsant les Instructions.

On dispose alors d'un Programme INSTRUCTION et d'un programme EXPRESSION se subdivisant eux mêmes en plusieurs programmes : les différents programmes d'Instructions peuvent s'appeler et appeler Expression ; les programmes d'expressions se limitent à appeler Expression. C'est la description des règles syntaxiques de [1] d'où la récursivité des appels.

Supposons que nous ayons 2 procédures EXPRESSION SIMPLE et EXPRESSION et que par ailleurs TYPE soit le type de l'expression analysée par un programme d'expression.

S[I] désignera l'unité syntaxique courante examinée en chaîne codée et pointée par l'index I. Le mécanisme de priorité des parenthèses droites implique que l'on termine l'activation d'un programme en reconnaissant que l'on veut empiler un délimiteur de catégorie parenthèse droite.

Le générateur est alors l'appel de la Procédure PROGRAMME et nous indiquerons sa structure par le principe de quelques procédures utilisées.

```
PROCEDURE ETIQUETTE ; E : SI S[I+1] = <:> ALORS
          DEBUT TRAITER ETIQUETTE (S[I]); I := I+2 ;
                ALLERA E
          FIN ;
PROCEDURE INSTRUCTION ;
   DEBUT ETIQUETTE ; SI S[I] = <SI> ALORS INSTRUCTION CONDITIONNELLE
        SINON SI S[I] = <POUR> ALORS INSTRUCTION POUR
              SINON INSTRUCTION INCONDITIONNELLE
   FIN ;
PROCEDURE INSTRUCTION CONDITIONNELLE ;
   DEBUT ETIQUETTE ; EMPILER (<SI>) ; I := I+1 ;
   EXPRESSION ; SI TYPE ≠ <TYPE BOOLEEN> V S[I] ≠ <ALORS>
   ALORS ERREUR ( ) ; EMPILER (<ALORS>) ; I := I+1 ;
   SI S[I] = <POUR> ALORS INSTRUCTION POUR
      SINON SI S[I] = <SI> ALORS ERREUR ( )
            SINON DEBUT INSTRUCTION ; SI S[I] = <SINON>
            ALORS DEBUT EMPILER (<SINON>); I := I+1 ;
                        INSTRUCTION ;
                  FIN
                  FIN ;
   FIN ;
```

On remarquera que le programme à appeler est déterminé par sa première unité syntaxique qui est l'unité couramment examinée en chaîne codée, mais qu'il est parfois nécessaire d'examiner l'unité syntaxique suivante : ceci se produit dans les instructions d'affectations par exemple (de même l'étiquetage d'une instruction). L'empilage des symboles de catégorie parenthèse gauche correspond on l'a déjà vu à l'analyse et génération des expressions par la priorité. Mais il est en outre nécessaire pour réaliser la récursivité d'appel des programmes ; on pourrait empiler une adresse de retour au programme et un délimiteur parenthèse gauche unique.

En fait on empile un délimiteur parenthèse gauche caractéristique du programme appelant et de son point d'appel : ce délimiteur est au besoin crée spécialement. Ceci joue le même rôle qu'une adresse de retour et permet dans la majorité des cas de faire la vérification de construction syntaxique, c'est-à-dire la validité du délimiteur parenthèse droite atteint, au moment de la détermination du point de retour, et non dans le programme qui a appelé et auquel on va revenir. Le générateur diffère par là, de l'exemple d'écriture précédemment donné.

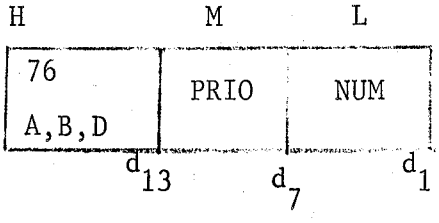
A cet effet on distingue des familles de délimiteurs, qui correspondent aux délimiteurs utilisés pour une même instruction. Par exemple la famille 5 sera constituée des délimiteurs PAS, JUSQUA,....FAIRE examiné en chaîne codée et jouant le rôle de parenthèse droite, et des délimiteurs parenthèses gauches créés spécialement pour caractériser un point où le programme INSTRUCTIONPOUR appelle par exemple le programme EXPRESSION : ainsi POUR-PAS, POUR-PAS-JUSQUA... Si en fin d'activation d'EXPRESSION on reconnait qu'il a été appelé par le Programme de la famille 5 et que la parenthèse droite atteinte n'appartient pas à la famille 5 il y a erreur.

Ce traitement ne peut être tout à fait systématique : il faut distinguer le délimiteur "," qui n'est pas spécifique d'une famille mais est utilisé dans toutes les listes. Aussi reconnait on d'abord la "," pour revenir selon la famille au programme qui a appelé, ou faire un traitement général ; c'est le cas pour le traitement des listes d'indices et de bornes : il existe autant de délimiteurs [que de programmes appelant "liste d'indices" et ils appartiennent naturellement à la même famille.

On verra de façon complète dans l'écriture en Algol du générateur comment ceci est réalisé. L'annexe de ce chapitre donne en particulier tous les symboles parenthèses gauches spéciaux et descriptifs de la syntaxe empilés par le générateur.

13.1 Expression et type.

Pour mettre en oeuvre le traitement indiqué le code d'un délimiteur comporte trois parties :



H qui vaut normalement 76 permet comme on le verra plus loin, d'incorporer le type à gauche d'un délimiteur.
L : la Priorité PRIO
M : le numéro NUM

. Pour un opérateur arithmétique, logique ou de relation, NUM est la valeur du code de l'ordre objet sans adresse correspondant à l'opération, la génération en est simplifiée.

. Pour un délimiteur parenthèse, NUM se divise en N (d_4 - d_5 - d_6) : numéro de famille, et K (d_1 - d_2 - d_3) ; numéro d'identification dans cette famille.

Le type de l'expression actuellement générée est donné par la variable XTYP du générateur. Pour un opérateur examiné en chaîne codée et que l'on empile, il correspond au type à gauche de l'opérateur et l'on vérifie la validité de la construction : on connait le type impliqué par l'opérateur. De même en désempilant un opérateur, le type désigné par XTYP est le type à droite de l'opérateur que l'on génère, et doit correspondre au type impliqué par celui-ci ; le type de l'expression générée demeure inchangé sauf si l'opérateur généré est une relation : on affecte alors la valeur TYPE BOOLEEN à XTYP. D'autre part chaque fois que l'on traite un primaire, XTYP reçoit le type de ce primaire. Pour un nombre pur ou une variable simple cela provient simplement de l'examen de l'identificateur en chaîne codée. Pour une variable indicée ou un indicateur de fonction on voit qu'il est nécessaire d'empiler le type de l'identificateur

correspondant, au moment où on l'examine en chaîne codée : la récursivité impose de le préserver de cette manière, avant d'appeler un programme de "liste d'expressions". Pour plus d'efficacité le type est en fait incorporé dans lapartie H du code du délimiteur parenthèse gauche caractéristique du point d'appel au programme que l'on va activer immédiatement : ce traitement est facile, car l'examen d'un identificateur en chaîne codée affecte systématiquement son type à XTYP. Au retour du programme ainsi activé, la valeur de type incorporée au délimiteur parenthèse gauche vient restaurer XTYP.
Nous avons noté l'incorporation possible d'une valeur de type, dans la partie H du code de symbole d'un délimiteur empilé, sous la forme 76_A, 76_B, 76_D, selon que le type incorporé est arithmétique, booléen ou désignation.

Il existe une autre raison d'incorporer le type de l'expression précédemment générée, en empilant un délimiteur parenthèse gauche : la vérification de l'homogénéîté de l'expression SI B ALORS E1 SINON E2. En empilant le délimiteur SI-ALORS-SINON avant l'appel d'EXPRESSION, on lui incorpore la valeur de XTYP ; au retour on vérifiera que XTYP, ou type à droite du SINON, est identique à la valeur de type incorporé au symbole SI-ALORS-SINON. Le mécanisme est identique pour les instructions d'affectation et ALLERA en empilant les délimiteurs := et ALLER avec le type courant (forcé à désignation pour ALLER A).

Ainsi on connait toujours le type de l'expression générée à un instant donné, d'où les vérifications de construction et la possibilité de donner une caractérisation des paramètres effectifs expression.

13.2.1. Pile Auxiliaire DEBR

On empile dans la pile auxiliaire DEBR l'adresse de l'indicateur de réservation que l'on vient de générer dans le programme objet, chaque fois que l'on pénètre dans un bloc ou procédure. Ceci permettra de réserver la mémoire locale attachée à chaque instruction POUR, réservation qui n'est pas faite à l'Edition. Contrôlée par l'index XDEBR la longueur de cette pile est limitée à 16 mots : l'édition impose de ne pas avoir plus de 16 niveaux imbriqués. En traitant une instruction POUR, XDEBR permet de connaître la première mémoire locale libre du niveau, que l'on attribue à cette instruction, et de progresser en conséquence l'indicateur de réservation.

13.3. Le programme générateur en Algol.

L'écriture reflète fidèlement le générateur écrit en langage machine (cf. [7]). Comme pour l'Interpréteur écrit en Algol, nous espérons faciliter la compréhension du programme générateur CAE 510 et montrer la valeur générale de l'algorithme.

Nous avons pas inclus, le traitement de l'instruction POUR, qui n'aurait pas donné d'indications supplémentaires sur la méthode.

PROGRAMME GENERATEUR :

DEBUT ENTIER STAIN, STAMAX, XCPMAX, TYPA, TYPB,
TYPD ; XCPMAX := .... ; STAIN := ... ; STAMAX := ... ;
TYPA := H200000 ; TYPB := H100000 ; TYPD := H000000 ;

DEBUT ENTIER TABLEAU S [STAIN : STAMAX],
DEBR [1 : 16], P[128 : XCPMAX] ;

COMMENTAIRE Le tableau S est le stack principal : double pile du stack et du"stack haut".Le tableau DEBR est la pile des adresses d'Indicateurs de réservation, dite stack auxiliaire.

La chaîne codée est la tableau CHAINE commun à l'éditeur et au générateur.

Le Programme objet sera le tableau P, la valeur affectée à XCPMAX étant normalement l'adresse future de chargement des procédures standard calculée en fin d'Edition.

Les conventions utilisées dans l'écriture de l'interpréteur restent valables. En outre, la valeur ou code des symbôles de base a été désignée symboliquement entre < > en indiquant soit le symbole lui-même si la notation est claire, soit en le faisant précéder de SB ou SYMB suivant que le délimiteur est le même qu'en chaîne codée ou est un délimiteur particulier utilisé à la génération et de code différent. Exemple < := >, <SB DEBUT>, <SYMB DEBUT-I>.

La valeur des codes générés est donnée de façon analogue, exemple : <CODE ØRTN>

TYPA, TYPB, TYPD sont les valeurs qui seront affectées à XTYP pour indiquer respectivement le Type Arithmétique, Booléen et Désignation.

Les déclarations des entiers et Booléens locaux du générateur ne sont pas écrites : les variables de contrôle de la génération sont indiquées, avec leur rôle, page suivante "hors programme".

De même les déclarations des aiguillages utilisés par les programmes particuliers ne sont pas écrites : ils seront donnés de manière analogue en cours de programme ;

ENTIER I,J,K,KEXP,N,XSYMB, SB,A,A7,XADP,CTAB,CTAB1,CTAB2,XPR1,XPR2,T1,
NUMCHAINE,PROCHAINE,NUMSTACK,PRIOSTACK ;
BOOLEEN SIAUTORISE, CLEAIG, CLE SI INSTRUCTION, CLE INSTRUCTION, PROCEDURE,
CLE IDEN FIN PE ;

HORS PROGRAMME

COMPTEURS et INDEX du générateur

XCP Compteur programme : 1ère mémoire libre du programme généré.
ICC Index I de Chaîne Codée : pointe l'unité syntaxique (u.s.) courante
JCC Index J " " : pourra pointer l'u.s. successeur de l'u.s. courante
XSTA : Sommet (occupé) du Stack opérateur
XDB Index du Stack "stack haut" { sommet (occupé) du "stack haut"
XDA niveau courant
XNL Compteur de lignes : dernier numéro de ligne détecté en Chaîne Codée
XDEBR Index du Stack auxiliaire DEBR

Origines et compteurs de Tables

ICREP
ICRET
ICRAIG
} mémorisation des origines d'implantation des tables { procédure / Etiquette / Aiguillage

XCREP
XCRET
XCRAIG
} Index de la première mémoire libre dans ces tables {Remplissage dans l'ordre des déclarations qui est celui de la numérotation} { procédure / Etiquette / Aiguillage

Variables particulières du générateur

A1 Numéro de catégorie de l'Unité Syntaxique examinée en chaîne codée

XTYP Type courant

XQAL
XOP
XENC
XPF
} pour l'unité syntaxique courante, pourront contenir : { partie qualitative / quantitive (adresse *) / encombrement * / manque paramètre formel *

{* : pour un identificateur}

HORS PROGRAMME

Mémoires de Liaison Editeur-générateur

CRAIG, CRET, CREP } Compteurs de l'Editeur { aiguillage, etiquette, procédure

permettent au générateur d'implanter les tables procédure, étiquette, aiguillage, ils définissent les longueurs des tables. Ces compteurs deviennent inutiles après les initialisations.

DTPS, REEL } Compteurs provenant de la codification et donnant respectivement l'origine et la Fin de la table des procédures standard, et utilisés en fin de génération.

FPS provient aussi de l'Editeur et fournit la dernière adresse qui sera occupée par les procédures standard.

```
PROCEDURE STACKS (CODE SYMBOLE) ; ENTIER CODE SYMBOLE ;
        DEBUT XSTA := XSTA+1 ; S[XSTA] := CODE SYMBOLE ;
        SI XSTA ⩾ XDB ALORS ERREUR (40) ;
        FIN Empilage simple dans le stack opérateur ;

PROCEDURE STACKT (CODE SYMBOLE) ; ENTIER CODE SYMBOLE ;
        STACKS (CODE SYMBOLE ∧ COMPLETEMENT LOGIQUE (XTYP)) ;
    COMMENTAIRE Empilage du symbole avec incorporation du type dans les
    digits de fort poids du code. Ceux ci valent 1 dans le code ordinaire :
    l'incorporation du type sous forme complémentée correspond à une ins-
    truction machine, la même instruction étant utilisée pour l'extraire
    lorsque nécessaire. Ceci est possible grâce aux valeurs prises le
    TYPE (voir remarques "hors programme" sur la procédure TYPE) ;

PROCEDURE SPAUXI (CARACTERISATION, SORTIE) ; ETIQUETTE SORTIE ;
    ENTIER CARACTERISATION ; DEBUT COMMENTAIRE Cette procédure empile dans
    le stack haut une caractérisation statique de paramètre effectif ou
    une adresse d'entrée de sous-programme de liste d'aiguillage ;
      XDB := XDB-1 ; SI XDB ≤ XSTA ALORS ERREUR (40) ;
      S[XDB] := CARACTERISATION ; ALLERA SORTIE
      FIN ;

PROCEDURE CONSULTER SYMBOLE D'INSTRUCTION (CLE, ERREUR) ;
        BOOLEEN CLE ; ETIQUETTE ERREUR ;
    DEBUT COMMENTAIRE Cette procédure est utilisée lorsqu'une instruction
    commence par un symbole de base, et pour vérification après une
    instruction procédure, ce que CLE permet de distinguer. Le tableau
    SYMBOLE D'INSTRUCTION est donné page hors programme.
```

```
POUR I := SI CLE ≡ VRAI ALORS (SI SIAUTORISE ≡ VRAI ALORS 1 SINON 2)
          SINON 8 PAS 1 JUSQUA 11 FAIRE
          DEBUT SB := I ; SI CHAINE [ICC] = SYMBOLE D'INSTRUCTION [1]
          ALORS ALLERA SORTIE
          FIN ; ALLERA ERREUR ;
       SORTIE : FIN de CONSULTER

PROCEDURE LOGAP (INDEX TABLE) ; ENTIER INDEX TABLE ;
          DEBUT COMMENTAIRE Cette procédure est utilisée pour constituer et
          placer dans les tables étiquette, aiguillage, procédure, au moment
          de sa déclaration, la valeur correspondante à l'identificateur.
          INDEX TABLE est un compteur indiquant la première position libre
          dans la table correspondante soit XCRET, XCRAIG, XCREP (les iden-
          tificateurs ont été numérotés dans l'ordre des déclarations par la
          codification) ;
          A := (XOP ∧ H017000) x 2↑5 ; TABLE [INDEXTABLE] := A ∨ XCP ;
          INDEXTABLE := INDEXTABLE+1 ;
          FIN La valeur placée en table est le numéro de bloc de l'identificateur
              accompagnée de l'adresse programme objet du prochain ordre qui
              sera généré ;

PROCEDURE GENERER (CODE) ; ENTIER CODE ;
DEBUT P[XCP] := CODE ; XCP := XCP+1 ;
SI XCP ≥ XCP MAX ALORS ERREUR (9)
FIN GENERER ;
```

```
PROCEDURE GSTA (ORIGINE TABLE) ; ENTIER ORIGINE TABLE ;
    DEBUT COMMENTAIRE Génération des ordres sur les étiquettes et aiguillage,
    'ETF Adresse Dynamique' pour un formel, ou 'STA Adresse Table', ORIGINE
    TABLE est le compteur origine de la table correspondante soit ICRET soit
    ICRAIG ;
    SI XPF = 0 ALORS DEBUT A := ORIGINETABLE + (XOP ∧ H000777) ;
    GENERER (H600000 ∨ A) FIN génération de l'ordre STA
    SINON GENERER (H220000 ∨ XOP) ;
    FIN GSTA ;

PROCEDURE INSR DEBUT XDEBR := XDEBR+1 ; DEBR [XDEBR] := XCP-1
            FIN de l'empilage dans le stack DEBR de l'adresse de l'indicateur
            de réservation qui vient d'être généré ;

PROCEDURE PROG DEBUT COMMENTAIRE Cette procédure progresse le pointeur ICC
        d'unité syntaxique courante examinée en chaîne codée. Il génère les
        numéros de ligne et met à jour le compteur de ligne XNL ;
        E : ICC := ICC+1 ; SI (CHAINE [ICC] ∧ H760000) = H020000 ALORS
        DEBUT XNL := CHAINE [ICC] ∧ H017777 ;
              GENERER (H640000 ∧ XNL) ; ALLERA E
        FIN H640000 est le code de l'ordre *L* ;
        FIN PROG ;

PROCEDURE PROLIR DEBUT COMMENTAIRE Cette procédure effectue le même travail
    que PROG, mais affecte à XQUAL et XOP respectivement la partie qualitative
    et la partie adresse de l'unité syntaxique courante de chaîne. Elle progresse
    en outre l'index JCC sur l'unité suivante ;
    PROG ; XQUAL := CHAINE [ICC] ∧ H760000 ;
    XOP := CHAINE [ICC] ∧ H017777 ; JCC := ICC+1 ;
E : SI (CHAINE [JCC] ∧ H760000) = H020000 ALORS DEBUT
    JCC := JCC+1 ; ALLERA E FIN ;
     FIN PROLIR ;
```

PROCEDURE TYPE DEBUT COMMENTAIRE Cette procédure permet de séparer les informations contenues dans l'unité syntaxique (u.s.) courante pointée par ICC en chaîne, et prédécodées par la procédure PROLIR : XQAL contient la partie qualitative à laquelle on s'intéresse.

TYPE affecte à : A1 le numéro de catégorie de l'unité syntaxique
XTYP le type Arithmétique, Booléen, Désignation de l'u. s. (éventuellement).
XENC "l'encombrement de valeur" s'il s'agit de l'identificateur d'une quantité pourvant être associée à une valeur.
XPF la marque paramètre formel ou non pour un identificateur.

XPF et XENC facilitent la génération des ordres, et la valeur de XENC est adaptée à ce travail. Les effets de PROLIR sont indiqués en annexe : le corps ne sera pas écrit ;

FIN TYPE ;

PROCEDURE SAVAR ;

DEBUT COMMENTAIRE Génération des ordres sur adresses de variables ou des ordres formels correspondants ARN, AEN, ARF, AEF ;
GENERER ($H100000_V$ XPF $_V$ XENC $_V$ XOP)
FIN ;

PROCEDURE CVVAR ;

DEBUT COMMENTAIRE Génération des ordres sur valeur : CRN, CEN, CRF, CEF ;
GENERER ($H020000_V$ XPF $_V$ XENC $_V$ XOP)
FIN ;

PROCEDURE CVVALS ;

DEBUT COMMENTAIRE Génération des ordres sur valeurs de nombres purs CRS et CES ; GENERER ($H520000_V$ XENC $_V$ XOP)
FIN ;

PROCEDURE APPZ ;

DEBUT COMMENTAIRE Génération de l'activation d'une procédure sans paramètre, A7 est le code APN ou AFN, XOP l'adresse table ou dynamique pour un formel ; GENERER (A7 $\vee$ XPF) ; GENERER (XOP)
FIN ;

PROCEDURE ADPXOP ;

DEBUT COMMENTAIRE Pour une procédure non formel, ADPXOP remplace la partie adresse de l'identificateur de procédure contenue dans XOP par l'adresse correspondante en Table procédure ;
SI XPF = 0 ALORS XOP := ICREP + XOP $\wedge$ H000777 ;
FIN ;

PROCEDURE GENERER FIN SSPPE ;

DEBUT COMMENTAIRE Génération de l'ordre de retour d'un sous programme de calcul de paramètre effectif, et lorsqu'il s'agit d'un sous programme expression constitution la partie qualitative de la caractérisation statique empilée dans le stack haut : elle est réduite à l'adresse d'entrée, on indiquera le type de l'expression ;
GENERER (<CODE Ø RTN>) ;
SI S[XDEB] > 0 ALORS S[XDEB] := (XTYP*2) $\vee$ H620000 $\vee$ S[XDEB]
FIN ;

PROCEDURE VAFSTA ; COMMENTAIRE XADP donne l'adresse de l'identificateur de procédure en cours de compilation : on impose que l'affectation de valeur à un identificateur de procédure ait lieu dans le corps de procédure lui même, et non dans le corps d'une procédure déclarée dans le corps considéré (restriction) ;

SI P[XADP] ≠ CHAINE [ICC] ALORS ERREUR ( ) ;
A := (CHAINE [ICC] $\wedge$ HO17000) x 2↑ (-9) + 1
STACKS (A $\vee$ XENC) ;
FIN on a empilé le second mot de l'ordre d'affectation à générer : Numéro de bloc de la fonction et indication de type de la valeur, donnée ici par XENC ;

```
PROCEDURE TRAPRO ;
    DEBUT XADP := XCP ; GENERER (CHAINE[ICC])
    FIN ;
```

```
PROCEDURE VERITY (OPE) ; ENTIER OPE ;
DEBUT COMMENTAIRE Cette procédure utilisée dans EXPRESSION, vérifie que le type
à gauche d'un opérateur que 1'on empile, ou le type à droite d'un opérateur que
1'on génère, est correct.
OPE est la priorité de cet opérateur ;
SI OPE < 10 ALORS DEBUT SI XTYP ≠ TYPB ALORS ERREUR (32)
                  FIN
            SINON SI XTYP ≠ TYPA ALORS ERREUR (32)
FIN ;
```

INITIALISATIONS : XCREP := ICREP := REEL ; XCRET := ICRET := ICREP + CREP ;
XCRAIG := ICRAIG := ICRET + CRET ; XCP := XLANCP := ICRAIG + CRAIG ;
COMMENTAIRE Implantation optimisée des tables non examinée. cf. NOTE :
GENERATEUR [7] ;
XSTA := STAIN ; XDB := XDA := STAMAX ; XDEBR := PSTA ;
ICC := CHAINI-1 ; COMMENTAIRE STAIN , STAMAX , PSTA sont des constantes paramètres du générateur de même CHAINI pour l'Editeur et le générateur (origine chaine codée). Il faut initialiser une clé utilisée dans le traitement de liste de paramètres effectifs ;
CLE IDEN FINPE := FAUX ;

ENTREE GENERATION : PROLIR ; SI CHAINE [ICC] ≠ <SB DEBUT>
ALORS ERREUR (49) ; SI CHAINE [JCC] = HO17777
ALORS CHAINE [JCC] := 5 ; ALLERA ZDEB ;
COMMENTAIRE Lorsque le programme est une instruction composée la codification (cf. ch. 12) n'a pas indiqué les réservations d'un bloc et sans variable locale. On appelle le programme INSTRUCTION ZDEB : Bloc/Instruction Composée.
La génération ou appel d'INSTRUCTION s'exécute ;

SORTIE GENERATION : WSORT : GENERER (<CODE Ø STP>) ;
FIN ET ENCHAINEMENT : COMMENTAIRE Cette séquence de programme effectue
1°) l'effacement des 3 digits de type dans les tables de procédures standard, tables provenant de la codification.

2°) l'appel et le chargement en mémoire des programmes de chargement des procédures standard, de listing et de perforation du programme objet.

3°) le chargement des procédures standard utiles (cf. note GENERATEUR) et l'impression de "génération terminée". Puis les différentes sorties possibles selon positionnement des clés par l'opérateur.

4°) Le chargement éventuel de l'interpréteur et lancement de l'exécution.

Rappelons que l'on a donc l'ordre sur le ruban compilateur : l'Editeur, le Générateur, les programmes mentionnés au 3°), la bibliothèque puis l'Interpréteur ;

HORS PROGRAMME

BRANCHEMENTS DE AIGINSTRUCTION

		COMMENTAIRE
	A1	Unité Syntaxique en chaine codée
ER2	0	Nombre Pur
AFQS1	1	Variable Simple
AFVI1	2	Tableau
AFFN1	3	Procédure Fonction déclarée
INSPS	4	Fonction Standard
INSPS	5	Procédure Standard
INSPN	6	Procédure déclarée
ZETI	7	Etiquette
ER2	8	Aiguillage
ER2	9	Chaine
SYMBOLE DE BASE	10	Symbole de Base

Identificateur de...

TABLEAU SYMBOLE D'INSTRUCTION	BRANCHEMENTS CORRESPONDANT PAR TABLE INSTRUCTION
SI	ZSI
POUR	ZPOUR
DEBUT	ZDEB
ALLERA	ZALLER
PROCEDURE	ZPROCD
TABLEAU	ZTAB
AIGUILLAGE	ZAIG
;	ZVIDE
FIN	ZVIDE
SINON	ZVIDE
;FP	ZVIDE

INSTRUCTION :

INSE : PROLIR ; TYPE ; ALLERA AIGINSTRUCTION [A1+1]

COMMENTAIRE Le numéro de catégorie A1 de l'unité syntaxique courante examinée en chaîne codée (index ICC) permet de s'orienter vers le programme d'INSTRUCTION correspondant. Ces appels sont donnés sour forme de table page précédente ;

SYMBOLE DE BASE : CONSULTER SYMBOLE D'INSTRUCTION (VRAI, ER1) ;
ALLERA TABLE INSTRUCTION [SB] ;

ER1 : ERREUR (1) ;...; COMMENTAIRE Les branchements vers les programmes d'INSTRUCTION sont donnés sous forme de Tableau page précédente ;

INS : SIAUTORISE := VRAI ; ALLERA INSE ;

INSNC : SIAUTORISE := FAUX ; ALLERA INSE ; COMMENTAIRE Cet appel d'INSTRUCTION n'est utilisé qu'après une proposition SI ;

ZDEB : PROG ; SI CHAINE [ICC] = HO17777 ALORS

INSTRUCTION COMPOSEE : STACKS (<SYMB DEBUT-I>)

SINON

INSTRUCTION BLOC : DEBUT GENERER (<CODE Ø EBL>) ;
GENERER (CHAINE [ICC]) ;
STACKS (<SB DEBUT>) ; INSR ; COMMENTAIRE
L'adresse de l'indicateur de réservation qui vient d'être généré est empilée dans le stack DEBR : ceci est nécessaire pour la réservation de la mémoire locale attribuée à une instruction POUR ;
FIN ;
ALLERA INS ;

COMMENTAIRE appel d'Instruction. Le délimiteur caractéristique DEBUT-I ou DEBUT permet de reconnaître le point de retour WSBL ou WSBI du programme appelant en atteignant le FIN de <compound tail> en chaîne codée ;

FIN INSTRUCTION BLOC : WSBL : GENERER (<CODE Ø SBL>) ; XDEBR := XDEBR-1 ;

FIN INSTRUCTION COMPOSEE ET BLOC : WSBI : XSTA := XSTA-1 ;

SI XSTA := STAIN ALORS ALLERA SORTIE GENERATION ;

PROG ; ALLERA STAGEN ; COMMENTAIRE on vient de générer une instruction et le présent appel de ce programme est terminé (progression de l'examen en chaîne codée et effacement du délimiteur caractéristique du programme de la pile). STAGEN permettra, par la méthode générale, la poursuite de l'analyse du texte source : le délimiteur maintenant au sommet du stack détermine le point de retour au programme d'Instruction qui a appelé. Des vérifications de constructions sont faites avant ce retour, sur l'unité syntaxique examinée en chaîne codée ;

ZTAB : DECLARATION DE TABLEAU : PROG ; CTAB := CTAB2 := 0 ;

CTAB1 := CHAINE [ICC] ;

NBRETABLEAUX : POUR I := 1 TANT QUE CHAINE [ICC] ≠ < [>

FAIRE DEBUT PROG ; CTAB2 := CTAB2+1 FIN ;

STACKS (<SYMB [DT>) ; ALLERA ENTEX1 ;

COMMENTAIRE on appelle le programme EXPRESSION pour traiter la liste de bornes. Chaque délimiteur ',' (les ':' sont devenus des ',' à l'édition) fait progresser le compteur CTAB : voir VIRCRO.

En fait on a ici l'appel du programme LISTE D'INDICES (ou de bornes). L'entrée ne diffère des variables indicées que par suite de la position de l'INDEX ICC. Le Retour a lieu sur CRODT en atteignant le délimiteur] en chaîne codée. Voir FLIND: ICC est déjà progressé sur l'unité syntaxique suivante et le stack regressé ;

```
CRODT : SI CHAINE [ICC] ≠ <;> ALORS ERREUR (25) ;
        CTAB := CTAB+1 ; SI IMPAIR (CTAB) ALORS ERREUR (26) ;
        GENERER (<CODE Ø TAB>) ; P[XCP] := CTAB ;
        P[XCP+1] := CTAB1 ; P[XCP+2] := CTAB2 ;
        XCP := XCP+3 ; COMMENTAIRE Génération de l'ordre de création de tableau
      et de ses opérandes ;

ZAIG : COMMENTAIRE DECLARATION D'AIGUILLAGE ;
       STACKS (<SB AIGUILLAGE>) ; PROLIR ;
       XDA := XDB := XDB-1 ; SI XDB < XSTA ALORS ERREUR (40) ;
       COMMENTAIRE Préparation de l'empilage dans le stack "haut" des adresses
       de sous programmes d'évaluation des expressions de désignation, éléments
       de la liste d'aiguillage ;
       XPR1 := XCP ; XCP := XCP+1 ; S[XDB] := XCP ;
       XPR2 := XOP ALLERA ENTEX2 ; COMMENTAIRE XPR2 et XPR2 retiennent respective-
       ment l'adresse de la mémoire réservée dans le programme objet pour l'ins-
       truction de saut de la déclaration, et la représentation de l'identifica-
       teur de l'aiguillage déclaré. On a aussi empilé l'adresse d'entrée du
       premier sous programme d'évaluation et on va à EXPRESSION pour traiter
       la liste d'aiguillage ;

VIRAIG : ; COMMENTAIRE Retour d'EXPRESSION avec le délimiteur <,> en chaîne
           codée. Une vérification est nécessaire ;
           SI S[XSTA] ≠ <SB AIGUILLAGE> ALORS ERREUR (22) ;
           CLE AIG := VRAI ; ALLERA SSPAIG ;
```

```
WP AIG : ; COMMENTAIRE Retour d'EXPRESSION avec le <;> de fin de déclaration
         d'aiguillage en chaîne codée. SSPAIG vérifie le type de l'expression
         élément de liste, génère l'ordre de retour du sous programme ;
            CLEAIG := FAUX ;
SSPAIG : SI XTYP ≠ TYPD ALORS ERREUR (37) ; GENERER (<CODE Ø RTN>) ;
         SI CLEAIG = FAUX ALORS ALLERA FIN AIG ;
SPAUXI (XCP,ENTEX1) ; COMMENTAIRE Bouclage sur EXPRESSION pour continuer le trai-
         tement de la liste d'aiguillage après avoir empilé l'adresse d'entrée
         du prochain sous programme objet ;
FIN AIG : T1 := 0 ; POUR I := 1 TANT QUE XDA ≥ XDB FAIRE
        DEBUT P[XCP] := S[XDB] ; XCP := XCP+1 ; XDB := XDB+1 ;
              T1 := T1+1 ;
        FIN du Transfert dans le programme objet des adresses d'entrée des
        sous programmes Expression de la liste d'aiguillage. On va générer le
        nombre d'éléments de la liste en séquence, l'instruction de saut déjà
        prévue après la déclaration, et entrer l'adresse de l'aiguillage (du
        nombre d'éléments) accompagnée de son numéro de bloc en Table aiguillage ;
        P[XCP] := T1 ; P[XPR1] := H560000 V (XCP+1) ;
        XOP := XPR2 ; LOGAP (XCRAIG) ; XCP := XCP+1 ;  XDA := XDB ;
        XSTA := XSTA-1 ; ALLERA INS ;

ZPROCD : DECLARATION DE PROCEDURE :
         STACKS (XADP) ; STACKS (<SB PROCEDURE>) ; PROLIR ;
         TRAPRO ; LOGAP (XCREP) ; PROG ;
       COMMENTAIRE La procédure TRAPRO met provisoirement l'identificateur de
procédure couramment compilée, à la place de l'instruction de SAUT après la
déclaration de procédure (ordre qui ne peut être généré que plus tard) et
conserve dans XADP l'adresse correspondante de programme objet : la procédure
VAFSTA peut alors vérifier facilement la légitimité de l'affectation à un
identificateur de procédure. Aussi faut-il commencer par préserver la valeur
précédente de XADP en l'empilant. ;
```

TRANSFERT DANS LE PROGRAMME OBJET DE LA TETE DE PROCEDURE :

E : GENERER (CHAINE [ICC]) ; PROG ; SI CHAINE [ICC] ≠ <;>
ALORS ALLERA E ;
INSR ; ALLERA INS
COMMENTAIRE Comme dans un bloc on empile l'adresse de l'indicateur généré dans le stack DEBR et on appelle INSTRUCTION pour compiler le corps de procédure. Le Retour a lieu sur WPFP en atteignant le délimiteur <;FP> en chaîne codée ;

WPVFP : GENERER (<CODE Ø SBL>) ;
P[XADP] := H56000 $_{V}$ XCP ; XSTA := XSTA-2 ;
XADP := S[XSTA+1] ; XDEBR := XDEBR-1 ;
ALLERA INS ; COMMENTAIRE On a généré l'ordre de saut après la déclaration de procédure, et en régressant le stack restauré XADP à sa précédente valeur ;

ZVIDE : ALLERA STAGEN ;
COMMENTAIRE Instruction vide. L'analyse du texte source se poursuit par la méthode habituelle ;

ZETI : ; COMMENTAIRE INSTRUCTION ETIQUETTE ;
SI CHAINE [JCC] ≠ <:> ALORS ERREUR (20) ;
LOGAP (XCRET) ; PROG ; ALLERA INSE ;
COMMENTAIRE l'appel d'INSTRUCTION conserve l'interdiction ou l'autorisation d'instruction SI données précédemment ;

ZALLER : ; COMMENTAIRE INSTRUCTION ALLERA ;
XTYP := TYPD ; STACKT (<SB ALLERA>) ; ALLERA ENTEX1 ;
COMMENTAIRE On appelle EXPRESSION pour évaluer l'expression de désignation après avoir empilé, avec incorporation du type forcé à désignation, le symbole ALLERA . Le traitement de fin de cette instruction se fait au niveau du programme EXPRESSION comme pour les instructions d'affectation. VOIR GENERATION INSTRUCTION 2 ;

COMMENTAIRE INSTRUCTION D'AFFECTATION

Premier niveau : il faut vérifier que l'on a bien une affectation, ou distinguer l'instruction procédure pour un identificateur de fonction. Second niveau : on déterminera si l'on a une affectation multiple. Ceci ne peut se faire qu'après traitement de la liste d'indices pour une variable indicée ;

AFQS1 : VARIABLE SIMPLE : SI CHAINE [JCC] ≠ < := > ALORS ERREUR (5) ;
SAVAR ; COMMENTAIRE Génération de l'ordre Prendre Adresse ;
STACKT (<SB := >) ; ALLERA SUITAF ;

AFVI1 : VARIABLE INDICEE : STACKT (<SYMB [VI2>) ;
ADVI : SAVAR ;
ADVI1 : SI CHAINE [JCC] ≠ <[> ALORS ERREUR (27) ;
ALLERA ENTEX2 ; COMMENTAIRE appel à EXPRESSION pour traiter la liste d'indices voir VIRCRO et FLIND. On revient après avoir régressé le stack après [, et progressé en chaîne après] ;

CRVI2 : ; COMMENTAIRE Retour d'EXPRESSION pour AFVI1 ;
SI CHAINE [ICC] ≠ < := > ALORS ERREUR (5) ;
STACKT (<SB := >) ;
ZUITAF : GENERER (<CODE Ø IND>) ; ALLERA SECOND NIVEAU ;

AFFN1 : IDENTIFICATEUR DE FONCTION : SI CHAINE [JCC] ≠ < := > ALORS ALLERA INSPN ;
VAFSTA ; COMMENTAIRE Vérifier la légitimité de l'affectation de valeur à un identificateur de fonction, empiler dans le stack opérateur son type et numéro de bloc : opérande de l'ordre d'affectation qui sera généré ; STACKT (<SYMB := P>) ;

SUITAF : PROG ;

SECOND NIVEAU : PROLIR ; TYPE ; ALLERA TABLE AFFEC [A1+1] ;

COMMENTAIRE Selon le numéro de catégorie A1 de l'unité syntaxique après le ' := ' on s'oriente vers un constat d'erreur, la détermination de l'affectation multiple ou l'appel d'EXPRESSION (VALO, VAL1, ENTEXO sont des entrées de ce programme comme plus loin CRVIO, VAL3N).

La fin de génération des instructions d'affectations (après génération de l'expression) se fait au niveau du programme EXPRESSION : voir page 46 GENERATION INSTRUCTION2.

TABLE AFFEC	A1
VALO	0
VARIABLE SIMPLE2	1
VARIABLE INDICEE2	2
IDEN DE FONCTION2	3
VAL3	4
ER3	5
ER3	6
ER4	7
ER4	8
ER4	9
ENTEXO	10

;

ER3 : ERREUR (3) ;.....;

VARIABLE SIMPLE2 : SI CHAINE [JCC] ≠ < := > ALORS ALLERA VAL1 ;

SAVAR ; STACKT (<SYMB := M>) ; ALLERA SUITAF ;

COMMENTAIRE Retour sur le second niveau après avoir empilé l'opérateur d'affectation multiple ;

```
VARIABLE INDICEE2 : STACKT (<SYMB [VI2M>) ; ALLERA ADVI ;
   COMMENTAIRE EXPRESSION sera appelé pour le traitement de la liste d'indices
   et le retour se fera ici en CRVI2M ;
CRVI2M : SI CHAINE [ICC] ≠ < := > ALORS ALLERA CRVIO ;
         STACKT (<SYMB := M>) ; ALLERA ZUITAF ;

IDENTIFICATEUR DE FONCTION2 : SI CHAINE [JCC] ≠ <:=> ALORS ALLERA VAL3N ;
VAFSTA ; STACKT (<SYMB := PM>) ; ALLERA SUITAF ;
```

```
ZSI : ; COMMENTAIRE INSTRUCTION SI ;
        STACKS (<SB SI>) ; ALLERA ENTEX1 ;
        COMMENTAIRE Appel d'EXPRESSION. Au Retour le programme sera aussi
        utilisé en grande partie aussi par les expressions SI - ALORS - SINON ;

WSII : ; COMMENTAIRE Retour d'EXPRESSION, on a atteint ALORS en chaîne codée ;
         CLE SI INSTRUCTION  := VRAI ; XSYMB := <SYMB SI ALORS I> ;

  COMMUN ALORS : SI XTYP ≠ TYPB ALORS ERREUR (36) ;
      S[XSTA] := XCP ; XSTA := XSTA+1 ; P[XCP] := H620000 ;
    COMMENTAIRE On génère un ordre TST sans partie adresse, et l'adresse de
    cet ordre dans le programme objet est empilée. On empile ensuite le déli-
    miteur SIALORS ;
      S[XSTA] := XSYMB ; XSTA := XSTA+1 ; XCP  := XCP+1 ;
    ALLERA SI CLE SI INSTRUCTION ≡ VRAI ALORS INSNC SINON SECBOO ;
    COMMENTAIRE Appel de INSTRUCTION(instruction SI interdite) ou d'EXPRESSION
    selon la clé. La fin de génération de l'instruction et alors le retour
    au programme appelant, ou le retour à WSIAI se fait au niveau du programme
    EXPRESSION (voir GENERATION INSTRUCTION) ;
```

```
WSIAI : ; COMMENTAIRE Retour d'INSTRUCTION, on a atteint le délimiteur correspon-
        dant SINON en chaîne codée ;
        CLE SI INSTRUCTION := VRAI ; XSYMB := <SYMB SI ALORS SINON I > ;
        COMMUN SINON : P[XCP] := H560000 ;
           P[S[XSTA-1]] := P[S[XSTA-1]] V (XCP+1) ;
  COMMENTAIRE On génère un ordre de SAUT inconditionnel et on complète la
  partie adresse de l'ordre TST généré à l'examen du ALORS. L'adresse de l'ordre
  SAU et le symbole SI ALORS SINON sont empilés dans le stack opérateur ;
      S[XTA-1] := XCP ; S[XSTA] := XSYMB ; XCP := XCP+1 ;
    ALLERA SI CLE SI INSTRUCTION ≡ VRAI ALORS INS SINON ENTEX1 ;
    COMMENTAIRE Appel de INSTRUCTION (instruction SI autorisée) ou d EXPRESSION
    selon la clé. La fin de génération de l'instruction, et le retour au programme
    appelant se fait au niveau d'EXPRESSION (voir GENERATION INSTRUCTION) ;

INSPN : ; COMMENTAIRE INSTRUCTION PROCEDURE.
        INSPN entrée procédure "déclarée" : on calculera et affectera à XOP
        l'adresse table correspondant à l'identificateur de procédure si ce
        n'est pas un formel. Pour l'entrée INSPS des procédures standard elle
        est déjà dans XOP ; ADPXOP ;

INSPS : A7 := H660000 ; COMMENTAIRE Retenir dans A7 l'ordre APN** ;
        SI CHAINE [JCC] = < ( > ALORS ALLERA LISTE PARAMETRES ;
        APPZ ; COMMENTAIRE Génération Appel de procédure sans paramètre ;

LISTPZ1 : PROG ; CONSULTER TABLE INSTRUCTION (FAUX, ER7) ;
          ALLERA STAGEN ; COMMENTAIRE L'appel de procédure sans paramètre ou
          avec paramètres (voir LISTPE) a été généré : on vérifie que l'unité
          syntaxique suivant la ) en chaîne codée est SINON, FIN, ';' ou ';FP'
          avant de poursuivre l'analyse du texte source en déterminant le pro-
          gramme qui a appelé grâce à STAGEN ;
          ER7 : ERREUR (7) ;
```

LISTE PARAMETRES : STACKS (XCP) ; STACKS (<SYMB (IP>) ;

COMMENTAIRE On utilise LISTPE pour traiter la liste de paramètres effectifs, programme appelé aussi par INDICATEUR DE FONCTION. Ici on a empilé le symbole caractéristique de parenthèse Instruction Procédure ;

LISTPE :

OUVPE : ; COMMENTAIRE Les caractérisations statiques vont être constituées et empilées dans le stack "haut' il faut y ouvrir un niveau (repéré par XDA), par suite de la récursivité de LISTPE. De même, il faut préserver l'adresse table de la procédure traitée, nécessaire pour générer ultérieurement l'appel de procédure (l'ordre lui-même est empilé, sans partie adresse, dans le stack haut). L'adresse table est mémorisée jusque là, à l'emplacement que l'on réserve dans le programme objet pour placer l'ordre de SAUT à l'activation, cet ordre de saut étant naturellement généré au même moment ;

S[XDB-1] := XDA ; XDA := XDB := XDB-2 ;

SI XDB ≤ XSTA ALORS ERREUR (40) ;

S[XDB] := A7 + XPF ; GENERER (XOP) ;

CLE INSTRUCTION PROCEDURE := SI A7 = H660000 ALORS VRAI SINON FAUX ;

LISTP1 : PROG ;

LISTPO : ; COMMENTAIRE ICC pointera en chaîne codée l'unité syntaxique après la (;

PROLIR ;

SINAM : SI XQAL ≠ H760000 ALORS ALLERA SINAMB ;

SINAM1 : PARAMETRE EFFECTIF COMMENCANT PAR SYMBOLE DE BASE :

SI CHAINE [ICC] = <SB '> ALORS ALLERA CHAINE ;

PARAMETRE EXPRESSION : SPAUXI (XCP, ENTEXO) ;

COMMENTAIRE on empile l'adresse d'entrée du sous programme évaluation d'expression dans le stack haut et on appelle EXPRESSION ;

SINAMB : PARAMETRE EFFECTIF COMMENCANT PAR UN IDENTIFICATEUR :
SI CHAINE [JCC] = <,> ALORS ALLERA SINAMC ;
SI CHAINE [JCC] ≠ <)> ALORS ALLERA SINAM9 ;
PROG ; CLE IDENFINPE := VRAI ; ALLERA SINAMC ;

SINAM9 : ; COMMENTAIRE On détermine si l'on a une variable indicée. Dans ce cas il faut attendre la fin du traitement des indices pour reconnaître si le paramètre effectif est une variable ou une expression ;
TYPE ; SI A1 ≠ 2 ALORS SPAUXI (XCP, ENTEX3) ;
COMMENTAIRE PARAMETRE EXPRESSION ;

INAM10 : PARAMETRE EFFECTIF COMMENCANT PAR UNE VARIABLE INDICEE ;
STACKT (<SYMB [PE>) ; SPAUXI (XQUAL $_V$ XCP $_V$ H400000, ADVI) ;
COMMENTAIRE Dans le stack haut on empile la caractérisation statique (présumée) de sous programme d'indice. On appelle LISTE D'INDICES soit ADVI, après avoir empilé dans le stack opérateur le symbole [de paramètre effectif. Il assurera le retour sur CROPE après traitement de la liste ;

CROPE : SI CHAINE [ICC] = <)> $_V$ CHAINE [ICC] = <,>
ALORS DEBUT GENERER (<CODE Ø IND>) ;
ALLERA STAGEN ;
FIN le paramètre effectif était une variable Indicée.
on retourne à STAGEN pour déterminer si la liste de paramètres est terminée et quel est alors le programme appelant LISTPE. On peut s'orienter vers ZPARP, ZPARFI, VIRPAR ;

CROPE1 : S[XDB] := S[XDB]$_{\Lambda}$H17777 ; ALLERA CRVIO ;

COMMENTAIRE le paramètre effectif est une expression commençant par une variable indicée : on corrige la caractérisation statique en sous programme d'expression et on appelle EXPRESSION ;

CHAINE : ; COMMENTAIRE PARAMETRE EFFECTIF CHAINE

On se ramenera au traitement d'un paramètre effectif identificateur. La chaîne est transférée dans le programme mot à mot telle qu'elle a été constituée par l'éditeur. KCC retient pendant cette phase l'adresse objet de la chaîne ;

KCC := XCP ; T1 := 0 ;

TRANSFERT : POUR I := 1 TANT QUE T1 ≥ 0 FAIRE

DEBUT : GENERER (CHAINE [ICC]) ; PROLIR ; SI CHAINE [ICC] = <SB /> ALORS T1 := T1+1 ; SI CHAINE [ICC] = <SB \> ALORS T1 := T1-1

FIN ; GENERER (CHAINE [ICC]) ;

; XQUAL := HO20000 ; XOP := KCC ; COMMENTAIRE Après avoir transféré le dernier symbole \ on considère un identificateur fictif chaîne d'où les affectations de valeurs à XQAL et XOP ;

ALLERA SINAMB ;

SINAMC : ; COMMENTAIRE PARAMETRE EFFECTIF IDENTIFICATEUR

On construit dans A la partie qualitative de la caractérisation statique (c'est généralement XQAL directement) puis on lui adjoint la partie adresse XOP ou l'adresse table obtenue à partir de XOP ;

SI XQUAL < 0 ALORS DEBUT

A := SI XQUAL = H600000 ALORS H500000 SINON

SI XQUAL = H400000 ALORS H440000 SINON H400000 ;

COMMENTAIRE On traite ici les paramètres effectifs : Nombre Pur Entier, Nombre Pur Réel, paramètre formel (dans l'ordre de l'expression) ;
ALLERA SINAM6 ; FIN ;
SI XQAL = 0 ALORS DEBUT
A := COMPLEMENT LOGIQUE (CHAINE [XOP])$_{\Lambda}$H700000) x 2↑(-2) + H200000 ;
COMMENTAIRE Identificateur de procédure standard. On extrait les informations : procédure, fonction réelle, fonction entière, fonction booléenne, contenues sous forme complémentée dans la table des procédures standard. Maintenant A est identique à XQUAL pour une procédure de catégorie correspondante déclarée dans le programme ;
ALLERA SINAM6 FIN ;

SI XQUAL > H200000 ALORS ALLERA QUANTITE AVEC TABLE ;
A := XQAL ; COMMENTAIRE paramètre effectif variable simple ou tableau ;

SINAM6 : A := A+XOP ;
SINAM7 : SPAUXI (A, SI CLE IDEN FINPE ALORS LISTP2 SINON LISTP1) ;
COMMENTAIRE Après empilage de la caractérisation statique, on va à la fin du traitement de la liste, ou l'on poursuit l'examen de celle ci ;
QUANTITE AVEC TABLE : ; COMMENTAIRE Procédure et Fonction, Etiquette, Aiguillage. La table TBPEF donne les adresses des Index détenant l'origine des tables pour chaque catégorie de quantité ; l'adresse (ou le nom) du compteur est notée P, sa valeur M[P] ;
P := TBPEF [XQUAL × 2↑(-13) -9] ;
A := M[P] + XOP $_{\Lambda}$ H777 ;
A := XQUAL $_{V}$ A ; ALLERA SINAM7 ;

COMMENTAIRE FIN DE GENERATION D'UN SOUS PROGRAMME DE PARAMETRE EFFECTIF - FIN DE TRAITEMENT D'UNE LISTE DE PARAMETRES ET RETOUR AU PROGRAMME APPELANT. L'expression ou la variable indicée paramètre peuvent être délimitées par <,> en chaîne codée : EXPRESSION provoque le retour sur VIRPAR. Si c'est le dernier paramètre de la liste (délimiteur <)> en chaîne codée), le retour a lieu sur ZPARP ou ZPARFI selon le programme appelant : instruction procédure ou évaluation d'un indicateur de fonction.;

ZPARP : CLE INSTRUCTION PROCEDURE := VRAI ; GENERER FIN SSPPE ;
ALLERA LISTP2 ; COMMENTAIRE il faut rétablir la clé de sortie qui a pu être modifiée par suite de la récursivité de LISTPE. La procédure GENERER FINSSPPE, génère l'ordre de retour Ø RTN et complète la caractérisation statique d'un paramètre effectif expression par incorporation du type de l'expression ;

ZPARFI : CLE INSTRUCTION PROCEDURE := FAUX ; GENERER FIN SSPPE ;
XTYP := COMPLEMENT LOGIQUE (P [XSTA])$_\Lambda$H300000 ; ALLER LISTP2 ;
COMMENTAIRE Le type de l'expression (indicateur de fonction) qui va être généré est affecté à XTYP. Il avait été incorporé au symbole (FI empilé en stack ;

VIRPAR : SI A2=1 ALORS DEBUT CLE INSTRUCTION PROCEDURE := VRAI ;
GENERER FIN SSPPE ; ALLERA LISTPO
FIN paramètre effectif dans liste d'une instruction procédure. On continue le traitement de la liste en examinant le paramètre suivant ;

SI A2 =2 ALORS DEBUT CLE INSTRUCTION PROCEDURE := FAUX ;
GENERER FIN SSPPE ; ALLERA LISTPO
FIN paramètre effectif dans une liste d'indicateur de fonction ;

ERREUR (24) ; COMMENTAIRE <,> dans une expression entre parenthèses, ou dans un indicateur d'aiguillage ;

LISTP2 : FINPE : CLE IDEN FINPE := FAUX ; T1 := 0 ;

GENERATION CARACTERISATIONS STATIQUES :
POUR I := 1 TANT QUE XDB ≠ XDA FAIRE
DEBUT P[XCP] := S[XDB] ; XCP := XCP+1 ; XDB := XDB+1 ; T1 := T1+1
FIN Désempilage des T1 caractérisations statiques constituées dans le stack haut qui sont sorties dans le programme objet ;

XSTA := XSTA-2 ; T3 := P[S[XSTA+1]] ;
P[S[XSTA+1]] := H560000 $_{V}$ XCP ; COMMENTAIRE L'adresse table de la procédure détenue dans le programme objet est retenue dans T3, et on génère à cet endroit le SAUT à l'ordre d'appel de la procédure ; P[XCP] := S[XDB]$_{V}$T1 ; P[XCP+1] := T3 ; XCP := XCP+2 ; XDA := S[XDB+1] ; XDB := XDB+2 ; COMMENTAIRE Génération de l'ordre d'appel XDB était redevenu sur XDA et pointait l'ordre d'appel empilé sans partie adresse. On restaure le précédent niveau du stack haut ;
ALLERA SI CLE INSTRUCTION PROCEDURE ≡ VRAI ALORS LISTPZ1
SINON LISTPX1 ;

COMMENTAIRE FIN DU TRAITEMENT DE LISTE DE PARAMETRES ET GENERATION DE L'ACTIVATION DE PROCEDURE ;

COMMENTAIRE PROGRAMME EXPRESSION

Selon le numéro de catégorie de l'unité syntaxique examinée en chaîne codée, que la procédure TYPE donne dans A1, le programme s'oriente vers une séquence particulière de traitement, et pour un symbole de base vers une séquence qui dépend du programme expression appelant : ENTEX, SECBOO, EXPAS, TERM ;

```
ENTEX2 : PROG ;
ENTEX1 : KEXP := 1 ;
NOYAU EXP : PROLIR ; TYPE ; ALLERA AIGEXPRESSION [A1+1]
            SYMBOLE EXP : ALLERA AIGSBEXP [KEXP] ;
            COMMENTAIRE AIGEXPRESSION envoie à SYMBOLE EXP lorsque l'expression
            commence par un symbole de base et AIGSBEXP détermine la séquence
            à exécuter selon le programme EXPRESSION appelé et caractérisé par
            KEXP. Ces aiguillages sont donnés page 41 ;

ENTEXO : SI CHAINE [ICC] = <SB SI> ALORS
         DEBUT STACKS (<SYMB SI-E>) ; ALLERA ENTEX1
         FIN PROPOSITION SI ;

        ESNON : SI CHAINE [ICC] = <SB ¬ > ALORS
         DEBUT STACKS (<SB ¬ >) ; ALLERA EXPAS
         FIN SECONDAIRE BOOLEEN ;

       ESMUNI : SI CHAINE [ICC] = <SB-u> ALORS
        DEBUT STACKS (<SB-u>) ; ALLERA TERM
        FIN Expression simple commençant par l'opérateur 'unitaire' ;

       ESPAR : SI CHAINE [ICC] = <SB ( > ALORS
        DEBUT STACKS (<SB (>) ; ALLERA ENTEX1
        FIN ;
       ERREUR (10) ;
```

SECBOO : SECONDAIRE BOOLEEN : KEXP := 2 ; ALLERA NOYAU EXP ;

EXPAS : EXPRESSION SIMPLE : KEXP := 3 ; ALLERA NOYAU EXP ;

TERM : TERME ARITHMETIQUE OU BOOLEEN :
KEXP := 4 ; ALLERA NOYAU EXP ;

HORS PROGRAMME

BRANCHEMENTS PAR AIGEXPRESSION	A1
VALO	0
VAL1	1
VAL2	2
VAL3	3
VAL3N	4
ER11	5
ER12	6
VAL7	7
VAL8	8
ER13	9
SYMBOLE EXP	10

BRANCHEMENTS PAR AIG SB EXP	KEXP
ENTEXO	1
ESNON	2
ESMUNI	3
ESPAR	4

RETOUR EN FIN D'EXPRESSION PAR AIGVIRGULE	K : famille délimiteur en stack
VIRAIG	0
VIRPAR	1
VIRCRO	2
ER23	3
NON EXISTANT	4
VIRPOU	5

TB REDUCTION	K
1	0
3	1
5	2
5	3
*	4
1	5

Ce tableau est utilisé par FERMA

COMMENTAIRE TRAITEMENT PRIMAIRE ;

VALO : CVALS ; COMMENTAIRE NOMBRE pur : génération des ordres CRS, CES ;
ALLERA OPERATEUR ;

VAL1 : CVVAR ; COMMENTAIRE VARIABLE SIMPLE ou PARAMETRE FORMEL
SPECIFIE TYPE : génération des ordres CRN, CEN, CRF, CEF ;
OPERATEUR : PROG ; ALLERA STAGEN ;

VAL2 : STACKT (<SYMB [VIO>) ; ALLERA ADVI ; COMMENTAIRE
VARIABLE INDICEE. On va générer l'ordre "Prendre Adresse" et traiter la liste d'indices. En fin de ce traitement le symbole particulier [VIO empilé dans le stack provoque le retour en CRVIO ;

CRVIO : GENERER (<CODE Ø PVA>) ; ALLERA STAGEN ;

VAL3N : ; COMMENTAIRE INDICATEUR DE FONCTION. Le traitement est analogue à celui des instructions procédure : il utilise le même programme LISTPE. Le retour a lieu en LISTPX1 grâce au délimiteur caractéristique (FI que l'on empile ici, dans le stack, en lui incorporant le type de la fonction. VAL3N correspond aux fonctions déclarées, VAL3 aux fonctions standard ;
ADPXOP ;

VAL3 : A7 := H700000 ; COMMENTAIRE Code de l'ordre AFN ** ;
SI CHAINE [JCC] = <SB (>
ALORS DEBUT STACKS (XCP) ; STACKT (<SYMB (FI>) ; ALLERA LISTPE
FIN on va traiter la liste de paramètres et générer l'ordre d'activation. Retour en LISTPX1 ;
SINON APPZ ; COMMENTAIRE générer activation de procédure sans paramètre ;
LISTPX1 : PROG ; ALLERA STAGEN ;

```
VAL7 : ; COMMENTAIRE ETIQUETTE. La procédure GSTA génère les ordres STA
       Adresse Table ou ETF Adresse Dynamique (de paramètre formel) ;
       GSTA (ICRET) ; PROG ; ALLERA STAGEN ;

ER11 : ERREUR (11) ;....;ER13 : ERREUR (13) ;

VAL8 : ; COMMENTAIRE INDICATEUR D'AIGUILLAGE.
       L'indice est traité comme liste d'indices d'une variable indicée.
       Un seul indice est autorisé, ce que le programme liste d'indices
       reconnait. Le délimiteur "," est interdit lorsque le traitement
       se rapporte à un indicateur d'aiguillage ;
       GSTA (ICRAIG) ; STACKT (<SYMB [AIG>) ; ALLERA ADVI1 ;
       COMMENTAIRE Retour a lieu sur CROAIG grâce au symbole caractéristique
       [AIG empilé en stack ;
       CROAIG : GENERER (<CODE Ø AIG>) ; ALLERA STAGEN ;

SUITEX : XSTA := XSTA-1 ; PROG ;
STAGEN : ; COMMENTAIRE TRAITEMENT D'UN OPERATEUR EXAMINE ENCHAINE CODEE.
         GENERATION DES ORDRES CORRESPONDANT PAR METHODE DE PRIORITE DES
         OPERATEURS ET DELIMITEURS. RETOUR AU PROGRAMME APPELANT CARACTERISE
         PAR UN DELIMITEUR (parenthèse d'Instruction et d'expression)
         APRES VERIFICATION DE LA SYNTAXE ; XQUAL := CHAINE [ICC] ∧ H760000 ;
         T1 := CHAINE [ICC] ; SI XQUAL ≠ H7600000 ALORS ERREUR (30) ;
         COMMENTAIRE L'unité syntaxique examinée en chaîne codée doit effecti-
         vement être un symbole de base. NUMCHAINE désigne son numéro et
         PRIOCHAINE sa priorité : ce ne doit pas être un délimiteur de caté-
         gorie ouverture de parenthèse ;
         PRIOCHAINE := (T1 ∧ H0077000) x2↑(-6) ; NUMCHAINE := T1 ∧ H000077 ;
         SI PRIOCHAINE = 0 ALORS ERREUR (35) ;
```

```
TESTP : MECANISME DE GENERATION PAR PRIORITE ET PILE D'OPERATEURS :
        PRIOSTACK := (S[XSTA]∧H007700)x2↑(-6) ; NUMSTACK := S[XSTA]∧H000077 ;
        COMMENTAIRE PRIOSTACK et NUMSTACK sont la priorité et le numéro de
        l'opérateur ou du délimiteur au sommet du stack ;
        SI PRIOSTACK ≥ PRIOCHAINE ALORS
           ALLERA SI PRIOSTACK < 4 ALORS GENERATION INSTRUCTION
                                   SINON GENERATION OPERATEUR ;
        SI PRIOCHAINE ≥ 5 ALORS ALLERA EMPILER OPERATEUR ;
        SI PRIOCHAINE = 4 ALORS
           DEBUT COMMENTAIRE On examine le symbole SINON en chaîne ;
           SI S[XSTA] := <SYMB SI-ALORS-I> ALORS ALLERA WSIAI ;
           SI PRIO STACK ≥2ALORS ALLERA GENERATION INSTRUCTION2 ;
           FIN ;
           ALLERA FIN EXPRESSION OU INSTRUCTION ;

EMPILER OPERATEUR : STACKE : COMMENTAIRE on vérifie avant d'empiler un opérateur
   arithmétique, logique, ou relation, examiné en chaîne, que le type courant,
   c'est-à-dire ici le type à gauche de cet opérateur correspond à la catégo-
   rie de l'opérateur, grâce à la procédure VERITY ;
   VERITY (PRIOCHAINE) ; XSTA := XSTA+1 ; SI XSTA ≥ XDB ALORS ERREUR (40) ;
   S[XSTA] := CHAINE [ICC] ; ALLERA SI PRIOCHAINE = 10 ALORS EXPAS
   SINON SI PRIOCHAINE > 10 ALORS SECBOO SINON TERM ;
   COMMENTAIRE On appele les programmes particuliers d'EXPRESSION correspondant
   à la syntaxe des expressions : EXPAS, SECBOO, TERM, lorsqu'on a empilé
   respectivement un opérateur de relation, un opérateur logique, un opérateur
   arithmétique ;
```

```
GENERER OPERATEUR : ;COMMENTAIRE on vérifie maintenant que le type courant,
c'est-à-dire de l'expression à droite de l'opérateur désempilé et généré cor-
respond bien à la catégorie de ce dernier. Après génération, le type courant (type
de l'expression actuellement générée) ne change pas sauf pour la génération
d'une relation ou il devient booléen. Le numéro présent dans le code d'un
opérateur est l'ordre à générer ;
VERITY (PRIOSTACK) ; P[XCP] := NUMSTACK;
SI NUMSTACK ≥ 8 ALORS XTYP := TYPB SINON DEBUT COMMENTAIRE
Condensation si l'ordre précédemment généré est CRN ou CEN, des ordres-opé-
rations arithmétiques en ordre avec adresse : TBORD1 et TBORD2 donnent res-
prectivement à une constante près(la valeur de CRN et CEN)les codes des ordres
correspondant aux Réels et aux Entiers ;
 A := P[XCP-1]∧H760000 ; SI A = HO20000 ALORS
 A := TBORD1 [NUMSTACK] SINON SI A = HO40000 ALORS
 A := TBORD2 [NUMSTACK] SINON ALLERA FINCD ;
SI A = 0 ALORS ALLERA FINCD ; COMMENTAIRE Il n'existe pas d'ordre arithmétique
avec adresse : cas de -unitaire ;
 P[XCP-1] := A + P[XCP-1] ; XCP := XCP-1 ;
 FINCD : FIN ;
XCP := XCP+1 ; SI XCP ≥ XCP MAX ALORS ERREUR ( ) ;
GENBA : XSTA := XSTA-1 ; ALLERA TESTP ;
  COMMENTAIRE On recommente le traitement a son début, c'est toujours le même
  opérateur délimiteur qui est examiné en chaîne ;

GENERATION INSTRUCTION : GENERB :
      SI PRIOSTACK = 1 ALORS ALLER GENERC ;
```

GENERATION INSTRUCTION 2 : GENERB2 : ; COMMENTAIRE La priorité du symbole en stack est soit 2 ce qui correspond aux INSTRUCTIONS de TYPE 2 : ALLER A et Affectations, soit 3 ce qui correspond au délimiteur SI-ALORS SINON-E dont le traitement ne transite ici que pour utiliser la même séquence de vérification : le type incorporé au délimiteur empilé doit être le même que le type courant XTYP ;
SI XTYP ≠ COMPLEMENT LOGIQUE (S[XSTA] ∧ H300000) ALORS
ERREUR (34) ; SI NUMSTACK = H40 ALORS ALLERA GENERD ;
COMMENTAIRE Ceci est le délimiteur SI-ALORS-SINON-E en stack ;
GENERER (NUMSTACK) ; SI NUMSTACK ≥ H35
ALORS DEBUT COMMENTAIRE Génération du second mot de l'ordre dans le cas des affectations de procédure : il est empilé en dessous de l'opérateur d'affectation ; XSTA := XSTA-1 ; GENERER (S[XSTA]) ;
FIN ;
ALLERA GENBA ; COMMENTAIRE Après désempilage de l'opérateur, le traitement est recommencé pour générer les opérateurs d'affectation éventuellement empilés dans le cas d'une Instruction d'affectation, et pour déterminer par la méthode générale le programme Instruction qui a appelé et y retourner grâce au délimiteur caractéristique parenthèse gauche d'Instruction ;

GENERATION INSTRUCTION 1 : GENERC : ; COMMENTAIRE La priorité du symbole en stack est 1 ce qui correspond aux INSTRUCTION de TYPE1 : instructions POUR, instruction SLALORS et SLALORS-SINON. Rappelons que la priorité du délimiteur en chaîne est inférieure ou égale à 1 et que les instructions SI ALORS SINON sont constituées ailleurs : le délimiteur SINON examiné en chaîne a une priorité 4 qui lui est propre et permet de le reconnaître ;

SI NUMSTACK = H54 ALORS DEBUT COMMENTAIRE Délimiteur POUR-FAIRE.
Séquence particulière à la fin d'Instruction POUR : génération de l'ordre Ø RTP (2 mots le second est empilé en dessous de l'opérateur) ; XSTA := XSTA-1 ;
GENERER (<CODE Ø RTP>) ; GENERER (S[XSTA]) ;
FIN ;

GENERD : XSTA := XSTA-1 ; P[S[XSTA]] := P[S[XSTA]] $\vee$ XCP ;
ALLERA GENBA ; COMMENTAIRE On a complété l'ordre du programme objet dont l'adresse est empilée dans le stack, en plaçant dans sa partie adresse, l'adresse du prochain ordre qui sera généré. Comme pour toutes les fins d'Instructions (cf. GENERB2) le traitement se poursuit par GENBA. La séquence GENERD est utilisée aussi en fin d'expression SLALORS SINONE : ceci est possible puisque le mécanisme de retour au programme appelant est unique ;

FIN INSTRUCTION OU EXPRESSION : FERME :
SI PRIOSTACK ≠ 0 ALORS ERREUR (31) ;
COMMENTAIRE La génération d'une expression ou d'une instruction est terminée.

On commence par examiner si l'on vient d'achever la génération d'une expression appartenant à une liste et délimitée par ',' en chaîne codée. Le numéro du délimiteur ouverture de parenthèse en stack se décompose en numéro de famille K, et N identification dans la famille. Pour une liste, K permet une première orientation vers le programme de liste correspondante dans lequel on fera le vérification de construction syntaxique grâce à N ;

SVIR : SI NUMCHAINE = H60 ALORS
DEBUT K := (NUMSTACK ∧ H000070)x2↑(-3) ;
A2 := N := NUMSTACK ∧ H000007 ;
ALLERA AIG VIRGULE [K+1] ; ER23 : ERREUR (23) ;
FIN Voir page 41 ;

FERMA : COMMENTAIRE Le classement des délimiteurs en famille permet d'éliminer les premières construction erronées : le délimiteur examiné en chaîne codée doit appartenir à la famille du délimiteur parenthèse gauche d'instruction ou d'expression en stack, ce qui est reconnu en faisant la différence XDIF de leurs numéros.

Le délimiteur en stack caractérise un programme appelant, auquel on pourrait retourner immédiatement; on y ferait les vérifications de construction nécessaires (plusieurs délimiteurs différents de chaîne codée peuvent appartenir à la même famille), et les orientations vers les sous programmes particuliers s'il peut effectivement y avoir plusieurs constructions différentes. cf. p. 6. Ceci est fait directement ici

XDIF := NUMCHAINE -NUMSTACK ; SI XDIF < 0 ∨ XDIF ≥ H10
ALORS ERREUR (21) ;
K := (NUMCHAINE ∧ H000070)x2↑(-3) ;
I := (NUMCHAINE ∧ H000007) -TBREDUCTION [K] ;
COMMENTAIRE K désigne une famille de délimiteurs parenthèses d'ou un premier aiguillage. Pour chaque programme Pj de la famille considérée, I, caractérisant le délimiteur parenthèse droite examiné en chaîne codée, détermine le retour (qui peut être un constat d'erreur) lorsque ce Pj a appelé : Dans l'ensemble de ces retours aux Pj de la famille, XDIF caractérise alors bien le programme appelant, d'où la sélection du retour correspondant. I est le numéro réduit du délimiteur examiné en chaîne codée (numérotation ramenée à 0 pour les délimiteurs parenthèse droite de la famille). XDIF donne un numéro réduit du délimiteur parenthèse gauche caractéristique du programme appelant, numéro rapporté au délimiteur examiné en chaîne. Ceci permet avec les familles, la programmation avec des matrices de taille minimum ;

SI K = 2 ALORS

TRAITEMENT FIN LISTE D'INDICES : FLIND : DEBUT SI XTYP ≠ TYPA ALORS ERREUR (28) ; XTYP := COMPLEMENT LOGIQUE (S[XSTA] ∧ H300000); XSTA := XSTA-1 ; PROG ; FIN On a vérifié le type arithmétique du dernier indice, rétablit comme type courant le type incorporé au délimiteur [empilé. Cette séquence commune au traitement des listes d'indices (et de bornes) regresse aussi le stack et progresse le pointeur de chaîne codée ;

AIGUILLAGE DE RETOUR AU PROGRAMME INSTRUCTION OU
EXPRESSION APPELANT ET VERIFICATIONS DE SYNTAXE :

DEBUT AIGUILLAGE PROGRAMME FAMILLE := BRIN [I] ;
BR PAR [I], BRCRO [I], BRSI [I], INEXISTANT , BRPOU [I] ;

AIGUILLAGE BRIN := ZFINA [XDIF], ZPVFA[XDIF], ZPVA[XDIF] ;
AIGUILLAGE ZFINA := WSBL, WSBI ;
AIGUILLAGE ZPVFA := WPVFP, ER38, ER38 ;
AIGUILLAGE ZPVA := WPVAIG, ER39, INS, INS ;

AIGUILLAGE BRPAR := ZPARA[XDIF] ;
AIGUILLAGE ZPARA := PAR INEXISTANT, ZPARFI, ZPARP, SUITEX ;
COMMENTAIRE Pour les Programmes d'Instructions (en fait liste de paramètres) ou expression classés dans la famille 1 (BRPAR) il n'existe qu'une seule construction syntaxique, et les constructions erronnées ont été éliminées grâce au test sur XDIF : aucun délimiteur du type parenthèse droite autre que) n'appartient à cette famille. Ce n'est pas le cas dans la famille 0 (BRIN) où l'on a les "Instructions" Bloc, Instruction Composée, Déclaration de Procédure, déclaration d'aiguillage.

Les autres aiguillages sont donnés sous forme de tableaux en pages "hors programme"... ;

K := K+1 ; I := I+1 ; XDIF := XDIF+1 ; ALLERA PROGRAMME FAMILLE [K] ;
ER38 : ERREUR (38) ; ER39 : ERREUR (39) ;....;
INEXISTANT : PARINEXISTANT : ERREUR ('NE PEUT SE PRODUIRE.ERREUR OPERATEUR OU MATERIEL ')
FIN AIGUILLAGE DE RETOUR ;

VIRCRO : SI XTYP ≠ TYPA ALORS ERREUR (28) ;
SI A2 = 0 ALORS CTAB := CTAB+1 ;
ALLERA ENTEX1 ;
COMMENTAIRE Traitement liste d'indices, ou de bornes (A2=0). Après vérification, on appelle EXPRESSION pour pour poursuivre le traitement ;
L'organigramme page [illegible] donne la syntaxe de celui-ci ;

WSIE : XSYMB := <SYMB SI-ALORS-E> ;
CLE SI INSTRUCTION := FAUX ; ALLERA COMMUN ALORS ;

WSIAE : XSYMB:= <SYMB SI-ALORS-SINON-E> ;
CLE SI INSTRUCTION := FAUX ; ALLERA COMMUN SINON ;

FIN FIN du GENERATEUR

HORS PROGRAMME

FERME :

RETOURS AUX PROGRAMMES,

AIGUILLAGES donnés sous forme de Tableaux .

FAMILLE 0

BRIN	ZFINA
	ZPVFPA
	ZPVA

ZFINA	WSBL
	WSBI
ZPVFPA	WPVFP
	ER38
	ER38
ZPVA	WPVAIG
	ER39
	INS
	INS

Délimiteurs

CHAINE	STACK	Commentaire
FIN	DEB-Bloc	fin de bloc
	DEB-I	fin instruction composée
;FP	PROCD	fin (de corps) de procédure
	DEB-Bloc	erreur
	DEB-I	erreur
;	AIGUILLAGE	fin liste d'aiguillage (déclaration)
	PROCD	erreur
	DEB-Bloc	bouclage Instruction (une instruction
	DEB-I	vient d'être générée)

FAMILLE 3

BRSI	ZSI5
	ZSI6

ZSI5	WSII
	WSIE
ZSI6	WSIAE
	ER8
	ER8

ALORS	SI-I	fin proposition SI d'Instruction
	SI-E	fin proposition SI d'Expression
SINON	SI-ALORS-E	fin 1ere expression de construction SI-ALORS-SINON
	SI-I	erreur
	SI-E	erreur

FAMILLE 1

BRPAR | ZPARA |

ZPARA **

ZPARFI
ZPARP
SUITEX

1 seul Aiguillage final

délimiteurs en chaîne	en STACK	Commentaire
)	Réservé	Inexistant {éventuellement F. Stand}
	(FI	Fin liste paramètres effectifs d'Indicateur de fonction
	(IP	Fin liste paramètres effectifs d' Instruction procédure
	(2	Fin d'expression parenthèsée

FAMILLE 2

BRCRO | ZCROA |

ZCROA

CROPE
CRVI2
CRVI2M
CRVIO
CROPOU
CRODT
CROAIG

1 Seul Aiguillage final

en chaîne	en STACK	Commentaire
]	[PE	Fin liste d'indices;v.I. en début de paramètre effectif
	[VI2	Fin liste d'indices;v.i. en 1er partie gauche affectation
	[VI2M	Fin liste d'indices;v.i. peut être en partie gauche affectation.
	[VIO	Fin liste d'indices;v.i. dans expression
	[POUR	Fin liste d'indices;v.i. variable contrôlée de POUR
	[DT	Fin liste de bornes {déclaration tableau}
	[AIG	Fin indice d'Indicateur d'Aiguillage

v.i. variable indicée

Nota Avant ces branchements on a restauré le type incorporé au délimiteur stacké, progressé l'index de chaîne ICC sur l'unité syntaxique qui suit], vérifié le type arithmétique des indices - ou bornes -

FAMILLE 4

BRPOU	ZPAS
	ZJUSQ
	ZTANT
	ZFAIR

Délimiteurs

		En chaîne	en STACK	Commentaire
ZPAS	ER16	PAS	POUR-PAS	Erreur construction
	FAS		POUR	fin 1re partie élément POUR3
ZJUSQ	ER17	JUSQUA	POUR-PAS-JUSQUE	erreur
	FPAJU		POUR-PAS	fin 2de partie élément POUR3
	ER17		POUR	erreur
ZTANT	ER18	TANT QUE	POUR-TANTQUE	erreur
	ER18		POUR-PAS-JUSQUE	erreur
	ER18		POUR-PAS	erreur
	FTANT		POUR	fin 1ère partie élément POUR2
ZFAIR	ER19	FAIRE	POUR-FAIRE	erreur
	TANTQF		POUR-TANT QUE	fin Elément POUR2 TERMINAL
	JUSQF		POUR-PAS-JUSQUE	fin Elément POUR 3 TERMINAL
	ER19		POUR-PAS	erreur
	POUQF		POUR	fin Elément POUR1 TERMINAL

Nota : TERMINAL c'est-à-dire élément terminant la liste de POUR, par opposition au même élément suivi du S.B. <,>

CHAPITRE XIV

CONCLUSION

DETECTION DES ERREURS - RESTRICTIONS - MISE EN OEUVRE DU COMPILATEUR ET RESULTATS

14.1. La détection des erreurs.

Une caractéristique très importante d'un compilateur est la manière dont il détecte les erreurs syntaxiques et sémantiques. L'utilisateur demande qu'elles soient toutes détectées, le plus tôt possible et que le diagnostic soit précis ; ces objectifs sont difficiles à atteindre.

On notera qu'il existe certaines catégories d'erreurs difficiles à détecter comme l'utilisation de la valeur d'une variable avant qu'une valeur ne lui ait effectivement été affectée, le non définition de la valeur d'un indicateur de fonction si aucune instruction d'affectation correspondante n'est exécutée dans la procédure. On en s'intéresse pas à ce genre d'erreurs que l'on considère comme inhérentes à l'algorithme décrit par le texte source (de la même manière que les boucles perpétuelles par exemple) mais la technique interprétative permettrait de les déceler.

Il est très intéressant que seul un minimum d'erreurs reste à détecter à l'exécution ; cela facilite la mise au point des programmes et rend l'exécution plus efficace : les tests sont moins nombreux. Outre les restrictions de dépassement de la zone de travail inévitables dans une méthode dynamique, les erreurs détectées à l'exécution sont (cf. ch. 11):

- Nombre de paramètres erronnés dans un appel de procédure
- Paramètres effectifs ne correspondant pas aux spécifications ou mal utilisées (paramètre effectif expression apparaissant en partie gauche d'une instruction d'affectation).

. Nombre d'indices d'une variable indicée ne correspondant pas à la dimension du tableau.
. Valeurs d'un indice de variable indicée non comprise entre les bornes correspondantes du tableau.
. Borne inférieure supérieure à la borne supérieure dans une création de Tableau.
. Division par zéro, Division Entière sur des Réels,...
. Débordements dans opérations arithmétiques ou à l'affectation d'une valeur arithmétique (cf. ch. 3).

Toutes les autres erreurs sont détectées à la compilation ; nous pensons que la détection est effectivement complète, et donne un diagnostic assez clair grâce à la méthode de génération.

La nécessité de détecter le maximum d'erreurs dès le premier passage d'Edition conduit à faire des vérifications qui seront refaites à la génération, car l'analyse effectuée au niveau de l'édition n'a pu être que très partielle : elle ne permet de détecter qu'un certain nombre d'erreurs courantes. Outre les vérifications habituelles de parenthèsages, on diagnostique ainsi la majorité des cas où les symboles DEBUT, ALLERA, POUR, SI, les := d'affectation, les étiquettes ne sont pas introduits de façon correcte. D'autre part on avertit le programmeur de certains points comme un commentaire comprenant des = et (ce qui peut être un oubli de ; , ou l'étiquetage d'une instruction contrôlée par une instruction POUR (cf. ch. 8). Les erreurs détectées lors des 2 phases de compilation provoquent l'arrêt excepté pour les premières erreurs du passage d'Edition.

Lors de l'édition, on a vu que l'on imprime la ligne où est détectée une erreur, ainsi que le numéro de l'unité syntaxique en traitement lors de cette détection. D'autre part, comme les identificateurs non déclarés ne sont connus qu'en fin de programme, on les liste en indiquant la ligne où ils ont été référés pour la première fois. Ceci est possible car la chaîne codée contient des numéros de lignes pour préciser les diagnostics d'erreurs au second passage :traité ligne par ligne à l'édition, le texte source n'existe jamais en machine.

Les numéros de ligne de la chaîne codée servent à tenir à jour un compteur de ligne à la génération et la valeur de ce compteur est indiquée dans un constat d'erreur. De plus ces numéros de ligne sont générés dans le programme objet en ordre de remise à jour *L* du compteur de ligne à l'exécution. Le but est de préciser le diagnostic d'une erreur détectée à l'exécution, sans que le programmeur ait besoin de regarder le programme objet ; on peut toujours lui donner l'adresse de l'ordre objet exécuté lorsque cette erreur s'est produite, mais ce n'est pas une information exploitable rapidement et facilement.

Les ordres *L* reflètent assez fidèlement la présentation typographique du texte source en ligne, mais la correspondance n'est pas parfaite : le compilateur traduit immédiatement le changement de ligne dès qu'il est obligé de le lire dans son examen génératif du texte source ; la présentation en ligne n'est exactement respectée que vis à vis des identificateurs. Précisons par un exemple simple ; la présentation suivante du texte source

```
I := A + B x
       C ;
```

donne la présentation suivante du programme objet (les 2 lignes sont séparées par un ordre *L*) :

I A B

C × + :=

Notons d'ailleurs, qu'une ligne source est considérée comme terminée sur le dernier délimiteur (ou valeur logique) sauf s'il appartient à 2 lignes.

En fait le problème intéressant serait d'obtenir qu'à l'exécution d'une partie de programme écrite sur la ligne K du texte source, le dernier ordre *L* exécuté ait bien affecté la valeur K au compteur de ligne. Ce problème de trace automatique dans un programme Algol n'est pas simple à résoudre et ainsi posé demanderait à être précisé. On serait amené par exemple à générer à nouveau un ordre *L* après chaque activation de procédure....Nous nous sommes limités à réintroduire dans le programme objet le numéro de ligne courant après un paramètre effectif chaîne ; ce pourrait être utile pour certaines procédures standard d'entrée-sortie.

14.2. Restrictions du langage.

Les restrictions du langage ALGOL 60 sont peu nombreuses (cf. annexe) ; indiquons les plus importantes :

1) Les variables rémanentes n'existent pas. Les autoriser conduirait à accroître l'importance des réservations ce qui n'est guère souhaitable sur une petite machine.

2) Tous les paramètres formels doivent être spécifiés (ch. 9)
3) Il est interdit d'appeler les tableaux par valeur (ch. 9)
4) Les étiquettes numériques sont interdites.
   Les identificateurs ne doivent pas comporter plus de 12 caractères dont seuls les 6 premiers sont significatifs.

A ces restrictions du langage s'ajoutent des restrictions imposées par la machine sur la dimension du programe, les valeurs. Elles sont aussi indiquées en annexe.

La représentation machine du langage est donnée en annexe du ch. 12. Les symboles de base entre " " peuvent être abrégés aux 3 premières lettres.

14.3. Implantation du Compilateur sur la machine.

Les CAE série 500 sont des calculateurs binaires à mots de 18 digits. La mémoire à tores est extensible de 8 K à 32 K par blocs de 8 K. Les ordres machines simples, ou logandes réguliers, s'exécutent en 12 ou 18 µs selon le mode d'adressage direct ou indirect (temps d'accès à la mémoire 3 µs). Le mode d'utilisation habituel est la microprogrammation ; la division en double précision (flottant 3 mots) par exemple s'effectue en 1128 µs. Ces calculateurs sont à logique d'interruption à 2 niveaux (CAE 510) ou 8 niveaux (RW 530).

Le système d'entrée-sortie minimum est constitué par
1) un lecteur rapide de ruban perforé (250-500 caractère/s)
2) un perforateur rapide de ruban (60 c/s)
3) une machine à écrire émettrice réceptrice.

Les extensions possibles du CAE 510 comportent 8 dérouleurs de bande magnétiques, 8 tambours magnétiques, Entrée-Sortie par cartes perforées, Imprimante, Table Traçante, Entrées-Sorties numériques et analogiques.

Le Compilateur fonctionne sur la version minimum : mémoire de 8 K et Entrées-Sorties minima indiquées précédemment. Son utilisation est plus commode si l'on dispose d'un dérouleur de bande magnétique, mais sa structure limite les manipulations si l'on ne dispose que d'un lecteur rapide.

L'ensemble compilateur effectue la compilation sans que la chaîne codée quitte la mémoire à l'Edition et génére directement le programme objet en mémoire ; les procédures standard utilisées sont ensuite chargées. Il est alors possible d'effectuer les opérations suivantes :

1) Perforation du programme objet et de ses procédures standard en vue d'une exécution ultérieure(tels qu'ils sont implantés en machine : binaire)
2) Listage du programme objet en symbolique sur machine à écrire ou perforateur.
3) Exécution du programme.

L'ensemble compilateur se compose donc essentiellement de 3 parties, qui se chargent successivement en haut de mémoire (l'enchaînement est automatisé): EDITEUR - GENERATEUR - INTERPRETEUR.

Utilisation pratique et implantation.

Sans dérouleur de bande magnétique,on a matériellement 2 rubans perforés :

Ruban I : Editeur binaire absolu

Ruban II : a) Générateur binaire absolu

b) Chargeur de Procédures standard-Listeur binaire absolu

c) Bibliothèque de Procédure Standard binaire pseudo translatable

d) Interpréteur binaire absolu

Les manipulations consistent à mettre successivement 3 rubans sur le lecteur rapide : Ruban I, Programme Algol, Ruban II.

a) enregistrer le ruban I : Editeur

b) lancer l'Edition, le programme étant sur le lecteur ;

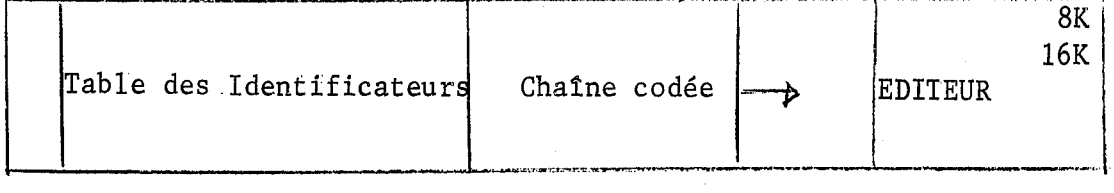
c) l'Edition terminée, et le ruban II sur le lecteur, on déclenche successivement:

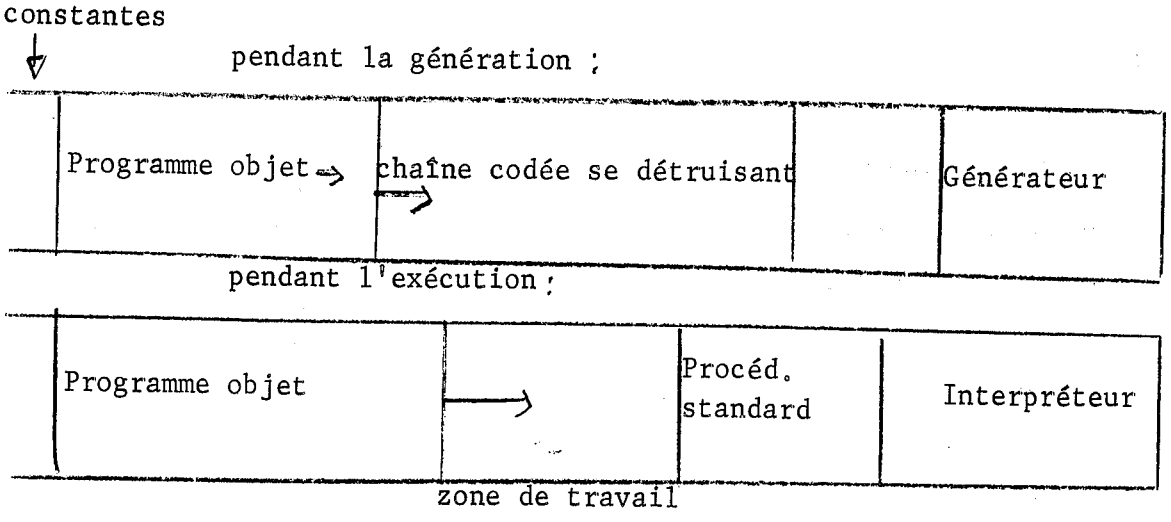
- . génération, puis chargement des procédures standard
- . listage } facultatif
- . exécution }

Si l'on dispose d'un dérouleur de bande magnétique RUBAN I et RUBAN II y sont enregistrés et il suffit de mettre le programme sur le lecteur.

Implantation en mémoire au cours des différentes phases.

pendant l'Edition :





14.3.1. Volume du Compilateur

Les encombrements sont les suivants

a) Editeur : 3100 mots

b) Générateur : 2100 mots,

Ceci comprend pour chaque programme ses tables, son programme de constat d'erreur, les enchaînements d'exécution. Il faut ajouter : pour l'éditeur la pile des identificateurs (1500 mots actuellement), et pour le générateur la pile opérateur (120 mots), le programme de chargement des procédures standard et des sorties du programme objet (800 mots).

c) Interpréteur : 3400 mots

où l'on peut distinguer . les logrammes d'opérations (arithmétiques 900 mots)

. un noyau de sorties (conversion décimale, impression 660 mots)

. les programmes dûs à la méthode interprétative (1700 mots)

S'il n'est guère possible de réduire les 2 premièrs points qui correspondent à l'adaptation de la bibliothèque CAE, la programmation pourrait être améliorée

sur le dernier point (on pourrait espérer la réduire de 10 %) . A l'exécution il faut tenir compte en outre de l'encombrement des procédures standard utilisées, ce qui varie selon les programmes source.

14.4. Résultats et Conclusion.

La compilation est rapide et le temps de compilation est pratiquement négligeable compte tenu du type de la machine ; l'édition se fait sensiblement à la vitesse de lecture du ruban perforé source ; à la génération le temps de chargement des procédures standard reste prépondérant.

On estime sans attacher une signification très grande à cette comparaison, que la méthode interprétative multiplie le temps d'exécution d'un programme écrit en mode normal (microprogrammé) par

2 pour les blocs sans faire intervenir les paramètres et les variables indicées,

5 en faisant intervenir les appels de procédures, les calculs avec paramètres et variables indicées.

Quant à l'encombrement du programme objet, on peut l'estimer en moyenne à 0,8 - 0,9 mot par unité syntaxique du programme source (0,7 à 1,2 dans les cas extrêmes. Ceci n'a été examiné que sur des programmes assez courts).

Nous nous sommes efforcés au cours de ce travail d'examiner simultanément les problèmes posés et les variantes possibles de la méthode,et d'indiquer leur avantages ou inconvénients. Aussi nous concluerons simplement en rappelant que si le langage objet est défini dans sa structure générale, il doit être adapté au problème particulier traité. Lorsque le langage intermédiaire est interprété comme sur le CAE 510, sa définition précise dépendra du service attendu du compilateur : compilation pour la mise au point de programmes ou compilation en vue d'une exécution efficace ; le compilateur CAE 510

nous paraît être une solution moyenne entre ces deux tendances. Rechercher l'efficacité d'exécution conduit naturellement à des compilateurs importants, et peut augmenter dans une certaine mesure la dimension de l'Interpréteur ; le langage objet est en effet plus riche. Généralement on sera limité par la taille de la machine, les moyens d'entrée-sortie et d'autres caractéristiques du calculateur (par exemple les 18 digits d'un mot ne laissent que de faibles possibilités de codage dans le cas du CAE 510). En effet la méthode interprétative s'applique souvent à des petites ou moyennes machines, et de façon plus intéressante si la machine est à logique de microprogrammes. Sur une grosse machine il ne semble guère concevable de ne disposer que d'un compilateur interprétatif ; mais il trouverait sa place à côté d'un compilateur génératif soit pour la mise au point de programmes, soit pour l'exécution de programme utilisant toutes les possibilités du langage, possibilités qu'un compilateur classique traite assez mal.

Il devient de plus courant qu'un tel langage intermédiaire au lieu d'être interprété serve de langage source pour un passage final d'un compilateur génératif. Son travail est alors grandement facilité par la séparation des tâches de compilation : l'analyse du programme source reflétée dans le cas du langage intermédiaire décrit par la notation post fixée, puis la traduction en langage machine.

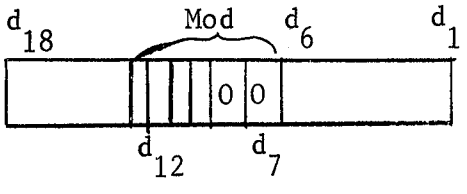
A N N E X E

Chapitre 9 - Annexe.

Modèles $d_{12}.....d_7$ (caractère M du mot)

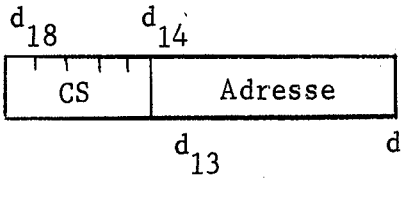
Valeur octale donnée cadrée à droite sur d_7 ($d_7 = d_8 = 0$)

Valeur octale d_{12}-d_7	Symbolique	Spécification du formel	Valeur octale	symbolique	Spécification du formel
04	CH	CHAINE	50	PR	REEL PROCEDURE
			60	PE	ENTIER PROCEDURE
10	QR	REEL	70	PB	BOOLEEN PROCEDURE
20	QE	ENTIER			
30	QB	BOOLEEN	44	PO	PROCEDURE
14	TR	TABLEAU | REEL TABLEAU	54	ET	ETIQUETTE
24	TE	ENTIER TABLEAU			
34	TB	BOOLEEN TABLEAU	64	AI	AIGUILLAGE



mot modèle dans la liste entête de procédure.

Caractérisations statiques



mot caractérisation statique avant l'ordre d'appel

la valeur octale de la partie qualitative Cs (d_{18}-d_{14}) est donnée cadrée à droite sur d_{13} que l'on suppose nul.
[d_{13} n'appartient pas à Cs mais cette notation sera plus claire : dans les listings CS est listé comme l'ordre qui a les même 5 digits d_{18}-d_{14} de code - voir 3.4.1.]

Valeur octale CS	Symbolique	Le paramètre est une quantité déclarée	La partie adresse de CSt est :
24	PR	REEL PROCEDURE	l'adresse de la mémoire en table correspondante
30	PE	ENTIER PROCEDURE	
34	PB	BOOLEEN PROCEDURE	
22	PO	PROCEDURE	
26	ET	ETIQUETTE	
32	AI	AIGUILLAGE	
04	QR	REEL	adresse dynamique de variable simple ou variable &
10	QE	ENTIER	
14	QB	BOOLEEN	
06	TR	REEL TABLEAU | TABLEAU	
12	TE	ENTIER TABLEAU	
16	TB	BOOLEEN TABLEAU	
		le paramètre est :	
46	VIR	une variable indicée. Le tableau correspondant est de type (1) REEL	adresse sous programme objet de calcul adresse
52	VIE	ENTIER	
56	VIB	BOOLEEN	
72	EXPA	Expression arithmétique (2)	adresse du sous programme objet d'évaluation
66	EXPB	Expression booléenne	
62	EXPD	Expression de désignation	
44	VR	Nombre sans signe ou positif REEL	adresse statique de la mémoire table correspondante
50	VE	ENTIER	
54	VL	BOOLEEN	
40	PF	un paramètre formel	adresse dynamique de Cdy
02	CH	une chaîne	adresse de la chaîne dans le programme objet.

(1) le tableau est déclaré ou est formel ; le type indiqué est celui de la déclaration ou de la spécification.

(2) le type de l'expression est inconnu (est réel ou entier)

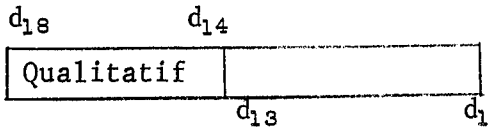
ANNEXE ch. XII

CODIFICATION DES SYMBOLES DE BASE

	ALGOL	PROGRAMME SOURCE (Flexo)	CHAINE CODEE Codification d_{18}-d_1 octal			Remarques
OP. Arithmétiques	+	+	76	13	01	
	- {	- {	76	13	02	
			76	13	03	- unitaire
	x	*	76	14	04	
	/	/	76	14	05	
	÷	%	76	14	06	
	↑	!	76	15	07	
OP. de relation	<	'INF'	76	12	10	
	≤	'ING'	76	12	11	
	=	'EG' ou 'EGA'	76	12	12	
	≥	'SUG'	76	12	13	
	>	'SUP'	76	12	14	
	≠	'DIF'	76	12	15	
OP logiques	≡	'EQU'	76	05	22	
	⊃	'IMPL'	76	06	21	
	V	'OU'	76	07	20	
	Λ	'ET'	76	10	17	
	¬	'NON'	76	11	16	
OP séquentielles	GO TO	'ALLERA'	76	02	34	
	IF	'SI'	76	00	34	
	THEN	'ALORS'	76	03	35	
	ELSE	'SINON'	76	04	36	
	FOR	'POUR'	76	00	50	
	DO	'FAIRE'	76	03	54	
SEPARATEURS	/	/	76	03	60	
	.	.				n'existent pas (délimiteurs internes à un nombre pur)
	10	$				
	:	:	76	03	61	
	; {	; {	76	01	03	
			76	01	02	;FP délimitant une déclaration de PROCEDURE
	:=	:= ou =	76	02	23	

	ALGOL	PROGRAMME SOURCE	CHAINE CODEE			
SEPARATEURS	␣					Caractère d'une chaîne
	STEP	'PAS'	76	03	51	
	UNTIL	'JUSQUA'	76	03	52	
	WHILE	'TANT QUE'	76	03	53	
	COMMENT	'COMMENTAIRE'				n'existe pas
CROCHETS	(	(	76	00	10	
	)	)	76	03	13	
	[	.(	76	00	20	
	]	).	76	03	25	
	⌜	<	76	00	70	
	⌝	>	76	01	70	
DECLARATEURS	BEGIN	'DEBUT'	76	00	01	
	END	'FIN'	76	01	01	
	BOOLEAN	'BOOLEEN'				sont supprimés
	INTEGRER	'ENTIER'				
	REAL	'REEL'				
	ARRAY	'TABLEAU'	76	00	04	
	SWITCH	'AIGUILLAGE'	76	00	03	
SPECIFICATEURS	STRING	'CHAINE'				sont supprimés
	LABEL	'ETIQUETTE'				
	VALUE	'VALEUR'				

Partie qualitative d'une unité syntaxique en chaîne codée



La représentation octale est cadrée à droite sur d_{13} (qui n'en fait pas partie), et d_{18} est constitué comme $\pm$

REPRESENTATION Symbolique	Octal	NOM	CODIFICATION d_{18}	d_{17}	d_{16}	d_{15}	d_{14}	REMARQUES
SB	- 36	Symbole de base	1	1	1	1	1	voir 12.1. et codification page suivante
VR	- 00	Nombre Pur R	1	0	0	0	0	Constantes
VE	- 20	Nombre Pur E	1	1	0	0	0	et valeurs logiques
VL	+ 20	Nombre Pur B	0	1	0	0	0	
QR	$\pm$ 04	QUANTITE SIMPLE (TYPE) R	← Indication Paramètre Formel →	0	0	1	0	Identificateur de quantité déclarée ou de formel spécifié, comme le nom indiqué. . R, E, B dénote le type respectivement REEL, ENTIER, BOOLEEN et est alors donné par les digits d_{16} d_{15} . Les formels sont reconnus par d_{18} = 1 (digit de signe négatif, 3ème lettre F dans représentation symbolique)
QE	$\pm$ 10	QUANTITE SIMPLE (TYPE) E		0	1	0	0	
QB	$\pm$ 14	QUANTITE SIMPLE (TYPE) B		0	1	1	0	
TR	$\pm$ 06	TABLEAU R		0	0	1	1	
TE	$\pm$ 12	TABLEAU E		0	1	0	1	
TB	$\pm$ 16	TABLEAU B		0	1	1	1	
PR	$\pm$ 24	PROCEDURE FONCTION (TYPE PROCEDURE) R		1	0	1	0	
PE	$\pm$ 30	PROCEDURE FONCTION (TYPE PROCEDURE) E		1	1	0	0	
PB	$\pm$ 34	PROCEDURE FONCTION (TYPE PROCEDURE) B		1	1	1	0	
PO	$\pm$ 22	PROCEDURE		1	0	0	1	
ET	$\pm$ 26	ETIQUETTE		1	0	1	1	
AI → FORMEL	$\pm$ 32	AIGUILLAGE		1	1	0	1	
CH F	- 02	CHAINE	1	0	0	0	1	uniquement formel
Ø	+ 00	PROCEDURE STANDARD	0	0	0	0	0	Identificateur de procédure Standard
NL	+ 01	Numéro de LIGNE	0	0	0	0	1	AUTRES ELEMENTS voir 12.1
MOD	+ 36	Modèle	0	1	1	1	1	
Ø	+ 00		0	0	0	0	0	Selon Contexte Indicateur de Réservation, Nombre paramètres

EXEMPLE DE CHAINE CODEE

```
000 0000  'DEBUT' 'REEL' A,B,C ; 'REEL' 'PROCEDURE' PA(X,Y,Z) ; 'VALEUR' Z ;
001 0022  'REEL' X,X,Y ; 'DEBUT' Y := X*Y ; PA := Y+2 'FIN' DE PA ;
002 0044  'ENTIER' I,J ;
003 0052  A := B := I := 2 ;
004 0057  PA (A,B,I) ;
005 0066 J := PA (A+B, I,PA(A,B,I)) ;
006 0086 'FIN'
```

↑ Numéro de ligne ; ↑ Numéro de la 1ère unité syntaxique de la ligne.

Des exemples de listing du programme source, produit par l'Editeur sont donnés avec les programmes en Annexe.

Listing symbolique de la chaîne codée : Les symboles de base sont indiqués éventuellement par les 3ers caractères, et les parties qualitatives avec les conventions du programme objet. Dans un modèle (ch. 9 Annexe, N et V dénotent le mode d'appel Nom ou Valeur)

NL	0	SB	FIN	NL	5		
SB	DEB	SB	;FP	QE 0	12		
	13	NL	2	SB	:=		
SB	PRØ	NL	3	PR 0	0		
PR 0	0	QR 0	5	SB	(		
NL	1	SB	:=	QR 0	5		
	3						
MOD	QRN	QR 0	7	SB	+		
MOD	QRN	SB	:=	QR 0	7		
MOD	QRV	QE 0	11	SB	,		
	14	SB	:=	QE 0	11		
SB	DEB	VE	128	SB	,		
17777		SB	;	PO 0	0		
QRF 1	8	NL	4	QR 0	5		
SB	:=	PR 0	0	SB	,		
QRF 1	5	SB	(	QR 0	7		
SB	*	QR 0	5	SB	,		
QRF 1	8	SB	,	QE 0	11		
SB	;	QR 0	7	SB	)		
PR 0	0	SB	,	SB	)		
SB	:=	QE 0	11	SB	;		
QR 1	8	SB	)	NL	6		
SB	+	SB	;	SB	FIN		
SB	128						

ANNEXE Ch. XIII

PRIORITES

Les opérateurs (arithmétiques, logiques et relations) ont même priorité en chaîne codée et dans le stack : cette priorité va de 5 à 11 (voir leur code en Annexe du Ch. 12)

Opérateurs en chaîne codée (source)	PRIORITE	Opérateurs en Stack
DEBUT PROCEDURE AIGUILLAGE, TABLEAU ([< (ouverture chaîne') SI POUR	0	DEBUT-bloc DEBUT-Instruction PROCEDURE AIGUILLAGE TABLEAU ((IP (FI $[_{DT}$ $[_{AIG}$ $[_{POUR}$ $[_{VIO}$ $[_{VI2M}$ $[_{PE}$ $[_{VI2}$ SI-I SI-E SI-ALORS-E POUR POUR-PAS POUR-PAS-JUSQUE POUR-TANTQUE
FIN ; ;FP	1	SI-ALORS-I SI-ALORS-SINON-I POUR-FAIRE
ALLER A :=	2	ALLER A := $:=_M$ $:=_P$ $:=_{PM}$
)] ALORS PAS JUSQUA TANT QUE FAIRE , > (fermeture chaîne')	3	SI-ALORS-SINON-E
SINON	4	

Délimiteurs de Chaîne codée donnant naissance
à des délimiteurs différents en pile opérateur
(stack)

Chaîne codée				Stack			
DELIMITEUR	CODE			CODE			DELIMITEUR et Remarques
DEBUT	76	00	01	76 76	00 00	00 01	DEBUT Instr. composée DEBUT Bloc
PROCEDURE	76	00	02	76	00	02	PROCEDURE (décla.)
AIGUILLAGE	76	00	03	76	00	03	AIGUILLAGE
TABLEAU	76	00	04				n'existe pas en Stack $\lceil_{DT}$ suffit
FIN	76	01	01				
;FP	76	01	02				;fin corps de procédure
;	76	01	03				
ALLERA	76	02	34	76 D	02	34	
:= (affectation)	76	02	23	76 A,B	02	23	:= Affect. Simple
				76 A,B	02	24	$:=_M$ Affect. Multiple
				76 A,B	02	36	$:=_P$ Affect. Procédure Simple
				76 A,B	02	37	$:=_{PM}$ Affect. Procédure Multiple

Nota 76 A,B,D | indique que le type est incorporé dans les 5 digits de fort poids du code du symbole (A : arithmétique, B : Booléen, D désignation)

CHAINE CODEE				STACK			
DELIMITEUR	CODE			CODE			DELIMITEUR et Remarques
[	76	00	20 →	76 D	00	17	$[_{AIG}$ Indicateur d'Aiguillage
				76 A,B	00	20	$[_{DT}$ Liste bornes(déclaration tableau)
				76 A,B	00	21	$[_{POUR}$ variable Indicée contrôlée par un POUR
				76 A,B	00	22	$[_{VIO}$ Variable Indicée dans expression
				76 A,B	00	23	$[_{VI2M}$ Variable Indicée supposée en partie gauche (affect multiple)
				76 A,B	00	24	$[_{VI2}$ Variable Indicée Affectation Simple
				76 A,B	00	25	$[_{PE}$ Variable Indicée paramètre effectif.
]	76	03	25				
(	76	00	10 →	76	00	10	(expression
				76	00	11	(IP liste paramètres effectifs procédure (Instruction)
				76 A,B	00	12	(FI liste paramètres effectifs fonction (Indicateur)
)	76	03	13				
SI	76	00	34 →	76	00	34	SI-I (instruction)
				76 A,B,D	00	35	SI-E (expression)
ALORS	76	03	35 →	76	01	36	SI-ALORS-I (instruction)
				76 A,B,D	00	36	SI-ALORS-E (expression)
SINON	76	04	36 →	76	01	37	SI-ALORS-SINON-I (Instruction)
				76 A,B,D	03	40	SI-ALORS-SINON-E (expression)

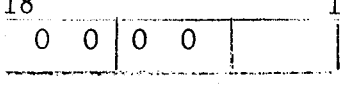
CHAINE CODEE					STACK			
POUR	76	00	50		76	00	50	POUR
PAS	76	03	51	→	76	00	51	POUR-PAS
JUSQUA	76	03	52	→	76	00	52	POUR-PAS-JUSQUE
TANTQUE	76	03	53	→	76	00	53	POUR-TANTQUE
FAIRE	76	03	54	→	76	01	54	POUR-FAIRE
,	76	03	60					
:	76	03	61					(étiquetage uniquement)
<	76	00	70					ouverture de chaîne
>	76	01	70					fermeture de chaîne

Les opérateurs arithmétiques, de relations, et logiques, ont le même code en chaîne codée qu'en stack (cf. annexe ch. 12)

Informations données par la procédure TYPE sur l'unité syntaxique examinée en chaîne codée

Cadrage des informations

numéro de catégorie A1



cadrage à droite

type XTYP		0 0 | 0 0	cadrage à gauche
Encombrement XENC		0 0 | 0 0	" "
Marque Paramètre formel	XPF	0 0 | 0 0	" "

Codage des informations

XPF
- 040000_8 (d_{15}) si paramètre formel
- 000000_8 si non paramètre formel

XTYP
- 200000 TYPA type Arithmétique
- 100000 TYPB type Booléen
- 000000 TYPD type Désignation

XENC
- 000000 Encombrement de Réel
- 020000 Encombrement d'entier sauf procédure-fonction
- 400000 Encombrement d'entier de Procédure-fonction

Remarques 1) pour une question de facilité de test on attribue les encombrements 000000 et 400000 respectivement aux étiquette et aiguillage (voir page suivante)

2) Procédures et Fonctions standard dans la chaîne codée le profil quatitatif est 00 (d_{18}-d_{15}), et le profil quantitatif et l'adresse dans une table.
Cette table (ou table des adresses de chargement futures des procédures standard) contient en début de génération 3 digits distinctifs d_{18}-d_{17}-d_{16} qui indiquent la catégorie de la procédure standard.

Ils seront effacés à la fin de la génération, et permettent à TYPE de donner les mêmes informations que pour une procédure déclarée dans XENC et XTYP.

d_{18}	d_{17}	d_{16}		
1	0	1	FSR	Fonction Standard Réel
0	1	1	FSE	" " Entier
0	0	1	FSB	" " Booléen
1	1	0	PS	Procédure Standard

Résultat de TYPE en fonction de l'unité syntaxique examinée : Résumé

Résultat de TYPE en fonction de l'unité syntaxique examinée :

Outre la marque paramètre formel éventuelle dans XPF, les valeurs affectées par TYPE sont les suivantes :

Chaîne codée partie quantitative d_{18}-d_{15} et représentation symbolique	A1 (1)	XTYP (2)	XENC (2)	Unité syntaxique examinée en chaîne codée
40 VR	0	2	00	Nombre Pur Réel
60 VE		2	02	" Entier
20 VL		1	02	" Booléen
+04 QR	1	2	00	Variable Simple Réel
±10 QE		2	02	" Entier
±14 QB		1	02	" Booléen
+06 TR	2	2	00	Identificateur de Tableau Réel
±12 TE		2	02	" Entier
±16 TB		1	02	" Booléen
+24 PR	3	2	00	Procédure-Fonction Réel
±20 PE		2	40	(déclarée) Entier
±34 PB		1	40	Booléen
00	4	2	00	Fonction Standard Réel
00		2	40	" Entier
00		1	40	" Booléen
00	5	//////	//////	Procédure Standard
+22 PO	6	//////	//////	Procédure (déclarée)
±26 ET	7	0	00	Etiquette
+32 AI	8	0	02	Aiguillage
-02 CH	9	/////	//////	Identificateur de Chaîne (pf)
76 SB	10	/////	//////	Symbole de base (opérateur)

1) d_{18} est représenté par ± lorsqu'il indique paramètre formel ou non paramètre formel, marque que TYPE met dans XPF ; code en octal.

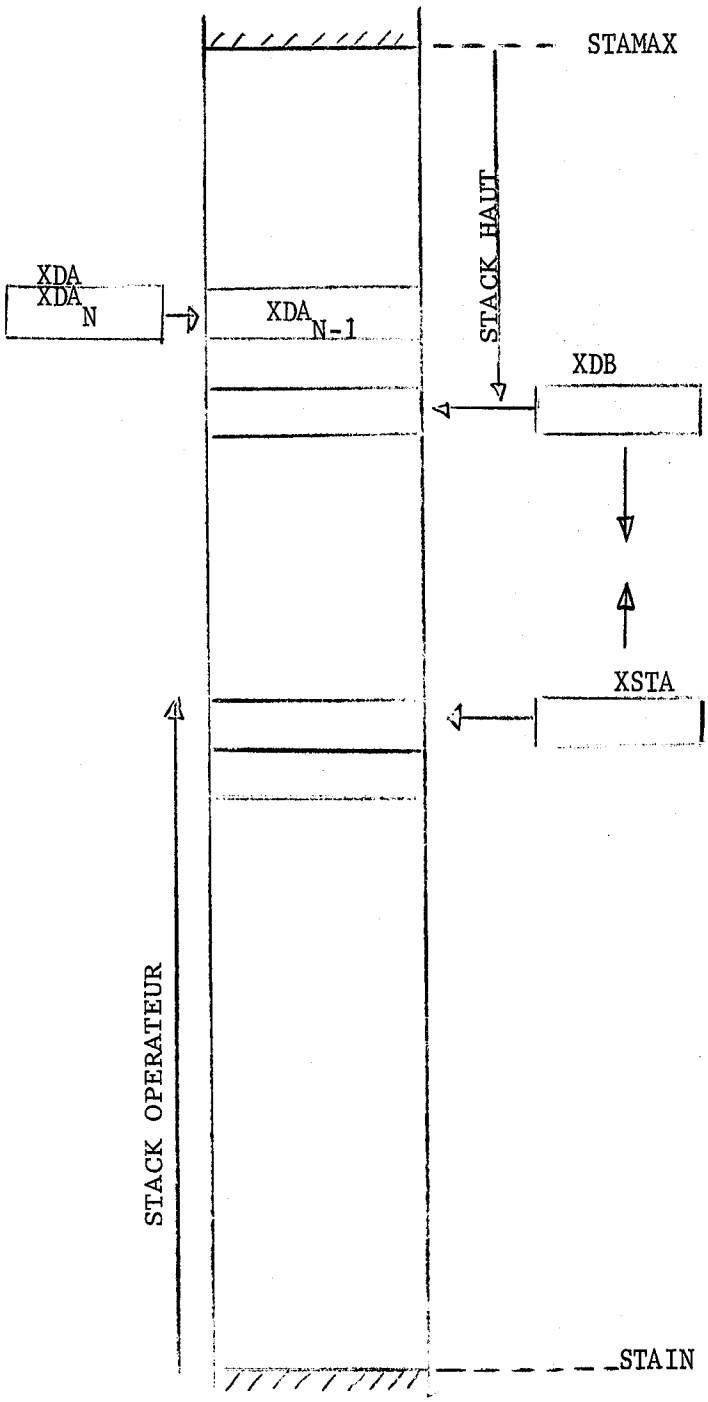
2) ////// indique sans signification.

Les 0 à gauche ne sont pas écrits, et le code est donné en octal.

13.2 PILES

Le générateur utilise un tableau de longueur fixe (compris entre STAIN et STAMAX) pour 2 piles distinctes croissant l'une vers l'autre.

La pile inférieure est la pile opérateur dite STACK (opérateur) et est utilisée pour l'empilage des opérateurs ou délimiteurs. Elle est contrôlée par l'index XSTA qui pointe le sommet occupé de la pile. Le stack opérateur se développe dans le sens des adresses croissantes.



La pile supérieure est dite "stack haut" et se développe dans le sens des adresses décroissantes. Son sommet (occupé) est indiqué par l'index XDB. Un second index XDA contrôle cette pile ; il repère l'origine du niveau courant dans cette pile (la première valeur empilée en entrant dans un niveau est la valeur précédente de XDA). En effet cette pile est utilisée pour empile les caractérisations statiques que l'on constitue en traitant une liste de paramètres effectifs : la récursivité impose de considérer des niveaux.

On aurait pu supprimer cette pile, en empilant les caractérisations dans le stack opérateur. Mais des considérations pratiques justifient un empilement distinct, et la mémoire est utilisée sans perdre de place par rapport à une pile unique. Le test de blocage entre XDA et XDB ne perd ici pas plus de temps qu'un test de dépassement ordinaire.

ANNEXE Ch. XIV

A-XIV-1

I - Restrictions du langage, Remarques

Les numéros indiqués sont ceux des paragraphes de [I]

2.1. Les lettres majuscules ou minuscules sont équivalentes.

2.3. Commentaire après 'FIN' ;
'FIN' < toute séquence ne contenant pas ' ou ; > équivalent à 'FIN'

Remarques

1) Dans un commentaire introduit par 'COMMENTAIRE' tout caractère = '(est signalé.
2) Un symbole 'COMMENTAIRE' ne suivant pas ; ou 'DEBUT' est signalé -il est traité comme s'il était précédé de ;
3) Le commentaire suivant le dernier symbole 'FIN' du programme peut contenir n'importe quel caractère (traité suivant le 1°). Il s'arrête sur la marque fin de Ruban (ℱ)
4) La longueur d'un commentaire ne doit pas excéder N caractères (N fixé à 120 par exemple).

2.4.1. Un identificateur ne doit pas comporter plus de 12 caractères et seuls les 6 premiers sont significatifs.

2.5. Nombre - se reporter pour les limitations au chapitre 3

2.8. Valeur et Type - idem

3.2. <délimiteur de paramètre> ::= ,

La , est le seul délimiteur de paramètre autorisé dans un indicateur de fonction.

3.5.

<Etiquette> ::= <identificateur>

Les étiquettes numériques sont interdites.

4.1.

Le programme ne peut être étiqueté.

4.6.6. Remarques :

Les étiquettes d'une instruction composée contrôlée par un POUR sont localisées, par le compilateur, dans l'instruction composée. (ch. 8)

4.7.1.

<délimiteur de paramètre> ::= ,

Dans une instruction procédure le seul délimiteur de paramètres autorisé est ,

4.7.3.1. Appel par valeur

Remarque Les paramètres appelés par valeur sont calculés dans l'ordre de la liste. On en déduira les effets de bord.

4.7.5.3.

Les tableaux ne peuvent être appelés par valeur [cette restriction peut être levée aisément, si un utilisateur particulier le juge indispensable]

4.7.5.5. Tout paramètre formel doit être spécifié.

Remarques.

1) correspondance paramètre formel-paramètre effectif. Un paramètre effectif procédure ne doit pas correspondre à un paramètre formel spécifié fonction.

2) Paramètres arithmétiques-Conversion de type réel, entier :

Dans le corps de procédure, chaque fois qu'est utilisée la valeur d'un paramètre, la valeur du paramètre effectif origine est éventuellement convertie pour s'accorder au type spécifié du paramètre formel.

Chaque valeur affectée dans le corps de procédure, à un paramètre qui ne peut être alors qu'une variable, est éventuellement convertie avant rangement, pour correspondre au type de la variable paramètre effectif origine.

Paramètre effectif-origine.

Appel par nom

a) le paramètre effectif n'est pas déjà formel :

Le paramètre effectif-origine est le paramètre effectif donné dans l'instruction procédure.

b) le paramètre effectif est déjà formel :

le paramètre effectif-origine est le paramètre effectif-origine du paramètre formel utilisé en paramètre effectif de l'instruction procédure.

Appel par valeur : Le paramètre effectif-origine est la variable locale du bloc-procédure (cf. ch. 2 et 9). Lorsqu'un tel paramètre formel est utilisé comme paramètre effectif on déduira facilement les conséquences d'après l'ensemble de ce 2°.

3) Constante en paramètre effectif.

Le paramètre est alors considéré comme appelé par valeur, qu'il figure ou non dans la partie valeur de la déclaration de procédure activée. On en déduira que si le paramètre n'apparaît pas en partie valeur de la tête de procédure, et que par exemple une valeur lui soit affectée dans le corps de procédure, le programme se déroulera "correctement" bien que le paramètre effectif soit une constante.

Un paramètre effectif est une constante s'il est VRAI,FAUX, un nombre positif ou sans signe.

5. Le déclarateur OWN n'existe pas ; il n'y a pas de variable rémanente.

5.2. Déclaration de Tableau

Remarques sur 5.2.4.2.

La présence dans l'expression d'une borne, d'un identificateur déclaré dans le bloc courant, n'est signalée par le compilateur que lorsque la déclaration de cet identificateur est écrite dans le programme source avant la déclaration de tableau.

On veillera particulièrement à ne pas écrire:

```
DEBUT                          ||   DEBUT
                               ||
DEBUT TABLEAU T [1:N]          ||   DEBUT REEL N ;
      REEL N ;                 ||        TABLEAU T [1:N] ;
                               ||
FIN                            ||   FIN
FIN                            ||   FIN
```

Ces 2 programmes équivalents, sont faux ; dans le premier cas l'erreur n'est pas signalée. L'exécution d'un tel programme ne peut être qu'aberrante.

Cependant on notera que le compilateur permet d'écrire des programmes interdits en Algol, et dont l'exécution a un sens évident. Nous les considérons corrects. Exemple :

```
DEBUT  TABLEAU T [P(N) : M] ;
 ENTIER N ; REEL M ;
ENTIER PROCEDURE P (X) ; ENTIER X ;
DEBUT LIRE(N) ; P := N ; LIRE(M) FIN ;
-----------
-----------
FIN
```

5.3. Déclaration d'Aiguillage.

Remarque sur 5.3.5.

Le sens est précisé par la phrase supplémentaire : "le bloc, en tête duquel est déclaré un aiguillage, doit appartenir à la portée de toute quantité entrant dans une expression de désignation de la liste d'Aiguillage".

D'autre part le compilateur localise les étiquettes d'une instruction composée contrôlée par un POUR, dans cette instruction composée.

Cela amène en ce qui concerne les Aiguillages une restriction ;

Exemple :

```
DEBUT    AIGUILLAGE AIG := E1, E2, E3 ; ....
   ....
   POUR I := A, B, C FAIRE
      DEBUT E2 : IP ; SI BOOL ALØRS ALLERA AIG [J] ; E3 : FIN ;
      ....
E1 : I ;
FIN
```

AIG est déclaré dans un bloc qui n'appartient pas à la portée de E2 et E3 ; ce programme est interdit.

Il ne le serait évidemment pas si l'instruction composée n'était pas contrôlée par un POUR.

(On notera qu'il suffit de changer un mot du compilateur pour supprimer la localisation des étiquettes dans le cas d'une instruction composée contrôlée par un POUR. La restriction précédente disparait, mais on perdra la vérification décrite à propos de 4.6.6.).

5.4. Déclaration de procédure

<délimiteur de paramètre> ::= , |) <identificateur> : (

de 120 caractères

5.4.5. Tous les paramètres formels doivent être spécifiés.

II - Restriction de Dépassement

La lettre P (paramètre) indiquera des limites qu'il est facile de modifier en changeant une constante

A Dans le programme nombre global de

1) déclarations de procédure: 512
2) déclarations d'aiguillage : 512
3) étiquettes : 512

B Nombre maximum de blocs (ou niveaux) imbriqués : 16

Nombre maximum de symboles DEBUT et PROCEDURE imbriqués : 24 (P)

Maximum de nombres non entiers : (P)

C Maximum d'entiers sans signe dans le programme : 20 (P)

D Maximum d'identificateurs de variable simple, de tableau et de paramètre formel dans un niveau:

En comptant 1 pour un identificateur de variable simple entière, ou de Tableau,

2 pour un identificateur de variable simple réelle,

3 pour un paramètre formel ,

Le maximum de 507 ne doit être dépassé.

E Maximum d'identificateurs utilisables à un endroit donné (P).

Les restrictions précédentes sur le nombre d'identificateurs sont globales soit pour le programme (procédure, étiquette, aiguillage) soit pour un niveau (autres identificateurs).

Indépendamment, la longueur de la table des identificateurs utilisée à la compilation impose une restriction fonction du nombre n de niveaux imbriqués, de d nombre d'identificateurs locaux à chaque niveau, et du taux k d'identificateurs globaux pour un niveau et utilisés à ce niveau.

En supposant d et k constants pour chaque niveau la longueur de table occupée, à la profondeur d'imbrication n est ;

$$Ln = \begin{cases} \dfrac{n - (n+1)k + k^{n+1}}{(1-k)^2} \cdot d & \text{si } k < 1 \\ \dfrac{n\,(n+1)}{2}\, d & \text{si } k = 1 \end{cases}$$

La longueur actuelle de table est 500 (1500 mémoires). (P)

Le tableau suivant donne le maximum d'identificateurs locaux par niveau, de façon que la compilation ne se bloque pas pour n niveaux imbriqués dans les hypothèses précédentes (k et d constants).

n \ k	0	0,25	0,50	1
4	125	100	81	50
5	100	80	62	33
8	62	49	35	13
16	31	24	17	4

F.Le degré de complexité des instructions et expressions est limitée à la compilation par la longueur du "Stack opérateurs, actuellement de 120 (P).

G.Restrictions sur la dimension du programme.

La longueur du programme objet est d'environ 0,7 à 1fois le nombre des unités syntaxiques définies par <Identificateur> <Nombre sans signe> <valeur logique> <délimiteur> .

A noter que les chaînes propres n'interviennent pas dans ce comptage ; chacune augmente, à raison de 1 mot par 3 caractères, la longueur effective du programme objet.

Celle ci ne peut en aucun cas dépasser la partie du haut de la mémoire, qu'occupent :

1) l'interpréteur 3500 mots

2) les procédures standard appelées dans le programme et qui seules seront chargées.

On peut donner les restrictions statiques suivantes :

Compilateur	Mémoire machine	Programme Objet
version 8k (*)	8000 mots	maximum d'environ 3500 mots
	16000 mots	maximum = 8000 mots
version 16k (**)	16000 mots	maximum d'environ 11000 mots
	24000 mots	maximum 16000 mots

(*) Version écrite.

(**) Cette version ne diffèrera que par le codage de la partie adresse de 4 ordres.

H.Restrictions d'Exécution.

L'occupation de la zone de travail évolue durant l'exécution, et les dépassements les plus fréquents seront provoqués par les réservations en particulier de Tableau.

Débordement (voir ch. 3)

Une valeur entière doit être comprise entre -131072 et 131072.
Une valeur réelle doit être comprise

a) entre -10^{51} et 10^{51} dans un calcul intermédiaire
b) entre -10^{38} et 10^{38} s'il est affecté à une variable (simple ou indicée).

EXEMPLES DE PROGRAMMES OBJETS

Ces programmes n'ont aucune valeur sémantique, et illustrent simplement le langage intermédiaire décrit dans cet ouvrage.

Nota.

Les procédures standard citées dans les exemples suivants sont : EXL, EXE, EXD, IMPR. Les 3 premières permettent de composer une "ligne de sortie" dans une zone tampon dont la sortie sur imprimante (machine à écrire ici) est déclenchée par IMPR. EXL place dans la "ligne" une chaîne alpha-numérique (libellé), où le caractère ! désigne le caractère retour chariot. EXE et EXD placent les valeurs d'une liste d'expressions respectivement sous forme entière et décimale ; leur premier paramètre effectif donne la dimension de la zone affectée à chaque valeur ; dans le cas de EXD le second paramètre indique le nombre de chiffres après la virgule. Il existe d'autres procédures de ce type.

```
000 0000 'DEBUT' 'COMMENTAIRE' EXPRESSIONS ET INSTRUCTIONS D AFFECTATIONS;
001 0000 'ENTIER' I,J; 'REEL' A,B,C; 'BOOLEEN' L,M,N;
002 0020 A : =1.1;
003 0024 B : =A*2;
004 0030 N : ='FAUX';
005 0034 L : =M : = 'VRAI';
006 0040 I : =J : =A + B;
007 0048 C : = A + B*C;
008 0056 'DEBUT' 'REEL' D,E,F ;'BOOLEEN' P;
009 0067 E:=C/A;
010 0073 D : = 'SI' A+B 'INF' CIE 'ALORS' A 'SINON' A/B+C;
011 0092 F : = A+B*('SI' M 'ALORS' -A 'SINON' 2*A);
012 0110 P : = C 'INF' B 'ET' F*D 'SUP' AIE;
013 0124 'FIN';
014 0126 'DEBUT' 'REEL' G,H;
015 0132 G : = 'SI' M 'ALORS' 1 'SINON' 'SI' L 'ALORS' 2 'SINON' 3;
016 0146 'DEBUT' 'ENTIER' K; 'BOOLEEN' S;
017 0153 H : = K : = G+(A-B*C)/(A+B);
018 0173 S := L 'OU' ('SI' N 'ALORS' 'NON' M 'SINON' L 'IMPL' N);
019 0189 'FIN'
020 0190 'FIN'
021 0191 'FIN'#
CODIFICATION TERMINEE
```

```
GENERATION TERMINEE

000002    00200   } Nombres Entiers
000001
000003
400000
400000    00204
400000
400000
400000
400000    00210
400000
400000
400000
400000    00214
400000
400000
400000
400000    00220
400000
400000
400000
214631    00224   } Nombre Reel
463201
*L* 00000
*L* 00001
   EBL    00230   Entrée Bloc Indicateur de réservation
 000020
*L* 00002
ARN 0 007
CRS 00224 00234
   RAN
*L* 00003
ARN 0 009         B
CRN 0 007 00240   A
CES 00200         2
   MUL            x
   RAN            :=
*L* 00004 00244
AEN 0 015
CES 00025         FAUX  (1 en 25)
   RAN
*L* 00005 00250
AEN 0 013
AEN 0 014
CES 00027         VRAI  (0 en 27)
   RAM    00254
   RAN
*L* 00006
AEN 0 005         I
AEN 0 006 00260   J
CRN 0 007         A
ADR 0 009         +B
   RAM            :=M
   RAN    00264   :=
*L* 00007
ARN 0 011
CRN 0 007
CRN 0 009 00270
MLR 0 011
   ADD
   RAN
*L* 00010 00274
   EBL
 000014
*L* 00011
ARN 1 007 00300
CRN 0 011
DVR 0 007
   RAN
*L* 00012 00304
ARN 1 005         D
CRN 0 007         A
ADR 0 009         +B
CRN 0 011 00310   C
EXR 1 007         ↑E
   INF            <
faux TST 00316
CRN 0 007 00314   A
SAU 00321 ──────────┐
CRN 0 007         A │
DVR 0 009         /B│
ADR 0 011 00320   +C│
   RAN            :=◄┘
*L* 00013
ARN 1 009
CRN 0 007 00324
CRN 0 009
CEN 0 014
TST 00333
CRN 0 007 00330
   ISI
SAU 00335
CES 00200
MLR 0 007 00334
   MUL
   ADD
   RAN
*L* 00014 00340
```

```
AEN 1 011
CRN 0 011
CRN 0 009
   INF      00344
CRN 1 009
MLR 1 005
CRN 0 007
EXR 1 007   00350
   SUP
   ET
   RAN
*L* 00015   00354
   SBL
*L* 00016
   EBL
 000011     00360
*L* 00017
ARN 1 005
CEN 0 014
TST 00367   00364
CES 00201
SAU 00374
CEN 0 013
TST 00373   00370
CES 00200
SAU 00374
CES 00202
   RAN      00374
*L* 00020
   EBL
 000007
*L* 00021   00400
ARN 1 007
AEN 2 005
CRN 1 005
CRN 0 007   00404
CRN 0 009
MLR 0 011
   SUB
CRN 0 007   00410
ADR 0 009
   DIV
   ADD
   RAM      00414
   RAN
*L* 00022
AEN 2 006
CEN 0 013   00420
CEN 0 015
TST 00426
CEN 0 014
   NON      00424
SAU 00431
CEN 0 013
CEN 0 015
   IMP      00430
   OU
   RAN
*L* 00023
   SBL      00434
*L* 00024
   SBL
*L* 00025
   SBL      00440
   STP
```

PROGRAMME TERMINE

```
          000 0000 'DEBUT' 'COMMENTAIRE' ETIQUETTE INSTRUCTION ALLER A;
001 0000 'REEL' A,B; B : = 1; A : = 2;
002 0014 E1:  B : = B*A;
003 0022 'ALLER A' E4;
004 0025 E2 : 'DEBUT' 'ENTIER' I; I : = A-B/2;
005 0039 'ALLER A' 'SI' B 'INF' A +I 'ALORS' E1 'SINON' E3;
006 0051 E3: 'FIN';
007 0055 B:=1.5*B;E4 : 'SI' B 'INF' 10 'ALORS' 'ALLER A' E2;
008 0071 'FIN'
009 0072 #
CODIFICATION TERMINEE


GENERATION TERMINEE

 000001     00200
 000002
 000012
 400000
 400000     00204
 400000
 400000
 400000
 400000     00210
 400000
 400000
 400000
 400000     00214
 400000
 400000
 400000
 400000     00220
 400000
 400000
 400000
 300000     00224
 000201
 000245     (E1)
 000255     (E2)
 040301     (E3)00230
 000307     (E4)
*L* 00000
*L* 00001
   EBL      00234
 000011
ARN 0 007
CES 00200
   RAN      00240
ARN 0 005
CES 00201
   RAN
*L* 00002   00244
ARN 0 007
CRN 0 007
MLR 0 005
   RAN      0250

*L* 00003
STA 00231          E4
   ALL             ALLER A
*L* 00004   00254
   EBL
 000006
AEN 1 005
CRN 0 005   00260
CRN 0 007
CES 00201
   DIV
   SUB      00264
   RAN
*L* 00005
CRN 0 007          B
CRN 0 005   00270  A
ADE 1 005          +I
   INF             <
FAUX TST 00276
   STA 00226  00274  E1
   SAU 00277
   STA 00230       E3
   ALL             ALLER A
*L* 00006   00300
   SBL
*L* 00007
ARN 0 007
CRS 00224   00304
MLR 0 007
   RAN
CRN 0 007
CES 00202   00310
   INF
TST 00315
STA 00227
   ALL      00314
*L* 00010
   SBL
   STP


PROGRAMME TERMINE
```

Table Etiquettes

```
000 0000 'DEBUT' 'COMMENTAIRE' INSTRUCTION SI, INSTRUCTION COMPOSEE;
001 0000 'ENTIER' I,J; 'REEL' A,B,C; 'BOOLEEN' BB;
002 0016 I := 1; J := 2+J; BB := I 'SUP' J;
003 0032 'SI' I 'EGAL'J 'ALORS' A:=B:=I;
004 0043 'SI' BB 'ALORS' 'DEBUT' 'SI' A 'DIF' 10 'ALORS' C := A 'SINON' C := A+I 'FI
005 0062 'SINON' 'SI' J 'SUG' I 'ALORS' B := A-J 'SINON'A:= (A+I)IJ;
006 0084 'FIN'
007 0085 #
CODIFICATION TERMINEE

GENERATION TERMINEE

 000001     00200
 000002
 000012
 400000
 400000     00204
 400000
 400000
 400000
 400000     00210
 400000
 400000
 400000
 400000     00214
 400000
 400000
 400000
 400000     00220
 400000
 400000
 400000
*L* 00000   00224
*L* 00001
   EBL
 000016
*L* 00002   00230
AEN 0 005
CES 00200
   RAN
AEN 0 006   00234
CES 00201
ADE 0 006
   RAN
AEN 0 013   00240
CEN 0 005
CEN 0 006
   SUP
   RAN      00244
*L* 00003
CEN 0 005
CEN 0 006
   EGA      00250
TST 00257
ARN 0 007
ARN 0 009
CEN 0 005   00254
   RAM

   RAN
```

```
*L* 00004
CEN 0 013   00260    B
TST 00300
CRN 0 007
CES 00202
   DIF      00264
TST 00272
ARN 0 011
CRN 0 007
   RAN      00270
SAU 00276
ARN 0 011
CRN 0 007
ADE 0 005   00274
   RAN
*L* 00005
SAU 00316
CEN 0 006   00300   I
CEN 0 005           J
   SUG              >
TST 00311
ARN 0 009   00304
CRN 0 007
SBE 0 006
   RAN
SAU 00316   00310
ARN 0 007
CRN 0 007
ADE 0 005
EXE 0 006   00314
   RAN
*L* 00006
   SBL
   STP      00320
```

Instruction Composée

PROGRAMME TERMINE

Programme erroné : valeur de J non définie--- ceci n'est pas détecté ; l'exécution peut avoir lieu sans incident selon l'état de la machine.

```
000 0000 'DEBUT' 'COMMENTAIRE' DECLARATION DE TABLEAU, VARIABLE INDICEE;
001 0000 'ENTIER' I,J; 'TABLEAU' T.(1:10).; 'REEL' A,B;
002 0020 T.(1). := 2;
003 0027 A := 2*T.(1).;
004 0036 I := J := T.(A).:= 1;
005 0047 'DEBUT' 'ENTIER' 'TABLEAU' TA,TB .('SI' A 'SUP' 4 'ALORS' I+J 'SINON' I-J:A
006 0073 'BOOLEEN' 'TABLEAU' TC, TE, TF.(1:10)., TG.(A:20,I:50).;
007 0100 TG .(A+1,J*A).:= T.('SI' A 'EGAL' 2 'ALORS' 1 'SINON' A). 'SUP' 4;
008 0125 'FIN'
009 0126 'FIN'
010 0127 #
CODIFICATION TERMINEE
```

GENERATION TERMINEE

```
 000001    00200
 000012
 000002
 000004
 000024    00204
 000062
 400000
 400000
 400000    00210
 400000    00260
 400000
*L* 00000
*L* 00001
   EBL     00264
 000014
CES 00200          Evaluation d'une borne
CES 00201          "
   TAB     00270   Ordre de Creation de Tableau
 000002            2 bornes
 060007            Adresse C attache au 1er Tableau
 000001            1 Tableau
*L* 00002  00274
ARN 0 007
CES 00200
   IND
CES 00202  00300
   RAN
*L* 00003
ARN 0 003          A
CES 00202  00304   2
ARN 0 007          CT
CES 00200          1                  } T[1]
   PVA             Valeur Variable Indicee
   MUL     00310   X
   RAN             :=
*L* 00004
AEN 0 005          I
AEN 0 006  00314   J
```

```
ARN 0 007          CT
CRN 0 008          A                  } T[A]
   IND             Adresse Variable Indicee
CES 00200  00320   1
   RAM             :=M
   RAM             :=M
   RAN             :=
*L* 00005  00324
   EBL
 000013
CRN 0 008
CES 00203  00330
   SUP
TST 00336
CEN 0 005
ADE 0 006  00334
SAU 00340
CEN 0 005
SBE 0 006
CRN 0 008  00340
EXE 0 006
   TAB
 000002
 121005    00344
 000002
*L* 00006
CES 00200
CES 00201  00350
   TAB             → 1 ordre par section de Tab
 000002
 161007
 000003    00354
CRN 0 008
CES 00204
CEN 0 005
CES 00205  00360
   TAB             →  "
 000004
 161012
 000001    00364
~~*L* 00007~~
~~CRN 0 008~~
```

```
*L* 00007
AEN 1 010
CRN 0 008
ADE 0 005   00370
CEN 0 006
MLR 0 008
   IND
ARN 0 007   00374
CRN 0 008
CES 00202
   EGA
TST 00403   00400
CES 00200
SAU 00404
CRN 0 008
   PVA      00404
CES 00203
   SUP
   RAN
*L* 00010   00410
   SBL
*L* 00011
   SBL
   STP      00414
```

RAMME TERMINE

```
000 0000 'DEBUT' 'COMMENTAIRE' INSTRUCTIONS POUR 1 ET 2;
001 0000 'ENTIER' I,J; 'TABLEAU' T.(1:10, 1:10).; 'REEL' A,B;
002 0024 A :=2; B :=6;
003 0032 'POUR' I:= 1,2 'FAIRE' T.(I,B).:=A+I;
004 0050 'POUR' I:= 3 'TANT QUE' B 'ING' 10 'FAIRE'
005 0059 'DEBUT' T.(I,B).:= B*I; B :=B+1 'FIN';
006 0078 'POUR' I := 4,5,6 'FAIRE'
007 0087 'POUR' J:= I 'TANT QUE' A+B 'ING' 20 'FAIRE'
008 0098 'DEBUT' T.(I,J).:= A*B; A:=1+A 'FIN';
009 0117 'POUR' I:= (A+B)/2*B, B*A 'TANT QUE' A12 'INF' 50+1,15
010 0142 'FAIRE' 'DEBUT' J:= I!A; A:=A+1 'FIN';
011 0158 'FIN'
012 0159 #
CODIFICATION TERMINEE

GENERATION TERMINEE

000001    00200
000012
000002
000006
000003    00204
000004
000005
000024
000062    00210
000017
400000
400000

  400000    00260
  400000
*L* 00000
*L* 00001
   EBL      00264
  000021
 CES 00200
 CES 00201
 CES 00200  00270
 CES 00201
    TAB
  000004
  060007    00274
  000001
*L* 00002
 ARN 0 008
 CES 00202  00300
    RAN
 ARN 0 010
 CES 00203
    RAN     00304
*L* 00003
 SAU 00310
  000324    Adresse Instruction "controlée"
 AEN 0 005  00310  I
 CES 00200         1
    RAN            :=

    DO
  000307    00314   } FAIRE
                    Adresse Indirecte de l'instruction
```

```
  000316    }   Adresse sucesseur
 AEN 0 005      I
 CES 00202      2
    RAN     00320 :=
    DO
  000307    } FAIRE
  000336
    SUP     00324 → Adresse dynamique de la mémoire locale où l'on preserve l'adresse de retour
 ARN 0 007
 CEN 0 005
 CRN 0 010
    IND     00330                    } Instruction "Controlee"
 CRN 0 008
 ADE 0 005
    RAN
    RTP     00334 } ordre de RETOUR
    SUP
*L* 00004
 SAU 00341
  000353    00340
 AEN 0 005      I
 CES 00204      3
    RAN         :=
 CRN 0 010  00344 B
 CES 00201      10
    ING         ≤
 TST 00373
    DO      00350 } FAIRE
  000340
  000341
    DIF
*L* 00005   00354
 ARN 0 007
 CEN 0 005
 CRN 0 010
    IND     00360
 CRN 0 010
 MLE 0 005
    RAN
 ARN 0 010  00364
```

```
CRN 0 010
CES 00200
   ADD
   RAN      00370
   RTP
   DIF
*L* 00006
SAU 00376   00374
 000420
AEN 0 005
CES 00205
   RAN      00400
   DO
 000375
 000404
AEN 0 005   00404
CES 00206
   RAN
   DO
 000375     00410
 000412
AEN 0 005
CES 00203
   RAN      00414
   DO
 000375
 000460
   NON      00420
*L* 00007
SAU 00424
 000437
AEN 0 006   00424
CEN 0 005
   RAN
CRN 0 008
ADR 0 010   00430
CES 00207
   ING
TST 00456
   DO       00434
 000423
 000424
   ET
*L* 00010   00440
ARN 0 007
CEN 0 005
CEN 0 006
   IND      00444
CRN 0 008
MLR 0 010
   RAN
ARN 0 008   00450
CES 00200
ADR 0 008
   RAN
   RTP      00454
   ET
   RTP
   NON
*L* 00011   00460
SAU 00463
 000522
AEN 0 005
CRN 0 008   00464
ADR 0 010
CES 00202
   DIV
MLR 0 010   00470
   RAN
   DO
 000462
 000475     00474
AEN 0 005
CRN 0 010
MLR 0 008
   RAN      00500
CRN 0 008
CES 00202
   EXP
CES 00210   00504
ADE 0 005
   INF
TST 00513
   DO       00510
 000462
 000475
AEN 0 005
*L* 00012   00514
CES 00211
   RAN
   DO
 000462     00520
 000536
   OU
AEN 0 006
CEN 0 005   00524
EXR 0 008
   RAN
ARN 0 008
CRN 0 008   00530
CES 00200
   ADD
   RAN
   RTP      00534
   OU
*L* 00013
   SBL
   STP      00540
```

```
000 0000 'DEBUT' 'COMMENTAIRE' INSTRUCTIONS POUR 1E2-VARIABLES INDICEES;
001 0000 'TABLEAU' T.(1:10).; 'ENTIER' A,B,C;
002 0017 A :=2; B:=1;
003 0025 'POUR' T.(A+B).:= A*B, A-B 'FAIRE'
004 0041 'DEBUT' T.(A+B+1).:= C:= T.(A+B).; A:=2+C 'FIN';
005 0067 'POUR' T.(A). := C*A  'TANT QUE' B 'INF' 10 'FAIRE'
006 0081 'DEBUT' C := B*A; B:= T.(A).+2 'FIN';
007 0098 'FIN'
008 0099 #
CODIFICATION TERMINEE

GENERATION TERMINEE

 000001     00200
 000012
 000002
 400000
 400000     00204
 400000

 400000
*L* 00000
*L* 00001
   EBL      00264
 000013
CES 00200
CES 00201
   TAB      00270
 000002
 060005
 000001
*L* 00002   00274
AEN 0 006
CES 00202
   RAN
AEN 0 007   00300
CES 00200
   RAN
*L* 00003
SAU 00313   00304
 000331
ARN 0 005
CEN 0 006
ADE 0 007   00310
   IND
   RSP      ordre de Retour

SSP 00306
CEN 0 006   00314
MLE 0 007
   RAN
   DO
 000305     00320
 000322
SSP 00306
CEN 0 006
SBE 0 007   00324
   RAN

   DO
 000305
 000356     00330
   ING
*L* 00004
ARN 0 005
CEN 0 006   00334
ADE 0 007
CES 00200
   ADD
   IND      00340
AEN 0 008
ARN 0 005
CEN 0 006
ADE 0 007   00344
   PVA
   RAM
   RAN
AEN 0 006   00350
CES 00202
ADE 0 008
   RAN
   RTP      00354
   ING
*L* 00005
SAU 00365
 000400     00360
ARN 0 005
CEN 0 006
   IND
   RSP      00364
SSP 00361
CEN 0 008
MLE 0 006
   RAN      00370
CEN 0 007
CES 00201
   INF
TSE 00417   00374
   DO
 000360
 000365
```

Sous-programme de Calcul d'adresse T[A+B]

Activation du Sous-programme

```
   EGA      00400
*L* 00006
AEN 0 008
CEN 0 007
MLE 0 006   00404
   RAN
AEN 0 007
ARN 0 005
CEN 0 006   00410
   PVA
CES 00202
   ADD
   RAN      00414
   RTP
   EGA
*L* 00007
   SBL      00420
   STP
```

PROGRAMME TERMINE

```
'
000 0000 'DEBUT' 'COMMENTAIRE' INSTRUCTION POUR 3;
001 0000 'ENTIER' I,J; 'REEL' A,B,D,C;
002 0015 B :=I := 10; J:= C:=1.2;
003 0027 'POUR' A := B*C+B 'PAS' I 'JUSQUA' B12 'FAIRE'
004 0042 'DEBUT' D := A*B; B:= B-J 'FIN';
005 0056 'POUR' A := B*C 'PAS' I+J 'JUSQUA' B12 'FAIRE'
006 0071 'DEBUT' J:= (I+J)/2+J; D := A*B; B:= B-J 'FIN';
007 0097 'POUR' I := B*C 'PAS' 8 'JUSQUA' 512 'FAIRE'
008 0108 C := CII ;
009 0114 'FIN'
010 0115 #
CODIFICATION TERMINEE

GENERATION TERMINEE

 000012     00200
 000002
 000010
 001000
 400000     00204
 400000
 400000
 400000
 400000      210



 400000     00260
 400000
 231463
 146201
*L* 00000   00264
*L* 00001
   EBL
 000022
*L* 00002   00270
ARN 0 009
AEN 0 005
CES 00200
   RAM      00274
   RAN
AEN 0 006
ARN 0 013
CRS 00262   00300
   RAM
   RAN
*L* 00003
SAU 00306   00304
 000331
ARN 0 007          A
CRN 0 009          B
MLR 0 013   00310  ×C
ADR 0 009          +B
SAU 00317          Saut pour l'affectation initiale
ARN 0 007
   VAL      00314  A
                   Valeur avec conservation de l'adresse
```

```
CEN 0 005        I
   ADD           +
   RAM           :=M
CRN 0 009  00320 B   } Valeur "finale"
CES 00201        2
   EXP           ↑
   SUB           -
CEN 0 005  00324 A
TFA 00345        Test d'épuisement
   DO            } FAIRE
 000305          } Adresse de retour
 000313    00330 }
   ET
*L* 00004
ARN 0 011
CRN 0 007  00334
MLR 0 009
   RAN
ARN 0 009
CRN 0 009  00340
SBE 0 006
   RAN
   RTP
   ET      00344
*L* 00005
SAU 00350
 000375
ARN 0 007  00350
CRN 0 009
MLR 0 013
SAU 00363
CEN 0 005  00354 } Sous programme de
ADE 0 006        } calcul du pas
   RSP           }
ARN 0 007
   VAL     00360
SSP 00354
   ADD
   RAM
CRN 0 009  00364
CES 00201
   EXP
   SUB
SSP 00354  00370
```

Instruction

```
TFA 00420
   DO
 000347
 000357      00374
   OU
*L* 00006
AEN 0 006
CEN 0 005    00400
ADE 0 006
CES 00201
   DIV
ADE 0 006    00404
   RAN
ARN 0 011
CRN 0 007
MLR 0 009    00410
   RAN
ARN 0 009
CRN 0 009
SBE 0 006    00414
   RAN
   RTP
   OU
*L* 00007    00420
SAU 00423
 000443
AEN 0 005          I
CRN 0 009    00424 B
MLR 0 013          XC
   RAM             :=M
CES 00203          B
   ING       00430 ≤
TST 00442
   DO
 000422
 000435      00434
   FAI
 000005
 000010
 001000      00440
 000422
SAU 00453
   MP
*L* 00010    00444
ARN 0 013
CRN 0 013
EXE 0 005
   RAN       00450
   RTP
   JMP
*L* 00011
   SBL       00454
   STP
```

Affectation initiale et test d'épuisement

FAIRE (activation initiale)

Adresse de Retour

FAIRE-simplifie (autres activations)

Saut quand epuisement

Instruction

PROGRAMME TERMINE

```
'
000 0000 'DEBUT' 'COMMENTAIRE' INSTRUCTION POUR 3 VARIABLE INDICEE;
001 0000 'ENT' I,J;'TABLEAU' T.(1:100).; 'REEL' A,B,C;
002 0022 A := C:= 1; B:= I:=10; J:=1.2;
003 0038 'POUR' T.(A).:= B*C 'PAS' I 'JUSQUA' B12 'FAIRE'
004 0054 'DEBUT' T.(1+A).:= T.(A).; B:= B-J; A:= A+C 'FIN';
005 0080 'POUR' T.(A).:= B*C 'PAS' I+J 'JUSQUA' B12 'FAIRE'
006 0098 C:= T.(A). II;
007 0107 'FIN'
008 0108 #
CODIFICATION TERMINEE


GENERATION TERMINEE

 000001     00200
 000144
 000012
 000002
 400000     00204
 400000
 400000
 400000

 400000
 400000
 400000
 400000     0026(
 400000
 231463
 146201
*L* 00000   00264
*L* 00001
    EBL
 000020
CES 00200   00270
CES 00201
    TAB
 000002
 060007     00274
 000001
*L* 00002
ARN 0 008
ARN 0 012   00300
CES 00200
    RAM
    RAN
ARN 0 010   00304
AEN 0 005
CES 00202
    RAM
    RAN     00310
AEN 0 006
CRS 00262
    RAN
*L* 00003   00314
SAU 00323
 000345
ARN 0 007
CRN 0 008   00320
    IND
    RSP
SSP 00317
CRN 0 010   00324
MLR 0 012
SAU 00333
SSP 00317
    VAL     00330
CEN 0 005
    ADD
    RAM
CRN 0 010   00334
CES 00203
    EXP
    SUB
CEN 0 005   00340
TFA 00371
    DO
 000316
 000327     00344
    NON
*L* 00004
ARN 0 007
CES 00200   00350
ADR 0 008
    IND
ARN 0 007
CRN 0 008   00354
    PVA
    RAN
ARN 0 010
CRN 0 010   00360
SBE 0 006
    RAN
ARN 0 008
CRN 0 008   00364
ADR 0 012
    RAN
```

```
    RTP
    NON       00370
*L* 00005
SAU 00400
 000425
ARN 0 007     00374
CRN 0 008
    IND
    RSP
SSP 00374     00400
CRN 0 010
MLR 0 012
SAU 00413
CEN 0 005     00404
ADE 0 006
    RSP
SSP 00374
    VAL       00410
SSP 00404
    ADD
    RAM
CRN 0 010     00414
CES 00203
    EXP
    SUB
SSP 00404     00420
TFA 00437
    DO
 000373
 000407       00424
    ET-
*L* 00006
ARN 0 012
ARN 0 007     00430
CRN 0 008
    PVA
EXE 0 005
    RAN       00434
    RTP
    ET
*L* 00007
    SBL       00440
    STP
```

PROGRAMME TERMINE

```
000 0000 'DEBUT' 'REEL' A,B,C,D; 'ENTIER' I,J,K;
001 0017 'POUR' I:= 0 'PAS' 10 'JUSQUE' 100 'FAIRE'
002 0026 'POUR' J :=0 'PAS' 1 'JUSQUA' 10 'FAIRE' K:=I+J;
003 0041 'DEBUT' 'REEL' F,G,H; K:=0; A:=0;
004 0057 'POUR' I:= 0 'PAS' 10 'JUSQUE' 100 'FAIRE'
005 0066 E: 'DEBUT' 'REEL' U,V; K:=K+1;

ER 03 0066
006 0080 'POUR' J:= 0 'PAS' 1 'JUSQUE' 8 'FAIRE' A:=J+I;
007 0095 'SI' K 'INF' 3 'ALORS' 'ALLER A' E;
008 0103 'SI' K 'SUP' 5 'ALORS' 'ALLER A' EE;
009 0111 A:=A+1 'FIN' ;
010 0118 EE: F:=1
011 0122 'FIN';
012 0125 B:=1
013 0127 'FIN'
014 0129 #
CODIFICATION TERMINEE
```

Le constat signale l'étiquetage de l'instruction sur laquelle porte la proposition POUR

```
GENERATION TERMINEE

 000000    00200
 000012
 000144
 000001
 000010    0020
 000003
 000005
 040400
 040470    00210
 400000
 400000
 400000

 400000
 400000
 400000
 400000    00254
 400000
 400000
 400000
 400000    00260
 400000
*L* 00000
   EBL
 000022    00264
*L* 00001
SAU 00270
 000307
AEN 0 013  00270

CES 00200
   RAM
CES 00202
   ING     00274
TST 00306
   DO
 000267
 000301    00300
   FAI
 000015
 000012
 000144    00304
 000267
SAU 00343
   OU
*L* 00002  00310
SAU 00313
 000332
AEN 0 014
CES 00200  00314
   RAM
CES 00201
   ING
TST 00331  00320
   DO
 000312
 000324
   FAI     00324
 000016
 000001
 000012
 000312    00330
SAU 00341
   IMP
AEN 0 015
CEN 0 013  00334
ADE 0 014
   RAN
   RTP
```

```
    IMP       00340
   RTP
   OU
*L* 00003
   EBL       00344
 000014
AEN 0 015
CES 00200
   RAN       00350
ARN 0 005
CES 00200
   RAN
*L* 00004    00354
SAU 00357
 000376
AEN 0 013
CES 00200    00360
   RAM
CES 00202
   ING
TST 00375    00364
   DO
 000356
 000370
   FAI       00370
 000015
 000012
 000144
 000356      00374
SAU 00467
   SUG
*L* 00005
   EBL       00400
 000012
AEN 0 015
CEN 0 015
CES 00203    00404
   ADD
   RAN
*L* 00006
SAU 00412    00410
 000431
AEN 0 014
CES 00200
   RAM       00414
CES 00204
   ING
TST 00430
   DO        00420
 000411
 000423
   FAI
 000016      00424
 000001
 000010
 000411
SAU 00440    00430
   ING
ARN 0 005
CEN 0 014
ADE 0 013    00434
   RAN
   RTP
   ING
*L* 00007    00440
CEN 0 015
CES 00205
   INF
TST 00447    00444
STA 00207
   ALL
*L* 00010
CEN 0 015    00450
CES 00206
   SUP
TST 00456
STA 00210    00454
   ALL
*L* 00011
ARN 0 005
CRN 0 005    00460
CES 00203
   ADD
   RAN
   SBL       00464
   RTP
   SUG
*L* 00012
ARN 1 005    00470
*L* 00013
CES 00203
   RAN
   SBL       00474
*L* 00014
ARN 0 007
*L* 00015
CES 00203    00500
   RAN
   SBL
   STP
```

PROGRAMME TERMINE

```
000 0000 'DEBUT''COM'POUR 3 SUITE;  'ENT'I; 'REEL'A,B,C;
001 0011 A:=3;B:=2;
002 0019 'POUR'I:=A/B'PAS'2 'JUSQU A'10*A'FAIRE' C:=3*I;
003 0038 'POUR'I:=A/B'PAS'A 'JUSQU A'10*A'FAIRE' C:=3*I;
004 0057 'FIN'#
CODIFICATION TERMINEE


GENERATION TERMINEE

 000003     00200
 000002
 000012
 400000
 400000     00204
 400000
 400000
 400000
 400000     00210
 400000
 400000

 400000
*L* 00000
   EBL
 000016     00214
*L* 00001
ARN 0 006
CES 00200
   RAN      00270
ARN 0 008
CES 00201
   RAN
*L* 00002   00274
SAU 00277
 000320
AEN 0 005
CRN 0 006   00300
DVR 0 008
SAU 00307
AEN 0 005
   VAL      00304
CES 00201
   ADD
   RAM
CES 00202   00310
MLR 0 006
   SUB
CES 00201
TFA 00327   00314
   DO
 000276
 000303
   SUP      00320
ARN 0 010
CES 00200
MLE 0 005
  RAN       00324
   RTP
   SUP
*L* 00003
SAU 00332   00330
 000353
AEN 0 005
CRN 0 006
DVR 0 008   00334
SAU 00342
AEN 0 005
   VAL
CRN 0 006   00340
   ADD
   RAM
CES 00202
MLR 0 006   00344
   SUB
CRN 0 006
TFA 00362
   DO       00350
 000331
 000336
   DIF
ARN 0 010   00354
CES 00200
MLE 0 005
   RAN
   RTP      00360
   DIF
*L* 00004
   SBQ
   STP      00364

PROGRAMME TERMINE
```

```
000 0000 'DEBUT' 'COM' INTERP 18; 'REEL' A,B,C; 'ENTIER' I,J;
001 0013 'AIGUILLAGE' AIG:=E1,E2,'SI' A 'SUP' B 'ALORS' E3'SINON' E4;
002 0029 A:=4; B:=I:=0; 'ALLER' SW;
003 0042 E1: EXL(<!PASSAGE E1>);IMPR; 'ALLERA' SW;
004 0055 E2:EXL(<!PASSAGE E2>);IMPR; 'ALLERA' SW;
005 0068 E3: EXL( <! PASSAGE _E3 >); IMPR;
006 0078 I:=I-1; 'ALLER A' SW;
007 0087 E4: EXL(<!PASSAGE E4>);
008 0095 S:SW: I:=I+1; A:=A-1; 'ALLERA' AIG.(I).;
009 0117 EXL(<! INSTRUCTION VIDE>);
010 0123 IMPR;'FIN' INTERP 18
011 0126 #
CODIFICATION TERMINEE

GENERATION TERMINEE

 000004    00200
 000000
 000001
 000325    E1
 000345    E2 00204
 000365    E3
 000413    E4
 000427    S
 000427    SW 00210      Table étiquette
 000310    AIG           Table aiguillage
 400000
 400000
 4

 400000    00260
 400000
 007773    (IMPR)
 007762    (EXL)         Table Procédures Standard
*L* 00000  00264
    EBL
 000015
*L* 00001
 SAU 00311  00270        Saut de la déclaration
 STA 00203
   RTN                   Sous programme (d'évaluation
 STA 00204                d'expression de designation)
   RTN      00274
 CRN 0 005
 CRN 0 007
    SUP
 TST 00303  00300
 STA 00205
 SAU 00304
 STA 00206
   RTN      00304
 000275
 000273                  Adresses des
 000271                  Sous-programmes

   ISI      00310        Nombre d'éléments (3)
*L* 00002

 ARN 0 005
 CES 00200
    RAN     00314
 ARN 0 007
 AEN 0 011
 CES 00201
    RAM     00320
    RAN
 STA 00210
    ALL
*L* 00003   00324
 SAU 00336
 760070
 604721
 626221     00330        Chaine
 272513
 250177
 760170
*L* 00003   00334
 CRN 0 214                Caracterisation statique de paramètre
 APN 00001                Appel Procédure
 000263
 APN 00000  00340
 000262
 STA 00210
    ALL
*L* 00004   00344
 SAU 00356
 760070
 604721
 626221     00350
 272513
 250277
 760170
*L* 00004   00354
 CRN 0 230
 APN 00001
 000263
 APN 00000  00360
 000262
 STA 00210
    ALL
```

```
*L* 00005    00364
SAU 00376
 760070
 604721
 626221      00370
 272513
 250377
 760170
*L* 00005    00374
CRN 0 246
APN 00001
 000263
APN 00000    00400
 000262
*L* 00006
AEN 0 011
CEN 0 011    00404
CES 00202
   SUB
   RAN
STA 00210    00410
   ALL
*L* 00007
SAU 00424
 760070      00414
 604721                    PASSAGE E1
 626221
 272513                    PASSAGE E2
 250477      00420
 760170                    PASSAGE E3
*L* 00007
CRN 0 268                  PASSAGE E4
APN 00001    00424        INSTRUCTION  VIDE
 000263
*L* 00010
AEN 0 011
CEN 0 011    00430        PROGRAMME TERMINE
CES 00202
   ADD
   RAN
ARN 0 005    00434
CRN 0 005
CES 00202
   SUB
   RAN       00440
STA 00211        AIG
CEN 0 011        I                          } AIG[I]
   AIG           Evaluation indicateur d'Alleullage
   ALL       00444   ALLERA
*L* 00011
SAU 00461
 760070
 603145      00450
 626351
 642363
 314645
 131365      00454
 312425

 760170
```

```
*L* 00011
CRN 0 295    00460
APN 00001
 000263
*L* 00012
APN 00000    00464
 000262
   SBL
   STP
```

```
000 0000
001 0000 'DEBUT' 'COM' INTERP 19; 'REEL' A,B,C; 'ENTIER' I,J;
002 0013 'AIGUILLAGE' AIG :=E1,E2,E3;    J:=0;
003 0026 SW:  J:=J+1; EXL(<!!!__J:=_>); EXD(5,1,J); IMPR;
004 0051
005 0051 'DEBUT' 'REEL' D; D:= 0.2;  EA:EXL(<!_______EA_PASSAGE__>);
006 0067 E4: D:=D+1;  EXL(<E4!>);IMPR;
007 0083 'DEBUT'  'AIGUILLAGE' AGI:= E5,E4,AIG.(J).;
008 0096 'ALLERA' AGI.(D).;   'ALLERA' EB;
009 0105 E5: EXL(<!  E5!>);IMPR;
010 0115  'ALLERA' EA;
011 0118 EB:   EXL(< !AGI __VIDE!>); IMPR;
012 0128   'ALLERA' SORTIE;
013 0131 'FIN''FIN';
014 0134 E1: EXL (< !E1!>); 'ALLERA' SW;
015 0145 E2: EXL (<!E2!>); 'ALLERA' SW;
016 0156 E3: EXL (<!E3!>); 'ALLER' SW;
017 0167  SORTIE : 'FIN'  INTERP 19
018 0170 #
CODIFICATION TERMINEE

GENERATION TERMINEE

000000     00200
000001
000005
000313
040351     00204
040370
100437
100456
000502     00210
000516
000532
000546
000306     00214
100426
400000
400000

400000
314631
463176
007773     00264
007756
007745
*L* 00000
*L* 00001  00270
  EBL
000015
*L* 00002

SAU 00307  00274
STA 00210
  RTN
STA 00211
  RTN      00300
STA 00212
  RTN
000301
000277     00304
000275
  ISI
AEN 0 012
CES 00200  00310
  RAN
*L* 00003
AEN 0 012
CEN 0 012  00314
CES 00201
  ADD
  RAN
SAU 00330  00320
760070
606060
131341
341513     00324
760170
*L* 00003
CRN 0 209
APN 00001  00330
000266
SAU 00336
CEF 0 012
TFA 00201  00334
TFA 00202
APN 00003
000265
APN 00000  00340
000264
```

```
*L* 00004
*L* 00005
   EBL      00344
 000007
ARN 1 005
CRS 00262
   RAN      00350
SAU 00365
 760070
 601313
 131313     00354
 131313
 252113
 472162
 622127     00360
 251313
 760170
*L* 00005
CRN 0 234   00364
APN 00001
 000266
*L* 00006
ARN 1 005   00370
CRN 1 005
CES 00201
   ADD
   RAN      0037
SAU 00403
 760070
 250460
 760170     00400
*L* 00006
CRN 0 254
APN 00001
 000266     00404
APN 00000
 000264
*L* 00007
   EBL      00410
 0000005
SAU 00427
STA 00206
   RTN      00414
STA 00205
   RTN
STA 00214
CEN 0 012   00420
   AIG
   RTN
 000417
 000415     00424
 000413
   ISI
*L* 00010
STA 00215   00430
CRN 1 005
   AIG
   ALL
STA 00207   00434
   ALL
*L* 00011
SAU 00446
 760070     00440
 602505
 607777
 760170
*L* 00011   00444
CRN 0 288
APN 00001
 000266
APN 00000   00450
 000264
*L* 00012
STA 00204
   ALL      00454
*L* 00013
SAU 00467
 760070
 602127     00460
 311313
 653124
 256077
 760170     00464
*L* 00013
CRN 0 303
APN 00001
 000266     00470
APN 00000
 000264
*L* 00014
STA 00213   00474
   ALL
*L* 00015
   SBL
   SBL      00500
*L* 00016
SAU 00511
 760070
 602501     00504
 607777
 760170
*L* 00016
CRN 0 323   00510
APN 00001
 000266
STA 00203
   ALL      00514
*L* 00017
SAU 00525
 760070
 602502     00520
 607777
 760170
*L* 00017
CRN 0 335   00524
```

```
APN 00001
 000266
STA 00203
   ALL       00530
*L* 00020
SAU 00541
 760070
 602503      00534
 607777
 760170
*L* 00020
CRN 0 347    00540
APN 00001
 000266
STA 00203
   ALL       00544
*L* 00021
   SBL
   STP
```

J:= 1.0

EA PASSAGE E4

E5

EA PASSAGE E4

E4

E1

J:= 2.0

EA PASSAGE E4

E5

EA PASSAGE E4

E4

E2

J:= 3.0

PASSAGE E4

E5

PASSAGE E4

E4

E3

J:= 0

EA PASSAGE E4

E5

EA PASSAGE E4

E4

AGI VIDE

PROGRAMME TERMINE

```
000 0000 'DEBUT' 'COMMENTAIRE' DECLARATION DE PROCEDURE, INSTRUCTION PROCEDURE;
001 0000 'REEL' A,B;
002 0006 'DEBUT' 'PROCEDURE' P (X, Y, Z, E, T); 'REEL' X; 'BOOLEEN' Z; 'ETIQUETTE' E;
003 0030 'TABLEAU' T; 'ENTIER' Y;'DEBUT' X := T.(A+Y). * B;
004 0049 'SI' Z 'ALORS' 'ALLER A' E 'SINON' T.(Y). := 2*Y;
005 0064 'FIN';
006 0066 'TABLEAU' TR.(1:10).; 'BOOLEEN' BB;
007 0078 'DEBUT' 'ENTIER' I,J; 'ENTIER' 'TABLEAU' TE.(1:10).;
008 0094 'POUR' I := 1 'PAS' 1 'JUSQUA' 10 'FAIRE' 'DEBUT' TE.(I). := 2*I;
009 0113 TR .(I).:= 1/ I+I 'FIN';
010 0125 'DEBUT' 'REEL' C, D; LIRE (C,D,I,J);
011 0142 EA : LIRE (A,B); BB := A 'SUP' B;
012 0157 P (A, I,BB,EA,TR);
013 0170 P (TR.(A)., TE.(I)., BB, 'SI' A -C 'SUP' D*I 'ALORS' EX 'SINON' EB, TR);
014 0200 'FIN';
015 0202 EB : P(J, TR.(A).+2, 'SI' A 'SUP' I+2*J'ALORS' BB 'SINON' 'VRAI' , EX, TE);
016 0233 'FIN';
017 0235 B := B-A
018 0239 'FIN';
019 0242 EX : 'FIN'
020 0245 #
CODIFICATION TERMINEE

GENERATION TERMINEE

 000002    00200
 000001
 000012
 040273          } Table Procédure
 140430    00204 } Table Etiquette
 100511
 000554
 400000
 400000    00210

 400000
 007760          } Table Procédure standard
*L* 00000
*L* 00001  00264
   EBL
 000011
*L* 00002
   EBL     00270
 000007
SAU 00332
*L* 00003
   DIV     00274  Nombre de paramètres : 5
ADE 9 000
ADE   000
ADE   000         } Modèles des paramètres
ADE  256  00300
ADE 9 256
   RAM            Indicateur de réservation : 20
ARF 2 005         } Corps de procédure
ARF 2 017  00304

   CRN 0 005
   CEF 2 008
     ADD
     PVA     00310
   MLR 0 007
     RAN
   *L* 00004
   CEF 2 011  00314  Z
  TST 00321
  ETF 2 014     6
    ALL        ALLER A
  SAU 00330  00320
  ARF 2 017     T
  CEF 2 008     Y   } T[Y]
    IND         I
  CES 00200  00324  2
  CEF 2 008         Y
    MUL             X
    RAN             :=
  *L* 00005  00330
    SBL
  *L* 00006
  CES 00201
  CES 00202  00334
    TAB
  000002
  061005
  000001     00340
  *L* 00007
    EBL
  000011
  CES 00201  00344
  CES 00202
```

```
   TAB
 000002
 122007      00350
 000001
*L* 00010
SAU 00355
 000374      00354
AEN 2 005
CES 00201
   RAM
CES 00202    00360
   ING
TST 00373
   DO
 000354      00364
 000366
   FAI
 002005
 000001      00370
 000012
 000354
SAU 00415
   INF       00374
AEN 2 007
CEN 2 005
   IND
CES 00200    00400
MLE 2 005
   RAN
*L* 00011
ARN 1 005    00404
CEN 2 005
   IND
CES 00201
DVE 2 005    00410
ADE 2 005
   RAN
   RTP
   INF       00414
*L* 00012
   EBL
 000011
SAU 00425    00420   Saut à l'appel de procédure
CEF 2 006    J
CEF 2 005    I       } Caractérisations statiques
CEN 3 007    D       } des paramètres
CEN 3 005    C 00424 }
APN 00004            } Appel LIRE 4 paramètres
 000262              }
*L* 00013
SAU 00433    00430
CEN 0 007
CEN 0 005
APN 00002
 000262      00434
```

```
AEN 1 006
CRN 0 005
CRN 0 007
   SUP       00440
   RAN
*L* 00014
SAU 00451
CRF 1 005    00444  TR
SBR 0 132           EA
AEN 1 006           BB   } Caracterisations statiques
CEF 2 005           I
CEN 0 005    00450  A
APN 00005           } Appel P 5 paramètres
 000203             }
*L* 00015
SAU 00504    00454
ARN 1 005           } SSP de calcul adresse de TR[A]
CRN 0 005           }
   IND              }
   RTN       00460  }
AEN 2 007           } SSP de calcul adresse de TE[I]
CEN 2 005           }
   IND              }
   RTN       00464  }
CRN 0 005           } SSP de calcul valeur d'expression de désignation
SBR 3 005           } si A-C>Dx1 ALORS EX SINON EB
CRN 3 007           }
MLE 2 005    00470  }
   SUP              }
TST 00475           }
STA 00206           }
SAU 00476    00474  }
STA 00205           }
   RTN              }
CRF 1 005           TR
TST 00465    00500  SSP valeur désignation
AEN 1 006           BB                      } caracterisations statiques
CRS 00461           SSP adresse d'entier    }
EXE 0 301           SSP adresse de Real     }
APN 00005    00504  } Appel P 5 paramètres
 000203             }
*L* 00016
   SBL
*L* 00017    00510
SAU 00540
ARN 1 005           } SSP Valeur de TR[A]+2
CRN 0 005           }
   PVA       00514  }
CES 00200           }
   ADD              }
   RTN              }
CRN 0 005    00520
CEN 2 005
CES 00200
MLE 2 006
   ADD       00524
   SUP
TST 00531
CEN 1 006
SAU 00532    00530
```

```
CES 00027
   RTN
ARN 2 007
SBR 0 134   00534
APN 00520
 720512
CEF 2 006
APN 00005   00540
 000203
*L* 00020
   SBL
*L* 00021   00544
ARN 0 007
CRN 0 007
*L* 00022
SBR 0 005   00550
   RAN
   SBL
*L* 00023
   SBL      00554
   STP
```

```
000 0000 'DEBUT' 'COMMENTAIRE' INDICATEUR DE FONCTION;
001 0000 'REEL' A,B,C;
002 0008 'REEL' 'PROCEDURE' F(X,Y); 'ENTIER' X,Y;
003 0022 'DEBUT' 'REEL' D; D:= B:= 2+Y*C; F := X/D 'FIN';
004 0044 'DEBUT' 'ENTIER' L,M;
005 0050 'ENTIER' 'PROCEDURE' SANPAR; L := SANPAR := L+1;
006 0062 L := C := 0; M := F (4,2); C := SANPAR;
007 0081 'DEBUT' 'REEL' R;
008 0085 R := F (SANPAR+3,L);
009 0096 F (B,R)
010 0101 'FIN';
011 0104 SANPAR
012 0104 'FIN'
013 0106 ;A := SIN (F(B*C,B+C))
014 0121 'FIN'
015 0123 #
CODIFICATION TERMINEE

GENERATION TERMINEE
```

```
000002    00200
000001
000000
000004
000003    00204
090271
040320
400000
400000    00210

400000
400000
400000
400000    00250
400000
400000
400000
400000    00254
400000
400000
400000
400000    00260
400000
007767
*L* 00000
*L* 00001 00264
  EBL
000013
*L* 00002
SAU 00313  00270
*L* 00003
  SUB
ADE  000
ADE  000  00274
  DIF
ARN 1 011   D
```

```
ARN 0 007        B
CES 00200 00300  2
CEF 1 008        Y
MLR 0 009        xC
  ADD            +
  RAM 00304      :=M
  RAN            =
CEF 1 005        X
DVR 1 011        /D
  RAP 00310      } Rangement simple valeur indicateur de fonction type Reel - Numéro bloc
000001 :=R       }
  SBL
*L* 00004
  EBL 00314
000007
*L* 00005
SAU 00332
  ***  00320    → 0 paramètre
  DIV            → Indicateur de réservation
AEN 1 005        L
CEN 1 005        L
CES 00201 00324  1
  ADD            +
  RAQ            } Rangement multiple valeur indicateur de fonction type Entier - numéro de bloc
400002 :=PM      }
  RAN 00330      :=
  SBL
*L* 00006
AEN 1 005
ARN 0 009 00334
CES 00202
  RAM
  RAN
AEN 1 006 00340 M
SAU 00344        Saut à l'appel de Procédure
TFA 00200        Valeur Entier } Caractérisations statiques
TFA 00203        Valeur Entier }
```

```
AFN 00002   00344   } Appel fonction 2 paramètres
 000205             (Evaluation Indicateur de fonction)
  RAN                    :=
ARN 0 009                C
AFN 00000   00350   SAMPAR } Appel fonction
 000206                      0 paramètre
  RAN                    :=
*L* 00007
  EBL       00354
 000007
*L* 00010
ARN 2 005                R
SAU 00370   00360
AFN 00000
 000206
CES 00204                  } SSP de calcul valeur de
  ADD       00364          } SAMPAR+3
  RTN
CEF 1 005                L
APF 00361   SSP Expression arithmetique } Caractérisations statiques
700002      00370   } Appel fonction 2 parametres
 000205
  RAN                    :=
*L* 00011
SAU 00400   00374
*L* 00012
CEN 2 005
CEN 0 007
APN 00002   00400
 000205
  SBL
*L* 00013
*L* 00014   00404
APN 00000
 000206
  SBL
*L* 00015   00410
ARN 0 005
SAU 00431
SAU 00424
CRN 0 007   00414
MLR 0 009
  RTN
CRN 0 007
ADR 0 009   00420
  RTN
APF 00417
 720414
AFN 00002   00424
 000205
*L* 00016
```

```
  RTN
APF 00413   00430
 700001
 000262
  RAN
  SBL       00434
  STP

PROGRAMME TERMINE
```

```
000 0000 'DEBUT' 'COMMENTAIRE' PARAMETRE EFFECTIF AIGUILLAGE_APPELS PAR VALEUR;
001 0000 'REEL' A,B; 'BOOLEEN' BB; 'ENTIER' J;
002 0012 'PROCEDURE' P (AIG,E,X); 'VALEUR' X,E; 'ETIQUETTE' E;
003 0030 'AIGUILLAGE' AIG; 'REEL' X;
004 0036 'DEBUT' A:= A+J;
005 0044 'SI' X 'SUP' A 'ALORS' 'ALLER A' 'SI' BB 'ALORS'  AIG .(J). 'SINON' E
006 0058 'FIN';
007 0061 A := B := 2; BB := 'VRAI'; J := 1;
008 0075 'DEBUT' 'AIGUILLAGE' AGI := E1,E3;
009 0083 P (AGI, 'SI' BB 'ALORS' E2 'SINON' AGI.(J)., B*A+J);
010 0104 E1: E2: J:= J+1
011 0112 'FIN';
012 0115 E3 : 'FIN'
013 0118 #
CODIFICATION TERMINEE

GENERATION TERMINEE

 000002     00200
 000001
 000270
 040375
 040375     00204
 000405
 040350
 400000
 400000     00210

 400000
*L* 00000
*L* 00001
   EBL      00264
 000013
*L* 00002
SAU 00322
*L* 00003   00270
*L* 00004
   ISI
ADE  256
ADE  257   00274
ADE 9 001
   NON
ARN 0 005
CRN 0 005   00300
ADE 0 010
   RAN
*L* 00005
CRF 1 011   00304
CRN 0 005
   SUP
TST 00321
CEN 0 009   00310
 TST 00316
ETF 1 005          } AIG[J]
CEN 0 010
   AIG      00314
SAU 00320
*L* 00006
ETF 1 008
   ALL      00320
   SBL
*L* 00007
ARN 0 005
ARN 0 007   00324
CES 00200
   RAM
   RAN
AEN 0 009   00330
CES 00027
   RAN
AEN 0 010
CES 00201   00334
   RAN
*L* 00010
   EBL
 000005     00340
SAU 00351
STA 00203
   RTN
STA 00205   00344
   RTN
 000344
 000342
   SUB      00350
*L* 00011
SAU 00372
CEN 0 009
TST 00357   00354
STA 00204
SAU 00362
STA 00206
CEN 0 010   00360
   AIG
```

```
   RTN
CRN 0 007
MLR 0 005    00364
ADE 0 010
   RTN
APF 00363
 620353      00370
DVR 0 134
APN 00003
 000202
*L* 00012    00374
AEN 0 010
CEN 0 010
*L* 00013
CES 00201    00400
   ADD
   RAN
   SBL
*L* 00014    00404
   SBL
   STP
```

PROGRAMME TERMINE

```
'
000 0000 'DEBUT' 'COMMENTAIRE' APPEL PROCEDURE FORMEL_PARAMETRE EFFECTIF FORMEL;
001 0000 'REEL' A,B,C;
002 0008 'REEL' 'PROCEDURE' PA (X,F); 'REEL' X ; 'ENTIER' 'PROCEDURE' F;
003 0024 'DEBUT' 'PROCEDURE' PB (PHI); 'ENTIER' 'PROCEDURE' PHI;
004 0036 A := PHI (2*X);
005 0046 B := F (A); C := F(X);
006 0060 PB (F);
007 0065 PA := A+X
008 0069 'FIN';
009 0072 'DEBUT' 'REEL' 'PROCEDURE' SANPAR; SANPAR := 2*G;
010 0083 'ENTIER' 'PROCEDURE' PC (Z); 'REEL' Z; PC:= Z*G;
011 0100 'REEL' D,G; D:= G:= 2;A:=3;
012 0115 D:= PA (D,PC);
013 0124 PA (SANPAR,PC)
014 0129 'FIN'
015 0131 'FIN'
016 0132 #
CODIFICATION TERMINEE

GENERATION TERMINEE

 000002     00200
 000003
 000270
 040276
 040353     00204
 040364
 400000
 400000
 400000     00210



 400000     00260
 400000
*L* 00000
*L* 00001
   EBL      00264
 000013
*L* 00002
SAU 00347
*L* 00003   00270
   SUB
ADE 9 000
ADE  000
   SUG      00274
SAU 00315
*L* 00004
   ADD
ADE  000    00300
   INF
ARN 0 005
SAU 00311
CES 00200   00304
CRF 1 005
   MUL
   RTN
APF 00304   00310
 740001
 002005
   RAN
   SBL      00314
*L* 00005
ARN 0 007
SAU 00321
CEN 0 005   00320
AFF 00001
 001010
   RAN
ARN 0 009   00324
SAU 00327
SBE 1 005
AFF 00001
 001010     00330
   RAN
*L* 00006
SAU 00335
SBE 1 008   00334
APN 00001
 000203
*L* 00007
CRN 0 005   00340
*L* 00010
CRF 1 005
   ADD
   RAP      00344
 000001
   SBL
*L* 00011
```

```
   EBL      00350
 000011
SAU 00362
   ***
   DIV      00354
CES 00200
MLR 1 007
   RAP
 000002     00360
   SBL
*L* 00012
SAU 00374
   ADD      00364
ADE 9 000
   INF
CRF 2 005
MLR 1 007   00370
   RAP
 400002
   SBL
*L* 00013   00374
ARN 1 005
ARN 1 007
CES 00200
   RAM      00400
   RAN
ARN 0 005
CES 00201
   RAN      00404
*L* 00014
ARN 1 005
SAU 00412
MLR 0 133   00410
CEN 1 005
AFN 00002
 000202
   RAN      00414
*L* 00015
SAU 00422
*L* 00016
MLR 0 133   00420
ADR 0 132
APN 00002
 000202
   SBL      00424
*L* 00017
   SBL
   STP
```

PROGRAMME TERMINE

B I B L I O G R A P H I E

1 - NAUR P. (Ed) et al. Revised Report on the Algorithmic Language ALGOL 60 - IFIP 62.

2 - DIJKSTRA E.W. An Algol 60 Translator for the X1 - 20 p.

3 - DIJKSTRA E.W. Making a translator for Algol 60 - 11 p. [1] et [2], dans ALGOL. Bulletin supplément n° 10 : Algol 60 Translation (nov. 1961).

4 - DOLOTTA T.A. Les langages symboliques et leur édition. Chiffres, 3 - 1962 - pp. 149-174.

5 - WERNER G. Etude de la syntaxe d'Algol - Application à la compilation Thèse de Docteur-Ingénieur-Université de Grenoble - juin 1964

6 - BOUSSARD J. C. Etude et réalisation d'un compilateur Algol 60 sur une calculatrice électronique du type IBM 7090/94 et 7040/44 - Thèse de Docteur es Sciences Appliquées - Université de Grenoble -juin 1964

7 - LE PALMEC J. Compilateur Algol CAE 510-530 : Editeur-Générateur - Note technique 24 - Institut de Mathématiques Appliquées de Grenoble nov. 1963 - juil. 1964.

8 - RANDELL B. RUSSEL L.J. Algol 60 implementation - APIC Studies in data Processing n° 5 - 1964

9 - BRASSEUR M. COHEN J. Description en Algol d'un compilateur - Algol simplifié - Note technique 27 - Institut de Mathématiques Appliquées de Grenoble - avril 1964.

10 - VAN DER POEL The Construction of an Algol translator for a Small computer Symbolic languages on data Processing - pp. 229-236 Garden and Breach Edt.- 1962.

VU

Grenoble, le

Le Président de la Thèse

VU

Grenoble, le

Le Doyen de la Faculté des Sciences

VU, et permis d'imprimer,

Le Recteur de l'Académie de GRENOBLE