HAL
open science

Etude et réalisation d'un compilateur Algol60 sur calculateur éléctronique du type IBM 7090/94 et 7040/44

Jean-Claude Boussard

To cite this version:

Jean-Claude Boussard. Etude et réalisation d'un compilateur Algol60 sur calculateur éléctronique du type IBM 7090/94 et 7040/44. Génie logiciel [cs.SE]. Université Joseph-Fourier - Grenoble I, 1964. Français. NNT : . tel-00009449

HAL Id: tel-00009449

https://theses.hal.science/tel-00009449v1

Submitted on 13 Jun 2005

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° d'ordre

THÈSES

présentées à

LA FACULTÉ DES SCIENCES DE GRENOBLE

pour obtenir

LE GRADE DE DOCTEUR ES SCIENCES APPLIQUEES

par

J. C. BOUSSARD

Ingénieur I. R. G. - Licencié ès Sciences

Première thèse :

ÉTUDE ET RÉALISATION D'UN COMPILATEUR ALGOL 60 SUR CALCULATRICE ÉLECTRONIQUE DU TYPE IBM 7090/94 ET 7040/44

Deuxième thèse :

PROPOSITIONS DONNÉES PAR LA FACULTÉ

Thèses soutenues le juin 1964, devant la Commission d'examen :

Jury
- MM. J. KUNTZMANN, Président
- B. VAUQUOIS, N. GASTINEL, J. ARSAC — Examinateurs

N° d'ordre

CENTRE DE RECHERCHE
SIDERURGIE ET MINES
BIBLIOTHEQUE
B.P. N° 7 - Téléphone 49-0
SAINT-MARTIN-D'HERES

THESES

présentées à la

FACULTE DES SCIENCES DE GRENOBLE

pour obtenir

LE GRADE DE DOCTEUR ES SCIENCES APPLIQUEES

par

J. C. BOUSSARD
Ingénieur I.R.G.
Licencié ès Sciences

Première thèse :

ETUDE ET REALISATION D'UN COMPILATEUR
ALGOL 60 SUR CALCULATRICE ELECTRONIQUE
DU TYPE IBM 7090/94 ET 7040/44

Deuxième thèse :

PROPOSITIONS DONNEES PAR LA FACULTE

Thèses soutenues le Juin 1964 devant la commission d'examen :

Jury
- MM. J. KUNTZMANN, Président
- B. VAUQUOIS, N. GASTINEL, J. ARSAC — Examinateurs

FACULTE DES SCIENCES

LISTE DES PROFESSEURS

DOYENS HONORAIRES

M. FORTRAT P.
M. MORET L.

DOYEN

M. WEIL L.

PROFESSEURS TITULAIRES

MM.	NEEL L.	MAGNETISME ET PHYSIQUE DU SOLIDE
	DORIER A.	ZOOLOGIE
	HEILMANN R.	CHIMIE ORGANIQUE
	KRAVTCHENKO J.	MECANIQUE RATIONNELLE
	CHABAUTY C.	CALCUL DIFFERENTIEL ET INTEGRAL
	PARDE M.	POTAMOLOGIE
	BENOIT J.	RADIOELECTRICITE
	CHENE M.	CHIMIE PAPETIERE
	BESSON J.	ELECTROCHIMIE
	WEIL L.	THERMODYNAMIQUE
	FELICI N.	ELECTROSTATIQUE
	KUNTZMANN J.	MATHEMATIQUES APPLIQUEES
	BARBIER R.	GEOLOGIE APPLIQUEE
	SANTON L.	MECANIQUE DES FLUIDES
	OZENDA P.	BOTANIQUE
	FALLOT M.	PHYSIQUE INDUSTRIELLE
	GALVANI O.	MATHEMATIQUES
	MOUSSA A.	CHIMIE NUCLEAIRE
	TRAYNARD P.	CHIMIE
	SOUTIF M.	PHYSIQUE
	CRAYA A.	HYDRODYNAMIQUE
	REULOS R.	THEORIE DES CHAMPS
	AYANT Y.	PHYSIQUE APPROFONDIE
	GALLISSOT F.	MATHEMATIQUES APPLIQUEES
Melle	LUTZ E.	MATHEMATIQUES
MM.	BLAMBERT M.	MATHEMATIQUES
	BOUCHEZ R.	PHYSIQUE NUCLEAIRE
	ILIBOUTRY L.	GEOPHYSIQUE
	MICHEL R.	GEOLOGIE ET MINERALOGIE
	BONNIER E.	ELECTROCHIMIE
	DESSAUX G.	PHYSIQUE ANIMALE
	PILLET E.	ELECTROCHIMIE
	DEBELMAS J.	GEOLOGIE
	GERBER R.	MATHEMATIQUES
	PAUTHENET R.	ELECTROTECHNIQUE
	VAUQUOIS B.	MATHEMATIQUES APPLIQUEES
	SILBER R.	MECANIQUE DES FLUIDES
	MOUSSIEGT J.	ELECTRONIQUE
	BARBIER J.C.	PHYSIQUE
	KOSZUL J. L.	MATHEMATIQUES
	BUYLE-BODIN M.	ELECTRONIQUE

PROFESSEURS SANS CHAIRE

M.	LACASE A.	THERMODYNAMIQUE
Mme.	KOFLER L.	BOTANIQUE
MM.	DREYFUS B.	THERMODYNAMIQUE
	VAILLANT F.	ZOOLOGIE ET HYDROBIOLOGIE
	GIRAUD P.	GEOLOGIE
	GIDON P.	GEOLOGIE ET MINERALOGIE
	ARNAUD P.	CHIMIE
	PERRET R.	SERVOMECANISMES
Mme.	LUMER I.	MATHEMATIQUES
Mme.	BARBIER M. J.	ELECTROCHIMIE
Mme.	SOUTIF J.	PHYSIQUE
MM.	BRISSONNEAU P.	PHYSIQUE
	COHEN J.	ELECTROTECHNIQUE
	DEPASSEL R.	MECANIQUE
	GASTINEL N.	MATHEMATIQUES APPLIQUEES

PROFESSEURS ASSOCIES

MM.	LUMER G.	MATHEMATIQUES
	HIGUCHI	BIOSYNTHESE DE LA CELLULOSE
	WAGNER	BOTANIQUE

MAITRES DE CONFERENCES

MM.	ROBERT A.	CHIMIE PAPETIERE
	ANGLES D'AURIAC P.	MECANIQUE DES FLUIDES
	BIAREZ J. P.	MECANIQUE PHYSIQUE
	COUMES A.	ELECTRONIQUE
	DODU J.	MECANIQUE DES FLUIDES
	DUCROS P.	MINERALOGIE ET CRISTALLOGRAPHIE
	GLENAT P.	CHIMIE
	HACQUES G.	CALCUL NUMERIQUE
	LANCIA R.	PHYSIQUE AUTOMATIQUE
	PEBAY-PEROULA J.C.	PHYSIQUE
	KAHANE	PHYSIQUE GENERALE
	DOLIQUE	ELECTRONIQUE
Mme.	KAHANE J.	PHYSIQUE
MM.	DEGRANGE C.	ZOOLOGIE
	GAGNAIRE D.	CHIMIE PAPETIERE
	RASSAT A.	CHIMIE SYSTEMATIQUE
	KLEIN J.	MATHEMATIQUES
	BETHOUX P.	MATHEMATIQUES APPLIQUEES
	POLOUJADOFF M.	ELECTROTECHNIQUE
	DEPOMMIER P.	PHYSIQUE NUCLEAIRE
	DEPORTES C.	CHIMIE
	BARRA J.	MATHEMATIQUES APPLIQUEES
Mme.	BOUCHE L.	MATHEMATIQUES
MM.	PERRIAUX J.	GEOLOGIE
	SARROT-REYNAULD J.	GEOLOGIE
	CAUQUIS G.	CHIMIE GENERALE
	LABBE A.	BOTANIQUE
	BONNET G.	PHYSIQUE GENERALE
	BARNOUD F.	BIOSYNTHESE DE LA CELLULOSE
Mme.	BONNIER M.J.	CHIMIE

MAITRES DE CONFERENCES ASSOCIES

MM.	ISHIKAWA Y.	MAGNETISME
	QUATTROPANI	THERMODYNAMIQUE

Au nom du Laboratoire de Calcul, j'exprime ma reconnaissance à la direction de la Compagnie IBM FRANCE pour la part qu'elle a pris à ce travail.

Il m'est personnellement agréable de remercier ici Monsieur M. GUERIN, dont la compétence et la disponibilité m'ont été particulièrement précieuses, et Monsieur G. MARS, Directeur du Centre d'Etudes et Recherches de la Gaude, ainsi que tout le personnel du Département ordinateurs de ce Centre, pour l'accueil sympathique et les conditions de travail excellentes qui m'y ont été réservés.

Je remercie les Directions des centres de Calcul de l'E.D.F. à Clamart, du CEA à Saclay et Limeil, pour la contribution qu'ils ont apportée,
et enfin tous les membres du Laboratoire qui ont pu m'apporter leur aide, tout spécialement Messieurs A. AUROUX et J. CHEMIN, ainsi que Mademoiselle M. GUELTON et Monsieur P. MOUNET, à qui je dois la réalisation matérielle de cet ouvrage.

A ma Mère,

A la mémoire de mon Père

- I N T R O D U C T I O N -

Le langage algorithmique international ALGOL* définitivement codifié en 1960 [1 et 2], a été l'objet, dès l'année 1961, de nombreuses tentatives d'analyse syntaxique ayant pour but sa traduction (ou compilation) dans des langages directement interprétables par les calculateurs arithmétiques modernes. L'Université de Grenoble, en particulier, réunit au mois d'Octobre 1961, sous la présidence de M. KUNTZMANN, Directeur de l'Institut de Mathématiques Appliquées de Grenoble, un groupe de chercheurs destinés à réaliser la compilation d'ALGOL 60 sur les machines les plus courantes en France.

Le groupe ALGOL s'est réunit régulièrement jusqu'au mois de Janvier 1962, date à laquelle il se scinda en trois sous-groupes orientés chacun sur une machine déterminée.

A la lumière d'analyses syntaxiques récentes [3] et de l'expérience acquise sur d'autres compilateurs [4 et 33], la méthode commune choisie par ces trois groupes fut celle des "piles de mémoires" [5], dont la technique a été maintes fois étudiée depuis [22 et 33]

Le travail que nous présentons ici a été élaboré à l'intérieur de celui de ces sous-groupes orienté sur les machines I. B. M. 7090/94 puis 7040/44 .

* Dans cet ouvrage, nous supposerons connues les caractéristiques générales du langage ALGOL ainsi que celles des machines I.B.M. 7090 et 7044.

Le résultat que nous avons obtenu est un programme compilateur entièrement intégré dans le système de programmation I.B.M. et exploité régulièrement depuis Janvier 1964 dans plusieurs centres de calcul, en particulier à l'Institut de Mathématiques Appliquées de Grenoble, avant d'être officiellement distribué à l'organisation SHARE des utilisateurs I.B.M.

Nous nous sommes particulièrement efforcés de respecter le plus possible les spécifications du langage ALGOL 60, éventuellement mis à jour [6] et, actuellement, les limitations et restrictions, apportées aux programmes origines (voir Appendice II) acceptés par le compilateur, sont sensiblement communes aux autres compilateurs déjà distribués [8, 9, 13,14].

Ceci n'exclut pas dans l'avenir d'apporter au programme des améliorations intéressantes, prises parmi les plus couramment souhaitées par les utilisateurs [7 et 10].

Le problème général de la compilation d'un langage symbolique L_S est le suivant :

Etant donné un programme origine P_S quelconque écrit dans ce langage à l'aide de règles syntaxiques et sémantiques que nous connaissons, ce programme est une phrase dont il faut reconnaître chaque caractère, analyser chaque construction (à l'aide des mêmes règles qui ont servi à l'écrire), et enfin traduire le sens en la réécrivant sous forme d'un programme objet P_M dans un langage L_M dont les règles sont différentes (dans le cas présent, beaucoup plus proche du langage de la machine M).

Cette traduction doit être telle que l'exécution par la machine M du programme P_M soit possible, exactement équivalente à

celle du programme P_S par une machine idéale M' dont le langage $L_{M'}$ serait identique à L_S , et enfin aussi efficace que possible.

De plus, si le programme compilateur P_C est lui-même écrit dans le langage L_M , on pourra dire que le système de programmation utilisant L_M sur machine M comprend également le langage L_S puisque l'exécution de tout programme P_S équivaut à l'enchaînement de l'exécution de deux programmes (P_C et P_M) écrits en langage L_M .

L'expérience sur les machines I.B.M. prouve [12, 15] qu'il est intéressant, pour des raisons de rapidité, d'encombrement du programme compilateur P_C et de facilité de traduction, de séparer dans ce problème, d'une part la recherche des caractères du programme P_S et la partie d'analyse syntaxique qui lui est liée, d'autre part l'analyse syntactico-sémantique et la génération du programme P_M que l'on en déduit.

La méthode de compilation adoptée (par "piles de mémoires") n'étant pas incompatible avec cette division en deux parties, nous avons donc adopté une méthode "à deux passages" du programme P_S.

Nous appellerons traitement morphologique et syntaxique immédiat ou passage "EDITEUR" le premier de ces "passages". Il fait l'objet de la première partie de cette thèse.

Le "deuxième passage" de chaque programme P_S ou passage "COMPILATEUR" réalise les traitements syntaxique et sémantique et la génération du programme P_M . Sa description est faite dans la deuxième partie de l'ouvrage.

Enfin, une troisième partie montre comment, une fois intégré dans un système de programmation, le compilateur P_C peut enchaîner ces deux passages d'une part, et commander une exécution aussi ef-

ficace et souple que possible du programme P_M d'autre part (planche 1).

Le compilateur a été écrit dans le langage L_M lui-même, FAP pour IBM 7090 [17 - 18] et MAP pour 7044 [26, 27, 28]*.

La première partie PC_1 du compilateur (traitement morphologique) a été écrite de Janvier à Août 1962 pour 7090 [11, 12, 15], réécrite en Novembre 1963 pour 7044, et remaniée en Février 1964 .

La deuxième partie (traitement syntaxique) a été écrite à partir de Mars 1962 et terminée en Mai 1962. Une première version de la partie générative PC_2 a été terminée en Août 1962 [23].

Le premier programme P_S transformé en langage L_M puis en langage machine (c'est à dire assemblé) l'a été en Décembre 1962, le premier P_S exécuté, l'a été en Février 1963 [Colloque ALGOL, Grenoble, le 25 Février 1963].

La mise sous le système 7090 a été faite en Mars et Mai 1963, et la version définitive de la partie générative a été terminée en Septembre 1963 .

Enfin, la version actuelle du système de programmation IBM comprenant ALGOL a été livrée sur 7090 en Octobre 1963 et sur 7044 en Décembre 1963.

Nous présenterons en conclusion quelques modifications ou améliorations dont ces quelques mois d'exploitation ont montré l'opportunité.

* Nous supposerons connues dans la suite les caractéristiques de ces langages.

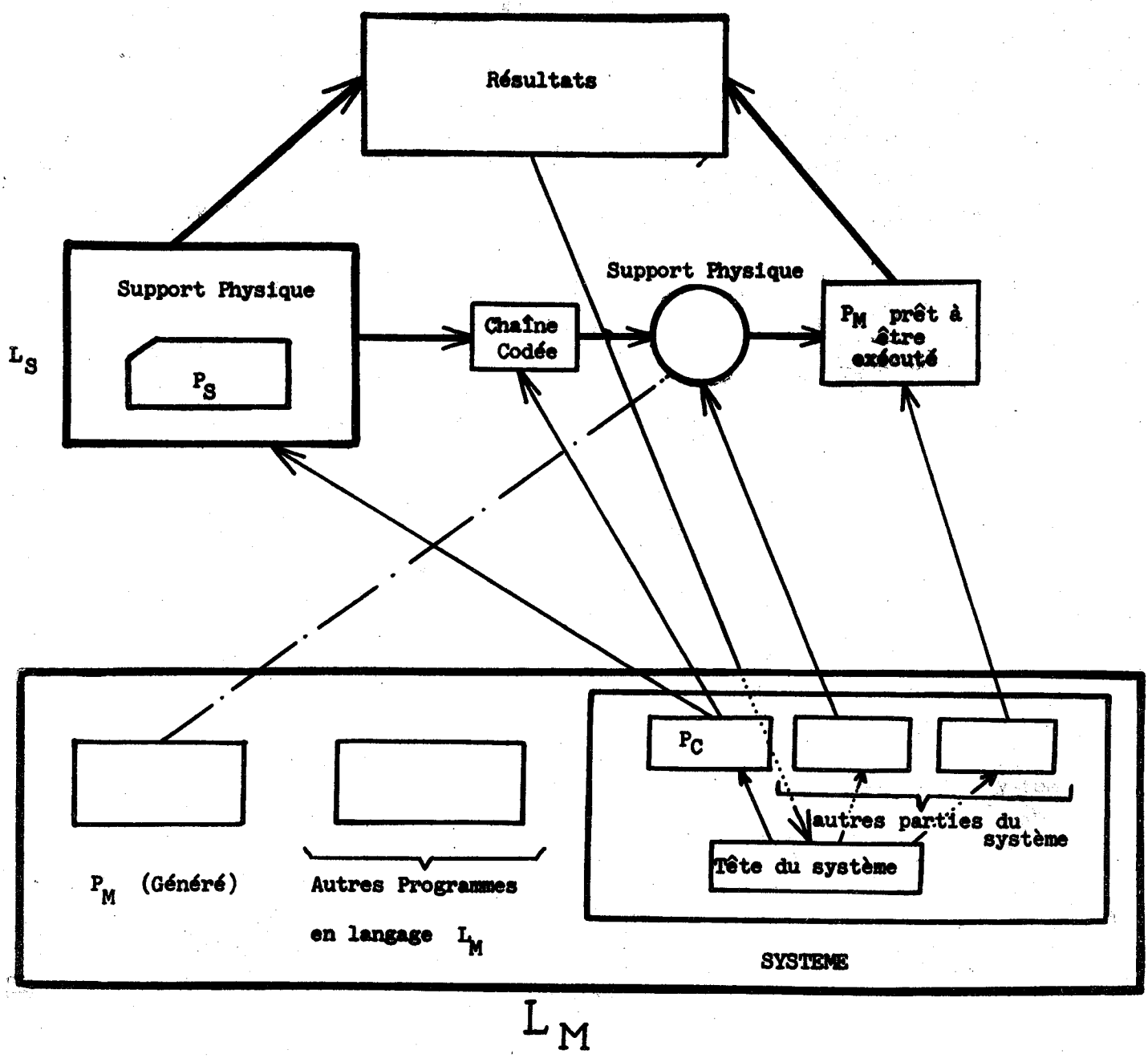


Planche 1 - COMPILATION ET EXECUTION D'UN PROGRAMME P_S

A L'AIDE D'UN SYSTEME DE PROGRAMMATION

Toutes les phrases du langage L_M sont supposées contenues dans la machine ; en réalité elles sont sur des supports physiques appropriés.

Enfin, on trouvera en annexe des tables et des renseignements pratiques concernant l'utilisation du compilateur (Appendice I, II et III), ainsi que des exemples de programmes compilés (Appendice IV).

- PREMIERE PARTIE -

P A S S A G E E D I T E U R

CHAPITRE I

DEFINITIONS ET GENERALITES

I - LES TACHES DU PROGRAMME EDITEUR*

Etant donné un programme origine P_S écrit en langage ALGOL L_S, le programme PC_1 a pour but d'une part de réaliser le traitement morphologique et syntaxique immédiat des constituants de P_S et d'autre part de faciliter au maximum la tâche du programme PC_2 (passage compilateur).

Pour cela, il a été assigné au programme PC_1 les tâches suivantes :

1 - Délimitation des unités syntaxiques du programme P_S, une unité syntaxique représentant la plus petite suite de caractères ayant une valeur sémantique en ALGOL.

Plus précisément, nous définissons dans ce qui suit :
<unité syntaxique> := <identificateur> | <valeur logique> | <entier sans signe> | <délimiteur> | <6 caractères d'une chaîne ou d'un corps de procédure en code machine> .

2 - Remplacement de l'unité syntaxique délimitée par un mot machine et un seul, convenablement codé, transmis au programme PC_2.

* Ce programme a été écrit et mis au point dans sa plus grande partie par T. A. DOLOTTA [11, 12, 15] .

3 - Résolution des questions de portée et de déclaration des identificateurs, ainsi que du problème des identificateurs utilisés avant d'être déclarés.

4 - Elimination des commentaires et des effets des dispositions typographiques du programme P_S .

5 - Détection des erreurs les plus fréquentes d'écriture.

Une fois ces tâches accomplies, l'éditeur PC_1 est en mesure de communiquer au compilateur PC_2 une "chaîne codée" formée des unités syntaxiques transformées, à laquelle le compilateur n'a qu'à appliquer les règles de calcul du langage, sans tenir compte des règles d'écriture * .

II - GENERALITES SUR LA CHAINE CODEE

Elle est formée de la suite des mots mémoires traduisant les unités syntaxiques dans un code interne au compilateur, qui comprend, pour les délimiteurs, le type et l'identité du symbole, et pour les identificateurs, son type, déterminé par la déclaration correspondante, et un numéro qui distingue cet identificateur de tout autre identificateur du même type.

Notons que chaque occurrence de chaque identificateur est remplacée dans la chaîne par le profil qualitatif ainsi défini et que, par conséquent, le compilateur aura toujours tous

* On verra que cette séparation entre les tâches de PC_1 et PC_2 ne peut en réalité être qu'assez grossière, certains problèmes d'édition restant à la charge du compilateur.

les renseignements concernant le même identificateur dans le seul mot mémoire qu'il sera en train d'analyser.

Tout se passera, pour PC_2, comme si tous les délimiteurs et tous les identificateurs de P_S étaient de même longueur, écrits en une suite régulière où n'interviendrait aucune disposition typographique particulière et comme si chaque identificateur portait en lui sa déclaration.

On verra qu'en plus, on peut considérer que la chaîne codée définit précisément l'adresse de chaque identificateur dans le programme généré L_M et que cette adresse est la même pour les identificateurs dont les portées sont disjointes, ce qui permet de réaliser une première optimisation du nombre de mémoires du programme généré.

Enfin, le passage éditeur supprimant au fur et à mesure les parties de P_S qui sont des commentaires, la chaîne codée constitue un programme en général plus court que P_S ; on verra par exemple, comment les deux programmes origines suivants :

1) <u>DEBUT</u> <u>REEL</u> <u>PROCEDURE</u> A (X, Y, Z) ;
<u>REEL</u> X, Y, Z ; A := X + Y x SIN (Z) ; <u>FIN</u> ;

2) <u>DEBUT</u> <u>COMMENTAIRE</u> : Exemple de déclaration de procédure ;
<u>REEL</u> <u>PROCEDURE</u>
ZETA (ALPHA) facteur : (BETA) angle : (GAMMA) ;
<u>REEL</u> GAMMA , ALPHA, BETA ;
ZETA := ALPHA + BETA x SIN (GAMMA) ;
<u>FIN</u> de la procédure ZETA ;

seront absolument identiques une fois transformés en chaîne codée .

III - INTERET DU PASSAGE EDITEUR

Outre les avantages constitués par l'obtention d'une chaîne codée plus courte que P_S , d'où les problèmes morphologiques et une partie des problèmes syntaxiques sont absents, l'existence du passage éditeur nous a permis :

1°) De modifier plusieurs fois, et toujours très rapidement, des parties plus ou moins importantes, parfois des algorithmes entiers* du passage compilateur PC_2 , sans jamais revenir sur la gestion matérielle du programme origine P_S ni sur des parties essentielles comme la portée des identificateurs.
Actuellement, ces mêmes parties sont encore appelées à subir des modifications ou des remaniements complets, et l'existence du programme éditeur nous permet d'affirmer que ces modifications seront toujours strictement et très précisément localisées dans le programme PC_2 .

2°) D'introduire très facilement dans le compilateur toutes les versions désirées - anglaise, française, synonymes et abréviations anglais et français - de tous les délimiteurs du langage, ce qui rend le compilateur compatible avec la plupart de ceux existant [8, 9, 13, 16 ...] .

* En particulier les algorithmes de gestion des mémoires de manoeuvre et des registres AC/MQ de la machine, ceux de compilation des listes de paramètres effectifs et des déclarations et exploitations de variables indicées.

3°) Lors de la mise sous système, de créer deux enregistrements différents sur la bande système et donc d'effectuer deux chargements consécutifs distincts de la mémoire rapide par PC_1 et PC_2, ce qui permet de conserver le programme édité (chaîne codée) en mémoire pendant le passage compilateur, réalisant ainsi un gain de temps de compilation très sensible (planche 2).

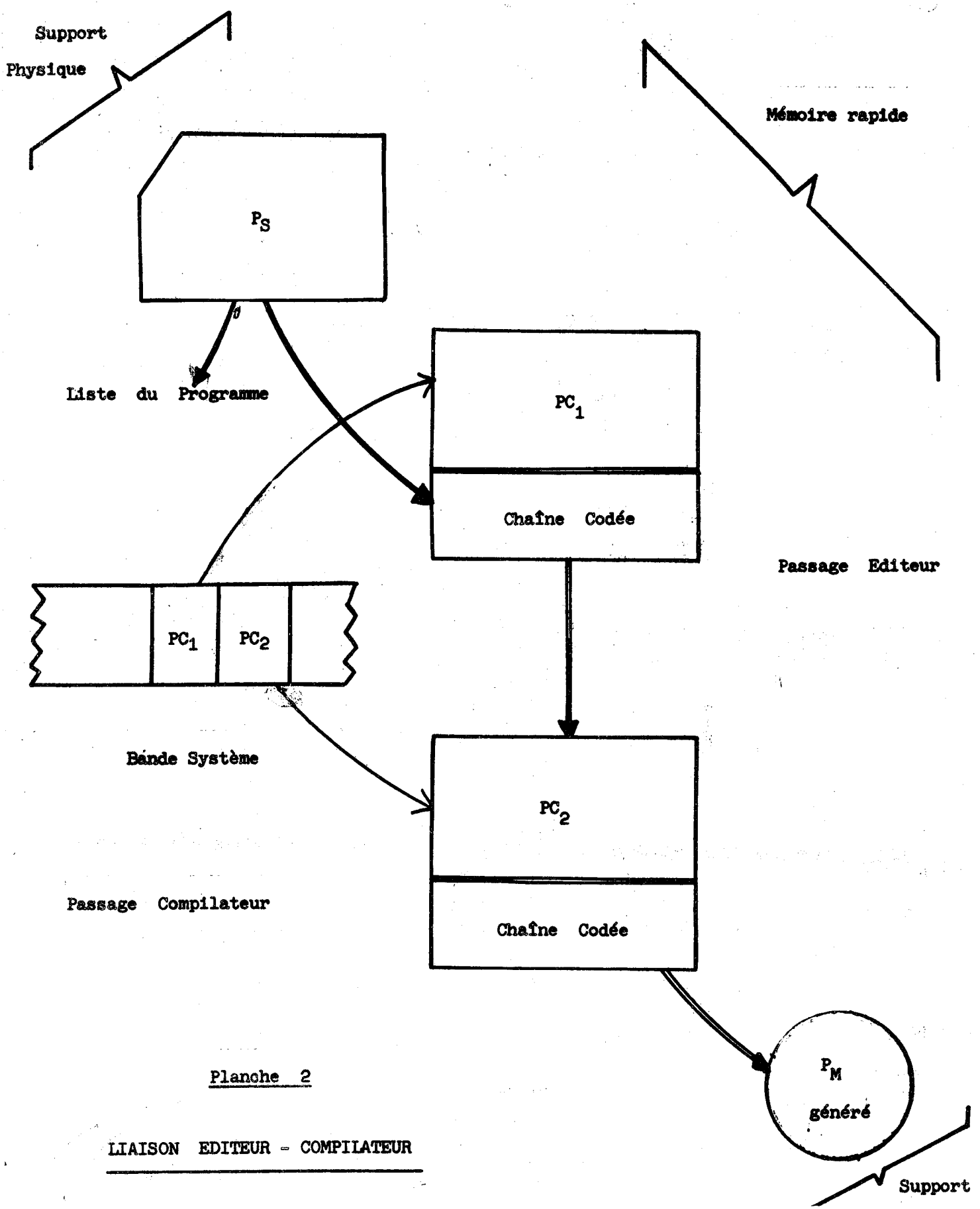


Planche 2

LIAISON EDITEUR - COMPILATEUR

CHAPITRE II

REALISATION DU PASSAGE EDITEUR

A - GENERALITES

Le programme éditeur PC_1 devant d'une part gérer physiquement la chaîne de caractères alphabétiques que constitue un programme P_S , d'autre part reconnaître et traiter syntaxiquement certaines suites de ces caractères en tant qu'identificateurs ou délimiteurs, son écriture a été divisée en deux ensembles de programmes principaux ; nous appellerons le premier :

Ensemble des séquences de gestion (E_G)

le second :

Ensemble des programmes de traitement (E_T)

Ces deux parties ont des fonctions très distinctes mais fonctionnent simultanément.

D'autre part des sous-programmes et des tables complètent ces deux ensembles, le premier pour les fonctions d'entrées - sorties et de transcodage, le second pour la gestion des piles de mémoires et de la chaîne codée.

Enfin, une dernière séquence d'"arrêt de traitement" est activée par l'ensemble de gestion quand il reconnait la fin du programme origine P_S .

Nous avons mis en évidence sur la planche 3 les trois points de liaison principaux entre les ensembles E_G et E_T.

Ce sont la délivrance d'un caractère par E_G , le signal de fin de traitement normal d'un symbole par E_T et le signal de fin de traitement avec erreur par E_T , formé par l'ensemble des retours d'erreurs portant sur des fautes morphologiques ou syntaxiques immédiates dans P_S .

Notons que dans la version actuelle de PC_1 , ces retours d'erreurs n'équivalent pas tous à un signal d'arrêt de traitement, et que la liaison correspondante peut se faire avec la séquence de fin de traitement normal.

Ce nombre très réduit de points de liaison entre E_G et E_T permettra en particulier d'introduire dans le langage L_S traité par P_C un nombre quelconque de délimiteurs nouveaux, en particulier des déclarateurs ou spécificateurs [7 - 10] .

Nous étudierons successivement l'ensemble de gestion, l'ensemble de traitement, les sous-programmes de traitement et les tables de traitement et de gestion. Nous évoquerons seulement les sous-programmes de gestion (entrées - sorties), ceux-ci étant variables d'un modèle de machine à l'autre et suivant le système utilisé.

Nous consacrerons un paragraphe à la séquence d'arrêt de traitement sur laquelle nous reviendrons dans la troisième partie.

Enfin, un chapitre entier sera consacré à la détection d'erreurs et à leur "pseudo-correction" telle qu'elle est pratiquée dans la version actuelle.

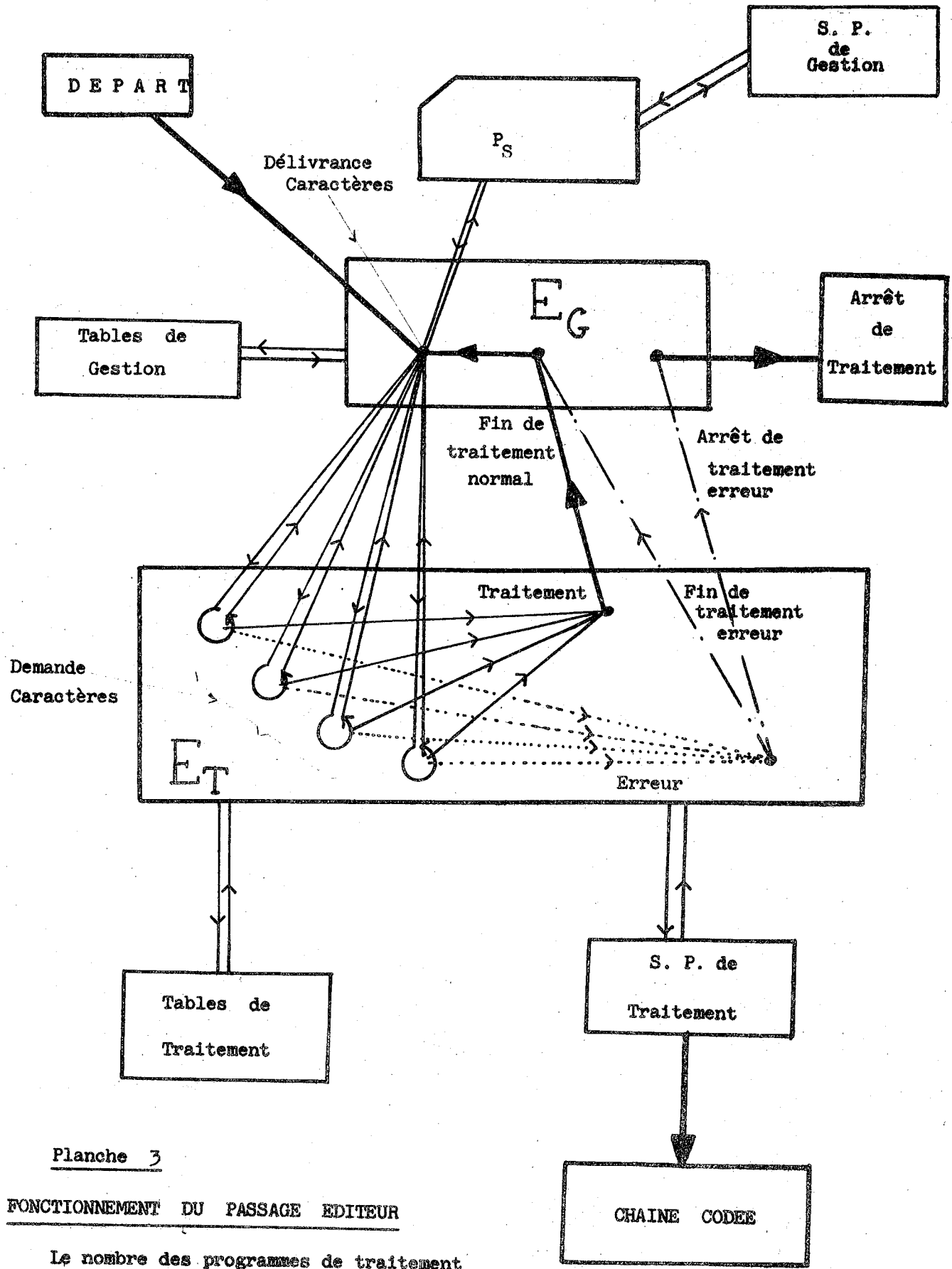


Planche 3

FONCTIONNEMENT DU PASSAGE EDITEUR

Le nombre des programmes de traitement a été réduit à 4

B - L'ENSEMBLE DE GESTION

B1 - SES FONCTIONS

L'ensemble des séquences de gestion E_G a trois rôles principaux :

Il doit d'abord communiquer caractère par caractère le texte P_S à l'ensemble de traitement E_T à la demande de celui-ci .

Il dispose pour cela de programmes d'entrée - sortie et de tables de transcodage.

Il doit ensuite reconnaître le signal d'arrêt de traitement donné par le programme P_S et aiguille alors le programme sur la séquence correspondante.

Il doit enfin assurer, par l'intermédiaire des programmes d'entrée - sortie évoqués ci-dessus, l'écriture sur support physique intermédiaire (bande magnétique) ou externe (imprimante), du programme P_S (toujours listé quelques soient les résultats de la compilation), de tout ce que l'ensemble de traitement désire voir imprimer (libellés d'erreurs en particulier) et enfin des résultats de l'édition, dont l'écriture est commandée par la séquence d'arrêt de traitement.

B2 - ORGANIGRAMME GENERAL

On trouvera sur la planche 4 l'organigramme général de l'ensemble de gestion.

L'entrée du programme est en A, où on lit une nouvelle carte du programme origine P_S dans la zone de lecture Z_1 .

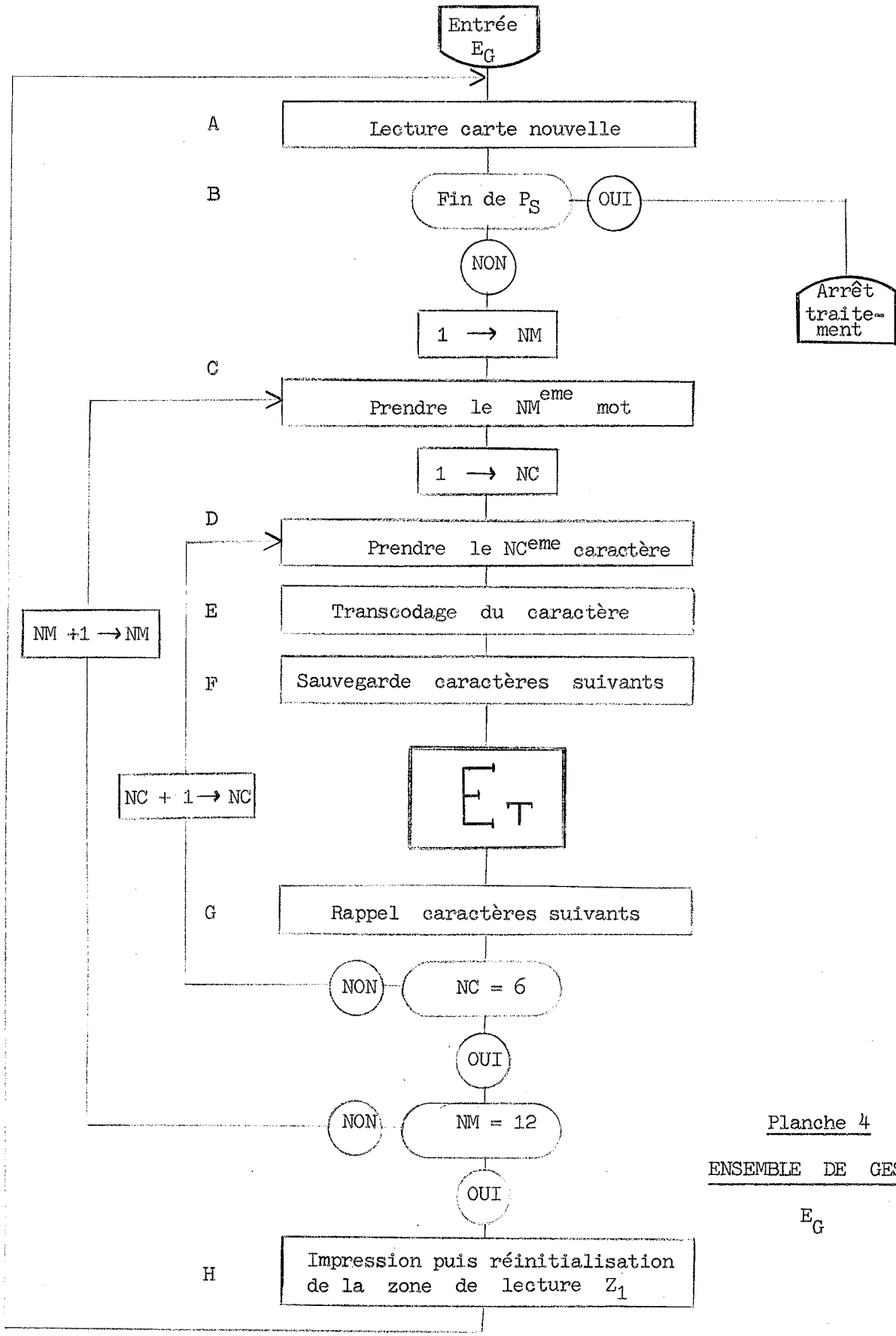


Planche 4

ENSEMBLE DE GESTION

E_G

La sortie se fait en B , lorsque cette carte marque la fin de P_S , avec aiguillage sur la séquence d'arrêt. Sinon, il y a appel de chaque caractère de chaque mot (C, D) de Z_1 , transcodage de ce caractère (Cf. B5 : délivrance de caractères) et aiguillage vers les programmes de traitement.

Ceux-ci retournent le contrôle en G pour obtenir la suite du texte P_S , qu'il y ait simple demande de caractère (premier point de liaison), ou signal de fin de traitement normal d'une unité syntaxique (deuxième point de liaison).

En G , le caractère suivant est directement obtenu si la carte lue en A n'est pas épuisée, sinon (H) il y a impression de cette carte (troisième fonction de E_G), et lecture de la suivante en A .

B3 - LECTURE DU TEXTE P_S

B3 - 1 La forme du texte P_S

Le texte origine est un programme ALGOL quelconque, respectant les limitations et restrictions rappelées en annexe (Appendice II), perforé sur cartes à l'aide de conventions appropriées rappelées également en annexe.

Ce texte peut comporter, outre des espaces qui sont sans signification, l'un quelconque des quarante sept caractères disponibles sur les perforatrices IBM, soit :

- 26 lettres majuscules de A à Z
- 10 chiffres de 0 à 9
- 11 caractères spéciaux : + - / * . , ( ) = : '

Tous ces caractères à l'exception de ' sont utilisés pour construire les symboles de base, identificateurs et nombres purs du langage P_S .

Le caractère ' est exclusivement utilisé pour encadrer les symboles de base longs du langage ALGOL , et toute autre occurrence de ce caractère dans P_S , hormis entre les symboles 'COMMENTAIRE' et ; se traduit par une erreur décelée par l'ensemble E_T .

Enfin, certains symboles simples du langage ALGOL n'ayant pas d'équivalent dans les caractères IBM, doivent être transformés soit en symbole long (ainsi ÷ s'écrira '/' et ' s'écrira '('), soit en suite de deux caractères IBM entre lesquels aucun espace n'est admis, ainsi ; s'écrira :: , [s'écrira .(, etc ...

On trouvera en appendice (II) la liste des symboles de base reconnus par l'éditeur et leur représentation sur carte.

B3 - 2 Sous-programme de lecture

Le texte P_S est lu, carte par carte, dans une zone Z_1 formée de vingt mémoires, les dix neuf premières contenant au départ des caractères "espace" et la dernière zéro.

La carte est lue dans les quatorze premières de ces mémoires, et seules les douze premières contiennent P_S . Les deux suivantes peuvent contenir une identification de la carte et les six dernières ne serviront qu'à l'impression.

B4 - ARRET DE TRAITEMENT

Le signal d'arrêt de traitement est donné en B par la présence, en première mémoire de la zone Z_1 des six caractères FALGOL, qui marquent donc la fin de P_S .

Seule une carte de P_S présentant ces six premiers caractères, sans espace entre eux, peut entraîner dès sa lecture l'arrêt du traitement.

Pratiquement, on verra (troisième partie) que ce test est doublé par une sécurité qui consiste à regarder en B si la carte qui vient d'être lue n'est pas une carte "système" d'un certain type, auquel cas l'édition est arrêtée avec diagnostic d'erreur.

B5 - TRANSCODAGE ET DELIVRANCE DES CARACTERES

La présence dans P_S de caractères doubles pouvant représenter des caractères simples d'ALGOL (B3 - 1), oblige E_G à traiter ces couples de caractères en bloc, et à leur affecter un code de six positions binaires analogue au code I.B.M. affecté aux caractères simples.

En fait, pour que tous les caractères de P_S, simples ou doubles, aient un code facile à traiter par E_T, c'est un transcodage complet qui est effectué en E, à l'aide d'une table de gestion (Appendice I.1).

Nous appellerons "code D" le résultat de cette opération, qui affecte aux caractères de P_S, simples ou doubles, les valeurs suivantes :

Espace	0
Lettres	1 à 32_8
Chiffres	40_8 à 51_8
Caractères spéciaux	52_8 à 77_8

La délivrance de caractères se fera donc aux programmes de traitement E_T après transcodage.

Cependant, sur la demande de ceux-ci, E_G peut supprimer cette opération. C'est le cas pour le traitement des chaînes au sens ALGOL et des corps de procédures en code machine.

Notons que le choix, pour former les symboles doubles, de caractères ayant déjà une signification comme symboles simples, n'affecte en rien la généralité du programme P_S, qui ne saurait comporter, par exemple, la suite de symboles simples .(, ou x x , etc ...

Des précautions peuvent être cependant nécessaires à la perforation, par exemple en écrivant :

::: , qui sans espace signifie ;: et n'a aucun sens en ALGOL, et avec un espace après le premier : signifie :; et représente une instruction vide étiquetée.

B6 - IMPRESSION ET REINITIALISATION DE Z_1

Quand une carte a été traitée en entier (H de planche 4), E_G procède à son impression sur le support physique (imprimante ou bande magnétique, cf. troisième partie) où seront listés les résultats de la compilation et de l'exécution de P_S.

Les vingt mémoires de Z_1 sont imprimées.

Les quatorze premières contiennent la carte lue en A.

On verra que les six dernières peuvent contenir des informations provenant du traitement des caractères de la carte par E_T (cf. E1 - 2 et chapitre III).

Enfin, la zone Z_1 est réinitialisée, c'est à dire remise à blanc pour les dix neuf premières mémoires et zéro pour la vingtième.

C - L'ENSEMBLE DE TRAITEMENT

C1 - FONCTIONS PRINCIPALES DE E_T

Les programmes de traitement contenus dans E_T ont pour premier rôle de délimiter les unités syntaxiques de P_S, en analysant, caractère par caractère, ce qui est fourni par E_G.

Une unité syntaxique ayant été isolée, il faut :

1°) La ranger, convenablement codée, dans la chaîne C communiquée à PC_2.

2°) Eventuellement dérouler une séquence de traitement spéciale pour cette unité syntaxique, le plus souvent modifier une pile de mémoires.

E_T a ensuite pour rôle de créer, au fur et à mesure qu'il détecte des identificateurs, une "table d'équivalences", qui sera imprimée dans la séquence d'arrêt de traitement et qui indique à l'utilisateur par quel symbole, dans le programme généré P_M, sera remplacé chaque identificateur de P_S.

Enfin, E_T doit détecter le maximum d'erreurs existant dans P_S et appliquer à chacune d'elles un traitement particulier.

Pour effectuer son travail, E_T dispose de tables et de sous-programmes de traitement dont nous donnerons le fonctionnement dans les parties E et F du présent chapitre.

Il dispose également de compteurs et de mémoires intermédiaires dont nous donnerons la définition dans le texte.

Enfin, E_T utilise deux piles de mémoires P1 et P2 qui sont décrites dans le paragraphe C3.

Les définitions, les algorithmes de traitement normal et d'erreurs ainsi que les sous-programmes de traitement, seront décrits en ALGOL sous forme de déclarations de variables, d'instructions composées et de déclarations de procédures supposées apparatenir à un programme fictif décrivant PC_1 en entier.

C2 - CONSTRUCTION DE LA CHAINE CODEE

La communication à PC_2 d'une chaîne C d'unités syntaxiques convenablement codées traduisant le programme P_S, est la fonction essentielle de E_T.

Cette chaîne, formée de mémoires nulles au départ, est placée au fur et à mesure de sa construction dans la partie haute de la mémoire rapide, du haut vers le bas. Son encombrement est un paramètre d'assemblage du compilateur et est lié à la dimension des piles de mémoires P1 et P2. Il fixe le nombre maximum d'unités syntaxiques que peut comporter le programme origine P_S. La fin de la chaîne codée est marquée par une mémoire nulle, et sa longueur est contenue à tout moment du traitement par le compteur CODEL. Cette longueur représente également le numéro de la dernière unité syntaxique rangée (NUS). Elle est imprimée avec chaque ligne de P_S (en H de la planche 4), dans la vingtième mémoire de la zone Z_1.

La chaîne codée C ne comprend aucune des parties commentaires éventuellement présentes en P_S.

Tous les caractères de P_S intervenant dans les chaînes ALGOL ou dans les procédures en code machine sont recopiés, sans transcodage dans C, les mots incomplets étant éventuellement remplis par des codes 77_8.

Toutes les autres unités syntaxiques sont transformées en un mot mémoire chacune, (trente six positions binaires de 0 à 35) ayant la configuration suivante :

C2 - 1 Symboles de base :

- Position 0 à 0 s'ils sont opérandes (valeurs logiques)
  à 1 dans tous les autres cas
- Position 1 à 1
- Position 2 à 1 s'ils marquent le début ou la fin d'une partie C recopiée de P_S (crochets de chaînes et de code).
  à 0 dans tous les autres cas
- Positions 3 -20 à 0 ou à 1 suivant l'identité du symbole (Appendice I2).
- Positions 21 -35 : un numéro de symbole de base, NSB, compté séquentiellement dans P_S à partir de zéro.

C2 - 2 Identificateurs :

- Position 0, 1, 2 à 0
- Positions 3 - 14 à 0 ou 1 suivant le type et le genre de l'identificateur, donnés par sa déclaration ou sa position dans P_S (Cf. Appendice I3). E_T vérifie pour chaque identificateur la validité de cette partie.
- Positions 18 - 35 : un numéro d'identificateur qui servira directement à PC_2 pour générer les réservations de mémoire et les champs variables des instructions relatives à chaque identificateur.

Ce numéro est pris parmi les huit numérotations suivantes, commençant toutes à un sauf NN qui commence à zéro.

NQ : quantités simples non persistantes
NT : tableaux non persistants
NQP : quantités simples persistantes
NTP : tableaux persistants
NE : étiquettes
NN : entiers sans signe
NP : procédures
NA : aiguillages

Chacune de ces numérotations est liée à l'ordre des déclarations correspondantes, la première apparition d'un entier sans signe lui servant de déclaration et la première apparition d'une étiquette suivie de : lui servant de déclaration.

Les numérotations NQ et NT étant "optimisées", deux identificateurs de portées disjointes et ayant le même rang de déclaration auront un même numéro, et correspondront donc à une seule mémoire réservée par P_M .

La numérotation NP présente la particularité d'être formée de deux parties :

Les positions 18 à 26 (NP1) correspondent à un numéro de procédure (dans l'ordre de déclaration).

Les positions 27 à 35 (NP2) forment une numérotation interne des paramètres formels à l'intérieur de NP1 , l'identificateur de la procédure lui-même étant numéroté zéro dans cette partie.

La position 18 indique alors s'il s'agit d'une procédure déclarée dans le programme P_S (NP1 $<$ 256) ou d'une procédure "standard" (NP1 $\geqslant$ 256), supposée déclarée à l'extérieur de tout programme P_S .

Notons enfin que seule la numérotation NP restreint le nombre d'identificateurs possibles dans P_S (255 procédures dé-

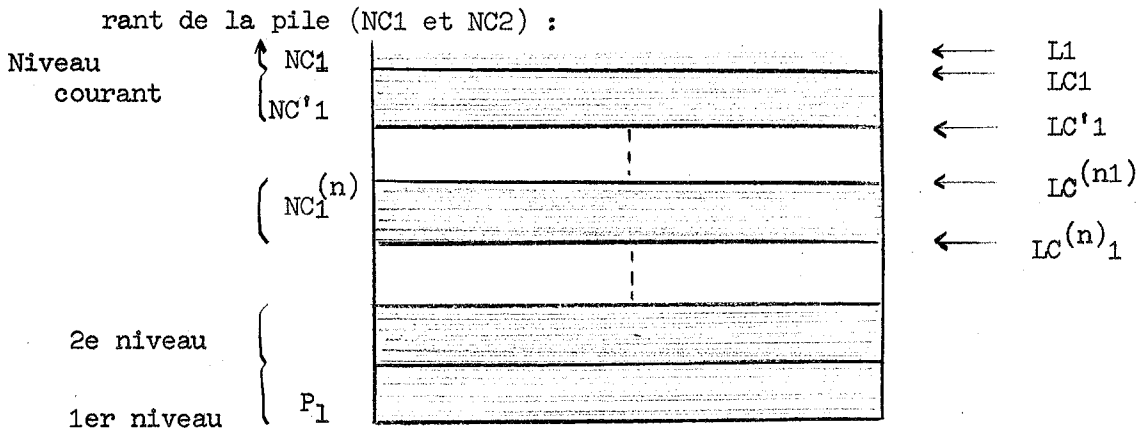
clarées, 511 paramètres par procédure), les autres numérotations étant au contraire restreintes elles-mêmes par l'encombrement de C .

C3 - LES PILES DE MEMOIRES P1 ET P2

Ces piles (ou "stacks", ou "push - down storages") sont constituées par deux ensembles de mémoires auxquels on associe les index L1 et L2 , indiquant à tout moment la dernière ligne occupée de chaque ensemble.

Chaque pile peut contenir plusieurs niveaux, marqués par exemple par une ligne nulle.

Le début du niveau supérieur de chaque pile est repéré par un index (LC1 et LC2), et ce niveau est appelé niveau courant de la pile (NC1 et NC2) :



Les opérations suivantes peuvent être effectuées sur une pile :

A) - Augmentation d'une ligne de NC . Elle a toujours lieu par le haut, et on fait alors $L + 1 \rightarrow L$

B) - Création d'un niveau. Elle se fait toujours en $L + 1$, et on fait alors $L + 1 \rightarrow L \rightarrow LC$.

C) - Effaçage d'un niveau. C'est toujours le niveau courant.
On fait alors $LC - 1 \longrightarrow L$
et $LC' \longrightarrow LC$, NC' devient NC

D) - Effaçage de n niveaux. On commence toujours par le haut de la pile et on fait

$$LC^{(n-1)} - 1 \rightarrow L$$
$$\text{et } LC^{(n)} \longrightarrow LC$$

E) - Fusion de deux niveaux. Ce sont toujours NC et NC'
on fait alors $L - 1 \rightarrow L$
$LC' \rightarrow LC$
et chaque ligne de NC est descendue de 1, le niveau unique devenant niveau courant.

Dans l'éditeur, la pile P1 est formée de lignes de trois mémoires, et on y place les identificateurs déclarés dans les différents niveaux de nomenclature.

Les deux premières mémoires de chaque ligne contiennent l'information alphabétique, code D, décrivant l'identificateur (= nom de l'identificateur dans P_S, éventuellement complété à douze caractères par des espaces à droite).

La troisième mémoire de chaque ligne contient l'unité syntaxique codée qui doit remplacer chaque occurrence de l'identificateur dans C .

Les niveaux créés dans P1 sont de deux sortes :

1°) Niveaux de DEBUT

Ils sont créés par E_T en rencontrant le symbole 'DEBUT' et effacés en rencontrant 'FIN' .

Le niveau est marqué par la première mémoire d'une ligne, nulle.

Les deux autres mémoires de la ligne contiennent les valeurs courantes des numérotations NQ et NT contenues

dans deux compteurs CQ et CT au moment où l'on trouve 'DEBUT', à condition que celui-ci soit extérieur à toute déclaration de procédure.

Si 'DEBUT' fait partie d'une procédure, les numérotations NQ et NT ne seront pas optimisées à cet endroit, et les trois mémoires de la ligne niveau sont alors nulles.

2°) Niveaux de PROCEDURE

Ils sont créés par E_T en trouvant une déclaration de procédure et effacés en trouvant le ; marquant la fin de la procédure.

Ils sont constitués par une ligne nulle dont la première mémoire est négative.

La pile P2 est également constituée par des lignes de trois mémoires, et on y place les identificateurs exploités sans avoir été encore déclarés (cas des sauts en avant et des déclarations dans un ordre quelconque), ou qui ont été déclarés dans des niveaux inférieurs au niveau courant de P1 .

Pratiquement, la plus grande partie des occurrences des identificateurs de P_S donnent lieu à création d'une ligne dans P2 . L'expérience montre que, si l'on désire un encombrement E de C (= E unités syntaxiques au maximum dans un P_S), il faut prévoir un encombrement E/4 de P1 et E/2 de P2 .

Remarquons que l'existence de P2 permet de résoudre les conflits d'identificateurs du type de celui apparaissant dans le programme suivant, où la première occurrence de A se réfère au nombre déclaré dans B1 et la deuxième à l'étiquette déclarée ultérieurement dans B2 .

```
B1 {        DEBUT ENTIER A ;
            A := 5 ;
            DEBUT ENTIER B ;
B2 {            ALLER A A ;
                  ¦
                A :
                  ¦
            FIN
            FIN
```

Le cas précédent est facile à résoudre puisque la deuxième occurrence de A est dans un ALLER A, mais le cas où cette occurrence interviendrait par exemple dans un paramètre effectif est loin d'être aussi trivial. La présence de P2 le résoud entièrement.

Chaque ligne de P2 contient dans les deux premières mémoires le nom de l'identificateur (Cf. P1) et dans la troisième l'adresse dans la chaîne codée d'un "trou" à remplir par le code de cet identificateur quand on aura trouvé sa déclaration.

Les niveaux de P2 (DEBUT et PROCEDURE) sont tous constitués par des lignes nulles.

Remarquons enfin que, si le compilateur ne comporte aucune limite sur la longueur des identificateurs, la reconnaissance de ceux-ci se fait sur les deux premières mémoires de chaque ligne de P1 ou P2 et que, par conséquent, des identificateurs ayant même douze premiers caractères seront considérés comme identiques.

C4 - ORGANIGRAMME GENERAL ET FONCTIONNEMENT

La planche 5 donne l'organigramme général de l'ensemble de traitement E_T .

Nous avons dessiné sur cet organigramme les branches principales par où chaque caractère délivré par E_G est susceptible de passer avant d'être, soit éliminé (commentaires variés, identificateurs ou nombres purs trop longs), soit recopié dans C (chaînes et code), soit le plus souvent emmagasiné dans des mémoires intermédiaires pour former avec d'autres caractères, un délimiteur ou un identificateur qui est alors traité puis rangé en chaîne codée.

Nous n'avons pas détaillé sur la planche 5 les séquences de traitement spéciales, qui seront exposées dans les paragraphes qui suivent.

Les détections d'erreur (ER1 à ER10) relatives aux branches dessinées ont été indiquées sans retours.

Ceux-ci seront étudiés dans leur ensemble dans le chapitre III de cette première partie.

Dès le début de la compilation d'un programme P_S, le contrôle est donné à E_T, dont tous les aiguillages sont initialement à zéro.

Une première séquence, dite "préliminaires" est alors déroulée, qui permet à E_T d'effectuer un certain nombre d'initialisations, de s'assurer que E_G dispose bien des supports physiques dont il a besoin (ouverture de fichiers [28]) et d'initialiser la zone de lecture Z_1.

En particulier, E_T, dans cette séquence, crée un premier niveau dans la pile des déclarations P1 (niveau identique à un niveau de 'DEBUT') et place tous les identificateurs de fonctions standard reconnus par le compilateur. Cette opération revient à simuler une déclaration extérieure à tout P_S pour chacun de ces identificateurs. Ceux-ci sont contenus dans une table TFS dont on verra plus loin la constitution.

Après cette séquence, E_T fait appel à E_G pour obtenir un premier caractère. Les aiguillages étant à zéro, ce caractère est éliminé s'il s'agit d'un espace (D), par simple retour en B pour

obtenir le caractère suivant.

Le premier caractère différent d'un espace est alors une lettre (E), un chiffre (E), le caractère ' (F) ou un symbole délimiteur court d'ALGOL.

Dans ce dernier cas (G) le symbole est simplement recherché dans la table T.A (voir plus loin) et son code est immédiatement rangé en chaîne codée (H).

Si un traitement spécial est nécessaire, il a lieu à ce moment, c'est le cas des symboles suivants :

; ([ ] :=

Enfin, la détection de ce symbole est susceptible de faire basculer un des aiguillages DLP (cas de)) ou DSL (cas de .) .

Le cas F (détection de ') marque le début de l'analyse d'un délimiteur long ou d'une valeur logique. L'aiguillage DSL est alors basculé à 1 et les caractères suivants sont emmagasinés (SA) jusqu'à trouver un nouveau caractère ' . Si au bout de douze caractères on n'a pas trouvé ' , il y a erreur (ER1), aucun symbole long n'ayant plus de douze caractères.

Quand le symbole est obtenu en entier, on le cherche dans la table TB. S'il n'y apparait pas, il y a erreur dans un symbole de base (ER2). S'il apparait, et si ce n'est pas 'COMMENTAIRE', il suit la même séquence que les symboles courts.

Il y a traitement spécial pour les symboles suivants : 'DEBUT' 'FIN' , tous les déclarateurs, tous les spécifieurs.

Enfin, il peut y avoir basculement d'un aiguillage par 'COMMENTAIRE' (DCM), 'FIN' (DCF), 'CODE' (DCO), '(' (ouverture de chaîne, DCH).

Le cas E correspond au début d'un identificateur ou d'un nombre pur.

Si le caractère est une lettre, c'est un identificateur, ses douze premiers caractères sont emmagasinés (IDB), les autres sont éliminés (IDC) (par bascule de DSL à 6 et 7). Une fois obtenu en entier, il est éventuellement traité (identificateurs en déclaration ou spécification), puis codé (à l'aide de la pile P1)et rangé en C ou en P2.

Le traitement continue en DCF, le caractère suivant l'identificateur ayant été sauvegardé.

Si le caractère est un chiffre, il s'agit en général d'un nombre pur. Le cas étiquette numérique présentant de réelles difficultés de traitement, un libellé d'rreur (ER6, Cf. Traitement nombres purs) est donné si l'on rencontre une telle occurrence.

Sinon, le nombre est emmagasiné sur neuf caractères maximum en basculant DSL à 2 et 3. S'il dépasse neuf chiffres, il est trop grand (ER5). La fin du nombre (toujours entier sans signe ici) est marquée par le premier caractère spécial détecté. Si une lettre est détectée avant, il y a erreur de syntaxe ER4).

Enfin, le nombre est traité (NPG), son code est construit et rangé en C (NPH) et le traitement continue comme pour les identificateurs (DCF).

Le cas d'une partie décimale : DSL mis à 4, puis 5 et 6 par la détection de . (L, puis PDA, PDB, PDF, NPF) présente avec le précédent deux différences : d'une part la présence d'un deuxième point dans le nombre est une erreur (ER5), d'autre part le nombre peut alors comporter un nombre quelconque de chiffres, les huit premiers étant seuls pris en considération.

<u>La branche DCMA</u> est activée par la détection en SB du symbole 'COMMENTAIRE'. On y élimine tous les symboles jusqu'au prochain ; .

<u>La branche DCFA</u> est activée par la détection de 'FIN'. Tous les caractères différents de ' ou ; sont éliminés. La détection de ; marque la fin du commentaire et le traitement continue avec le symbole suivant.

Si l'on trouve ' , le commentaire est également terminé, mais on traite le délimiteur correspondant en vérifiant que c'est FIN ou SINON , seuls symboles longs possibles après FIN . Ceci est réalisé en basculant PDF1 à 1 dans SB . Si FIN ou SINON n'est pas trouvé, il y a erreur commentaire après fin (ER3).

La branche DLPA est activée par la détection de) . Elle permet d'une part d'éliminer les commentaires dans les listes de paramètres, de la forme) chaîne de lettres : (, en remplaçant cette suite de caractères par , dans C , d'autre part de détecter un certain nombre d'erreurs syntaxiques (ER8).

Dans le cas où il n'y a pas de commentaire (DLPB), le symbole) est véritablement traité.

Enfin, les branches DCHA et DCOA vérifient la bonne construction syntaxique des chaînes ALGOL et des procédures en code machine, dont nous verrons le traitement plus loin. Rappelons seulement que ces deux branches fonctionnent en supprimant le transcodage dans l'ensemble de gestion, ce qui leur permet de recopier intégralement des caractères de P_S dans C .

C5 - PROGRAMME ALGOL DECRIVANT E_T

C5 - 1 Organisation de PC_1

Afin de décrire les algorithmes de traitement normal et d'erreur, et les sous-programmes de traitement, nous supposerons dans la fin de ce chapitre et dans le chapitre III que l'ensemble E_T forme le bloc le plus extérieur d'un programme écrit en ALGOL équivalent au programme PC_1 entier.

E_G sera considéré comme une procédure d'entrée-sortie généralisée de ce programme et l'organigramme de la planche 5 comme la suite d'instructions formant le corps du bloc E_T .

Nous supposerons que l'appel à E_G se fait dans une instruction étiquetée B (équivalent au B de la planche 5) avec une constante T égale à un si le transcodage doit être activé et zéro s'il doit être supprimé. T est égale à un au départ.

Deux aiguillages AIG1 et AIG2 sont supposés déclarés en tête de E_T avec pour listes deux listes d'étiquettes

figurant sur la planche 5.

Les valeurs courantes de ces aiguillages sont celles de deux variables I1 et I2, nulles au départ.

L'étiquette E de la planche 5 est celle de la première instruction qui suit ces aiguillages.

Enfin, nous supposerons définies en tête une mémoire CAR d'emmagasinage des caractères provenant de E_G et une valeur ENC de l'encombrement maximum de C .

La configuration de PC_1 peut alors être schématisée de la façon suivante (planche 6) :

```
DEBUT   COMMENTAIRE  Ceci est la représentation simplifiée du
        programme  ALGOL  équivalent à  PC1  ;

        ENTIER  T, I1, I2, CAR, ENC  ;
        ENTIER  ESPACE,  APOSTROPHE  ;
                COMMENTAIRE  On déclare ici des variables con-
                tenant les valeurs de chaque caractère  ;
        TABLEAU  Z1 [1 : 20] , Z2 [1 : 6]  ;
        PROCEDURE  PRELIMINAIRES ;
        DEBUT  COMMENTAIRE  Cette procédure initialise les cons-
               tantes, les piles et les zones de lecture  ;
        FIN  ;
        PROCEDURE  EG  ;
        DEBUT  COMMENTAIRE  Cette procédure délivre un caractère
               en  CAR, en code  D  si  T  =  1, en code IBM  si
               T  =  0.
               Les vingt mémoires de  Z1  servent de zone de lec-
               ture et d'impression pour les cartes de  PS . Elle
               donne le contrôle à  ARRET  TRAITEMENT  en trouvant
               Z1 [1] = 'FALGOL'  ;
               FIN  ;
        AIGUILLAGE  AIG1  :=  DCHA ,  DCOA ,  DCMA  ;
        AIGUILLAGE  AIG2  :=  SA, NPA, NPD, PDA, PDB, PDF, IDA,
                              IDC, DCFA, DLPA  ;

                    [ Autres déclarations  ; ]

        T  :=  1  ;
        I1  :=  I2  :=  0  ;
A :     PRELIMINAIRES  ;
B :     EG  ;
C :     ALLER A  AIG1 [I1]  ;
        SI  CAR  =  ESPACE  ALORS  ALLER A  B  ;
        ALLER A  AIG2  [I2]  ;
E :             [ Traitements et rangements  ;
                  Séquences dont les étiquettes apparaissent dans
                  les listes de  AIG1  et  AIG2  ; ]
        ALLER A  B
FIN  du programme  PC1  ;
```

Planche 6 - CONFIGURATION D'UN PROGRAMME ALGOL DECRIVANT PC_1

C5 - 2 Déclarations de E_T

La partie de la planche 6 que nous avons appelée "autres déclarations" se décompose de la façon suivante :

1°) Déclarations de variables et de tableaux qui sont décrites ci-dessous.

2°) Déclarations de procédures équivalentes aux sous-programmes de traitement que nous verrons plus loin.
Leurs noms sont :
RANGP1, RANGP2, RANGC, RECHERCHE A, RECHERCHE B, VIDAGE P2 et CONVERSION NC1.

3°) Déclarations de procédures équivalentes aux programmes d'erreurs (Chapitre III). Leurs noms sont :
ER1, ER2, ... ER36 .

Les variables et tableaux déclarés à cet endroit sont les suivants :

ENTIER NUS, CN, L1, L2, LC1, LC2, L ;
ENTIER TABLEAU C [0 : ENC] , TNP [0 : ENC/6] ;
COMMENTAIRE Ces deux tableaux représentent la chaîne codée et la table des nombres purs, communiquées ensuite à PC_2 . Leurs indices courants sont NUS et CN , nuls au départ ;

ENTIER TABLEAU P1 [1 : ENC/12, 1 : 3] , P2 [1 : ENC/6, 1 : 3] ;
COMMENTAIRE Ces deux tableaux représentent les piles P1 et P2, à lignes de trois mémoires. Les indices courants (numéros des dernières lignes occupées) sont L1 et L2. Ceux contenant les numéros de ligne des niveaux courants sont LC1 et LC2 . Ces tableaux, comme C et TNP sont mis à zéro au départ ainsi que leurs indices ;

ENTIER MEMSB1, MEMSB2, MEMID1, MEMID2, MEMID3, MEMNP ;

COMMENTAIRE Mémoires d'emmagasinage des symboles de base longs, des identificateurs et de leur code, et des nombres purs ;

ENTIER CODF, COPAR, COBRA, COPF, COCH, IDP ;

COMMENTAIRE Compteurs des unités syntaxiques à structure de parenthèses (Début, fins, parenthèses, crochets, chaînes, listes de paramètres formels, niveaux de procédures). Nuls au départ, ces compteurs doivent l'être à la fin du traitement.

En plus, certains d'entre eux doivent être nuls, en certains points du traitement. (par exemple, COPAR et COBRA quand on trouve FIN, point-virgule, etc ...) ;

ENTIER TABLEAU NUM [1 : 7] ; ENTIER INUM ;

COMMENTAIRE Ces mémoires contiennent les valeurs atteintes à tout moment par les numérotations respectives NQ, NT, NQP, NTP, NA, NE, NP . La numérotation NN a son compteur en CN , déjà défini. L'indice INUM indique au traitement des identificateurs en déclaration quelle numérotation utiliser. Il est positionné lors du traitement des symboles déclarateurs ;

ENTIER D, S, P, F, V, OWN, SD, SOWN ;

COMMENTAIRE Ces mémoires contiennent des identificateurs de déclaration et de spécification.

$D \neq 0$ indique que le prochain identificateur rencontré est en train d'être déclaré ou spécifié.

$OWN \neq 0$ indique qu'il est persistant.

$S \neq 0$ précise que l'identificateur est en train d'être spécifié.

P est différent de 0 après qu'on ait trouvé le symbole PROCEDURE et avant qu'on ait détecté le premier point-virgule suivant.

F est différent de 0 après qu'on ait traité la parenthèse qui ouvre une liste de paramètres formels et avant qu'on ait trouvé la première spécification, la partie VALEUR, ou le corps de procédure.

SD et SOWN sont des mémoires de sauvegarde de D et OWN ;

ENTIER PROQUA, SPROQUA ;

COMMENTAIRE Mémoires intermédiaires où sont construits et sauvegardés les profils qualitatifs des identificateurs en déclaration avant qu'ils soient rangés en P1 ;

ENTIER PPF, PAV, PNP, PET, PAIG, PCH, PPERS, PREEL, PENTIER, PBOOLEEN, PTABL, PPROC, PPVFP ;

COMMENTAIRE Ces mémoires contiennent chacune la position mise à un dans le profil qualitatif, respectivement, des paramètres formels, des paramètres formels par VALEUR, des nombres, étiquettes, aiguillages, chaînes, variables persistantes, réelles, entières, booléennes, des tableaux, des procédures, et enfin des points-virgules de fin de procédure ;

C5 - 3 Instructions de traitement

La partie de la planche 6 étiquetée E comporte les instructions décrites par l'organigramme de la planche 5 et les instructions composées ou les procédures qui décrivent les algorithmes de traitement des délimiteurs déjà cités (C4 -), des identificateurs, des nombres purs (entiers sans signe), des chaînes et des corps de procédures en code machine.

Ce sont ces instructions et procédures que nous exposons maintenant, en D .

D - ALGORITHMES DE TRAITEMENT

D1 - DELIMITEURS SPECIAUX

D1 - 1 DEBUT

```
PROCEDURE DEBUT ;
    DEBUT  CODF  :=  CODF  + 1  ;
    SI     CODF  <  0  ALORS  ER14  ;
    SI     COPAR ≠  0  ALORS  ER15  ;
    SI     COBRA ≠  0  ALORS  ER16  ;
    S := F := PROQUA := 0  ;
    SI     COPF  ≠  0  ALORS  ER7   ;
    COMMENTAIRE  On a vérifié que le compteur de paramètres for-
    mels est revenu à zéro pour s'assurer que tous les paramètres
    formels sont spécifiés. Cette vérification se fait pour toute
    unité syntaxique possédant une séquence de traitement spécia-
    le et susceptible de marquer le début d'une instruction, donc
    d'un corps de procédure  ;
    SI     IDP   =  0  ALORS  DEBUT  MEMID2  :=  NUM  [ 1]  ;
                                     MEMID3  :=  NUM  [ 2]  FIN
                       SINON  DEBUT  IDP     :=  IDP +  1  ;
                                     MEMID2  :=  MEMID3 := 0 FIN ;
    MEMID1  :=  0  ;
    RANGP1  ;
    LC1  :=  L1  ;
    MEMID2  :=  MEMID3  :=  0  ;
    RANGP2  ;

    LCA     :=  L2   ;
```

```
COMMENTAIRE On a créé en P1 un niveau de DEBUT, tout nul
si on est à l'intérieur d'une procédure, avec les valeurs de
CQ et CT sinon. Ces valeurs seront reprises par CQ et CT
en trouvant le FIN correspondant. Ceci crée l'optimisation
des numérotations NQ et NT ;

FIN de la procédure DEBUT ;

D1 - 2   FIN

PROCEDURE FIN (X) ; ENTIER X ;
    DEBUT
    COMMENTAIRE FIN (X) est appelée avec X = 0 pour le trai-
    tement de FIN et X ≠ 0 pour point-virgule de fin de procé-
    dure ;
    ENTIER I, J ;
    SI X ≠ 0 ALORS ALLER A PV ;
    CODF := CODF - 1 ;
    SI CODF < 0 ALORS ER14 ;
    SI COPAR ≠ 0 ALORS ER15 ;
    SI COBRA ≠ 0 ALORS ER16 ;
    SI P1 [LC1, 1] < 0 ALORS ER25 ;
    COMMENTAIRE On a vérifié que le niveau courant de P1 n'est
    pas un niveau de PROCEDURE ;
PV : SI IDP ≠ 0 ALORS IDP := IDP - 1
                SINON DEBUT NUM [1] := P1 [ LC1,2 ] ;
                            NUM [2] := P1 [ LC1,3 ] FIN ;
    SI L1 < LC1 ALORS ER26 ;
    SI L1 = LC1 ALORS ALLER A NIVEAU 1 VIDE ;
    SI P1 [LC1 + 1,3] > PET ET P1 [LC1 + 1,3] < PAV ALORS
        DEBUT COMMENTAIRE La première ligne du niveau courant
              correspond à une déclaration d'étiquette, le ni-
              veau est donc celui d'une instruction composée.
```

On peut le fondre avec le niveau précédent ;

```
POUR I := LC1 PAS 1 JUSQUA L1 - 1 FAIRE
    POUR J := 1 PAS 1 JUSQUA 3 FAIRE
        P1 [ I, J ] := P1 [ I + 1, J ] ;
ALLER A NIVEAU 1 VIDE FIN ;
```

```
SI L2 < LC2 ALORS ER26 ;
SI L2 > LC2 ALORS VIDAGE P2 ;
```

COMMENTAIRE On a procédé au vidage du niveau courant de P2 s'il n'était pas vide, c'est à dire à l'élimination des trous dans la chaîne codée correspondant à l'exploitation à ce niveau d'identificateurs qui avaient leurs déclarations au niveau courant de P1 ;

CONVERSION NC1 ;

COMMENTAIRE On procède là à la conversion du niveau courant de P1, chacune de ses lignes donnant naissance à une ligne de la table d'équivalences des symboles externes et internes. On peut alors effacer NC1 et prendre LC1 pour L1 ;

```
POUR I := LC1 + 1 PAS 1 JUSQUA L1 FAIRE
    POUR J := 1 PAS 1 JUSQUA 3 FAIRE
        P1 [ I, J ] := 0 ;
L1 := LC1 ;
```

```
NIVEAU 1 VIDE : P1 [ L1,1 ] := P1 [ L1,2 ] := P1 [ L1,3 ] := 0 ;
```

COMMENTAIRE Il faut trouver les nouvelles valeurs de L1 et LC1 . Celle de L1 est l'ancienne diminuée de un. Celle de LC1 est le numéro de la première ligne avant L1 dont la première mémoire est nulle. S'il n'y en a aucune, il y a erreur ;

```
L1 := L1 - 1 ;
POUR I := L1 PAS - 1 JUSQUA 1 FAIRE
    SI ABS (P1 [I, 1 ] ) = 0 ALORS ALLER A SORTIE ;
ER26 ;
```

```
SORTIE :
     LC1 := I ;
     SI L2 < LC2 ALORS ER26 ;
     COMMENTAIRE On peut alors fondre le niveau courant de P2
                 (qui peut être vide) avec le précédent et défi-
                 nir le nouveau niveau NC2 comme on a défini le
                 nouveau niveau NC1 ;
     L2 := L2 - 1 ;
     POUR J := 1 PAS 1 JUSQUA 3 FAIRE
          DEBUT P2 [LC2, J] := P2 [L2 + 1, J] ;
                P2 [L2 + 1, J] := 0 FIN ;

NIVEAU 2 VIDE :
     POUR I := LC2 - 1 PAS - 1 JUSQUA 1 FAIRE
          SI P2 [I,1] = 0 ALORS ALLER A SORTIE 2 ;
     ER26 ;
SORTIE 2 :
     LC2 := I ;

     FIN de la procédure FIN ;


     D1 - 3 Point-virgule.


PROCEDURE POINT-VIRGULE ;

     DEBUT
     SI COPAR ≠ 0 ALORS ER15 ;
     SI COBRA ≠ 0 ALORS ER16 ;
     D := OWN := V := PROQUA := 0 ;
     SI S = 0 ALORS
```

```
        DEBUT
        SI  P ≠ 0  ALORS  DEBUT  P := 0  ;
                                 ALLER A  FINPV  FIN
                  SINON  SI  P1 [ LC1,1 ] < 0  ALORS
                         DEBUT
                         COMMENTAIRE  Le niveau courant de  P1
                  est un niveau de procédure, le point-virgule
                  est affecté d'un code spécial en  C  et trai-
                  té comme  FIN  ;
                  C [ NUS ]  :=  C [ NUS ]  +  PPVFP  ;
                  FIN  (1)  FIN
        FIN
FINPV  :
     FIN  de la procédure  POINT - VIRGULE  ;
```

D1 - 4 Parenthèse ouverte

```
PROCEDURE  PARENTHESE  OUVERTE  ;
     DEBUT
     COPAR  :=  COPAR  +  1  ;
     SI  P  ≠  0  ALORS  DEBUT
                         COMMENTAIRE  Il s'agit d'une parenthèse
                  de début de liste de paramètres formels. On
                  commence à construire le profil de ces para-
                  mètres  ;
                  F  :=  S  :=  1  ;
                  PROQUA  :=  PPF  +  NUM [ 7 ]  FIN  FIN  ;
```

D1 - 5 Parenthèse fermée

```
PROCEDURE  PARENTHESE  FERMEE  ;
     DEBUT
     COMMENTAIRE    Cette procédure est appelée dans la branche
```

```
                DLPB  de la planche  5  ;
COPAR  :=  COPAR - 1  ;
SI  COPAR  < 0  ALORS  ER15  FIN  ;
```

D1 - 6 Crochet ouvert

```
PROCEDURE  CROCHET  OUVERT  ;
    DEBUT
    SI  COBRA  < 0  ALORS  ER16  ;
    SI  COBRA  = 0  ALORS  DEBUT
                    COMMENTAIRE  Il peut s'agir d'un début de dé-
                 claration de tableau. On sauvegarde les indica-
                 teurs utiles  ;
                    SD  :=  D  ;
                    SOWN  :=  OWN  ;
                    SPROQUA  :=  PROQUA  ;
                    D  :=  OWN  :=  PROQUA  :=  0  FIN  ;
    COBRA  :=  COBRA + 1
    FIN  de la procédure  CROCHET  OUVERT  ;
```

D1 - 7 Crochet fermé

```
PROCEDURE  CROCHET  FERME  ;
    DEBUT
    COBRA  :=  COBRA  - 1  ;
    SI  COBRA  <  0  ALORS  ER16  ;
    SI  COBRA  =  0  ALORS  DEBUT
                            COMMENTAIRE  Il peut s'agir d'une fin
                 de section de tableau. On restaure les indicateurs
                 de déclaration pour la section suivante éventuelle ;
                            D  :=  SD  ;
                            OWN  :=  SOWN  ;
```

```
                    PROQUA  :=  SPROQUA  ;
                    SD  :=  0  FIN
FIN  de la procédure CROCHET   FERME  ;
```

D1 - 8 Deux points égale

```
PROCEDURE  DEUX  POINTS  EGALE  ;
    DEBUT
    SI  COPAR  ≠  0  ALORS  ER15  ;
    SI  COBRA  ≠  0  ALORS  ER16  ;
    COMMENTAIRE  Le symbole  :=  marque la fin de la validité du
                 déclarateur aiguillage. On doit donc annuler
                 l'indicateur  ;
    D  :=  PROQUA  :=  0  FIN  ;
```

D1 - 9 Déclarateurs de type (REEL - ENTIER - BOOLEEN)

```
PROCEDURE  TYPE  (X)  ;  ENTIER  X  ;
    DEBUT
    COMMENTAIRE  La procédure  TYPE (X)  est appelée en remplaçant
                 X  par  PREEL, PENTIER, ou PBOOLEEN  suivant le
                 déclarateur que l'on traite. Si  S  est diffé-
                 rent de zéro, c'est une spécification. Si OWN est
                 différent de zéro, la déclaration est persistante.
                 Dans tous les cas, on peut commencer à construire
                 le profil qualitatif des identificateurs qui sui-
                 vront et donner une valeur à  INUM  pour indiquer
                 au traitement des identificateurs quelle numérota-
                 tion utiliser  ;
    SI  S = 0  ALORS  DEBUT
            SI  D + P + V + F + PROQUA ≠ 0  ALORS  ER17  ;
            SI  OWN = 0  ALORS  DEBUT
                                PROQUA  :=  X  ;
                                INUM  :=  1  FIN
```

```
                                SINON  DEBUT
                                       PROQUA  := X + PPERS ;
                                       INUM  := 3  FIN
                        FIN
                SINON   DEBUT
                        SI  D + V + P + OWN + PROQUA ≠ 0  ALORS  ER18 ;
                        F  := 0 ;
                        PROQUA  := X  FIN  ;
        D  := 1
        FIN  de la procédure  TYPE  ;
```

D1 - 10 Déclarateur PERSISTANT

```
PROCEDURE  PERSISTANT  ;
    SI  S + V + F + P ≠ 0  ALORS  ER18
                           SINON  SI  D + OWN + PROQUA ≠ 0 ALORS ER17
                                                   SINON OWN := 1  ;
```

D1 - 11 Déclarateur TABLEAU

```
PROCEDURE  TABLEAU  ;
    DEBUT
    COMMENTAIRE  Si  S  est différent de zéro, c'est une spécifi-
                 cation. Si  D  est différent de zéro, il y a eu
                 déclaration de type avant, le profil qualitatif
                 est déjà commencé. Sinon, on met  PREEL. Si OWN
                 est différent de zéro, le tableau sera persis-
                 tant et il doit y avoir eu une déclaration de
                 type avant  ;
    SI  S  =  0  ALORS
        DEBUT
        SI  OWN  =  0  ALORS
```

```
                DEBUT
                SI  D = 0  ALORS  DEBUT
                                  SI  P + F + V + PROQUA ≠ 0 ALORS ER17 ;
                                  D := 1  ;
                                  PROQUA  :=  PREEL + PTABL  FIN
                          SINON  SI  F + P + V ≠  0  ALORS  ER17
                                  SINON  PROQUA  := PROQUA + PTABL  ;
                INUM  :=  2  FIN
            SINON  SI  PROQUA  =  0  ALORS  ER17
                                  SINON  DEBUT
                                         PROQUA := PROQUA + PTABL  ;
                                         INUM  :=  4  FIN  FIN
        SINON  DEBUT
               F  :=  0  ;
               SI  OWN  + P + V ≠ 0  ALORS  ER18  ;
               PROQUA  :=  PTABL  +  (SI  D  ≠  0  ALORS  PROQUA
                                                  SINON  PREEL)  FIN
        FIN  de la procédure  TABLEAU  ;
```

D1 - 12 Déclarateur PROCEDURE

```
PROCEDURE  PROCEDURE  ;
    DEBUT
    SI  S  =  0  ALORS  DEBUT
                        COMMENTAIRE  Il s'agit d'une déclaration,
                 l'indicateur de préparation à une liste de para-
                 mètres formels doit devenir non nul  ;
                        SI  P + F + V + OWN ≠ 0 ALORS  ER17  ;
                        SI  D = 0  ALORS  DEBUT
                                  SI  PROQUA ≠ 0  ALORS  ER17  ;
                                  D  :=  1  FIN
                        SINON  SI  PROQUA  =  0  ALORS  ER17  ;
                        P  :=  1  FIN
```

```
                SINON  SI  OWN + P + V ≠ 0  ALORS  ER18
                                            SINON  DEBUT  D := 1  ;
                                                          F := 0  FIN  ;
    PROQUA  :=  PROQUA  + PPROC
    FIN  de la procédure  PROCEDURE  ;
```

D1 - 13 Déclarateur AIGUILLAGE

```
PROCEDURE  AIGUILLAGE  ;
    DEBUT
    SI  S = 0  ALORS  DEBUT
                      SI  D + F + P + V + OWN + PROQUA ≠ 0  ALORS  ER17  ;
                      INUM  :=  5  FIN
              SINON  SI  D + P + V + OWN + PROQUA ≠ 0  ALORS  ER18
                                                       SINON  F := 0  ;
    D  :=  1  ;
    PROQUA  :=  PAIG  FIN  ;
```

D1 - 14 Spécifieur VALEUR

```
PROCEDURE  VALEUR  ;
    SI  S  =  0  ALORS  ER17
        SINON  SI  F  =  0  ALORS  ER18
               SINON  SI  D + V + P + OWN + PROQUA ≠ 0  ALORS  ER18
                          SINON  DEBUT
                                 V  :=  D  :=  1  ;
                                 F  :=  0  ;
                                 PROQUA  :=  PAV  FIN  ;
```

D1 - 15 Spécifieurs ETIQUETTE et CHAINE

```
PROCEDURE SPECIF (X) ; ENTIER X ;
    DEBUT
    COMMENTAIRE  Cette procédure est appelée en remplaçant X
                 par PET pour ETIQUETTE et par PCH pour CHAINE ;
    SI S = 0 ALORS ER17 ;
    SI D + V + P + OWN + PROQUA ≠ 0 ALORS ER18 ;
    D := 1 ;
    F := 0 ;
    PROQUA := X FIN ;
```

D2 - IDENTIFICATEURS

```
IDENTIFICATEUR :
    DEBUT
    COMMENTAIRE  Ce bloc d'instructions est placé dans la branche
                 IDE de la planche 5 .
                 On y suppose l'identificateur emmagasiné sur dou-
                 ze caractères dans les mémoires MEMID1 et MEMID2,
                 le caractère suivant cet identificateur étant sau-
                 vegardé en CAR .
                 Le bloc fait appel aux procédures de traitement
                 décrites ci-après qui font elles-mêmes appel aux
                 sous-programmes de traitement et de rangement dé-
                 crits en partie E ;
    SI D ≠ 0 ALORS DEBUT
                 COMMENTAIRE Il s'agit d'une déclaration ou
                 d'un en-tête de procédure ;
                 SI S = 0 ALORS DECLARATION
                          SINON ENTETE FIN
```

```
                SINON  DEBUT
                       COMMENTAIRE  OWN  doit être nul aussi. Si
                  COBRA  est nul et que le caractère suivant est
                  :  , il s'agit d'une déclaration d'étiquette,
                  sinon c'est l'exploitation d'un identificateur,
                  en dehors ou à l'intérieur de crochets  ;
                       SI  OWN ≠ 0  ALORS ER31  ;
                       SI  COBRA = 0  ET  CAR = DEUX POINTS
                           ALORS  DECLARATION  ETIQUETTE
                           SINON  EXPLOITATION  FIN  ;
FIN  IDENTIFICATEUR  :
     FIN  identificateur  ;
```

D2 - 1 Cas d'une déclaration

```
PROCEDURE  DECLARATION  ;
     DEBUT
     COMMENTAIRE  On vérifie que l'identificateur n'est pas déjà
                  en NC1. Si P ≠ 0, on a une déclaration de pro-
                  cédure, la numérotation utilisée est NP et on
                  a un niveau de procédure en  P1  et  P2 .
                  Si P = 0, on a une déclaration de quantité et
                  la numérotation utilisée est celle indiquée par
                  INUM . Dans tous les cas, on range l'identifi-
                  cateur et son code en  P1  et  C  ;
     SI  RECHERCHE A  ALORS  ER28  ;
     SI  P = 0  ALORS  DEBUT
                       NUM [INUM]  :=  NUM [INUM  +  1] ;
                       MEMID3  :=  PROQUA  +  NUM [INUM]  ;
                       RANGP1  ;
                       RANGC  FIN
```

```
SINON  DEBUT
       NUM [ 7 ] := NUM [ 7 ] + 2 ↑ 9 ;
       SI NUM [ 7 ] > 255 x 2 ↑ 9 ALORS ER29 ;
       MEMID3 := PROQUA + NUM [ 7 ] ;
       RANGP1 ;
       RANGC ;
       MEMID1 := MEMID2 := MEMID3 := 0 ;
       RANGP2 ;
       LC2 := L2 ;
       MEMID1 := - 0 ;
       RANGP1 ;
       LC1 := L1 ;
       IDP := IDP + 1 FIN
FIN de la procédure DECLARATION ;
```

D2 - 2 Cas d'un en-tête de procédure

```
PROCEDURE EN TETE ;
    DEBUT
    COMMENTAIRE Si F est non nul, il s'agit d'une liste de pa-
                ramètres formels, on augmente le compteur de pa-
                ramètres formels de un. Sinon, il peut s'agir
                d'une partie valeur ou d'une spécification. Dans
                ce cas, la construction du profil du paramètre
                formel est finie et on diminue de un le comp-
                teur de paramètres formels qui restent à spéci-
                fier ;
    SI F ≠ 0 ALORS DEBUT
             SI RECHERCHE A ALORS ER28 ;
             COPF := COPF + 1 ;
             SI COPF > 511 ALORS ER30 ;
```

```
        MEMID3  :=  PROQUA + COPF  ;
        RANGP1  ;
        COMMENTAIRE  On a rangé en  P1  le code d'u-
ne unité syntaxique incomplète (il manque les spé-
cifications et la partie valeur) dont on avait com-
mencé la construction en traitant parenthèse ouver-
te (Cf. D1 - 4). On lui fait correspondre un trou
en  C  et une adresse de trou en  P2, ce trou devant
être bouché à la fermeture du niveau correspondant
(traitement de point-virgule de fin de procédure)  ;
        MEMID3  :=  0  ;
        RANGC   ;
        MEMID3  :=  NUS  ;
        RANGP2  FIN
SINON  SI  V ≠ 0  ALORS  DEBUT
                SI  ¬ RECHERCHE A   ALORS  ER19  ;
                P1  [ L, 3]  :=  MEMID3 + PAV   ;
                MEMID3  :=  0  ;
                RANGC   ;
                MEMID3  :=  NUS  ;
                RANGP2  ;
                COMMENTAIRE  On a vérifié que l'i-
dentificateur existait en NC1. On a ajouté à son co-
de la position  VALEUR  et on a rangé un trou en  C
et en  P2  ;
                FIN
          SINON  DEBUT
                SI  ¬ RECHERCHE A  ALORS  ER19  ;
                COMMENTAIRE  On obtient cette fois
le code complet du paramètre, que l'on range en  C
et en  P1  ;
                P1 [ L,3]  :=  MEMID3 := MEMID3 + PROQUA  ;
                RANGC  ;
                COPF  :=  COPF - 1  FIN
FIN  de la procédure  EN  TETE  ;
```

D2 - 3 Cas d'une déclaration d'étiquette

```
PROCEDURE  DECLARATION  ETIQUETTE  ;
    DEBUT
    SI  RECHERCHE A  ALORS  ER28  ;
    NUM [6 ] :=  NUM [ 6 ] + 1   ;
    MEMID3  :=  PET + NUM [ 6 ]   ;
    RANGP1  ;
    RANGC   ;
    F  :=  S  :=  0  ;
    SI  COPF ≠ 0  ALORS  ER7  ;
    COMMENTAIRE  Une déclaration d'étiquette ne pouvant pas appa-
                 raître dans un en-tête, on peut vérifier ici que
                 tous les paramètres formels sont spécifiés  ;
    FIN  de la procédure  DECLARATION  ETIQUETTE  ;
```

D2 - 4 Cas d'une exploitation

```
PROCEDURE  EXPLOITATION  ;
    DEBUT
    COMMENTAIRE  Si on est à l'intérieur de crochets de déclaration
                 de tableau, l'identificateur doit être trouvé dans
                 P1, à l'extérieur du niveau courant [6,5.2.4.2]  .
                 Sinon, on  cherche en  NC1. S'il y est, on range
                 son code en  C, sinon on crée un trou dont on lais-
                 se l'adresse en  P2  ;
    SI  COBRA  =  0  OU  SD  =  0
                ALORS  DEBUT
                       SI  RECHERCHE A  ALORS  ALLER A  EXPLOIT1
                                        SINON  DEBUT
                                               MEMID3  :=  0  ;
```

```
                                        RANGC ;
                                        MEMID3 := NUS ;
                                        RANGP2 FIN
                FIN
        SINON SI ¬ RECHERCHE B ALORS ER27
                                SINON
                                        RANGC ;
EXPLOIT1 :
```

COMMENTAIRE Dans tous les cas, on ne peut plus être dans un en-tête de procédure. On doit donc vérifier que tous les paramètres formels sont spécifiés ;

```
F := S := 0 ;
SI COPF ≠ 0 ALORS ER7
FIN de la procédure EXPLOITATION ;
```

D3 - ENTIERS SANS SIGNE

```
PROCEDURE NOMBRE PUR ;
    DEBUT ENTIER I ;
```

COMMENTAIRE Le nombre (neuf ou huit chiffres au maximum suivant qu'il s'agit d'une partie entière ou décimale) est supposé emmagasiné en MEMNP .
Il est cadré à droite s'il s'agit d'une partie entière et à gauche si c'est une partie décimale .
On le range dans la table TNP s'il est différent de tous ceux qui y sont déjà.
On vérifie que l'on n'est pas dans une déclaration ou une spécification, que tous les paramètres formels sont spécifiés, et qu'il ne s'agit pas de la déclaration d'une étiquette numérique.
On notera que contrairement aux autres, la numérotation NN (CN) commence à zéro.

Le numéro zéro correspond au nombre zéro, qui est la valeur de chaque mémoire de TNP au départ. Cette particularisation du nombre zéro sera exploitée par la compilateur PC2 ;

```
    SI  MEMNP > 2 ↑ 27  ALORS  ER5  ;
    SI  COBRA  =  0  ET  CAR  =  DEUX  POINTS  ALORS  ER6  ;
    SI  COPF  ≠  0  ALORS  ER7  ;
    SI  D ≠  0  OU  S ≠ 0  ALORS  ER36  ;
    POUR  I  :=  0  PAS  1  JUSQUA  CN  FAIRE
         SI  TNP [ I ]  =  MEMNP  ALORS  ALLER A  NPEGAL  ;
    I  :=  CN  :=  CN + 1  ;
    SI  CN > ENC  / 6  ALORS  ER21  ;
    TNP [ I ]  :=  MEMNP  ;
NPEGAL  :
    MEMID3  :=  PNP + PENTIER + I  ;
    RANGC
    FIN  de la procédure  NOMBRE PUR  ;
```

D4 - CHAINES

DCHA : DEBUT COMMENTAIRE Le bloc de traitement des chaînes est initialisé par la détection de '(' qui positionne IAIG1 à 1 et supprime le transcodage en E_G (T = 0). On définit un compteur de chaînes COCH qui augmente de un en détectant le symbole d'ouverture de chaîne '(' .

Les caractères sont emmagasinés six par six, puis rangés en C, jusqu'à ce que l'on trouve ' .

On regarde alors si les caractères suivants sont)' ou (' .

Dans le premier cas, la chaîne est terminée, on diminue de un le compteur COCH et quand il devient nul, le traitement est terminé.

Dans le deuxième cas, il s'agit de chaînes imbriquées, qui se traitent de la même façon, les symboles '(' et ')' étant simplement codés par la table T.B et rangés en C . Notons que chaque mot incomplet d'une chaîne (chaînes n'ayant pas un multiple exact de six caractères) est complété par des codes 77_8 inexistant dans la codification IBM des caractères. Les symboles '(' et ')' ont eux-mêmes un code commençant par 76_8, inexistant dans le code IBM, ce qui permet au compilateur PC_2 de les distinguer facilement des caractères de chaîne ;

FIN ;

D5 - PROCEDURES EN CODE MACHINE

DCOA : DEBUT COMMENTAIRE Le compilateur autorise l'écriture dans P_S de procédures possédant un en-tête ALGOL et un corps composé d'instructions machines du type de celles générées pour former P_M .
Ces instructions, qui doivent par ailleurs respecter un certain nombre de conventions, doivent être encadrées par deux délimiteurs introduits en ALGOL qui sont des symboles longs s'écrivant respectivement 'CODE' et 'FCODE' .
Ces délimiteurs jouent le même rôle que les crochets de chaîne '(' et ')', et le traitement des caractères compris entre eux est identique à celui des chaînes, à ceci près que 'CODE' et 'FCODE' ne peuvent s'imbriquer.
L'aiguillage AIG1 est alors positionné à 2 et T = 0.

Nous verrons que, pour plus de facilité dans PC_2, 'CODE' et 'FCODE' doivent être écrits dans P_S, le premier à l'extrême droite de la carte, le deuxième à l'extrême gauche.
En plus, 'CODE' pouvant être le début d'un corps de procédure, on vérifie ici que COPF est nul ;

<u>SI</u> COPF $\neq$ 0 <u>ALORS</u> ER7 <u>FIN</u> ;

E - SOUS - PROGRAMMES DE TRAITEMENT

Les sous-programmes de traitement auxquels nous avons déjà fait allusion sont utilisés par E_T pour gérer d'une part les piles P1 , P2 et la chaîne codée C , d'autre part la table d'équivalences et son support physique.

E1 - SOUS-PROGRAMMES DE GESTION DES PILES ET DE LA CHAINE CODEE

Ces cinq sous-programmes permettent de ranger une ligne en P1 ou en P2, de ranger une unité syntaxique complète ou un "trou" en C , de procéder à un vidage de NC_2 , enfin d'effectuer la recherche d'un identificateur dans P1 .

E1 - 1 Rangements en P1 et P2

```
PROCEDURE  RANGP1  ;
    COMMENTAIRE  Les trois mémoires intermédiaires d'identifica-
                 teurs sont rangées en P1, dans la première li-
                 gne libre, l'index L1 est mis à jour, on vérifie
                 qu'il ne dépasse pas le maximum permis. Le même
                 programme sert à créer un niveau en P1. De même
                 pour P2  (RANGP2)  ;
    DEBUT
    L1  :=  L1 + 1  ;
    SI  L1 > ENC/12  ALORS  ER22  ;
    P1  [L1, 1]  :=  MEMID1  ;
    P1  [L1, 2]  :=  MEMID2  ;
    P1  [L1, 3]  :=  MEMID3  FIN  ;
```

```
PROCEDURE  RANGP2  ;
     DEBUT
     L2  :=  L2 + 1  ;
     SI  L2 > ENC/6  ALORS  ER24  ;
     P2 [L2, 1]  :=  MEMID1  ;
     P2 [L2, 2]  :=  MEMID2  ;
     P2 [L2, 3]  :=  MEMID3  FIN  ;
```

E1 - 2 Rangement en C

```
PROCEDURE  RANGC  ;
     COMMENTAIRE  En plus du rangement en chaîne codée, on augmente
                  de un le numéro d'unité syntaxique et on le range
                  dans la vingtième mémoire de la zone de lecture
                  Z1, en vérifiant que l'encombrement maximum n'est
                  pas dépassé  ;
     DEBUT
     C [NUS]  :=  MEMID3  ;
     NUS  :=  NUS + 1  ;
     SI  NUS > ENC  ALORS  ER23  ;
     Z1 [20]  :=  NUS  FIN  ;
```

E1 - 3 Recherche d'un identificateur dans P1

```
BOOLEEN  PROCEDURE  RECHERCHE A  ;
     COMMENTAIRE  L'identificateur est recherché dans NC1. S'il est
                  trouvé, la procédure prend la valeur VRAI, le code
                  de l'unité syntaxique est placé en MEMID3 et le nu-
                  méro de ligne de cet identificateur dans P1 est
                  placé en L. S'il n'est pas trouvé, la procédure
                  prend la valeur FAUX  ;
```

```
DEBUT ENTIER I ;
RECHERCHE A := FAUX ;
POUR I := NC1 + 1 PAS 1 JUSQUA L1 FAIRE
     SI P1 [I, 1] = MEMID1 ALORS
          DEBUT
          SI P1 [I, 2] = MEMID2 ALORS
          DEBUT
          L := I ;
          MEMID3 := P1 [I, 3] ;
          RECHERCHE A := VRAI FIN
     FIN
FIN de la procédure RECHERCHE A ;
```

```
BOOLEEN PROCEDURE RECHERCHE B ;
    DEBUT
    COMMENTAIRE Cette procédure est analogue à la précédente, I
                variant de 1 à NC1 - 1, pour chercher l'identi-
                ficateur dans tous les niveaux extérieurs au ni-
                veau courant (cas d'exploitation dans une liste
                de bornes) ;
    FIN ;
```

E1 - 4 Vidage de NC2

```
PROCEDURE VIDAGE P2 ;
    COMMENTAIRE Cette procédure est appelée au traitement de cha-
                que FIN de bloc et par la séquence d'arrêt de
                traitement.
                Chaque ligne de NC2 est comparée à toutes les li-
                gnes de NC1. Les unités syntaxiques correspondant
                aux lignes identiques sont rangées en C à la pla-
```

ce des trous dont les adresses sont dans les troisièmes mémoires des lignes de P2. Celles-ci sont annulées, puis éliminées dans P2.
A la fin du sous-programme, NC2 peut être vide ou non. S'il ne l'est pas, ses lignes ne peuvent contenir que des identificateurs non encore déclarés (étiquettes avec saut en avant) ou déclarés dans un bloc extérieur ;

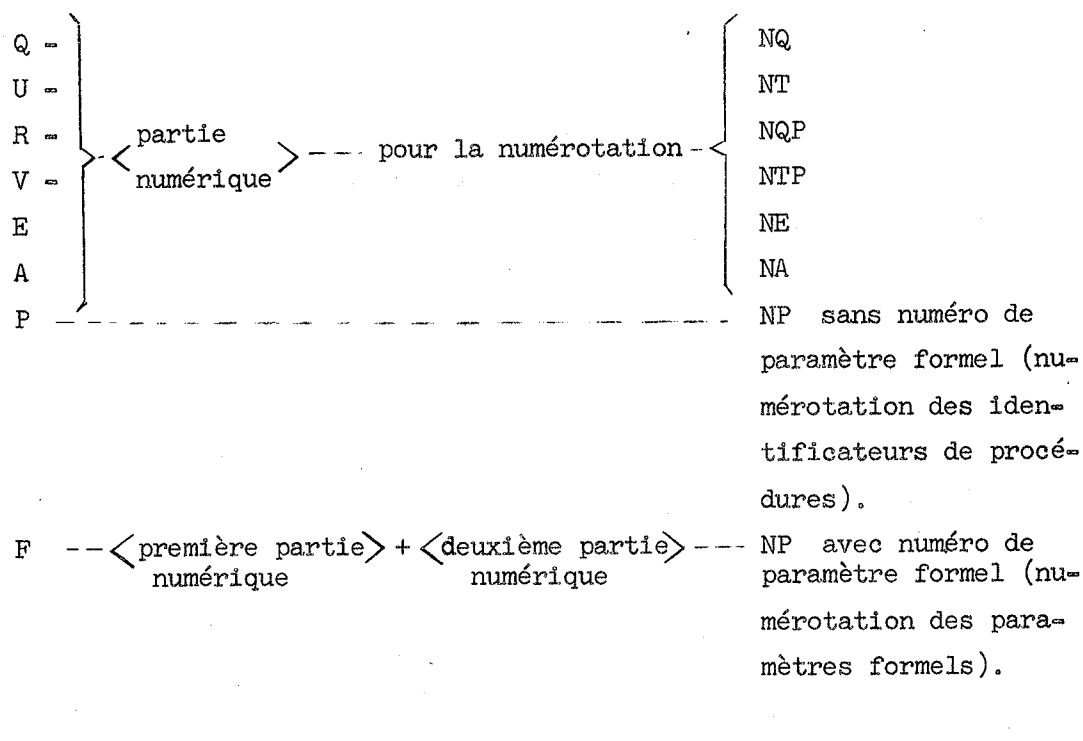
```
DEBUT  ENTIER  I, J  ;
POUR  I  :=  NC2 + 1  PAS  1  JUSQUA  L2  FAIRE
      POUR  J  :=  NC1 + 1  PAS  1  JUSQUA  L1  FAIRE
            SI  P2 [I, 1] = P1 [J, 1] ALORS
                DEBUT
                SI  P2 [I, 2 ] =  P1 [ J, 2 ] ALORS
                    DEBUT
                    C [P2 [ I, 3 ] ] :=  P1 [ J, 3 ]  ;
                    P2 [ I,1 ] :=  P2 [ I,2 ] :=  P2 [ I,3 ] := 0
                    FIN
            FIN  ;
POUR  I  :=  NC2 + 1  PAS  1  JUSQUA  L2  FAIRE
A  : SI  P2 [ I,1 ] = 0  ALORS
     DEBUT
     SI  I  =  L2  ALORS  DEBUT
                          L2  :=  L2 - 1  ;
                          ALLER A  FIN  VIDAGE  FIN  ;
     P2 [I, 1] :=  P2 [L2, 1]  ;
     P2 [I, 2] :=  P2 [L2, 2]  ;
     P2 [I, 3] :=  P2 [L2, 3]  ;
     P2 [L2,1] :=  P2 [L2, 2]  := P2[L2, 3]  :=  0  ;
     L2  :=  L2 - 1  ;
     ALLER A  A  FIN  ;
FIN  VIDAGE  :
     FIN  de la procédure  VIDAGE  P2  ;
```

E2 - GESTION DE LA TABLE D'EQUIVALENCES

PROCEDURE CONVERSION NC1 ;

DEBUT

COMMENTAIRE On a vu que E_T avait pour rôle secondaire de délivrer, après la liste du programme P_S, et sur le même support physique, une table de correspondance entre les identificateurs écrits dans P_S et les symboles qui seront utilisés pour les représenter dans P_M.
Ces symboles possèdent une partie alphabétique (un ou deux caractères) et une (ou deux) partie (s) numérique (s).
Les parties numériques correspondent aux numérotations déjà mentionnées. Les parties alphabétiques indiquent de quelle numérotation il s'agit.



Dans ce dernier cas, la première partie correspond à la numérotation NP (numéro de procédure) et la deuxième à la numérotation introduite par COPF (numéro de paramètre formel à l'intérieur de la procédure).

Ces symboles sont construits à l'aide de trois sous-programmes appelés par E_T à la fin de chaque niveau de bloc .

Le premier de ces sous-programmes retransforme les douze caractères de l'identificateur (deux premières mémoires des lignes de P1) du code D en code IBM .

Le deuxième calcule, étant donné le code de l'identificateur (troisième mémoire des lignes de P1) la partie alphabétique du symbole correspondant.

Le troisième, dans les mêmes conditions, calcule la ou les parties numériques du symbole équivalent.

Enfin, ces trois informations sont rassemblées dans la zone d'impression Z_2 et imprimées sur le support physique intermédiaire de la table d'équivalences (bande magnétique) par un sous-programme de gestion (IMPRESSION Z_2).

Remarquons qu'un mot de contrôle, placé en tête de Z_2 à chaque fin de niveau de bloc permettra à la séquence d'impression finale de la table de distinguer ces différents niveaux de nomenclature ; <u>FIN</u> ;

F - TABLES DE TRAITEMENT

L'ensemble E_T dispose de quatre tables lui permettant d'associer un symbole de base, un identificateur de fonction standard ou un entier sans signe à une unité syntaxique complète et éventuellement à une adresse de traitement.

F1 - TABLE DES SYMBOLES COURTS OU T.A.

Elle est formée de vingt deux lignes de un mot chacune.

Ces entrées sont disposées dans l'ordre de valeur en code D des symboles délimiteurs courts d'ALGOL.

Chaque mot comporte en positions S1 - 20 le code du symbole et en positions 21 - 35 une adresse de traitement. Pour les symboles sans traitement spécial, cette adresse correspond au point B de l'organigramme général (planche 5).

La branche -IKLMNP - de cet organigramme est pratiquement résumée dans cette adresse.

Si on cherche dans T.A. un symbole inexistant, il y a erreur (ER13).

F2 - TABLE DES SYMBOLES LONGS OU T.B

Elle est formée de lignes de trois mots chacune.

On peut supprimer ou rajouter des lignes à volonté.

Les deux premières mémoires de chaque ligne contiennent douze caractères alphabétiques représentant un symbole long, sans les ' , en code D .

La troisième mémoire est analogue à celle qui forme chaque ligne de T.A.

Chaque symbole long compris par le compilateur apparaît dans cette table et on peut ajouter de cette façon tous les symboles désirés à la liste donnée en Appendice II - 4 .

Remarquons que le même symbole peut apparaître dans T.A. et dans T.B : ainsi le symbole ; s'écrit :: en symbole court et 'PV' en symbole long. De même, [s'écrit .(en symbole court, 'OCRO' ou 'OBRA' en long, etc ...

La table B est balayée quand on trouve le caractère ' début de symbole long (SB de planche 5).

Actuellement, cette table comporte la version officielle française des symboles longs [32] , leur version officielle ALGOL- ALCOR [16] , des abréviations françaises et anglaises et des synonymes français.

Les symboles ')' et 'FCODE' apparaissant dans cette table aiguillent le traitement vers les erreurs ER11 et ER12 , leur détection par SB étant en principe impossible.

F3 - TABLE DES FONCTIONS STANDARD OU TFS

Elle est formée de lignes de trois mots analogues à celles de T.B., sauf la dernière partie des troisièmes mémoires.

On peut supprimer et rajouter des lignes à volonté.

Les troisièmes mémoires de chaque ligne contiennent en positions 18 - 35 la numérotation de l'identificateur correspondant de sorte que ces troisièmes mémoires constituent des unités syntaxiques complètement codées, placées dès l'entrée dans PC_1 dans le niveau le plus bas de P1.

On trouvera en Appendice II - 5 la liste des fonctions actuellement disponibles et leurs caractéristiques générales.

La description détaillée de chaque fonction est faite en troisième partie.

F4 - TABLE DES NOMBRES PURS OU TNP

Cette table est construite au fur et à mesure de l'édition et constituée de mémoires comprenant chacune un entier sans signe écrit en code décimal à quatre positions binaires.

Traduite en code binaire pur au cours de la séquence d'arrêt de traitement, TNP est placée en partie haute de la mémoire rapide, juste avant C , et transmise avec la chaîne codée au programme PC_2 qui la complète avec des nombres purs complets.

G - SEQUENCE D'ARRET DE TRAITEMENT

Cette séquence est activée par E_G en détectant les six caractères FALGOL au début d'une carte de P_S. Elle procède aux opérations suivantes, décrites sous forme ALGOL.

G1 - VERIFICATIONS DES COMPTEURS ET DES AIGUILLAGES

```
ARRET  TRAITEMENT  :  DEBUT
                      ENTIER  I, J  ;
                      SI  I1  =  2  ALORS  ER32  ;
                      SI  I1  =  3  ALORS  ER33  ;
                      SI  I2  =  0  ALORS  ALLER A  SUITE  ;
                      SI  I2  <  9  ALORS  ER34  ;
                      SI  I2  = 11  ALORS  ER35  ;
SUITE  :              SI CODF ≠  0  ALORS  ER14  ;
                      SI COPAR ≠ 0  ALORS  ER15  ;
                      SI COBRA ≠ 0  ALORS  ER16  ;
                      SI COCH  ≠ 0  ALORS  ER9   ;
```

G2 - TRAITEMENT DU DERNIER NIVEAU DE P1

COMMENTAIRE On vérifie que P1 et P2 ne comportent pas plus d'un niveau chacune.
On procède éventuellement au dernier vidage de P2, et si elle n'est pas entièrement vide après, il y a identificateurs non déclarés.

Dans le cas normal, on efface P1 et on imprime une marque de fin sur la table d'équivalences ;

```
                SI  NC1 ≠ 1  OU  NC2 ≠ 1  ALORS  ER26  ;
                SI  I2 ⩽ 0  ALORS  ER26  ;
                SI  I2  =  1  ALORS  ALLER A  P2  VIDE  ;
                VIDAGE  P2  ;
                SI  I2  ≠  1  ALORS  ALLER A  IDENT NON DECLARES  ;
P2  VIDE  :     SI  L1  ⩽  0  ALORS  ER26  ;
                POUR  I  :=  L1  PAS  - 1  JUSQUA  1  FAIRE
                      POUR  J  :=  1  PAS  1  JUSQUA  3  FAIRE
                                  P1 [ I, J ] := 0  ;
                      Z2 [ 1 ]  :=  MARQUE  ;
                      IMPRESSION  Z2  ;
```

G3 - CONVERSION DES NOMBRES PURS

COMMENTAIRE On convertit en code binaire pur les entiers sans signe contenus dans la table TNP, pour obtenir la table définitive qui sera communiquée à PC_2 .
Notons que dans cette conversion, les parties entières restent cadrées à droite dans les mémoires de TNP et les parties décimales à gauche ;

G4 - ECRITURE DE LA TABLE D'EQUIVALENCES

COMMENTAIRE On imprime alors, à l'aide d'un sous-programme de gestion, les lignes de la table d'équivalences sur le support physique où est listé P_S .

Cette écriture se fait en lisant le fichier de la table dans Z_2 , ligne par ligne. On groupe ces lignes par trois dans Z_1 et on imprime Z_1 . Quand on trouve dans la table une marque de fin de niveau, on imprime Z_1 avec le nombre de lignes qu'elle contient, puis une ligne blanche. L'écriture est terminée quand on trouve la marque de fin de table ;

G5 - FIN NORMALE DE PC_1

COMMENTAIRE Si l'on n'a détecté aucune erreur au cours de l'édition et si un point-virgule a été perforé après le dernier FIN de P_S , il y a passage de PC_1 à PC_2 , avec conservation en mémoire rapide de la chaîne codée C , de la table d'entiers TNP , et des valeurs maximum atteintes par les numérotations NQP, NTP et NN, qui serviront à PC_2 , les deux premières pour des réservations de mémoires, la troisième pour continuer à construire la table de nombres purs.
Si le dernier FIN de P_S n'est pas suivi d'un point-virgule, l'aiguillage AIG2 est positionné sur la valeur 10 (commentaire après FIN) et une convention interne dit que l'utilisateur ne désire qu'une édition de P_S .
Le contrôle est alors rendu au système (Cf. troisième partie) ;
SI ER $\neq$ 0 ALORS ARRET ERREUR ;
SI I2 = 10 ALORS EDITION SEULEMENT ;
PASSAGE A PC_2 ;

COMMENTAIRE Les procédures ARRET ERREUR (Cf. Chapitre III), EDITION SEULEMENT et PASSAGE A PC_2 sont écrites en code machine et leur constitution dépend essentiellement du système employé ;

G6 - FIN AVEC IDENTIFICATEURS NON DECLARES

IDENT NON DECLARES :

COMMENTAIRE S'il a été impossible de vider la pile P2, on imprime ce qu'elle contient encore sous forme de lignes formées chacune d'un identificateur (douze caractères pris dans les deux premières mémoires de chaque ligne de P2, et retransformés en code IBM) suivi du numéro d'unité syntaxique (troisièmes mémoires des lignes de P2) où il a été trouvé.
Notons que de cette façon chaque occurrence d'un identificateur dont on ne trouve pas la déclaration est signalée à l'utilisateur. Les erreurs d'écriture ou de perforation des identificateurs, en particulier, sont facilement décelées par ce moyen.
Comme dans ARRET ERREUR, le contrôle est rendu au système sans passer par PC_2 ;
FIN du programme PC_1 ;

CHAPITRE III

PROGRAMMES D'ERREURS

Les différentes erreurs morphologiques, syntaxiques ou d'écriture détectées dans le traitement du programme P_S conduisent toutes à l'impression d'un diagnostic précis dans les cinq mémoires de Z_1 qui suivent celles où est placée la carte en train d'être analysée. En même temps est placé en vingtième mémoire de Z_1 le NUS de l'erreur, qui permet à l'utilisateur de la localiser facilement dans la ligne.

Dans une première version, tous les programmes d'erreur, après l'écriture de ce libellé, arrêtaient l'édition et se reliaient à E_G par un signal de fin de traitement avec erreur qui activait la séquence d'arrêt de traitement sans passage du contrôle à PC_2 .

Dans la version actuelle, un grand nombre d'erreurs dites "non fatales" n'empêchent pas l'édition de continuer, après correction plus ou moins exacte de l'erreur. Ceci permet de détecter plusieurs erreurs en une seule édition et d'avoir dès le premier essai, en général, soit la table d'équivalences de P_S, soit ses identificateurs non déclarés.

Quelques erreurs cependant restent considérées comme "fatales", ce sont en particulier les erreurs provoquées par

le non - respect des conditions d'encombrement.

En plus certaines configurations peuvent amener une ou plusieurs erreurs erreurs "non fatales" à entrainer une erreur telle que l'éditeur ne peut pas poursuivre son travail. Il y a alors sortie par ER26, qui, dans le cas où il n'y a pas encore eu d'erreur "non fatale" , ne peut correspondre qu'à une erreur de PC_1 lui-même.

Pour ces erreurs "non fatales" la ligne correspondante est listée avec le dignostic et le NUS convenables, mais elle peut l'être plusieurs fois si plusieurs erreurs y sont décelées. Dans tous les cas la ligne est listée par E_G à la fin du traitement de son dernier caractère, sans diagnostic et avec le NUS correspondant à son dernier symbole traité.

Un indicateur (ER) est mis à différent de zéro dans le cas où au moins une erreur "non fatale" est survenue au cours du traitement. C'est cet indicateur qui permet à la séquence d'arrêt de traitement de décider qu'elle doit passer le contrôle à PC_2 ou non.

Nous donnons ci-dessous, sous forme de procédures supposées appartenir au programme ALGOL équivalent à PC_1 , la liste des programmes d'erreurs avec, pour chacun le libellé correspondant, la cause certaine ou la plus probable de l'erreur et la correction qui lui est appliquée si elle est considérée comme "non fatale" .

<u>DEBUT</u>

<u>COMMENTAIRE</u> Nous supposons déclaré en tête du programme un sous-programme de gestion LIBELLE (X), dont le paramètre X est une chaîne.

Ce sous-programme remplit les six dernières mémoires de Z_1 avec la chaîne X et le NUS courant, imprime Z_1 , réinitialise ces six mémoires et fait

ER non nul.
On notera que certaines procédures d'erreur ne reviennent pas à l'endroit de leur appel.
Ce sont alors de simples instructions composées.
Les libellés correspondant aux erreurs "fatales" sont précédés d'un caractère * ;

PROCEDURE ER1 ;

COMMENTAIRE Douze caractères après ' sont emmagasinés en MEMSB1 et MEMSB2 sans trouver un autre caractère ' . On reprend le traitement à partir du premier caractère spécial qui suit (C de la planche 6) ;

DEBUT

LIBELLE (' ERREUR APOSTROPHE (SYMBOLE DE BASE) ') ;

CARACTERE :

EG ;

SI CAR < CARACTERE SPECIAL ALORS ALLER A CARACTERE ;

I2 := 0 ;

ALLER A C

FIN ;

PROCEDURE ER2 ;

COMMENTAIRE Les caractères situés entre deux ' emmagasinés en MEMSB1 et MEMSB2 ne correspondent à aucun symbole dans T.B.
Si ces caractères comportent au moins un caractère spécial, on suppose qu'il manque un caractère ' et que le deuxième détecté est en fait le commencement d'un autre symbole long. On le traite ainsi.
Sinon on suppose qu'il y a eu faute d'orthographe et on continue le traitement après le deuxième ' ;

```
    LIBELLE (' ERREUR DE SYMBOLE DE BASE OU APOSTROPHE ') ;

PROCEDURE ER3 ;
    COMMENTAIRE FIN est suivi d'un symbole illégal. On traite ce
                symbole ;
    LIBELLE (' ERREUR COMMENTAIRE APRES FIN ') ;

PROCEDURE ER4 ;
    COMMENTAIRE Un entier sans signe est suivi d'une lettre. On
                traite cette lettre ;
    LIBELLE (' ERREUR SYNTAXE NOMBRE PUR ') ;

PROCEDURE ER5 ;
    COMMENTAIRE Un entier sans signe comporte plus de neuf chif-
                fres en partie entière, ou bien est plus grand
                que 2^27 , ou bien est suivi de . alors qu'il est
                déjà en partie décimale, ou bien ne contient au-
                cun chiffre après .  . On continue le traite-
                ment avec le premier caractère qui n'est pas un
                chiffre ;
    DEBUT
    LIBELLE (' ERREUR DE NOMBRE PUR OU DE POINT ') ;
CARACTERE : SI CAR < CHIFFRE MINI OU CAR > CHIFFRE MAXI
            ALORS DEBUT I2 := 0 ;
                        ALLER A C FIN ;
          EG ;
          ALLER A CARACTERE FIN ;

PROCEDURE ER6 ;
    COMMENTAIRE Un entier sans signe est suivi de : et COBRA
                est nul. Il s'agit d'une étiquette numérique,
                on traite : ;
    LIBELLE ('ETIQUETTE NUMERIQUE ILLEGALE ') ;
```

```
PROCEDURE  ER7  ;
    COMMENTAIRE  Au début d'un corps de procédure, tous les para-
                 mètres formels ne sont pas spécifiés. On simule
                 ces spécifications en annulant le compteur de pa-
                 ramètres formels  ;
    DEBUT  LIBELLE  ('MANQUE  SPECIFICATION  DANS  EN-TETE ')  ;
           COPF  :=  0  FIN  ;

PROCEDURE  ER8  ;
    COMMENTAIRE  Le symbole  )  est suivi d'un caractère illégal
                 ou d'une chaîne de caractères ne formant pas un
                 commentaire dans une liste de paramètres. On sup-
                 pose que le symbole est  )  arithmétique ou de
                 fin d'instruction procédure, on le traite et on
                 passe au caractère suivant  ;
    DEBUT  LIBELLE  ( ' ERREUR  SYNTAXE  PARENTHESE ')  ;
           COPAR  :=  COPAR - 1  ;
           SI  COPAR  < 0  ALORS  ER15  ;
           I2  :=  0  ;
           ALLER A  C  FIN  ;

PROCEDURE  ER9  ;
    COMMENTAIRE  On a trouvé  '  dans une chaîne, sans trouver le
                 symbole  ')'  ou  '('  . On suppose que c'est la
                 fin de la chaîne  ;
    LIBELLE  ('ERREUR  DE  CROCHET  DE  CHAINE  OU  APOSTROPHE ')  ;

PROCEDURE  ER10  ;
    COMMENTAIRE  On a trouvé  '  dans un corps de procédure en
                 code machine, sans trouver  'FCODE'  . On suppo-
                 se que c'est la fin du corps de procédure  ;
    LIBELLE  ( ' ERREUR  DE  CROCHET  DE  CODE  OU  APOSTROPHE ')  ;
```

```
PROCEDURE  ER11  ;
     COMMENTAIRE  On a trouvé 'FCODE' sans avoir trouvé 'CODE' .
                  On ignore 'FCODE'  ;
     LIBELLE  ( ' FCODE  ILLEGAL  A  CET  ENDROIT ')  ;

PROCEDURE  ER12  ;
     COMMENTAIRE  On a trouvé  ')'  sans avoir trouvé  '('  . On
                  ignore  ')'  ;
     LIBELLE  ( ' FIN  DE  CHAINE  ILLEGAL  A  CET  ENDROIT ' )  ;

PROCEDURE  ER13  ;
     COMMENTAIRE  On a trouvé un symbole court inexistant dans
                  T.A. C'est une erreur du compilateur ou de la
                  machine. On arrête le traitement  ;
     DEBUT  LIBELLE  (' * ERREUR  SYMBOLE  DE  BASE  (MACHINE) ')  ;
            ARRET  ERREUR  FIN  ;

PROCEDURE  ER14  ;
     COMMENTAIRE  Le compteur de débuts - fins a pris une valeur
                  illégale. On l'annule  ;
     DEBUT  LIBELLE  (' ERREUR  COMPTE  DE  DEBUT - FIN ' )  ;
            CODF  :=  0  FIN  ;

PROCEDURE  ER15  ;
     COMMENTAIRE  Le compteur de parenthèses a pris une valeur il-
                  légale. On l'annule  ;
     DEBUT  LIBELLE  ( ' ERREUR  DE  PARENTHESE ')  ;
            COPAR  :=  0  FIN  ;

PROCEDURE  ER16  ;
     COMMENTAIRE  Le compteur de crochets a pris une valeur illé-
                  gale. On l'annule  ;
     DEBUT  LIBELLE  ( ' ERREUR  DE  CROCHET ' )  ;
            COBRA  :=  0  FIN  ;
```

PROCEDURE ER17 ;

COMMENTAIRE Une déclaration comporte plus d'un déclarateur de type, PERSISTANT est écrit après un déclarateur de type, ETIQUETTE ou CHAINE interviennent dans une déclaration. On traite le caractère suivant ;

DEBUT LIBELLE (' DECLARATION FAUSSE OU ILLEGALE ') ;
ALLER A B FIN ;

PROCEDURE ER18 ;

COMMENTAIRE Une spécification comporte plusieurs déclarateurs de types ou comporte PERSISTANT. On traite le caractère suivant ;

DEBUT LIBELLE (' ERREUR DE SPECIFICATION ') ;
ALLER A B FIN ;

PROCEDURE ER19 ;

COMMENTAIRE Un paramètre formel est spécifié plus d'une fois, ou apparait plus d'une fois dans la partie VALEUR . Ou bien une partie VALEUR est écrite après une spécification, ou bien un identificateur apparaissant dans une spécification ou une partie VALEUR n'apparait pas dans la liste de paramètres formels correspondante. On ignore la spécification ;

DEBUT LIBELLE (' SPECIFICATION MULTIPLE OU ILLEGALE ') ;
I2 := 0 ;
ALLER A C FIN ;

PROCEDURE ER20 ;

COMMENTAIRE L'ensemble de gestion a lu une carte appartenant à un niveau du système supérieur à ALGOL. (Cf. troisième partie). Il y a arrêt de traitement et

```
                retour au système en simulant Edition seulement  ;
    DEBUT  LIBELLE  ( ' * ERREUR  CARTE  CONTROLE ' )  ;
           EDITION  SEULEMENT  FIN  ;

PROCEDURE  ER21  ;
    COMMENTAIRE  La table de nombres purs est pleine  ;
    DEBUT  LIBELLE  ( ' * TROP  DE  NOMBRES  PURS ' )  ;
           ARRET  ERREUR  FIN  ;

PROCEDURE  ER22  ;
    COMMENTAIRE  La pile  P1  est pleine  ;
    DEBUT  LIBELLE  ( ' * TROP  DE  DECLARATIONS ' )  ;
           ARRET  ERREUR  FIN  ;

PROCEDURE  ER23  ;
    COMMENTAIRE  La chaîne codée est pleine  ;
    DEBUT  LIBELLE  ( ' * PROGRAMME  TROP  LONG ' )  ;
           ARRET  ERREUR  FIN  ;

PROCEDURE  ER24  ;
    COMMENTAIRE  La pile  P2  est pleine  ;
    DEBUT  LIBELLE  (' * TROP  D'IDENTIFICATEURS  GLOBAUX ' )  ;
           ARRET  ERREUR  FIN  ;

PROCEDURE  ER25  ;
    COMMENTAIRE  On trouve un niveau de procédure en  P1  quand
                 il faudrait un niveau de bloc. On décrémente IDP
                 et on cherche le premier niveau de bloc  ;
    DEBUT  LIBELLE  ( ' ERREUR  SYNTAXE  FIN  DE  PROCEDURE ' )  ;
RECHERCHE  NIVEAU  BLOC  :  IDP  :=  IDP - 1  ;
RECHERCHE  NIVEAU        :  LC1  :=  L1  :=  LC1 - 1  ;
                            SI  P1 [ LC1,1 ]  = 0  ALORS  ALLER A  NIVEAU :
                            ALLER A  SI  P1 [ LC1,1 ]  = - 0  ALORS
                                          RECHERCHE  NIVEAU  BLOC
```

```
                                                    SINON
                                          RECHERCHE  NIVEAU  ;
NIVEAU  :                    FIN  ;

PROCEDURE  ER26  ;
     COMMENTAIRE  L'éditeur se trouve dans l'impossibilité de conti-
                  nuer. Si  ER  est nul, c'est une erreur de l'édi-
                  teur lui-même, sinon c'est que les erreurs déjà
                  décelées ont trop perturbé son fonctionnement.
                  Dans tous les cas, il y a arrêt du traitement  ;
     DEBUT  SI  ER  =  0  ALORS
                  LIBELLE  (' * ERREUR  PROGRAMME  EDITEUR.  STOP')
            SINON  LIBELLE ( '  POURSUITE  EDITION  IMPOSSIBLE  ' )  ;
            ARRET  ERREUR  FIN  ;

PROCEDURE  ER27  ;
     COMMENTAIRE  Un identificateur apparaissant dans une liste de
                  bornes n'est pas déclaré dans les blocs extérieurs.
                  Il est remplacé par une unité syntaxique imaginaire  ;
                  LIBELLE ('IDENTIFICATEUR NON DECLARE DANS UNE BORNE ')  ;

PROCEDURE  ER28  ;
     COMMENTAIRE  Un identificateur est déclaré plusieurs fois dans
                  le même niveau, ou apparait plus d'une fois dans une
                  liste de paramètres formels  ;
                  LIBELLE  ('DECLARATION  MULTIPLE')  ;

PROCEDURE  ER29  ;
     COMMENTAIRE  La numérotation  NP  dépasse  255  ;
     DEBUT  LIBELLE  ( ' * TROP  DE  PROCEDURES ')  ;
            ARRET  ERREUR  FIN  ;
```

```
PROCEDURE  ER30  ;
     COMMENTAIRE  Le compteur  COPF  dépasse  511  ;
     DEBUT  LIBELLE  (' * TROP  DE  PARAMETRES  FORMELS ' )  ;
            ARRET  ERREUR  FIN  ;

PROCEDURE  ER31  ;
     COMMENTAIRE  Une déclaration est de type PERSISTANT A. On si-
                  mule un déclarateur de type pour avoir une vraie
                  déclaration de  A  ;
     DEBUT  LIBELLE  ( ' DECLARATION  INCOMPLETE ' )  ;
            D  :=  1  ;
            INUM  :=  3  ;
            DECLARATION  ;
            ALLER A  FIN  IDENTIFICATEUR  FIN  ;

PROCEDURE  ER32  ;
     COMMENTAIRE  La fin du traitement arrive alors que l'aiguil-
                  lage  AIG1  est à deux  ;
                  LIBELLE  (' SYNTAXE  FIN  ILLEGALE  (CODE MACHINE)')  ;

PROCEDURE  ER33  ;
     COMMENTAIRE  Arrêt du traitement avec  AIG1  à trois  ;
                  LIBELLE  (' SYNTAXE  FIN  ILLEGALE  (COMMENTAIRE)')  ;

PROCEDURE  ER34  ;
     COMMENTAIRE  Arrêt  du traitement avec AIG2 à un, deux, trois,
                  quatre, cinq, six, sept ou huit  ;
                  LIBELLE  (' SYNTAXE FIN ILLEGALE (U.S.) ' )  ;

PROCEDURE  ER35  ;
     COMMENTAIRE  Arrêt du traitement avec  AIG2  à onze  ;
                  LIBELLE ('SYNTAXE FIN ILLEGALE (COMMENTAIRE L.P.)')  ;

PROCEDURE  ER36  ;
     COMMENTAIRE  Une déclaration ou une spécification contient un
                  nombre pur. On annule l'identificateur de déclaration  ;
     DEBUT  LIBELLE  ('DECLARATION OU SPECIFICATION NOMBRE PUR ')  ;
            D  :=  0  FIN  ;
```

CHAPITRE IV

CONCLUSION - DISCUSSION

A - RESULTATS OBTENUS

La réalisation rapide du passage éditeur complet nous a permis d'aborder la phase de compilation propre avec le maximum de facilité et d'efficacité.

L'expérience acquise sur ce premier passage est maintenant assez longue pour pouvoir affirmer qu'un programme P_S quelconque, respectant les conventions d'écriture et d'encombrement déjà citées, peut être rapidement transformé, grâce à ce programme, en une chaîne d'unités syntaxiques dans laquelle les règles morphologiques et syntaxiques immédiates du langage ne sont plus à considérer.

On trouvera en annexe des exemples de programmes édités, avec ou sans erreur, avec ou sans identificateurs non déclarés.

On remarquera dans ces exemples la numérotation NUS des unités syntaxiques : ce numéro, dans le cas d'une ligne sans erreur, est celui de la dernière unité syntaxique complète de la ligne qui n'est pas un identificateur ou un entier sans signe.

Dans le cas où une ligne renferme une erreur, ce numéro est celui de l'unité syntaxique où s'est produite l'erreur. La ligne est alors recopiée sans erreur au-dessous.

On remarquera également que chaque occurrence des identificateurs non déclarés quand il y en a, est indiquée à l'utilisateur.

Enfin, on notera la disposition, par niveau de nomenclature, de la table d'équivalences.

B - DISCUSSION

Malgré le bon fonctionnement de l'éditeur sous sa forme actuelle, nous pouvons envisager de lui apporter quelques améliorations, sur trois plans différents.

B1 - Amélioration des conventions d'écriture

Il a déjà été dit que les tables T.B et TFS pouvaient être augmentées à volonté, ce qui est un moyen facile d'accroître les possibilités de l'éditeur.

Les conventions portant sur les symboles courts à deux caractères sont essentiellement attachées au type de matériel employé pour la perforation des programmes. Tout changement dans ce matériel peut amener à reconstruire la table T.A.

Les restrictions les plus définitives sont certainement apportées par la constitution de la chaîne codée C : la présence d'un nombre limité de positions binaires dans un mot mémoire et la simplicité de décodification souhaitée pour PC_2 limitent obligatoirement le nombre de déclarateurs et de spécifieurs que l'on peut ajouter au langage, ainsi que les maxima des numérotations de procédures et de paramètres formels.

Enfin, l'interdiction d'employer dans P_S des étiquettes purement numériques étant liée à la méthode employée, on ne peut envisager de la lever sans modifier profondément les

algorithmes de traitement.

B2 - Limites d'encombrement

Ces limites portent sur la chaîne codée C , la table TNP, et les piles P1 et P2.

On peut à volonté modifier ces encombrements, sans que leur total dépasse - actuellement - vingt et un mille cinq cents mémoires.

Chaque installation utilisant le compilateur peut donc à son gré, modifier les rapports d'encombrement que nous avons indiqués. Il est probable que ces rapports deviendront immuables après qu'une longue période d'exploitation en ait fait ressortir les valeurs optima.

Enfin, on peut envisager de modifier l'algorithme de traitement des niveaux d'instructions composées (Cf. procédure FIN (X)) de façon à réduire très sensiblement l'encombrement de la pile P2. Cette modification est actuellement à l'étude.

B3 - Programmes d'erreurs

L'effort des études actuelles porte sur l'amélioration des "corrections" effectuées par les programmes d'erreurs, en particulier les procédures ER1 et ER2.

Actuellement, toute erreur de perforation sur le caractère ' précédant ou suivant un symbole long supprime le traitement de ce symbole.

On peut envisager que les procédures ER1 et ER2 analysent caractère par caractère les mémoires MEMSB1 et MEMSB2 afin d'y détecter la présence d'un tel symbole. Ceci est particulièrement important pour les erreurs sur DEBUT et sur FIN, qui entraînent souvent l'impossibilité de continuer l'édition.

Enfin, on peut envisager de mettre en oeuvre un programme complet de diagnostic d'erreurs, à l'aide par exemple d'une matrice de succession, et au prix d'un allongement très sensible du temps d'édition.

- DEUXIEME PARTIE -

P A S S A G E C O M P I L A T E U R

CHAPITRE I

GENERALITES

A - LES FONCTIONS DU PASSAGE COMPILATEUR

A1 - LE ROLE DU COMPILATEUR - PROBLEMES D'OPTIMISATION

Le programme compilateur PC'_2 a pour rôle de transformer la chaîne codée C fournie par le programme éditeur PC_1 en une suite d'instructions écrites en code machine MAP [26, 27, 28], traduisant exactement les instructions ALGOL du programme P_S.

Le programme ainsi obtenu (P_M) est appelé programme "généré" et fait l'objet, en général, d'une exécution qui suit immédiatement sa compilation.

Le nombre de programmes P_M équivalents au programme P_S n'est pas limité.

En particulier, on peut imaginer d'écrire directement en langage machine de tels programmes.

Nous supposerons qu'il existe parmi ceux-ci un programme au moins, que nous appellerons P, qui satisfait le mieux aux exigences d'efficacité et de concision demandées par l'utilisateur.

Le programme P_M écrit par le compilateur doit alors, non seulement traduire exactement P_S, mais aussi s'approcher le plus possible du programme optimum P.

Cette recherche de l'optimum de P_M a lieu sur deux

plans :

1°) L'encombrement de P_M et de ses mémoires de travail doit être minimum.

2°) La vitesse d'exécution de P_M doit être maximum.

Nous nous sommes efforcés de respecter la condition d'encombrement de P_M et de ses mémoires. Pour cela et sans rien ôter à la généralité du langage L_S , des algorithmes ont été construits que nous décrirons.

Ces algorithmes portent principalement sur la gestion des registres opérateurs de la machine et des mémoires de manoeuvre nécessaires à la compilation des expressions arithmétiques.

En revanche, il parait inconcevable qu'un compilateur puisse, sans diminuer la généralité de L_S , fournir à l'utilisateur un programme P_M équivalent à P pour l'efficacité de l'exécution.

En particulier, les aspects dynamiques du langage ALGOL (tableaux à bornes variables, fonctions récursives) rendent la condition de vitesse d'exécution très difficile à respecter.

Les solutions adoptées dans notre compilateur l'ont été pour respecter au maximum la généralité de L_S . Ce n'est que dans le cas où plusieurs solutions s'offraient que nous avons retenu celle qui donnait à P_M la meilleure efficacité.

On verra en conclusion de cette thèse la solution que nous proposons aux utilisateurs pour résoudre ce problème de la manière la moins astreignante qui semble possible.

Enfin, le troisième but recherché en écrivant ce programme a été la rapidité de la compilation des programmes P_S . Ce but a pu être atteint grâce aux possibilités de simultanéité de traitement et de sortie des résultats sur les machines étudiées.

A2 - DONNEES DU COMPILATEUR

Le programme PC_2 a pour données la chaîne codée C du programme P_S et la table d'entiers TNP construites par l'éditeur, ainsi que les valeurs maxima atteintes par les numérotations NQP, NTP et NN en fin d'édition (Cf. première partie).

Ces données sont en mémoire rapide lors du chargement de PC_2 .

Des algorithmes de traitement et de la séquence d'arrêt de traitement exposés dans la première partie, il résulte que le reste de la mémoire rapide est nul et qu'en particulier, la fin de la chaîne codée est marquée par un mot nul.

Nous verrons (troisième partie) que les données du compilateur comportent aussi quatorze mémoires du haut de la mémoire rapide, où est emmagasinée par PC_1 la carte C_S qui suit la fin de P_S , et quatorze mémoires du bas de la mémoire (partie système) qui contiennent la carte C_M permettant aux autres parties du système de prendre en charge le programme P_M généré.

Enfin, plusieurs indicateurs I_S , dont le nombre peut être augmenté à volonté, sont communiqués à PC_2 par le système (troisième partie). Ils traduisent les indications ou "options" données par l'utilisateur au système pour la compilation et l'exécution de son programme et leur fonctionnement est expliqué dans le texte au fur et à mesure de leur utilisation.

A3 - SORTIES DU COMPILATEUR

PC_2 écrit sous forme alphabétique sur une bande magnétique "de génération" B_G les instructions MAP qu'il construit au fur et à mesure de l'analyse de la chaîne codée.

Ces instructions sont générées dans une zone tampon

Z_G qui peut en contenir un nombre égal au maximum que peut engendrer une seule opération du programme P_S . La zone Z_G est vidée à la fin de chaque séquence générative dans les zones intermédiaires d'impression attribuées à B_G par le système. Ces zones sont à leur tour enregistrées sur B_G de façon à assurer la simultanéité du traitement et de l'enregistrement [28] .

Notons que cette méthode confère au compilateur une grande rapidité et qu'elle n'est applicable que grâce au fonctionnement "à un seul balayage" du passage PC_2 (Cf. B).

Les instructions générées du programme P_M sont suivies sur B_G d'une dernière instruction qui assurera le retour au système en fin d'exécution de P_M .

Puis sont générées les réservations de mémoires nécessaires à P_M et enfin la carte marquant la fin de P_M .

Enfin, P_M est lui-même encadré, sur B_G , par la carte C_M de prise en charge par le système, placée en tête, et, en queue, par la carte C_S qui suivait P_S sur le support physique d'entrée.

Dans le cas où le compilateur détecte une erreur syntaxique ou sémantique dans C , il y a toujours arrêt de la génération et sortie d'un numéro d'erreur et d'un numéro d'unité syntaxique qui sont ensuite interprétés par le système pour écrire un libellé approprié du type de ceux imprimés par PC_1 .

B - PRINCIPE DE FONCTIONNEMENT

B1 - PRINCIPE GENERAL

Le compilateur PC_2 effectue un seul balayage de la chaîne codée C , de gauche à droite, avec mise en réserve des éléments non utilisables immédiatement.

La mise en oeuvre nécessite l'ouverture de plusieurs piles de mémoires et la définition d'une table de hiérarchies dans laquelle doivent figurer tous les délimiteurs susceptibles d'avoir une priorité déterminée dans l'analyse d'une instruction ALGOL.

Pour le fonctionnement d'une pile de mémoires, nous renvoyons à la première partie et à la bibliographie déjà citée.

B2 - EXEMPLE DE FONCTIONNEMENT : EXPRESSIONS ARITHMETIQUES SIMPLES

La méthode de compilation par piles de mémoires et table de hiérarchies est couramment employée pour la génération des expressions arithmétiques ordinaires [19, 20] et elle s'expose aisément de la façon suivante :

Soit E1 la suite de mots machines formant la chaîne codée du programme P_S édité. Soit $(E1)_I$ la première unité syntaxique à analyser.

Soit E2 une pile d'opérateurs, $(E2)_J$ représentant sa dernière mémoire occupée.

Soit E3 une pile d'adresses, $(E3)_K$ représentant sa première mémoire libre.

Soit alors une expression comportant les signes d'opérations :

↑	auquel on attribue la hiérarchie	4
x /	auxquels on attribue la hiérarchie	3
+ -	auxquels on attribue la hiérarchie	2
) ;	auxquels on attribue la hiérarchie	1
(	auquel on attribue la hiérarchie	0

L'organigramme de la planche 7, où P [A] représente la hiérarchie de l'opérateur A, permet d'effectuer la compilation de l'expression. Il se traduit par l'algorithme suivant :

1 - Si la première unité syntaxique à analyser est un opérateur autre que ; , placer cet opérateur dans la pile E2 et passer à l'unité syntaxique suivante.

2 - Si l'unité syntaxique est ; , fin de l'analyse de l'expression

3 - Si l'unité syntaxique est un opérande, le placer en $(E3)_K$. Regarder si la priorité de l'opérateur placé au sommet de la pile E2 est supérieure ou égale à celle de l'opérateur qui suit l'opérande dans E1. Si non, passer à l'unité syntaxique suivante. Si oui, générer l'opération correspondant à l'opérateur situé au sommet de E2 et portant sur les deux dernières adresses de E3. Remplacer celles-ci par une adresse de manoeuvre. Descendre d'une ligne dans E2 et E3, et recommencer le test des priorités.

4 - Si l'unité syntaxique est) , vérifier que l'opérateur situé au sommet de E2 est (, descendre d'une ligne dans E2 et passer au test de priorités comme en 3 .

Ainsi l'expression :

A x B + (C x D - E / (F + D)) ;

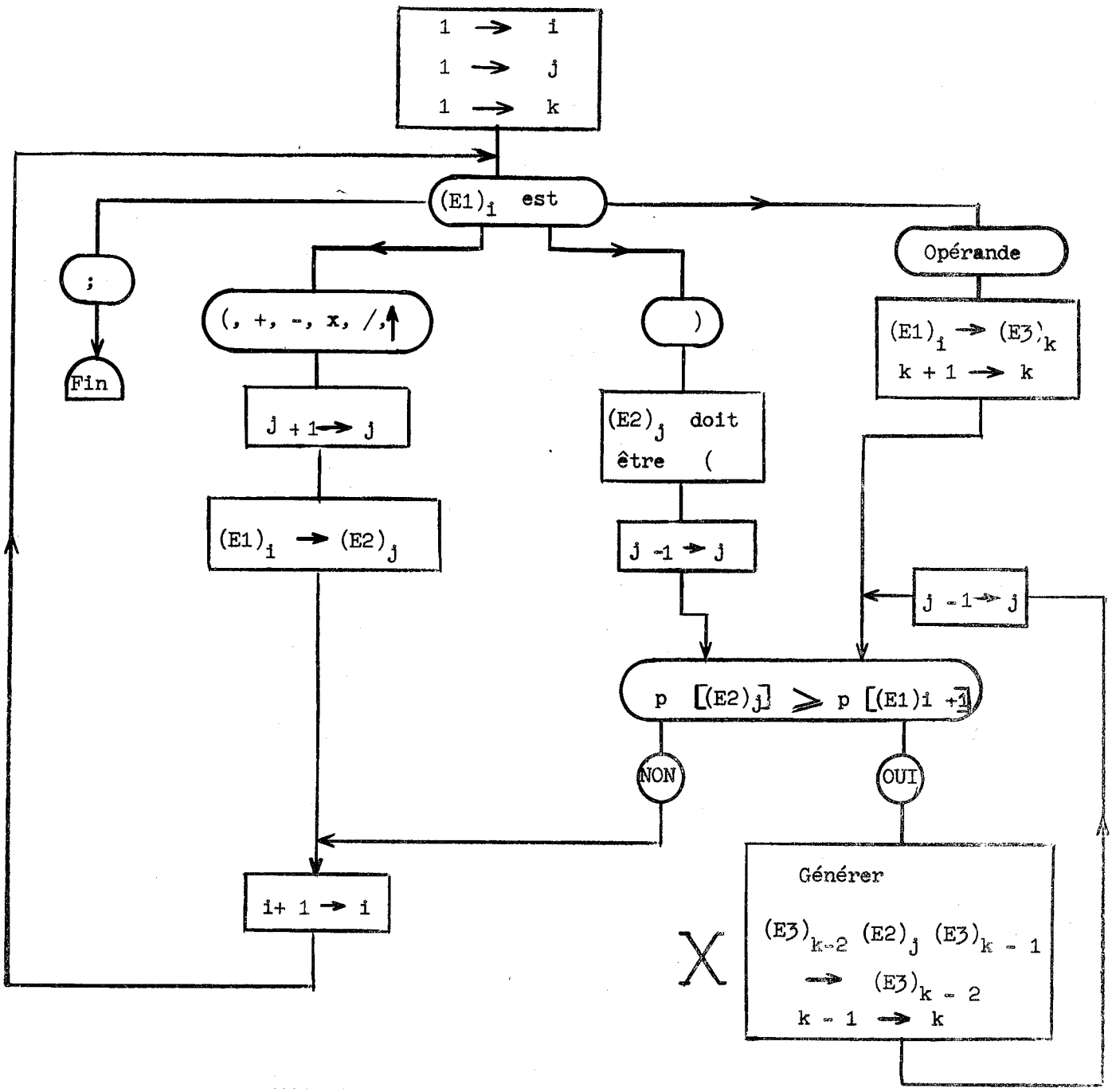


Planche 7

ORGANIGRAMME DE COMPILATION D'UNE EXPRESSION ARITHMETIQUE SIMPLE

est générée de la façon suivante :

A	x	B,	résultat en	M1
C	x	D,	" "	M2
F	+	D,	" "	M3
E	/	M3,	" "	M4
M2	-	M4,	" "	M5
M1	+	M5,	" "	M6

Fin

La programmation effective de l'algorithme comporte en fait différentes optimisations, en particulier celle de la gestion du bloc de mémoires de manoeuvre :

- Si les deux opérandes sont des variables simples, il y a création d'une nouvelle mémoire de manoeuvre, où est placé le résultat.

- Si l'un des deux opérandes est une mémoire de manoeuvre, le résultat utilise la même mémoire.

- Si les deux opérandes sont des mémoires de manoeuvre, le résultat utilise celle de niveau inférieur et libère la seconde.

Pour la même expression, on obtient alors :

A	x	B,	résultat en	M1
C	x	D,	" "	M2
F	+	D,	" "	M3
E	/	M3,	" "	M3
M2	-	M3,	" "	M2
M1	+	M2,	" "	M1

B3 - BLOC D'INDICATEURS DES MEMOIRES DE MANOEUVRE

On voit sur l'exemple qui précède que l'utilisation optimisée de l'ensemble des mémoires de manoeuvre M repose à son tour sur le principe d'une pile de mémoires.

Cette pile est à prendre au sens dynamique, c'est à dire qu'elle n'est effective qu'à l'exécution du programme P_M .

Cependant, le compilateur doit simuler pendant le passage PC_2 le fonctionnement de cette pile, afin de pouvoir générer les champs variables d'instructions de P_M relatives à ces mémoires.

Pour cela on définit un bloc d'indicateurs, appelé BIM, dont l'état indique à tout moment de la compilation le nombre et l'identité des mémoires de manoeuvre disponibles ou non pour P_M . Chaque indicateur de BIM correspond à une mémoire dans P_M , le m ème indicateur étant associé à la mémoire M - m .

M est ainsi dans P_M le symbole générique des mémoires de manoeuvre, comme Q est celui des quantités simples (cf. première partie).

Chaque indicateur peut prendre les valeurs zéro ou un. Zéro indique que la mémoire correspondante est libre, un qu'elle est occupée.

L'algorithme de réservation des mémoires de manoeuvre est alors très simple :

Quand PC_2 a besoin pour P_M d'une mémoire de manoeuvre, il cherche dans BIM l'indicateur de plus petit rang m qui soit nul. La mémoire cherchée est M - m.

Quand la mémoire M - m ne sert plus (son contenu ayant fait l'objet d'une opération), l'indicateur m est annulé.

Cependant, on verra que la syntaxe du langage L_S oblige à créer dans la pile de mémoires de manoeuvre, et donc dans BIM, des niveaux d'utilisation au-dessous desquels les mémoires, mê-

mes libérées, sont inutilisables. C'est le cas par exemple des mémoires utilisées dans un corps de procédure, qui doivent rester inemployées dans toutes les instructions situées dans la portée de la procédure.

On définit alors un niveau courant NIM au-dessous duquel, à tout moment, il est interdit à PC_2 de prendre une mémoire. Ce niveau est mis à jour chaque fois que la structure syntaxique du programme l'exige, et ses différentes valeurs sont à leur tour emmagasinées, quand cela est nécessaire, dans une pile PNIM.

Enfin, un compteur CIM est mis à jour chaque fois qu'un indicateur de BIM est employé pour la première fois, c'est à dire quand le nombre total de mémoires de manoeuvre nécessaires à P_M augmente de un. Sa valeur initiale étant zéro, celle qu'il prend à la fin de PC_2 permet de générer la réservation du nombre exact de mémoires nécessaires à P_M . Cette valeur ne peut dépasser l'encombrement de BIM. C'est donc celui-ci qui limite le nombre maximum de mémoires de manoeuvre utilisables dans un programme compilé (Cf. Appendice II).

Deux autres compteurs CIM1 et CIM2, et une pile PCIM2 sont encore définis pour emmagasiner des valeurs intermédiaires de NIM en certains points de compilation (procédures, aiguillages, paramètres effectifs).

Nous en verrons le fonctionnement plus loin.

Enfin, les mémoires de manoeuvre devant servir d'opérandes à des instructions générées, il est nécessaire, pour que PC_2 puisse construire le champ variable M - m de ces instructions, qu'un profil qualitatif soit défini pour chacune de ces mémoires.

Ce profil est le plus souvent emmagasiné dans la pile d'adresses E3.

Il est formé de la même façon que celui des variables simples, la numérotation Nm (m) remplaçant la numérotation NQ (CQ).

La position deux dite "de mémoire de manoeuvre" y est obligatoirement mise à un (Cf. Appendice I4).

La position quinze y est également mise à un si la mémoire M - m contient non pas un résultat, mais l'adresse calculée d'une variable indicée.

Cette position sera utilisée pour ajouter aux codes opérations des instructions générées la marque d'adressage indirect.

B4 - INDICATEURS DE REGISTRES

Les machines pour lesquelles a été écrit le compilateur comportent deux registres principaux AC et MQ dans lesquels se font les opérations arithmétiques et logiques, et à travers lesquels passe toute information susceptible d'être transférée d'un mot à un autre.

Par exemple, pour additionner à un premier résultat, rangé en M - 1, la quantité contenue en Q - 1 et renvoyer le résultat final en M - 1 on peut commander :

Appel	C (M - 1) en AC
Addition	C (AC) + C (Q - 1), résultat en AC
Renvoi	C (AC) en M - 1

Si l'opération qui suit consiste alors à affecter ce résultat à la variable Q - 2, on pourra commander :

Appel	C (M - 1) en AC
Renvoi	C (AC) en Q - 2

On voit sur cet exemple simple, que nous effectuons ainsi deux opérations inutiles de transfert d'information entre le registre AC et M - 1.

De plus, si le premier résultat est lui-même en registre AC, ce qui est le cas le plus courant, nous pouvons nous

passer totalement de la mémoire M - 1, en commandant seulement :

Addition C (AC) + C (Q - 1), résultat en AC

Renvoi C (AC) en Q - 2

En gérant de la même manière le registre MQ, on voit que l'on peut espérer générer le minimum d'instructions de transfert d'information et utiliser le minimum de mémoires de manoeuvre (pratiquement le nombre qu'utiliserait le programme idéal P). La condition d'encombrement minimum de P_M est ainsi respectée, celle d'efficacité également lorsque l'on reste dans des cas simples d'expressions arithmétiques.

Pour réaliser ce fonctionnement, on définit deux indicateurs IAC et IMQ, respectivement pour les registres AC et MQ.

Ces indicateurs sont formés d'un mot mémoire chacun et fournissent à tout moment de la compilation l'état des registres AC et MQ, comme si le programme P_M se déroulait effectivement.

Ils peuvent prendre les valeurs suivantes :

- Zéro, si le registre correspondant est nul
- Un, si le contenu du registre est inconnu (Ex : C(IMQ) = 1 après que l'on ait généré une instruction FAD),

  ou inutilisable dans les instructions qui suivent (Ex : C(IAC) = C(IMQ) = 1 après un étiquetage)
- Profil qualitatif complet d'une variable si celle-ci vient d'être appelée dans le registre ou si on vient de lui affecter la valeur contenue dans ce registre.
- Profil qualitatif d'une mémoire de manoeuvre sans partie adresse si le registre contient un résultat intermédiaire non encore exploité.

Un numéro de mémoire m ne sera affecté à ce profil que si le résultat correspondant doit effectivement être rangé en M - m.

Enfin, les indicateurs IAC et IMQ ont eux-mêmes un profil qualitatif qui est placé en E3 pour signaler la présence d'un opérande en AC ou MQ. Ces profils sont zéro pour IAC et un pour IMQ (Cf. Appendice I4).

B5 - MEMOIRE DYNAMIQUE - MEMOIRES SPECIFIQUES DE BLOCS

Le programme P_M ayant été compilé, il doit pouvoir disposer pendant son exécution d'un certain nombre de mots mémoires qui contiendront les variables indicées du programme P_M d'une part, les valeurs sauvegardées lors d'un appel récursif de procédure d'autre part.

Ce bloc de mémoires est essentiellement dynamique et PC_2 ne tient pas compte de son encombrement maximum qui n'est connu qu'après assemblage de P_M. Cependant, afin de rendre minimum le nombre de mémoires nécessaires au moment de l'exécution, ce bloc est encore géré comme une pile de mémoires, appelée mémoire dynamique.

Elle est formée, au moment de l'exécution, de la partie de la mémoires rapide laissée libre par le système, P_M et les sous-programmes standard.

En fait, cette pile est double.

Elle comporte en effet, à son sommet, une partie DYNR réservée aux variables indicées de P_M déclarées de type rémanent et à sa base, une partie DYNL réservée aux variables indicées de type local et aux mémoires de récursivité.

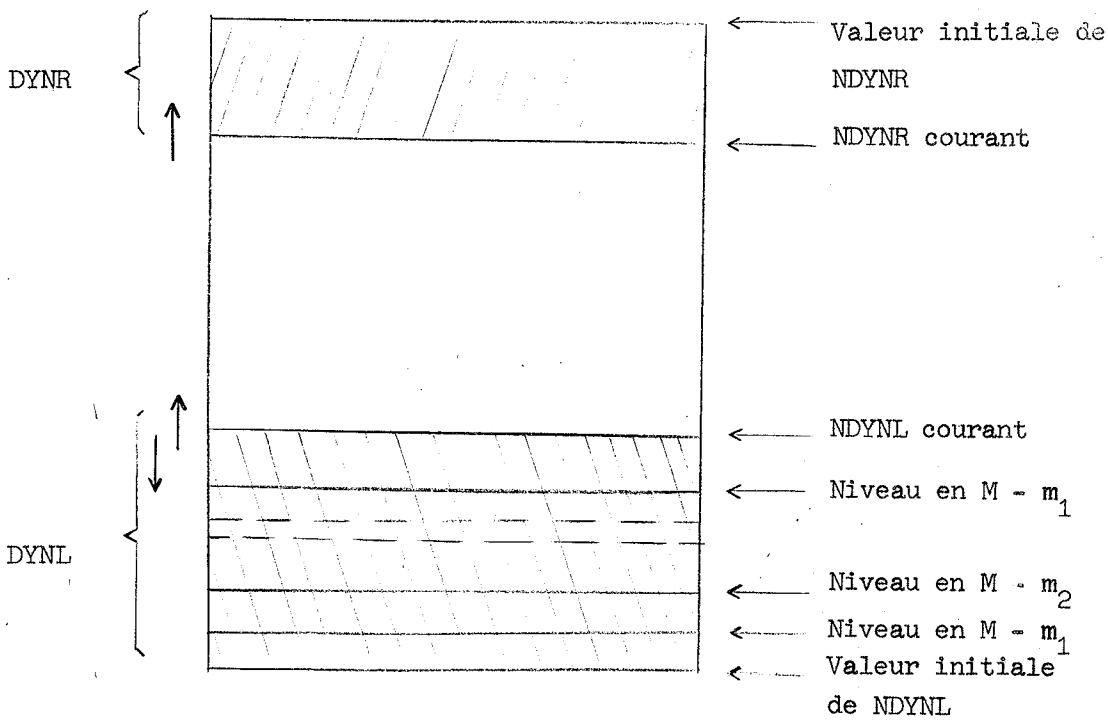
La partie supérieure est limitée par la valeur du niveau NDYNR, qui ne peut que décroître. En effet, par définition, les valeurs de tous les tableaux rémanents de P_S doivent être conservées pendant toute l'exécution de P_M. De plus, le compilateur considère les bornes de ces tableaux comme non dynamiques, c'est à dire ne don-

nant pas lieu à compression ou extension (Cf. Appendice II).

Le fonctionnement de la partie DYNR est donc celui d'une pile dont l'encombrement ne peut qu'augmenter, au fur et à mesure de l'exécution des déclarations de tableaux rémanents dans P_S .

La partie DYNL, au contraire, fonctionne comme une véritable pile. Elle est limitée à tout moment par le compteur NDYNL et possède autant de niveaux qu'il y a de blocs renfermant le bloc en cours d'exécution. Les valeurs de ces niveaux sont contenues dans les mémoires de manoeuvre ordinaires du programme P_M définies pour chaque bloc de P_S et dites mémoires spécifiques de blocs.

On verra que, dans ce cas, la notion de bloc s'étend d'une part aux corps de procédures, d'autre part aux paramètres effectifs et aux parties de déclarations d'aiguillages renfermant au moins un appel de procédure, d'aiguillage ou de paramètre formel.



C'est le programme PC_2 qui génère les mises à jour et les sauvegardes nécessaires du niveau NDYNL et des mémoires spécifiques de blocs.

C'est donc au moment de la compilation qu'est en fait construit l'algorithme d'optimisation de la mémoire dynamique propre à chaque programme P_S . (Alors que celui relatif aux mémoires de manoeuvre ordinaires est défini une fois pour toutes).

De même, c'est seulement à l'exécution de P_M que l'on peut vérifier que l'encombrement maximum de la mémoire dynamique n'est pas dépassé.

Il faut pour cela qu'on ait toujours NDYNL < NDYNR. On verra que cette vérification est effectuée par des sous-programmes standard chaque fois que l'on fait décroître NDYNR (déclaration de tableau rémanent) ou croître NDYNL (déclaration de tableaux locaux et entrée en récursivité).

C - PARTIE ANALYTIQUE DE PC_2

TABLES DE HIERARCHIES

C1 - SEPARATION DES SEQUENCES D'ANALYSE ET DE GENERATION

L'organigramme de la planche 7 relatif aux expressions arithmétiques simples s'étend facilement, comme on le verra, à la presque totalité des structures syntaxiques d'ALGOL [21, 23] .

On voit facilement alors que les fonctions de PC_2 se divisent en deux grands groupes :

D'une part la reconnaissance, l'analyse, et l'emmagasinage éventuel des éléments de la chaîne codée C, avec comme fonction corrélative la gestion des diverses piles de mémoires ; nous appellerons ce groupe partie analytique.

D'autre part, la génération effective des instructions du programme P_M , de ses réservations de mémoires et de son algorithme de gestion de la mémoire dynamique, que nous appellerons partie générative.

Dans le compilateur en fonctionnement, ces deux parties sont intimement liées, la partie générative se trouvant fortement disséminée dans la partie analytique.

Ainsi, si PC_2 se réduisait à l'organigramme de la planche 7, la partie générative serait formée du seul bloc X , entièrement intégré dans la partie analytique.

Cependant, pour des raisons de programmation et de mise au point évidentes, il nous a été très utile de séparer nettement, dans l'écriture de PC_2 , les séquences analytiques des sé-

quences génératives.

Nous respecterons cette distinction dans les chapitres qui suivent, en nous permettant d'insister sur son caractère purement artificiel.

C2 - PHASE DE MISE EN NOTATION POSTFIXEE

La partie analytique de PC_2 étant séparée de sa partie générative, mais devant constituer un programme complet, il a fallu pour l'écrire, simuler l existence des blocs tels que X de la planche 7.

Pratiquement, ceci est réalisé en supprimant à E3 son caractère de pile. Le bloc X est alors remplacé par le bloc Y (planche 8). On voit que chaque opérateur est simplement placé en E3 à la suite des deux opérandes qui lui sont associés.

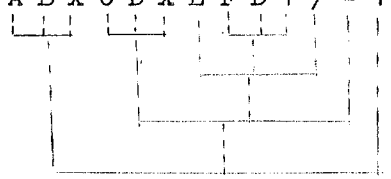
L'index k de la pile E3 ne décroissant jamais, le résultat d'une compilation est alors une chaîne d'opérandes et d'opérateurs associés, dont la constitution ne dépend que des hiérarchies respectives données aux opérateurs.

Ainsi, avec les hiérarchies déjà indiquées (Cf. B2), la même expression :

A x B + (C x D - E / (F + D))

fournit en E3 la chaîne suivante :

A B x C D x E F D + / - +



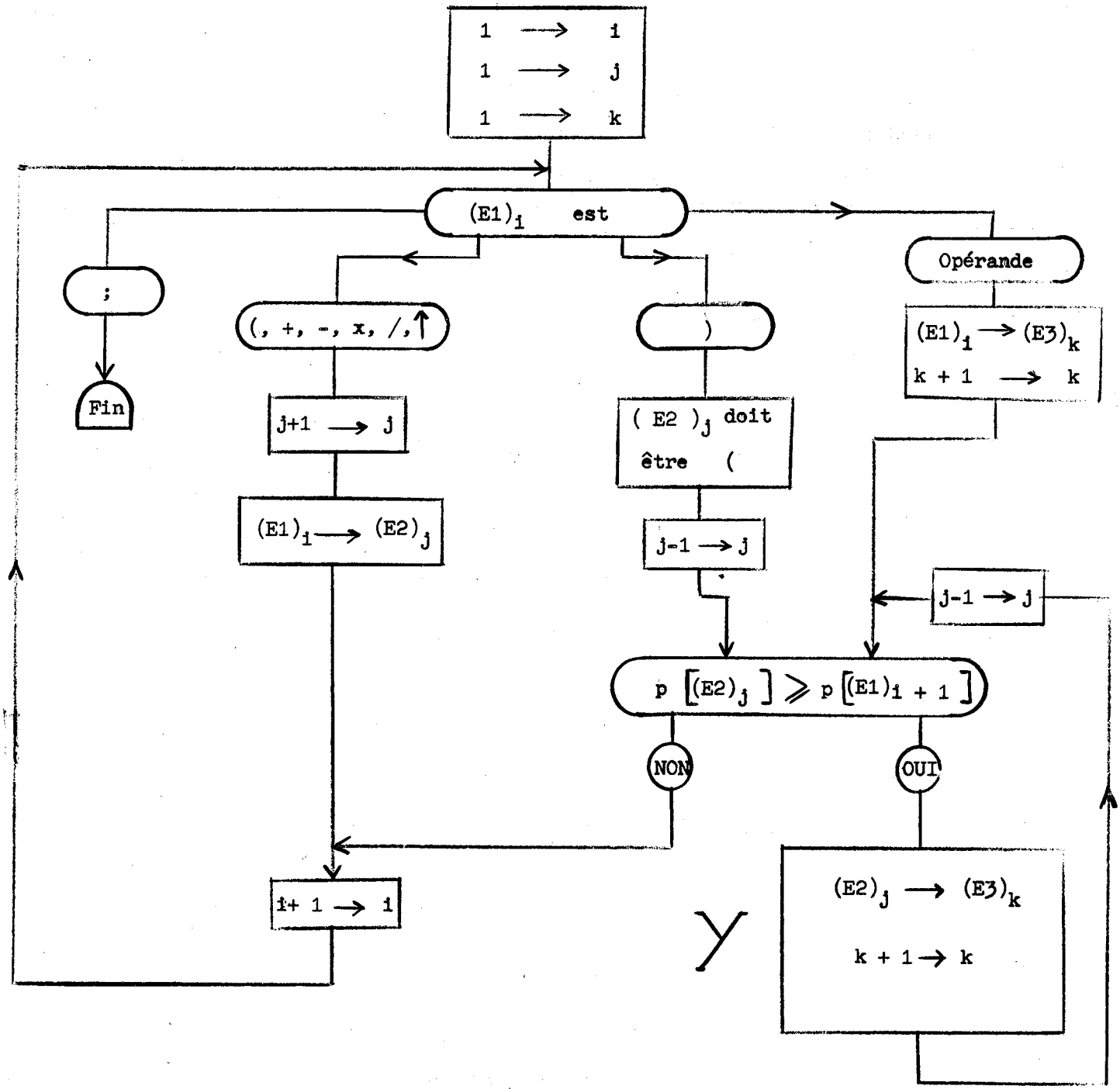


Planche 8

PARTIE ANALYTIQUE DE L'ORGANIGRAMME DE LA PLANCHE 7

Cette chaîne est dite "production en notation postfixée" de l'expression. Si les hiérarchies adoptées sont bonnes, elle fait apparaître clairement, après suppression de toutes les structures parenthésées, l'ordre d'exécution des opérations de l'expression considérée.

L'expérience montre que presque toutes les opérations du langage ALGOL peuvent donner lieu à une telle analyse.

Nous appellerons cette analyse, étendue aux programmes P_S entiers, phase de production en notation postfixée pour le langage L_S .

Elle exprime la généralisation à tous les opérateurs du langage du processus simple appliqué initialement aux expressions arithmétiques.

C3 - TABLE DE HIERARCHIES

La table de hiérarchies définitivement adoptée pour PC_2 est donnée en Appendice (I5).

Elle a été construite expérimentalement à l'aide des définitions du langage L_S [6] d'une part et des résultats préliminaires donnés par la phase de production en notation postfixée d'autre part.

On y trouve tous les délimiteurs du langage susceptibles d'entrer dans une instruction ou une expression d'un programme P_S , ce qui exclut les déclarateurs, les spécifieurs, et les crochets de chaîne et de code.

On y trouve également des séparateurs supplémentaires tels que $\downarrow$, $\downarrow^{*}$ .... , tous de hiérarchie zéro, destinés à jalonner, dans la chaîne en notation postfixée, certaines zones d'expression ou d'instructions.

Le fonctionnement détaillé de la phase de production en notation postfixée est exposé dans le chapitre II de cette partie.

D - PARTIE GENERATIVE DE PC_2

D1 - INTERPENETRATION DES PARTIES GENERATIVE ET ANALYTIQUE

Comme nous l'avons déjà souligné, la séparation que nous conservons, dans l'exposé du fonctionnement de PC_2 , entre ses parties analytique et générative, n'est qu'un artifice de programmation qui nous a permis de construire le plus rapidement possible une table de hiérarchies satisfaisante, tout en mettant au point par ailleurs des séquences de génération relatives à différents points du programme.

Cette différenciation a été - et reste - très utile pour modifier à volonté les séquences d'instructions générées pour chaque opération du langage L_S .

En fait, la compilation d'un programme P_S est, avec la méthode adoptée et après édition du programme, un problème purement séquentiel, qui n'exige qu'un seul balayage de P_S , et où les séquences analytiques et génératives peuvent être activées sans solution de continuité, dans un ordre qui ne dépend que de la forme syntaxique de P_S .

D2 - POINTS DE GENERATION

En reprenant l'organigramme de la planche 8, on voit qu'il suffit, pour en faire un organigramme complet de compilation, de remplacer son bloc Y, où un opérateur est transféré de E2 en E3, par un bloc de génération portant sur cet opérateur et qui prend pour opérandes les quantités dont les adresses sont dans les deux dernières mémoires occupées de E3.

Rappelons que l'organigramme de la planche 7 a été écrit directement sous cette forme.

Ce remplacement est tout à fait général. Etant don-

né le programme complet de production en notation postfixée, on peut dire que chaque bloc où un opérateur est transféré d'une pile quelconque dans E3 est susceptible d'être remplacé par un "point de génération" portant sur cet opérateur.

On définit ainsi un certain nombre de phases de génération indépendantes entre elles, dont on trouvera la liste et le fonctionnement au chapitre IV.

Actuellement au nombre de quinze, P1 à P15, ces phases peuvent elles-mêmes se scinder en sous-phases et en sous-phases secondaires quand l'identité de l'opérateur générateur, ou sa valeur syntactico-sémantique exacte, n'est pas déterminée par le seul emplacement du point de génération dans l'organigramme.

Dans tous les cas, le niveau le plus bas dans les sous-phases de génération est assez indépendant pour être modifié ou complété à volonté.

D3 - AUTRES PHASES DE GENERATION

Les points de génération ne sont pas les seuls endroits du programme où des instructions du langage L_M sont construites.

En particulier, les détections de certains opérandes peuvent activer des phases de génération particulières, toujours sous la conduite du programme de production en notation postfixée.

C'est le cas pour les paramètres formels appelés par nom, les identificateurs d'aiguillage et de procédure en déclaration, etc ...

Enfin, certains délimiteurs, sans fonction opératrice, peuvent tout de même donner lieu à une phase de génération, quelquefois très succincte. C'est le cas de DEBUT, des crochets de chaîne et de code, etc ...

Ces phases particulières sont numérotées dans la suite de P16 à P24 et sont également exposées en détail dans le chapitre IV de cette partie.

D4 - SOUS-PROGRAMMES DE GENERATION

Enfin, la partie générative de PC_2 comporte un grand nombre de séquences communes à plusieurs, ou à la totalité, des phases et points de génération.

Ces séquences ont été groupées sous le titre de "sous-programmes" et sont étudiées dans le chapitre V .

En fait, elles se divisent en trois parties :

1°) - Il existe de véritables phases ou sous-phases de génération susceptibles d'être activées par plusieurs branches du programme de production en notation postfixée.
Ces séquences ne doivent qu'à cette particularité d' être placées dans l'ensemble des sous-programmes.
Dans la suite, ces séquences sont numérotées de P25 à P29 pour les phases et de S30 à S41 pour les sous-phases.

2°) - D'autres séquences sont, elles, de véritables sous-programmes fermés appelés, pour certains, par la totalité des points et phases de génération. Nous les appellerons "programmes de servitude" de la partie générative.
Elles assurent la génération des garnissages et rangements des registres AC et MQ, des conversions de type d'opérandes, des sauvegardes et mises à jour du registre index X4 et du niveau NDYNL, etc ...

3°) - Enfin, des sous-programmes que nous appellerons "programmes d'édition" permettent à la partie générative de construire caractère par caractère, et d'imprimer sur la bande magnétique BG, les instructions commandées par les phases de génération.
L'écriture de ces sous-programmes est liée à la configuration de la machine utilisée et au système d'assemblage employé.

D5 - PHASES COMPLEMENTAIRES DE GENERATION

Il existe, en plus des phases et sous-programmes de génération, deux phases complémentaires, situées en début et en fin de PC_2 .

Elles assurent la liaison, la première avec le passage éditeur PC_1 , la seconde avec l'assembleur du système.

Nous les étudierons respectivement dans les chapitres III et VI de cette partie.

E - ALGORITHME GENERAL

E1 - VUE GENERALE

Le schéma présenté sur la planche 9 traduit dans son ensemble l'algorithme général du passage compilateur (P 113).

La dissémination des points et phases de générations dans le programme y est particulièrement visible.

Les divisions et branches principales y sont marquées d'un trait plus fort, et il est facile en particulier d'isoler dans ce schéma l'algorithme de compilation des expressions arithmétiques simples tel qu'il avait été donné en planche 7.

On notera seulement que le bloc X de ce dernier est remplacé par un bloc H beaucoup plus général puisqu'il donne naissance au point deux de génération qui traite pratiquement tous les opérateurs du langage.

E2 - ANALYSE DES DIVERSES BRANCHES

La fonction de chaque branche de l'algorithme est clairement indiquée sur le dessin.

On remarque en particulier la structure très séquentielle du programme (retour presque systématique en E pour chaque unité syntaxique).

Rappelons que la fin des unités syntaxiques de la chaîne codée est détectée en E par la présence d'une mémoire nulle dans cette chaîne. Il y a alors activation de la phase de fin de génération.

Sinon, le traitement se divise en trois branches principales qui, toutes, peuvent donner lieu à des points de génération (P1 à P15), à des phases particulières (P16 à P24) ou à des phases communes (P25 à P29).

E2 - 1 Branche Opérandes (OPERA)

Cette branche assure le traitement de tous identificateur, nombre pur ou valeur logique apparaissant dans une instruction ou une expression.

La plupart de ces opérandes sont directement traités par le bloc H et le point de génération deux.

Notons cependant le cas particulier des identificateurs d'appel de procédure avec paramètres, dont le traitement ne revient en H qu'après épuisement de la liste de paramètres effectifs, et des listes de parties gauches qui ne sont traitées qu'après qu'on ait trouvé le dernier symbole := de la liste.

Enfin, les étiquettes en déclaration sont traitées immédiatement (point trois) et le symbole : qui les suit n'est pas analysé.

E2 - 2 Branche déclarateurs (DECLA)

Les identificateurs en déclaration sont traités immédiatement dans le plus grand nombre de cas.

Si la déclaration est simple, ce traitement ne comporte aucune phase de génération.

Si la déclaration est d'aiguillage ou de procédure, on déroule plusieurs phases de génération qui ont pour rôle de placer en tête des instructions générées par le corps de la déclaration elle-même, des instructions de transfert et d'initialisation.

Dans le cas d'une procédure, il peut même y avoir déroulement du véritable point de génération quatre associé au spécifieur VALEUR.

Enfin, si cette procédure est en code machine, le corps est traité en entier et l'analyse ne reprend qu'avec le ; de fin de procédure.

Seules les déclarations de tableau ne donnent lieu à aucune génération immédiate.

Celle-ci est reportée au traitement du symbole [, afin de

tenir compte de tous les identificateurs susceptibles d'être placés entre TABLEAU et [(section de tableau).

Dans tous les cas, on peut donc dire que l'analyse ne reprend en E qu'avec des symboles faisant partie d'instructions ou d'expressions.

E2 - 3 Branche Opérateurs (SOPER)

Cette branche assure l'analyse et le traitement éventuel de tous les autres éléments du langage.

La plupart des symboles ne subissent aucune phase de génération et l'analyse continue en E.

D'autres marquent un début, une séparation, ou une fin de liste. Il y a alors en général une phase de génération particulière. Nous prenons ici liste dans un sens large. Ce peut être par exemple une liste d'instructions (DEBUT, ;),de paramètres effectifs ((, '(' , virgule) ou de POUR (POUR, virgule, FAIRE).

D'autres sont reconnus comme marquant une fin de déclaration ( ] , ;) .

Remarquons que, dans le cas de] marquant une fin de déclaration de tableau, il y a retour direct à la branche DECLA si une nouvelle section de tableau suit] .

Enfin, certains symboles marquent la fin d'opérandes. C'est le cas pour) (fin de primaire arithmétique ou d'appel de procédure),] (fin de variable indicée ou d'aiguillage), et FIN (fin d'instruction). Le retour se fait alors en G ou H suivant que l'opérande est susceptible d'appartenir à une partie gauche ou non.

F - ORGANISATION DE PC_2

F1 - CHARGEMENT DE PC_2 EN MEMOIRE

Actuellement, PC_2 constitue un programme d'environ sept mille instructions, toutes chargées en même temps dans la partie basse de la mémoire rapide, à la suite du système.

Ce programme est organisé de la façon suivante :

En tête est écrite la phase complémentaire de début de génération PDG.

Puis vient la partie analytique du programme, que nous nommerons PC_2 - A .

Dans cette partie sont insérées les phases particulières de génération P16 à P24 dont nous avons déjà parlé.

Nous avons regroupé à la suite de PC_2 - A le reste de la partie générative, PC_2 - G , qui comprend les points de génération (P1 àP15), les phases et sous-phases organisées en sous-programmes (P25 à P29 et S30 à S41), les sous-programmes de servitude (SP1 à SP13) et les sous-programmes d'édition (SP14 à SP16).

Enfin viennent la phase complémentaire de fin de génération PFG, les sous-programmes d'erreur du compilateur (ER i) et les constantes, mémoires de travail, et tables utilisées par PC_2 entier.

F2 - PILES DE MEMOIRES

La chaîne codée C et la table de nombres purs TNP sont en partie haute de la mémoire rapide, telles que les a fournies le passage PC_1 .

L'encombrement de cette partie est le même que dans PC_1.

Au-dessous de TNP, PC_2 définit un certain nombre de piles de mémoires, puis le bloc BIM, et enfin la zone intermédiaire de génération Z_G .

Les piles sont actuellement au nombre de dix-sept. Ce sont :

PR, pile d'opérateurs et PN, pile d'opérandes, qui jouent respectivement les rôles de E2 et E3 dans l'organigramme 7 .

PPTA où sont mis en réserve les identificateurs de procédures, tableaux, et aiguillages.

PE où sont emmagasinés les profils qualitatifs des étiquettes internes à P_M créées par PC_2 .

PNIM et PCIM2 , que nous avons déjà mentionnées.

PB et PMB contenant respectivement les caractéristiques des niveaux de la mémoire dynamique et les profils qualitatifs des mémoires de manoeuvre correspondant à ces niveaux (mémoires spécifiques de blocs).

PPF où sont mises en réserve les valeurs d'un compteur de paramètres formels.

PCPED, PPED, PEPE où sont emmagasinés respectivement les valeurs d'un compteur de paramètres effectifs de procédures déclarées, les profils qualitatifs de ces paramètres et les profils qualitatifs de leurs étiquettes d'appel.

PCPES, PPES, PEST où sont emmagasinés respectivement les valeurs d'un compteur de paramètres effectifs de fonctions standard (ou d'indices de variables indicées), les profils qualitatifs de ces paramètres, et ceux des étiquettes d'appel de ces fonctions.

PEA où sont mis en réserve les profils qualitatifs des différentes expressions de listes d'aiguillages,

et enfin PQTM où sont mises en réserves les valeurs de compteurs de quantités, de tableaux et de mémoires de manoeuvre relatifs à une même procédure (ces valeurs sont utilisées seulement pour permettre les appels récursifs de procédure).

Dans la suite, nous nommerons l'index de chaque pile par le nom de la pile en remplaçant la première lettre P par L .

Ces index indiquent toujours la dernière mémoire occupée de la pile correspondante.

La chaîne codée C possède également un index LC indiquant toujours l'unité syntaxique en train d'être analysée.

Enfin, la zone Z_G possède un index LZ_G qui est remis à un

quand on transfère Z_G dans les zones d'impression. IZ_G indique à tout moment la première ligne libre de Z_G , chaque ligne ayant la longueur maximum d'une instruction générée (actuellement cinq mémoires). Les mémoires non employées de Z_G sont remplies de caractères "espace".

F3 - CONDITIONS D'ENCOMBREMENT

Cette organisation de PC_2 entraine l'existence d'un certain nombre de conditions d'encombrement, portant sur la dimension des piles et autres blocs de mémoires.

Ces conditions ne sont pas indépendantes entre elles et sont difficiles à exprimer en fonction des programmes P_S à compiler.

Dès que l'une d'elles n'est plus respectée, PC_2 donne le contrôle à un programme d'erreur qui arrête le traitement et imprime un libellé approprié.

Ces conditions portant essentiellement sur des paramètres d'assemblage du compilateur modifiables à volonté, dans la limite de la mémoire rapide disponible, nous ne les considérons pas comme des restrictions à la généralité du langage L_S .

C'est pourquoi elles n'apparaissent pas dans l'appendice II où ces restrictions sont listées.

Nous nous permettons d'insister sur la différence essentielle existant entre ces conditions et d'autres que nous verrons plus loin ou que nous avons déjà vues dans la première partie (nombre de déclarations de procédures limité à 256, identificateurs reconnus sur douze caractères, etc ...) : alors que celles-ci sont inhérentes au fonctionnement même du compilateur et exigeraient, pour être modifiées, de réécrire tout ou partie de PC_1 ou PC_2 , celles-là sont modifiables à volonté par chaque installation utilisant le compilateur.

CHAPITRE II

PHASE DE PRODUCTION

EN NOTATION POSTFIXEE

A - RAPPEL DES FONCTIONS

A1 - FONCTION PRINCIPALE

Nous rappelons que la partie analytique PC_2 - A ou phase de production en notation postfixée, a pour rôle principal de transformer la chaîne codée C fournie par PC_1 en une chaîne C' dite "en notation postfixée" où chaque opérateur ou pseudo-opérateur suit immédiatement les opérandes qui lui sont associés, l'ensemble <opérandes> <opérateur> pouvant à nouveau servir d'opérande.

Les notions d'opérateur et d'opérandes sont prises ici dans un sens très large.

Ainsi, étant donnée l'instruction suivante :

E : SI <expression booléenne B> ALORS <instruction I1> SINON <instruction I2> ;

Nous distinguerons les opérateurs et opérandes suivants :

Opérateurs	Opérandes associés
:	E
ALORS	B
SI	B et I1
SINON	B et I2
;	aucun

On voit en particulier dans ce cas qu'un opérande peut être constitué d'éléments très variés du langage : etiquette, expression, instruction.

Le même opérateur n'est pas toujours associé aux mêmes types et au même nombre d'opérandes.

Ainsi, dans l'instruction suivante :

X := SI B ALORS < expression E1 > SINON < expression E2 > ;

nous avons

Opérateurs	Opérandes
ALORS	B
SI	B et E1
SINON	B et E2
:=	X et <SI B ALORS E1 SINON E2>
;	aucun

C'est un cas où SI ou SINON n'ont pas les mêmes types d'opérandes que précédemment (expressions au lieu d'instructions).

Prenons maintenant le cas d'un appel de variable indicée :

X := T [A + B, C] ;

Nous sommes obligés de faire intervenir un opérateur artificiel du type index (↓), qui prend en charge la liste d'indices, et nous aurons :

Opérateurs	Opérandes
+	A et B
↓	T, < A + B> et C
:=	X et <T [A + B, C]>
;	aucun

Il est visible que l'opérateur ↓ n'aura alors pas toujours le mêmes nombre d'opérandes.

Il en sera de même dans un appel de procédures, dans une liste de POUR, etc ...

Ces considérations montrent que la conduite à tenir au moment où un opérateur est sur le point d'entraîner l'activation d'une phase de génération, est loin d'être déterminée par la seule identité de cet opérateur.

Nous verrons que ces problèmes sont résolus d'une part grâce aux opérateurs artificiels du type index introduits par PC_2 - A d'autre part par la présence dans PC_2 - G de parties analytiques permettant l'activation d'une sous-phase ou d'une autre à l'intérieur d'un même point ou d'une même phase de génération.

Ces solutions sont essentiellement empiriques, et ont été bâties au fur et à mesure des résultats obtenus par PC_2 - A. Elles ont été modifiées plusieurs fois au cours de l'écriture de PC_2 - G , ce qui nous a obligé en fait à mettre au point plusieurs versions du programme de mise en notation postfixée.

Nous n'exposerons ici que la dernière version de ce programme, qui n'est définitive que dans la mesure où la partie générative l'est également.

A2 - PRINCIPE DE FONCTIONNEMENT - ROLES ANNEXES -

Le principe de fonctionnement de la phase PC_2 - A est celui, généralisé à tous les opérateurs du langage L_S , de l'organigramme de la planche 8.

La pile PR sert de pile d'opérateurs (E2) et la pile PN contient le résultat C' du passage. PN fonctionne en fait comme un bloc de mémoires dont l'index varie dans un seul sens.

Insistons encore sur le fait que cette matérialisation temporaire de la chaîne C' n'est pas effective dans le compilateur complet, où aucun opérateur n'est susceptible d'apparaître dans la pile PN.

Enfin, rappelons que la partie PC_2 - A a pour fonction annexe de gérer les niveaux de la pile de mémoires de manoeuvre et de la mémoire dynamique, ainsi que la pile de récursivité PQTM. Ces mémoires sont ensuite exploitées par la partie générative PC_2 - G .

PC_2 - A partage également avec PC_2 - G la détection du plus grand nombre possible d'erreurs syntactico-sémantiques dans le programme P_S , celui-ci étant supposé épuré, sous sa forme C, de toutes ses erreurs morphologiques ou d'écriture.

B - FONCTIONNEMENT GENERAL

B1 - ORGANIGRAMME

La planche 10 représente l'organigramme général de la phase de mise en notation postfixée (P 169).

On y retrouve facilement les branches principales de l'algorithme de la planche 9, augmentées d'un petit nombre de branches complémentaires relatives à des opérateurs susceptibles d'un traitement très court et se ramenant toujours à une branche principale.

Nous avons indiqué sur l'organigramme, par tracé double, tous les points du programme susceptibles d'être remplacés par une phase ou un point de génération, dont le numéro a été placé à coté.

B2 - ANALYSE DE L'UNITE SYNTAXIQUE EN COURS : E

Le point E , au centre de l'organigramme général marque l'endroit où une unité syntaxique nouvelle est prise en C pour être analysée.

Il donne le contrôle aux branches opérandes, déclarateurs ou opérateurs suivant les valeurs des positions correspondantes dans le profil qualitatif.

Le contrôle est donné la première fois à E par le début du programme. Ensuite, E obtient le contrôle lorsque l'analyse d'une unité ou d'un groupe d'unités syntaxiques est finie, ou encore lorsque cette analyse ne peut être terminée que lorsque celle d'une ou plusieurs unités qui suivent l'est également.

C - BRANCHE OPERANDES

C'est la branche droite de l'organigramme.

Elle se charge de l'analyse de tous les opérandes du programme P_S , à l'exception des occurrences de paramètres formels apparaissant dans les en-têtes de procédures, des variables simples et des aiguillages en train d'être déclarés.

Ces opérandes peuvent être de trois sortes :

1°) Identificateurs

2°) Nombres purs

3°) Valeurs logiques

Dans la presque totalité des cas, le traitement se termine en G et H .

En plus, le contrôle peut être donné à ces points par la branche opérateurs en détectant la fin d'un opérande composé.

C1 - PRELIMINAIRES

Dans tous les cas, la branche opérandes procède au rangement de l'opérande en PN, où il constituera un élément de C' , après incrémentation de l'index LN de PN.

Ce rangement a été séparé en deux cas sur la planche 10, suivant que l'opérande est un paramètre formel appelé par nom ou est un opérande ordinaire. Cette distinction n'est pas effective dans PC_2 - A . Elle est là pour rappeler l'insertion, dans le programme complet, de la phase P16 de génération (opérateur virtuel $\downarrow^9$).

C2 - VALEURS LOGIQUES

Si l'opérande est une valeur logique, le traitement se poursuit directement en H, cet opérande ne pouvant pas être l'élément d'une partie gauche (GA) ou d'une déclaration d'étiquette (GC). On avance d'une unité dans C pour être prêt au test de hiérarchies.

C3 - IDENTIFICATEURS ORDINAIRES

Si l'opérande est un identificateur qui n'est pas de procédure ou de tableau, son traitement se continue immédiatement en G après avancement d'une unité dans C.

C4 - IDENTIFICATEURS DE TABLEAUX

On procède là à une vérification : si on n'est pas dans l'état déclaration, l'identificateur doit être suivi de [. Le retour se fait directement en E pour traiter [. (Le test de hiérarchie est en effet inutile, [ayant la plus forte hiérarchie, et ne pouvant apparaître en PR).

C5 - IDENTIFICATEURS DE PROCEDURES

Si l'opérande est un identificateur de procédure, trois cas sont possibles :

1°) Il peut s'agir de l'affectation d'un indicateur de fonction. Il est alors suivi immédiatement de := . L'affectation pouvant être multiple, il y a retour en GB (liste de parties gauches).

2°) Il peut s'agir de l'appel d'une procédure avec paramètres.

L'analyse est alors seulement initialisée, puisqu'il s'agit d'un opérande composé qui ne pourra donner lieu au test de hiérarchies qu'après le traitement de) terminant la liste de paramètres effectifs.
En fait, le traitement de l'identificateur consiste uniquement à le transférer de la pile PN à la pile PPTA où il est mis en réserve pour le traitement des paramètres effectifs et de la fin de la liste.

Pratiquement, c'est une partie de la branche opérateurs qui se place ici, relative au symbole (. Quand la procédure considérée est standard, celui-ci a la même valeur syntaxique que [dont on verra le traitement plus loin. Ceci explique la présence du sous-programme B (Cf. E1), et des phases P25 et P26 relatives respectivement aux opérateurs virtuels $\downarrow^{7}$ (initialisation de l'appel) et $\downarrow^{7*}$ (début du premier paramètre effectif).
On note en plus la présence de la branche directe relative aux fonctions standard du type ouvert. Ces fonctions sont en effet compilées comme de véritables opérations, et ne donnent lieu à aucun autre traitement que la mise en réserve en PPTA de leur identificateur. Ainsi, l'expression

P (A + B)

où P(X) est une fonction du type ouvert, donnera la chaîne

A B + P ↓

et l'ensemble P ↓ peut être considéré comme un opérateur simple.
Enfin, le cas des véritables procédures déclarées entraîne, outre les phases P25 et P26, l'exécution du sous-programme C relatif à la mise à jour du niveau de la mémoire dynamique (Cf. E1) et le début de constitution du profil qualitatif de

la mémoire spécifique du "bloc" constitué par le premier paramètre effectif (Cf. F2). Ce profil est emmagasiné dans la pile PB, les appels de procédures pouvant s'imbriquer.
A cet endroit est également mis à jour le compteur CIM2 qui détermine les mémoires de manoeuvre utilisables pour les résultats des paramètres (Cf. F2).

3°) Le seul cas possible ensuite est l'appel d'une procédure sans paramètre.
On verra en effet que les autres occurrences possibles d'identificateurs de procédures (en-tête de procédure, liste de paramètres effectifs) sont traitées directement.
On simule alors la fin d'appel de procédure par l'index ↓ placé en PN.
L'ensemble de l'identificateur et de l'index forme un nouvel opérande. Ce traitement est à remplacer, comme il est indiqué, par le point un de génération.

Ainsi l'instruction suivante, où P et P' sont des identificateurs de procédure :

P := A + P'

sera transformée en :

P A P' ↓ + :=

C6 - NOMBRES PURS

Les nombres purs tels qu'ils sont fournis dans la chaîne codée C par PC_1 sont formés d'autant d'unités syntaxiques qu'ils contiennent d'éléments.

Ainsi le nombre

3 . 8

fournit trois unités syntaxiques en C.

Le maximum est obtenu quand le nombre comprend six unités, qui sont alors de la forme

$$E \cdot D_{10} \pm P$$

où E,D,P représentent respectivement une partie entière, une partie décimale et une partie exposant.

Ces unités possèdent chacune un profil qualitatif, qui, pour E, D, P est toujours celui d'un entier sans signe.

Le rôle de la branche opérande dans ce cas est de réduire ce groupe de profils qualitatifs à un seul, que l'on place en PN, à la place de celui de E, lui-même mis en PN au début du traitement comme un identificateur ordinaire.

Ce profil unique est doté du type du nombre pur ainsi calculé, suivant les conventions rappelées en annexe (Appendice II). En même temps on vérifie que la valeur du nombre pur ne dépasse pas la grandeur d'une mémoire (2^{35} pour un nombre entier, $\sim 10^{38}$ pour un nombre réel).

Notons que les nombres ainsi calculés sont rangés en TNP, à la suite des entiers sans signe construits par PC_1 , et que deux nombres identiques occupent la même mémoire de TNP.

En particulier, les entiers sans signe suivis d'aucune partie décimale ou exposant, gardent le même profil qualitatif et on ne définit pas pour eux de nouvelle mémoire dans TNP.

Les nombres réels, quelque soit l'origine de leur construction (E.D, ou $E_{10} \pm P$, etc ...), sont en mémoire des nombres du type binaire flottant [17, 26] , et leur valeur est calculée avec arrondi, en tenant compte du fait que les parties décimales sont cadrées par PC_1 à gauche des mémoires de TNP.

Notons que les nombres purs ne possédant pas de partie entière (formes .D, $.D_{10}$ P, $.D_{10} \pm$ P) sont également traités par cette branche de l'organigramme.

Cependant, leur première unité syntaxique étant un délimiteur, le début de leur traitement se trouve dans la branche opérateurs.

Les nombres réduits à une seule partie exposant ($_{10}$P et $_{10} \pm$ P) sont traités en entier dans la branche opérateur, par une séquence analogue.

Enfin, un nombre pur ne pouvant pas être l'élément d'une partie gauche ou d'une déclaration d'étiquette, le contrôle est donné

immédiatement à H, l'index LC de C étant tel que le test de hiérarchies se fasse sur l'unité syntaxique suivant la dernière du nombre pur.

Ainsi, soit l'expression :

$A + 2 . 3 \times 4_{10} + 2 \uparrow B - 400 . 0 \times 2$;

D'après nos conventions, les trois nombres composés apparaissant ici sont du type réel.

Supposons que PC_1 ait affecté à

2	le numéro	1	dans	TNP
0.3	le numéro	2	dans	TNP
4	le numéro	3	dans	TNP
400	le numéro	4	dans	TNP

zéro étant toujours affecté du N° 0, et qu'aucun autre entier n'apparaisse dans P_S .

Alors, si nous représentons par NE 1 le nombre entier numéro un, par ND une partie décimale, et NR un nombre réel, la chaîne codée correspondante a la forme suivante :

$A + NE\ 1 . ND2 \times NE\ 3_{10} + NE\ 1 \uparrow B - NE\ 4 . NE\ 0 \times NE\ 1$

La branche opérande calcule alors le nombre 2.3 et lui donne le numéro cinq, puis le nombre $4_{10} + 2$ et lui donne le numéro six, enfin le nombre 400.0, auquel elle donne également le numéro six. Elle laisse au nombre 2 son profil et son numéro.

On obtient la chaîne C' suivante :

A NR5 NR6 B ↑ x + NR6 NE1 x - ,

et l'unité syntaxique sur laquelle se fait le prochain test de hiérarchies est ; .

C7 - POINT G

Si l'opérande est un identificateur ordinaire, il convient de déterminer à cet endroit s'il s'agit d'une déclaration d'étiquette. Il faut pour cela avoir la disposition suivante :

Etiquette :

Dans ce cas, l'opérateur : est directement associé à son seul opérande Etiquette, de sorte que l'instruction

E : A := B ;

fournira la chaîne postfixée suivante :

E : A B :=

Quand on remplacera ce point par le point P3 de génération, l'étiquette se rapportera bien à la première instruction à générer, puisque celle-ci correspondra au prochain opérateur sur le point d'être placé en PN.

Le cas d'étiquetage multiple n'offre pas de difficultés :

E1 : E2 : E3 : A := B

donne : E1 : E2 : E3 : A B := ,

et toutes les étiquettes seront considérées comme portant sur le même opérateur := , donc équivalentes.

Notons que dans ce cas le retour est en E, pour analyser l'unité qui suit : .

C8 - POINTS GA ET GB

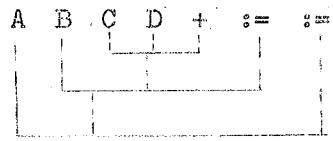
Si le cas n'est pas celui d'une déclaration d'étiquette, on doit regarder s'il s'agit d'une liste de parties gauches.

S'il en est ainsi (:= à la fois en C et en PR), il faut éviter le test de hiérarchies qui, portant sur deux opérateurs identiques, placerait celui de PR en PN, ce qui donnerait un résultat faux.

En effet, soit l'instruction

A := B := C + D ;

La notation postfixée correspondante est



et non pas

A B := C D + :=

qui ne laisse aucun opérande au deuxième :=

Le deuxième symbole := est donc placé en PR, on verra que c'est là le traitement de := quand il ne fait pas partie d'une liste de POUR, et cette éventualité est incompatible avec l'existence de plusieurs parties gauches.

Le contrôle est ensuite rendu à E pour analyser l'unité suivant := .

Notons que ce fonctionnement n'est pas perturbé par la présence dans les parties gauches de variables indicées.

Celles-ci sont en effet entièrement compilées à cet endroit et regardées à leur tour comme un seul opérande dont le traitement est renvoyé à GB par la branche opérateurs quand celle-ci détecte] .

C9 - POINT H

Dans la plupart des cas, la fin du traitement des opérandes, simples ou composés, donne lieu au test de hiérarchies H.

Si la hiérarchie du dernier symbole opérateur de PR est supérieure ou égale à celle du prochain opérateur de C, c'est que l'opérateur de PR a maintenant tous ses opérandes en PN, l'opération qui lui correspond peut être effectuée (générée).

C'est au point deux de génération, remplaçant le bloc correspodant (Cf. organigramme) que "descendent" ainsi de PR tous les opérateurs arithmétiques, logiques et de relation, ainsi que les symboles := , SI , SINON, ALLER A, POUR, PAS, JUSQUA, TANTQUE, FAIRE .

Le test de hiérarchies est recommencé avec tous les opérateurs de PR en faisant décroître LR, jusqu'à ce que le test ne soit plus satisfait.

Il est inutile de tester si l'on arrive à épuisement de PR car le premier symbole dans celle-ci est forcément DEBUT, de hiérarchie zéro, qui ne peut pas satisfaire au test, le symbole de C auquel on compare ne pouvant pas être DEBUT puisqu'il suit un opérande, et le seul symbole de hiérarchie zéro apparaissant dans C étant DEBUT.

Quand le test est insatisfait, le traitement retourne à E, pour analyser le symbole qui vient d'être l'objet de la comparaison.

Le fonctionnement du point H réalise à lui seul la compilation de la plusgrande partie des instructions des programmes P_S .

En particulier, il permet de déterminer la portée exacte de chaque opérateur.

Ainsi, dans l'expression

$$A + B \times C \uparrow D,$$

ce fonctionnement montre clairement que le symbole + porte sur l'opérande $B \times C \uparrow D$, etc ...

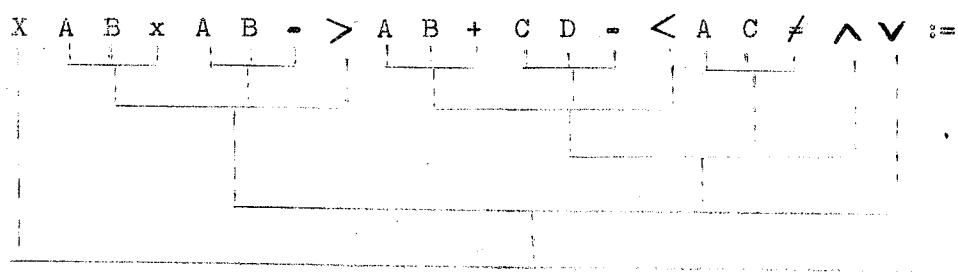
La chaîne postfixée obtenue dans ce cas est :

et le résultat est ainsi rendu exactement identique à celui d'une expression complètement parenthèsée.

De même, dans l'instruction plus compliquée qui suit :

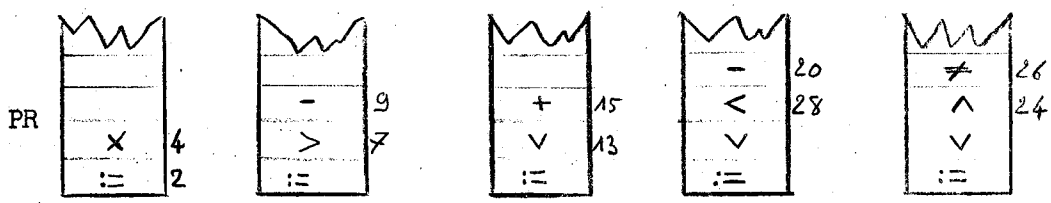
$$X := A \times B > A - B \vee A + B < C - D \wedge A \neq C \;;$$

où X est une variable booléenne, et où la notation postfixée obtenue entièrement au point H fixe entièrement l'ordre d'exécution des opérations et la portée de chaque opérateur, compte tenu de la table de hiérarchies adoptée :



En particulier si nous représentons les états successifs de l'ensemble des piles PN et PR d'un bout à l'autre de la compilation de l'instruction, le fonctionnement de la pile PR apparait clairement :

PN	X	A	B	x	A	B	-	>	A	B	+	C	D	-	<	A	C	≠	∧	∨	:=
	1	3	5	6	8	10	11	12	14	16	17	19	21	22	23	25	27	28	29	30	31



On dénombre ainsi trente et un états, qui n'utilisent que quatre mémoires de la pile PR .

Remarquons enfin que si, au point H , les symboles POUR ou FAIRE sont "descendus" de PR en PN , des séquences analytiques particulières sont déroulées, qui permettent d'éviter dans la branche opérateurs des recherches supplémentaires. Ces séquences sont normalement inclues dans PC_2 - G , aux sous-phases correspondantes.

Ainsi, à la suite de la génération pour POUR, un opérateur $\downarrow^5$ est placé en PR si le symbole qui suit l'opérande qui vient d'être analysé est PAS ou TANTQUE . Nous verrons le rôle de $\downarrow^5$ plus loin.

Dans le cas de FAIRE , on procède à une vérification qui permet de détecter l'existence d'une expression incomplète (mémoires de manoeuvre non libérées dans BIM) en fin de boucle.

D - BRANCHE DECLARATEURS

C'est la branche dessinée à gauche de l'organigramme général (planche 10).

Le contrôle lui est donné par E quand une unité syntaxique est détectée qui est un déclarateur.

Grâce aux vérifications syntaxiques effectuées par PC_1 et d'après le fonctionnement de cette branche, aucun spécifieur, ou déclarateur situé dans un en-tête de procédure, ne peut être analysé en E.

Il s'agit donc ici des véritables sortes de déclarations :

1°) Variables simples

2°) Tableaux

3°) Procédures

4°) Aiguillages

En plus, on traite le symbole PERSISTANT, qui se ramène obligatoirement à l'un des deux premiers de ces cas.

Enfin, la branche déclarateurs peut obtenir le contrôle de la branche opérateurs à la fin d'une section de tableau.

D1 - VARIABLES SIMPLES

E a détecté un des symboles REEL, ENTIER ou BOOLEEN. Si ce symbole n'est pas suivi immédiatement de TABLEAU ou PROCEDURE, il s'agit d'une déclaration de variables simples.

On ne peut alors avoir qu'une disposition du type suivant :

REEL ID1, ID2 , ... IDn - 1 , IDn ;

où ID1 ..... IDn sont des identificateurs.

On vérifie que cette disposition est respectée.

Puis, yant trouvé ; , on détermine si la déclaration est de type local ou rémanent.

Si REMANENT était apparu devant REEL, ENTIER ou BOOLEEN, un indicateur IPERS avait été mis à un. On le remet à zéro ici.

On a défini deux compteurs COQLR et COQPR destinés à décompter les variables simples locales ou rémanentes apparaissant dans une même procédure. Le compteur désiré est mis à jour ici par la valeur atteinte par la numérotation correspondante (située dans le profil qualitatif du dernier identificateur déclaré).

Le contrôle est enfin rendu à E pour traiter la première unité syntaxique suivant ; .

D2 - TABLEAUX

Si TABLEAU a été trouvé en E, ou s'il suit REEL, ENTIER ou BOOLEEN, la disposition est la suivante :

$$\underline{\text{TABLEAU}}\ \underbrace{\underset{v1}{ID_1}\ ,\ \underset{v2}{ID_2}\ ,\ \ldots\ ,\ \underset{v3}{ID_{n1}}}_{\text{Première section de tableau}}\ \underset{v4}{[}\ \underset{v5}{[\ ]}\ ,\ \underset{v6}{\ ,\ ]}\ ,\ \underbrace{\underset{v7}{ID_{n1+1}}\ ,\ \ldots\ ID_{n2}\ldots}_{\text{Deuxième section de tableau}}\ ;$$

Deux aiguillages sont alors positionnés de façon à différencier d'une part le traitement des virgules telles que v1, v2, v3 de celui des virgules ordinaires v4, v5 de liste de paire de bornes, d'autre part le traitement du [marquant le début des paires de bornes d'une section de celui des symboles [ordinaires apparaissant éventuellement dans ces bornes.

Puis le contrôle est retourné à E qui traitera les identificateurs ID_1 ID_2 ... ID_{n1} comme des opérandes ordinaires.

Rappelons que si plusieurs sections de tableau se suivent, les virgules du type v6 remplacent le symbole TABLEAU et que la branche opérateurs renvoie alors le traitement à la branche déclarateurs. Ceci permet de traiter à nouveau les virgules du type v7 comme celles du type v1 v2 v3 .

Notons que contrairement aux autres déclarations, le symbole ; n'a pas ici de traitement spécial, la déclaration se terminant obligatoirement par] , qui active les phases de génération correspondantes.

D3 - PROCEDURES

Que PROCEDURE ait été détecté en E, ou qu'il suive immédiatement un déclarateur de type, la disposition la plus générale des unités syntaxiques qui suivent est :

PROCEDURE	P	(	PF1,		PF2,	...	PFn	)	;		
1	2	3	4	5	6	7	9	10	11		
VALEUR	PF_{v1}	,	PF_{v2}	,	...	PF_{vr}	;				
12	13	14	15	16		17	18				
Spécifieur	PF_{s1}	,	...	PF_{sm}	;						
19	20	21		22	23						
Spécifieur	$PF_{s'1}$	,	...	$PF_{s'm'}$	;						
24	25	26		27	28						
Spécifieur	$PF_{s(p)}$	,	...	$PF_{s(p)_m(p)}$	;						
29	30^1	31		32	33						

Corps de procédure
;

La branche qui nous intéresse a alors pour rôle de traiter toutes les unités syntaxiques de l'en-tête de procédure (ici 1 à 33) et aussi toutes celles du corps si celui-ci est en code machine.

Seul le ; marquant la fin de déclaration ne peut pas être traité ici.

La branche commence par insérer dans P_N un opérateur artificiel sans opérande $\downarrow^1$.

Cet opérateur, commun au déclarateur AIGUILLAGE, active dans PC_2 - G la phase P20 qui assurera dans P_M l'isolement du corps de procédure par rapport au programme principal.

Puis, on met à jour le compteur CIM1.

Ce compteur contient la valeur maximum atteinte par les indicateurs du BIM occupés par une procédure précédente.

On verra que si aucun FIN ne sépare le ; final de cette première procédure du symbole PROCEDURE qui nous intéresse, la mise à jour effectuée est inutile.

Par contre, si un FIN a été rencontré qui marquait la fin de la portée de la première procédure, le niveau NIM a repris la valeur qu'il avait au début de cette portée, et on réinitialise CIM1 par cette valeur, toutes les mémoires de manoeuvre à partir de NIM étant à nouveau utilisables (Cf. F).

Cette mise à jour est commune à la branche AIGUILLAGE.

Puis, D3 étant à zéro, vient l'analyse proprement dite de l'en-tête de procédure.

On commence par emmagasiner dans PQTM les valeurs atteintes par les diverses numérotations :

Quantités simples locales (COQLR) ou rémanentes (COQPR),

Tableaux locaux (COTLR) ou rémanents (COTPR),

Mémoires de manoeuvre (CIM1).

Ces valeurs seront comparées en fin de procédure à celles atteintes par les mêmes numérotations afin de déterminer les mémoires à protéger en cas d'appel récursif de la procédure.

L'identificateur de procédure P (unité syntaxique 2) est mis dans la pile PPTA d'où il sera retiré également en fin de procédure.

Enfin, le compteur de paramètres formels COPARF est réinitialisé, après emmagasinage de sa valeur en PPF, également utilisée en fin de procédure.

Si l'unité qui suit n'est pas (, c'est que la procédure est sans paramètres. Le compteur COPARF reste à zéro, on passe directement à l'en-tête de procédure (unité syntaxique 11).

Sinon, chaque paramètre PF_i est traité.

On vérifie la présence des virgules du type 5, 7 ...

Chaque paramètre donne lieu à l'insertion d'un second opérateur virtuel sans opérande, $\downarrow^{1*}$ dans PN. Ce sera pour PC_2 - G la phase P18 - 1, générant une instruction dite "vecteur de transfert".

Quand la fin de la liste de paramètres est atteinte (unité 10), ou qu'il n'y avait pas de paramètres, on vérifie que l'unité qui suit (11) est ; .

Il y a alors définition d'une mémoire de manoeuvre M - m pour la procédure.

Cette mémoire est une mémoire spécifique de bloc, la notion de bloc étant ici étendue au corps de procédure.

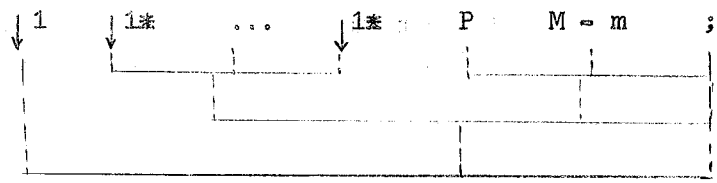
Elle contiendra pendant l'exécution de la procédure la valeur du niveau NDYNL de la mémoire à l'entrée dans cette procédure.

La définition de M - m se fait à l'aide d'un programme commun à PC_2 - G (DEFMEM), qui sera décrit plus loin. Notons seulement qu'il met à jour les compteurs CIM, CIM1 et CIM2 par la valeur m trouvée.

Le profil qualitatif [M - m] est alors emmagasiné dans la pile PMB.

Enfin, M - m et l'identificateur de procédure lui-même (situé en PPTA) constituent dans PN les opérandes du symbole ; . PC_2 - G placera ici la phase P18 - 2 , commune à tous les paramètres d'une même procédure, et réalisant les initialisations necessaires à l'exécution de celle-ci.

A ce niveau, la chaîne postfixée s'écrit donc :



Il peut alors apparaître dans PC_2 - G un véritable point de génération : P4, relatif au symbole VALEUR.

Pour PC_2 - A , le traitement de VALEUR consiste à placer tous ses opérandes en PN, suivis de VALEUR lui-même.

On vérifie en même temps la bonne construction de la partie valeur.

La chaîne postfixée C' prend alors la forme

..... ; PF_{v1} PF_{v2} ... PF_{vr} <u>VALEUR</u>

Notons que cette disposition n'est pas ambigüe, le nombre d'opérandes du symbole VALEUR étant tout à fait défini par la présence de ; .

Quand la partie valeur est épuisée, ou qu'il n'y en avait pas, on épuise, sans traitement, les parties spécifications, dont l'analyse a été effectuée en fait par PC_1 .

Quand on trouve dans C un ; non suivi d'un spécifieur, le corps de procédure commence.

Si la première unité de ce corps n'est pas CODE, il s'agit d'une instruction ALGOL, et le contrôle est rendu à E qui traite cette unité.

S'il s'agit d'un code machine, il est traité en entier ici :

L'opérateur CODE est supprimé, on place en PN tous les

groupes de six caractères du corps en code recopiés par PC_1 .

Quand 'FCODE' est trouvé, ces groupes de caractères sont considérés comme ses opérandes. (Là encore il n'y a pas d'ambiguïté, étant donnée la présence de VALEUR ou de ; dans PN). Pour PC_2 - G ce sera la phase P19.

Le traitement continue en E, pour analyser le ; de fin de déclaration qui suit obligatoirement FCODE dans P_S .

D4 - AIGUILLAGE

E ayant détecté le symbole AIGUILLAGE, la configuration des unités syntaxiques de C est la suivante :

AIGUILLAGE A := E1, ... , E_p ;

E1 ... E_p étant des expressions de désignation.

Ces expressions devant être analysées comme des expressions ordinaires, il y a retour en E, C étant positionnée sur le début de E1.

Auparavant, l'ensemble A := est traité d'une façon analogue aux premières unités syntaxiques de procédure :

L'identificateur A est placé dans une mémoire spécialisée MEMAIG, qui joue pour les aiguillages le rôle de PPTA pour les procédures. Les déclarations d'aiguillages ne pouvant pas s'imbriquer, une mémoire suffit là où il fallait une pile.

On vérifie ensuite que le numéro de l'aiguillage ne dépasse pas le maximum autorisé (paramètre d'assemblage de PC_2 au même titre que les encombrements de piles).

Puis, la phase P20 est déroulée, comme pour une procédure. Rappelons qu'elle consiste ici à placer l'opérateur virtuel $\downarrow^1$ dans PN.

Enfin, la mise à jour de CIM1 est faite, comme pour une procédure.

D3 étant mis à un auparavant, on effectue ensuite l'initialisation de l'analyse de la première expression E1, en mettant un dans

le compteur d'expressions COEA et l'opérateur virtuel $\downarrow^{8}$ dans PN, qui sera remplacé par la phase P29.

Ainsi la notation postfixée obtenue est simplement

$\downarrow^{1}$ $\downarrow^{8}$

et le traitement continue normalement avec le début de E1.

Notons que la même branche met l'aiguillage D4 à un, ce qui permet à la branche opérateurs d'affecter un traitement spécial au point-virgule terminant la déclaration.

D5 - VARIABLES PERSISTANTES

Quand le symbole PERSISTANT est trouvé, la chaîne codée peut avoir deux configurations :

1°) PERSISTANT REEL X1, X2, ... Xp ;

2°) PERSISTANT REEL TABLEAU T1, T2, ... Tp [...]

où REEL peut être remplacé par ENTIER ou BOOLEEN .

Dans le premier cas, on a affaire à une déclaration simple, l'indicateur IPERS est mis à un et on retourne à E pour traiter le déclarateur de type.

On a vu que ce traitement remettait immédiatement IPERS à zéro.

Dans le deuxième cas, il s'agit d'une déclaration de variable indicée, et un opérateur virtuel $\downarrow^{4}$ est placé en PN pour donner lieu, dans PC_2 - G, à la phase d'initialisation P17.

Puis la déclaration est traitée comme quand l'on trouve TABLEAU.

Remarquons qu'il n'est pas possible de trouver ici la configuration

PERSISTANT TABLEAU T ... ,

cette erreur syntaxique étant détectée par PC_1 .

E - BRANCHE OPERATEURS

E1 - CONSTITUTION GENERALE - SOUS - PROGRAMMES

La branche opérateurs (SOPER) est la plus importante du programme de production en notation postfixée.

Elle a pour fonctions essentielles d'une part d'emmagasiner dans la pile PR les symboles du langage susceptibles de donner lieu à un point de génération, d'autre part de supprimer complètement dans C' toute structure parenthésée ou à crochets.

Ceci implique qu'elle délimite les opérandes composés (parenthèses arithmétiques, variables indicées, instructions procédures et indicateurs de fonction, proposition SI, listes de POUR, etc ...).

Enfin, elle gère complètement les piles relatives aux niveaux du bloc de mémoires de manoeuvre et à la mémoire dynamique.

Nous avons vu qu'une partie de cette gestion se faisait dans les branches opérandes et déclarateurs. En fait, il s'agit de véritables parties de la branche opérateurs dispersées dans ces branches (PROCEDURE, (d'appel de procédure, etc ...).

La branche opérateurs dispose, pour réaliser ces fonctions, de quatre sous-programmes que nous décrirons plus loin (Cf. F2) et qui sont :

DEFMEM, commun à PC_2 - G , et que les branches opérandes et déclarateurs utilisent.

Il fournit un numéro m de mémoire de manoeuvre utilisable au moment où on l'appelle.

Les compteurs CIM, CIM1 et CIM2 y sont automatiquement mis à jour par m si c'est nécessaire.

Au retour de ce programme, la mémoire M - m est considérée comme employée et ne peut être libérée qu'en annulant l'indicateur correspondant dans BIM.

C , qui analyse le profil qualitatif du "bloc" en cours de compilation et en déduit s'il faut ou non effectuer la sauvegarde ou la mise à jour de NDYNL. Ce profil qualitatif est contenu dans la pile PB pour les véritables blocs et pour les paramètres effectifs, dans la mémoire indicatrice IAIG pour les expressions des listes d'aiguillages (cf. F3). Au retour de C, un opérateur virtuel $\downarrow^{10}$ ou $\downarrow^{10*}$ a pu être placé dans P_N .

C est également appelé par la branche opérandes (paramètres formels par nom, appels de procédures avec ou sans paramètres).

B , également utilisé par OPERA , qui permet de changer les profils qualitatifs des opérateurs + et - dans les cas où ces symboles sont relatifs à des opérations unitaires. B est appelé chaque fois qu'un opérateur est susceptible d'être suivi par + ou -

Au retour, + ou - est remplacé dans C par un symbole à un seul opérande que nous écrirons ⊕ ou ⊖ et qui fournira au point deux une sous-phase de génération spéciale.

S , appelé quand on trouve une fin d'instruction, et qui traite les structures syntaxiques de la forme

FAIRE SI <expression booléenne> ALORS <instruction>;

ou

SINON SI <expression booléenne> ALORS <instruction>;

Ce sous-programme est susceptible de placer en PN les symboles FAIRE ou SINON (point huit de génération).

Dans la description qui suit, de chaque séquence de traitement, un même point ou une même phase de génération peuvent apparaître en plusieurs endroits. Ceci ne signifie pas toujours que la phase est construite comme un sous-programme fermé (ce n'est en particulier jamais le cas pour les points véritables de génération), mais qu'elle se divise naturellement en plusieurs parties relatives à ces différents appels. Nous retrouverons ces divisions dans la description de $PC_2 - G$.

E2 - DEBUT et FIN

Les symboles DEBUT et FIN délimitent dans P_S une unité syntaxique précise, qui est soit un bloc si DEBUT est suivi d'un déclarateur, soit une instruction composée.

Cette propriété n'est partagée que par le couple formé d'un symbole FAIRE et du symbole FIN ou ; associé marquant la fin d'une boucle.

Dans les deux cas, l'ensemble des mémoires de manoeuvre utilisées entre ces symboles forme un tout, ce qui se traduit ici par la mise en réserve dans la pile PNIM du niveau NIM à partir duquel les mémoires de manoeuvre sont utilisables.

En plus, un profil d'instruction est défini et mis en réserve dans la pile PB.

Si l'instruction est un bloc, ce profil contient la position D et une mémoire spécifique est définie, son profil qualitatif mis en réserve dans PMB et libérée en FIN. Dans le cas du premier bloc, cette mémoire est M - 1 , on verra qu'elle contient, au début de l'exécution de P_M , la valeur initiale de NDYNL.

Dans le cas d'une instruction composée, le profil défini est nul.

Notons qu'en FIN , le programme S est effectué, de façon à faire "descendre" de PR les symboles FAIRE et SINON susceptibles de s'y trouver encore. Puis on vérifie que PR contient le symbole DEBUT , qui y avait été placé, comme un symbole ordinaire, après le traitement ci-dessus.

DEBUT est alors supprimé par simple recul de PR.

Enfin, des phases de génération sont associées à DEBUT et FIN, qui ont pour rôle, comme on le verra, d'initialiser et d'épuiser la pile d'étiquettes internes PE.

E3 - POINT-VIRGULE

La détection de ; dans la chaîne codée C peut marquer

1°) La fin d'une instruction
2°) La fin d'une déclaration de procédure
3°) La fin d'une déclaration d'aiguillage

E3 - 1 Fin d'instruction

L'aiguillage D4 est à zéro. La compilation d'une instruction est terminée, le symbole ; ayant une hiérarchie un inférieure à celle de tous les opérateurs susceptibles d'y apparaître.

On peut vérifier alors que toutes les mémoires de manoeuvre utilisées dans l'instruction sont libérées. Pour cela, tous les indicateurs de BIM correspondant à des numéros compris entre NIM et CIM1 doivent être nuls.

S'ils ne le sont pas tous, l'instruction est mal construite ; en particulier elle peut comporter des expressions incomplètes, par exemple

A [I] := ;

ou

X := A + ;

etc ...

On vérifie pour la même raison que la pile PN est vide, puisqu'il ne doit rester aucun opérande à traiter. Une telle erreur serait provoquée par une structure de la forme

X := A (B + C) ;

où A , identificateur ordinaire, ne serait pas suivi d'un opérateur arithmétique par exemple.

Dans ce cas, la notation postfixée obtenue serait

X A := B C +

et l'opérande (B + C) resterait inutilisé.

Notons que cette vérification, qui nous paraît intéressante, oblige à interdire dans P_S l'emploi d'un identificateur de procédure déclarée avec un type (indicateur de fonction) dans une instruction procédure. Ces indicateurs devront toujours être employés comme primaire d'une expression. Ceci n'est pas une véritable restriction, l'instruction procédure pouvant toujours se ramener dans ce cas à l'affectation à une variable fictive de la valeur de l'indicateur.

Ces vérifications étant effectuées, on déroule le sous-programme S et ";" est mis en P_N pour simuler le point P_{10} de génération (gestion de la pile PE analogue à celle effectuée en FIN).

L'analyse continue avec l'unité syntaxique suivant " ; " .

E3 - 2 Fin de procédure

Le point-virgule marquant la fin d'une déclaration de procédure comporte dans son profil la position quinze mis à un (cf. première partie).

Le traitement est d'abord celui d'une fin d'instruction, le corps de procédure étant formé d'une instruction qui

se termine ici, quelque soit sa forme (instruction composée, bloc, instruction conditionnelle, instruction POUR). Si le corps est en code machine, l'ensemble 'CODE' texte 'FCODE' est considéré également comme une instruction.

Puis, on traite la fin de procédure proprement dite. Ce traitement est symétrique de celui de l'en-tête de procédure :
on libère la mémoire spécifique définie pour la procédure, on efface un niveau dans PQTM et PPTA, et COPARF reprend la valeur qu'il avait à l'entrée dans la procédure. Auparavant, l'index $\downarrow^{***}$ a été placé en PN (point P5), avec pour opérande l'identificateur de procédure qui avait été mis en réserve dans PPTA.

Enfin, le compteur NIM prend la valeur immédiatement supérieure à celle atteinte par CIM1 au cours de la compilation du corps de procédure. Rappelons que la valeur NIM ne pourra reprendre une valeur inférieure qu'en trouvant le symbole FIN marquant la fin de la portée de la procédure la plus extérieure à la procédure en cours (Cf. F3).

Comme pour une fin d'instruction, l'analyse reprend en E avec l'unité suivant ";" .

E3 - 3 Fin d'aiguillage

Quand le symbole AIGUILLAGE est analysé, on peut affirmer que le premier " ; " rencontré ensuite dans C marque la fin de la déclaration correspondante.

Pour cela, l'aiguillage D4 est mis à un.

Dans ce cas, la fin de la dernière expression dans la liste d'aiguillages est traitée (opérateur $\downarrow^{8*}$, phase P29) et la mémoire spécifique éventuellement définie pour cette expression, considérée comme un bloc, est libérée.

L'opérateur $\downarrow^{**}$ marquant la fin de la déclaration est alors placé en PN (point P9). Il a pour opérande l'identificateur d'aiguillage placé dans MEMAIG.

Enfin, les mêmes vérifications sont effectuées que pour une fin d'instruction, et NIM prend la valeur immédiatement supérieure à celle de CIM1, l'ensemble des expressions formant la liste d'aiguillage devant avoir les mêmes propriétés d'indépendance qu'un corps de procédure (Cf. F3).

E4 - CROCHETS DE VARIABLES INDICEES ET D'AIGUILLAGES

La structure syntaxique de la forme

identificateur [....]

peut correspondre aux structures sémantiques suivantes :

1°) Fin d'une section dans une déclaration de tableau

2°) Exploitation d'une variable indicée

3°) Exploitation d'un aiguillage.

E4 - 1 Déclaration de tableau

Rappelons que la structure syntaxique complète d'une déclaration de tableau est

$$\left\{ \begin{array}{l} \text{local ou} \\ \underline{\text{REMANENT}} \end{array} \right\} \ \underline{\text{TABLEAU}} \ \ T1, \ \ldots\ldots, \ T_P \ [\ \ldots\ldots\ ] \ T_{P+1} \ \ldots$$

Quand TABLEAU est analysé (Cf. D), les aiguillages D1 et D2 sont mis à un.

On verra (Cf. E6) que D2 à un permet de transférer les identificateurs T1 ... T_{P-1} de PN (où ils sont placés par la branche opérandes) dans PPTA où ils sont mis en réserve jusqu'à la fin de la section, marquée par] .

Quand [est rencontré et que D1 est à un, le dernier identificateur T_P est mis en PPTA et les opérateurs [et $\downarrow^*$ sont placés dans cet ordre dans la pile PR. L'opérateur $\downarrow^*$ bloque au point H le test de hiérarchies de l'opérateur [susceptible de "descendre" de PR en PN.

Les aiguillages D1 et D2 sont alors restaurés, la compilation entre [et] étant de toute façon celle d'expressions ordinaires, simplement séparées par les symboles, et : .

Si la déclaration était précédée de REMANENT, l'indicateur IPERS est à un, et le compteur de tableaux rémanents COTPR est mis à jour par le numéro du dernier identificateur T_P . Sinon c'est COTLR, compteur de tableaux locaux, qui est mis à jour.

Enfin, on place en PN l'opérateur $\downarrow^7$ qui donnera lieu à l'initialisation de l'appel d'une procédure standard, comme cela est fait dans la branche opérandes en trouvant "(" de procédure (phase P25). Cette procédure, est en fait un programme interpréteur inclus dans la bibliothèque du système, qui, à l'exécution de P_M , construit effectivement la réservation des mémoires correspondant à la déclaration. Nous symbolisons par TABF le profil qualitatif de cette procédure. Ce profil est placé en PPTA comme tout identificateur de procédure en exploitation, pour être utilisé par la compilation de la liste des paramètres effectifs (qui sont ici les bornes du tableau) et de la fin de l'appel, où il sera supprimé.

Enfin l'opérateur virtuel $\downarrow^{7*}$ est placé en PN pour marquer le début du premier paramètre effectif (première borne, phase P26).

Remarquons que le sous-programme B est effectué en tête du traitement, "[" pouvant être suivi de + ou - qui sont alors unitaires.

Quand "]" est rencontré et qu'il marque la fin d'une section de tableau ($\downarrow^{*}$ en PR), le véritable appel de la fonction TABF peut se faire.

Pour cela, on transfère le profil de TABF de PPTA dans la mémoire spécialisée MEM APPEL et un indicateur de fonction standard est mis à un.

On termine le traitement de la dernière borne du tableau, par $\downarrow^{7**}$ mis dans PN (phase P27 de fin de paramètre effectif).

Puis, l'appel proprement dit est effectué, en descendant l'index $\downarrow^{*}$ de PR en PN, précédé de l'opérande TABF (P 6). Enfin, $\downarrow^{*}$ est répété autant de fois qu'il y a d'identificateurs de tableau en réserve dans PPTA. Ces identificateurs T1 ... T_P servent alors d'opérandes à $\downarrow^{*}$.

La pile PPTA est automatiquement positionnée sur sa dernière mémoire non utilisée. Notons qu'il ne peut alors rester dans PPTA que des identificateurs de procédures en déclaration.

Ainsi, la chaîne postfixée correspondant à la structure syntaxique indiquée est, au moment où l'on trouve "[" :

$\downarrow^{7}$ $\downarrow^{7*}$

si la déclaration est locale et

$\downarrow^{4}$ $\downarrow^{7}$ $\downarrow^{7*}$

si elle est précédée de REMANENT.

Quand on trouve "]", cette chaîne est

$\downarrow^{7**}$ TABF $\downarrow^{*}$ T_P $\downarrow^{*}$ T_{P-I} $\downarrow^{*}$ ...T_1 $\downarrow^{*}$

Enfin, on teste si la déclaration est terminée. Si une autre section suit (virgule après]), le contrôle est rendu à la branche déclarateurs.

Si la déclaration est terminée (; après]), un dernier opérateur ($\downarrow^{4*}$) est mis en PN, pour donner lieu à la phase P22, si la déclaration était persistante. IPERS est alors restauré.

Dans tous les cas, l'analyse se poursuit en E avec l'unité suivant ; .

E4 - 2 Exploitation de variable indicée

Ce cas diffère du précédent par le fait que l'exploitation est relative à un seul identificateur T, que la procédure à appeler est évidemment différente, et enfin que le symbole REMANENT n'intervient pas.

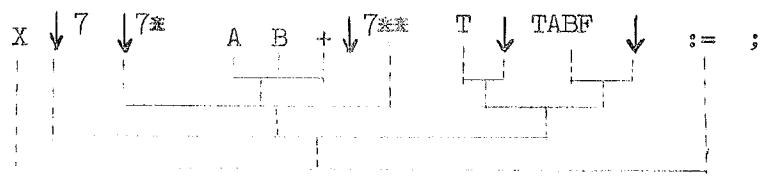
On voit que pour marquer cette distinction, l'opérateur placé en PR par l'analyse de "[" est ↓ (le même que pour un appel de procédure, dont le traitement est tout à fait analogue en OPERA), que cet index entraîne l'exécution de la phase P7 et enfin que le contrôle est rendu après analyse de] au point GA pour traiter l'opérande composé ainsi délimité.

Les paramètres de la procédure artificielle TABF sont alors les indices de la variable.

Pour une variable à un seul indice, l'instruction

X := T [A + B] ;

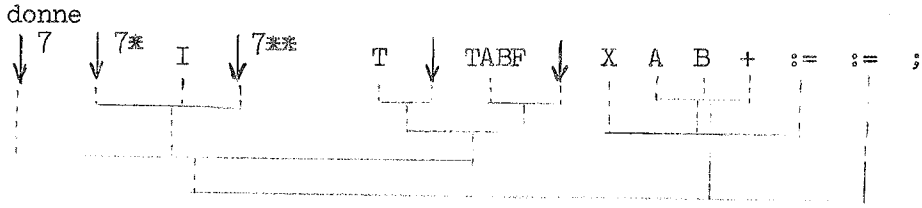
fournit la chaîne C' suivante :



Pour une variable indicée en partie gauche nous avons de même :

T [I] := X := A + B ;

qui donne



On verra (Cf. E5) que l'appel d'un indicateur de fonction est tout à fait analogue au premier exemple ci-dessus, la plus grande différence est que ↓ n'apparait qu'une fois en fin d'appel, l'identificateur de la procédure étant seul à définir ce que doit faire la phase P7, alors qu'il est accompagné ici de l'identificateur de tableau.

E4 - 3 Exploitation d'Aiguillage

Le cas d'une exploitation d'aiguillage est simplifié par le fait qu'un seul indice (c'est à dire une seule expression) est permis entre "[" et "]", et que l'appel de la fonction représentant l'aiguillage est entièrement défini par l'identificateur de celui-ci.

On peut remarquer que la compilation d'un appel d'aiguillage est ainsi tout à fait analogue à celui d'une fonction ouverte, à ceci près que l'exécution du programme C est obligatoire, comme pour une procédure ordinaire.

Comme pour les variables indicées, on exécute B en trouvant "[", et l'identificateur est placé en PPTA.

Puis l'opérateur ↓ est placé en PR à la suite de [, pour bloquer, comme dans le cas précédent, les opérateurs de PR susceptibles de "descendre" sur ceux de C au point H. Aucune chaîne postfixée particulière n'est créée ici.

En trouvant "]" de fin d'aiguillage (↓ en PR et identificateur d'aiguillage en PPTA), l'identificateur est retransféré de PPTA à PN, où il servira d'opérande à l'index ↓. Le programme C, si l'aiguillage est non simple (Cf. F3), aura placé en PN l'opérateur $\downarrow^{10}$ ou $\downarrow^{10*}$.

Enfin, comme dans les autres cas de crochets, le symbole "[" est ignoré dans PR, et l'analyse continue avec l'unité suivant "] ", directement au point H puisque l'opérande composé ainsi délimité ne peut pas être l'élément d'une partie gauche.

Ainsi, pour l'instruction

<u>ALLER A</u> A [N1 + N2]

on obtiendra en C' :

N1 N2 + A ↓ <u>ALLER A</u>

si A a été trouvé "simple" à sa déclaration et

N1 N2 + A ↓ 10 ↓ <u>ALLER A</u>

ou N1 N2 + A ↓ 10* ↓ <u>ALLER A</u>

s'il est trouvé non simple (Cf. F3).

E5 - EXPLOITATION DE PROCEDURE : ")"

Quand ")" est trouvé dans la chaîne postfixée C, il peut s'agir d'une fin de parenthèse arithmétique, booléenne, ou de désignation, ou d'une fin d'appel de procédure, que l'identificateur de celle-ci soit indicateur de fonction ou non.

Dans tous les cas, ")" marque la fin d'un opérande composé, susceptible de donner lieu au point deux de génération.

Dans le cas d'une fin de parenthèse, il y a retour direct au point H, après vérification de la présence de "(" en PR et recul de celle-ci.

Dans le cas d'un appel de procédure, le traitement est analogue, on l'a vu, à celui de "] ", et est symétrique de celui de "(" dans la branche opérandes.

Notons seulement que l'opérateur principal, dans ce cas, est toujours ↓, avec pour seul opérande l'identificateur de procédure mis en réserve dans PPTA.

Si une mémoire spécifique a été définie pour le dernier paramètre effectif (position B dans le profil courant dans PB), cette mémoire est libérée ici. La pile PB a son niveau diminué de un.

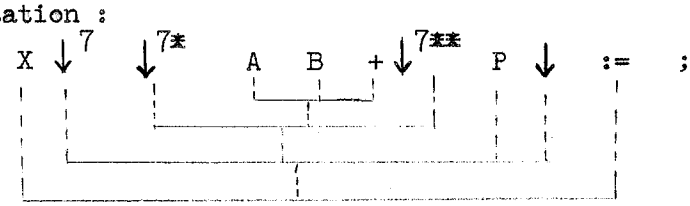
Enfin, dans tous les cas de procédure non standard, le compteur CIM2 est remis à jour, si cela est nécessaire, par la valeur contenue au sommet de PCIM2.

Le retour se fait en H après élimination dans PR des symboles ↓ et "(" et positionnement de C sur l'unité suivant ")".

Ainsi, pour l'instruction :

X := P (A + B) ;

on aura la notation :



Rappelons que si la procédure P était sans paramètre on aurait :

X P ↓ := ;

avec dans tous les cas de procédure non standard, l'éventuelle insertion de ↓10 ou ↓10* par le programme C .

E6 - SEPARATEURS ":" ET ","

Le symbole " : " ne peut être trouvé en E que s'il sépare deux bornes dans une déclaration de tableau. Il a un rôle tout à fait analogue aux symboles "," séparant les paramètres effectifs de procédures.

Le symbole virgule trouvé en E peut, lui, appartenir à quatre structures syntaxiques différentes :

1°) Virgule de section de tableau

2°) Virgule de liste d'aiguillage

3°) Virgule de liste de POUR

4°) Virgule de liste de bornes, d'indices ou de paramètres effectifs. C'est à ce cas que se rattache le traitement de " : " .

E6 - 1 Virgule de section de tableau

Nous avons vu que l'analyse de ce type de virgule est caractérisée par l'aiguillage D2 à un.

Le traitement consiste uniquement à transférer l'identificateur de tableau situé dans PN dans la pile PPTA où il sera repris en fin de déclaration pour servir d'opérande à $\downarrow^{*}$ (P6) .

E6 - 2 Virgule de liste d'aiguillage

Ce type de virgule sépare deux expressions de désignation d'une liste d'aiguillage. Il ne peut alors y avoir dans PR que des symboles 'DEBUT' .

On place en PN les opérateurs $\downarrow^{8*}$ (P29) et $\downarrow^{8}$ (P28) pour marquer successsivement la fin d'une expression et le début de l'autre.

Avant de placer $\downarrow^{8}$, on libère la mémoire spécifique éventuellement définie pour l'expression terminée et on réinitialise l'indicateur d'aiguillage IAIG avec la position A indiquant qu'on est dans un bloc susceptible d'une mise à jour de NDYNL par C .

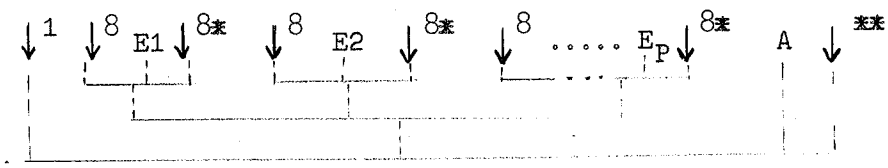
Enfin, le compteur d'expressions COEA est augmenté de un et on vérifie que PN est vide, aucun opérande ne pouvant rester inutilisé à cet endroit.

L'analyse continue en E avec l'unité suivant ",".

Ainsi, la déclaration complète

<u>AIGUILLAGE</u> A := E1, E2, ... E_P ;

où E1 .... E_P sont des expressions de désignation, fournit la chaîne postfixée suivante :



<u>E6 - 3 Virgule de liste de POUR</u>

Il s'agit du type de virgule séparant deux éléments d'une liste de POUR .

Si l'élément précédent était une expression arithmétique, PR contient la suite de symboles

$\downarrow^{2}$ <u>POUR</u>

Sinon (élément avec PAS ou TANTQUE), l'index $\downarrow^{5}$ a été placé à la suite de POUR par le point H . Dans ce cas, on élimine $\downarrow^{5}$ en reculant PR d'une ligne.

Dans tous les cas le symbole virgule est placé en PN pour simuler le point P11, et l'analyse continue avec la premiè-

re expression de l'élément suivant.

Le dernier opérateur de PR est donc toujours POUR après analyse de "," .

On verra qu'ainsi chaque élément d'une liste de POUR entraînera l'exécution du point deux de génération pour le symbole POUR, ce qui assure la réinitialisation de la variable contrôlée à chaque début d'élément.

E6 - 4 Virgule de liste de paramètres

Ce cas est celui d'une virgule séparant deux paramètres effectifs d'un appel de procédure. Il s'y rattache les cas de virgule séparant deux expressions en indices, de ":" séparant les bornes inférieure et supérieure d'une même paire et de virgule séparant deux paires de bornes.

Dans tous ces cas, la pile PR contient à son sommet l'un des symboles $\downarrow$ (virgule de paramètres ou d'indices) ou $\downarrow^{*}$ (virgule ou ":" de liste de bornes).

L'identificateur de la procédure appelée (TABF pour les bornes et indices) est transféré de PPTA en MEM APPEL, l'indicateur de procédure standard est mis à la valeur convenable, et l'opérateur $\downarrow^{7**}$ est placé en PN pour marquer la fin d'un paramètre effectif. (P27).

Dans le cas d'une virgule séparant deux paramètres effectifs d'une procédure déclarée, on libère la mémoire spécifique éventuellement définie pour le"bloc" du paramètre termine et le profil du bloc dans PB est réinitialisé (position A). Enfin, CIM2 est emmagasiné dans la pile spécialisée.

Dans tous les cas, B est exécuté et l'analyse continue avec la première unité du paramètre suivant, après avoir placé en PN l'opérateur $\downarrow^{7*}$ (P26) pour marquer le début de ce paramètre.

Nous pouvons décrire maintenant la chaîne postfixée complète obtenue dans chacun des cas examinés.

Dans ce qui suit nous représenterons par T_i un identificateur de tableau, par A_i une expression arithmétique, par P_i un identificateur de procédure, par PE_i un paramètre effectif, et par C' (c) la chaîne postfixée obtenue pour la partie c de chaîne codée.

Ainsi, la déclaration de tableau suivante :

<u>TABLEAU</u> T1, T2 [A1 : A2, A3 : A4], T3 [A5 : A6] ;

fournit la chaîne :

$$\downarrow^{7} \downarrow^{7*} C'(A1) \downarrow^{7**} \downarrow^{7*} C'(A2) \downarrow^{7**} \downarrow^{7*} C'(A3) \downarrow^{7**} \downarrow^{7*} C'(A4) \downarrow^{7**}$$
$$\text{TABF} \downarrow^{*} \text{T2} \downarrow^{*} \text{T1} \downarrow^{*} \downarrow^{7} \downarrow^{7*} C'(A5) \downarrow^{7**} \downarrow^{7*} C'(A6) \downarrow^{7**}$$
$$\text{TABF} \downarrow^{*} \text{T3} \downarrow^{*}$$

Dans cette chaîne, la première occurrence de $\downarrow^{7}$ peut être précédée de l'opérateur virtuel $\downarrow^{4}$ et la dernière occurrence de $\downarrow^{*}$ peut être suivie de $\downarrow^{4*}$ si la déclaration est précédée du symbole REMANANT.

De même l'expression :

T1 [A1, A2] + T2 [A3]

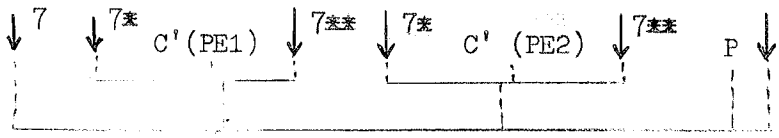
donne :

$$\downarrow^{7} \downarrow^{7*} C'(A1) \downarrow^{7**} \downarrow^{7*} C'(A2) \downarrow^{7**} \text{T1} \downarrow \text{TABF} \downarrow \downarrow^{7} \downarrow^{7*} C'(A3) \downarrow^{7**} \text{T2} \downarrow \text{TABF} \downarrow +$$

Enfin, l'appel de procédure :

P1 (PE1, PE2)

est transformé en :



Rappelons que si la procédure est déclarée dans le programme, il y a exécution du programme C à l'analyse de (et qu'en conséquence $\downarrow^{10}$ ou $\downarrow^{10*}$ sont susceptibles d'apparaître dans C' avant $\downarrow^{7}$.

E7 - OPERATEURS ORDINAIRES

Tous les symboles n'entrainant aucune génération immédiate et aucune vérification de l'état des piles de mémoires sont traités dans la branche centrale de l'organigramme.

Le traitement consiste, après exécution du sous-programme B, à placer l'opérateur au sommet de PR .

C'est le cas pour les opérateurs arithmétiques (+ , - , $\oplus$, $\ominus$, x , / , $\div$, $\uparrow$), de relation ($<$ $\leqslant$ = $\neq$ $\geqslant$ $>$) et logiques ($\vee$, $\wedge$, $\neg$, $\supset$, $\equiv$), ainsi que ALLER A, PAS, JUSQUA et TANTQUE, et enfin "(" quand celui-ci marque le début d'une expression arithmétique, booléenne ou de désignation simple.

Ce traitement réalise complètement la compilation des véritables opérations du langage et on verra qu'à chacun de ces opérateurs (sauf "(") est associée une sous-phase dans le point deux de génération, que l'on peut pour cette raison considérer comme le bloc de base du passage compilateur.

En plus, certains autres symboles (SI, SINON, POUR, FAIRE) subissent le même traitement après exécution des séquences spéciales qui leur sont réservées.

La table de hiérarchies adoptée a été construite de façon à ce que chaque opérateur ainsi placé dans PR ne "descende" dans PN que lorsque tous ses opérandes y ont été déjà placés.

L'insertion supplémentaire des symboles DEBUT, (, [, $\downarrow$, $\downarrow^{*}$, $\downarrow^{2}$, $\downarrow^{3}$, $\downarrow^{5}$ dans la pile PR n'est nécessaire que pour bloquer ce fonctionnement quand les règles syntaxiques l'imposent. Aucun de ces symboles ne "descend" automatiquement de PR, il y attend toujours que se fasse l'analyse du symbole marquant la fin de la zone parenthésée ou à forme de liste dont il est le début.

Enfin, les symboles suivants bien qu'apparaissant dans PN, ne sont jamais placés en PR et servent uniquement de séparateurs dans C' :

FIN ALORS , : ; VALEUR FCODE ' '

et tous les opérateurs virtuels non cités plus haut :

$\downarrow^{**}$ $\downarrow^{***}$

$\downarrow^{1}$ $\downarrow^{4}$ $\downarrow^{4*}$ $\downarrow^{6}$

$\downarrow^{7}$ $\downarrow^{7*}$ $\downarrow^{7**}$ $\downarrow^{8}$ $\downarrow^{8*}$

$\downarrow^{9}$ $\downarrow^{10}$ $\downarrow^{10*}$

E8 - OPERATEURS SI ALORS SINON

Les différentes valeurs sémantiques que peuvent prendre les symboles SI et SINON, caractérisées par les différentes natures de leurs opérandes, que nous avons déjà soulignées (cf. A), obligent à faire subir à ces opérateurs des traitements spéciaux.

En particulier un index $\downarrow^{3}$ est introduit sans lequel certains opérateurs devraient changer de hiérarchie en cours de compilation.

Soit par exemple, l'instruction conditionnelle suivante, où X, Y, Z sont des variables et B une expression booléenne :

<u>SI</u> B <u>ALORS</u> X := Y <u>SINON</u> X := Z ;

La chaîne postfixée correspondante doit être, comme on l'a vu :

C'(B) <u>ALORS</u> X Y := <u>SI</u> X Z := <u>SINON</u> ;

Considérons maintenant l'instruction équivalente écrite à l'aide d'une instruction d'affection et d'une expression arithmétique :

X := <u>SI</u> B <u>ALORS</u> Y <u>SINON</u> Z ;

Cette instruction, où <u>SI</u> et <u>SINON</u> ont des opérandes du type expression, doit fournir la chaîne suivante :

X C'(B) <u>ALORS</u> Y <u>SI</u> Z <u>SINON</u> := ;

Il apparait clairement que si le traitement de SI et SINON était celui d'opérateurs ordinaires, le symbole := devrait avoir une hiérarchie, dans le premier cas supérieure ou égale à celle de SINON, dans le deuxième cas strictement inférieure. Ce changement dans la priorité de := (et d'autres symboles) devient rapidement inextricable quand l'instruction conditionnelle du premier cas contient elle-même des instructions d'affectations du type de celle du deuxième cas.

Remarquons que le cas où la même expression serait rendue simple en l'enfermant dans des parenthèses fonctionnerait sans changement, le symbole := étant alors protégé en PR par le symbole (.

L'emploi de l'index $\downarrow^3$ résout le problème en "bloquant dans PR, à la façon d'une parenthèse, les opérateurs telsque " := " ici, tant que SINON n'a pas été détecté.

Cet index est placé dans PR, dans la ligne précédant SI , quand celui-ci est trouvé en C .

Le fonctionnement est normal jusqu'à ce que SI soit transféré de PR à PN par le point deux (le test de hiérarchies s'effectuant alors avec SINON). L'index $\downarrow^3$ arrête alors la génération au point deux, et SINON est analysé par E , de sorte qu'il reste à ce moment dans PR :

$\downarrow^3$ dans le premier cas

:= $\downarrow^3$ dans le second cas.

Le traitement de SINON consiste alors uniquement à vérifier la présence au sommet de PR de $\downarrow^3$, qu'il remplace immédiatement. De la sorte, on obtient dans les deux cas la chaîne postfixée désirée, SINON et := "tombant" dans cet ordre au point deux, sur le symbole ; dans les instructions du type du deuxième cas.

On vérifie facilement que ce fonctionnement assure correctement la compilation des instructions conditionnelles et des expressions arithmétiques non simples, quelle que soit la complexité de leur imbrication. En particulier, le " ; " de fin d'instruction peut être remplacé par un FIN ou un SINON.

Cependant, cette méthode est mise en défaut quand l'expression booléenne B est elle-même une expression non simple.

En effet, soit l'instruction :

<u>SI</u>	<u>SI</u>	B	<u>ALORS</u>	B1	<u>SINON</u>	B2	<u>ALORS</u>	A1	<u>SINON</u>	A2	;
s1	s2		s3		s4		s5		s6		

La notation postfixée à obtenir est :

C'(B) <u>ALORS</u> C'(B1) <u>SI</u> C'(B2) <u>SINON</u> <u>ALORS</u> C'(A1) <u>SI</u> C'(A2) <u>SINON</u> ;

et on voit que le symbole SINON (s4) dans C doit être de hiérarchie supérieure à ALORS (s5) afin de "tomber" au bon endroit.

Ce cas se présentant dans des conditions bien déterminées il a paru préférable à l'introduction d'un nouveau type d'index, de créer une exception au fonctionnement général : le traitement du symbole ALORS comporte pour cette raison une recherche de tous les symboles SINON situés au sommet de PR . Ces symboles sont successivement descendus dans PN pour donner lieu au point douze de génération.

Enfin, le traitement ordinaire de ALORS est effectué au retour du point H où, si aucune erreur n'est détectée, aucun symbole ne doit être tombé de PR en PN.

Le traitement ordinaire de ALORS consiste à le placer directement en PN où il symbolise le point P13 de génération.

Ce traitement s'accompagne de l'exécution du sous-programme B (pour les constructions du type <u>ALORS</u> $\pm$ < expression >) et du sous-programme C avec DC à un.

Ce dernier est déroulé pour assurer la sauvegarde du niveau NDYNL quelle que soit la branche de programme effectivement déroulée à l'exécution de P_M (cf. F2).

Enfin, les symboles SI et ALORS peuvent apparaître dans des propositions SI non suivies de SINON.

Les diverses constructions syntaxiques relatives à ce cas se ramènent aux trois suivantes, où B est une expression booléenne

et I une instruction.

1°) < DEBUT ou ;> SI <B> ALORS < I > <; ou FIN >

2°) SINON SI < B > ALORS < I > <; ou FIN >

3°) FAIRE SI < B > ALORS < I > <; ou FIN >

Le premier cas entre dans le fonctionnement général, le symbole SI "tombant" sur ; ou FIN au lieu de tomber sur SINON. FIN et ; ont alors un opérande, qui, on le verra, se réduit, dans PC2 - G , à une ou plusieurs étiquettes internes en PE.

Le symbole $\downarrow^3$ restant en PR est alors éliminé sans traitement.

Ainsi, l'instruction :

SI B ALORS X := Y ;

fournit la chaîne postfixée suivante :

C'(B) ALORS X Y := SI ;

Par contre, les cas 2 et 3 présentent la particularité de laisser en PR, bloqués par l'index $\downarrow^3$, un ou plusieurs symboles FAIRE ou SINON.

En plus, plusieurs symboles $\downarrow^3$ et SI peuvent être laissés en PR, dans le cas par exemple où une instruction POUR suit le symbole ALORS.

Pour obtenir dans tous les cas une chaîne postfixée correcte, il faut forcer ces symboles à venir en PN.

C'est le rôle du programme S (qui, en même temps élimine l'index $\downarrow^3$ de PR), exécuté pendant le traitement de ; et FIN comme nous l'avons déjà vu. Ce programme vide complètement la pile PR de tous les symboles SINON, FAIRE, SI et $\downarrow^3$ qu'elle contient. Les points de génération correspondant aux mêmes symboles quand on les trouve en H sont alors effectués.

Ainsi, l'instruction suivante :

<u>SI</u> B <u>ALORS</u> I1 <u>SINON</u> <u>SI</u> B' <u>ALORS</u> I2 ;

donne la chaîne :

C'(B) <u>ALORS</u> C'(I1) <u>SI</u> C'(B') <u>ALORS</u> C'(I2) <u>SI</u> <u>SINON</u> ;

De même, l'instruction plus complexe

<u>SI</u> B <u>ALORS</u> <u>POUR</u> L <u>FAIRE</u> <u>SI</u> B' <u>ALORS</u> I ;

où L est une liste de POUR, donne :

C'(B) <u>ALORS</u> C'(L) $\downarrow^{2}$ C'(B') <u>ALORS</u> C'(I) <u>SI</u> <u>FAIRE</u> <u>SI</u> ;

E9 - OPERATEURS := POUR ET FAIRE

La forme la plus générale d'une instruction POUR est la suivante :

POUR V := EL1, EL2 ..., EL_p FAIRE I ;

où V est une variable, qui peut être variable simple, variable indicée, ou paramètre formel, EL1 ... EL_p des éléments de liste de POUR et I une instruction.

Nous étudierons successivement la compilation de l'en-tête POUR V := , des éléments de liste, et enfin de la boucle proprement dite.

E9 - 1 En-tête de POUR :

Le symbole POUR subit dans tous les cas le traitement d'un opérateur ordinaire quand il est trouvé dans C.

En plus, si la variable contrôlée (qui le suit immédiatement dans C) est un paramètre formel par nom ou une variable indicée, l'opérateur virtuel $\downarrow^{2*}$ est placé en PN pour remplacer la phase P23 de génération.

Cette phase consiste à définir dans ce cas un sous-programme de calcul de l'adresse de la variable contrôlée à chaque exécution de la boucle.

Puis la variable contrôlée V est traitée par la branche opérandes.

En trouvant := dans C , l'état des piles est donc le suivant, POUR ne tombant pas en PN sur := , sa hiérarchie étant inférieure :

PR : POUR

PN : $\downarrow^{2*}$ C'(V)

Le traitement de := , quand POUR est trouvé en PR consiste alors à placer dans PR l'opérateur $\downarrow^2$ suivi lui-même de POUR. Cet opérateur bloque les opérateurs de PR le précédant durant la compilation de la liste de POUR entière.

Enfin l'opérateur virtuel $\downarrow^6$ est placé en PN (point P15) pour marquer la fin du sous-programme de calcul éventuel de V.

L'état des piles au moment où l'on trouve la première unité syntaxique de EL1 est alors :

PR : $\downarrow^2$ POUR

PN : $\downarrow^{2*}$ C'(V) $\downarrow^6$

E9 - 2 Eléments de liste de POUR

Chaque élément EL_i peut avoir l'une des trois formes suivantes :

1°) A_i

2°) A_i PAS A'_i JUSQUA A''_i

3°) A_i TANTQUE B_i

La fin d'un élément est marquée par FAIRE ou "," .

On voit que dans tous les cas le symbole POUR tombe au point H sur le symbole suivant la première expression arithmétique de l'élément (",," dans le premier cas, PAS dans le deuxième cas, TANTQUE dans le troisième cas).

La chaîne obtenue est donc toujours au début de l'élément :

C' (A_i) POUR

Il y a déroulement de la sous-phase du point deux relative à POUR , qui génère l'initialisation de la variable contrôlée.

Puis on teste (Cf. branche opérandes) si A_i est suivi de PAS ou TANTQUE.

Si oui, l'index $\downarrow^5$ est placé en PR devant POUR , celui-ci restant dans tous les cas dans PR , pour servir à l'élément suivant.

La compilation de l'élément continue suivant le fonctionnement général, les symboles PAS, JUSQUA et TANTQUE étant considérés comme des symboles ordinaires donnant lieu à des sous-phases spécialisées au point deux.

Au moment où PAS , ou TANTQUE sont détectés, l'état des piles devient donc :

PR : ... $\downarrow^2$ <u>POUR</u> $\downarrow^5$ < <u>PAS</u> ou <u>TANTQUE</u> >

PN : ... $C'(A_i)$ <u>POUR</u>

Pour un élément de la deuxième forme, la chaîne postfixée obtenue en PN est donc :

... $C'(A_i)$ <u>POUR</u> $C'(A'_i)$ <u>PAS</u> $C''(A_i)$ <u>JUSQUA</u>

Pour un élément de la troisième forme, elle est :

... $C'(A_i)$ <u>POUR</u> $C'(B_i)$ <u>TANTQUE</u>

Si l'élément est de la première forme, sa compilation est terminée et l'état de la pile PR est :

PR : <u>POUR</u>

Dans tous les cas, la détection de "," marque le début d'un nouvel élément. L'opérateur "," est placé en PN (phase P11 d'initialisation de cet élément) et le fonctionnement reprend en supprimant éventuellement $\downarrow^5$ de PR .

Si la fin de l'élément est marquée par FAIRE , il s'agit également de la fin de la liste de POUR , et l'index $\downarrow^2$, dont on vérifie la présence en PR est placé en PN pour marquer le début de la génération de la boucle proprement dite (P14) . POUR (et éventuellement $\downarrow^5$) est supprimé dans PR et FAIRE vient y remplacer $\downarrow^2$. L'état des piles au moment où l'on détecte FAIRE dans l'instruction suivante

<u>POUR</u> V := EL1 , EL2 ... , EL_p <u>FAIRE</u> I ;

est donc :

PN : $\downarrow^{2*}$ C'(V) $\downarrow^6$ C'(EL1), C'(EL2) ..., $C'(EL_p)$ $\downarrow^2$

PR : <u>FAIRE</u>

E9 - 3 Boucle de l'instruction POUR

Une liste de POUR a pour portée l'instruction qui suit le symbole FAIRE.

La fin de l'instruction peut être marquée par ";" , ou FIN. Dans tous les cas, le symbole FAIRE situé en PR "tombe" en PN sur le symbole marquant cette fin. Dans le cas où un index $\downarrow^3$ bloquerait ce fonctionnement, le sous-programme S assure l'exécution de la sous-phase du point deux relative à FAIRE (Cf. E8).

Ainsi l'instruction suivante :

POUR I := A + B , X PAS Y - Z JUSQUA W FAIRE T [I] := 0 ;

devient en notation postfixée :

I $\downarrow^6$ A B + POUR, X POUR Y Z - PAS W JUSQUA $\downarrow^2$ C'(T[I]) NO := FAIRE

Dans tous les cas, l'exécution de la sous-phase du point deux relative à FAIRE est suivie en H par la vérification du fait que toutes les mémoires de manoeuvre utilisées dans la boucle sont libérées (Cf. E3. 1).

E10 - NOMBRES PURS COMMENCANT PAR . OU $_{10}$

Lorsque . ou $_{10}$ sont détectés en E, on est en présence d'un nombre pur ne possédant pas de partie entière, ou réduit à une seule partie exposant (Cf. C6).

On a vu que les nombres de la forme .D, $.D_{10}$ P ou $.D_{10} \pm$ P étaient traités en fait par la branche opérandes. Le seul traitement effectué ici consiste à simuler une partie entière nulle.

Quand aux nombres de la forme $_{10}$P ou $_{10} \pm$ P , ils subissent un traitement analogue à celui des nombres ordinaires dans la branche opérandes.

Dans tous les cas, le nombre est rangé dans la table de nombres purs, son profil qualitatif est construit et placé en PN, et le contrôle est rendu au point H comme pour une fin d'opérande ordinaire, C étant positionnée sur l'unité syntaxique suivant la dernière du nombre.

E11 - TRAITEMENT DES CHAINES

Les chaînes alphabétiques au sens ALGOL sont traitées en entier dès que le symbole d'ouverture de chaîne ('en langage de référence, et '(' avec les conventions de perforation adoptées) est détecté par PC_2 - A .

Dans la production en notation postfixée, les symboles ' et ' sont placés en PN , la chaîne étant simplement recopiée en PN.

La phase de génération correspondante sera P24 , divisée en deux parties.

Notons que l'emploi de chaînes imbriquées ne trouvant aucune application sur les machines considérées, est interdit par le compilateur.

F - PARTICULARITES DE FONCTIONNEMENT

F1 - DETERMINATION DE LA HIERARCHIE DES DELIMITEURS

Nous avons vu que la notion de hiérarchie des délimiteurs du langage intervenait constamment dans le programme PC_2 - A . Nous verrons qu'elle apparait également dans PC_2 - G .

Il a donc fallu trouver un moyen simple, rapide et peu encombrant d'associer une hiérarchie à chaque délimiteur pour lequel c'est nécessaire.

Cette hiérarchie (Cf. Appendice I) est un nombre compris entre zéro et quinze. Sa représentation binaire est donc possible sur quatre positions : ces quatre positions étant disponibles en 17 - 20 du profil qualitatif des délimiteurs (Cf. première partie), c'est dans ce profil qu'est insérée la hiérarchie.

Pratiquement, cette insertion est réalisée automatiquement par PC_1 , au momentoù est pris le profil de l'opérateur en TABA ou TABB , et la hiérarchie de chaque délimiteur "suit" ce symbole de TABA / TABB en C , puis en PR et enfin en C' .

Ceci a pour avantages d'une part de minimiser le temps de recherche de la hiérarchie, d'autre part de limiter à une seule carte MAP dans PC_1 les modifications ou insertions à réaliser pour faire reconnaître par le compilateur un nouveau symbole de base.

F2 - DESCRIPTION DES SOUS-PROGRAMMES

Les fonctions des sous-programmes de production en notation postfixée ont déjà été mentionnées (Cf. E1). Nous nous bornerons à décrire ici les organigrammes correspondants.

F2 - 1 . DEFMEM (Cf. planche 11)

Utilisé également par PC_2 - G , ce sous-programme fournit le m d'une mémoire de manoeuvre libre et utilisable.

CIM, CIM1 et CIM2 sont automatiquement mis à jour quand cela est nécessaire.

F2 - 2 . C (Cf. planche 12)

Ce programme réalise, à l'aide de la sous-phase de génération SP12, la sauvegarde (index $\downarrow^{10}$) ou la mise à jour ($\downarrow^{10*}$) éventuelles du niveau NDYNL avant que l'on procède à la génération de l'appel d'une partie du programme P_M susceptible de modifier ce niveau (Cf. F4).

Si l'on n'est pas dans un bloc artificiel (paramètre effectif, position A dans la dernière mémoire de PB, ou expression d'aiguillage, position A en IAIG), une simple mise à jour du niveau suffit et l'index $\downarrow^{10*}$ est employé, sauf si DC est à un auquel cas aucune opération n'est effectuée.

La même mise à jour est faite dans le cas où l'on est dans un bloc artificiel mais où la sauvegarde a déjà été faite (positions A et B dans PB ou IAIG).

Dans le cas contraire, l'index $\downarrow^{10}$ est employé pour marquer le rangement de NDYNL dans une mémoire spécifique du bloc, définie par DEFMEM et dont le profil est emmagasiné en PMB. La position B est alors insérée dans PB ou IAIG suivant le type du bloc où l'on se trouve et, s'il s'agit d'une expression dans une liste d'aiguillage, celui-ci est spécifié "non simple" dans la suite du programme.

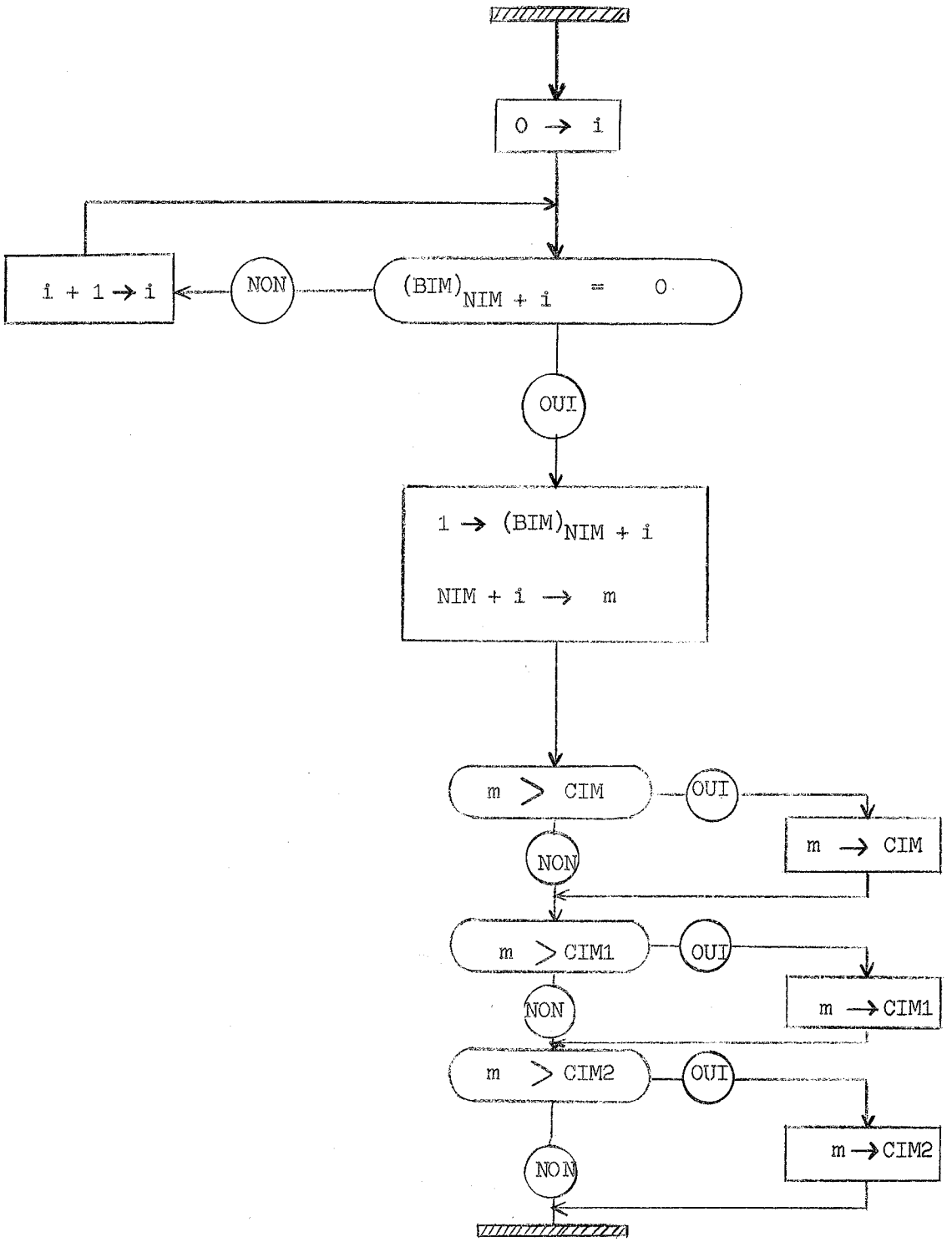
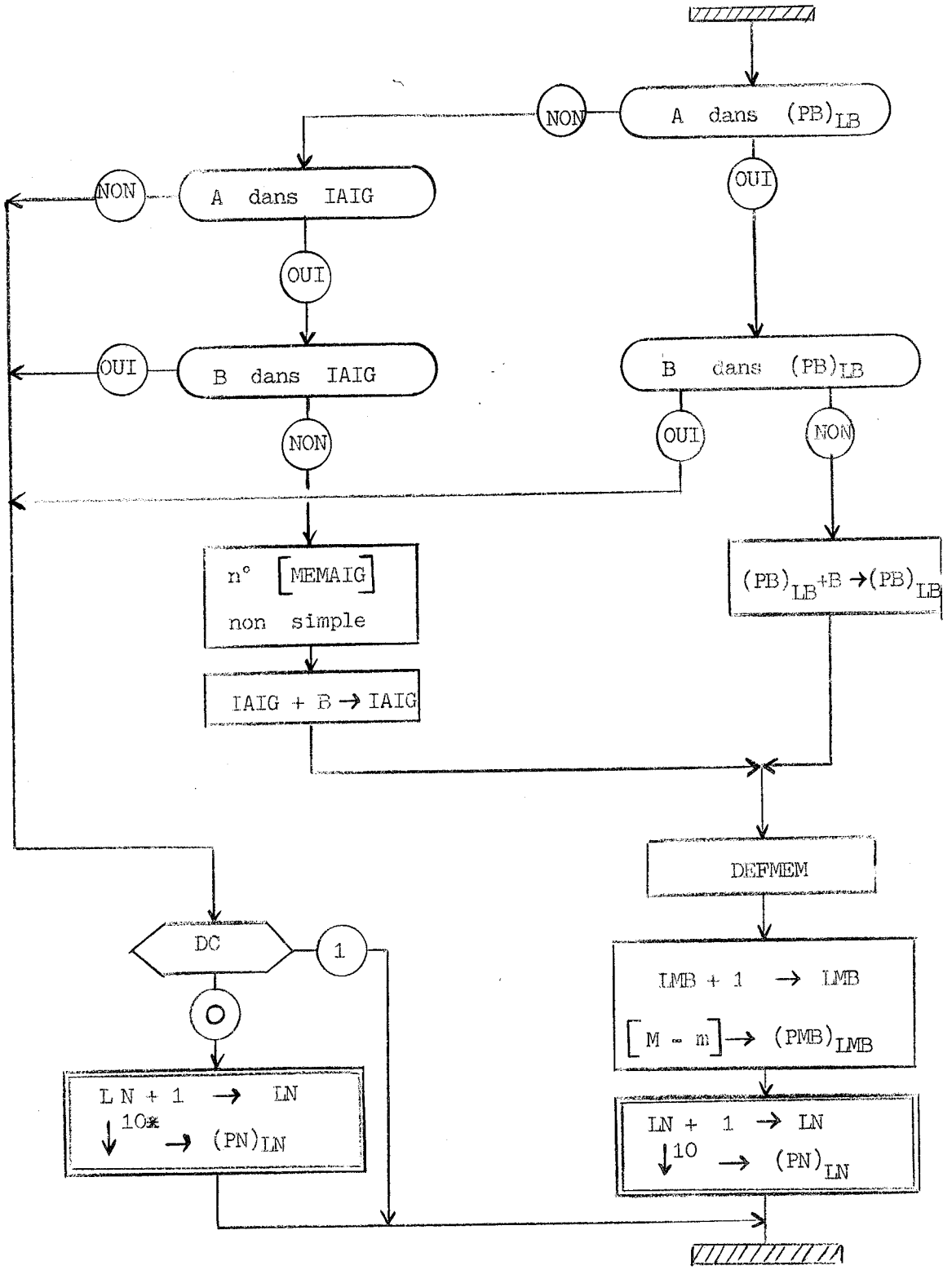


Planche 11

SOUS-PROGRAMME DEFMEM



SP12

Planche 12

SOUS-PROGRAMME C

F2 - 3 . B (planche 13).

L'organigramme explicite clairement la fonction du sous-programme B décrite en E1.

F2 - 4 . S (planche 14).

Si aucun $\downarrow^3$ ne bloque le fonctionnement de PR , le sous-programme est sans effet. Si $\downarrow^3$ est présent au sommet de PR mais n'est précédé d'aucun FAIRE ou SINON , le seul effet est de supprimer $\downarrow^3$ par recul de PR.

Si SINON apparait en PR , il est mis en PN pour symboliser la génération correspondante (P8). Ensuite ne peuvent apparaître en PR que SINON ou FAIRE (Cf. E8).

Si FAIRE apparait, la génération correspondante est marquée par FAIRE en PN , et le fonctionnement continue au point deux, d'autres symboles SI ou $\downarrow^3$ pouvant rester en PR.

Notons que si un autre symbole $\downarrow^3$ apparait, l'appel de S , effectué autant de fois qu'il est nécessaire quand on trouve ";" ou FIN , le fera disparaître.

F3 - FONCTIONNEMENT DU BLOC DE MEMOIRES DE MANOEUVRE

Nous pouvons maintenant donner l'algorithme complet du fonctionnement du bloc BIM et des niveaux CIM, CIM1 et CIM2. Nous ferons cette description en ALGOL , les divers identificateurs ayant la signification donnée au premier chapitre de cette partie.

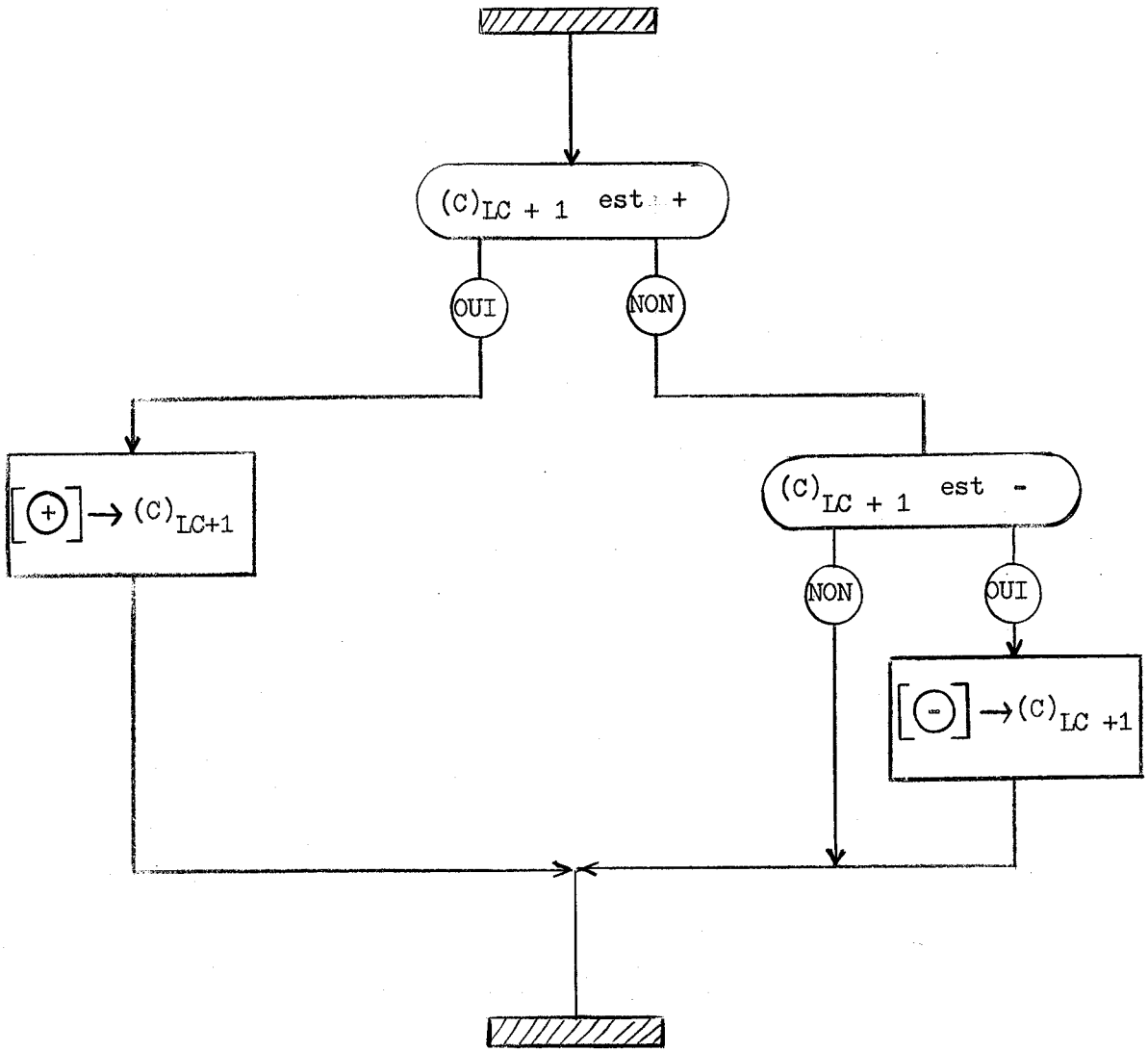


Planche 13

SOUS-PROGRAMME B

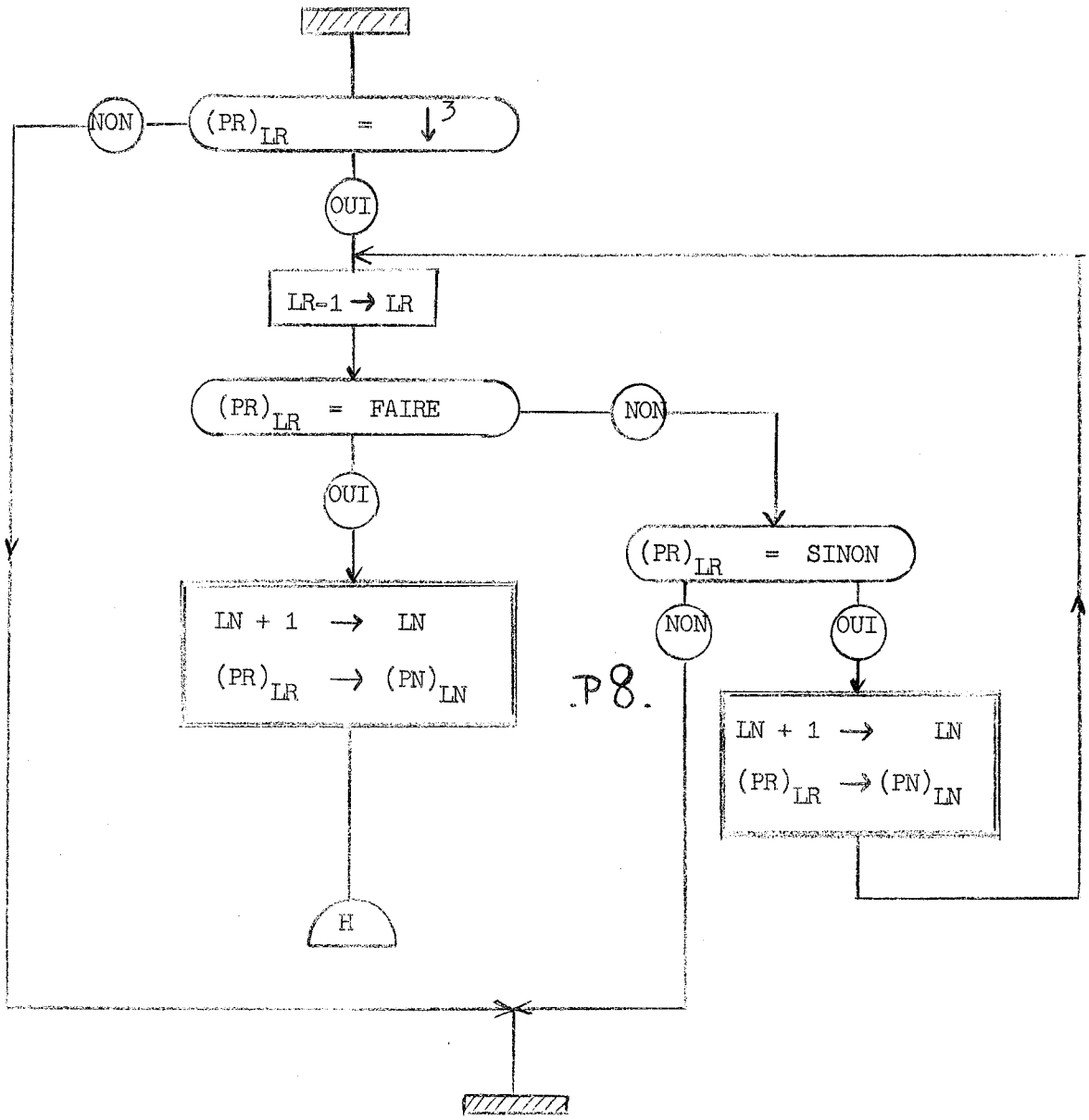


Planche 14

SOUS-PROGRAMME S

F3 - 1 Initialisation

DEBUT COMMENTAIRE Au début du programme, le bloc BIM est nul, NIM vaut un et les autres compteurs sont nuls ;

F3 - 2 DEFMEM

```
PROCEDURE  DEFMEM ;
   DEBUT  COMMENTAIRE  i étant une variable extérieure et EBIM
          l'encombrement maximum de BIM , le sous-programme DEFMEM
          s'écrit comme suit ;
          POUR     := NIM PAS 1 JUSQUA EBIM FAIRE
            SI BIM [ i ] = 0 ALORS ALLER A sortie ;
          ALLER A trop de mémoires de manoeuvre ;
sortie :  m := i ;
          BIM [i] := 1 ;
          SI m > CIM ALORS CIM := m ;
          SI m > CIM1 ALORS CIM1 := m ;
          SI m > CIM2 ALORS CIM2 := m
   FIN  de la procédure DEFMEM ;
```

F3 - 3 Début - fin

COMMENTAIRE Quand on trouve DEBUT de bloc, une mémoire spécifique est définie. Si c'est le premier bloc, cette mémoire correspond à M - 1

Dans tous les cas, NIM est en réserve en PNIM.

Quand on trouve FIN, la mémoire spécifique est libérée et NIM est restaurée par la valeur en PNIM. Une procédure booléenne BLOC est définie, qui vaut VRAI si DEBUT marque le début d'un bloc, et faux si c'est une instruction composée. La position D dans PB indique un bloc ;

```
début :   LB := LB + 1 ;
          SI ¬ BLOC ALORS DEBUT PB [LB] := 0 ;
                              ALLER A fin début FIN ;
```

```
                PB [LB]   := D  ;
                LMB  := LMB + 1  ;
                DEFMEM  ;
                PMB [LMB]  := m ;
                NIM  := NIM + 1  ;
fin  début  :   LNIM  := LNIM + 1  ;
                PNIM [LNIM]  := NIM  ;
                     |
                     |
                     |
fin  :          NIM  := PNIM [ LNIM ];
                SI  PB [LB]  =  0  ALORS  ALLER A  fin  fin  ;
                BIM [PMB [LMB]]   := 0  ;
                LMB  := LMB - 1  ;
                NIM  := NIM - 1  ;
fin fin :       LB  := LB - 1  ;
```

F3 - 4 Procédure et fin de procédure

COMMENTAIRE Quand on trouve PROCEDURE, le compteur CIM1 est mis à jour, et une mémoire spécifique est définie.

En fin de procédure, cette mémoire est libérée et NIM est remis à jour par la valeur maximum qu'il a atteinte. Cette valeur est automatiquement contenue dans CIM1, grâce au fonctionnement de DEFMEM ;

```
procédure  :    CIM1  := NIM - 1  ;
                DEFMEM  ;
                LMB  := LMB + 1  ;
                PMB [LMB]  := m  ;
                NIM  := NIM + 1  ;
                  |
                  |
                  |
```

```
                    |
                    |
                    |
fin procédure  :  NIM  :=  NIM - 1  ;
                  BIM [ PMB [ LMB ] ]  :=  0  ;
                  LMB :=    LMB  -  1  ;
                  NIM  :=  CIM1 + 1  ;
```

F3 - 5 Début et fin de liste de paramètre effectif

COMMENTAIRE Au début d'une liste de paramètres effectifs de procédure déclarée, le compteur CIM2 est mis à jour et emmagasiné en PCIM2. En fin de paramètre effectif, s'il est nécessaire d'avoir une mémoire de manoeuvre pour ranger le résultat (Cf. chapitre V), cette mémoire est prise à partir de la valeur contenue dans PCIM2, et cette valeur est remplacée par celle prise par CIM2 en effectuant DEFMEM. En fin de liste de paramètres, CIM2 est remis à jour si cela est nécessaire par la valeur contenue en PCIM2.

Cette méthode permet de rendre complètement indépendants vis à vis des mémoires de manoeuvres les sous-programmes de paramètres, tout en utilisant le moins de mémoires possible ;

```
Début de liste  :   CIM2  :=  NIM - 1  ;
                    LCIM2 :=  LCIM2 + 1  ;
                    PCIM2 [ LCIM2 ] :=  CIM2  ;
                       |
                       |
fin de paramètre :  SNIM  :=  NIM  ;
                    NIM   :=  PCIM2 [ LCIM2 ] + 1  ;
                    DEFMEM  ;
                    NIM   :=  SNIM  ;
                    PCIM2 [ LCIM2 ] :=  CIM2  ;
```

COMMENTAIRE SNIM est une mémoire de sauvegarde de NIM. La mémoire cherchée est M - m ;

```
                    |
                    |
                    |
```

```
                    ¦
                    ¦
fin de liste  :  LCIM2  :=  LCIM2 - 1  ;
                 SI  PCIM2 [ LCIM2 ] > CIM2  ALORS
                                 CIM2  :=  PCIM2 [ LCIM2 ] ;
```

F3 - 5 Début et fin d'aiguillage

COMMENTAIRE Au début d'une déclaration d'aiguillage, CIM1 est mis à jour. A la fin de la déclaration, NIM est remis à jour par la valeur maximum atteinte, contenue dans CIM1 ;

```
aiguillage  :     CIM1  :=  NIM - 1  ;
                    ¦
                    ¦
fin aiguillage :  NIM  :=  CIM1 + 1  ;
```

F3 - 6 Sous-programme C

```
PROCEDURE  C  (DC)  ;
   DEBUT   COMMENTAIRE Eventuellement, C peut définir une mémoire
           spécifique de bloc, on se retrouve dans le cas d'un début  ;
                 SI  PB [LB] < A ∧ IAIG < A ∨ PB [LB] ≥ A + B ∨ IAIG ≥ A + B
                   ALORS  DEBUT  SI  DC = 0  ALORS  SP12  ;
                                 ALLER A  sortie  FIN  ;
                 SI  PB [LB]  =  A  ALORS  PB [LB]  :=  PB [LB]  + B
                                    SINON  IAIG  :=  IAIG + B  ;
                 DEFMEM  ;
                 LMB  :=  LMB + 1  ;
                 PMB [LMB]  :=  m  ;
                 SP12  ;
sortie  :    FIN  ;
```

F3 - 7 Synthèse du fonctionnement

En plus des cas que nous venons d'énumérer, le bloc de mémoires de manoeuvre est mis en oeuvre dès qu'un point de génération demande une mémoire de manoeuvre pour ranger un résultat. Ces mémoires sont libérées au fur et à mesure que d'autres points de génération utilisent ces résultats.

Le fonctionnement général du bloc apparait alors clairement dans les exemples qui suivent.

F3 - 7 - 1 . Exemple 1

Soit le programme suivant, ne comprenant que des blocs et des instructions composées. Nous avons appelé D_i l'ensemble des déclarations suivant le ième DEBUT, et pI_i les suites d'instructions placées entre le ième DEBUT et le FIN correspondant. Nous avons porté à droite les nombres de mémoires de manoeuvre supposées nécessaires à chacun de ces ensembles :

```
DEBUT     D1  ;  }
          1I1 ;  }                                  4
          DEBUT D2  ;  }
                1I2 ;  }                            3
                DEBUT   I3   FIN  ;
                DEBUT   D4  ;                       2
                        I4   FIN  ;  }              5
                2I2   FIN  ;                        2
          2I1  ;                                    3
          DEBUT   D5  ;  }
                  I5   FIN  ;  }                    5
          3I1   FIN  ;                              2
```

Il est clair que les attributions de mémoires seront les suivantes :

DEBUT	n°1	:	mémoire spécifique	M - 1
			mémoires utilitaires	M - 2 à M - 5
DEBUT	n°2	:	mémoire spécifique	M - 2
			mémoires utilitaires	M - 3 à M - 5
DEBUT	n°3	:	mémoire spécifique	aucune (instruction composée)
			mémoires utilitaires	M - 3 et M - 4
DEBUT	n°4	:	mémoire spécifique	M - 3
			mémoires utilitaires	M - 4 à M - 8
			Au FIN correspondant,	M - 3 est libérée .
			Puis,	
	2I2	:	mémoires utilitaires	M - 3 et M - 4
			Au FIN correspondant au 2e DEBUT, M - 2 est libérée.	
			Puis,	
	2I1	:	mémoires utilitaires	M - 2 à M - 4
DEBUT	n°5	:	mémoire spécifique	M - 2
			mémoires utilitaires	M - 3 à M - 7
			Au FIN ,	M - 2 est libérée
			Puis,	
	3I1	:	mémoires utilitaires	M - 2 à M - 3

En fin de compilation, toutes les mémoires sont libérées. Les valeurs de CIM, CIM1 et CIM2 sont égales à 8 . Le bloc réservé dans P_M sera donc composé de 8 mémoires.

F3 - 7 - 2 . Exemple 2

Avec les mêmes conventions, étudions le programme suivant, où P_i est la ième procédure, Si sa partie spécification, J_i son corps, A_i le ième aiguillage, P_iE_p le p ème paramètre effectif d'un appel de la procédure P_i et P_iF_p le p ème paramètre formel de cette procédure :

```
DEBUT   D1 ; ---------------------------------------------- 2
        PROCEDURE P1 (P1F1 , P1F2 , P1F3 ) ; S1 ;
        J1 { DEBUT AIGUILLAGE A1 := ... ; ------------- 3
             I2            FIN ; ------------------------ 2
        1I1 ; ----------------------------------------------- 3
        DEBUT PROCEDURE P2 ; S2 ;
        2I1 { J2 { DEBUT D4 ; ------------------------------ 1
                   1I4 ; ---------------------------------- 2
                   2I4 { DEBUT PROCEDURE P3 ; S3 ;
                         J3 ; ------------------------------ 4
                   I5    FIN ; ---------------------------- 2
                   3I4   FIN ; ---------------------------- 2
                I3 FIN ; ------------------------------------ 3
        3I1 ; ---------------------------------------------- 5
        4I1 { DEBUT D6 ; ----------------------------------- 2
              I6 FIN ; ------------------------------------ 3
        5I1 ; ---------------------------------------------- 3
        P1 (P1E1 , P1E2 , P1E3 ) ; ------------------------ 4
                                                            2
        6I1 ---------------------------------------------- 4
FIN ;
```

On voit alors que l'attribution des mémoires dans P_M sera la suivante :

DEBUT	n°1	:	mémoire spécifique	M = 1		
			mémoires utilitaires	M = 2	et	M = 3
PROCEDURE	n°1	:	mémoire spécifique	M = 2		
DEBUT	n°2	:	mémoire spécifique	M = 3		
AIGUILLAGE	n°1	:	mémoires utilitaires	M = 4	à	M = 6

Ces mémoires utilitaires restant inutilisables sur toute la portée de A1, nous trouvons ensuite :

I2 : mémoires utilitaires M = 7 et M = 8

puis, les mémoires de toute la procédure P1 étant inutilisables, on trouve :

1I1 : mémoires utilitaires M = 9 à M = 11

DEBUT	n°3	:	mémoire spécifique	M - 9
PROCEDURE	n°2	:	mémoire spécifique	M - 10
DEBUT	n°4	:	mémoire spécifique	M - 11
			mémoires utilitaires	M - 12 et M - 13
DEBUT	n°5	:	mémoire spécifique	M - 12
PROCEDURE	n°3	:	mémoire spécifique	M - 13
			mémoires utilitaires	M - 14 à M - 17
			puis, les mémoires de P3 étant bloquées :	
	I5	:	mémoires utilitaires	M - 18 et M - 19

Quand FIN est rencontré, NIM reprend la valeur qu'il avait au DEBUT numéro 5, et on trouve :

3I4	:	mémoires utilitaires	M - 12 et M - 13

En recontrant un nouveau FIN, NIM reprend la valeur qu'il avait au DEBUT numéro 4, mais la perd aussitôt, le ";" qui suit marquant la fin de la procédure P2. C'est alors CIM1 + 1 qui est pris comme NIM, soit 20 (le maximum atteint dans la procédure étant 19), d'où :

I3	:	mémoires utilitaires	M - 20 à M - 22

Le FIN correspondant au DEBUT numéro 3 est rencontré, NIM reprend la valeur 9 puisque la portée de la procédure P2 est terminée :

	3I1	:	mémoires utilitaires	M - 9 à M - 13
DEBUT	n°6	:	mémoire spécifique	M - 9
			mémoires utilitaires	M - 10 à M - 12
	5I1	:	mémoires utilitaires	M - 9 à M - 11

On trouve alors l'appel de la procédure P1.

Nous supposerons que chacun des paramètres a besoin, en plus des mémoires utilitaires indiquées, d'une mémoire de résultat, libérée seulement en fin d'appel de procédure. Cette mémoire doit être différente pour chaque paramètre et en plus, ne doit pas être une des mémoires utilitaires employées par chacun des autres paramètres. (Ceci pour que le calcul d'un paramètre ne détruise pas le résultat d'un autre). Le fonctionnement ex-

posé plus haut donne :

P1E1	:	mémoires utilitaires	M - 9 à M - 11
		mémoire de résultat	M - 9
P1E2	:	mémoires utilitaires	M - 10 à M - 13
		mémoire de résultat	M - 12
P1E3	:	mémoires utilitaires	M - 10 et M - 11
		mémoire de résultat	M - 14

Enfin,

6I1	:	mémoires utilitaires	M - 9 à M - 12

On voit facilement que ce procédé, tout en rendant minimum le nombre de mémoires employées (en fait, une mémoire n'est demandée "qu'en cas de besoin"), autorise les imbrications les plus compliquées de paramètres effectifs et tient compte de toutes les portées de procédures et d'aiguillages. Remarquons qu'ici le nombre de mémoires réservées en fin de compilation sera vingt-deux, maximum atteint par CIM.

F4 - FONCTIONNEMENT DU NIVEAU DE MEMOIRE DYNAMIQUE

Nous avons vu (Cf. Chapitre I, B5) que le niveau courant de la partie DYNL de la mémoire dynamique devait être constamment contenu dans NDYNL au cours de l'exécution de P_M .

Pratiquement, la valeur initiale de NDYNL, fixée par l'encombrement de la partie système, de P_M et de ses sous-programmes, est placée, au début de l'exécution, en M - 1 qui est toujours la mémoire spécifique d'un bloc fictif extérieur à P_S .

Cette valeur est ensuite automatiquement mise à jour lors des déclarations de tableau appartenant aux différents blocs.

Pour résoudre le problème des "ALLER A" conduisant à des débuts de blocs contenant des déclarations de tableaux, tout en optimisant au maximum le nombre de mémoires utilisées (problèmes des bornes dynamiques de tableaux), la mémoire spécifique d'un bloc est mise à jour dans P_M quand on entre dans ce bloc, par la valeur contenue dans la mémoire spécifique du bloc immédiatement extérieur.

Considérons l'exemple simple suivant :

```
E0 : DEBUT  ENTIER  K1, K21, K22, K3 ; K1 := 50 ; K21 := 300 ; K22 := 100 ; K3 := 200 ;
E1 :        DEBUT  TABLEAU  T1 [1 : K1] ;
                    |
E2 :                DEBUT  TABLEAU  T21 [1 : K21] , T22 [1 : K22] ;
                    |
                    FIN ;
E3 :                DEBUT  TABLEAU  T3 [1 : K3] ;
                           |
                           SI K21 > K3  ALORS  DEBUT  K21 := 150 ;
                                                      ALLER A  E2  FIN ;
                           K21 := 300 ; ALLER A  E1 ;
                             |
                             etc ...
```

Soient $M - m_0$, $M - m_1$, $M - m_2$, $M - m_3$ les mémoires spécifiques des quatre blocs considérés.

Supposons que NDYNL parte de zéro.

Au départ de P_M , zéro est mis en $M - 1$, puis immédiatement, $M - m_0$ est mis à jour par $M - 1$.

Après les affectations de K1, K21, K22 et K3, on passe dans le second bloc, et $M - m_1$ est mise à jour à son tour par $M - m_0$, c'est à dire zéro.

Le tableau T1 est alors déclaré : il occupe cinquante mémoires de DYNL à partir de zéro. $M - m_1$ vaut maintenant cinquante.

En entrant dans le troisième bloc (E2), $M - m_2$ est portée à cinquante et les tableaux T21 et T22 occupent quatre cents mémoires au dessus des cinquante premières. $M - m_2$ vaut alors quatre cent cinquante.

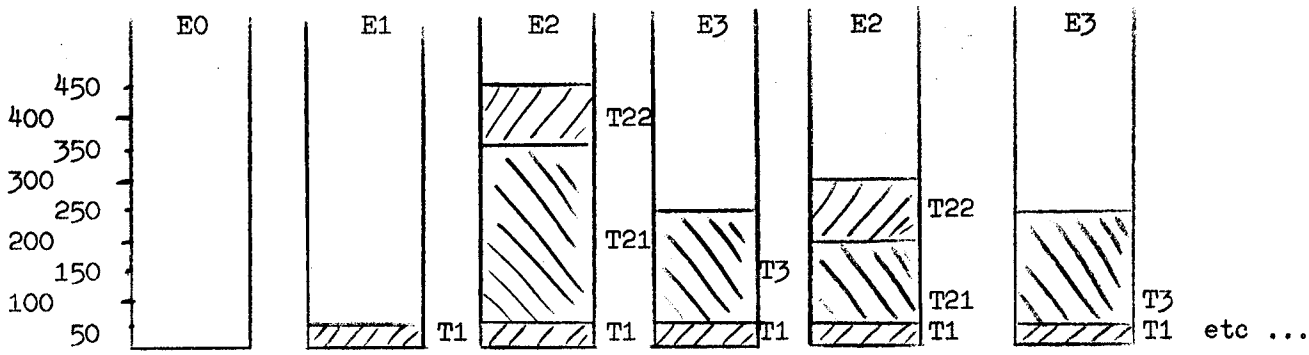
Le premier FIN rencontré marquant la fin de la portée de T21 et T22, le niveau est remis à jour dès le DEBUT qui suit. C'est la valeur de $M - m_1$ qui est utilisée : elle vaut cinquante, le tableau T3 viendra donc se superposer aux anciens tableaux T21 et T22, et $M - m_3$ est portée à deux cent cinquante.

On a alors $K21 > K3$, le ALLER A E2 s'effectue donc, et T3 perd sa signification. Le niveau est à nouveau remis à jour par $M - m_1$

en passant par E2, et $M - m_2$ est portée à trois cents après redéclaration de T21 et T22, qui se superposent à l'ancien T3.

Le même processus recommence, puis c'est le <u>ALLER A</u> E1 qui s'effectue, $K2_1$ reprend sa valeur initiale et $M - m_0$ réinitialise $M - m_1$ à zéro, etc ...

Les différents états de la mémoire dynamique se représentent donc de la façon suivante :



$M - m_0$	0	0	0	0	0	0
$M - m_1$		0,50	50	50	50	50
$M - m_2$			50,350,450	450	50,200,300	300
$M - m_3$				50,250	250	50,250

Nous avons vu à plusieurs reprises que dans ce fonctionnement la notion de bloc s'étendait aux corps de procédures, aux sous-programmes de paramètres effectifs et à ceux traduisant les expressions de liste d'aiguillage.

En effet, une procédure peut comporter des déclarations de tableaux locaux venant s'ajouter à ceux du programme principal. Ces tableaux ont une portée "statique" identique à celle de la procédure où ils sont déclarés, mais leur portée "dynamique" est , au maximum, l'étendue de la procédure elle-même.

Un appel de procédure est donc considéré à l'exécution de P_M , comme un <u>ALLER A</u> vers un bloc intérieur à celui où l'on se trouve.

On communiquera donc à ce bloc la valeur du niveau NDYNL, contenue dans la mémoire spécifique du bloc où l'on se trouve, et cette valeur sera rangée, en tête de la procédure, dans la mémoire spécifique définie pour celle-ci.

Les sous-programmes de paramètres effectifs et d'expressions d'aiguillages pouvant à leur tour faire appel à des procédures, des mémoires spécifiques peuvent être définies pour ces sous-programmes.

Il est à remarquer que grâce à l'emploi généralisé de la pile d'indicateurs PB et de la mémoire indicatrice IAIG , la réservation d'une mémoire spécifique pour ces sous-programmes n'est pas systématique (contrairement aux blocs ou aux procédures).

C'est seulement dans le cas où le sous-programme fait lui-même appel à une procédure ou à un autre tel sous-programme que la sauvegarde de NDYNL est faite (programme C ,opérateur $\downarrow^{10}$) dans une mémoire spécifique. A partir du deuxième appel situé dans le même sous-programme, c'est une mise à jour de NDYNL qui est effectuée (programme C , opérateur $\downarrow^{10*}$).

Cependant, si le sous-programme considéré traduit une expression conditionnelle, la réservation de la mémoire spécifique est faite systématiquement, puisque l'on ne sait pas, à la compilation, quelle branche sera déroulée à l'exécution (Cf. traitement de ALORS, programme C avec DC à un).

Enfin, plusieurs sous-programmes de paramètres effectifs, de types quelconques, pouvant être associés à un même paramètre formel, l'appel de celui-ci s'accompagnera systématiquement, comme pour un appel de procédure, de la mise à jour de NDYNL par la mémoire spécifique du bloc d'où se fait l'appel.

C'est de cette façon que la sauvegarde évoquée ci-dessus peut se faire.

Au contraire, dans le cas des aiguillages, un ensemble de sous-programmes et un seul est susceptible d'être déroulé à chaque ap-

pel. Il suffit de qualifier l'aiguillage d'une certaine manière à sa déclaration pour que ses appels s'accompagnent ou non de la mise à jour de NDYNL.

Ainsi, le programme C a pour rôle annexe de qualifier, dans une table, l'aiguillage en déclaration (dont le numéro est dans MEMAIG) de "non simple" si l'une au moins de ses expressions contient un appel de procédure, de paramètre ou d'un autre aiguillage non simple.

G - INSERTION DES POINTS DE GENERATION

Nous allons, dans les chapitres qui suivent, exposer le fonctionnement et les propriétés les plus intéressantes de la partie générative $PC_2 - G$ de PC_2 .

Nous insisterons plus particulièrement sur la façon dont cette partie - dont nous avons déjà évoqué la structure - s'insère aux endroits et aux moments désirés dans la partie analytique $PC_2 - A$.

Nous nous étendrons peu, au contraire, sur les séquences d'instructions générées, cette partie nous paraissant tout à fait liée aux machines utilisées. Dans tous les cas où nous décrirons l'une de ces séquences, il s'agira de celle adoptée sur la calculatrice 7044 (7040). Les séquences équivalentes sur 7090 (7094) s'en déduisent très aisément.

Enfin, nous faisons une dernière fois remarquer que la division en parties générative et analytique du compilateur, n'exclut pas la présence dans PC_2 de parties analytiques caractéristiques, en particulier en ce qui concerne la sémantique des programmes compilés.

Dans l'avenir (Cf. dernière partie) la partie $PC_2 - G$ est certainement appelée, pour des raisons d'optimisation des programmes générés, à comporter une grande proportion d'analyse syntaxique. Nous n'évoquerons que très rapidement cette possibilité dans le texte.

CHAPITRE III

DEBUT DE GENERATION

La phase complémentaire de début de génération PDG obtient le contrôle du fonctionnement après l'exécution par PC_1 de la procédure PASSAGE A PC_2 (Cf. première partie, chapitre II, G5).

Cette phase a trois fonctions principales :

1°) Initialisations diverses
2°) Analyse et traitement des options de génération
3°) Génération des macros - instructions utilisées dans P_M et de l'en-tête de P_M .

Enfin, cette phase donne le contrôle à la partie analytique PC_2 - A , au point E , la chaîne codée C étant positionnée sur sa première unité syntaxique.

A - INITIALISATION ET OPTIONS

A1 - INITIALISATIONS

Elles consistent à "ouvrir" le fichier de génération B_G [28], et à initialiser la zone tampon Z_G (caractères "espace").

En plus, un bloc d'indicateurs de fonctions standard (fonctions bibliothèques) est défini, qui est annulé entièrement ici. Ces indicateurs seront mis à un quand les fonctions respectives seront trouvées dans P_S. On verra (chapitre VI) que seules les fonctions dont les indicateurs sont à un en fin de compilation sont appelées en mémoire rapide au moment de l'exécution de P_M.

Puis, le compteur de nombres purs prend la valeur mise en réserve à la fin de PC_1.

Enfin, la mémoire $M - 1$ est réservée systématiquement et NIM est initialisé à deux.

A2 - OPTIONS DE GENERATION

On verra (Cf. troisième partie) que le déroulement du programme généré P_M dépend dans une certaine mesure de plusieurs options prises par l'utilisateur quand il fait compiler son programme.

Une option est considéré comme effective si l'indicateur I_S correspondant (Cf. chapitre I, A2) est à un (Cf. troisième partie). Des aiguillages sont alors positionnés dans $PC_2 - G$ qui entrainent la génération d'une séquence ou d'une autre dans tous les cas où l'option correspondante intervient. Ces cas seront étudiés

dans le texte au fur et à mesure de leur apparition.

Toute augmentation du nombre d'options, qui n'est pas limité en principe, doit se traduire par le test à cet endroit d'un nouvel I_S et le positionnement des aiguillages associés dans le cas où l'indicateur est à un.

B - GENERATION DE L'EN-TETE

Les premières instructions de P_M sont alors générées.

Elles sont constituées des macros-instructions [18, 27] utiles à P_M d'une part, des instructions d'initialisation du programme généré d'autre part.

B1 - MACROS-INSTRUCTIONS

Les macros utilisées dans P_M sont la traduction en MAP d'un certain nombre de fonctions standard ouvertes susceptibles d'être appelées dans P_S .

B1 - 1 CONVER

Cette séquence assure les conversions de variables du type entier vers le type réel.

La variable entière étant supposée en registre AC et inférieure en valeur absolue à 2^{27} , la macro employée est, compte tenu de la configuration des nombres réels en machine [17, 26] :

```
CONVER   MACRO
         ORA      AAA
         FAD      AAA
         ENDM     CONVER
```

où AAA représente un nombre égal à

$$+ 233\ 000\ 000\ 000_8$$

Ce nombre sera généré en fin de compilation (chapitre VI). La variable convertie apparait en registre AC .

L'appel de la macro CONVER est généré soit par le compilateur lui-même (changement de type dans une expression arithmétique) soit à la suite d'un appel par l'utilisateur de la fonction CONVER (X).

B1 - 2 CONVRE

Cette séquence assure le changement de type inverse de celui réalisé par la précédente et est également susceptible d'être appelée par le compilateur ou par l'utilisateur (fonction CONVRE (X)).

Sa génération comporte toutefois la particularité de dépendre de l'utilisateur par l'intermédiaire de l'option "VERIF" (Cf. troisième partie).

En effet, la conversion réel - entier n'est possible dans notre compilateur que pour un nombre inférieur à 2^{27} .

Si VERIF est demandée, cette condition est vérifiée et un libellé approprié est listé à l'exécution, qui indique l'endroit du programme où la conversion est impossible. Le sous-programme d'erreur INEX employé est inclus dans la bibliothèque du système.

Les séquences générées sont respectivement

```
CONVRE    MACRO
          LAS     BBB
          TSX     INEX,4
          TSX     INEX,4
          UFA     AAA
          RQL     9
          LGL     1
          ACL     EEE
          LGR     1
          ENDM    CONVRE
```

s'il y a VERIF, et

```
CONVRE   MACRO
         UFA      AAA
         RQL      9
         LGL      1
         ACL      EEE
         LGR      1
         ENDM     CONVRE
```

si VERIF n'est pas demandée.

Comme dans CONVER , la variable à convertir est supposée en AC quand on appelle cette macro-instruction, et AAA à la même signification.

BBB aura la valeur $234\ 000\ 000\ 000_8$
et EEE la valeur $312\ 000\ 000\ 000_8$ (Cf. chap. VI).

La variable convertie apparait également en AC .

B1 - 3 ENTIE

L'appel de la fonction ENTIER (X) dans P_S se traduit dans P_M par l'appel de la macro ENTIE générée ici.

La génération présente la même particularité que celle de CONVRE à propos de l'option VERIF.

Si VERIF n'est pas demandée, on génère

```
ENTIE    MACRO
         UFA      AAA
         ANA      CCC
         LLS      0
         TPL      *+6
         STO      A
         VLM      = 1B1,,34
         TZE      *+2
         CLS      = 1
         ADD      A
         ENDM     ENTIE
```

La valeur du résultat est un nombre du type entier apparaissant en registre AC . La mémoire A est celle contenant la première instruction de P_M (Cf. B2), qui peut être employée pour placer des résultats intermédiaires.

Rappelons que, par définition [1] , la fonction ENTIER (E) représente le plus grand nombre entier inférieur ou égal à la valeur de l'expression E .

B1 - 4 SIGN

L'appel de la fonction SIGN (X) , ou SIGNE (X) , se traduit dans P_M par l'appel de la macro-instruction SIGN générée ici.

La variable est supposée en AC .

Le résultat apparaît en AC . C'est un nombre entier qui vaut 1 si X est positif, 0 si X est nul et - 1 si X est négatif [1] .

La séquence générée est la suivante :

```
SIGN    MACRO
        TZE     * + 3
        LRS     35
        ORA     RB00
        ENDM    SIGN
```

où RB00 représente une mémoire contenant un (Cf. chapitre VI).

B2 - EN-TETE DE P_M

Les macros-instructions générées dans Z_G sont imprimées sur B_G , par l'intermédiaire du sous-programme SP16 .

Puis on construit les instructions d'initialisation de P_M . Elles consistent en l'appel d'un sous-programme d'ouverture OUVALG , dont on verra plus loin la decription (Cf. troisième partie, chapitre II), et qui a pour fonction principale de charger la valeur

initiale de NDYNL dans la mémoire M - 1 , qui lui est donnée en paramètre.

Là encore la génération dépend de la présence de l'option VERIF . Afin de garder trace dans les programmes interpréteurs de la présence de cette option, cette information est transmise à OUVALG de la manière suivante :

Si VERIF n'est pas demandée, on génère ici

```
          EXTERN    OUVALG
A         TSL       OUVALG
          PZE       M - 1
```

Si VERIF est demandée, on génère MZE au lieu de PZE. Dans tous les cas, cette séquence est imprimée par SP 16, et le contrôle est donné à E pour le début de l'analyse syntaxique.

CHAPITRE IV

POINTS ET AUTRES PHASES DE GENERATION

A - GENERALITES

Nous allons dans ce chapitre décrire les phases de génération P1 à P29 que nous avons évoquées dans les chapitres précédents.

Nous donnerons en particulier les séquences générées par chaque phase. Toutefois, cette description sera toujours sommaire et très liée à la machine pour laquelle ces séquences sont écrites (IBM 7044). La classification des phases adoptée dans ce chapitre nous permettra de décrire dans leur ensemble les générations de plusieurs parties d'un même programme qui forment un tout.

Cependant il pourra être utile de se reférer aux exemples de compilation donnés à l'Appendice IV pour avoir une vue générale de synthèse sur tout un programme généré, en partculier quand la présence d'options de génération apporte à celle-ci des variantes qui ne seront pas toutes décrites ici.

Enfin, il faut rappeler que chaque phase de génération est susceptible de faire appel à un ou plusieurs sous-programmes de génération ou de servitude. Ces sous-programmes seront décrits dans le chapitre qui suit, et nous supposerons connus ici leur existence et leur fonctionnement.

B - PHASE GENERATRICE D'OPERATIONS

Nous avons regroupé ici toutes les parties de la phase P2 de génération relatives aux opérateurs arithmétiques, logiques et de relation.

Rappelons que le fonctionnement du point deux de génération illustre parfaitement la méthode de compilation par production en notation postfixée [19, 20, 21, 23] , et qu'en particulier les sous-phases qui nous occupent ici, P2.S2 à P2.S5, sont appelées une fois et une fois seulement, quand l'opérateur qui leur est associé a sa hiérarchie supérieure à celui de l'opérateur qui suit en C.

B1 - GENERALITES

Les sous-phases P2.S2 à P2.S5 ont plusieurs points communs, dont le principal est qu'elles sont toutes relatives à un opérateur situé en PR au moment de leur appel, et qu'elles ont toutes pour fonction de générer une opération effective sur les variables du programme P_S . En cela, elles forment la charpente de la partie générative PC_2 - G .

Les sous-phases P2.S2 à P2.S4 trouvent en PN deux opérandes, sous la forme de deux profils qualitatifs de variables simples ou indicées, de nombres purs, de valeurs logiques ou de résultats d'expression. Dans le dernier cas, les profils peuvent spécifier un des registres AC/MQ ou une mémoire de manoeuvre M - k .

Dans tous les cas, nous supposerons que, grâce aux programmes de servitude SP5 - SP9 (Cf. chapitre V), la compatibilité des types a été vérifiée entre les deux opérandes, avec sortie d'un libellé d'erreur dans le cas contraire, et que, partout où cela était néces-

saire, les transferts de types adéquats ont été générés (appel des macros CONVER et CONVRE). En effet, les opérations arithmétiques et de relation générées ne peuvent porter que sur des opérandes de mêmes types.

Pratiquement, ce travail est effectué dans une séquence préliminaire au point deux, déroulée au moment où l'on effectue le test de hiérarchies.

La sous-phase P2.S5 est relative, elle, à un seul opérande en PN . Seule la compatibilité entre le type de cet opérande et l'opérateur en cause est vérifiée.

Dans tous les cas, la sortie des sous-phases considérées se fait avec un nouveau profil qualitatif en PN , qui est celui du résultat de l'opération générée.

Ce profil spécifiera en général un des registres AC ou MQ , et c'est l'indicateur correspondant IAC ou IMQ qui contiendra les caractéristiques effectives du résultat (position 2 de mémoire de manoeuvre et 10 - 11 - 12 de type).

L'index de la pile PR sera dans tous les cas décrémenté de un, avant que le contrôle ne soit rendu au point H pour effectuer à nouveau le test de hiérarchies sur l'opérateur précédent.

L'index de pile PN sera également diminué de un dans les sous-phases P2.S2 à P2.S4, où un résultat remplace deux opérandes.

B2 - SOUS-PHASE P2.S2 : ↑

L'avant-dernière et la dernière mémoires de PN contiennent respectivement les profils de la base et de l'exposant de l'opération élévation à la puissance.

On génère l'appel du sous-programme bibliothèque PUAL, auquel on donne en paramètres les adresses dans P_M des deux opérandes. Si l'un de ceux-ci était contenu en AC ou MQ , son rangement en mémoire de manoeuvre est effectué, et la mémoire immédiatement libérée.

Dans tous les cas, AC et MQ sont rangés s'ils contiennent un résultat (SP3 et SP4), et une protection du registre X4 est effectuée si cela est nécessaire (SP11).

On communique également à PUAL le type des opérandes, afin qu'il puisse effectuer correctement l'opération.

Dans tous les cas, le résultat est obtenu en AC, et est de type réel.

Ceci est caractérisé par le fait que la dernière mémoire de PN (après décrémentation de un) est mise à zéro, et que dans IAC, les positions 2 et 10 sont mises à un. IMQ, lui, est rendu égal à un pour spécifier que le registre MQ possède une valeur inconnue au retour de PUAL.

EXEMPLE A et B étant deux variables simples de types respectifs réel et entier, la séquence générée pour A ↑ B est :

```
TSX     PUAL,4
PZE     Q - 1,,4
PZE     Q - 2,,2
```

si Q - 1 et Q - 2 sont les symboles internes représentant A et B (Cf. SP14). La valeur 4 dans le décrément d'un paramètre correspond au type réel, la valeur 2 au type entier.

B3 - SOUS-PHASE P2.S3 : + , - , x , / , ÷ , < , ≤ , = , ≠ , ≥ , > .

Comme dans P2.S2, les deux opérandes, dont les profils sont en PN, doivent être de types réel ou entier. Ces types ont été rendus identiques par l'appel des sous-programmes de conversion adéquats (SP6 -SP9).

La sous-phase se divise en cinq sous-phases secondaires :

B3 - 1 Sous-phase P2.S3.S1 : +

Si les opérandes sont de type entier (cas de deux opérandes entiers au départ), l'opération à générer est ADD, le résultat est spécifié en AC , MQ étant inchangé.

Si les opérandes sont de type réel (deux opérandes réels au départ ou un réel et un entier), l'opération générée est FAD, le résultat est en AC , MQ étant cette fois inconnu (et donc rangé auparavant s'il contenait un résultat, cf. SP4).

Dans tous les cas, on teste la disposition des deux opérandes dans les registres ou en mémoire de manoeuvre. L'optimisation du nombre d'instructions d'appel - renvoi dans P_M , et du nombre de mémoires de manoeuvre nécessaires aux résultats intermédiaires, est réalisé ici (Cf. chapitre I, B4). Si aucun des opérandes n'est en AC, son appel est généré par CLA après rangement éventuel du contenu de AC (SP1).

Toute mémoire de manoeuvre qui contient un opérande de l'opération est libérée ici.

EXEMPLES

a) Soient A, B, C, D quatre variables entières dont les symboles internes sont Q - 1, Q - 2, Q - 3, Q - 4. L'expression (A + B) se traduira par

```
CLA     Q - 1
ADD     Q - 2
ADD     Q - 3
```

L'expression A + (B + C) donnera au contraire

```
CLA     Q - 2
ADD     Q - 3
ADD     Q - 1
```

On voit dans ce dernier cas que la deuxième opération ADD voit ses opérandes inversés de façon à profiter du fait que l'opérande (B + C) est en registre AC .

b) De même, si une conversion est nécessaire :
Soient X, Y, Z, T des variables réelles dont les symboles internes sont Q - 5, Q - 6, Q - 7, Q - 8 :
Soit M - 5 la première mémoire de manoeuvre utilisable, l'expression

A + B + X + (Y + C) + D

se traduira par

```
CLA      Q - 1
ADD      Q - 2
CONVER
FAD      Q - 5
STO      M - 5
CLA      Q - 3
CONVER
FAD      Q - 6
FAD      M - 5
STO      M - 5
CLA      Q - 4
CONVER
FAD      M - 5
```

Au contraire, l'expression identique, écrite

A + B + C + D + X + Y

se traduira par

```
CLA      Q - 1
ADD      Q - 2
ADD      Q - 3
ADD      Q - 4
CONVER
FAD      Q - 5
FAD      Q - 6
```

On voit ici que les nombres d'instructions et de mémoires de manoeuvre prennent les valeurs minimales qu'ils auraient dans un programme idéal P équivalent à P_S .

Remarquons que ces nombres étaient aussi minimaux sous la première forme, compte tenu de la maladresse d'écriture.

B3 - 2 Sous-phase P2.S3.S2 : -

Cette phase est tout à fait analogue à la précédente, Les opérations SUB et FSB remplaçant respectivement ADD et FAD.

Cependant, l'ordre des opérandes importe ici, la séquence générée étant différente si l'ordre naturel est inversé pour tenir compte de la présence éventuelle du second opérande en registre AC . Le signe de cet opérande est alors inversé, par une instruction CHS , et c'est une addition qui est générée.

EXEMPLES

a) Avec les mêmes variables, l'expression

A + B - C

se traduit par

```
CLA      Q - 1
ADD      Q - 2
SUB      Q - 3
```

b) Au contraire, l'expression

X + Y - C

où une conversion de C est nécessaire, se traduit par

```
CLA      Q - 5
FAD      Q - 6
STO      M - 5
CLA      Q - 3
CONVER
CHS
FAD      M - 5
```

B3 - 3 Sous-phase P2.S3.S3 : x

Les conditions de type sont les mêmes que dans les cas précédents.

Cependant, la nature des opérations générées (MPY pour le type entier et FMP pour réel) oblige à appeler l'un des opérandes en registre MQ, à moins qu'il ne s'y trouve. Le contenu éventuel de

MQ est rangé en même temps (SP2).

Dans tous les cas on génère le rangement du registre AC (SP3), et le résultat apparaît en MQ , AC inconnu, si le type des opérandes est entier, en AC , MQ inconnu, si le type est réel.

Là encore la disposition des opérandes dans les registres est testée, et l'ordre naturel éventuellement inversé de façon à minimiser les nombres d'instructions et de mémoires de manoeuvre.

EXEMPLES

a) Avec les mêmes variables que précédemment, soit l'expression

A + B x C x D

La génération correspondante sera

```
LDQ      Q - 2
MPY      Q - 3
MPY      Q - 4
ZAC
LLS      35
ADD      Q - 1
```

où l'on remarque le transfert MQ → AC du deuxième opérande de l'addition, pour éviter l'emploi d'une mémoire de manoeuvre.

b) Au contraire, l'expression

X + A x Y

se générera

```
CLA      Q - 1
CONVER
LRS      35
FMP      Q - 6
FAD      Q - 5
```

où l'on constate le transfert inverse AC → MQ après la conversion de A .

B3 - 4 Sous-phase P2.S3.S4 : / et ÷

La présence de deux opérateurs de division dans le langage ALGOL modifie sensiblement les conditions de types entre les opérandes.

Par définition [1] , l'opérateur / doit donner un résultat toujours réel, quels que soient les types de ses opérandes, tandis que ÷ doit donner un résultat entier, ses opérandes devant être entiers au départ.

Ces conditions ont pour conséquence :

1°) De générer systématiquement l'opération FDP pour l'opérateur / , les deux opérandes pouvant éventuellement être convertis d'entier vers réel,

2°) De générer DVP pour ÷ , toute configuration de type d'opérandes autre que entier - entier conduisant à un libellé d'erreur approprié. En plus, la génération correspondant à ÷ dépend de l'utilisateur par l'intermédiaire de l'option VERIF, la présence de celle-ci entrainant à l'exécution la vérification de la validité de l'opération (programme DINEX).

Ces conditions étant respectées, les problèmes de minimisation d'instructions et de mémoires sont les mêmes que précédemment, avec toutefois l'impossibilité ici d'inverser l'ordre des deux opérandes.

Le premier opérande doit être en MQ , avec AC nul, pour DVP (÷) et en AC pour FDP (/).

Le résultat est spécifié en MQ dans les deux cas, AC restant inconnu.

Les rangements et appels nécessaires sont générés pour respecter ces conditions (SP1 à SP4).

Dans le cas de / , les deux opérandes sont susceptibles d'être contenus dans deux mémoires de manoeuvre. Ces deux mémoires sont libérées à la fois.

EXEMPLES

a) Avec les mêmes notations, l'expression

(X + Y) / Z x T

se générera

```
CLA        Q - 5
FAD        Q - 6
FDP        Q - 7
FMP        Q - 8
```

tandis que l'expression

X / Z + (A + B) / T

donnera

```
CLA        Q - 5
FDP        Q - 7
CLA        Q - 1
ADD        Q - 2
STQ        M - 5
CONVER
FDP        Q - 8
ZAC
LLS        35
FAD        M - 5
```

où l'on note que le rangement du registre MQ se fait uniquement en cas de nécessité (ici à cause d'une conversion).

b) L'expression

B ÷ C x D

se générera

```
CLA        Q - 2
LRS        35
DVP        Q - 3
MPY        Q - 4
```

si VERIF n'est pas demandée, et

```
CLA        Q - 2
LRS        35
DVP        Q - 3
DCT
TSX        DINEX ,4
MPY        Q - 4
```

si VERIF est présente.

B3 - 5 Sous-phase P2.S3.S5 : relations

Les deux opérandes d'une relation doivent être de types réel ou entier. Le résultat est de type booléen et vaut VRAI ou FAUX.

Ces valeurs sont représentées en machine par les quantités respectives 1 et 0 .

L'opération générée est toujours CAS, les comparaisons se faisant algèbriquement. Le résultat reste en AC , où doit également être appelé un des opérandes.

On profite de la possibilité d'inverser les opérandes (en modifiant éventuellement les séquences générées) pour générer le minimum d'instructions d'appel - renvoi.

La valeur FAUX est obtenue en soustrayant de AC la valeur de l'opérande qui s'y trouve ou qui est appelé. Un rangement éventuel est donc nécessaire.

La valeur VRAI est obtenue en ajoutant un à l'AC avant cette soustraction.

Le registre MQ n'est pas modifié sauf s'il contient un des opérandes, qui est alors soit rangé soit transféré en AC.

EXEMPLES

a) La relation

B = C

se traduit par

```
CLA     Q - 2
CAS     Q - 3
TRA     * + 2
ADD     = 1
SUB     Q - 2
```

b) La relation

$X \geqslant A$

se traduit par

```
CLA          Q - 1
CONVER
STO          M - 5
CAS          Q - 5
TRA          * + 3
TRA          * + 2
ADD          = 1
SUB          M - 5
```

où on a été obligé de ranger A converti, pour pouvoir effectuer la soustraction. Notons qu'il n'y a aucun ennui à ce que cette soustraction soit toujours SUB.

B4 - SOUS-PHASE P2.S4 : OPERATEURS LOGIQUES

Cette sous-phase génère les instructions relatives aux symboles logiques à deux opérandes :

$$\wedge\ ,\ \vee\ ,\ \supset\ ,\ \equiv$$

L'ordre des opérandes peut être inversé, mais leur type doit toujours être booléen (tout autre type entrainant un libellé d'erreur).

Le résultat est booléen : il vaut VRAI ou FAUX, et est contenu en registre AC .

Le fonctionnement de cette sous-phase est tout à fait analogue à celui de P2.S3.S1.

Les opérations générées sont

ANA pour l'opérateur $\wedge$
ORA pour l'opérateur $\vee$

{ COM
ANA = 1 pour l'opérateur $\supset$
ORA

et { SUB
COM pour l'opérateur $\equiv$
ANA = 1

B5 - SOUS-PHASE P2.S5 : OPERATEURS UNITAIRES

Cette sous-phase se distingue des précédentes par le fait qu'un seul opérande est associé aux opérateurs traités qui sont

⊕ , ⊖ et ¬

Dans le cas de ⊕ , aucune instruction n'est générée et le retour est effectué immédiatement vers le test de hiérarchie, le profil de l'opérande restant sans changement en PN .

Dans le cas de ⊖ , l'opérande doit être appelé en AC (SP1) s'il ne l'est pas, et l'instruction générée est alors CLS. Si l'opérande est en AC au départ, on change son signe par CHS. Dans tous les cas on vérifie que le type est entier ou réel, et le résultat est spécifié en AC , du même type.

Enfin, l'opérateur ¬ se traduit, sur un opérande dont le type booléen est vérifié, par la génération des opérations suivantes :

```
COM
ANA   =  1 ,
```

après génération de l'appel de l'opérande en AC .

Le résultat est en AC , de type booléen.

Remarquons que, dans cette sous-phase, aucune libération de mémoire de manoeuvre n'est susceptible d'intervenir, aucun opérande n'étant rangé ici au moment de servir à son opérateur.

B6 - IMPRESSION DES SEQUENCES - EXEMPLE COMPLET

Dans tous les cas de génération que nous venons d'exposer, les séquences produites sont imprimées à chaque fin de sous-phase, c'est à dire transférées de Z_G aux zones tampons du fichier B_G .

Ces séquences ne sont donc pas susceptibles d'être modifiées par la suite de l'analyse syntaxique de P_S . Ce fait, lié à l'insertion de la phase deux dans PC_2 - A , illustre clairement le fonctionnement séquentiel du programme compilateur.

Donnons pour conclure un exemple de génération d'une expression booléenne contenant un opérateur de chacun des types étudiés :

Soit, avec les mêmes variables que précédemment, auxquelles on ajoute les variables booléennes U , V , W de symboles internes Q - 9, Q - 10, Q - 11 , l'expression suivante :

$+ B - C \neq D \times 2 / Z \vee A < T \equiv V \vee W \supset (\neg U \wedge X/Y > - 5 \supset \underline{\text{FAUX}})$

se traduira par :

```
CLA     Q - 2   }
SUB     Q - 3   }   B - C
STO     M - 5   }
LDQ     Q - 4   }
MPY     N + 1   }
ZAC             }
LIS     35      }   D x 2
CONVER          }
FDP     Q - 7   }   / Z
STQ     M - 6   }
CLA     M - 6   }
CAS     M - 5   }
TRA     * + 2   }   ≠
TRA     * + 2   }
ADD     = 1     }
SUB     M - 6
STO     M - 5
CLA     Q - 1   }
CONVER          }
STO     M - 6   }
CAS     Q - 8   }
TRA     * + 3   }   A<T
TRA     * + 2   }
ADD     = 1     }
SUB     M - 6   }
```

```
ORA     M - 5   }   ∨
STO     M - 5   }
CLA     Q - 10  }
ORA     Q - 11  }   V ∨ W
STO     M - 6   }
CLA     Q - 9   }
COM             }   ¬ U
ANA     = 1     }
STO     M - 7
CLA     Q - 5   }   X / Y
FDP     Q - 6   }
CLS     N + 2       - 5
STO     M - 8   }
STQ     M - 9   }
CAS     M - 9   }
TRA     A + 3   }   >
TRA     * + 2   }
ADD     = 1     }
SUB     M - 8   }
ANA     M - 7   }
COM             }
ANA     = 1     }   ⊃ FAUX
ORA     J BOO   }
COM
ANA     M - 6   }
COM             }
ANA     = 1     }   ⊃
SUB     M - 5   }
COM             }   ≡
ANA     = 1     }
```

Où N + 1 , N + 2 sont les symboles internes correspondant aux nombres purs numéro 1 (2) et 2 (5), et JB00 celui correspondant à FAUX .

Le résultat de l'expression est en AC .

Rappelons que cela signifie que la dernière mémoire de PN est nulle et que dans IAC, les positions 2 (mémoire de manoeuvre) et 12 (type booléen) sont à un.

C - SOUS - PHASE D'AFFECTATION : P2.S1

Les fonctions de la sous-phase P2.S1 sont les suivantes :

1°) Verifier la compatibilité des types des opérandes associés à l'opérateur := .

2°) Déterminer la longueur en opérandes des listes de parties gauches.

3°) Générer la séquence traduisant l'ensemble

< liste de parties gauches > := < expression >

en un nombre minimum d'instructions.

Rappelons que les opérateurs := contenus dans des listes de POUR ou d'aiguillage ont leur traitement analytique propre. Par contre, le := de liste de POUR passe par une phase analogue à P2.S1 pour ce qui est de la partie générative.

C1 - DETERMINATION DES TYPES

Soit l'instruction d'affectation la plus générale

V1 := V2 := .... := V_p := E , (1)

où V1 ... V_p sont des variables, indicées ou non, et E une expression, arithmétique ou booléenne.

La première fonction de P2.S1 est de vérifier successivement que les variables V1 ... V_p ont leur type compatible avec celui de E .

Pour cela, si E est booléen, il faut que V1 ... Vp soient booléennes.

Si E est réel ou entier, il faut que V1 ... Vp soient réelles ou entières. Dans ce cas, des conversions éventuelles sont générées, avec la convention suivante :

Soit T le type de E , Ti celui de la variable Vi, on doit générer successivement

Conversion de T vers Tp
Tp vers Tp - 1
⋮
T2 vers T1

Ces conversions peuvent être soit CONVER , soit CONVRE. Si deux types successifs sont les mêmes, il n'y a pas de conversion.

Par conséquent, une ou plusieurs variables Vi peuvent renfermer, après affectation, une valeur différente de celle de E, tout en étant de même type qu'elle.

EXEMPLE : Soit A entier, X et Y réels, et l'instruction :

```
X := A := Y := 1.7 ;
```

Cette instruction sera équivalente à la suite :

```
Y := 1.7 ;
A := 2   ;
X := 2.0 ;
```

et on voit que X n'a pas la même valeur que Y .

C2 - ANALYSE DES LISTES DE PARTIES GAUCHES

Nous avons vu que l'instruction d'affectation générale (1) se traduisait en production en notation postfixée par l'état des piles suivant :

$$\begin{cases} PR : \underbrace{:= \quad := \quad \ldots}_{P} \quad := \\ PN : C'(V1)\ C'(V2)\ \ldots\ C'(Vp)\ \ C'(E) \end{cases}$$

Ceci est réalisé grâce au fonctionnement du point GB .

Une seconde partie analytique consiste dans P2.S1 à reconnaître cette disposition et à en tenir compte dans le fonctionnement de la pile PN .

En effet, la sous-phase P2.S1 traite, en principe, deux opérandes, situés au sommet de PN. Suivant le fonctionnement général, l'index de PN devrait reculer de deux après P2.S1, puisqu'aucun résultat ne vient remplacer ces opérandes.

Cependant, pour assurer la compilation de toute la liste, il faut suspendre cette décrémentation jusqu'à ce que l'on ait trouvé le dernier := dans PR .

Ainsi, le premier := de PR aura pour opérandes C'(Vp) et C'(E) , et le deuxième C'(Vp - 1) et C'(Vp) . Pour diminuer le nombre d'instructions générées, on tiendra compte du fait que Vp, après la première affectation, est dans l'un des registres AC ou MQ , et, pratiquement, le ième := aura pour opérandes

C'(Vp - i + 1) et [AC] ou [MQ] .

Au dernier := , LN sera réellement décrémenté de deux.

C3 - GENERATION DE LA SEQUENCE

La génération des instructions correspondantes traduit les propriétés que nous venons de voir, en réalisant à la fois les conversions de type nécessaires (SP5 à SP9), et l'optimisation grâce à l'un des deux registres, du nombre d'instructions générées.

Si E est une variable , ou une expression dont le résultat est en MQ , c'est ce dernier qui est utilisé comme registre intermédiaire.

Si E est une expression dont le résultat est en AC (ou si une conversion est effectuée), c'est le registre AC qui est employé.

EXEMPLES

a) Soit l'instruction suivante, où les variables ont les mêmes caractéristiques que dans la partie précédente :

A := B := X := Y ;

On générera :

```
IDQ      Q - 6
STQ      Q - 5
ZAC
LLS      35
CONVRE
STO      Q - 2
STO      Q - 1
```

b) De même, l'instruction

A := B x C ;

se traduira par

```
LDQ      Q - 2
MPY      Q - 3
STQ      Q - 1
```

c) Et l'instruction

W := U ∧ V ;

par

```
CLA      Q - 9
ANA      Q - 10
STO      Q - 11
```

Remarquons que P2.S1 laisse dans l'indicateur du registre employé les caractéristiques de la dernière variable affectée si celle-ci est simple. L'autre registre est spécifié inconnu. Ainsi, une instruction d'appel est évitée si cette variable est aussitôt réutilisée dans l'instruction qui suit.

Ainsi, soit la séquence :

A := B + C ;

D := - A ;

On aura pour génération :

```
CLA     Q - 2
ADD     Q - 3
STO     Q - 1
CHS
STO     Q - 4
```

D - SOUS - PHASE D'ETIQUETAGE : P3

La fonction essentielle du point P3 de génération est d'associer l'opérateur : au dernier opérande situé dans PN pour étiqueter la prochaine instruction à générer.

Il y a donc construction (SP14) du symbole externe Ep correspondant à l'étiquette numéro p (numérotation NE de PC_1), et ce symbole est placé en Z_G , en tête de la prochaine ligne à remplir. Z_G n'est pas imprimé à la fin de la phase de sorte que la prochaine instruction générée, par n'importe quelle phase, sera étiquetée par Ep .

Si plusieurs étiquetages sont à faire sur la même instruction, seule une étiquette est affectée à l'instruction, les autres étant mises en réserve par SP14 pour être imprimées par SP16 sous forme de pseudo - instructions EQU [27] .

Ainsi l'instruction

E : F : A := B ;

où E et F sont supposées être les étiquettes numéros un et deux, et A et B les variables habituelles donnera la génération suivante :

```
E1      LDQ     Q - 2
E2      EQU     E1
        STQ     Q - 1
```

Remarquons que la génération des étiquettes créées par le compilateur (étiquettes internes) est identique à celle des étiquettes externes Ep . Seuls les indicatifs alphabétiques varient suivant le rôle de l'étiquette (R, V, B, J etc ...). Nous en verrons des exemples aux sous-phases correspondantes. Remarquons cependant qu'il peut y avoir génération d'une pseudo - instruction EQU entre une étiquette externe et une interne.

E - PHASES RELATIVES A SI, SINON, ALORS, ALLERA

Les expressions de la forme la plus générale, les instructions SI et conditionnelles, et les instructions ALLER A font intervenir les opérateurs SI, SINON, ALORS, ALLER A qui ont pour point commun de commander, dans leur (s) sous-phase (s) respective (s), la génération d'un étiquetage interne ou d'un transfert, ou des deux.

Les phases et sous-phases relatives à ces opérateurs sont

P2.S6 pour SI
P2.S7 , P8.S1 et P12 pour SINON
P13 pour ALORS
P2.S8 pour ALLER A

En plus, des sous-programmes de génération (S35 et S36) sont communs à ces phases et sont décrits dans le chapitre suivant.

Enfin, ces phases font appel aux sous-programmes de servitude déjà mentionnés : appel et rangement des opérandes (SP1 à SP4), vérification et conversions des types (SP5 à SP9), construction des symboles internes (SP14) et impression de Z_G (SP16).

E1 - FONCTIONNEMENT GENERAL

E1 - 1 Expressions arithmétiques et booléennes

Etant donnée l'expression la plus générale, de la forme

SI B ALORS A1 SINON A2

où B est une expression booléenne, A1 une expression simple et

A2 une expression quelconque, la forme de la séquence compilée doit être, compte tenu du fait que la programmation de la machine considérée est séquentielle, la suivante :

(1) - Instruction traduisant B

(2) - Instruction de test disant :

Si le résultat de B est faux, sauter les instructions suivantes jusqu'à R_i

(3) - Instructions traduisant A1

(4) - Instruction de transfert disant :

Sauter les instructions suivantes jusqu'à R_{i+1}

(5) R_i - Instructions traduisant A2

(6) R_{i+1} - ..... etc ...

Cinq phases sont donc à distinguer dans cette séquence.

Les phases numéros 1, 3 et 5 sont connues, puisqu'elles traduisent des expressions (si ces expressions sont à nouveau non simples, le fonctionnement est récursif et il y a toujours un niveau d'expression simple).

On voit que la phase numéro 2 n'est autre que P13 puisqu'elle s'insère au moment où ALORS est trouvé dans C . Les phases numéros 4 y compris l'étiquetage par R_i, et 6 sont alors les sous-phases du point deux relatives à SI et SINON, soient P2.S6 et P2.S7 .

En effet, le symbole SI contenu dans PR "tombe" sur SINON, et le symbole SINON qui remplace $\downarrow^3$ dans PR "tombe" sur le symbole suivant A2 , forcément de hiérarchie inférieure (ce peut être ; , FIN , virgule,] ,), : , PAS , JUSQUA, TANTQUE,FAIRE).

Le rôle et le fonctionnement de ces phases se déduisent alors de ce qui précède.

E1 - 2 Instructions conditionnelles

Le cas d'une instruction conditionnelle générale, telle que

SI B ALORS I1 SINON I2

où I1 et I2 sont deux instructions (la première inconditionnelle) diffère peu du précédent puisque seules les phases numéros 3 et 5, déjà connues, sont changées.

Par contre, des cas particuliers peuvent se présenter qui modifient la forme de la séquence générée :

1°) Si l'instruction I1 se termine par un ALLER A, la phase numéro 4 est inutile, et la sous-phase P2.S6 doit se réduire à l'étiquetage par R_i.

On verra que le cas encore plus particulier où I1 se réduit à un ALLER A est traité en modifiant les instructions générées par la phase numéro 2 (P13). La phase numéro 3 est alors elle-même supprimée.

2°) Si l'instruction conditionnelle est réduite à une instruction SI , ou à une proposition SI suivie d'une instruction POUR , la phase numéro 5 disparait et les étiquettes R_i et R_{i+1} sont relatives à la même instruction.

E1 - 3 Instructions ALLER A

L'instruction ALLER A portant sur une étiquette simple ne présente aucune difficulté de compilation (Cf. E5).

Par contre, si ALLER A porte sur une expression de désignation non réduite à une étiquette, plusieurs cas peuvent se présenter qui entrainent des générations très différentes.

1°) L'expression est un indicateur d'aiguillage, soit

ALLER A G [A]

où G est un aiguillage et A une expression arithmétique.

On verra que l'appel de l'aiguillage génère la séquence relative au ALLER A .

Cependant, si l'instruction ALLER A termine l'instruction I1 d'une instruction conditionnelle, la phase 4 de celle-ci redevient utile car l'indicateur d'aiguillage peut effectuer un retour en séquence. Ce cas est caractérisé par le fait que l'opérande est une étiquette de manoeuvre.

2°) L'expression est non simple, mais ne comporte pas d'indicateur d'aiguillage, soit

ALLER A SI B ALORS Ei SINON Ej

On voit que les phases P2.S6 et P2.S7 sont déroulées avant P2.S8, relative à ALLER A.Il faut donc prévoir dans ces sous-phases le cas où ALLER A précède $\downarrow^{3}$ en PR.

Si l'instruction ALLER A termine elle-même l'instruction I1 d'une instruction conditionnelle, la phase numéro 4 est inutile, et ce cas est caractérisé par un opérande en PN qui possède le type étiquette mais dont la partie adresse est nulle (opérande étiquette calculée, Cf. Appendice I).

3°) Les deux cas précédents peuvent être réunis en un seul, et ceci peut être trouvé à deux niveaux différents.

Par exemple :

ALLER A SI B ALORS G [A] SINON D

ou

ALLER A SI B ALORS (SI B' ALORS D1 SINON D2) SINON D

Dans ces cas, ce sont encore les phases relatives à SI et SINON qui sont susceptibles de générer les instructions traduisant en fait le ALLER A.

Ceci explique que la génération pour ALLER A soit pratiquement disséminée dans les générations pour SI et SINON que nous verrons plus loin.

E2 - GENERATION POUR ALORS : P13

La phase P13 relative au symbole ALORS a pour opérande de la dernière mémoire de PN.

On vérifie que cet opérande est de type booléen et, s'il n'est pas contenu dans le registre AC, il y est appelé (SP1).

L'opération générée pour effectuer le test voulu (Cf. E1 - 1) est TZE, zéro représentant la valeur logique FAUX .

La construction de l'étiquette R_i se fait alors à partir de la valeur d'un compteur d'étiquettes internes COET, initialement à zéro, et augmenté à chaque création d'une telle étiquette.

Le symbole créé est alors Rt , et le profil qualitatif en est placé en PE d'où il sera retiré quand il faudra étiqueter par Rt (phase P2.S6). PE fonctionne comme une pile, la gestion des étiquettes internes s'adaptant tout à fait à ce fonctionnement.

Aucune impression n'est effectuée à la fin de ce point P13, la séquence générée pouvant être modifiée par la suite (Cf. E5).

Enfin LN est diminué de un, l'opérande de ALORS devenant inutile, avant de retourner à l'analyse syntaxique.

E3 - GENERATION POUR SI : P2.S6

Quand SI tombe de PR sur un symbole de moindre hiérarchie (en général SINON), on a vu qu'il pouvait s'agir d'une fin d'instruction ou d'unefin d'expression. Dans le dernier cas, il peut s'agir d'une expression de désignation.

Nous trouvons donc trois fonctions à la phase P2.S6.

1°) Fin d'instruction

Ce cas est caractérisé par le fait que la pile PN est vide, c'est à dire que LN est nul.

a) Si l'instruction (I1 de l'instruction conditionnelle générale) était un ALLER A sur une étiquette simple, on verra qu'un aiguillage DP2.S5 est mis à un. P2.S5 se borne alors à spécifier les registres inconnus et à diminuer LR de un avant de retourner au test de hiérarchies.

b) Si la dernière instruction générée pour I1 est un TRA Ei, on a vu que la phase numéro 4 (Cf. E1) était inutile. Le seul rôle de P2.S6 est alors d'étiqueter l'instruction suivante par Rt, dont le profil occupe la dernière mémoire de PE. LE est alors diminué de un et le reste du traitement est comme a).

c) Dans les autres cas, le traitement consiste à générer TRA Rt' , t' étant la nouvelle valeur de COET. Le profil de Rt' est placé en PE à la place de celui de Rt qui vient étiqueter l'instruction suivante, comme en b).

2°) <u>Fin d'expression ordinaire</u>

Dans le cas d'une expression (A1 de l'expression générale du E1), PN contient un opérande de type entier, réel ou booléen.

Si cet opérande n'est pas en AC, on génère son appel (SP1), mais un profil non nul reste en PN, à la place de l'opérande. Ce profil contient le type (positions 10 - 11 - 12), qui sera comparé à celui du deuxième opérande (A2) dans la phase P2.S7 relative à SINON.

Puis, le traitement est comme en c) ci-dessus.

3°) <u>Fin d'expression de désignation</u>

Ce cas est caractérisé par la présence en PN d'un opérande du type étiquette.

Cet opérande peut être une étiquette simple (position 6, adresse non nulle), le résultat d'un indicateur d'aiguillage (positions 2 et 6), ou celui d'une expression de désignation non simple (position 6, adresse nulle, éventuellement position 2), ou encore une étiquette paramètre formel (positions 3).

Dans tous les cas, la génération pour ALLER A est effectuée (S35) et, au retour, la partie adresse du profil de l'opérande est annulée pour spécifier une expression de désignation non simple.

Enfin, le traitement continue comme dans le cas d'une fin d'instruction, Cf. 1°).

E4 - GENERATION POUR SINON : P2.S7, P8.S1,P12

La phase ordinaire de génération pour SINON se déroule au point deux (P2.S7) . Cependant des sous-phases analogues sont effectuées quand on appelle le sous-programme S (P8.S1) ou quand SINON est trouvé lors de l'analyse de ALORS (P12).

Pratiquement, cette génération est assurée par le sous-programme S36 qui tient compte de tous les cas d'opérandes susceptibles d'apparaître en PN.

La fonction essentielle de cette phase est d'assurer l'étiquetage de la suite des instructions par l'étiquette dont le profil est au sommet de PE .

Dans le cas où SINON tombe sur une fin d'expression de désignation, il y a génération comme pour ALLER A (S35, Cf. E3).

Dans le cas d'une expression, on vérifie la compatibilité des deux opérandes (A1 et A2) présents en PN , grâce à la position type laissée par la sous-phase P2.S6. Si les types sont entier-réel ou réel - entier, une conversion est générée qui transforme le deuxième opérande dans le type du premier, et l'expression conserve ce type. On notera la différence avec le cas d'une opération simple où le type est réel si les deux opérandes sont de types différents. Le fait que l'on ne puisse pas modifier après coup la génération du calcul du premier opérande explique cette différence.

EXEMPLE

Nous pouvons maintenant donner des exemples de génération pour des instructions et des expressions complètes.

a) Soit l'instruction

A := SI U ∧ V ALORS B + C SINON X ;

La génération correspondante est, si le compteur COET vaut 5 à cet endroit :

```
CLA     Q - 9     R5     CLA     Q - 5
ANA     Q - 10           CONVRE
TZE     R5        R6     STO     Q - 1
CLA     Q - 2
ADD     Q -
TRA     R6
```

b) L'instruction

SI U ∧ V ALORS A := B + C SINON A := X ;

se traduira par

```
CLA     Q - 9             ADD     Q - 3
ANA     Q - 10            STO     Q - 1
TZE     R5                TRA     R6
CLA     Q - 2     R5      CLA     Q - 5
                          CONVRE
                          STO     Q - 1
                R6        .... etc ...
```

E5 - GENERATION POUR ALLER A : P2.S8

La véritable phase de génération pour ALLER A s'insère au point deux (P2.S8), quand ALLER A "tombe" sur un symbole de hiérarchie inférieure. Pratiquement, le même sous-programme S35 est appelé que lors de la simulation par SI ou SINON d'un ALLER A dans PR (cas d'expressions de désignation dans PN).

La fonction essentielle de S34 est de générer une instruction de transfert TRA portant sur l'opérande dont le profil est en PN.

Cependant, si l'opérande est un paramètre formel, un indicateur d'aiguillage, ou le résultat d'une expression non simple, ce transfert est déjà effectué et P2.S8 n'a d'autre effet que de diminuer LR et LN de un. On trouvera lors de la description de S34 (Cf. chapitre suivant) des exemples de génération pour des instructions ALLER A .

F - GENERATION DES INSTRUCTIONS POUR

F1 - GENERALITES*

F1 - 1 Dissémination des diverses phases

Les symboles intervenant dans la génération des instructions POUR sont successivement :

POUR := PAS JUSQUA TANTQUE , FAIRE

Rappelons que := dans ce cas ne subit pas le traitement d'un := d'affectation. La phase qui lui est attachée est associée à l'opérateur fictif $\downarrow^{6}$. Elle s'insère dans PC_2 - A au moment où := est trouvé (P15, Cf. chapitre II, E9).

Le symbole POUR donne lieu à deux phases : l'une qui s'insère dans PC_2 - A quand POUR est trouvé dans C , et que l'on associe à l'opérateur fictif $\downarrow^{2*}$ (P23) ; l'autre qui entre dans le fonctionnement général du point deux, et qui correspond à la "chute" de POUR sur un opérateur de plus faible hiérarchie (PAS, TANTQUE, virgule ou FAIRE). On verra que cette phase (P2.S9) est analogue à la phase d'affectation P2.S1 .

Les symboles PAS, JUSQUA et TANTQUE donnent lieu à des phases faisant partie du point deux, réunies dans la sous-phase P2. S10.

Le symbole "," est traité quand on le trouve en C , il donne alors lieu à la phase P11 .

Enfin, le symbole FAIRE donne lieu, comme POUR, à deux phases distinctes : la première, P14, est déroulée quand FAIRE est trouvé dans C et est associée au symbole fictif $\downarrow^{2}$; la seconde (P2.S11) fait partie du point deux et est déroulée quand FAIRE "tombe" sur un symbole marquant la fin de l'instruction POUR

* Cette partie de programme a été écrite et mise au point par M. BERTHAUD.

(FIN ou ";"). Rappelons qu'une phase analogue est déroulée par le programme S , en P8.S2 .

F1 - 2 Principe de la génération

Etant donnée l'instruction POUR la plus générale suivante :

POUR V := EL1 , EL2 , ... ELp FAIRE I

et compte tenu des définitions relatives à cette forme [1, 4 . 6] , la séquence générée par le compilateur est constituée des phases suivantes :

(1)	Ri + 1	- Transfert en Ri
	Ri	- Sous-programme de V
(2)	$R_{EL1} \equiv$ Ri	- Affectation de V par EL1
(3)		- Appel de la boucle pour EL1
(4)	R_{EL2}	- Affectation de V par EL2
(5)		- Appel de la boucle pour EL2
⋮		
(2p)	R_{ELp}	- Affectation de V par ELp
(2p+1)		- Appel de la boucle pour ELp
(2p+2)		- Transfert en Rj
	B_k	- En-tête de boucle
(2p+3)		- Corps de boucle (fonctionnement général)
(2p+4)	R_{j+1}	- Fin de boucle
		- Retour de boucle
	R_j	- ... etc ...

La phase numéro 1 comprend la génération du sous-programme de variable contrôlée, qui est inexistant dans le cas général où celle-ci est une variable simple.

Dans le cas où un sous-programme existe (variable indicée ou paramètre formel par nom), son corps est généré suivant le fonctionnement général .

Les phases numéros 2, 4, ... 2p comprennent l'appel au sous-programme de variable contrôlée si celui-ci existe, la génération des diverses expressions entrant dans les éléments, qui suit le fonctionnement général, et les générations correspondant respectivement à PAS, JUSQUA et TANTQUE .

Enfin, la phase (2p+4) assure l'étiquetage de l'instruction qui suit l'instruction POUR , et à laquelle un transfert est exécuté après épuisement de la liste des éléments.

Ces phases utilisent, en plus des piles PR , PN et PE, les cinq indicateurs suivants :

ETVARC qui est non nul quand un sous-programme de variable contrôlée existe, et nul sinon ;

ETPAS et ETVAF qui contiennent les profils qualitatifs d'étiquettes identifiant les sous-programmes éventuels de calcul de la valeur du pas et de la valeur finale de la variable contrôlée ;

ETTEST qui contient le profil de l'étiquette à laquelle on doit se transférer quand un élément n'est pas épuisé après exécution de la boucle ;

ETBOU qui contient le profil qualitatif de B_k , étiquette d'appel de la boucle proprement dite.

Les quatre derniers indicateurs sont positifs quand ils contiennent un profil qualitatif, négatif sinon.

Enfin, ces phases font appel aux sous-programmes SP10 et S37 à S41, ainsi qu'aux sous-programmes habituels de génération d'appels - renvois et de conversion SP1 à SP9 .

Nous allons succinctement exposer le rôle et le fonctionnement de chacune des phases de génération évoquées en F1 - 1 dans l'établissement des phases 1 à 2p + 4 .

F2 - PARTIE GAUCHE D'UNE LISTE DE POUR

F2 - 1 Sous-programme de variable contrôlée

Si la variable contrôlée est une variable indicée ou un paramètre formel appelé par nom, il est nécessaire de définir un sous-programme de calcul de son adresse. Ce programme est appelé au début de chaque élément EL .

La séquence générée par P23 (opérateur $\downarrow^{2*}$) est alors la suivante :

```
                    TRA     R_COET
R_COET + 1  ***
```

et le profil de $R_{COET + 1}$ est placé en ETVARC, qui devient non nul, tandis que COET est augmenté de deux.

En P15 (opérateur $\downarrow^{6}$), ETVARC est testé.

S'il est non nul, on génère un transfert indirect à l'étiquette R_A dont il contient le profil, et on étiquette l'instruction suivante par l'étiquette dont le numéro précède de un celui de R_A . Enfin, on génère l'appel au sous-programme de variable controlée relatif au premier élément EL1, par

```
TSL    R_A
```

Ainsi, l'en-tête de la liste de POUR se traduit par :

```
        ┌  TRA   Ri
Ri+1    │  ***
        │  { sous-programme de calcul d'adresse
        │
        │  TRA * Ri+1
Ri      └→ TSL   Ri+1
```

Si la variable contrôlée est simple, ETVARC reste nul et aucun sous-programme n'est généré.

Dans tous les cas, le profil qualitatif de la variable contrôlée reste en PN (il y restera jusqu'à la fin de la génération de la liste de POUR) pour servir de premier opérande aux symboles POUR (dans P2.S9), PAS, JUSQUA ou TANTQUE.

F2 - 2 Initialisation de la variable contrôlée

Quelle que soit la forme de la variable contrôlée, la phase P15 place dans l'indicateur ETTEST, qui devient positif, le profil d'une étiquette à laquelle on génèrera un transfert si le premier élément est de la forme A TANTQUE B . En effet, la valeur initiale A doit être dans ce cas recalculée après chaque exécution de la boucle [1, 4, 6] .

Si ETVARC est non nul, cette étiquette n'est autre que Ri, sinon elle est créée par P15, en augmentant COET de un.

Pour les éléments autres que le premier, c'est la phase P11 qui place dans ETTEST le profil de l'étiquette générée en tête de l'élément. On verra que cette étiquette est automatiquement placée au sommet de PE par la génération de l'élément précédent.

La phase P2.S9 a ensuite pour rôle,après compilation de la première expression de l'élément, de générer l'affectation à la variable contrôlée (premier opérande en PN) de la valeur de cette expression (deuxième opérande en PN).

La génération est analogue à celle qu'effectuerait P2.S1.

En particulier, on vérifie dans cette phase la compatibilité des types entre la variable contrôlée et sa valeur initiale. Eventuellement, une conversion est générée (Cf. SP10), de façon à ramener le type de la valeur initiale à celui de la variable contrôlée.

Notons que le compilateur n'interdit pas l'emploi d'une variable contrôlée de booléen, à condition que les éléments soient des expressions booléennes (c'est là une extension au langage source).

Toutefois, un libellé d'erreur est imprimé si une telle variable est employée avec un élément de la forme Ai PAS A'i JUSQUA A"i.

Enfin, rappelons que, grâce au fonctionnement de la partie analytique, la phase d'initialisation est déroulée au début de chaque élément, le symbole POUR n'étant effacé dans PR qu'en atteignant la fin de la liste.

F3 - ELEMENTS DE POUR

F3 - 1 Eléments de la première forme

Quand l'élément est réduit à une expression, la phase numéro 2 (ou 4, ... 2p, Cf. F1 - 2) est formée de la seule phase d'initialisation.

Ce cas est caractérisé par le fait que "," ou "FAIRE" est trouvé en C à la suite de la première expression (c'est à dire dans P2.S9). On génère alors (phases numéros 3,5 ... 2p+1) l'appel à la boucle, par TSX B_k,4 (Cf. S37).

Le profil Bk est construit et rangé dans ETBOU lors du premier appel à S37 pour la liste de POUR étudiée.

Dans tous les autres éléments, le même profil est utilisé, ETBOU étant rendu positif.

Puis, un nouveau profil d'étiquette est créé (COET étant augmenté de un) et rangé en PE.

Si l'élément est suivi de "," , cette étiquette adressera la première instruction de l'élément suivant et, à ce titre, sera transférée de PE en ETTEST par la phase P11 . Cette phase générera aussi, éventuellement, l'appel au sous-programme de variable contrôlée.

Si l'élément est suivi de FAIRE (dernier élément de la liste), l'étiquette qui vient d'être créée adressera la première instruction qui suivra la boucle Bk. On génère alors, dans P2. S9, le transfert à cette étiquette (transfert en Rj de la phase (2p + 2)).

EXEMPLE

Soit l'instruction POUR suivante :

POUR I := K1 , K2 FAIRE ... ;

où I, K1 et K2 sont des variables entières simples dont les symboles internes sont Q - 1 , Q - 2, Q - 3.

En supposant que la prochaine étiquette interne à générer soit R5 et le prochain symbole de boucle B3, on obtient la génération suivante :

```
R5      CLA     Q - 2
        STO     Q - 1
        TSX     B3, 4
R6      CLA     Q - 3
        STO     Q - 1
        TSX     B3, 4
        TRA     R7
B3      { Boucle
R7      ..... etc ...
```

F3 - 2 Eléments de la deuxième forme

F3 - 2 - 1 Forme de la génération

Soit l'élément général de la seconde forme :

Ai PAS A'i JUSQUA A"i

L'initialisation de la variable contrôlée par l'expression Ai , est générée de la même façon que pour les éléments de la première forme (phases P15, P2.S9 et P11).

Par contre, la compilation des expressions A'i et A"i

présente la particularité de créer éventuellement deux nouveaux sous-programmes de calcul, l'un du pas, l'autre de la valeur finale.

Les étiquettes d'appel et les en-têtes de ces sous-programmes sont construits par les phases P2.S9, P2.S10.S1 et P2.S10.S2. Les symboles internes correspondant aux étiquettes d'appel sont du type Vi , et leur profil est placé, en cas d'existence, dans les indicateurs respectifs ETPAS et ETVAF .

Enfin, dans ce cas, le retour de la boucle doit se faire sur le test de la variable contrôlée et non sur le calcul de la valeur initiale comme dans les éléments de la troisième forme. C'est pourquoi le contenu de ETTEST est alors interprété comme une étiquette de la forme J , générée après les instructions d'initialisation, et à laquelle on génèrera un transfert après exécution de la boucle (Cf. S38).

Etant donnés l'étiquette Jn, les sous-programmes Vn et Vn + 1 de calcul du pas et de la valeur finale, et le sous-programme RA de variable contrôlée, la forme de la séquence générée pour la phase numéro 2 (4, ... , 2p) est alors

R_{EL}	TSL	R_A
	{ Initialisation	
	TRA	Ri
	{ Vn	
	{ Vn + 1	
Ri	TSL	Vn
J_{EL}	TSL	Vn + 1
	TEST si élément épuisé	

Si oui, transfert à Rj

Si non,

{ phase numéro 3 (TSX B_k,4)

TSL R_A

TSL Vn

{ Avance variable contrôlée

TRA J_{EL}

Rj ... etc ...

Les sous-programmes Vn et Vn+ 1 sont de la même forme que le sous-programme R_A , et appelés par "TSL". La numérotation V est indépendante de la numérotation R, de sorte que le fontionnement des indicateurs ETPAS et ETVAF est tout à fait autonome.

Chacun des sous-programmes Vn et Vn+1 existe dans tous les cas où l'opérande correspondant (deuxième ou troisième opérande dans PN) est autre chose qu'une variable simple ou un paramètre formel appelé par valeur.

Les mémoires de manoeuvre éventuellement utilisées pour ces opérandes sont libérées à la fin de la compilation des éléments, et les signes des indicateurs ETPAS et ETVAF sont remis à "-" .

Enfin, le transfert TRA Ri est supprimé si aucun des deux sous-programmes n'existe.

F3 - 2 - 2 Génération du test

Elle s'effectue dans la sous-phase P2.S10.S2 relative à JUSQUA .

Quand le test est satisfait, la séquence générée doit assurer le transfert à l'élément suivant. Le profil de l'étiquette Rj est placé à cet effet en PE , afin de jouer le rôle de R_{EL_i} si un autre élément suit, ou de Rj si FAIRE est

trouvé (phase numéro (2p + 2)).

Le test généré à la forme suivante dans le cas général :

- Appel variable contrôlée (premier opérande)
- Soustraction valeur finale (troisième opérande) (entière ou réelle suivant le type)
- Si résultat nul, continuation de la boucle
- Sinon, si le pas (deuxième opérande) est négatif, changer le signe du résultat
- Si résultat positif, transfert en Rj

<u>EXEMPLE</u> Soit l'instruction

<u>POUR</u> I := K1 <u>PAS</u> K2 <u>JUSQUA</u> K3 , ... , etc ...

Aucun sous-programme n'est nécessaire.

On obtient alors la génération suivante, en supposant que Q - 1 , Q - 2 , Q - 3 , Q - 4 sont les symboles internes respectifs des variables entières I, K1, K2, K3, que la numérotation K(B) est à 3 et COET (R) à 5 :

```
R5      CLA     Q - 2
        STO     Q - 1
J5      SUB     Q - 4
        TZE     * + 4
        PLT     Q - 3
        CHS
        TPL     R6
        TSX     B3,4
         |
         | (avance)
         |
R6      .... etc ...
```

Cependant, une génération plus simple est obtenue quand l'opérande de PAS est un nombre pur.

Celui-ci, en effet, ne peut être négatif, de sorte que PLT et CHS sont inutiles.

F3 - 2 - 3 Avance de la variable contrôlée

L'avance se fait par addition entière ou réelle suivant le type, après appel éventuel des sous-programmes de variable contrôlée et de pas (phase P2.S10.S2).

Puis un transfert est généré en J_{EL} .

EXEMPLE Soit l'instruction

POUR I := K1 PAS 1 JUSQUA K2 FAIRE ...

On obtient, si 1 est le nombre pur numéro 1 :

```
R5      CLA     Q - 2
        STO     Q - 1
J5      SUB     Q - 3
        TZE     * + 2
        TPL     R6
        TSX     B3, 4
        CLA     Q - 1
        ADD     N + 1
        STO     Q - 1
        TRA     J5
B3      { boucle
        {
R6      ... etc ...
```

F3 - 3 Eléments de la troisième forme

Les éléments de la forme

Ai TANTQUE Bi

donnent lieu à la génération de l'initialisation de la variable contrôlée, comme les autres éléments.

Puis, le résultat de Bi formant le deuxième opérande en PN du symbole TANTQUE, la phase P2.S10.S3 est déroulée, qui génère les phases numéros 2, 4 .... 2p de la séquence générale.

La génération a la forme suivante

- Appel du résultat de Bi en AC
- Test si ce résultat est faux

- Si oui, transfert à Rj
- Si non, appel de la boucle (phase numéro 3)
- Transfert en R_{EL}

Rj - ..... etc ...

Rj est l'étiquette créée puis placée en PE qui adresse l'élément suivant si le symbole qui suit est "," , et l'instruction suivant la boucle si le symbole qui suit est FAIRE .

R_{EL} est l'étiquette dont le profil est placé en ETTEST par l'initialisation (P15 ou P11).

<u>EXEMPLE</u> l'instruction

<u>POUR</u> I := K1 <u>TANTQUE</u> K2 = K3 <u>FAIRE</u> ..... ;

se traduit par

```
R5      CLA     Q - 2
        STO     Q - 1
        CLA     Q - 3
        CAS     Q - 4
        TRA     * + 2
        ADD     = 1
        SUB     Q - 3
        TZE     R6
        TSX     B3,4
        TRA     R5
B3
        Boucle
R6      .....etc ...
```

F4 - GENERATION DE LA BOUCLE

La numérotation BK est indépendante des numérotations R_{COET} et Vn . C'est le sous-programme S37 , qui gère cette numérotation, à l'aide de l'indicateur ETBOU.

F4 - 1 Début de boucle : P14

Quand FAIRE est trouvé en C , il y a génération pour $\downarrow^{2}$, la liste de POUR étant terminée (phase numéro 2p + 2).

Rappelons que PE contient alors à son sommet le profil d'une étiquette Rj qui doit adresser l'instruction suivant l'instruction POUR étudiée.

A la suite de Rj est alors placé le profil d'une nouvelle étiquette R_{j+1} , créée ici, et qui adressera, elle, la fin de la boucle de l'instruction POUR .

Celle-ci se termine obligatoirement par un des symboles ; ou FIN . On verra que la phase de génération P10 relative à ces symboles est susceptible de vider PE des étiquettes du type R inutilisées, par suite, par exemple, de la présence d'une instruction conditionnelle réduite à une instruction SI . Afin que les étiquettes Rj et R_{j+1} ne soient pas prises pour de telles étiquettes, elles sont "protégées" ici par un niveau nul en PE .

Puis, les indicateurs ETVARC et ETBOU sont réinitialisés, après qu'on ait généré

Bk SXA R_{j+1},4

Enfin, l'index IN est diminué de un pour "effacer" le profil de la variable contrôlée dans PN .

F4 - 2 Fin de boucle : P2.S11

La fin de la boucle (phase numéro 2p + 4) est compilée dans le fonctionnement normal du point deux, quand FAIRE "tombe" de PR .

L'étiquette R_{j+1} située au sommet de PE , après le niveau de protection nul, est générée ici, avec les deux instructions suivantes :

R_{j+1}	AXT	** , 4
	TRA	1 , 4

Enfin, l'étiquette Rj qui précédait R_{j+1} est générée. Elle adressera effectivement la séquence générée pour l'instruction qui suit l'instruction POUR terminée.

F5 - EXEMPLE COMPLET

Etant données les variables entières I, J, K, L, M, N de symboles internes Q - 1 à Q - 6 ,

la variable réelle A de symbole interne Q - 7,

les variables booléennes U et V , de symboles Q - 8 et Q - 9 ,

les numérotations NIM, R, B, V arrivées respectivement à 7, 5, 3, 2 ,

soient les instructions (sans valeur sémantique) :

POUR I := J PAS L JUSQUA M + N , A , 1 TANTQUE U ∧ V
FAIRE K := K + 1 ; I := 1 ;

La séquence générée est alors la suivante :

```
R5  CLA     Q - 2   }
    STO     Q - 1   }  I := J
    TRA     R6      }
V2  ***
    CLA     Q - 5   }
    ADD     Q - 6   }  M + N
    STO     M - 7   }
    TRA*    V2      }
R6  TSL     V2
J5  EQU     R6      }
    CLA     Q - 1   }
    SUB     M - 7   }
    TZE     * + 4   }  TEST
    PLT     Q - 4   }
    CHS             }
    TPL     R7      }
    TSX     B3, 4
    CLA     Q - 1   }
    ADD     Q - 4   }  Avance
    STO     Q - 1   }
    TRA     J5      }

R7  CLA     Q - 7   }
    CONVRE          }  A
    STO     Q - 1   }
    TSX     B3, 4   }
R8  CLA     N + 1   }
    STO     Q - 1   }
    CLA     Q - 8   }  1
    ANA     Q - 9   }  TANTQUE
    TZE     R9      }  U ∧ V
    TSX     B3, 4   }
    TRA     R8
B3  SXA     R10,4   }
    CLA     Q - 3   }  Boucle
    ADD     N + 1   }
    STO     Q - 3   }
R10 AXT     * *,4   }
    TRA     1,4
    CLA     N + 1   }  Instruction
    STO     Q - 1   }  suivante
```

DEBUTS FINS

G - DECLARATIONS DE TABLEAUX

POINT VIRGULE

G1 - GENERATION POUR DEBUT : P21

La phase de génération P21 a deux fonctions essentielles :

1°) Gestion de la pile PE

2°) Génération de la mise à jour de la mémoire spécifique dans le cas d'un bloc autre que le premier.

G1 - 1 Gestion de PE

La pile d'étiquettes internes du type Ri est susceptible d'être vidée (Cf. P10) des profils qu'elle contient lors de la détection d'un symbole ";" , les étiquettes correspondantes étant immédiatement générées.

Certaines configurations d'instructions, telles que la suivante :

SI B ALORS DEBUT I1 ; I2 ; .... FIN SINON ..... etc ...

entrainent cependant une génération erronée d'étiquettes si ce vidage est fait sans précaution.

C'est pourquoi l'on définit dans PE des "niveaux" qui arrêtent le vidage, protégeant ainsi les étiquettes précédentes.

L'utilisation d'un tel niveau a été vue au chapitre précédent.

La première fonction de P21 est de créer ici, systématiquement, un de ces niveaux dans PN .

Bien entendu, ce niveau est effacé en recontrant le symbole FIN correspondant.

G1 - 2 Mémoires spécifiques

M - 1 , mémoire spécifique du bloc le plus extérieur du programme est, on l'a vu, automatiquement chargée par le programme d'ouverture OUVALG.

Par contre, les premières instructions générées à l'entrée dans un autre bloc doivent être pour mettre à jour la mémoire spécifique du nouveau bloc à partir de celle du précédent C'est la seconde fonction de P21.

Ainsi, soit la configuration

DEBUT REEL A, B, C ;

E : DEBUT REEL D, E, F .... etc ...

L'en-tête de la génération sera alors :

```
A       TSL     OUVALG
        PZE     M - 1
E1      CLA     M - 1
        STO     M - 2
```

et tout ALLER A en E fera réinitialiser automatiquement NDYNL par la première déclaration de tableau à venir (Cf. chapitre II, F4).

G2 - DECLARATIONS DES TABLEAUX : P17 ET P22

Les phases P17 (opérateur $\downarrow^{4}$) et P22 ($\downarrow^{4*}$) sont déroulées en début et fin des déclarations de tableaux persistants.

Compte tenu des conventions adoptées dans le compilateur, le programme P_M doit exécuter une fois seulement les instructions relatives à ce type de déclarations.

Pour cela, la phase P17 génère, en tête de la déclaration, la séquence suivante :

```
CLA     Ri
CLA     DDD
STD     *-2
```

Ri est construite en incrémentant COET de un, rangée en PE et protégée par un niveau nul.

DDD est un nombre, généré en fin de compilation, et tel que la séquence ci-dessus transforme, au premier passage, l'instruction CLA Ri en une instruction de transfert TRA Ri .

La phase P22 étiquette alors l'instruction suivant la déclaration par Ri , en faisant reculer convenablement l'index de PE , et le fonctionnement désiré est obtenu.

G3 - POINT-VIRGULE ET FIN : P10

Des étiquettes peuvent être en attente de génération dans PE quand on trouve une fin d'instruction, par ";" ou "FIN" .

C'est le cas en particulier quand une instruction conditionnelle se réduit à une instruction SI .

Le rôle de P10 est alors de vider PE en générant au fur et à mesure les étiquettes correspondant aux profils trouvés.

Le fonctionnement de P10 est arrêté au premier niveau nul trouvé dans PE à partir du sommet, conformément au principe de gestion de PE (Cf. G1 - 1).

Enfin, le niveau nul est lui-même effacé si c'est le symbole FIN qui a entrainé l'exécution de P10 .

H - DECLARATIONS DE PROCEDURES

H1 - GENERALITES ET FONCTIONS ESSENTIELLES

Les phases de génération relatives aux déclarations de procédures sont respectivement :

- P20 , commune aux déclarations d'aiguillages, et
  P18 , pour la liste de paramètres formels
- P4 pour la partie VALEUR de l'en-tête
- P19 pour les corps de procédure en code machine
- P5 pour la fin de la déclaration

Les déclarations de procédures doivent donner naissance à des sous-programmes fermés, insérés dans le programme principal à l'endroit de la déclaration.

Cette insertion est réalisée par P20 .

Les sous-programmes ainsi créés doivent correspondre à une séquence d'appel, en paramètres de laquelle sont donnés (Cf.I) :

- La valeur de NDYNL au moment de l'appel
- Le type et le genre de chaque paramètre effectif
- L'adresse des paramètres effectifs du genre tableau, procédure ou aiguillage.
- L'adresse d'un sous-programme de calcul d'adresse pour tous les autres paramètres.

C'est la fonction de P18 de générer les premières instructions de la procédure, avec l'étiquette de celle-ci, et l'appel à un sous-programme interpréteur chargé

- 1°) de mettre à jour la mémoire spécifique de la procédure,
- 2°) de vérifier la conformité des types et genres entre paramètres effectifs et formels,
- 3°) de charger, dans des mémoires réservées à cet effet, les adresses des sous-programmes de paramètres effectifs spécifiées à l'appel,
- 4°) En cas de récursivité, d'empiler dans la mémoire dynamique, les valeurs ou adresses des variables locales et des paramètres de la procédure.

Enfin, la phase P5 de fin de procédure doit assurer le retour de la procédure à la suite de la séquence d'appel, avec, éventuellement, la valeur de l'indicateur de fonction correspondant en registre AC .

C'est également par P5 que sera généré d'une part l'appel à un sous-programme interpréteur de désempilage des variables en cas de récursivité, d'autre part un certain nombre de mémoires destinées à contenir, à l'exécution, les valeurs ou les adresses des paramètres de l'appel (suivant qu'ils sont appelés par valeur ou par nom), un compteur de récursivité, et les adresses de toutes les variables et mémoires de manoeuvre destinées à être empilées en cas de récursivité.

H2 - FORME GENERALE DE LA GENERATION

Etant donnée la déclaration de procédure la plus générale suivante :

(type) <u>PROCEDURE</u> PROC (PF1, PF2 .... PFp) ;

<u>VALEUR</u> PF_{V1} , PF_{V2} ... PF_{Vm} ; $(m \leqslant p)$

{ Spécifications

{ Corps

;

La génération a la forme suivante :

(1) - Transfert en Ri

(2) - Vecteur transfert des paramètres

(3) Pn - Appel du sous-programme d'entrée en procédure

(4) - Pour chaque PF_{Vi} :

- Appel du sous-programme correspondant
- Rangement de la valeur en $F_n + vi$

(5) - Corps :

- Fonctionnement général ou
- Code machine

(6) - Appel du sous-programme de sortie

(7) { Fn
- Réservation d'un compteur de récursivité
- Réservation de p + 1 mémoires de paramètres
- Indication des variables locales et mémoires de manoeuvre

Ri - .... etc ...

H2 - 1 Vecteur transfert des paramètres

Pratiquement, les mémoires où sont conservées les adresses des sous-programmes de paramètres sont des instructions de transfert indirect (TRA*) chargées par le programme d'entrée.

Il y a p de ces mémoires, la mémoire P_{n-i} étant associée au ième paramètre .

H2 - 2 Appel du programme d'entrée

Le sous-programme est ALPRR ou ALPRO suivant que l'option RECUR (récursivité) est présente ou non (Cf. troisième partie, chapitre II).

Dans tous les cas, ce programme a pour paramètres la mémoire spécifique de la procédure, et les caractéristiques (type et genre) des paramètres formels.

ALPRO assure la vérification de la conformité de type et de genre entre paramètres effectifs et paramètres formels, et charge, dans les mémoires réservées après Fn les adresses fournies à l'appel pour les paramètres du genre tableau, procédure ou aiguillage, et dans les mémoires du vecteur transfert les adresses des sous-programmes des autres paramètres.

ALPRR assure en plus le recopiage en mémoire dynamique des paramètres, du vecteur de transfert, et de toutes les variables locales et mémoires de manoeuvre de la procédure.

Dans tous les cas, la mémoire spécifique est mise à jour par NDYNL et un compteur de récursivité défini pour la procédure, nul au départ, est décrémenté de un.

H2 - 3 Partie valeur

La partie valeur peut ne porter sur des paramètres formels du genre variable simple ou tableau.

Dans le cas d'une variable simple, on fait appel au sous-programme de calcul d'adresse du paramètre, et la valeur de celui-ci est rangée dans la mémoire réservée après Fn .

Dans le cas d'un tableau, c'est un programme interpréteur (ALVTA, Cf. troisième partie, chapitre II) qui définit dans la mémoire dynamique DYNL un tableau local équivalent au tableau effectif. La mémoire spécifique de la procédure, donnée en paramètre à ALVTA, est remise à jour. Notons que le fonctionnement de ALVTA n'autorise pas l'emploi de tableaux appelés par valeur dans un appel récursif de procédure.

H2 - 4 Corps de procédure

Dans le cas d'un corps en ALGOL, il suit le fonctionnement général.

Dans le cas d'un corps en code machine, il est entièrement compilé par la phase P19, les instructions MAP étant simplement insérées sur B_G après les instructions déjà générées.

H2 - 5 Appel du programme de sortie

Dans le cas où l'option RECUR est présente, on génère en fin de procédure l'appel au sous-programme de sortie ALDST.

Ce programme assure le désempilage des mémoires locales et des paramètres placés dans DYNL par ALPRR .

En plus, il fait décroître le compteur de récursivité de un, appelle éventuellement la valeur de l'indicateur de fonction Fn en registre AC , et assure le retour à la séquence d'appel.

Dans le cas où RECUR n'est pas demandée, on génère directement une remise à zéro du compteur de récursivité, l'appel de Fn en AC et l'instruction de retour.

H2 - 6 Réservations de fin de procédure

Une mémoire est réservée qui contient les valeurs du compteur de récursivité de la procédure.

Puis, étiqueté Fn, est réservé un bloc de p + 1 mémoires, p étant le nombre de paramètres formels.

La première de ces mémoires est appelée à contenir éventuellement la valeur de l'indicateur de fonction PROC . Dans le corps de procédure, l'occurrence de l'identificateur PROC en partie gauche sera remplacée par le symbole Fn. Dans toutes les autres positions, l'occurrence de PROC signifie l'appel de Pn.

Les p mémoires qui suivent Fn sont appelées à contenir les adresses ou les valeurs des paramètres effectifs de l'appel. Au k ème paramètre est ainsi associée la mémoire Fn + k .

Enfin, les mémoires contenant les adresses des variables locales et mémoires de manoeuvre sont générées à l'aide des valeurs conservées dans la pile PQTM .

H3 - DESCRIPTION DES PHASES DE GENERATION

H3 - 1 Phase P20 (opérateur $\downarrow^1$)

Cette phase génère un transfert à la suite du groupe de sous-programmes formé par la suite éventuelle de plusieurs déclarations d'aiguillages ou de procédures.

Dans ce cas, un aiguillage dans cette phase permet d'éviter la génération d'une chaîne de transferts d'un sous-programme à l'autre. Cet aiguillage est positionné à un par la phase P5 de fin de déclaration.

Si un transfert est généré (aiguillage à zéro), une

étiquette Ri est construite, placée en PE et protégée par un niveau nul. L'instruction générée est

TRA Ri

et Ri sera elle-même générée par P5 devant l'instruction qui suit la (les) déclaration (s).

H3 - 2 Phase P18

Cette phase se divise en deux parties :

- La première se déroule autant de fois qu'il y a de paramètres formels dans la liste, et est associée à l'opérateur fictif $\downarrow^{1*}$. Elle a pour rôle d'une part de générer le vecteur transfert de la procédure, d'autre part d'emmagasiner dans un bloc d'indicateurs spécialisés BLOCOC, les caractéristiques de ces paramètres. Celles-ci sont conservées sur quatre positions binaires par paramètre. Les deux positions de droite donnent le type du paramètre :

0 = sans type
1 = booléen
2 = entier
3 = réel

Les deux positions de gauche donnent le genre du paramètre :

0 = quantité simple ou étiquette
1 = procédure
2 = tableau
3 = aiguillage

- La deuxième partie de P18 est relative au ";" marquant la fin de la liste des paramètres.

L'étiquette Pn est alors générée, puis l'en-tête de procédure, dont l'essentiel est l'appel au sous-programme ALPRO

ou ALPRR .

En paramètres sont donnés l'adresse Fn , le nombre p de paramètres, les caractéristiques de ces paramètres, prises dans BLOCOC et regroupées par mémoire, et enfin l'adresse de la mémoire spécifique de la procédure.

EXEMPLE Soit l'en-tête de procédure suivant :

PROCEDURE PROC (A, B, C) ;

ENTIER A ; TABLEAU B ;

BOOLEENNE PROCEDURE C ; .... etc ...

Si P6 est le numéro de la procédure PROC , si les numérotations R et NIM sont arrivées respectivement à 5 et 4, la séquence générée est, s'il n'y a pas RECUR :

```
        TRA       R5
        TRA *     **
        TRA *     **
        TRA *     **
P6      ***
        TSL       ALPRO
        BCI       1,P6
        PZE       F6,,3
        OCT       12' 500000000
        PZE       M - 4
        ... etc ...
```

H3 - 3 Phase P4

La phase P4 a pour opérateur le symbole VALEUR et pour opérandes les profils en PN des paramètres formels appelés par valeur.

Pour un paramètre du type variable simple, il est fait appel au sous-programme de calcul de l'adresse du paramètre, dont l'adresse elle-même est chargée dans le vecteur transfert par ALPRO .

Pratiquement, il faut mettre NDYNL à jour avant d'effectuer l'appel (sauf au premier appel, puisque ALPRO l'a fait), et l'adresse est obtenue au retour en registre index X2 (Cf. I). La valeur du paramètre est alors placée dans la mémoire associée suivant Fn .

<u>EXEMPLE</u> Si, dans le cas précédent, A était appelé par valeur, on génèrerait

```
TSX     P6 - 1,4
SXA     * + 1,2
CLA     * *
STO     F6 + 1
```

Si A n'était pas le premier paramètre appelé par valeur, cette séquence serait précédée de

```
CLA     M - 4
STO     NDYNL
```

Si le paramètre est du genre tableau, on fait appel au sous-programme ALVTA, en lui fournissant l'adresse de la mémoire spécifique en index X2 . Si plusieurs paramètres successifs sont des tableaux appelés par valeur, un seul appel à ALVTA est fait, avec autant de paramètres. Ces paramètres sont les adresses des mémoires associées (Fn + k) , chargées par ALPRO des caractéristiques du tableau effectif, trouvées dans la séquence d'appel de la procédure.

<u>EXEMPLE</u> Si, dans l'exemple précédent, B avait été appelé par valeur, on aurait généré à la suite de l'en-tête :

```
AXT     M - 4,2
TSX     ALVTA,4
PZE     F6 + 2
```

H3 - 4 Phase P19

Cette phase a pour opérateur le symbole FCODE et pour opérande en PN le corps de la procédure recopié sans transcodage par PC_1 (Cf. première partie).

Pratiquement, on définit

<corps de procédure en code machine>
::= <CODE perforé à l'extrémité droite d'une carte>
<toute suite d'instructions MAP non étiquetées par des symboles susceptibles d'être générés par le compilateur>
<FCODE perforé à l'extrémité gauche d'une carte>

Ainsi, le corps en instructions MAP est directement intégrable au programme généré sur B_G, carte par carte, sans aucune transformation.

De cette règle découlent également les caractéristiques d'emploi des procédures en code machine, notamment en ce qui concerne l'étiquetage des instructions et l'appel des paramètres.

En particulier, on voit qu'il est nécessaire de connaître tous les symboles générés par le compilateur : pour plus de facilité on pourra par exemple se limiter à l'emploi dans les corps en code de symboles débutants par la lettre Z .

D'autre part, l'appel des paramètres nécessite la connaissance des symboles internes exacts correspondants à ces paramètres : ceux-ci sont donnés par la table d'équivalences construite par PC_1 (Cf. première partie).

H3 - 5 Phase P5

La phase P5 (opérateur $\downarrow^{***}$) est déroulée en trouvant ";" de fin de déclaration de procédure.

Sa première fonction est de générer l'appel en registre AC du contenu de la mémoire Fn si la procédure qui se termine possède un type (c'est à dire si c'est un indicateur de fonction).

Puis, la séquence générée diffère suivant qu'il y a ou non demande de l'option RECUR .

Si RECUR n'est pas demandée, on génère (P5.S1) la remise à zéro du compteur de récursivité, puis l'instruction de retour de la procédure, dont l'adresse est chargée dès l'entrée par ALPRO , et enfin la réservation des p + 2 mémoires, dont la deuxième est étiquetée Fn , appelées à contenir respectivement le compteur de récursivité de la procédure, la valeur de l'indicateur de fonction et celle des paramètres ou leurs adresses.

Si RECUR est demandée , P5 génère l'appel au programme de désempilage ALDST , en lui donnant pour paramètres l'adresse Pn, le nombre p de paramètres, le compteur de récursivité et l'adresse de retour, les p + 2 mémoires réservées en Fn et enfin les adresses des mémoires locales de la procédure (également utilisées par ALPRR pour l'empilage).

Ces adresses sont elles-mêmes générées par la sous-phase P5.S2 , à l'aide de la pile PQTM : celle-ci contient à son sommet les valeurs, au début de la compilation de la procédure, des numérotations utiles ici : quantités simples locales ou rémanentes, tableaux locaux ou rémanents, mémoires de manoeuvre. Ces valeurs donnent directement les adresses des premières mémoires à sauvegarder dans chaque numérotation. Les différences entre ces valeurs et les valeurs actuelles donnent le nombre de mémoires à sauvegarder respectivement dans chaque numérotation.

Si une numérotation n'a pas varié, aucune adresse n'est donnée pour elle, de sorte que le nombre de paramètres donnés à ALDST n'est pas fixé. Pour cela, une mémoire toute nulle est générée pour indiquer la fin de la procédure.

Enfin, dans tous les cas, on teste si le symbole qui suit ";" marque le début d'une autre déclaration de procédure (ou d'aiguillage). Si oui, un aiguillage est mis à un dans P20 , pour éviter de générer un transfert intermédiaire. Sinon, le niveau nul créé en P20 dans PE est effacé, et l'étiquette se trouvant alors au sommet de PE est générée, pour adresser l'instruction qui suit la déclaration terminée.

H4 - EXEMPLE COMPLET

Soit la procédure suivante (sans valeur sémantique). On suppose que le symbole interne de son identificateur est P6 , que les numérotations R , NIM et quantités simples, sont respectivement arrivées à 5, 3, et 7, que RECUR est demandée et qu'une instruction suit cette déclaration :

```
ENTIER PROCEDURE PROC (A, B) ; VALEUR A, B ;
          ENTIER A, B ;
            DEBUT ENTIER I ;
            PROC := I := (A + 1) ÷ (B + 2) FIN ;
```

Si 1 et 2 sont respectivement les nombres N + 3 et N + 7 , on aura la génération suivante :

```
     TRA     R5                                  CLA     F6 + 1
     TRA*    **          vecteur                 ADD     N + 3
     TRA*    **          transfert               STO     M - 5
P6   ***                                         CLA     F6 + 2
     TSL     ALPRR                               ADD     N + 7
     BCI     1,P6                                STO     M - 6
     PZE     F6,,2       en-tête                 CLA     M - 5      Affectation
     OCT     104000000000                        LRS     35
     PZE     M - 3                               DVP     M - 6
     TSX     P6 - 1,4                            STQ     Q - 7
     SXA     * + 1,2                             STQ     F6
     CLA     **                                  CLA     F6
     STO     F6 + 1                              TSX     ALDST,4
     CLA     M - 3                               PZE     P6,,2      fin procédure
     STO     NDYNL       Valeur                  TRA     **
     TSX     P6 - 2,4                            BES     1
     SXA     * + 1,2                     F6      BSS     3          (p + 1)
     CLA     **                                  PZE     4,,M - 3   (m.m)
     STO     F6 + 2                              PZE     1,,Q - 7   (q.s)
     CLA     M - 3       DEBUT                   PZE
     STO     M - 4                       R5      .... etc ...
```

I - SOUS - PROGRAMMES DE PARAMETRES

APPELS DE PROCEDURES

I1 - GENERALITES

Tout appel de procédure se traduit dans un programme généré P_M par l'appel d'un sous-programme, fermé ou non, écrit lui-même dans P_M ou non, toujours susceptible de modifier les valeurs des divers registres de la machine ainsi que celle du niveau NDYNL de la mémoire dynamique. Ce sous-programme revient au programme P_M pour exécuter les instructions qui suivent la séquence d'appel. La valeur du registre AC au retour peut avoir une signification, si la procédure en question est employée comme indicateur Enfin, en général, un ou plusieurs paramètres sont communiqués au sous-programme appelé. Puisque l'on ne sait pas à la compilation si ces paramètres sont utilisés par le sous-programme pour leur adresse ou pour leur valeur, seule leur adresse est communiquée. On verra que la notion d'adresse est prise ici dans un sens très large.

Ces propriétés essentielles étant rappelées, il est clair que toutes les phases de génération relatives aux appels de procédure et de paramètres se diviseront naturellement en plusieurs branches déroulées respectivement suivant que la procédure en cause

est une procédure déclarée dans le programme ou une fonction dite "standard" incluse dans la bibliothèque du système de programmation. Ce dernier cas se subdivise lui-même, suivant que le sous-programme traduisant la fonction est ouvert ou fermé et, s'il est fermé, suivant que la fonction est un programme "utilisateur" ou un programme "compilateur" de calcul ou de déclaration d'une variable indicée.

Nous allons successivement étudier les formes des séquences générées relatives à ces divers cas, puis décrire succinctement le fonctionnement des phases de génération correspondantes.

I2 - PROCEDURES DECLAREES

I2 - 1 Appel proprement dit

Etant donnée la procédure P déclarée dans le programme, son appel se traduit par un transfert à l'instruction étiquetée Pn située en tête de sa déclaration.

Au retour de Pn, le registre AC est susceptible de contenir la valeur de l'indicateur de fonction correspondant.

Tout appel de procédure devant être considéré comme un ALLER A vers un début de bloc (Cf. chapitre II, F4), le niveau NDYNL doit être mis à jour par la mémoire spécifique du bloc où l'on se trouve, avant d'effectuer l'appel. La génération de cette mise à jour est réalisée, on l'a vu, par le sous-programme C .

On a vu que ALPRO (Cf. H) mettait lui-même à jour, par cette nouvelle valeur de NDYNL, la mémoire spécifique de la procédure.

I2 - 2 Liste des paramètres

A la suite de l'instruction d'appel, on génère la liste des paramètres effectifs communiqués à la procédure, précédée du nombre de paramètres.

Cette liste comporte une mémoire pour chaque paramètre, construite de la façon suivante :

1°) Le type et le genre du paramètre sont indiqués en zone préfixe :

+ 0 si le paramètre est lui-même un paramètre formel, ou une variable sans type (étiquette, chaîne ...), ou un identificateur de tableau.
+ 1 si le type est booléen
+ 2 si le type est entier
+ 3 si le type est réel
- 0 si le paramètre est un identificateur d'aiguillage.

2°) La zone adresse contient :

- L'adresse du sous-programme de calcul du paramètre si celui-ci est une expression arithmétique ou booléenne, une chaîne, ou une expression de désignation non réduite à une étiquette simple.
- Le symbole Pn si le paramètre est un identificateur de procédure. Dans ce cas, on ajoute en partie décrément le symbole Fn.
- Le symbole U - k si le paramètre est un identificateur de tableau (Cf. J).
- Le symbole An si le paramètre est un identificateur d'aiguillage (Cf. K).
- Le paramètre lui-même s'il est réduit à un paramètre formel ou à une étiquette simple.

Ces deux groupes de spécifications permettent au programme ALPRO de vérifier la conformité des types des paramètres avec ceux des paramètres formels, et de charger dans les mémoires spécialisées Pn - k et Fn + k les adresses désirées.

I2 - 3 Compilation des paramètres

Au fur et à mesure qu'une liste de paramètres effectifs est compilée, un sous-programme est construit pour chaque paramètre qui est une variable, une expression, une chaîne, ou une expression de désignation non réduite à une étiquette simple.

Les étiquettes adressant ces sous-programmes sont du type L_r et ce sont elles qui sont placées dans la liste des paramètres.

Ces sous-programmes sont destinés à être appelés chaque fois que l'on désire obtenir dans un corps de procédure l'adresse d'un paramètre effectif de la procédure.

Cette adresse est fournie en registre index X2 et on en a vu l'utilisation en H3 - 3 .

Les valeurs d'un compteur COPARE de paramètres effectifs, les caractéristiques de ces paramètres, et les profils de leurs étiquettes d'appel sont emmagasinés respectivement dans les piles PCPED , PPED et PEPE , qui sont vidées lors de la construction de la liste des paramètres (S32).

Enfin, on a vu (chapitre II, F2) que si un paramètre était une expression, la mémoire de manoeuvre où est rangée sa valeur (et dont l'adresse est transmise en X2) est choisie suivant un fonctionnement particulier faisant intervenir CIM2 et la pile PCIM2.

Ces mémoires de manoeuvre ont leurs profils qui restent en PN jusqu'à la fin de la compilation de l'appel de procédure (P7) et sont isolées par un "marqueur" des autres profils susceptibles d'être conservés dans PN .

I2 - 4 Forme de la séquence générée

Pratiquement, la séquence générée pour l'appel de procédure le plus général :

P (PE1, PE2 ... PEp)

est

(1)		- Mise à jour de NDYNL
(2)		- Transfert en Ri
(3)	L_r	- Début de sous-programme pour le premier paramètre
(4)		- Corps de sous-programme
⋮		
(2i+1)	$L_{r'}$	- Fin de sous-programme pour le paramètre i et - Début pour i'
⋮		
(N)	Ri	- Fin du dernier sous-programme - Appel à Pn - Nombre de paramètres - Liste des paramètres

La phase (1) est réalisée par le sous-programme C . La phase (2) est réalisée par P25 (avec S30) et toutes les phases 3, 5 .... 2i + 1 ... sont réalisées conjointement par P26 et P27.

Le nombre de ces phases ne peut pas être précisé puisque tous les paramètres ne donnent pas lieu à un sous-programme.

Enfin, la phase finale N est réalisée par P7.S2, à l'aide des sous-programmes S31 et S32 .

Notons que les phases 2 à N non comprise sont supprimées, ainsi que l'appel à S32, dans le cas d'une procédure sans paramètre (P1).

I3 - PROCEDURES STANDARD FERMEES

Grâce à l'indicateur PROCST mis à un dans ce cas, l'appel d'une procédure standard fermée entre entièrement dans le fonctionnement des phases précédentes. La forme de la séquence générée diffère peu, mais les spécifications des paramètres dans la liste ne sont pas données de la même manière.

I3 - 1 Changements dans la forme de la séquence

L'absence d'effet de bord dans les fonctions adoptées comme "standard" permet de générer des sous-programmes ouverts de calcul des paramètres effectifs. Ceux-ci ne pouvant être que des variables, des expressions, des chaînes ou des étiquettes simples, la liste de paramètres renfermera directement les adresses des paramètres.

Ainsi la phase numéro 2 disparait (la phase numéro 1 est également supprimée, Cf. chapitre II), ainsi que les phases 3, 5 .... 2i + 1 . Mais l'adresse d'un paramètre pouvant elle-même être le résultat d'un calcul (variable indicée, paramètre formel par nom, chaîne), l'instruction d'appel est étiquetée par un symbole du type Yr , afin de pouvoir, dans la compilation d'un tel paramètre, générer un rangement de l'adresse calculée dans la mémoire de la liste associée au paramètre, et qui a pour adresse Yr + i si le paramètre est le ième de la liste.

Trois piles sont utilisées alors : PCPES , PPES , PEST, contenant respectivement les valeurs d'un compteur de paramètres de

fonctions standard, les caractéristiques de ces paramètres et les profils des étiquettes Yr .

I3 - 2 Liste des paramètres

La liste des caractéristiques est construite en P7 (sous-programme S33) avec les conventions suivantes :

Une mémoire est générée pour chaque paramètre avec

1°) Un préfixe toujours nul

2°) Le type du paramètre indiqué en décrément par

0 = pas de type (étiquette)

1 = Booléen

2 = entier

3 = chaîne

4 = réel

3°) La partie adresse contenant l'adresse du paramètre. Cette adresse est générée en blanc et chargée à l'exécution par le sous-programme du paramètre quand celui-ci est une variable indicée, un paramètre formel par nom ou une chaîne.

I3 - 3 Cas particulier des variables indicées

Tout occurrence - déclaration ou exploitation - d'une variable indicée dans P_S se traduit dans P_M par l'appel d'une fonction standard de type générique TABF (Cf. chapitre II). Cet appel entre dans le fonctionnement général de l'appel d'une fonction standard. Cependant, des phases particulières (P6 et P7.S1) étant alors déroulées au lieu de P7.S2 , des aiguillages sont susceptibles d'être positionnés dans diverses sous-phases pour rappeler que l'on est dans ce cas.

I4 - FONCTIONS STANDARD OUVERTES

Des fonctions standard particulièrement simples sont générées en sous-programmes ouverts dans P_M .

Ces fonctions admettent zéro ou un paramètre, et leur liste est donnée en appendice avec les autres fonctions standard actuellement disponibles.

Pratiquement l'appel de ces fonctions se traduit par la génération d'un sous-programme ouvert pour le paramètre, de l'appel en registre AC de ce paramètre, et d'une macro-opération spécialisée. Celle-ci est générée par P7.S1 qui fait alors appel à la sous-phase S34 au lieu de S31 et S33 .

I5 - DESCRIPTION DES PHASES DE GENERATION

I5 - 1 Début d'appel : P25 (↓7)

La phase P25 commence par appeler la sous-phase S30 qui génère les protections de registres nécessaires, et qui contient en fait l'appel au sous-programme C (SP12) qui, pour plus de clarté, a été séparé sur l'organigramme de la planche 10 .

Les compteurs de paramètres effectifs de procédure déclarées (COPARE) ou standard (COPSTT) sont rangés dans leurs piles respectives PCPED ou PCPES , puis réinitialisés à zéro.

Si la procédure est déclarée, un marqueur M est mis en PN , AC et MQ sont spécifiés inconnus, et un transfert est généré à une étiquette du type R , construite ici et rangée au sommet de PE.

Si la procédure est standard (fermée obligatoirement), une étiquette du type Y est créée et rangée au sommet de PEST .

I5 - 2 Début de paramètre : P26 (↓ 7*)

Cette phase assure d'une part l'initialisation des sous-programmes de paramètres effectifs, d'autre part la reconnaissance de ceux de ces paramètres ne donnant pas lieu à des sous-programmes.

Si la procédure appelée est déclarée, le compteur COPARE est incrémenté de un, et les profils des paramètres ne donnant pas lieu à des sous-programmes sont emmagasinés directement dans PPED .

Si un sous-programme doit être généré, une étiquette du type L est créée, rangée au sommet de PEPE , et générée afin d'adresser la première instruction du sous-programme. Rappelons que c'est cette étiquette qui apparaîtra dans la mémoire associée de la liste de paramètres.

Si la procédure est standard, c'est COPSTT qui est augmenté de un , et les seuls paramètres emmagasinés directement en PPES sont ceux formés d'aucune unité syntaxique (Cf. programmes d'entrée - sortie). Le paramètre est alors pris comme nul, et c'est une mémoire nulle qui lui sera associée par S33 dans la liste suivant l'appel.

Remarquons que, dans tous les cas où un paramètre est directement traité par P26, c'est à dire placé en PPED ou PPES , la phase P26 passe directement le contrôle à la phase P7 si l'unité syntaxique qui suit est ")" et reprend le contrôle au début de P26 si c'est "," .

Dans tous les autres cas, il y a retour à la partie analytique (Cf. chapitre II), pour que le sous-programme de calcul se génère suivant le fonctionnement général.

I5 - 3 Fin de paramètre : P27

Cette phase (opérateur ↓ 7**) est déroulée pour chaque paramètre effectif de procédure possédant un sous-programme et a pour opérande, le profil du paramètre situé au sommet de PN .

On vérifie que cet opérande est bien présent (PN ne doit pas avoir le marqueur M à son sommet) et la fin du sous-programme du paramètre est alors générée.

Le profil qualitatif de la procédure appelée est en MEM APPEL et l'indicateur PROCST est à un si c'est une procédure standard.

I5 - 3 - 1 Cas d'une procédure déclarée

PROCST est à zéro et les séquences générées dans les différents cas sont alors les suivantes :

1°) L'opérande est un résultat en registre AC ou MQ.
Si ce résultat est celui d'une expression arithmétique ou booléenne, on génère son rangement dans une mémoire de résultat M - k (Cf. chapitre II, F2), dont le profil remplace en PN celui du paramètre. Puis on génère l'appel en X2 de l'adresse M - k (instruction AXT).

Si l'opérande est le résultat du calcul de l'adresse d'une variable indicée, l'adresse du paramètre est elle-même en AC (Cf. I2) et il suffit de générer son chargement en X2 (Instruction PAX). Si c'est le résultat d'un calcul d'adresse de chaîne, celle-ci est déjà en X2 (Cf. L), on ne génère rien.

Dans ces deux cas, l'opérande est effacé dans PN.

2°) L'opérande est une variable simple , un nombre pur, une valeur logique ou un paramètre formel par valeur. L'adresse en est alors placée en X2 (instruction AXT). L'opérande est effacé dans PN .

3°) L'opérande est un paramètre formel par nom. L'appel du sous-programme du paramètre a déjà été fait en plaçant l'opérande en PN (phase P16). L'adresse du paramètre effectif initial est donc déjà en X2 et rien n'est généré. L'opérande est effacé dans PN .

4°) L'opérande est une expression de désignation. C'est alors une étiquette calculée et la génération s'est faite comme pour ALIER A , avec un transfert en dernière instruction. Aucune adresse n'est à placer en X2 et l'opérande est effacé dans PN .

Dans tous les cas, la fin du sous-programme est marquée par une instruction de retour :

TRA 1,4

Cependant, le registre X4 a pu être utilisé au cours du sous-programme du paramètre, et sauvegardé auparavant (phase de génération SP11) par une instruction du type

SXA N_r , 4

L'étiquette Nr est alors générée à la fin du sous-programme (elle a été construite à l'aide du profil de L_r contenu dans PEPE) et se réfère à une instruction

AXT ** , 4

précédant l'instruction de retour.

On sait que l'instruction étiquetée N_r est à générer grâce à la présence dans la mémoire indicatrice du bloc de paramètre effectif (pile PB) de la position C insérée par SP11.

Enfin, l'index de la pile PEPE est reculé de un et le profil qualitatif de L_r est placé dans la pile PPED .

Puis, la séquence générée est imprimée (SP16) et les indicateurs IAC et IMQ sont mis à un pour spécifier que les registres ont une valeur inconnue. Le retour se fait sur la phase P7.S1 ou P6 si l'unité syntaxique qui suit le paramètre est] , sur P7.S2 si c'est ")" et sur P26 si c'est "," .

I5 - 3 - 2 Cas d'une procédure standard

Les différents cas de génération sont alors les suivants :

1°) L'opérande est une variable simple , un paramètre formel par valeur, une valeur logique, un nombre pur ou une étiquette simple.

Le profil de l'opérande est placé au sommet de PPES et effacé dans PN .

2°) L'opérande est un résultat d'expression en registre AC ou MQ .

On génère le rangement du résultat en mémoire de manoeuvre et le profil de celle-ci est rangé en PPES , le profil de l'opérande étant effacé dans PN .

3°) L'opérande est le résultat d'un calcul d'adresse en AC (variable indicée ou chaîne) ou en X2 (paramètre formel par nom). Dans ce dernier cas, on génère le rangement de X2 en AC (instruction PXA) et de toute façon

on génère le rangement de la partie adresse du registre AC (instruction STA) dans la mémoire de la liste associée au paramètre. Rappelons que cette mémoire est Yr + i , Yr étant l'étiquette d'appel placée au sommet de PEST et i le numéro du paramètre dans la liste, contenu à ce moment dans COPSTT.

Dans ce cas, le profil rangé dans PPES est réduit au type du paramètre (positions 10, 11, 12) qui sera généré par S33 (Cf. I5 - 4).

Enfin, le retour se fait dans tous les cas comme pour une procédure déclarée.

I5 - 4 Fin d'appel : P7.2 et P1 (↓)

La génération de l'appel proprement dit de la procédure, et éventuellement de la liste de paramètres se divise en trois branches suivant qu'il s'agit d'une procédure déclarée, standard fermée ou standard ouverte.

I5 - 4-1 Procédures déclarées

On génère avant l'instruction d'appel l'étiquette Ri placée au sommet de PE par P25 . (toutefois, cette étiquetage, et le TRA Ri qui s'y rapporte, sont supprimés si la procédure est sans paramètres (phase P1) ou si ses paramètres n'ont donné lieu à aucun sous-programme (le sommet de Z_G contient alors une instruction du type TRA Ri).

Puis, on fait appel successivement aux sous-phases S31 et S32 générant respectivement l'instruction d'appel proprement dite et la liste des paramètres de l'appel, dont les profils sont en PPED .

L'instruction d'appel est de la forme

Ri TSL Pn

si Pn est le symbole interne de la procédure appelée et

Ri TSL Fn' + k'

si la procédure appelée est elle-même le k'ième paramètre de la procédure Pn' . En effet, dans ce cas, le programme ALPRO a placé, à l'entrée dans Pn', l'adresse du paramètre effectif numéro k' dans la mémoire Fn' + k' . Cette adresse étant elle-même Pn si Pn est l'identificateur de procédure qui est k'ième paramètre effectif de Pn' , l'instruction d'appel est bien équivalente à

TSL Pn

Après l'instruction d'appel est générée une mémoire du type

PZE ,,p

où p est le nombre de paramètres effectifs de l'appel (contenu ici en COPARE).

Enfin, IMQ est mis à un, le registre MQ ayant une valeur inconnue au retour d'une procédure, et la liste des paramètres est générée (S32).

Au fur et à mesure de cette génération, l'index de la pile PPED est reculé, les mémoires de manoeuvre ayant servi de mémoires résultats aux paramètres sont libérées et leur profil effacé dans PN , ainsi que le marqueur qui les limitait.

Puis COPARE est rechargé par la valeur qu'il avait au début de l'appel et placée alors en PCPED (ceci afin d'assurer l'imbrication des appels de procédures).

Enfin, si la procédure est un indicateur de fonction (positions 10, 11, 12 de MEM APPEL non toutes nulles), le résultat est spécifié en AC, avec les positions de mémoire de manoeuvre et de type en IAC , et un profil nul en PN. Sinon, AC est spécifié inconnu.

Dans tous les cas, il y a impression de la séquence générée avant le retour sur le test de hiérarchies.

I5 - 4 - 2 Procédures standard fermées

L'étiquette générée devant l'instruction d'appel est ici Yr , dont le profil est en PEST, qui est ici reculée de un.

L'instruction d'appel générée est

TSX XXXX,4

où XXXX représente le symbole interne de la procédure standard. Ce symbole est fourni par une table TAFST dont chaque ligne contient le symbole correspondant à son numéro. C'est le sous-programme SP14 de construction du champ variable des instructions qui consulte la table TAFST . En même temps, l'indicateur correspondant au numéro de la fonction est mis à un dans le bloc BLFST d'indicateurs de fonctions standard, nuls au départ (Cf. chapitre III B2).

Comme pour une procédure déclarée,le registre MQ est spécifié inconnu.

Puis la liste de paramètres est générée, par S33, en libérant au fur et à mesure les mémoires de manoeuvre éventuellement utilisées pour les résultats des paramètres et dont les profils sont en PPES . L'index de cette pile est automatiquement reculé et le compteur COPSTT reprend la valeur qu'il avait avant l'appel et conservée dans PCPES.

Enfin, le retour s'effectue comme pour les procédures déclarées, en laissant un opérande à profil nul en PN si la procédure est un indicateur de fonction (résultat en AC).

I5 - 4 - 3 Procédures ouvertes

Le sous-programme ouvert du seul paramètre éventuel d'une procédure ouverte a été généré suivant le fonctionnement général, et le profil du résultat reste comme opérande en PN.

La génération de la procédure elle-même se fait alors comme pour une opération, une table APROU fournissant l'adresse des sous-phases correspondant à chaque numéro de fonction standard ouverte.

Numéro 1 : ABS (X)

Après vérification du type de X (réel ou entier), on génère l'appel de X en AC (SP1) et l'instruction

SSP

Le retour se fait sur le test de hiérarchies en laissant en PN un profil d'opérande nul (résultat en AC) dont le type est celui de X .

Numéro 2 : SIGN (X) ou SIGNE (X)

On génère, après l'appel du paramètre, la macro - instruction (Cf. chapitre III B1)

SIGN

Le résultat est en AC , de type entier.

Numéro 3, 4, 5 : ENTIER (X)
CONVRE (X)
CONVER (X)

On génère, après l'appel du paramètre, les macros - instructions respectives

ENTIE
CONVRE
CONVER

Les résultats sont en AC, respectivement de type entier, entier et réel.

Numéro 6 : HEURE

HEURE est une procédure ouverte sans paramètres qui fait appel au sous-programme de prise de l'heure intégré dans le système [29, 30] et imprime l'heure sur l'unité physique de sortie des résultats.

Dernière procédure : MODELE (CH)

MODELE est une procédure d'entrée - sortie dont le paramètre est une chaîne (formelle ou non) constituée par un modèle de lecture / impression du type FORTRAN . Employée conjointement avec les procédures ENTREE et SORTIE (Cf.troisième partie), elle livre à celles-ci, en registre AC , l'adresse du modèle employé.

I5 - 5 Appel d'un paramètre formel : P16 (↓ 9)

Chaque fois qu'un paramètre formel appelé par nom est rencontré dans P_S , on génère l'appel au sous-programme correspondant s'il en existe un (c'est à dire si le paramètre n'est pas un identificateur de tableau, de procédure ou d'aiguillage).

Cet appel, on l'a vu, se fait par

TSX Pn - k,4

si le paramètre est le k ième de la n ème procédure.

Avant le TSX, le programme C a été susceptible de générer la mise à jour (SP12.S1) ou la sauvegarde de NDYNL et la protection de l'index X4 (SP12.S2, appelant SP11).

Au retour du sous-programme, l'adresse du paramètre est en index X2. En général on la rangera dans la mémoire associée Fn + k et toute référence à l'opérande dans PN se traduira par l'insertion de la marque d'adressage indirect (*) dans les instructions correspondantes (SP14).

Remarquons que, si le paramètre formel est du type étiquette, le sous-programme ne reviendra pas - en général - derrière l'appel. On a vu en effet qu'alors l'instruction ALLER A était automatiquement simulée. Le sous-programme ne reviendra dans le corps de la procédure Pn que si le paramètre effectif correspondant est un indicateur d'aiguillage dont la valeur est indéterminée (Cf. K). Dans tous les cas où un sous-programme de paramètre ne revient pas dans la procédure Pn, on devra considérer cette procédure comme appelée récursivement.

Enfin, si l'option RECUR est présente, la génération de l'appel d'un paramètre formel est compliquée par le fait qu'on ne doit pas faire réexécuter le sous-programme d'un paramètre par nom une fois entré dans la procédure, sous peine de boucler indéfiniment sur cet appel. La séquence générée contient alors un test du compteur de récursivité de la procédure (d'adresse Fn - 1).

I6 - EXEMPLE COMPLET

Soit l'instruction

A := P (3, SIN(X), ETIQ) + ABS (LN(B)) ;

Nous supposons que cette instruction se situe dans le corps d'une procédure numéro 3 dont A et B sont les paramètres de type réel appelés par nom et M - 3 la mémoire spécifique, que P est la procédure déclarée numéro 4 de type réel, que 3 est le nombre N + 1, que SIN, de type réel, a pour symbole interne SINA, que X est la quantité entière Q - 7, que ETIQ est l'étiquette E5, que LN, de type réel, a pour symbole interne LOGA.

Enfin, les numérotations M, R, L, Y sont supposées arrivées respectivement à 6, 4, 3, 5 .

La génération obtenue est alors la suivante :

```
      CLA    M - 3        )
      STO    NDYNL        }   A
      TSX    P3 - 1,4     }
      SXA    F3 + 1,2     )
      CLA    M - 3        )
      STO    NDYNL        }   P(
      TRA    R4           )
L3    AXT    N + 1,2      }   3
      TRA    1,4          }
L4    SXA    N4, 4        ,
Y5    TSX    SINA,4       )
      PZE    Q - 7 ,,2    }   SIN(X)
      STO    M - 6        )
      AXT    M - 6,2
N4    AXT    **,4
      TRA    1,4          ,
```

```
R4    TSL    P4           )
      PZE    ,,3          }
      PTW    L3           }   )
      PTH    L4           }
      PZE    E5           )
      STO    M - 6
      CLA    M - 3        )
      STO    NDYNL        }
      TSX    P3 - 2,4     }   B
      PXA    ,2           }
      STA    Y6 + 1       )
Y6    TSX    LOGA,4       )
      PZE    **,,4        }   LN(B)
      SSP                 )   ABS
      FAD    M - 6            +
      STO*   F3 + 1           :=
```

J - APPELS DES VARIABLES INDICEES

Seule la phase de génération déroulée en détectant] de fin de variable indicée diffère de la phase P7.S1 déroulée en détectant) de fin d'appel de fonction standard.

Cette phase est P6 si] marque la fin d'une section de tableau (opérateur $\downarrow^{*}$) et P7.S2 s'il marque la fin d'une exploitation (opérateur $\downarrow$) .

En plus, la liste des paramètres suivant l'appel est construite par la même sous-phase S33 que pour les fonctions standard. C'est donc la sous-phase S31 qui est remplacée dans P6 et P7.S1 par la génération de l'appel au sous-programme interpréteur de type TABF .

J1 - FIN DE SECTION DE TABLEAU : P6

Après avoir généré le calcul des paramètres de l'appel, qui sont ici les bornes du (des) tableau (x) en déclaration, il faut générer l'appel au sous-programme interpréteur TABD (Cf. troisième partie) qui construit en mémoire dynamique le "bloc de bornes" du tableau et réserve ses mémoires en positionnant le niveau dynamique.

Ce niveau est NDYNL ou NDYNR (Cf. chapitre I, B5) suivant que le tableau est de type local ou rémanent.

En plus, TABD communique à P_M , dans une (plusieurs) mémoire (s) U - k (ou V - k si le tableau est rémanent) les caractéristiques du (des) tableau (x), c'est à dire le type, le nombre de bornes et l'adresse en mémoire dynamique du début de tableau. C'est cette mémoire qui est donnée par PC_1 comme le symbole in-

terne du tableau et qui est communiquée, par l'intermédiaire de ALPRO , aux mémoires associées du type Fn + k si le tableau est employé en paramètre effectif d'une procédure.

Pratiquement, ces divers paramètres sont fournis à TABD de la façon suivante :
avant de générer

Y_r TSX TABD , 4

on place en registre X2 l'adresse de la mémoire spécifique du bloc où l'on se trouve (instruction AXT) et en registre AC une mémoire contenant en partie adresse le nombre de bornes (donné ici par COPSTT), en partie index le type du (des) tableau (x) (dont le (s) profil (s) est (sont) au sommet de PN) et en partie décrément l'indication de rémanence (position 17 à un). Le type est 1 pour un tableau booléen, 2 pour un entier et 3 pour un réel.

Puis la liste de paramètres est générée (S33) et, à la suite, on génère une mémoire par tableau déclaré dans la section, contenant - 0 en préfixe et U - k en adresse.

Enfin, COPSTT reprend la valeur qu'il avait au début de l'appel (contenu dans PCPES), PEST est reculée de un, et la séquence générée est imprimée.

IAC et IMQ sont mis à un pour spécifier l'AC inconnu, et un indicateur est positionné dans la séquence de fin de génération PFG qui fera générer l'instruction d'appel de TABD en mémoire lors de l'exécution.

J2 - FIN D'EXPLOITATION : P7.S1

Le programme appelé est cette fois TABE, qui fournit en AC l'adresse de la variable indicée cherchée. TABE a besoin, en plus de la liste de paramètres (qui sont ici les indices de la

variable), du contenu de la mémoire U - k spécifique du tableau appelé.

On génère pour cela :

```
        CIA         U - k
Y_r     TSX         TABE,4
```

Remarquons que si le tableau est un paramètre formel, c'est Fn + k qui est appelé en AC ; on a vu qu'il contenait automatiquement la mémoire U du tableau effectif correspondant.

Après génération de la liste, COPSTT reprend son ancienne valeur, PEST est reculée de un et le registre MQ est spécifié inconnu.

On spécifie un résultat en AC en laissant un opérande nul en PN . IAC contient alors les positions 2 de mémoire de manoeuvre et 10, 11 ou 12 suivant le type du tableau appelé. La position 15 lui est ajoutée pour signifier aux autres phases de génération (P2.S1 , SP1 , SP2, SP14) que le résultat est une adresse.

Ainsi, la phase P2.S1, si elle doit compiler l'affectation suivante :

variable indicée := variable simple

générera :

- Appel variable indicée, résultat en AC
- Appel variable simple en MQ (LDQ)
- Rangement de la partie adresse de AC dans l'instruction suivante (STA *+1)
- Rangement du MQ dans l'adresse obtenue (STQ **)

De même, si SP1 doit assurer l'appel en AC de la variable indicée dont on vient de calculer l'adresse, elle générera

```
STA         * + 1
CLA         * *
```

Enfin, si le résultat obtenu est rangé dans une mémoire de manoeuvre pour libérer AC , toute opération sur cette mémoire se traduira (SP14) par l'insertion dans le code opération de la marque d'adressage indirect * .

J3 - EXEMPLE COMPLET

Soit le bloc suivant à compiler :

```
DEBUT  TABLEAU  T1, T2 [A : N + 1]  ;
        B := 3 ;
     T1 [B + 2] := B x C ;
      T2 [B] := B + 2 ;
          C := T1 [T2 [B]] + T2 [3] FIN ;
```

On suppose que les numérotations M, U et Y sont arrivées respectivement à 4, 3 et 5 , que A, B, C, N sont des variables entières de symboles internes respectifs Q - 1 à Q - 4 , que les nombres 1, 2 et 3 sont représentés respectivement par N + 2, N + 4 et N + 5 .

La séquence générée est alors :

```
     CLA    M - 4   } DEBUT
     STO    M - 5
     CLA    Q - 4   }
     ADD    N + 2   } N + 1
     STO    M - 6   }
     AXT    M - 5,2
     CAL    = Ø 300002
Y5   TSX    TABD,4  }
     PZE    Q - 1,,2 } ]
     PZE    M - 6,,2 }
     MZE    U - 3   }
     MZE    U - 4   }
     CLA    N + 5   } B := 3
     STO    Q - 2   }
     ADD    N + 4   } B + 2
     STO    M - 6   }
     CLA    U - 3   }
Y6   TSX    TABE,4  } T1 [ B + 2 ]
     PZE    M - 6,,2 }
     STO    M - 6   }
     LDQ    Q - 2   } B x C
     MPY    Q - 3   }
     STQ*   M - 6   :=
```

```
T2[B]  Y7   CLA     U - 4
            TSX     TABE,4
            PZE     Q - 2,,2
            STO     M - 6
B +2        CLA     Q - 2
            ADD     N + 4
:=          STO*    M - 6
T2[B]  Y8   CLA     U - 4
            TSX     TABE,4
            PZE     Q - 2,,2
            STA     Y9 + 1
T1     Y9   CLA     U - 3
            TSX     TABE,4
            PZE     ,, 4
            STO     M - 6
T2[3]  Y10  CLA     U - 4
            TSX     TABE,4
            PZE     N + 5,,2
            STA     * + 1
+           CLA     * *
            FAD*    M - 6
            CONVRE
:=          STO     Q - 3
```

K - A I G U I L L A G E S

K1 - GENERALITES - FONCTIONS ESSENTIELLES

Les phases de génération associées aux aiguillages sont relatives d'une part à leur déclaration, d'autre part à leur exploitation.

Pour chaque déclaration, il y a déroulement des phases : P20 (opérateur $\downarrow^{1}$), commune aux déclarations de procédures (Cf. H), quand le symbole AIGUILLAGE est détecté, P28 ($\downarrow^{8}$) et P29 ($\downarrow^{8*}$) quand "," est trouvé entre deux expressions de la liste d'aiguillage, et P9 ($\downarrow^{**}$) quand ";" de fin de déclaration est trouvé. En plus, P28 et P29 sont déroulées respectivement en début et en fin de la liste.

Une déclaration d'aiguillage [1, 5 . 3] doit se traduire dans P_M par la présence d'autant de sous-programmes fermés qu'il y a d'expressions dans la liste, destinés à évaluer à chaque appel de l'aiguillage l'expression correspondante à la valeur numérique de l'indice de l'indicateur.

En plus, un vecteur transfert doit être construit pour ces sous-programmes, afin de pouvoir s'y transférer plus facilement.

Il est important enfin que ces sous-programmes soient considérés comme des blocs, puisqu'ils sont susceptibles de renfermer des appels de procédures ou de paramètres formels, ou d'autres indicateurs d'aiguillages. Dans ce dernier cas, ils doivent également avoir la possibilité de revenir aux instructions suivant l'appel, tout indicateur d'aiguillage étant susceptible de prendre une valeur indéterminée et d'être équivalent alors à une instruction vide [1, 4. 3. 5] .

Pour chaque indicateur d'aiguillage, l'expression en indice doit être compilée suivant le fonctionnement général, et c'est seulement en trouvant le] qui termine cette expression que l'appel peut être généré (phase P7.S3, opérateur ↓). La séquence construite doit tenir compte du fait que l'indicateur peut être équivalent à une instruction vide. On a vu que cette éventualité devait également être prévue en compilant l'appel d'un paramètre formel spécifié étiquette.

K2 - SCHEMA GENERAL DE GENERATION

K2 - 1 Déclaration

Etant donnée la déclaration d'aiguillage la plus générale :

AIGUILLAGE AIG := ED1, ED2 ... EDp ;

où EDi représente une expression de désignation quelconque, la séquence générée a la forme suivante :

(1) { - Transfert en Ri
K_{ED1} - Début de sous-programme pour ED1

(2) - Sous-programme pour ED1 (fonctionnement général)

(3) { - Fin du sous-programme pour ED1
K_{ED2} - Début de sous-programme pour ED2

|
|
|
|
|

(2j) - Sous-programme EDj

- Fin de sous-programme pour EDj

(2j+1) { $K_{ED_{j+1}}$ - Début de sous-programme pour $ED_j + 1$

|
|
|

$$(2p+1)\begin{cases} & \text{- Fin de sous-programme pour EDp} \\ A_{AIG} & \text{- Nombre d'entrées dans l'aiguillage} \\ & \text{- Vecteur transfert} \\ Ri & \text{- ..... etc ...} \end{cases}$$

La phase numéro 1 est construite par P20 et par le premier appel à P28.

Puis les phases numéros 2,4 ....2p sont construites suivant le fonctionnement général, et les phases numéros 3,5 .. ... 2p - 1 par les appels successifs à P29 et P28 .

Enfin, la phase numéro (2p + 1) est construite par le dernier appel à P29 et par P9 .

K2 - 2 Exploitation

Pour l'indicateur d'aiguillage général
AIG [EX]

où EX est une expression arithmétique, on génère une séquence de la forme suivante, où $C[A_{AIG}]$ représente le contenu de la mémoire étiquetée A_{AIG} :

(1) - Calcul de l'expression EX

(2) $\begin{cases} \text{- Si EX} \leqslant 0 \text{ , ou EX} > C\,[A_{AIG}] \text{ , sauter après la phase numéro 2} \\ \text{- Sinon, effectuer un transfert à la mémoire } A_{AIG} + EX \end{cases}$

La phase numéro 1 suit le fonctionnement général, la phase numéro 2 est assurée intégralement par P7.S3 . Rappelons que le transfert effectué en $A_{AIG} + EX$ peut être rendu équivalent à une instruction vide si le sous-programme K_{EDi} correspondant est lui-même sans effet.

K3 - DESCRIPTION DES PHASES

K3 - 1 Phases de déclaration

Nous ne reviendrons pas sur la phase P20 dont le fonctionnement a été exposé en H.

La phase P28 de début d'expression est sans opérande mais possède en COEA, le rang de cette expression dans la liste d'aiguillage et gère elle-même la pile PEA (Cf. chapitre I, F2) où elle emmagasine les profils des étiquettes du type qui seront générées dans le vecteur transfert par P9 .

Pratiquement, la phase 28 détermine également si un sous-programme est nécessaire pour l'expression qui commence : c'est le cas si celle-ci n'est pas réduite à une étiquette simple non formelle. Si le sous-programme n'est pas nécessaire, aucune étiquette K n'est construite et c'est l'étiquette qui forme l'expression qui elle - même placée en PEA .

Dans ce cas, le retour s'effectue immédiatement en supprimant de IAIG la position A , ce qui indique à P29 (fin d'expression) qu'elle n'a aucune opération à effectuer. Dans le cas contraire, l'étiquette K_{ED} est construite, à partir de la valeur d'un compteur COETA qui rend la numéroation K autonome.

La phase P29 , une fois le sous-programme compilé, et si IAIG n'est pas vide, génère ses instructions de fin et de retour.

Elle a pour opérande en PN le profil du résultat de l'expression, qui doit être de désignation.

Si l'expression terminée comportait des appels de variables indicées, de la fonction puissance, de procédure, de paramètres formels ou d'aiguillages, le programme SP11 , effectué dans ces cas, aura placé en IAIG la position B .

Si, en plus, la valeur de l'expression dépend finalement de celle d'un indicateur d' aiguillage ou d'un paramètre formel (ce qui apparait dans le profil de l'opérande), le sous-programme est susceptible d'être équivalent à une instruction vide et il faut assurer une instruction de retour. Dans ce cas, une instruction de rechargement du registre index X4 est nécessaire, SP11 ayant généré une sauvegarde de cet index dans une mémoire étiquetée par I_{ED} , construite à partir de K_{ED} dont le profil est en PEA .

La séquence générée est alors :

I_{ED}	AXT	**,4
	TRA	1,4

Dans les autres cas, l'instruction de retour est inutile, et si une sauvegarde de X4 avait été générée, son rechargement est sans objet. Afin que I_{ED} soit malgré tout défini, on génère une équivalence (EQU) entre I_{ED} et A , première instruction du programme servant de mémoire banale.

Remarquons que ce fonctionnement est tout à fait analogue à celui de P27 quand un paramètre effectif est une expression de désignation.

<u>Enfin, la phase P9</u> génère la mémoire A_{AIG} à partir du profil de l'aiguillage situé dans MEMAIG et du compteur d'entrées COEA.

Puis, le vecteur transfert est construit à partir des profils du type K ou E situés en PEA.

Celle-ci est vidée au fur et à mesure et la séquence est imprimée.

La fin de la génération est comme en P5, pour étiqueter l'instruction suivante à partir du profil contenu au sommet de PE.

K3 - 2 Phase d'exploitation : P7.S3

Cette phase a pour premier opérande en PN le profil du résultat de l'expression en indice de l'indicateur d'aiguillage, qui doit être de type réel ou entier (SP5), et pour deuxième opérande le profil de l'aiguillage appelé. Si l'expression est du type réel, on génère la conversion CONVRE (SP8).

Dans tous les cas, le registre MQ est rangé (SP4) et la valeur de l'expression appelée en AC (SP1).

La sous-phase SP11 est déroulée pour assurer la sauvegarde éventuelle de l'index X4 .

Puis on génère les instructions traduisant la phase numéro 2 du K2 - 2 .

Si l'aiguillage est spécifié non simple (Cf. chapitre II, F4), le sous-programme C est appelé qui génère, par SP12, la mise à jour de NDYNL par la mémoire spécifique du bloc où l'on se trouve.

L'instruction d'appel proprement dite est de la forme TSX ** , 4 , sa partie adresse correspondant à la mémoire du vecteur transfert de l'aiguillage dont le rang est égal à la valeur de l'expression en indice.

Puis la séquence est imprimée, AC et MQ sont spécifiés inconnus et le retour s'effectue en laissant dans PN le profil d'une étiquette de manoeuvre.

K4 - EXEMPLE COMPLET

Soit à compiler le bloc suivant :

```
DEBUT AIGUILLAGE AIG1 := ETIQ1 , ETIQ2 ;
      AIGUILLAGE AIG2 := SI U ALORS ETIQ1 SINON ETIQ2 . AIG1 [I] ;
            ALLER A AIG2 [J + K] FIN ;
```

On suppose que AIG1 et AIG2 sont les aiguillages A1 et A2, que ETIQ1 et ETIQ2 sont des étiquettes de symboles internes E1 et E2, que U est une variable booléenne Q - 1 et I, J, K trois variables entières Q - 2 , Q - 3 , Q - 4 , enfin que les numérotations M, R et K sont respectivement arrivées à 5, 3 et 2 et que le bloc où l'on se trouve a pour mémoire spécifique M - 2 .

La séquence générée est alors :

```
     CLA    M - 2   }  DEBUT
     STO    M - 5   }
     TRA    R3
A1   PZE    2       }
     TRA    E1      }  AIG1
     TRA    E2      }
K2   SXA    N2,4    }
     LDQ    NDYNL   }  ALORS
     STQ    M - 6   }
     CLA    Q - 1   }
     TNZ    E1         SI
     TRA    E2         SINON
N2   EQU    A
K3   SXA    N3,4    }
     CLA    Q - 2   }
     SUB    = 1     }
     TMI    * + 7   }  AIG1 [I]
     SUB    A1      }
     TPL    * + 5   }
     CAL    * - 2   }
     ADD    Q - 2   }
     STA    * + 1
     TSX    * *,4
N3   AXT    * *,4
     TRA    1,4
A2   PZE    2       }
     TRA    K2      }  AIG2
     TRA    K3      }
R3   CLA    Q - 3   }  J + K
     ADD    Q - 4   }
     STO    M - 7   }
     SUB    = 1     }
     TMI    * + 9   }
     SUB    A2      }
     TPL    * + 7   }
     CAL    * - 2   }  AIG2 [J+K]
     ADD    M - 7   }
     STA    * + 3   }
     CLA    M - 5   }
     STO    NDYNL   }
     TSX    * * , 4 }
```

L - CHAINES

La phase P24 qui traite les chaînes alphabétiques au sens ALGOL se divise en deux parties, ayant pour opérateurs les crochets d'ouverture et de fermeture de chaîne.

Rappelons qu'aucune chaîne imbriquée n'est autorisée dans l'écriture des programmes P_S .

Les chaînes étant uniquement utilisées comme paramètres effectifs de procédures, c'est par une adresse placée dans un registre déterminé qu'elles peuvent être caractérisées.

Ceci est réalisé de la façon suivante :

Une numérotation H dépendant d'un compteur COCH est définie, qui permet d'étiqueter les chaînes par des symboles du type Hn .

A cette numérotation correspond dans P_M un compteur d'assemblage [27] H . Toutes les instructions construites sous le contrôle de ce compteur seront assemblées après le programme P_M proprement dit.

Ainsi, la première partie de P24 génère les instructions

```
AXT        Hn,2
PXA        ,2
USE        H
```

L'adresse de la chaîne est alors à la fois en X2 et en AC .

Puis la deuxième partie, dont l'opérande est la chaîne elle-même, construit les mémoires traduisant cette chaîne dont la première est étiquetée Hn et contient le nombre de mots alphabétiques qui suivent, et génère pour finir une instruction de reprise du compteur d'assemblage ordinaire (compteur "blanc").

Ainsi, la chaîne

'A B C D E F G H'

se traduira, si la numérotation H est à 3 , par :

```
        AXT     H3, 2
        PXA     ,2
        USE     H
H3      PZE     2
        BCI     1,ABCDEF
        BCI     1,GH
        USE     PREVIOUS
```

CHAPITRE V

SOUS-PROGRAMMES ET SOUS-PHASES DE GENERATION

A - SOUS-PHASES DE GENERATION

A1 - GENERATION POUR ALLER A ET SINON

Deux sous-phases réalisent la génération pour les symboles ALLER A et SINON . Nous en avons vu les fonctions au chapitre précédent (Cf. E) .

A1 - 1 ALLER A : S35

Cette sous-phase est appelée d'une part quand le symbole ALLER A "tombe" de PR (phase P2.S8), d'autre part quand SI ou SINON ont pour opérande une expression de désignation (P2.S6 et P2.S7, P8.S1 , P12). Dans ce cas, le symbole ALLER A est susceptible d'être bloqué en PR par $\downarrow^3$, et quand il tombera il sera inutile : S35 doit donc vérifier que la génération correspondant à ALLER A n'a pas été déjà faite. ALLER A peut également être inutile si l'expression de désignation est un paramètre formel ou un indicateur d'aiguillage.

Enfin, E donne le contrôle à la fin du programme lorsqu'il détecte une unité syntaxique nulle en C.

Si l'opérande en PN est une étiquette calculée, une étiquette formelle ou un indicateur d'aiguillage, le retour de S35 est donc immédiat, et l'opérande est effacé dans PN . (Cependant si l'appel a S35 est fait à partir d'une autre phase que P2.S9, cet opérande est remis dans PN par cette phase).

Si l'étiquette est simple, la génération se fait effectivement, par une instruction du type

TRA E_i

Cependant, si une instruction ALLER A constitue l'instruction inconditionnelle qui suit un ALORS , le test TZE Rj généré par P13, mais non imprimé (Cf. chapitre IV, E2) est transformé en TNZ E_i , et, l'étiquette Rj devenant inutile, PE et COET sont diminués de un . S35 positionne dans ce cas l'aiguillage DP2.S6 à un (Cf. chapitre IV,E3), pour que la génération effectuée quand SI tombe de PR soit compatible avec cette modification.

Dans tous les cas, l'impression de la séquence est faite et le retour se fait en effaçant l'opérande en PN .

A1 - 2 SINON : S36

La sous-phase S36 est appelée par les trois points de génération (deux, huit et douze) où SINON "tombe" de PR . Pratiquement, cette phase regroupe la génération pour SINON dans tous les cas, et en particulier effectue l'appel à S35 si l'opérande en PN est une expression de désignation. Dans ce cas, un opérande est laissé en PN , dont le profil est l'union des profils des deux opérandes traités par S35 et dont le dernier vient d'être "effacé" .

Les autres fonctions de S36 ont été exposées au chapitre précédent (Cf. E4) .

A1 - 3 Exemple complet

Soient les quantités booléennes U et V de symboles internes Q - 1 et Q - 2, les étiquettes simples ETIQ1 et ETIQ2 de symboles E1 et E2 , et les étiquettes formelles EF1 et EF2 , de symboles P1 - 1 et P1 - 2 .

Nous supposerons la numérotation R arrivée à 5 et la mémoire spécifique du bloc où l'on se trouve sera M - 3 .

a) L'instruction suivante :

ALLER A SI U ALORS (SI V ALORS ETIQ1 SINON ETIQ2) SINON EF2 ;

se traduira par :

```
        CLA     Q - 1
        TZE     R5
        CLA     Q - 2
        TNZ     E1
        TRA     E2
R5      CLA     M - 3
        STO     NDYNL
        TSX     P1 - 2,4
        .... etc ...
```

b) L'instruction suivante :

SI U ALORS ALLERA (SI V ALORS EF1 SINON E2) SINON ALLERA EF2 ;

se traduira par :

```
CLA     Q - 1
TZE     R5
CLA     Q - 2
TZE     R6
CLA     M - 3
STO     NDYNL
TSX     P1 - 1,4
        TRA     R7
R6      TRA     E2
R5      CLA     M - 3
        STO     NDYNL
        TSX     P1 - 2,4
R7      .... etc ...
```

où l'on voit qu'il faut prévoir le retour de l'appel à EF1, susceptible d'être équivalent à une instruction vide.

A2 - GENERATION DANS LES INSTRUCTIONS POUR

Les sous-phases relatives aux instructions POUR sont appelées par les différentes phases du point deux correspondantes : P2.S9 et P2.S10 , par les phases d'initialisation P11 et P15 et par la phase relative à $\downarrow^2$(P14). Elles communiquent avec ces phases par l'intermédiaire des indicateurs ETVARC , ETPAS, ETVAF, ETTEST et ETBOU .

S37 , S39 et S41 ont pour rôle de générer respectivement les appels aux sous-programmes de boucle proprement dite (B_k), de variable contrôlée (Ri) et de pas (Vn) .

S38 a pour fonction de générer l'étiquetage de l'instruction de renvoi des éléments de la deuxième forme. L'étiquette dont le profil est en ETTEST est générée sous la forme Jn.

Enfin, S40 génère l'étiquette Ri à laquelle on se transfère pour sauter les sous-programmes du pas et de la valeur finale. Cette étiquette a son profil en PE , qui recule de un ici.

A3 - SOUS-PHASES POUR LES APPELS DE PROCEDURES

La première de ces sous-phases (S30) est appelée par P25 ou P1 au début de la génération d'un appel de procédure.

Les quatre autres (S31 à S34) sont appelées par P7.S2. En plus, la sous-phase S33 est appelée également par P6 et P7.S1 (fin de variable indicée).

Les fonctions de ces sous-phases ont été exposées au I du chapitre IV .

S30 réalise, par l'appel des sous-programmes SP3, SP4, SP11, les sauvegardes éventuelles des registres AC , MQ et X4 .

Si la procédure appelée est déclarée dans P_S , S30 fait appel aussi à SP12 par l'intermédiaire de C , pour réaliser la sauvegarde ou la mise à jour de NDYNL.

Dans tous les cas, la séquence générée est imprimée.

S31 a pour rôle de générer, pour une procédure et un nombre p de paramètres donné , l'instruction d'appel proprement dite à la procédure, que celle-ci soit déclarée ou standard fermée. Rappelons que dans le premier cas, une mémoire du type

PZE ,, p

est générée après l'instruction d'appel (TSL).

S32 et S33 génèrent les listes de paramètres qui suivent respectivement les instructions d'appel aux procédures déclarées et aux procédures standard (et variables indicées).

Les caractéristiques de ces listes ont été données au chapitre IV (Cf. I). Les listes sont imprimées ici, au fur et à mesure de la génération.

Enfin, S34 assure la génération des appels de procédures standard ouvertes dont nous avons vu les caractéristiques au chapitre IV .

Pratiquement, S34 se réduit à la table d'adresses APROU dans laquelle il est nécessaire de prévoir une entrée nouvelle pour toute augmentation de un du nombre NPROU de procédures ouvertes.

B - SOUS - PROGRAMMES DE SERVITUDE ET D'EDITION

Nous avons regroupé dans cette partie tous les sous-programmes assurant la gestion des registres de la machine, les reconnaissances et conversions de types, la sauvegarde ou la mise à jour du niveau de mémoire dynamique, la construction effective des instructions générées, en particulier de leur champ variable, et enfin l'impression de ces instructions sur B_G .

Pratiquement, ces sous-programmes sont appelés par toutes les parties de PC_2 - G .

De plus, certaines séquences du compilateur non décrites plus haut reproduisent avec des variantes certains de ces sous-programmes, en en gardant les propriétés et le mode de fonctionnement.

B1 - SOUS-PROGRAMMES DE GESTION DES REGISTRES

B1 - 1 Rangement des registres AC et MQ (SP3, SP4)

Quand une instruction est susceptible, dans le programme généré P_M , de détruire le contenu des registres AC ou / et MQ , un rangement de ces registres est généré. Ce rangement est effectué dans une mémoire de manoeuvre M - k définie à cet effet (DEFMEM) et n'a lieu que si l'indicateur du registre contient la position 2 à un. Le profil de la mémoire de manoeuvre est alors placé dans la première mémoire de PN, en partant de son sommet, qui con-

tient le profil spécificatif du registre (0 pour AC et 1 pour MQ).

Enfin, l'indicateur du registre est mis à un pour spécifier le registre inconnu.

Les instructions générées sont respectivement

```
      STO      M - k
et    STQ      M - k
```

B1 - 2 Chargement des registres AC et MQ

Les sous-programmes SP1 et SP2 procèdent à la génération de l'appel en AC ou MQ des opérandes de la plupart des opérations décrites au chapitre précédent.

Le profil de l'opérande en cause est placé, avant l'appel à SP1 ou SP2, dans une mémoire spécialisée G.APPEL .

SP1 et SP2 testent le contenu de G.APPEL .

B1 - 2 - 1 Appel en AC : SP1

Si le profil en G.APPEL est différent de zéro ou un, l'opérande n'est pas un résultat. Si l'indicateur IAC contient le même profil, rien n'est généré. Sinon le rangement de AC est commandé (SP3), et on génère l'appel par une instruction CLA . Le profil est transféré de G.APPEL à IAC . Enfin, si l'opérande était une mémoire de manoeuvre, celle-ci est libérée.

Si le profil en G.APPEL est un, l'opérande est un résultat en MQ et on génère le transfert de MQ vers AC , après rangement de celui-ci.

Le profil est transféré de IMQ à IAC, et IMQ est rendu nul, les instructions générées étant

```
ZAC
LLS      35
```

Enfin, si le profil est zéro, l'opérande est un résultat en AC . Si ce résultat n'est pas une adresse (position 15 à zéro dans IAC), rien n'est généré. Sinon, on appelle la quantité correspondant à cette adresse par

```
STA    *+1
CLA    **
```

et la position 15 est effacée dans IAC .

B1 - 2 - 2 Appel en MQ : SP2

Le fonctionnement est symétrique du précédent. Les instructions générées sont

```
LDQ    Quantité
```

si l'opérande a son profil différent de zéro ou un,

```
LRS    35
```

si l'opérande est un résultat effectif en AC ,

et

```
STA    *+1
LDQ    **
```

si c'est un résultat - adresse en AC .

Le rangement de MQ , la libération des mémoires de manoeuvre et les modifications apportées à IAC et IMQ sont symétriques du cas appel en AC .

B1 - 3 Sauvegarde du registre index X4 : SP11

Ce sous-programme est appelé dans les cas suivants :

- Appel de la fonction puissance (P2.S2)
- Appel d'un indicateur d'aiguillage (P7.S3)
- Appel d'une procédure non ouverte (P25, P1) ou d'une variable indicée
- Appel d'un paramètre formel par nom (P16)
- Détection de ALORS (P13)

Son rôle est d'assurer la sauvegarde du registre index X4 dans tous les cas où les conditions ci-dessus surviennent à l'intérieur d'un sous-programme de paramètre formel ou d'expression d'aiguillage.

Afin de déterminer si l'on est dans un de ces cas, on a vu que chacun de ces sous-programmes est considéré comme un bloc (Cf. chapitre II, F3 - F4) pour lequel une mémoire indicatrice est définie. Cette mémoire est contenue au sommet de la pile PB dans le cas d'un paramètre formel et dans IAIG dans le cas d'une expression d'aiguillage. Si cette mémoire contient la position A , c'est que l'on est à l'intérieur d'un sous-programme. Quand la sauvegarde est effectuée par SP11 , la position C est ajoutée à la mémoire indicatrice, tout autre appel à SP11 ne donnant alors lieu à aucune génération tant que C n'aura pas été effacée.

Remarquons que SP11 doit tester la présence de A d'abord dans la dernière mémoire de PB , une expression d'aiguillage pouvant donner lieu à des sous-programmes de paramètres alors que l'inverse est impossible.

On a déjà vu que les instructions générées pour la sauvegarde étaient

	SXA	N_r , 4
ou	SXA	I_j , 4

N_r et I_j étant respectivement construites à partir des profils de L_r pour les paramètres formels, contenu en PEPE et de K_j pour les aiguillages, contenu en PEA . Remarquons enfin que malgré l'existence de la mémoire indicatrice de bloc, la sauvegarde générée est susceptible d'être rendue inutile si le sous-programme est la traduction soit d'une expression conditionnelle où n'apparaît aucun appel détruisant effectivement X4, soit d'une expression de désignation ne pouvant pas être équivalente à une instruction

vide. Rappelons que dans ce cas, l'instruction de retour du sous-programme n'est pas générée mais que N_r ou I_j sont tout de même définis par une pseudo-instruction d'équivalence avec la mémoire banale A .

B2 - SOUS-PROGRAMMES DE RECONNAISSANCE DE TYPES ET DE CONVERSIONS :

SP5 à SP10 ;

B2 - 1 Reconnaissance des types

Les sous-programmes SP5 et SP10 ont pour rôle de reconnaître les types d'opérandes dont on leur fournit les profils en PN et éventuellement de faire donner le contrôle à un sous-programme d'erreur si ces types sont incompatibles entre eux ou avec l'opération en cause.

SP5 possède deux entrées suivant que le nombre d'opérandes est un ou deux. On tient compte du fait que ces opérandes peuvent être des résultats en AC ou MQ, leur véritable profil étant alors en IAC ou IMQ.

SP10 vérifie la compatibilité des types entre les variables contrôlées d'instructions POUR et les expressions constituant les éléments de la liste.

Des conversions sont éventuellement générées par appels à SP6 ou SP8 .

B2 - 2 Conversion des types

Les sous-programmes SP6 à SP9 réalisent la génération, par appel des macros - instructions CONVER ou CONVRE , des conversions arithmétiques de types entre deux opérandes dont les profils sont en PN .

On fait appel à SP1 pour générer l'appel en AC du (des) opérande (s) à convertir, et la séquence générée est immédiatement imprimée.

AC et MQ sont spécifiés inconnus au retour, un rangement de MQ ayant été éventuellement généré.

B3 - SAUVEGARDE ET MISE A JOUR DE NDYNL : SP12

Ce sous-programme est appelé, par l'intermédiaire du sous-programme C de PC_2 - A (Cf. chapitre II, F2 - 2) dans les cas suivants :

- Appel d'une procédure déclarée dans P_S (P25, P1)
- Appel d'un paramètre formel par nom (P16)
- Appel d'un indicateur d'aiguillage (P7.S3)
- Détection de ALORS (P13)

Pratiquement, cet appel est toujours précédé de l'appel au sous-programme SP11 (Cf. B1 - 3) auquel on peut comparer le fonctionnement de C .

On a vu que celui-ci, en effet, utilisait la mémoire indicatrice de bloc pour déterminer s'il fallait ou non générer une sauvegarde ou une mise à jour de NDYNL .

Pratiquement, SP12 assure la génération de

- La mise à jour de NDYNL par la mémoire spécifique (PMB) du bloc où l'on se trouve si celui-ci est un bloc naturel et que DC n'est pas à un (appel survenant de P13).

- La sauvegarde de NDYNL dans une mémoire spécifique si l'appel survient pour la première fois dans un sous-programme de paramètre formel ou d'aiguillage (bloc artificiel, position A en PB ou IAIG).
- La mise à jour de NDYNL pour tout appel autre que le premier dans un bloc artificiel (positions A et C dans PB ou IAIG), et si DC est à zéro.

Les instructions générées sont, si M - k est la mémoire spécifique du bloc où l'on se trouve :

```
LDQ     NDYNL
STQ     M - k
```

pour la sauvegarde de NDYNL, et

```
CLA     M - k
STO     NDYNL
```

pour sa mise à jour.

B4 SOUS-PROGRAMMES D'EDITION

Un premier groupe de sous-programmes, regroupés ici en SP14, assure la construction des mots alphabétiques représentant les instructions générées et leur rangement en Z_G , d'index g .

Un autre sous-programme SP15 génère les parties numériques apparaissant dans les pseudo-instructions de réservation de mémoires.

Enfin, le groupe SP16 assure l'impression de la zone Z_G , sa réinitialisation, et la génération éventuelle de pseudo-instructions EQU entre des étiquettes adressant la même instruction.

B4 - 1 Construction des instructions : SP14

Plusieurs sous-programmes assurent la construction et le rangement en Z_G des instructions générées.

1°) Si une instruction générée est constituée d'un code opération associé à un champ variable indépendant des opérandes de P_S , on fait appel à l'une de six macros - instructions, suivant la longueur du champ variable et l'emplacement de l'appel. Ces macros - instructions sont autant de sous-programmes ouverts à qui l'on donne en paramètres les zones alphabétiques à générer et à ranger en Z_G . Chaque appel construit une ligne (une instruction) de Z_G et g est automatiquement augmenté de un.

2°) Une autre macro - instruction fait appel à la sous-phase fermée SP14 - S1 quand une instruction doit être construite dont la champ variable dépend de plusieurs paramètres, en particulier du profil de l'opérande sur lequel porte l'opération.

Le code opération est généré et rangé comme précédemment, puis c'est SP14.S1 qui construit effectivement les symboles internes dans les champs variables d'instruction, tels que Q - i , M - k , U - j , Fn + k , etc ...

Pour cela, une mémoire qualitative CAQCV reçoit le profil de l'opérande à construire, et plusieurs indicateurs sont susceptibles d'être positionnés si le champ variable à construire comporte des particularités : par exemple, l'indicateur ICVX2 est rendu négatif si le champ variable doit comporter en zone index la spécification du registre X2 .

Au retour de SP14.S1, ces indicateurs sont restaurés, mais CAQCV reste inchangé.

En plus, SP14.S1 est susceptible de modifier le code opération construit, en lui ajoutant la marque d'adressage indirect si CAQCV contient le profil d'un paramètre formel appelé par nom ou celui d'une mémoire de manoeuvre contenant une adresse (positions 15 à un).

Une ou plusieurs instructions sont susceptibles d'être construites par un même appel de SP14.S1 . L'index g est automatiquement mis à jour.

3°) La même macro - instruction fait appel à la sous-phase SP14.S2 dans le cas où un symbole interne du type étiquette est à construire (E_i , R_i , V_n , J_n , P_n , etc ...).

Pratiquement, un indicateur ICVE est rendu négatif pour appeler SP14.S2 au lieu de SP14.S1.

L'étiquette est construite grâce au profil placé en CAQCV et à un indicateur IQET dont les valeurs caractérisent le type générique du symbole dans le cas d'une étiquette de manoeuvre.

Si le symbole est à générer en champ variable (en général, ce cas correspond à un code opération de transfert), le fonctionnement est identique à celui de SP14.S1.

En particulier, si l'on trouve en CAQCV le profil d'une fonction standard, le champ variable caractéristique de cette fonction est construit grâce à la table TAFST et l'indicateur correspondant dans BLFST est mis à un (Cf. chapitre IV, I).

Les mêmes indicateurs que dans SP14.S1 servent à la génération de champs variables particuliers. De même, ces indicateurs sont restaurés au retour, et g automatiquement mis à jour.

Si le symbole est à générer en champ symbolique, un indicateur spécial est mis à un avant l'appel à SP14.S2 , et la première mémoire de la prochaine ligne de Z_G à construire reçoit le

le symbole alphabétique interne traduisant l'étiquette. Compte tenu de la configuration de ces symboles (1 lettre suivie de chiffres), 99999 symboles du même type générique pourraient être construits, limite bien supérieure au nombre d'instructions qu'il est possible de générer.

Comme en SP14.S1, les indicateurs utilisés sont restaurés, mais g reste ici inchangé de manière à ce que la prochaine instruction construite contienne en champ symbolique l'étiquette générée .

Ce fonctionnement interdit cependant de générer une étiquette si la première mémoire de la prochaine ligne de Z_G n'est pas vide : c'est le cas du multi-étiquetage que nous avons déjà évoqué (Cf. chapitre IV, D). Dans ce cas, l'ancienne étiquette et la nouvelle sont mises en réserve dans une zone spéciale H_G qui sera vidée par SP16 (Cf. B4 - 3).

B4 - 2 Construction des champs variables numériques : SP15

Quand le champ variable à construire est une partie numérique (cas des réservations de mémoires en particulier), le sous-programme SP15 est appelé par l'intermédiaire de deux macros - instructions spécialisées.

SP15 prend en CAQCV le profil du champ variable, ici un nombre, et le transforme en partie alphanumérique, qu'il range en Z_G . La partie code opération est générée directement par PC_2 - G qui assure également la progression de g .

B4 - 3 Impression des instructions : SP16

L'index g de Z_G étant à zéro au départ et augmenté de un à chaque ligne construite dans Z_G , SP16 a pour rôle, à chaque appel, de transférer les g premières lignes de Z_G dans les zo-

nes intermédiaires d'entrée - sortie du fichier B_G .

Z_G est ensuite réinitialisée, c'est à dire remplie par des caractères "espace " .

Puis, afin de tenir compte de la présence éventuelle d'étiquettes en attente dans H_G (Cf. B4 - 1), on procède également à l'impression de cette zone, sous-forme de pseudo-instructions EQU, à raison d'une pseudo EQU pour chaque couple d'étiquettes de H_G. Celle-ci est restaurée (mémoires nulles) avant que SP6 ne rende le contrôle aux phases de PC_2 - G , en réinitialisant g à zéro.

CHAPITRE VI

FIN DE GENERATION

La phase de fin de génération PFG a trois fonctions principales :

1°) Vérifications diverses de fin de compilation et génération d'une instruction terminant la partie exécutable de P_M

2°) Génération des constantes et réservations de mémoires utiles à P_M .

3°) Génération des instructions assurant le chargement en mémoire des sous-programmes désirés au moment de l'exécution de P_M , et phase terminale du compilateur.

A - VERIFICATIONS ET GENERATION DES CONSTANTES

A1 - ETAT FINAL DU COMPILATEUR - INSTRUCTION DE SORTIE

Si aucun sous-programme d'erreur n'est appelé au cours de l'analyse de P_S, le compilateur PC_2 doit être dans un état "final" quand la phase PFG est atteinte.

Cet état est caractérisé par le fait que toutes les piles doivent être vidées, que certains niveaux doivent être revenus à zéro, etc ...

On vérifie ici que les piles PN,PR,PNIM, et PPTA ont leurs index revenus à zéro, et que NIM lui-même est revenu à un, c'est à dire que toutes les mémoires de manoeuvre sont libérées. Suivant les cas, un libellé d'erreur approprié ou un diagnostic spécifiant une erreur du compilateur lui-même sont donnés à l'utilisateur si l'état final de PC_2 n'est pas atteint.

Puis est générée une instruction de transfert à la partie système, qui marque la fin de la partie exécutable de P_M. Cette instruction traduit, dans P_M, l'intégration du compilateur dans le système général de programmation (Cf. troisième partie), et assure en particulier, dans le cadre de cette intégration, le passage d'un "travail" à un autre (TRA S.JXIT sur machine 7040/44).

A2 - RESERVATIONS DE MEMOIRES ET CONSTANTES

Nous avons évoqué (Cf. chapitre II, F2) le fonctionnement du compteur CIM de maximum de mémoires de manoeuvres utilisées dans le programme P_M . De la même façon sont définis des compteurs de quantités et de tableaux non rémanents. Les compteurs de quantités et de tableaux rémanents sont, eux, directement fournis par PC1 (Cf. Première partie, chapitre II, G5).

Les valeurs atteintes par ces compteurs sont successivement fournies à SP15 (Cf. chapitre V, B4 - 2) pour générer les pseudo - instructions de réservations de mémoires correspondantes.

Ces pseudo - instructions sont du type BES et sont étiquetées respectivement par M, Q, U, R et V.

Puis sont générées les contantes susceptibles d'être utilisées par P_M .

La table des nombres purs TNP est d'abord imprimée, sa première mémoire (zéro) se traduisant par

N PZE

et les suivantes par des pseudo - instructions du type

OCT

Ensuite viennent les valeurs logiques VRAI et FAUX , représentées respectivement par RBOO et JBOO. La première est générée sous la forme du nombre 1 , la seconde est équivalente à N .

Enfin, on construit cinq constantes utiles aux programmes générés dans diverses séquences de conversions (Cf. AAA, BBB, etc ... , chapitre III).

B - PHASE TERMINALE

Les dernières lignes construites par PC_2 - G sont des pseudo - instructions du type EXTERN [27] , servant, à l'exécution de P_M , à assurer le chargement en mémoire rapide de tous les sous-programmes nécessaires.

Une série d'indicateurs, parmi lesquels ceux du bloc BLFST (Cf. SP14, chapitre V, B4 - 1), sont mis à un au cours de la compilation quand le sous-programme correspondant est utilisé dans P_M .

L'encombrement de ces sous-programmes en mémoire est ainsi rendu minimum.

Sont générées successivement des pseudo EXTERN pour

- Toutes les fonctions standard dont les indicateurs dans BLFST sont à un.
- ALPRR et ALDST (ou ALPRO) si l'option RECUR était présente (ou non) (Cf. chapitre IV, H2 - 2).
- PUAL si l'opérateur ↑ a été trouvé dans P_S (Cf. chapitre IV, B2).
- TABD et TABE si des tableaux ont été déclarés dans P_S (Cf. chapitre IV, J).
- ALVTA si des tableaux sont appelés par valeur dans une procédure (Cf. chapitre IV, H2 - 3).
- INEX et DINEX si l'option VERIF était présente (Cf. chapitre III, B1 et chapitre IV,B3 - 4).

Enfin, on génère une dernière ligne marquant la fin des instructions de P_M qui doivent être assemblées par IBMAP [27] et donnant à celui-ci l'adresse A de la première instruction exécutable de P_M .

Cette ligne est de la forme

END A

La phase terminale de PC_2 consiste alors à imprimer les dernières lignes générées, à vider les zones tampons du fichier B_G , et à passer le contrôle à l'assembleur IBMAP, (Cf. troisième partie).

CHAPITRE VII

SOUS-PROGRAMME D'ERREUR

A - GENERALITES

Les sous-programmes d'erreurs du compilateur PC_2 sont tous du type "fatal" , c'est à dire que toute erreur décelée au cours de la compilation entraîne l'arrêt du traitement et le retour au système général, de la même façon que certaines erreurs décelées par PC_1 (Cf. première partie, chapitre III).

Pratiquement, les programmes d'erreurs de PC_2 fournissent au système général un numéro de diagnostic et positionnent un indicateur à un pour que le libellé correspondant soit imprimé avant qu'un autre "travail" ne soit commencé.

Ces diagnostics, au nombre de douze actuellement, et que nous avons à plusieurs reprises évoqués, sont de deux types :

1°) Certains correspondent à des erreurs "cherchées" par le compilateur, afin de s'assurer que le programme P_S respecte les structures syntactique et sémantique du langage source.

Dans ce groupe entrent en particulier les vérifications faites sur la compatibilité de types entre opérandes et opérateur d'une même opération.

2°) Un autre groupe est relatif à des erreurs commises par rapport aux conventions et limitations adoptées dans le compilateur.

La recherche systématique de ces erreurs sert plutôt à s'assurer que PC_2 peut continuer son travail sans risquer de créer d'autres erreurs qui seraient, elles, indécelables.

B - ERREURS SYNTACTICO-SEMANTIQUES

Sept diagnostics entrent dans la première catégorie. Leur apparition est significative d'un manquement aux règles de syntaxe ou de sémantique du langage source L_S [1] .

B1 - "ERREUR SYNTACTICO-SEMANTIQUE"

Ce libellé apparait dans tous les cas où une erreur est décelée, telle que le traitement de P_S est obligatoirement ambigu vis à vis des opérations à générer.

EXEMPLES :

- Si un identificateur de tableau est exploité sans expression en indice (en dehors du cas paramètre effectif).
- Si une procédure standard ouverte est appelée avec plus d'un paramètre effectif.
- Si une procédure ouverte de conversion est appelée avec un paramètre dont le type n'est ni réel ni entier.
- Si un paramètre effectif n'a pas une forme syntaxique autorisée.
- Si une partie valeur porte sur autre chose qu'une quantité simple ou un tableau, etc ...

B2 - "ERREUR TYPE OU SEMANTIQUE"

Cette erreur est diagnostiquée quand le type d'un opérande est incompatible avec celui d'un autre opérande ou de l'opérateur associé.

EXEMPLES :

- Si une variable contrôlée d'instruction POUR est de type booléen et qu'un élément est réel ou entier.
- Cas inverse.
- Si une expression en indice est de type booléen
- Si un opérateur unitaire autre que ¬ est associé à un opérande de type booléen
- etc ...

B3 - "EXPRESSION NON BOOLEENNE APRES SI"

Ce libellé correspond au cas où un symbole SI est suivi d'une expression dont le type est réel ou entier, ou n'est pas suivi d'une expression.

EXEMPLE :

-Si l'on écrit SI A := B ALORS etc ...

au lieu de SI A = B ALORS etc ...

B4 - "OPERANDE DIVISION ENTIERE TYPE REEL" .

On a vu (Cf. chapitre IV, B3 - 4) que l'opérateur ÷ devait obligatoirement être associé à des opérandes du type entier.

B5 - "ERREUR SYMBOLE DE BASE "

Cette erreur correspond au cas où le compilateur ne trouve pas un symbole de base qu'il devrait trouver, compte tenu de la structure syntaxique en cours d'analyse.

EXEMPLES :

- Si la première expression d'un élément de liste de POUR est suivie d'un symbole autre que "," , PAS , TANTQUE ou FAIRE .
- Si "," est directement suivie de ";" dans une déclaration
- Si une partie valeur n'est pas suivie par ";"
- Id. pour partie spécification.
- Si au moment d'analyser FIN,),] , SINON, FAIRE, on ne trouve pas au sommet de PR les symboles DEBUT, (ou ↓,↓ou ↓* , ↓3 , ↓5 ou ↓2 .
- Etc ...

B6 - "EXPRESSION INCOMPLETE"

On a vu que ce libellé provenait de la vérification, effectuée au traitement de ";" (et en fin de compilation), du fait que toutes les mémoires de manoeuvre utilisées dans une instruction (un programme) sont libérées et réutilisables.

B7 - "NOMBRE PUR TROP GRAND "

Ce diagnostic provient du calcul par PC_2 - A d'un nombre pur dont la valeur s'avère dépasser la capacité d'un mot mémoire. Rappelons que cette capacité est 2^{27} pour les nombres du type entier et $\sim 10^{38}$ (ou 10^{-38}) pour ceux du type réel.

C - ERREURS DE CONVENTIONS

OU D'ENCOMBREMENT

Ces erreurs sont telles que le compilateur ne peut plus continuer le traitement de P_S sans entrainer d'autres erreurs.

Ce sont :

C1-"DEPASSEMENT DE CAPACITE DES PILES"

L'encombrement de chaque pile est un paramètre d'assemblage de PC_2 . Le non dépassement de cet encombrement est vérifié à chaque incrémentation de l'index d'une pile.

C2 - "TROP D'EQUIVALENCES"

Ce libellé est imprimé si trop d'étiquettes sont associées à la même instruction générée. Il y a alors dépassement de l'encombrement de la zone H_G .

C3 - "TROP DE NOMBRES PURS"

De la même façon, la table des nombres purs TNP peut être pleine alors que l'on doit calculer un nouveau nombre pur.

C4 - "ERREUR COMPILATEUR" OU "ERREUR MACHINE"

Un de ces deux libellés est imprimé si une erreur est décelée, et qu'aucune raison ne peut être donnée du mauvais fonctionnement du compilateur.

CHAPITRE VIII

CONCLUSION - DISCUSSION

A - CONCLUSION

En conclusion de cette partie, nous rappelons que la méthode de génération adoptée dans notre compilateur est intimement liée à la forme et au fonctionnement des machines intéressées.

Cependant cette étude nous permet d'affirmer que le langage ALGOL est facilement compilable par la méthode d'analyse à production en notation postfixée et que la transformation dans le langage d'une autre machine peut s'en faire - avec plus ou moins d'efficacité - en mettant en oeuvre des algorithmes simples qui ont été décrits ici, réalisant en particulier l'optimisation complète du nombre de mots mémoires nécessaires au programme objet P_M .

Nous nous permettons d'insister sur le fait qu'une grande partie de cette étude a été réalisée de façon empirique.

En particulier, la table de hiérarchies des opérateurs, le fonctionnement des indicateurs de registres et la construction de l'algorithme de gestion de la mémoire dynamique ont été modifiés plusieurs fois avant d'être appliqués définitivement au compilateur.

Nous allons voir que ces modifications, loin de conduire à un ensemble intangible, sont susceptibles d'être suivies de plusieurs autres, dans la seule partie PC_2 , relatives à des parties du compilateur dont le mode d'écriture ou l'efficacité peuvent être améliorées.

B - D I S C U S S I O N

Sans vouloir exposer des projets importants relatifs à la génération des programmes objets, dont nous donnerons un aperçu en conclusion générale, nous donnerons ci-dessous une liste de modifications qu'il nous paraît souhaitable d'apporter, en particulier à la partie PC_2 - G . Nous verrons également que la détection des erreurs à l'intérieur de PC_2 peut être rendue plus efficace et plus précise.

B1 - MODIFICATION DE L'ETIQUETAGE DES INSTRUCTIONS

Toute étiquette générée ne trouvant pas de place dans Z_G est transférée actuellement dans la zone intermédiaire H_G dont la limite d'encombrement crée une contrainte inutile à l'utilisateur. Il est possible de faire disparaître cette contrainte en générant, dès que l'on trouve une étiquette R_S , une pseudo-instruction

R_S EQU *

Cette modification peut s'accompagner de la mise en oeuvre d'une méthode de génération intéressant les instructions de transfert (TRA).

On remarque en effet que toute instruction du type

R_n TRA R_s

générée actuellement peut être transformée, grâce aux nouvelles possibilités de l'assembleur MAP , en

R_n EQU R_s
TRA R_s

qui permet d'optimiser le nombre de transferts effectifs dans P_M .

B2 - GENERATION RELATIVE AUX OPERATEURS DE RELATION

Les séquences générées actuellement quand un opérateur de relation "tombe" de PR peuvent être modifiées de façon à réduire notablement le nombre d'instructions dans tous les cas où une comparaison est effectuée dans laquelle intervient le nombre pur zéro et en particulier quand le résultat de cette opération est l'opérande d'un symbole ALORS .

En plus, la technologie des machines intéressées est telle [17, 26] qu'un nombre nul peut comporter le signe - et que dans ce cas il est considéré par les séquences générées (instruction CAS) comme différent du nombre zéro. La modification envisagée permettrait de supprimer cette anomalie.

B3 - PARTICULARISATION DU NOMBRE ZERO

Actuellement, l'instruction

A := 0

où A est la variable entière Q - 1 , se traduit par

```
LDQ        N + 0
STQ        Q - 1
```

Si A est de type réel, une conversion inutile est générée en plus.

On peut envisager une particularisation du nombre pur zéro telle que ce nombre ne subisse pas de conversion et que des instructions du type STZ soient employées dans des cas comme ci-dessus. Cette particularisation est rendue facile par le fait que le profil qualitatif du nombre zéro est toujours reconnaissable par le compilateur.

B4 - EMPLOI DES COMPTEURS D'ASSEMBLAGES

Les possibilités de l'assembleur IBMAP relatives à l'emploi de compteurs d'assemblages en nombre illimité permettent d'envisager la génération systématique d'instructions du type USE et BEGIN partout où cela est possible dans P_M (boucles d'instructions POUR et sous-programmes de variable contrôlée, de pas et de valeur finale, procédures déclarées, paramètres effectifs, aiguillages,etc ...).

Cette modification entrainerait dans certains cas une grande simplification dans la compilation et une réduction de sa durée, et dans tous les cas une optimisation des programmes générés.

B5 - SOUS-PROGRAMMES D'ERREURS

Afin de rendre plus efficace et plus précise la détection d'erreurs par PC_2 , on peut envisager deux types de modification dans le fonctionnement actuel des sous-programmes d'erreur (Cf. chapitre VII).

La première amélioration consisterait à multiplier le nombre de libellés différents, ceux-ci pouvant alors préciser exactement le type d'erreur décelée. Actuellement, douze libellés différents sont imprimés par cent soixante appels aux sous-programmes d'erreurs, disséminés dans PC_2 .

Une deuxième amélioration consisterait à réduire au minimum le nombre d'erreurs "fatales". Cette modification déjà réalisée dans PC_1 (Cf. première partie, chapitre III), augmenterait considérablement l'efficacité de la détection d'erreurs.

- TROISIEME PARTIE -

I N T E G R A T I O N D A N S L E S Y S T E M E

Nous nous proposons d'étudier dans cette partie les fonctions et propriétés du compilateur PC qui font de celui-ci un outil pratique et efficace s'intégrant facilement dans le système général de programmation IBM .

Après quelques rappels sur le fonctionnement de ce système, nous verrons les avantages que nous pouvons espérer trouver à cette intégration et la manière dont elle a été effectuée. Nous verrons également comment, grâce à elle, l'utilisateur peut jouer un rôle actif dans la compilation et l'exécution des programmes sources.

Puis un chapitre sera consacré à la description des programmes "bibliothèque" ajoutés à ceux existant déjà dans le système.

Enfin, nous verrons comment les possibilités des versions les plus récentes du système général sont susceptibles d'apporter à l'ensemble compilateur - système une efficacité et une souplesse accrues.

CHAPITRE I

FONCTIONS DU SYSTEME

A - ETUDE GENERALE

A1 - RAPPEL DES PROPRIETES

L'intégration du compilateur a été réalisée successivement sur la machine 7090/94 sous le système de programmation IBSYS - FORTRAN II, Version 3 [24, 25] et sur machine 7040/44 sous le système IBSYS/IBJOB [29, 30] .

Bien que celui-ci soit beaucoup plus évolué que celui-là, les mêmes fonctions essentielles peuvent être dégagées dans les deux systèmes :

1°) Un système est pourvu d'un "moniteur" de base, qui assure d'une part les liaisons entre les différentes phases du système appelées à compiler, assembler, charger en mémoire et exécuter le même "travail", d'autre part le passage d'un travail à un autre.

La notion de "travail" est elle-même définie comme étant la succession en nombre et en ordre quelconques d'applications des composants du système sur des données groupées sous une même dénomination (un nom de personne en général).

2°) La propriété essentielle des systèmes en cause est de pouvoir faire exécuter un nombre illimité en principe de travaux successifs sans intervention d'un opérateur extérieur.

3°) Les systèmes sont dotés d'un "superviseur" général des fonctions d'entrées - sorties, qui est tel qu'aucune référence absolue à une unité physique d'entrée - sortie n'est nécessaire à l'intérieur des phases gérées par le moniteur de base.

4°) Enfin, un système tel que ceux que nous avons cités est susceptible de se transformer lui-même, à l'aide d'un "éditeur", qui est une phase particulière reconnue par le moniteur de base et capable en particulier de faire reconnaître par celui-ci une nouvelle phase du système.

A2 - AVANTAGES DE L'INTEGRATION

Des propriétés des systèmes de programmation, en particulier de celles énumérées ci-dessus, il découle que l'intégration d'un compilateur du langage source L_S dans l'un de ces systèmes augmente considérablement la facilité d'emploi de ce langage par l'utilisateur et la souplesse de fonctionnement d'un centre de calcul doté d'un tel système.

La facilité d'emploi du compilateur est caractérisée

1°) par le petit nombre d'informations extérieures aux programmes P_S à fournir à la machine par l'utilisateur, le moniteur de base et le superviseur d'entrées - sorties réalisant les fonctions de gestion nécessaires à chaque "travail".

2°) Par la présence en "bibliothèque" d'un grand nombre de programmes standard universels dont une seule occurrence dans P_S permet d'obtenir le chargement en machine par le moniteur de base.

L'efficacité de cette réalisation est dûe en particulier à la possibilité d'enchaîner sans intervention de l'utilisateur un nombre théoriquement illimité de "travaux" exprimés en langa-

ge source L_S .

Le fait de pouvoir, dans cet enchaînement mélanger des travaux faisant appel à des phases entièrement différentes du système caractérise la souplesse d'emploi de l'ensemble.

En plus, il faut noter que la possibilité offerte à l'utilisateur d'écrire dans un programme ALGOL des procédures en code MAP augmente encore cette souplesse.

Nous verrons à ce sujet, en conclusion de cette partie, que souplesse et efficacité peuvent encore être accrues en tenant compte des possibilités de la phase "chargeur" des systèmes les plus évolués du type IBJOB .

A3 - REALISATION DE L'INTEGRATION

Afin de répondre aux impératifs donnés par la plus grande efficacité possible de l'ensemble du système, l'intégration du compilateur à cet ensemble s'est faite sur la base de la plus grande analogie avec les compilateurs du type FORTRAN existant déjà sur les machines étudiées.

Cette analogie a porté spécialement sur le contenu et la forme des informations échangées par l'utilisateur avec le moniteur du système sur les deux types de machine.

Cette méthode nous a paru donner un gage d'efficacité suffisant puisqu'elle évitait pratiquement à l'utilisateur de "réapprendre" le fonctionnement du système avant de donner à la machine des travaux comportant des programmes écrits en ALGOL .

En plus, cette analogie nous a permis de faire bénéficier très rapidement l'utilisateur des avantages que nous avons soulignés précédemment.

Pratiquement, dans chacun des deux systèmes étudiés, l'échange d'information entre compilateur et moniteur de base se décompose de la façon suivante :

Au commencement d'un travail :

- Reconnaissance par le moniteur d'un travail à communiquer au compilateur.
- Initialisation à cet effet des indicateurs de liaison entre moniteur et compilateur.
- Reconnaissance par le moniteur des paramètres de travail (options) donnés par l'utilisateur. Transfert au compilateur de l'information qui lui est réservée dans ces paramètres.
- Mise en route du compilateur et positionnement adéquat des indicateurs de liaison.

A la fin du traitement par PC :

- Retour au moniteur avec positionnement des indicateurs de liaison. Ces indicateurs indiqueront si le traitement s'est bien déroulé ou si au contraire une erreur a été décelée dans P_S .
- Dans le cas erreur, passage au travail suivant avec impression d'un message approprié.
- Dans le cas normal, passage à la phase assembleur du système avec transfert à cette phase de l'information qui l'intéresse dans les paramètres donnés par l'utilisateur.
- Enfin, qu'il y ait exécution ou non du programme généré P_M , conservation jusqu'au travail suivant de l'information spécifiant au moniteur de base qu'un travail ALGOL vient d'être traité.

Ces échanges d'informations se font, dans tous les cas, par l'intermédiaire d'une partie du système appelée "noyau" appelée à occuper la partie basse de la mémoire rapide de la machine durant l'exécution d'une même suite de travaux.

Cependant, afin de ne pas encombrer la mémoire par des parties de programme inutiles, il est apparu nécessaire de créer dans le compilateur une troisième partie, que nous nommerons ici "moniteur ALGOL" ou PC_0 qui outre les fonctions précédentes, se charge d'enchaîner le fonctionnement des deux autres parties PC_1 et PC_2 .

La plus grande partie de PC_0 est constituée par un "enregistrement" spécial de l'unité physique comportant le système. PC_1 et PC_2 forment deux autres enregistrements de cette unité, placés à la suite de PC_0 . Des séquences de PC_0 disséminées dans PC_1 et PC_2 assurent le passage d'une phase à l'autre, par l'intermédiaire du moniteur de base, et réduisent au minimum les mouvements de l'unité portant le système.

A3 - 1 Intégration sous IBSYS

Compte tenu de la configuration du système IBSYS - FORTRAN II - V3 sur machine 7090/94 (Cf. planche 15), l'intégration du compilateur s'est faite sur cette machine directement sous IBSYS.

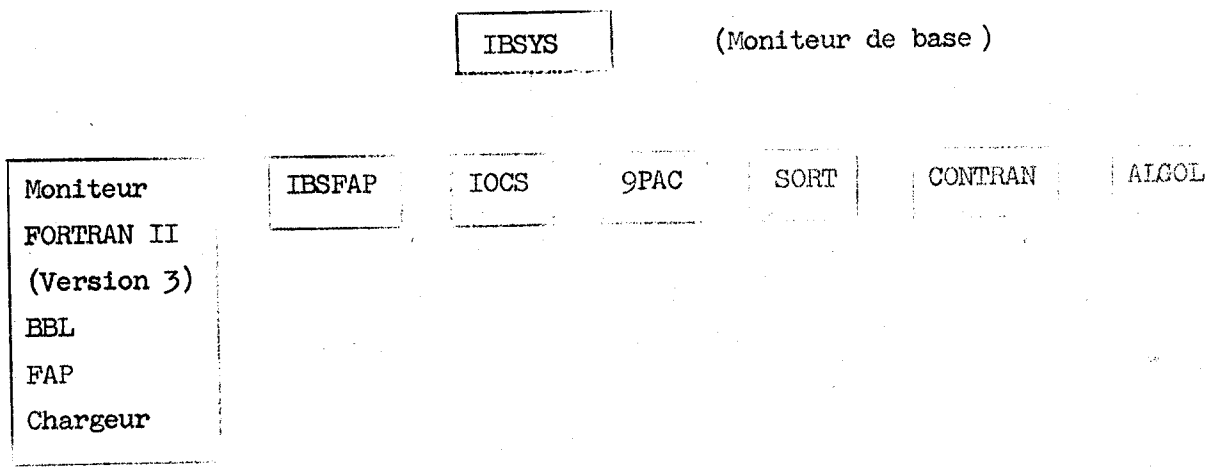


Planche 15 - SYSTEME IBSYS (Schéma)

le passage du compilateur ALGOL à la phase assembleur FAP se fait en simulant sur le fichier B_G un "travail" à exécuter par le système FORTRAN. C'est alors le moniteur de celui-ci qui prend le contrôle.

Des indicateurs sont positionnés dans le noyau afin de spécifier le fichier B_G comme unité d'entrée, et d'assurer le retour à PC_0 après l'exécution du travail.

Si ce retour est impossible (sortie par erreur à l'exécution), un travail fictif PASSIBS, généré sur B_G, réalise le transfert à PC_0.

Le moniteur ALGOL comprend lui-même trois entrées, caractérisées par les valeurs d'un indicateur du noyau.

La première entrée est employée quand le contrôle est donné à PC_0 par le moniteur de base. PC_0 génère alors sur B_G les instructions de simulation d'un travail FORTRAN.

La deuxième entrée est employée au retour du compilateur PC_2 et assure le passage à l'assembleur après génération du travail fictif PASSIBS, spécification de B_G comme bande d'entrée et de FORTRAN comme phase courante.

Au retour du programme P_M, par le programme EXIT de FORTRAN (transformé) ou par PASSIBS en cas d'erreur, l'unité d'entrée est à nouveau changée et la phase spécifiée est ALGOL.

Enfin, la troisième entrée est utilisée en cas d'erreur à l'édition ou à la compilation. Un message d'erreur est alors imprimé sur l'unité physique de sortie, et PC_0 perd le contrôle au profit du moniteur de base.

Remarquons que, dans ce cas, il est défini dans PC_0 un programme d'impression de l'heure sur l'unité de sortie par interrogation du nombre de cycles de l'une des unités électromécaniques connectées à la machine.

Ce programme est activé en début et fin de compilation (première et deuxième entrée de PC_0)

On trouvera en annexe la forme des informations à donner au moniteur de base ("cartes contrôles") pour exécuter un programme ALGOL sous ce système.

A3 - 2 Intégration sous IBJOB *

La configuration du système de programmation disponible sur les machines 7040/44 nous a permis de mettre au point une forme plus évoluée de l'intégration du compilateur.

En particulier, la grande souplesse du sous-système IBJOB nous a permis d'établir un parallèle complet entre le compilateur ALGOL et ceux déjà existant pour FORTRAN et COBOL.

Pratiquement, la liaison après compilation avec le système général se fait maintenant directement au niveau de l'assembleur MAP, le moniteur situé dans IBJOB remplaçant les parties communes des moniteurs FORTRAN et ALGOL de la version sous IBSYS. (Cf. planche 16).

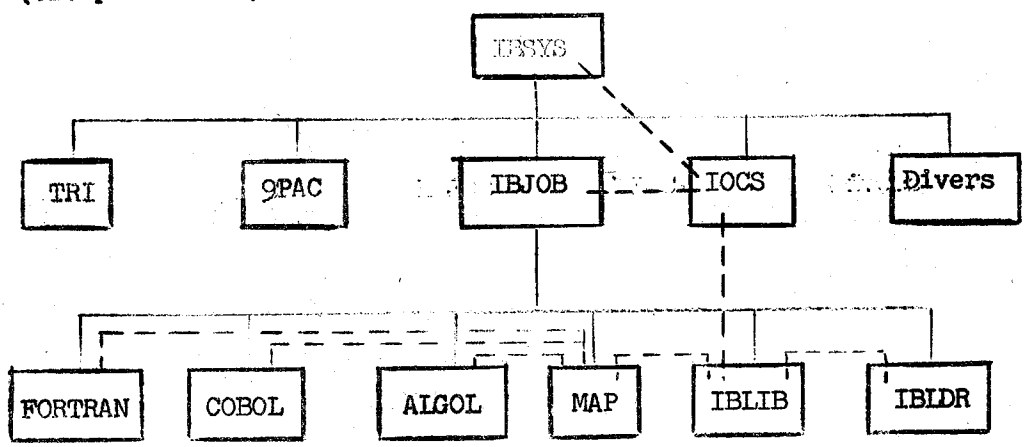


Planche 16 - SYSTEME IBSYS/IBJOB (Schéma)

* Cette partie a été écrite et mise au point par A. AUROUX [35]

En plus, les possibilités de l'ensemble IOCS sont exploitées ici, aussi bien dans le compilateur que dans les programmes générés.

Le moniteur ALGOL existe toujours mais ne comporte plus que deux entrées:

la première quand un nouveau travail ALGOL doit être pris en charge, la seconde quand le compilateur (PC_1 ou PC_2) détecte une erreur dans un programme source P_S .

Un indicateur est alors positionné et un message, qui peut dépendre d'un numéro d'erreur (Cf. deuxième partie, chapitre VII) est imprimé.

Dans les deux cas, le rôle de PC_0 est d'assurer la liaison avec le moniteur IBJOB et les phases qui sont sous son contrôle. En particulier PC_0 est chargé, dans sa première entrée, d'une part de simuler dans une partie réservée du noyau, une "carte contrôle" C_M d'appel à IBMAP (Cf. deuxième partie, chapitre I, A3), d'autre part de positionner les indicateurs d'options I_S après interprétation de l'information donnée par l'utilisateur.

La carte C_S qui suit FALGOL dans le programme P_S est mise en réserve par une séquence de PC_0 placée dans PC_1 . Cette carte est générée sur B_G à la suite du programme P_M par une séquence de PC_0 placée dans la phase terminale de PC_2, qui se contente ensuite de donner le contrôle à IBJOB, pour qu'il interprète la carte C_M .

Le passage à l'assembleur se fait dons sans retour à PC_0 , et le travail fictif PASSIBS est inutile puisque ALGOL est alors un des composants reconnus par IBJOB.

Pratiquement, la première partie de la phase assembleur MAP est réécrite, à la suite de PC_2 , sur l'unité de support du système.

Dans cette partie, le fichier B_G est spécifié comme unité d'entrée.

Le compilateur est donc formé en fait de quatre enregistrements sur l'unité système. Le contrôle est donné, après exécution de ce quatrième enregistrement, à la deuxième partie de l'assembleur.

Remarquons qu'une différence apparait dans ce fonctionnement avec les compilateurs FORTRAN ou COBOL. Ceux-ci, en effet, ne produisent pas de langage généré MAP, mais un langage intermédiaire L_1 où une partie de l'assemblage est déjà faite. Le passage à l'assembleur se fait alors directement dans sa deuxième partie. L'expérience a montré que la solution adoptée pour notre compilateur ne diminuait pas la rapidité de traitement des programmes sources.

Enfin, on trouvera en appendices la forme de l'information à transmettre à IBSYS et IBJOB pour faire exécuter un "travail" en ALGOL.

B - OPTIONS DE TRAITEMENT

Une partie de l'information transmise par l'utilisateur au système est destinée à l'ensemble compilateur - assembleur et caractérise la possibilité pour l'utilisateur d'avoir un rôle actif dans le traitement de son travail par cet ensemble.

Cette information se présente sous forme d'options dans une ou plusieurs "cartes contrôles". Pour les options concernant le compilateur, les indicateurs I_S , nuls au départ, sont respectivement mis à un. Celles concernant l'assembleur, sont régénérées en tête de P_M dans l'intégration sous IBSYS, et reproduites dans la simulation de la carte C_M d'appel à l'assembleur dans l'intégration sous IBJOB.

B1 - OPTIONS COMPILATEUR

Actuellement au nombre de deux, ces options sont appelées à être multipliées dans les versions à venir du compilateur (Cf. conclusion générale).

Elles ont une action directe sur le fonctionnement de PC_2 et sur la forme du programme généré P_M .

B1 - 1 Option VERIF

On a vu que l'effet de cette option était de commander, dans le programme généré, et dans les sous-programmes bibliothèques par l'intermédiaire d'un indicateur spécialisé (Cf. OUVALG, chapitre II, A1), la vérification de la validité des opérations effectuées par P_M .

En particulier, on vérifie à l'exécution, si VERIF est présente, que toutes les expressions en indice ont des valeurs comprises entre les bornes déclarées du tableau correspondant. Cette vérification est particulièrement efficace lors de la mise au point d'un programme.

On vérifie également, dans le cas où VERIF est demandée, que les divisions du type $\div$ sont possibles. On peut envisager d'étendre ce cas à toutes les opérations arithmétiques du langage L_S .

Dans tous les cas où une vérification est négative à l'exécution, un libellé approprié est imprimé, accompagné du numéro de l'instruction générée où l'incident s'est produit. On verra (Cf. B2) comment ce numéro peut-il être exploité efficacement par l'utilisateur.

B1 - 2 Option RECUR

Cette option est à utiliser lorsque dans un programme P_S est déclarée au moins une procédure P définie ou / et appelée récursivement.

L'option a pour effet de charger en mémoire, au moment de l'exécution de P_M , les programmes interpréteurs d'empilage et de désempilage ALPRR et ALDST (Cf. deuxième partie, chapitre IV, H2 - 2). Si une procédure est appelée ou définie récursivement dans un programme P_S sans que RECUR ne soit présente, un libellé d'erreur approprié est imprimé, avec le numéro de l'instruction générée où l'incident s'est produit.

Rappelons que l'option RECUR est nécessaire si P_S contient une procédure susceptible de revenir au programme principal sans passer par sa fin normale.

B2 - OPTIONS ASSEMBLEUR

Ces options sont celles habituellement employées par les compilateurs du type FORTRAN intégrés au système :

L'option LIST permet d'obtenir sur l'unité de sortie la liste d'assemblage des instructions générées de P_M .

L'option NODECK permet de ne pas obtenir le paquet de cartes écrites en code binaire et correspondant au programme traité.

Enfin, l'option REF est la plus utile à l'utilisateur. Elle permet d'obtenir le dictionnaire de références donnant le numéro d'instruction affecté par l'assembleur à chaque symbole interne de P_M . Associé à la table d'équivalences de symboles externes et internes (Cf. première partie, chapitre II, E2), ce dictionnaire permet d'interpréter facilement les libellés d'erreur imprimés à l'exécution.

CHAPITRE II

PROGRAMMES BIBLIOTHEQUE

L'ensemble des sous-programmes disponibles sur l'unité physique qui supporte le système est nommé bibliothèque du système*. Une partie de la bibliothèque n'est pas accessible à l'utilisateur : elle est formée de sous-programmes d'interprétation directement appelés par le compilateur.

Une autre partie est constituée d'un ensemble de fonctions "standard" dont l'existence est recommandée dans la définition du langage source [1] , ou dont la présence antérieure dans le système a été exploitée pour offrir des possibilités plus grandes à l'utilisateur.

Enfin, des sous-programmes d'entrées - sorties sont définis, qui sont appelés dans P_S de la même façon que les fonctions standard fermées.

* Cette partie a été écrite et mise au point par M. BERTHAUD

A - PROGRAMMES INTERPRETEURS

Ces programmes assurent, à l'exécution de P_M, le fonctionnement de toutes les parties dynamiques du compilateur et sont susceptibles de faire imprimer un certain nombre de libellés d'erreurs, spécifiant une mauvaise utilisation du langage source dans une instruction de P_M dont on donne le numéro (Cf. chapitre I, B2 pour l'exploitation de ces libellés).

A1 - OUVERTURE DE P_M

Le sous-programme OUVALG, appelé en tête de P_M (Cf. deuxième partie, chapitre III), assure l'initialisation de plusieurs mémoires indicatrices et le chargement de la mémoire de manoeuvre M - 1 par l'adresse la plus basse de la mémoire dynamique DYNL .

Ce sous-programme qui contient en entrée les mémoires NDYNL et NDYNR appelées à contenir les niveaux courants de DYNL et DYNR . NDYNR est initialisé ici, par l'adresse de la mémoire qui précède immédiatement les zones intermédiaires d'entrée - sortie.

Deux indicateurs ALMOD et FORIM utilisés par les sous-programmes d'entrée - sortie sont également définis ici.

Enfin, la mémoire OUVALG elle-même est rendue négative si l'option VERIF est demandée (paramètre MZE au lieu de PZE). Cette information sera conservée durant l'exécution de P_M, et sera utilisée par le sous-programme interpréteur TABE .

A2 - SOUS-PROGRAMMES DE CALCUL DES VARIABLES INDICEES

Les programmes TABD et TABE sont chargés en mémoire si au moins un tableau est déclaré dans P_S (Cf. deuxième partie, chapitre IV, J).

A2 - 1 Déclaration : TABD

Un seul appel de TABD peut traiter la déclaration des tableaux d'une même section.

On donne en paramètres à ce programme

- L'adresse de la mémoire spécifique M - k du bloc d'où il est appelé
- Les caractéristiques des tableaux déclarés
- Les adresses des bornes des tableaux
- Les mémoires spécifiques (types U - j ou V - j) des tableaux déclarés.

Le rôle de TABD est de construire successivement pour chacun des tableaux un "bloc de bornes" qui servira à l'exploitation (Cf. TABE) et de ranger dans U - k les caractéristiques du tableau, ainsi que l'adresse de début de ses mémoires.

Si les tableaux sont de type local, les blocs de bornes construits sont placés en mémoire dynamique à partir de la valeur courante de NDYNL . Les mémoires des tableaux elles-mêmes sont réservées derrière les blocs de bornes respectifs.

Au retour, NDYNL , et la mémoire spécifique M - k sont mis à jour par l'adresse de la mémoire qui suit la dernière réservée. On vérifie que le nouveau niveau ne dépasse pas la capacité de la mémoire dynamique.

Dans le cas où le tableau est rémanent, les mémoires sont réservées dans DYNR et NDYNR est mis à jour.

Remarquons que dans ce cas, les blocs de bornes restent construits en tête des tableaux, ceux-ci étant exploités de la même manière quelque soit leur type (Cf. TABE).

Dans tous les cas, TABD vérifie que les bornes inférieures ne sont pas supérieures aux bornes supérieures [1, 5, 2, 4] et effectue éventuellement les conversions de type nécessaires. Rappelons qu'une borne de type réel est convertie dans l'entier le plus proche de sa valeur.

Des libellés d'erreur appropriés sont imprimés si une déclaration est illégale.

La constitution des blocs de bornes est la suivante :

Soit la déclaration générale

<u>TABLEAU</u> T $\left[a_1 : b_1, a_2 : b_2, \ldots, a_n : b_n\right]$;

Le bloc de bornes construit est le suivant :

$B : a_1$

$B + 1 : (b_1 - a_1 + 1)\ C\ [B + 3]$

$B + 2 : a_2$

$B + 3 : (b_2 - a_2 + 1)\ C\ [B + 5]$

$\vdots$

$B + 2n - 3 : (b_{n-1} - a_{n-1} + 1)\ C\ [B + 2n - 1]$

$B + 2n - 2 : a_n$

$B + 2n - 1 : (b_n - a_n + 1)$

En particulier, la deuxième mémoire du bloc donne l'encombrement du tableau.

L'augmentation de M - k (ou la diminution de NDYNR) après cette déclaration est donc égale à

$2n + C\ [B + 1]$

L'adresse $B + 2n$ est alors placée en $U - j$, qui reçoit également (Cf. deuxième partie, chapitre IV, J1) le nombre de bornes du tableau et son type.

A2 - 2 Exploitation : TABE

Le programme TABE a pour fonction (Cf. deuxième partie, chapitre IV, J2) de fournir au programme généré P_M l'adresse en mémoire dynamique d'une variable indicée.

Les paramètres donnés à TABE sont

- L'adresse de U - j , mémoire spécifique du tableau, qui contient l'adresse du tableau, donc celle de son bloc de bornes, et le nombre de mémoires occupées par celui-ci.
- L'adresse des mémoires (et leur type) où se trouvent les valeurs des expressions en indice.

TABE construit l'adresse demandée en registre AC , par itérations successives à l'aide du bloc de bornes B . En même temps on vérifie la validité des types des expressions en indice, et on effectue les conversions réel → entier nécessaires.

En plus, si l'option VERIF est demandée, on vérifie que la valeur de chaque expression en indice est comprise entre celles des bornes inférieure et supérieure correspondantes. Un libellé approprié avec référence au numéro d'instruction dans P_M est imprimé si une erreur est détectée. Cette erreur peut également porter sur le nombre de bornes du tableau.

L'adresse de chaque variable indicée est calculée de telle façon que chaque tableau peut être considéré comme rangé "en lignes" à partir de la mémoire B + 2n . C'est à dire que, en reprenant l'exemple général de A2 - 1 , le tableau est rangé de la façon suivante :

$$
\begin{array}{lll}
A_0 = A\left[U - j\right] = B + 2n & \text{-------} & T\left[a_1, a_2 \ldots, a_{n-1}, a_n\right] \\
B + 2n + 1 & \text{-----} & T\left[a_1, a_2 \ldots, a_{n-1}, a_n + 1\right] \\
\vdots & & \vdots \\
 & & T\left[a_1, a_2 \ldots, a_{n-1}, b_n\right] \\
 & & T\left[a_1, a_2 \ldots, a_{n-1} + 1, a_n\right] \\
 & & T\left[a_1, a_2 \ldots, a_{n-1} + 1, a_n + 1\right] \\
\vdots & & \vdots \\
 & & T\left[a_1, a_2 \ldots, a_{n-1} + 1, b_n\right] \\
\vdots & & \vdots \\
 & & T\left[a_1, a_2 \ldots, b_{n-1}, b_n\right] \\
\vdots & & \vdots \\
B + 2n - 1 + C\left[B + 1\right] & \text{---} & T\left[b_1, b_2 \ldots, b_n\right]
\end{array}
$$

Le calcul de l'adresse A par TABE est alors, pour la variable indicée $T\left[i_1, i_2 \ldots i_n\right]$, et si A_0 est l'adresse contenue en $U - j$:

$$
\begin{aligned}
A = A_0 & + (i_1 - C\left[A_0 - 2n\right]) \; C\left[A_0 - 2n + 3\right] \\
& + (i_2 - C\left[A_0 - 2n - 2\right]) \; C\left[A_0 - 2n + 5\right] \\
& \vdots \\
& + (i_k - C\left[A_0 - 2(n - k + 1)\right]) \; C\left[A_0 - 2(n - k) + 1\right] \\
& \vdots \\
& + (i_{n-1} - C\left[A_0 - 4\right]) \; C\left[A_0 - 1\right] \\
& + (i_n - C\left[A_0 - 2\right]) .
\end{aligned}
$$

A3 - SOUS-PROGRAMMES D'APPELS DE PROCEDURES

L'appel d'une procédure déclarée P s'accompagne dans P_M de celui d'un sous-programme d'entrée ALPRO (Cf. deuxième partie, chapitre IV, H2 - 1). Ce sous-programme est remplacé par ALPRR si l'option RECUR est présente. Dans ce cas, le programme de sortie ALDST est également appelé.

Enfin, le sous-programme ALVTA assure la gestion des tableaux appelés par valeur.

A3 - 1 Entrée normale : ALPRO

On a vu que ce sous-programme avait six fonctions :

1°) Le garnissage du vecteur transfert de la procédure est réalisé en prenant les adresses des paramètres de l'appel de P_N et en les plaçant dans les parties adresses des mémoires associées du type $P_N - k$.

2°) Pour ceux des paramètres qui ne comportent pas de sous-programmes, la mémoire associée dans la liste qui suit l'appel est transférée dans la mémoire associée du type Fn + k .

Pour cela, l'adresse Fn est donnée en paramètre à ALPRO.

3°) En même temps que le garnissage du vecteur transfert ou des mémoires Fn + k, ALPRO vérifie la conformité de type, genre et nombre entre les paramètres effectifs et les paramètres formels dont on lui donne les caractéristiques en paramètres.

Toute erreur décelée ici entraine l'arrêt de l'exécution et l'impression d'un libellé approprié avec référence du numéro d'instruction de l'appel et du symbole interne P_N .

4°) ALPRO assure la mise à jour par la valeur contenue dans NDYNL de la mémoire spécifique. La valeur de NDYNL est elle-même celle de la mémoire spécifique du bloc où se trouve l'appel de P_N.

5°) L'adresse de retour de la procédure est calculée et placée en partie adresse de l'instruction TRA générée en Fn - 2. Si Ri est l'adresse de l'instruction d'appel à P_N et p le nombre de paramètres, cette valeur est Ri + p + 2.

6°) Enfin, le compteur de récursivité Fn - 1 est testé. S'il est non nul, c'est que la procédure est appelée récursivement et un libellé d'erreur est imprimé. Sinon, il est mis à un. Il sera annulé à la sortie de la procédure.

A3 - 2 Entrée récursive : ALPRR

Ce sous-programme a les mêmes fonctions que ALPRO. Le compteur de récursivité, cependant, est toujours augmenté de un ici, sans entrainer de libellé d'erreur.

La fonction essentielle de ALPRR est en plus, de recopier dans la mémoire dynamique, à partir de la valeur de NDYNL, les valeurs des paramètres et des variables et mémoires de manoeuvre locales à la procédure. Ces valeurs sont désempilées et réaffectées aux variables par le programme de sortie en même temps que le niveau de récursivité est diminué de un.

Le recopiage n'intervient que si le compteur de récursivité est au moins égal à un à l'entrée dans la procédure. Il s'effectue de la façon suivante, avec vérification à chaque pas du non dépassement de la capacité mémoire.

On met d'abord en réserve l'adresse de retour de la procédure (instruction TRA en Fn - 2), puis Fn lui-même et les p mémoires associées qui le suivent.

Puis, à l'aide des mémoires de la forme

PZE x ,,A

générées à cet effet en fin de procédure (Cf. deuxième partie, chapitre IV, H3 - 5), on empile successivement

- Les x_1 mémoires de manoeuvre (M - m) de la procédure
- Les x_2 mémoires spécifiques (V - j) des tableaux rémanents locaux à la procédure
- Les x_3 mémoires spécifiques (U - j) des tableaux locaux de la procédure
- Les x_4 valeurs des quantités rémanentes (R - i) locales à la procédure
- Les x_5 valeurs des quantités locales (Q - i) de la procédure.

Rappelons que si un des x est nul, la mémoire PZE correspondante n'est pas générée. L'empilage s'arrête quand une mémoire nulle est trouvée.

La valeur de NDYNL est alors augmentée de $p + 2 + \sum x_i$. C'est cette nouvelle valeur qui est placée dans la mémoire spécifique de la procédure, qui se trouve être la première des mémoires locales dont l'ancienne valeur a été empilée. Cette mémoire conservera sa valeur jusqu'à la sortie de la procédure si aucun appel récursif de plus haut niveau n'a lieu. En effet, NDYNL ne peut varier alors que lors d'une déclaration de tableau. Celle-ci intervenant obligatoirement au début d'un bloc, c'est la mémoire spécifique du bloc qui serait mise à jour, et non celle de la procédure.

A3 - 3 Sortie de procédure : ALDST

Le programme de désempilage est symétrique de ALPRR : si le compteur de récursivité est égal à un, tout se passe comme pour une sortie normale : le compteur Fn - 1 est remis à zéro, la valeur en AC est conservée (c'est celle de Fn dans la cas d'un indicateur de fonction), et le retour est assuré à l'adresse contenue dans l'instruction TRA en Fn - 2 .

S'il doit y avoir désempilage (compteur supérieur à un) le compteur est diminué de un et la valeur du registre AC est sauvegardée ainsi que l'adresse de retour.

Puis, les valeurs empilées par ALPRR sont réaffectées aux mémoires locales, aux paramètres, à Fn et à l'adresse de retour.

On note que la première mémoire à désempiler est trouvée en diminuant de un l'adresse contenue dans la mémoire spécifique de procédure.

Au cours du désempilage, celle-ci est automatiquement restaurée à son ancienne valeur, de sorte que si un autre appel récursif survient, l'empilage se fera à l'emplacement du précédent.

Puis, le vecteur transfert de la procédure est reconstruit à l'aide de l'adresse de retour qui vient d'être désempilée. Cette adresse est en effet celle de l'instruction qui suit la liste des paramètres à l'appel de P_n .

Enfin, le retour est assuré à l'adresse sauvegardée en tête, avec un registre AC la valeur également sauvegardée.

A3 - Tableaux appelés par valeur : ALVTA

Ce programme est appelé en tête de procédure avec pour paramètres l'adresse de la mémoire spécifique M - m et de la mémoire Fn + k associée au tableau. Rappelons que celle-ci a la même valeur que la mémoire U - j ou V - j spécifique du tableau effectif corres-

pondant.

La fonction de ALVTA est alors de recopier dans DYNL le bloc de bornes du tableau et le tableau lui-même, dont l'adresse de la première mémoire est en partie adresse de Fn + k . La longueur du bloc de bornes est elle-même donnée par la partie décrément de Fn + k . La partie adresse de Fn + k est mise à jour pour tenir compte du recopiage.

La mémoire spécifique M - m est automatiquement mise à jour par la nouvelle valeur de NDYNL . Notons que la modification de M - m interdit tout appel récursif d'une procédure comportant des tableaux appelés par valeur.

B - FONCTIONS STANDARD FERMEES

B1 - DEFINITION - GENERALITES

Nous appellerons fonction standard fermée tout sous-programme susceptible d'être incorporé à la bibliothèque du système et par conséquent employé dans tout programme P_S sans déclaration préalable .

Ces sous-programmes sont écrits en langage MAP et doivent respecter les conventions suivantes :

- Appel et retour par instructions TSX XXXX,4
  et TRA 1,4
- Point d'entrée XXXX : un symbole de quatre caractères qui doit apparaître en bonne place dans la table de fonctions standard de PC_2 .
- Appel dans PC_S : par un symbole externe quelconque du type identificateur de L_S , dont les caractères en code D et le profil qualitatif doivent être placés dans une ligne de TFS (Cf. première partie, chapitre II, F3).
- Paramètres : uniquement des expressions arithmétiques ou booléennes, des étiquettes simples non formelles ou des chaînes.

  Ces paramètres sont évalués une fois et une fois seulement avant d'entrer dans la procédure.

  Si Yr est l'adresse de l'instruction d'appel, Yr + k contient l'adresse du k ème paramètre en partie adresse et son type en partie décrément : zéro pour une étiquette, un pour une expression booléenne, deux pour une expression entière, trois pour une chaîne, quatre pour une expression réelle.
- Valeur : si une fonction est appelée comme indicateur de fonction, son type doit apparaître dans le profil qualitatif en TFS, et sa valeur au retour doit être en registre AC .

B2 - FONCTIONS ACTUELLEMENT DISPONIBLES

Les fonctions standard disponibles actuellement se divisent en fonctions arithmétiques et fonctions d'entrée - sortie .

Celles-ci feront l'objet de la partie suivante (Cf. C).

Les fonctions arithmétiques, elles, sont toutes des indicateurs de fonction de type réel, acceptant un et un seul paramètre de type réel ou entier. On en trouvera la liste en appendice II - 5.

C - FONCTIONS D'ENTREE - SORTIE

Cet ensemble de programmes est constitué actuellement de dix fonctions standard fermées et de deux fonctions ouvertes que nous étudierons en D .

Les procédures fermées forment elles-mêmes quatre groupes distincts [32] .

C1 - FONCTIONS D'ENTREE IMMEDIATE

Trois indicateurs de fonction sans paramètre RDONNEE , EDONNEE et BDONNEE , de type respectif réel, entier et booléen, permettent de lire un par un des nombres purs ALGOL ou des valeurs logiques écrits à la suite de P_S sur l'unité physique d'entrée. Ces données doivent être séparées par des ";" .

C2 - FONCTIONS D'ENTREE - SORTIE ELEMENTAIRES

Deux procédures LIRE et ECRIRE sans type, et avec un nombre de paramètres indéterminé, permettent de lire sur l'unité physique d'entrée, ou d'imprimer sur l'unité de sortie les valeurs des variables et expressions arithmétiques ou booléennes placées en paramètres.

A l'entrée, le format des données est le même que pour les fonctions d'entrée immédiate.

On vérifie la conformité de type entre les paramètres et les valeurs lues.

A la sortie, le format d'impression est fixé par le type de l'expression à écrire.

Des chaînes peuvent être imprimées si elles ne dépassent pas quinze caractères.

C3 - FONCTIONS D'ENTREE - SORTIE ETRANGERES

Deux procédures ENTREE et SORTIE , sans type, et avec un nombre de paramètres indéterminé permettent de lire et écrire des données de type quelconque, suivant un format et sur une unité physique choisis par l'utilisateur.

C3 - 1 Numéro d'unité

Le premier paramètre de l'appel constitue le numéro de l'unité choisie par l'utilisateur. Afin que celui-ci n'ait aucun rapport avec les identités physiques des supports d'information, ce numéro est symbolique. Une table interne au superviseur d'entrée - sortie lui fait correspondre un numéro d'unité physique.

C3 - 2 Format des données

Le format choisi par l'utilisateur est indiqué en deuxième paramètre de l'appel sous forme d'une étiquette simple E_i .

Celle-ci se refère à une partie du programme P_S qui doit comporter un appel de la procédure MODELE (Cf. D). Le premier appel à MODELE exécuté à partir de E_i détermine le format voulu.

C3 - 3 Liste des données

Les autres paramètres de l'appel à ENTREE ou SORTIE sont pris par couples pour déterminer le nombre et l'adresse des variables ou expressions à lire ou écrire.

C4 - FONCTIONS UTILITAIRES

Les trois procédures REBOBINER, ECRIRE FDB , ESP ARRIERE permettent respectivement de ramener une unité physique choisie par l'utilisateur à son point de chargement, d'y inscrire une marque de fin de fichier ou de lui faire effectuer autant d'espacements arrière qu'on le désire, sur des enregistrements de données effectués auparavant.

D - AUTRES FONCTIONS

D1 - FONCTIONS STANDARD OUVERTES

Nous avons vu que cinq fonctions standard ouvertes arithmétiques et deux d'entrée - sortie (Cf. deuxième partie, chapitre IV, I5 - 4 - 3 et chapitre V, A3) sont à la disposition de l'utilisateur. Pour celui-ci, l'appel de ces procédures est identique à celui des procédures standard fermées.

Dans le compilateur, une ligne de TFS (PC_1) est toujours consacrée à chacune de ces fonctions, mais aucun point d'entrée n'apparaît dans la table correspondante de PC_2 .

On trouvera en appendice la liste et les spécifications de ces fonctions.

D2 - FONCTION PUISSANCE

L'occurrence dans un programme P_S de l'opérateur ↑ entraine le chargement, à l'exécution de P_M , du sous-programme PUAL de calcul de puissances. Ce sous-programme est identique à une fonction standard fermée, appelée par TSX PUAL,4 .

En particulier, ses deux paramètres peuvent être de type réel ou entier et le résultat est toujours de type réel , en registre AC .

CHAPITRE III

CONCLUSION - DISCUSSION

L'intégration dans le système IBSYS/IBJOB du compilateur PC semble apporter à l'utilisateur un grand nombre d'avantages, parmi lesquels nous retiendrons

- Une facilité et une souplesse d'emploi dûes en particulier à l'alignement sur les compilateurs existant en ce qui concerne la forme et le type des informations échangées avec le système.
- L'accès à une bibliothèque de programmes théoriquement illimitée et la possibilité d'enrichir cette bibliothèque par l'intermédiaire des procédures écrites en code machine.
- La définition d'une gamme étendue de fonctions d'entrée - sortie n'exigeant de l'utilisateur aucune connaissance de la gestion des unités physiques connectées à la machine.
- Enfin, la disponibilité d'une série très complète de libellés d'erreurs qui, associés à la table d'équivalences donnée par PC_1 et au dictionnaire de références (option REF) fournis par l'assembleur, forment un puissant auxiliaire de mise au point des programmes P_S.

Les possibilités nouvelles des phases de chargement des systèmes du type IBJOB permettent d'envisager une efficacité encore accrue dûe à cette intégration.

Il est souhaitable en particulier que des sous-programmes écrits extérieurement à un programme principal P_S soient directement utilisables dans celui-ci, sans que l'utilisateur ait à les inclure dans son propre programme ou dans la bibliothèque.

Les sous-programmes externes pourraient être écrits eux-mêmes en ALGOL ou dans un autre langage, FORTRAN , MAP , etc ...

Ceci peut être réalisé grâce à des fonctions standard "de communication" , dont l'appel serait compilé comme celui de fonctions ouvertes, mais qui se traduiraient par l'appel au sous-programme spécifié comme si celui-ci faisait partie du programme P_S (s'il est écrit en ALGOL) ou de la bibliothèque (s'il est écrit en FORTRAN ou MAP).

Les paramètres de ces fonctions seraient ceux du sous-programme, auxquels on ajouterait sous forme de chaîne, le point d'entrée du sous-programme. Celui-ci peut être défini lui-même, en MAP ou FORTRAN par un ordre EXTERN ou SOUS-PROGRAMME. En ALGOL, une fonction REFERENCE pourrait associer par exemple,un point d'entrée à un identificateur de procédure.

Nous pensons qu'ainsi complètée, l'intégration de PC au système permettra à l'utilisateur de profiter au maximum des possibilités conjuguées du langage source ALGOL et du système général de programmation IBM.

- C O N C L U S I O N G E N E R A L E -

A - RESULTATS OBTENUS

En conclusion de cet ouvrage nous présentons d'une part les résultats obtenus grâce au programme PC , d'autre part les projets d'améliorations et d'extensions qu'il nous parait intéressant de préciser et de mettre au point sur le compilateur dans les mois à venir.

Les résultats auxquels a conduit l'écriture du compilateur sont étayés dès à présent par plusieurs mois d'exploitation, en particulier au Laboratoire de Calcul de l'Institut de Mathématiques Appliquées de Grenoble, sur machine 7044.

Grâce à l'intégration du programme dans le système IBJOB utilisé sur cette machine, l'exploitation a pu en être faite de façon régulière et systématique. Les travaux exécutés sur la machine sous forme de "trains moniteurs" comportent en effet un pourcentage très élevé de programmes écrits en ALGOL. La mise à la disposition de l'utilisateur des possibilités évoquées en troisième partie accroîtra certainement encore ce pourcentage.

A la lumière de cette expérience pratiquement exhaustive, nous nous permettons d'insister sur les conclusions qui suivent.

A1 - PRESENTATION DES PROGRAMMES SOURCES

Les conventions adoptées pour la perforation des programmes P_S (Cf. Appendice) paraissent donner pleine satisfaction aux utilisateurs. En particulier, la possibilité d'employer dans un

même programme les symboles de base du langage exprimés en français ou en anglais semble très intéressante. Les conventions relatives aux symboles courts du langage qui n'existent pas sur les perforatrices du type IBM offrent plus de contraintes à l'utilisateur.

A2 - CONTENU DES PROGRAMMES SOURCES

Les limitations des programme P_S relatives à leur encombrement ou au nombre de certains types d'éléments (nombres purs, procédures ...) ont été fixées au mieux après un assez grand nombre d'expériences. Il est évident toutefois que ces limitations ne doivent pas devenir des contraintes pour l'utilisateur désireux de faire exécuter un travail plus important. Nous verrons quelles solutions peuvent être apportées à ces problèmes.

Au contraire, les conventions relatives à la forme syntaxique ou sémantique des programmes sources ont été choisies pour n'être à l'origine d'aucune contrainte. Des facilités particulières sont prévues dans le compilateur lorsque ces conventions vont à l'encontre des recommandations sur le langage L_S [1].

EXEMPLE

L'existence des fonctions standard CONVER et CONVRE résout complètement les difficultés pouvant survenir relativement aux types d'expressions (type toujours réel du résultat de l'opération puissance, concordance des types à l'appel d'une procédure, etc ...).

Parmi les possibilités du compilateur particulièrement employées il faut noter d'une part l'emploi de procédures en code MAP, d'autre part le profit tiré de la gestion dynamique de la mémoire en cours d'exécution. Pour tous les travaux courants, cette réalisation supprime pratiquement les problèmes d'encombrement en mémoires de travail.

La compilation de programmes comportant des déclarations de procédures appelées ou / et définies récursivement s'est trouvée appliquée dans trop peu d'exemples jusqu'à présent pour que des conclusions définitives sur son efficacité soient tirées. Il apparaît cependant que les projets d'extensions établis à ce sujet, que nous évoquerons plus loin, sont loin d'avoir un caractère d'urgence. Il en est de même en ce qui concerne la systématisation de l'effet de bord dans les appels de procédures, dont il n'est actuellement pas tenu compte dans la majorité des cas.

A3 - MISE AU POINT DES PROGRAMMES

Nous avons évoqué à plusieurs reprises les facilités de mise au point des programmes sources données par le compilateur, aussi bien pendant les passages édition et compilation que lors de l'exécution du programme généré lui-même. Ces facilités sont relatives à la première phase de mise au point des programmes, qui consiste à définir un texte P_S correct vis à vis des définitions du langage source.

En attendant la disponibilité d'un programme d'analyse complet intégré au système général * susceptible d'assurer la phase finale de mise au point des travaux, relative aux résultats effectifs obtenus par l'utilisateur, plusieurs possibilités sont offertes à celui-ci.

Les plus intéressantes paraissent être celles basées sur l'emploi de programmes simplifiés d'impression de résultats intermédiaires.

Ces programmes peuvent être les procédures de sortie élémentaires définies à cet effet dans le langage L_S , ou des programmes déjà existant en bibliothèque (DUMP, PDUMP, et S.SNAP sous le système IBJOB) et qu'il est toujours possible d'utiliser grâce à une procédure en code. Dans les cas extrêmes, une procédure d'arrêt de la ma-

* Un tel programme est effectivement en cours de mise au point (Mme J. GUELTON)

chine peut être définie de la même façon pour une mise au point manuelle.

A4 - PERFORMANCES DU COMPILATEUR

Les performances du programme P_C sont évidemment en liaison directe avec celles des machines utilisées. Elles sont de deux genres : performances en encombrement et performance en vitesse. Celle-ci se divise à son tour en performance à la compilation et à l'exécution.

Ces performances ne sont pas indépendantes. En particulier l'amélioration de l'une d'entre elles entraine le plus souvent l'effet inverse sur l'une des deux autres ou sur les deux.

Actuellement, on peut dire que, compte tenu du fait que la plus grande généralité a été conservée au langage source, la performance vitesse d'exécution, en rapport inverse avec cette généralité, est le plus souvent sacrifiée aux deux autres types.

La vitesse de compilation obtenue est en particulier tout à fait comparable à celle obtenue sur d'autres machines, ou avec d'autres langages. Quant à l'encombrement en mots mémoires, on a vu que les algorithmes mis en oeuvre au cours de la compilation étaient destinés à le rendre aussi proche que possible de celui d'un programme idéal P équivalent, écrit en langage machine.

Le tableau ci-dessous donne quelques résultats relatifs à des programmes de structures syntaxiques différentes, pris au hasard dans un train moniteur sur la machine 7044.

N est le nombre d'unités syntaxiques du programme. t_1 (sec) est le temps de compilation et t_2 la somme des temps d'assemblage et de chargement du programme généré. Les temps marqués d'un astérisque sont relatifs à des travaux qui comportaient l'option LIST.

N1 est le nombre de mémoires occupées par le programme généré, N2 celui pris par les sous-programmes bibliothèques et N3 l'encombrement maximum que peut occuper la mémoire dynamique. Ces nombres n'ont pu être fournis que lorsque l'option MAP (géographie de la mémoire à l'exécution de P_N) était spécifiée dans l'information donnée au sous-système IBJOB.

N	t_1	t_2	N1	N2	N3
0	5	24	11	294	26941
119	6	34 *	143	2379	23758
271	6	30			
286	6	30			
329	7	32			
416	8	30			
479	7	39	371	4303	19337
522	10	48 **	519	3041	22873
588	8	36			
810	10	40			
1391	14	66	1437	3101	21742
1915	18	64			
2512	22	82	2766	4400	16816
3474	28	126	3712	4459	15840
4258	30	128			
4654	36	138	3654	4312	16053
6169	42	145			
6662	40	180			

On voit sur ce tableau que le temps de compilation d'un programme ne dépend pas seulement de sa taille (N) mais aussi de sa struture syntaxique, qui n'est pas chiffrée ici.

Toutefois, on note un minimum de temps de 5 secondes (aucune unité syntaxique), qui, pratiquement, représente le temps pendant lequel l'unité système se déplace pour charger en mémoire les quatre phases successives du compilateur. Notons que l'essai a été effectué alors que l'unité système était portée par un dérouleur de bande magnétique du type 729 - II.

La colonne relative à l'encombrement en mots mémoire du programme généré P_M paraît plus régulière que la colonne t_1.

Très grossièrement, on peut dire qu'un programme de N unités syntaxiques se compile en un temps $t_1 \sim 5 + N/120$ secondes et qu'il donne naissance à un nombre $N1 \sim N$ d'instructions ou de mémoires réservées.

Enfin, la valeur des deux dernières colonnes dépend essentiellement du nombre de sous-programmes appelés dans P_S (et aussi de N pour N3).

Pour le programme fictif de zéro unité syntaxique, on trouve ainsi le maximum de l'encombrement de la mémoire dynamique disponible.

Rappelons qu'avec le système employé, le noyau et le superviseur d'entrées - sorties peuvent occuper jusqu'à 5.500 mots mémoires et les zones intermédiaires d'entrées - sorties jusqu'à environ 3.200 mots.

La performance de vitesse à l'exécution des programmes générés dépend, elle, essentiellement de la structure du programme source. Elle est fortement diminuée en particulier si celui-ci comporte un grand nombre de variables indicées à l'intérieur de boucles POUR.

Un réel souci d'optimisation de la part de l'utilisateur peut en fait considérablement augmenter cette performance. On verra comment, par l'intermediaire d'une option, on peut prévoir d'ôter cette contrainte à l'utilisateur.

Actuellement, il est admis qu'un programme d'analyse numérique comportant un grand nombre de variables indicées a, en moyenne, une performance de vitesse à l'exécution 3,5 fois inférieure à celle du même programme écrit dans le langage FORTRAN.

Pour un programme comportant des variables indicées en nombre réduit, la performance est comparable à celle du langage FORTRAN, ce qui est du certainement , malgré la généralité du langage source, aux algorithmes d'optimisation évoqués dans le texte et à la grande simplicité des appels de procédures par exemple.

A5 - EXTENSIONS DU LANGAGE SOURCE

Une première extension intéressante du langage source L_S acceptée actuellement par le compilateur est la possibilité d'affecter à une variable contrôlée le type booléen, que nous avons déjà évoquée.

Surtout, grâce aux fonctions d'entrées - sorties étendues et aux formats spéciaux de lecture et d'impression, il est possible de banaliser complètement le contenu des mémoires associées à des variables d'un type quelconque.

Deux extensions sont utilisées par ce fait :

1°) La possibilité d'avoir dans un mot mémoire jusqu'à cinq caractères alphabétiques permet, d'une part, grâce aux opérations de relation, d'exécuter des programmes de tri alphanumérique, d'autre part, à l'aide du format A [31] , de faire imprimer sur une unité de sortie la courbe représentative d'une fonction quelconque.

2°) La possibilité de considérer un mot mémoire comme la juxtaposition de une à trente cinq variables booléennes permet de traiter très facilement et avec l'encombrement minimum, à l'aide des opérations $\vee$, $\wedge$, et des instructions conditionnelles, des problèmes relatifs à de telles variables.

B - PROJETS SUR LE COMPILATEUR

Nous voudrions montrer ici combien le travail qui a été réalisé en étudiant et en mettant au point le compilateur, loin de réunir les caractéristiques d'un ensemble définitif, est certainement appelé à être profondément amélioré, modifié ou étendu.

Nous nous sommes attachés, au cours des chapitres précédents, à mettre en relief les parties de ce travail susceptibles d'une refonte plus ou moins totale ou d'une amélioration rendue possible grâce à de nouvelles dispositions adoptées sur les machines considérées.

Ces modifications étaient internes au compilateur et sans rapport avec l'utilisateur.

Nous essaierons maintenant de définir, à la lumière de l'expérience d'exploitation qui se déroule actuellement, quelques projets qui nous paraissent susceptibles d'améliorer les conditions de travail de l'utilisateur, ou de lui ouvrir de nouvelles possibilités dans le cadre du langage source L_S .

B1 - AMELIORATIONS INTRINSEQUES DU COMPILATEUR

Nous voulons indiquer ici comment les restrictions et les limitations les plus importantes citées ci-dessus (Cf. A2) pourront être supprimées.

B1 - 1 Limitations d'encombrement

Elles seront "tournées" par la définition d'un programme bilbiothèque d'enchaînement des divers maillons d'un même travail.

B1 - 2 Restrictions sur l'emploi des tableaux

Grâce à la définition d'une véritable chaîne de reprise générée par les programmes interpréteurs ALPRO et ALVTA, on pourra employer des tableaux appelés par valeur dans des procédures récursives. La définition d'un nouveau programme interpréteur de recopiage des tableaux persistants permettra également l'appel de ceux-ci par valeur, et l'existence d'une partie réellement dynamique de la mémoire réservée à ces tableaux.

B1 - 3 Conventions relatives à la récursivité

1°) Actuellement, les programmes interpréteurs d'empilage et de désempilage des variables locales à une procédure en cas de définition récursive de celle-ci sont tels que si, au niveau de récursivité n, un paramètre formel X appelé par nom est remplacé par une variable locale A, c'est la variable de niveau n également qui est considérée. Ceci n'est pas vrai si le paramètre est appelé par valeur (dans ce cas, la variable A est celle de niveau n - 1).

EXEMPLE : soit le programme :

```
DEBUT PROCEDURE F(X) ; REEL X ;
    DEBUT REEL A ;
        A := 1 ;
        SI X ≠ 1 ALORS F(A) ;
        X := 2 ;
        ECRIRE (A, X) FIN ;
REEL AO ;
    AO := 3 ;
    ECRIRE (AO) ; F(AO) ; ECRIRE (AO) FIN ;
```

On obtient avec notre compilateur le résultat suivant :

```
3
2     2
1     2
2
```

Si on respecte exactement les profondeurs de niveau lors d'un appel par nom, on obtient

```
3
1     2
2     2
2
```

On peut obtenir ce résultat en augmentant la complexité du programme interpréteur d'empilage. En particulier on peut lui donner en paramètres les adresses de toutes les instructions où sont employées des variables locales à la procédure à l'intérieur d'un paramètre effectif. Une constante d'empilage peut alors être ajoutée à ces adresses, avant de rendre le contrôle à la procédure. Bien entendu, le programme de désempilage doit effectuer le travail inverse.

2°) Quand un paramètre formel appelé par nom intervient comme paramètre effectif de sa propre procédure (dans l'exemple précédent, cela revient à appeler F(X) dans le corps de procédure), il faut éviter actuellement de faire exécuter indéfiniment le sous-programme de calcul de l'adresse du paramètre.

Pour cela, on a vu que l'appel du paramètre (P16) s'accompagne du test du compteur de récursivité correspondant. Si le compteur est au moins égal à un, l'appel est ignoré.

Ceci entraine deux contraintes pour l'utilisateur :

a) Il lui est impossible de prévoir un effet de bord sur l'adresse d'un paramètre formel appelé par nom en cas de récursivité.

b) Tous les paramètres formels appelés par nom doivent être "évoqués" au moins une fois à l'exécution du niveau 0 de récursivité de la procédure, afin que leur mémoire associée Fn + k soit chargée.

Remarquons que ce problème est analogue à celui du 1°), où la variable A est ici remplacée par la mémoire du vecteur transfert Pn - k associée au paramètre. Il faut pouvoir accéder, au niveau n, à une mémoire relative au niveau n - 1 , et donc déjà empilée. Là encore, ceci peut être réalisé grâce à un sous-programme interpréteur d'appel des paramètres par nom, auquel on fournit ici les adresses Pn - k et Fn.

3°) Enfin, nous avons vu que toute procédure se terminant par un ALLER A vers l'extérieur devait être considérée comme récursive à l'appel suivant.

Il est souhaitable de supprimer cette contrainte en effectuant dynamiquement un test sur la valeur de toute instruction ALLER A générée à l'intérieur d'un corps de procédure.

Si l'option RECUR n'est pas demandée, on génère la remise à zéro du compteur de récursivité quand le test montre que l'adresse de l'ALLER A est extérieure à la procédure. Cependant, on doit prévoir le retour en séquence d'un indicateur d'aiguillage en générant à la suite une réincrémentation éventuelle du compteur.

Si l'option RECUR est demandée, il faut, après le test, générer l'appel au sous-programme de désempilage si le ALLER A sort de la procédure. Cependant, on doit générer un autre appel au programme d'empilage pour tenir compte d'un retour éventuel en séquence d'un indicateur d'aiguillage.

B2 - DISPONIBILITE DE NOUVELLES OPTIONS

Deux options nouvelles peuvent être livrées à l'utilisateur dans le but d'une part de faire optimiser les programmes générés, d'autre part de systématiser complètement l'emploi de l'effet de bord des procédures.

B2 - 1 Option d'optimisation (OPT)

L'amélioration des programmes générés peut porter essentiellement sur la performance de vitesse à l'exécution. D'une façon générale, elle s'accompagne d'une augmentation de l'encombrement et d'une diminution de la généralité du langage source.

Les optimisations principales portent sur trois fonctions du compilateur et peuvent être commandées par l'utilisateur à l'aide de l'option OPT .

1°) Variables indicées

L'amélioration la plus importante des programmes générés porte sur l'exploitation des variables indicées. Actuellement, l'appel d'une variable indicée à deux dimensions demande le déroulement d'une centaine de cycles machines.

Il est possible de ramener ce nombre à une moyenne de six, grâce à l'emploi systématique des registres index de la machine. Le rapport de performance avec le langage FORTRAN serait alors ramené à une valeur moyenne de 1,3.

L'optimisation porterait sur tous les tableaux à une ou deux dimensions, dont les bornes inférieures sont 0 ou 1 . Elle mettrait en oeuvre, à la compilation, trois indicateurs de registres index, analogues à ceux des registres AC et MQ, et à l'exécution, une table d'adresses construite par le programme interpréteur TABD pour chaque tableau bénéficiant de l'optimisation. Afin d'assurer l'exploitation optimisée des paramètres formels remplacés par des tableaux à bornes inférieures égales à 0 ou 1 , la mémoire spécifique de chacun de ces tableaux contiendrait une indication spéciale.

Enfin, l'existence des indicateurs de registres index permettrait d'envisager une optimisation analogue à celle des registres AC/MQ relatives aux instructions de chargement de ces index.

Ainsi, une instruction du type

T [I, J] := T [I, J] + 1 ;

actuellement exécutée en deux cents cycles environ, pourrait demander douze cycles seulement.

2°) Boucles POUR

On peut envisager également d'utiliser les registres index de la machine pour assurer l'avance des variables contrôlées d'instruction POUR dont la liste aurait une forme particulièrement simple.

Quoique intéressante, cette optimisation s'avère d'une importance très inférieure à celle concernant les variables indicées.

3°) Appels de paramètres

Si l'option OPT est interprétée comme un engagement de la part de l'utilisateur à ne pas modifier par effet de bord l'adresse ou la valeur des paramètres effectifs d'appels de procédures déclarées, les sous-programmes relatifs à ces paramètres peuvent être appelés une fois et une fois seulement en tête de chaque procédure. Ceci réduirait très sensiblement le temps passé en communications entre corps de procédure et paramètres effectifs.

B2 - 2 Option effet de bord (S.E.)

Cette option, actuellement reconnue par le compilateur mais non "distribuée", permettra d'assurer, à la demande de l'utilisateur, la systématisation de l'effet de bord des appels de procédures.

Par la génération, quand l'option S.E. est présente, de la mise en réserve de toutes les variables apparaissant dans une expression à gauche d'un appel de procédure, on peut assurer à l'utilisateur que l'effet de bord se produit "à droite" seulement.

Actuellement, la présence de l'option S.E. permet déjà de protéger de cette façon les paramètres effectifs et les indices de variables indicées se trouvant, dans les listes, à gauche d'un paramètre comportant un appel de procédure.

B3 - EXTENSIONS PREVISIBLES

Pour conclure cet ouvrage, nous pensons que le langage ALGOL est appelé à évoluer dans les années à venir, dans un sens propre à augmenter ses possibilités et sa souplesse d'emploi.

L'exposé des principes de fonctionnement du compilateur PC a montré clairement que ces extensions seront le plus souvent facilement intégrables à ce programme. Nous pensons en particulier à l'introduction de nouveaux symboles de base du langage L_S, à la définition de nouvelles fonctions standard, etc ...

Dans les limites imposées par l'aspect technologique de la machine, nous estimons que l'évolution du langage concernant les déclarations de variables de types nouveaux (double précision, complexes, unités physiques ...) la définition d'une arithmétique booléenne véritable, le traitement efficace de fichiers alphabétiques, etc ... ne devrait pas offrir d'incompatibilité avec le fonctionnement actuel du compilateur.

Nous espérons que l'utilisateur pourra toujours profiter de la meilleure efficacité et de la plus grande généralité du langage source, même si ces deux notions sont parfois incompatibles, puisque dans tous les cas un choix lui sera laissé pour "orienter" l'exécution de son travail vers l'une ou vers l'autre.

B I B L I O G R A P H I E

1 - NAUR P. (Ed) et al. Report on the Algorithmic Language Algol 60. Comm. ACM 3(1960) N°5,299 - 314, ou Numer. Math., 2 (1960), 106 - 136 .

2 - GENUYS M. F. et al. Rapport sur le langage algorithmique Algol 60. (Traduction française de [1]), Chiffres 3 (1960), 1 - 44 .

3 - TAYLOR W. and al. , A Syntactical Chart of Algol 60 Comm. ACM 4, 9 (Sept. 1961), 393.

4 et 5 - Groupe Algol
INSTITUT DE MATHEMATIQUES APPLIQUEES DE GRENOBLE
Notes Techniques numéros 3, 4, 5, 6, 11, 15 (1961)

6 - NAUR P. (Ed) et al. Revised Report on the Algorithmic Language Algol 60, Comm. ACM 6 (1963), N° 1, 1 - 17.

7 - YERSHOV A. P. and al., Input Language for Automatic programming Systems, APIC Studies in Data Processing 3 (1963).

8 - NAUR P. The Design of the Gier Algol Compiler, BIT 3 (1963), N° 2 and 3, on Annal Review in Automatic Programming 4.

9 - BUMGARNER L.L. The Oak Ridge Algol Compiler for the Control Data Corporation 1604, July 1963.

10- INGERMAN P. Z. and MERNER J., Attached Revision of the Revised Algol Report. Non publié (24/4/63).

11- Groupe Algol, INSTITUT DE MATHEMATIQUES APPLIQUEES DE GRENOBLE
Codification des symboles Algol 60 ; Note technique N°17 (Fevrier 1962).

12- DOLOTTA T. A. Méthode d'édition d'un programme en langage symbolique. Proc. of the International Computation Center's Symposium on Symbolic Languages, Gordon and Breach - N. Y. (1962).

13- IRONS E. T. A Syntax directed Compiler for Algol 60 Comm. ACM 4 (1961), 51 - 55 .

14 - ALGOL : Bulletin N° 15, Regnecentralem, Copenhagen (1962, 3 - 51).

15 - DOLOTTA T.A. Les langages symboliques et leur édition, Chiffres 3 (1962), 149 - 174.

16 - ALGOL Manual der ALCOR - Gruppe, Elektronische Rechenanlagen 3 (1961), 7 - 5

17 - 7090 Data Processing System Reference Manual, IBM publication A22 - 6528 - 2, White Plains, N. Y. (1961).

18 - FORTRAN Assembly Program (FAP) for the IBM 7090 . IBM publication J.28. 6098, White Plains, N. Y. (1960), 13-15.

19 - Groupe Algol INSTITUT DE MATHEMATIQUES APPLIQUEES DE GRENOBLE Exposé rapide du compilateur destiné au calculateur BESM, d'après YERSHOV A.P. Note technique N°8 (Nov. 1961).

20 - Groupe ALGOL INSTITUT DE MATHEMATIQUES APPLIQUEES DE GRENOBLE Compilation des Expressions Arithmétiques simples, Note technique N°13 (DEC.61) et Compilation des expressions arithmétiques simples sur 60, Note technique N° 20 (Mars 1962).

21 - BURROUGHS Compilogram. Datamation (Fev. 1962)

22 - CHOMSKY N. Formal properties of Grammar, Hand book of Mathematical Psychology. ed D. Luce, E. Bush, E. Galanter -1963

23 - BOUSSARD J.C. Description d'une version réduite d'un compilateur ALGOL 60 sur calculateur IBM 7044/7090. Note technique Institut de Mathématiques Appliquées de Grenoble (Juillet 63).

24 - IBM 7090/7094 Operating Systems basic monitor (IBSYS) IBM publication White Plains, N. Y. 1962

25 - IBM 709/7090 Programming Systems : FORTRAN II Opérating under the 7090/7094 basic monitor (IBSYS) IBM publication, White Plains , N. Y. 1962

26 - IBM 7040/7044 Principles of operation IBM publication, A22 - 6649 - 2 White Plains, N.Y. 1962

27 - IBM 7040/7044 Operating Systems. Macro Assembly Program Language, IBM publication C28 - 6335, White Plains , N.Y. 1963

28 - IBM 7040/7044 Operating Systems, Input/ Output Control System. IBM publication C28 - 6309, White Plains , N.Y. 1963

29 - IBM 7040/7044 Operating System. Programmer's guide IBM publication C28 - 6318, White Plains, N.Y. 1963

30 - IBM 7040/7044 Operating System. System programmer's guide, IBM publication C28 - 66339, White Plains, N.Y. 1963

31 - IBM 7040/7044 Operating System. FORTRAN IV Language IBM publication C28- 6329, White Plains, N.Y. 1963

32 - BOLLIET L, GASTINEL N., LAURENT P.J., Un nouveau langage scientifique Algol - HERMANN, Paris 1964

33 - BOLLIET L. L'évolution des techniques de compilation des langages symboliques. 3e Congrès AFCALTI - TOULOUSE Mai 1963.

34 - BOUSSARD J.C. Réalisation d'un compilateur ALGOL 62 pour les machines IBM 7090/94 et 7040/44. 4e Congrès - VERSAILLES, Avril 1964

35 - AUROUX A. Incorporation d'un compilateur ALGOL 60 dans le système IBSYS/IBJOB 7040/44 . 4e Congrès AFCALTI VERSAILLES, Avril 1964.

A P P E N D I C E I

T A B L E S

Symbole	Code IBM octal	Code D octal
Espace	60	0
A	21	1
B	22	2
C	23	3
D	24	4
E	25	5
F	26	6
G	27	7
H	30	10
I	31	11
J	41	12
K	42	13
L	43	14
M	44	15
N	45	16
O	46	17
P	47	20
Q	50	21
R	51	22
S	62	23
T	63	24
U	64	25
V	65	26
W	66	27
X	67	30
Y	70	31
Z	71	32

Symbole perforé	Algol	Code IBM octal	Code D octal
0		0	40
1		1	41
2		2	42
3		3	43
4		4	44
5		5	45
6		6	46
7		7	47
8		10	50
9		11	51
.		33	52
:*	^10		53
+		20	54
-		40	55
*		54	61
/		61	62
**	↑		63
=		13	65
:=	:=		66
::	;		70
,		73	71
:		53	72
(		74	73
)		34	74
.(	[		75
).	]		76
'	sans équivalent	14	77

A.I.1 CORRESPONDANCE CODE D - CODE IBM

SPECIFICATION	POSITION MISE A 1
Paramètre formel	3
Appel par valeur	4
Entier sans signe	5
Etiquette	6
Aiguillage	7
Chaîne	8
Persistant	9
Réel	10
Entier	11
Booléen	12
Tableau	13
Procédure	14

A.I.3. CODIFICATION DES UNITES SYNTAXIQUES - IDENTIFICATEURS

SPECIFICATIONS	DESCRIPTION DU PROFIL
I - Identificateurs ordinaires	- Type et genre : cf. A.I.3 - Numérotation en 18 - 35
II- Autres identificateurs simples -Nombres purs	- Position 5 à 1 + type entier (position 11) ou réel (position 12) - Numérotation en 21 - 35, (0 si le nombre est 0)
- Valeurs logiques	- Position 14 à 0 si FAUX à 1 si VRAI - 15 - 35 nul - Positions 1 et 3 à 1 + type booléen (position 12)
III-Identificateurs de manoeuvre -Mémoire contenant un résultat -Mémoire contenant une adresse - Etiquette résultat d'une expression de désignation	- Position 2 à 1 + type (10, 11 ou 12) - Numérotation en 21 - 35 - Id. + position 15 - Position 6 à 1 - Position 3 à 1 si paramètre formel - Position 2 à 1 si valeur d'un aiguillage - Numérotation en 21 - 35 si étiquette simple. Sinon, 21 - 35 nul.
IV- Registres AC MQ	 Tout nul position 35 à 1

A.I.4. PROFILS QUALITATIFS APPARAISSANT EN E3 (PN).

Hiérarchies	Symboles
0	DEBUT (↓1 ↓2 ↓*3 ↓5 ↓*6 ...
1	FIN) ; FAIRE
2	,]
3	PAS JUSQUA TANTQUE : POUR
4	SI SINON
5	:= ALORS ALLER A
6	≡
7	⊃
8	∨

Hiérarchies	Symboles
9	∧
10	¬
11	< ≤ = ≠ >>>>>
12	+ - ⊕ ⊖
13	x / ÷
14	↑
15	[. 10

A.I.5. TABLE DE HIERARCHIES DES DELIMITEURS DANS PC_2

APPENDICE II

C O N V E N T I O N S

L I M I T A T I O N S

R E S T R I C T I O N S

II - 1 RESTRICTIONS DANS LE LANGAGE L_S

a) Etiquettes purement numériques interdites

b) Spécification obligatoire de tous les paramètres formels
Dans les deux cas, libellé d'erreur à l'édition.

c) Conformité des types de paramètres effectifs avec ceux des paramètres formels correspondant. Libellé à l'exécution, avec numéro d'instruction et symbole interne de la procédure.

d) Interdiction d'employer récursivement une procédure comportant des tableaux par valeur ou d'appeler par valeur des tableaux persistants.

II - 2 CONVENTIONS D'EMPLOI

a) Nombres purs

Ils sont considérés comme réels s'ils comportent une partie décimale (avec ou sans exposant) ou une partie exposant dotée d'un signe (+ ou -). Entiers dans tous les autres cas.

Exemple : $0,_{10}3$, $-4_{10}2$ sont entiers
0.0 , $_{10}+3$, $2_{10}-2$ sont réels

Relativement à c) de II - 1 , il y a erreur si un paramètre formel spécifié réel est remplacé par le paramètre effectif 0 .

b) Fonction puissance

Son résultat est toujours réel

c) Tableaux persistants

Ils sont toujours considérés à bornes non dynamiques, leur encombrement est calculé une fois pour toutes au premier passage sur leur déclaration.

d) Récursivité

- Si une procédure est appelée récursivement, tous ses paramètres par nom doivent être "évoqués" au moins une fois dans l'appel de niveau 1 .
- Si un ALLER A conduit à l'extérieur d'une procédure, celle-ci reste au même niveau de récursivité.
- Si, au niveau de récursivité n, un paramètre appelé par nom est remplacé par une variable locale à la procédure, c'est la variable du niveau n qui est employée.

c) Effet de bord

- Dans une expression, il n'est tenu compte actuellement d'aucune possibilité d'effet de bord .
- Dans une liste de paramètres effectifs appelés par valeur, il n'est pas tenu compte d'un effet de bord possible (cas d'un appel récursif en particulier).
- Dans une liste de paramètres effectifs de la fonction ECRIRE, dans une liste d'indices, et dans une liste de POUR , il est tenu compte d'un effet de bord éventuel par l'option "S.E." (non encore distribuée). L'effet de bord ne peut alors avoir lieu que "de la gauche vers la droite".

II - 3 PRINCIPALES LIMITATIONS

a) Encombrements maximaux :

- 12000 unités syntaxiques
- 2000 nombres ou parties de nombres purs
- 2500 déclarations simultanément valables
- 5000 occurences d'identificateurs simultanément valables

- 25 étiquettes équivalentes pour une même instruction générée
- 360 aiguillages
- 150 entrées par aiguillage
- 100 éléments dans une liste d'indices ou de paramètres de fonctions standard
- 100 éléments dans une liste de paramètres de fonctions déclarées
- 100 niveaux de blocs imbriqués, naturels ou artificiels
- 50 niveaux d'appels de procédures imbriqués
- 10 niveaux de déclarations de procédures imbriquées en cas de récursivité.
- 500 mémoires de manoeuvre statiques
- 100 structures parenthèsées imbriquées dans une expression.
- etc ...

Ces limitations ne sont pas indépendantes entre elles et il est très difficile de donner une limite effective aux programmes compilés.

Dans tous les cas, une installation de calcul dotée du compilateur peut, par réassemblage de celui-ci, modifier tout ou partie de ces limitations.

b) <u>Limitations définitives</u>

- 255 procédures déclarées
- 255 procédures standard
- 511 paramètres formels par procédure
- 12 premières lettres interprétées dans les identificateurs.

- Nombres purs entiers limités à 2^{27}
- Nombres purs réels limités à $2^{\pm 128}$
- 8 premiers chiffres interprétés dans une partie décimale
- etc ...

Ces limitations sont dûes à la constitution des machines employées ou au fonctionnement même du compilateur. Elles ne sont pas modifiables.

c) <u>Limitations à l'exécution</u>

- Variables entières ne donnant pas lieu à conversion ou à impression limitées à 2^{35} , sinon 2^{27}
- Variables réelles limitées à $2^{\pm 128}$
- Mémoire dynamique limitée en fonction de la taille du programme et de la machine.

II - 4 TABLE DES SYMBOLES PERMIS

AIGOL (I_S)	SYMBOLES PERMIS (Cartes perforées)	
≡	'IDENT'	'EQUIV' 'EQUI' 'EQV'
⊃	'IMPL'	
∨	'UNI'	'ØR' 'ØU'
∧	'INT'	'AND' 'ET'
¬	'NØN'	'NØT'
<	'INFER'	'LESS' LSTH'
≤	'INFEG'	'L=' 'NØTGREATER' 'LSTHEQ'
=	=	'EQUAL' 'EGAL'
≥	'SUPEG'	'S=' NØTLESS' 'GRTHEQ'
>	'SUPER'	'GREATER' 'GRTH'
≠	'NØNEG'	'NØTEQUAL' 'NØT='
+	+	
-	-	
×	*	
/	/	
÷	'/'	'DIVENT'
↑	**	'PØWER' 'EXP'
GO TO	'ALLER A'	'GØTØ'
IF	'SI'	'IF'
THEN	'ALØRS'	'THEN'
ELSE	'SINØN'	'ELSE'
FOR	'PØUR'	'FØR'
DO	'FAIRE'	'DØ'
TRUE	'VRAI'	'TRUE' 'V' '1'
FALSE	'FAUX'	'FALSE' 'F' '0'
PROCEDURE	'PRØCEDURE'	'PRØC' 'PRØCEDURES'
OWN	'PERSISTANT'	'ØWN' 'PERS' 'PERSISTANTS' 'REMANENT'
BOOLEAN	'BØØLEEN'	'BØØLEAN' 'B' 'BØØL' 'BØØLEENS' 'BØØLEENNE' 'BØØLEENNES'
REAL	'REEL'	'REAL' 'R' 'REELS' 'REELLE' 'REELLES'
INTEGER	'ENTIER'	'INTEGER' 'E' 'ENT' 'ENTIERS' 'ENTIERE' 'ENTIERES'
ARRAY	'TABLEAU'	'ARRAY' 'TAB' 'TABLEAUX'
SWITCH	'AIGUILLAGE'	'SWITCH' 'AIGU'
STRING	'CHAINE'	'STRING' 'CHAINES'
LABEL	'ETIQUETTE'	'LABEL' 'ETIQ' 'ETIQUETTES'
VALUE	'VALEUR'	'VALUE' 'VAL' 'VALEURS'
,	,	
.	.	
10	:*	'DIX'

ALGOL (L_S)	SYMBOLES PERMIS (Cartes perforées)		
:	:		
;	::	'PVIR'	
:=	:=		
STEP	'PAS'	'STEP'	
UNTIL	'JUSQUA'	'UNTIL'	'A'
WHILE	'TANTQUE'	'WHILE'	'TANDISQUE'
COMMENT	'CØMMENTAIRE'	'CØMMENT'	'CØMM'
space	espace		
(	(		
)	)		
[	.(	'ØCRØ'	'ØBRA'
]	).	'FCRØ'	'CBRA'
'	'('		
'	')'		
BEGIN	'DEBUT'	'BEGIN'	
END	'FIN'	'END'	
n'existe pas	'CØDE'		
n'existe pas	'FCØDE'		

II - 5 TABLE DES FONCTIONS STANDARD

FONCTIONS ARITHMETIQUES

IDENTIFICATEURS	CARACTERISTIQUES
ABS	1 paramètre entier ou réel - Résultat du même type
SIGN ou SIGNE	1 paramètre entier ou réel - Résultat entier, 0, 1 ou - 1
ENTIER	1 paramètre entier ou réel - Résultat entier.
CONVRE	1 paramètre entier ou réel - Résultat entier
CONVER	1 paramètre entier ou réel - Résultat réel
SQRT ou RAC2	1 paramètre entier ou réel - Résultat réel
SIN	1 paramètre entier ou réel - Résultat réel
COS	1 paramètre entier ou réel - Résultat réel
ARCTAN	1 paramètre entier ou réel - Résultat réel
ARCSIN	1 paramètre entier ou réel - Résultat réel
ARCCOS	1 paramètre entier ou réel - Résultat réel
LN	1 paramètre entier ou réel - Résultat réel
LOG10	1 paramètre entier ou réel - Résultat réel
EXP	1 paramètre entier ou réel - Résultat réel
TANH	1 paramètre entier ou réel - Résultat réel

FONCTIONS DE COMMUNICATIONS

ALGOL	sans type	Liste de paramètres, 1er chaîne communication avec programme ALGOL assemblé extérieurement
ALGOLE	type entier	
ALGOLB	type booléen	
ALGOLR	type réel	
FORTRAN	sans type	Liste de paramètres, 1er chaîne communication avec programme FORTRAN assemblé extérieurement
FORTRANE	type entier	
FORTRANB	type booléen	
FORTRANR	type réel	
REFERENCE	sans type	communication avec programme ALGOL principal (1 paramètre chaîne, 1 paramètre procédure).

FONCTIONS D'ENTREES - SORTIES

EDONNEE	type entier	sans paramètre - Entrée immédiate
BDONNEE	type booléen	
RDONNEE	type réel	
LIRE	sans type	- liste de paramètres. Lecture de 1 carte
ECRIRE	sans type	- liste de paramètres. Ecriture de 1 ligne
ENTREE	sans type	- liste de paramètres. Lecture avec MODELE
SORTIE	sans type	- liste de paramètres. Ecriture avec MODELE
MODELE	sans type	1 paramètre chaîne - procédure d'édition
HEURE	sans paramètre, sans type, impression heure	
REBOBINER	sans type, liste de paramètres	Rebobinage des bandes indiquées
ECRIRE FDB		Ecriture fin de bloc sur les bandes indiquées
ESP ARRIERE		Espace arrière des nombres d'enregistrements indiqués sur les bandes indiquées

APPENDICE III

SUR L'UTILISATION PRATIQUE DU COMPILATEUR

III - 1 CONVENTIONS DE PERFORATION

a) Symboles courts

Ils sont formés d'un ou deux caractères apparaissant sur le clavier des perforatrices IBM 026. Aucun blanc n'est permis entre les deux caractères d'un même symbole court. Réciproquement, si deux caractères représentent chacun un symbole et qu'ensemble ils en forment un troisième, toute perforation sans espace des deux caractères représente ce troisième symbole.

Exemple : ::: équivaut à ;: en langage de référence.

b) Symboles longs (soulignés dans L_S)

Ils doivent être encadrés de deux caractères ' .

c) 'CODE' doit être perforé de manière à ce que le dernier caractère ' soit en colonne 72 d'une carte. 'FCODE' doit être perforé de façon à ce que le premier caractère ' soit en colonne 1 d'une carte.

d) Sauf dans les cas cités en a) , à l'intérieur des chaînes alphabétiques et des procédures en code machine, les caractères espaces sont ignorés dans les cartes. Celles-ci peuvent être perforées de la colonne 1 à la colonne 72, les colonnes 73 - 80 pouvant servir à leur identification.

III - 2 INFORMATIONS DONNEES AU SYSTEME

Actuellement un "travail" ALGOL sur machine 7090/94 (système IBSYS) se présente de la façon suivante :

1	7	16
*	JOB	numéro de travail
*	XEQ	(optionnelle)
*	LIST	(optionnelle)
*	VERIF	(optionnelle)
*	RECUR	(optionnelle)

```
ALGOL
        { Texte de  P_S
FALGOL
*       DATA
        { Données éventuelles
7/8
```

Les cartes 7/8 sont dites "fin de bloc".

La carte "JOB" identifie le travail. Les autres cartes avec * en colonne 1 (cartes "moniteur" FORTRAN ou ALGOL) ne sont pas obligatoires. Elles spécifient les options de l'utilisateur.

La carte "XEQ" est nécessaire si l'exécution du programme généré est demandée. Dans ce cas, la carte "DATA" est également obligatoire.

Un autre travail ALGOL peut suivre un tel travail. Le premier d'une même suite doit être précédé de la carte

$EXECUTE ALGOL

qui est une carte "moniteur IBSYS" .

Tout changement de sous-système après une suite de travaux ALGOL doit également être suivi d'une carte $EXECUTE.

Sur machine 7040/44 (système IBSYS - IBJOB), un travail ALGOL est constitué de la façon suivante :

```
1          8          16
$JOB                  numéro de travail
$IBJOB                   options
$ALGOL     Nom           options
           { Texte de  P_S
FALGOL
$ENTRY
           { données éventuelles
$IBSYS
```

Rappelons que la carte $IBJOB peut comporter trois options :

MAP qui fait imprimer la configuration du programme en mémoire rapide à l'exécution.

NOGO qui empêche cette exécution

DECK qui fait perforer les cartes en code binaire équivalentes au programme P_S .

Les options de la carte $ALGOL sont

RECUR , VERIF , REF , LIST

dont nous avons vu la signification.

Enfin, la carte $ENTRY est obligatoire si l'exécution, ou / et l'option MAP , est demandée.

Rappelons que l'absence d'un point virgule en fin du texte de P_S entraîne l'arrêt du compilateur après le passage PC_1 .

Si une carte $7/8$ en 7090/94 ou $IBSYS en 7040/44 apparaît dans le texte de P_S , le travail est abandonné et un libellé d'erreur approprié est imprimé.

III - 3 MISE AU POINT DES PROGRAMMES

Nous avons vu dans le texte de l'ouvrage les possibilités de mise au point d'un programme P_S disponibles aux différents niveaux rencontrés.

On trouvera en appendice IV un exemple de programme mis au point en quatre passages.

1er passage : erreurs à l'édition et identificateurs non déclarés

2e passage : erreur à la compilation proprement dite

3e passage : erreur à l'exécution dûe à une mauvaise utilisation d'une variable indicée. Dans ce passage, on utilise le dictionnaire de références (REF) qui montre que l'instruction où se produit l'erreur (numéro 271) se situe au début de la boucle B3 (instruction 263). On voit facilement que le tableau T est exploité avec l'indice 0 , trop petit.

4e passage : bon résultat. Si le résultat avait été faux, on aurait mis en oeuvre des écritures intermédiaires où une impression partielle de l'état de la mémoire (DUMP).

N° d'ordre

T H E S E S

présentées à la

FACULTE DES SCIENCES DE GRENOBLE

pour obtenir

LE GRADE DE DOCTEUR ES SCIENCES APPLIQUEES

par

J. C. BOUSSARD

Ingénieur I.R.G.

Licencié ès Sciences

Première thèse :

ETUDE ET REALISATION D'UN COMPILATEUR
ALGOL 60 SUR CALCULATRICE ELECTRONIQUE
DU TYPE IBM 7090/94 ET 7040/44

Deuxième thèse :

PROPOSITIONS DONNEES PAR LA FACULTE

Thèses soutenues le Juin 1964 devant la commission d'examen :

Jury
- MM. J. KUNTZMANN, Président
- B. VAUQUOIS, N. GASTINEL, J. ARSAC — Examinateurs

TABLE DES MATIERES

INTRODUCTION 1

PREMIERE PARTIE - PASSAGE EDITEUR

Chapitre I - Définitions et généralités

1 - Les tâches du programme éditeur 8

2 - Généralités sur la chaîne codée 9

3 - Intérêt du passage éditeur 11

Chapitre II - Réalisation du passage éditeur

A - Généralités 14

B - L'ensemble de gestion 17

C - L'ensemble de traitement 23

D - Algorithmes de traitement 40

E - Sous-programmes de traitement 58

F - Tables de traitement 64

G - Séquence d'arrêt de traitement 67

Chapitre III - Programmes d'erreurs 71

Chapitre IV - Conclusion - Dicussion

A - Résultats obtenus 80

B - Discussion 81

DEUXIEME PARTIE - PASSAGE COMPILATEUR

Chapitre I - Généralités

A - Les fonctions du passage compilateur 84

B - Principes de fonctionnement 88

C - Partie analytique 98

D - Partie générative 102

E - Algorithme général 106

F - Organisation 110

TYPE	SYMBOLE DE BASE	CODES positions 3 à 14
Opérateur séquentiel	ALLER A	0001
	SI	0002
	ALORS	0004
	SINON	0005
	POUR	0010
	FAIRE	0011
Opérateur Logique	$\equiv$	0420
	$\supset$	0440
	$\vee$	0460
	$\wedge$	0500
	$\neg$	0520
Opérateur de Relation	$<$	1141
	$\leqslant$	1142
	$\geqslant$	1144
	$=$	1146
	$>$	1150
	$\neq$	1151
Opérateur Arithmétique	+	2161
	-	2162
	x	2201
	/	2202
	$\div$	2204
	$\uparrow$	2220
Valeur Logique	FAUX	4004
	VRAI	4005

TYPE	SYMBOLE DE BASE	CODES positions 3 à 14
Déclarateur	BOOLEEN	5001
	ENTIER	5002
	REEL	5004
	AIGUILLAGE	5006
	TABLEAU	5007
	PERSISTANT	5010
	PROCEDURE	5011
Spécifieur	CHAINE	5401
	ETIQUETTE	5402
	VALEUR	5404
Séparateur	PAS	6001
	JUSQUA	6002
	TANTQUE	6003
	:=	6004
	:	6005
	,	6006
	;	6007[1]
	.	6010
	10	6011
	COMMENTAIRE	Inexistant
	Espace	
Crochet	DEBUT	6400
	FIN	6401
	(	6402
	)	6403
	[	6404
	]	6405
	‘	6406
	’	6407
	CODE	6410
	FCODE	6411

A.I.2. CODIFICATION DES UNITES SYNTAXIQUES - SYMBOLES DE BASE -

1 - On ajoute 1 en postion 15 si fin de procédure