Conception d'un outil de mise au point de programmes écrits en langage de haut niveau : application au langage PL/1

Maurice Rey

▶ To cite this version:

Maurice Rey. Conception d'un outil de mise au point de programmes écrits en langage de haut niveau : application au langage PL/1. Langage de programmation [cs.PL]. 1974. dumas-00293477

HAL Id: dumas-00293477
https://dumas.ccsd.cnrs.fr/dumas-00293477v1

Submitted on 4 Jul 2008

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

THESE

présentée au

CENTRE UNIVERSITAIRE

D'EDUCATION ET DE FORMATION DES ADULTES DE GRENOBLE

pour obtenir le titre

d' INGENIEUR du CONSERVATOIRE NATIONAL des ARTS et METIERS

par

Maurice REY

—— o ——

Conception d'un outil de mise au point de programmes écrits en langage de haut niveau Application au langage PL/1

—— o ——

Thèse soutenue le 22 mars 1974 devant la commission d'examen

Président	Monsieur	L. Bolliet
Président adjoint	Monsieur	P. Namian
Examinateurs	Messieurs	M. Lucas
		M. Bellot

UNIVERSITE SCIENTIFIQUE
ET MEDICALE DE GRENOBLE

LISTE DES PROFESSEURS

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

PROFESSEURS TITULAIRES

	Nom	Discipline
MM.	ANGLES D'AURIAC Paul	Mécanique des fluides
	ARNAUD Georges	Clinique des maladies infectieuses
	ARNAUD Paul	Chimie
	AUBERT Guy	Physique
	AYANT Yves	Physique approfondie
Mme	BARBIER Marie-Jeanne	Electrochimie
MM.	BARBIER Jean-Claude	Physique expérimentale
	BARBIER Reynold	Géologie appliquée
	BARJON Robert	Physique nucléaire
	BARNOUD Fernand	Biosynthèse de la cellulose
	BARRA Jean-René	Statistiques
	BARRIE Joseph	Clinique chirurgicale
	BENOIT Jean	Radioélectricité
	BERNARD Alain	Mathématiques Pures
	BESSON Jean	Electrochimie
	BEZES Henri	Chirurgie générale
	BLAMBERT Maurice	Mathématiques Pures
	BOLLIET Louis	Informatique (IUT B)
	BONNET Georges	Electrotechnique
	BONNET Jean-Louis	Clinique ophtalmologique
	BONNET-EYMARD Joseph	Pathologie médicale
	BONNIER Etienne	Electrochimie Electrométallurgie
	BOUCHERLE André	Chimie et Toxicologie
	BOUCHEZ Robert	Physique nucléaire
	BOUSSARD Jean-Claude	Mathématiques Appliquées
	BRAVARD Yves	Géographie
	BRISSONNEAU Pierre	Physique du solide
	BUYLE-BODIN Maurice	Electronique
	CABANAC Jean	Pathologie chirurgicale
	CABANEL Jean	Clinique rhumatologique et hydrologie
	CALAS François	Anatomie
	CARRAZ Gilbert	Biologie animale et pharmacodynamie
	CAU Gabriel	Médecine légale et Toxicologie
	CAUQUIS Georges	Chimie organique
	CHABAUTY Claude	Mathématiques Pures
	CHARACHON Robert	Oto-Rhino-Laryngologie
	CHATEAU Robert	Thérapeutique
	CHENE Marcel	Chimie papetière
	COEUR André	Pharmacie chimique
	CONTAMIN Robert	Clinique gynécologique
	COUDERC Pierre	Anatomie Pathologique
	CRAYA Antoine	Mécanique

Mme	DEBELMAS Anne-Marie	Matière médicale
MM.	DEBELMAS Jacques	Géologie générale
	DEGRANGE Charles	Zoologie
	DESRE Pierre	Métallurgie
	DESSAUX Georges	Physiologie animale
	DODU Jacques	Mécanique appliquée
	DOLIQUE Jean-Michel	Physique des plasmas
	DREYFUS Bernard	Thermodynamique
	DUCROS Pierre	Cristallographie
	DUGOIS Pierre	Clinique de Dermatologie et Syphiligraphie
	FAU René	Clinique neuro-psychiatrique
	FELICI Noël	Electrostatique
	GAGNAIRE Didier	Chimie physique
	GALLISSOT François	Mathématiques Pures
	GALVANI Octave	Mathématiques Pures
	GASTINEL Noël	Analyse numérique
	GEINDRE Michel	Electroradiologie
	GERBER Robert	Mathématiques Pures
	GIRAUD Pierre	Géologie
	KLEIN Joseph	Mathématiques Pures
Mme	KOFLER Lucie	Botanique et Physilogie végétale
MM.	KOSZUL Jean-Louis	Mathématiques Pures
	KRAVTCHENKO Julien	Mécanique
	KUNTZMANN Jean	Mathématiques appliquées
	LACAZE Albert	Thermodynamique
	LACHARME Jean	Biologie végétale
	LAJZEROWICZ Joseph	Physique
	LATREILLE René	Chirurgie générale
	LATURAZE Jean	Biochimie pharmaceutique
	LAURENT Pierre-Jean	Mathématiques appliquées
	LEDRU Jean	Clinique médicale B
	LLIBOUTRY Louis	Géophysique
	LOUP Jean	Géographie
Mle	LUTZ Elisabeth	Mathématiques Pures
MM.	MALGRANGE Bernard	Mathématiques Pures
	MALINAS Yves	Clinique obstétricale
	MARTIN-NOEL Pierre	Seméiologie médicale
	MASSEPORT Jean	Géographie
	MAZARE Yves	Clinique médicale A
	MICHEL Robert	Minéralogie et Pétrographie
	MOURIQUAND Claude	Histologie
	MOUSSA André	Chimie nucléaire
	NEEL Louis	Physique du solide
	OZENDA Paul	Botanique
	PAUTHENET René	Electrotechnique
	PAYAN Jean-Jacques	Mathématiques Pures
	PEBAY-PEYROULA Jean-Claude	Physique
	PERRET René	Servomécanismes
	PILLET Emile	Physique industrielle
	RASSAT André	Chimie systématique
	RENARD Michel	Thermodynamique
	REULOS René	Physique industrielle
	RINALDI Renaud	Physique
	ROGET Jean	Clinique de pédiatrie et de puériculture
	SANTON Lucien	Mécanique
	SEIGNEURIN Raymond	Microbiologie et Hygiène
	SENGEL Philippe	Zoologie
	SILBERT Robert	Mécanique des fluides
	SOUTIF Michel	Physique générale

MM.	TANCHE Maurice	Physiologie
	TRAYNARD Philippe	Chimie générale
	VAILLAND François	Zoologie
	VALENTIN Jacques	Physique nucléaire
	VAUQUOIS Bernard	Calcul électronique
Mme	VERAIN Alice	Pharmacie galénique
M.	VERAIN André	Physique
Mme	VEYRET Germaine	Géographie
MM.	VEYRET Paul	Géographie
	VIGNAIS Pierre	Biochimie médicale
	YOCCOZ Jean	Physique nucléaire théorique

PROFESSEURS ASSOCIES

MM.	BULLEMER Bernhard	Physique
	HANO JUN-ICHI	Mathématiques Pures
	STEPHENS Michaël	Mathématiques appliquées

PROFESSEURS SANS CHAIRE

MM.	BEAUDOING André	Pédiatrie
Mme	BERTRANDIAS Françoise	Mathématiques Pures
MM.	BERTRANDIAS Jean-Paul	Mathématiques appliquées
	BIAREZ Jean-Pierre	Mécanique
	BONNETAIN Lucien	Chimie minérale
Mme	BONNIER Jane	Chimie générale
MM.	CARLIER Georges	Biologie végétale
	COHEN Joseph	Electrotechnique
	COUMES André	Radioélectricité
	DEPASSEL Roger	Mécanique des fluides
	DEPORTES Charles	Chimie minérale
	GAUTHIER Yves	Sciences biologiques
	GAVEND Michel	Pharmacologie
	GERMAIN Jean-Pierre	Mécanique
	GIDON Paul	Géologie et Minéralogie
	GLENAT René	Chimie organique
	HACQUES Gérard	Calcul numérique
	JANIN Bernard	Géographie
Mme	KAHANE Josette	Physique
MM.	MULLER Jean-Michel	Thérapeutique
	PERRIAUX Jean-Jacques	Géologie et Minéralogie
	POULOUJADOFF Michel	Electrotechnique
	REBECQ Jacques	Biologie (CUS)
	REVOL Michel	Urologie
	REYMOND Jean-Charles	Chirurgie générale
	ROBERT André	Chimie papetière
	DE ROUGEMONT Jacques	Neurochirurgie
	SARRAZIN Roger	Anatomie et chirurgie
	SARROT-REYNAULD Jean	Géologie
	SIBILLE Robert	Construction mécanique
	SIROT Louis	Chirurgie générale
Mme	SOUTIF Jeanne	Physique générale

MAITRES DE CONFERENCES ET MAITRES DE CONFERENCES AGREGES

Mle	AGNIUS-DELORD Claudine	Physique pharmaceutique
	ALARY Josette	Chimie analytique
MM.	AMBLARD Pierre	Dermatologie
	AMBROISE-THOMAS Pierre	Parasitologie
	ARMAND Yves	Chimie
	BEGUIN Claude	Chimie organique
	BELORIZKY Elie	Physique
	BENZAKEN Claude	Mathématiques appliquées
	BILLET Jean	Géographie
	BLIMAN Samuel	Electronique (EIE)
	BLOCH Daniel	Electrotechnique
Mme	BOUCHE Liane	Mathématiques (CUS)
MM.	BOUCHET Yves	Anatomie
	BOUVARD Maurice	Mécanique des fluides
	BRODEAU François	Mathématiques (IUT B)
	BRUGEL Lucien	Energétique
	BUISSON Roger	Physique
	BUTEL Jean	Orthopédie
	CHAMBAZ Edmond	Biochimie médicale
	CHAMPETIER Jean	Anatomie et organogénèse
	CHIAVERINA Jean	Biologie appliquée (EFP)
	CHIBON Pierre	Biologie animale
	COHEN-ADDAD Jean-Pierre	Spectrométrie physique
	COLOMB Maurice	Biochimie médicale
	CONTE René	Physique
	COULOMB Max	Radiologie
	CROUZET Guy	Radiologie
	DURAND Francis	Métallurgie
	DUSSAUD René	Mathématiques (CUS)
Mme	ETERRADOSSI Jacqueline	Physiologie
MM.	FAURE Jacques	Médecine légale
	GENSAC Pierre	Botanique
	GIDON Maurice	Géologie
	GRIFFITHS Michaël	Mathématiques appliquées
	GROULADE Joseph	Biochimie médicale
	HOLLARD Daniel	Hématologie
	HUGONOT Robert	Hygiène et Médecine préventive
	IDELMAN Simon	Physiologie animale
	IVANES Marcel	Electricité
	JALBERT Pierre	Histologie
	JOLY Jean-René	Mathématiques Pures
	JOUBERT Jean-Claude	Physique du solide
	JULLIEN Pierre	Mathématiques Pures
	KAHANE André	Physique générale
	KUHN Gérard	Physique
	LACOUME Jean-Louis	Physique
Mme	LAJZEROWICZ Jeannine	Physique
MM.	LANCIA Roland	Physique atomique
	LE JUNTER Noël	Electronique
	LEROY Philippe	Mathématiques
	LOISEAUX Jean-Marie	Physique nucléaire
	LONGEQUEUE Jean-Pierre	Physique nucléaire
	LUU DUC Cuong	Chimie organique
	MACHE Régis	Physiologie végétale
	MAGNIN Robert	Hygiène et Médecine préventive
	MARECHAL Jean	Mécanique
	MARTIN-BOUYER Michel	Chimie (CUS)

MM.	MAYNARD Roger	Physique du solide
	MICHOULIER Jean	Physique (IUT A)
	MICOUD Max	Maladies infectieuses
	MOREAU René	Hydraulique (INP)
	NEGRE Robert	Mécanique
	PARAMELLE Bernard	Pneumologie
	PECCOUD François	Analyse (IUT B)
	PEFFEN René	Métallurgie
	PELMONT Jean	Physiologie animale
	PERRET Jean	Neurologie
	PERRIN Louis	Pathologie expérimentale
	PFISTER Jean-Claude	Physique du solide
	PHELIP Xavier	Rhumatologie
Mle	RIERY Yvette	Biologie animale
MM.	RACHAIL Michel	Médecine interne
	RACINET Claude	Gynécologie et obstétrique
	RENAUD Maurice	Chimie
	RICHARD Lucien	Botanique
Mme	RINAUDO Marquerite	Chimie macromoléculaire
MM.	ROMIER Guy	Mathématiques (IUT B)
	SHOM Jean-Claude	Chimie générale
	STIEGLITZ Paul	Anesthésiologie
	STOEBNER Pierre	Anatomie pathologique
	VAN CUTSEM Bernard	Mathématiques appliquées
	VEILLON Gérard	Mathématiques appliquées (INP)
	VIALON Pierre	Géologie
	VOOG Robert	Médecine interne
	VROUSSOS Constantin	Radiologie
	ZADWORNY François	Electronique

MAITRES DE CONFERENCES ASSOCIES

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

CHARGES DE FONCTIONS DE MAITRES DE CONFERENCES

Mme	BERIEL Hélène	Physilogie
Mme	RENAUDET Jacqueline	Microbiologie

Fait le 30 mai 1972.

Je tiens à remercier,

Monsieur le Professeur L.BOLLIET qui a bien voulu me faire l'honneur de présider le Jury de cette Thèse et qui a toujours montré la plus grande bienveillance pour mon travail,

Monsieur le Professeur P.NAMIAN qui a accepté d'être Président Adjoint du Jury,

Messieurs M.BELLOT et M.LUCAS pour leurs suggestions, leurs conseils et leurs précieuses critiques apportées au manuscrit,

Tous mes collègues du Centre Interuniversitaire de Calcul de Grenoble qui m'ont aidé dans mon travail, et plus particulièrement, mes voisins de bureau, J.C.CALLEGHER et J.GUILLOU.

Je voudrais aussi remercier,

Le service dactylographique qui a assuré la frappe de ce texte,

Le service de reprographie qui a réalisé ce document.

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

L'apparition des systèmes conversationnels, utilisés en temps partagé a profondément modifié les habitudes des utilisateurs. En effet, par rapport aux systèmes classiques à traitement par lot, les systèmes conversationnels à temps partagé permettent une utilisation plus souple et un accès plus facile à l'ordinateur; de plus, ils ont réduit de façon très appréciable les temps d'attente des résultats.

Bien qu'originellement (juillet 1968), ces systèmes ne disposaient que d'un nombre réduit de composants (un assembleur et un compilateur FORTRAN, dans le cas du système CMS), les possibilités nouvelles qu'ils offrent ont assuré leur succès. De ce fait, le nombre des composants disponibles a très rapidement augmenté à la suite des travaux conjugués des utilisateurs et des responsables du développement de ces systèmes.

Pour notre part, nous avons dans une première étape, adapté au système CMS, un compilateur PL/1 qui provient du système OS/360; ce faisant, nous avons développé une intéraction directe entre le compilateur et l'utilisateur, en présentant immédiatement les erreurs syntaxiques détectées dans le programme, sur le terminal de l'utilisateur.

Le cycle proposé par le système pour l'écriture d'un programme est alors :

- création conversationnelle du fichier contenant le programme source,
- compilation du programme et sortie immédiate des indications d'erreur,
- correction du programme par modification conversationnelle du fichier,
- nouvelle compilation du programme.

La rapidité d'obtention d'un programme syntaxiquement correct est donc augmentée par l'utilisation d'un système conversationnel; par contre, les méthodes de mise au point de programmes écrits en langage de

haut niveau n'ont que peu évolué. La détection et la correction des erreurs de logique s'effectuent par essais successifs du programme, ce qui conduit à autant de reprise du cycle précédemment cité: chaque essai ne permet la détection que d'un nombre limité d'erreurs.

Bien que dans le système CMS, une aide à la mise au point appelée DEBUG soit disponible, elle ne peut satisfaire que les programmes écrits en langage d'assemblage. L'accompagnateur d'exécution DEBUG autorise, pendant l'exécution du programme:

- l'arrêt de l'exécution sur des instructions-machine nommées par l'utilisateur,
- la consultation et la modification du contenu de la mémoire,
- la reprise de l'exécution.

En d'autres termes, l'exécution d'un programme peut être contrôlée de façon conversationnelle et, en cours d'exécution, l'utilisateur peut influer sur le déroulement en modifiant certaines données du programme.

Pour les langages de haut niveau, à notre connaissance, un outil de mise au point fût développé pour FORTRAN: l'accompagnateur FORTBUG[J

A notre tour, nous avons défini et construit un nouvel outil: l'accompagnateur BUGPLI offre aux utilisateurs qui programment en PL/1, des possibilités de mise au point intéractives et symboliques.

Les principaux problèmes rencontrés lors de la construction de cet outil sont liés à la structure même du langage PL/1 et à la notion de bloc que possède ce langage. Il nous a fallu créer les informations nécessaires pour que notre outil puisse accéder par nom aux variables du programme PL/1 et, définir un "adressage symbolique" des instructions de ce programme.

Notre but, avec l'accompagnateur BUGPLI, est donc d'offrir à tout utilisateur, la possibilité de :

- suspendre l'exécution de son programme à une instruction PL/1 qu'il nomme,
- consulter et modifier le contenu des variables du programme,
- placer de nouveaux points d'arrêt dans le programme,
- reprendre l'exécution du programme.

Pour permettre une meilleure approche du sujet, nous avons découpé notre présentation en trois parties.

Dans la première partie, nous exposons brièvement le système conversationnel CMS et les problèmes posés par l'implantation du compilateur PL/1 dans ce système. Nous détaillons, par exemple, les solutions proposées pour permettre de traiter correctement les entrées-sorties relatives au fichier utilitaire.

Dans la deuxième partie, nous présentons les possibilités que notre accompagnateur d'exécution offre aux utilisateurs.

Dans la troisième partie, nous développons la structure interne de notre outil de mise au point; nous abordons en particulier certains problèmes rencontrés et les solutions que nous proposons pour construire les liens entre le programme PL/1 et la bibliothèque d'exécution d'une part et l'accompagnateur d'exécution d'autre part.

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

PL/1 sous CMS

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

PL/1 SOUS CMS

I INTRODUCTION AU SYSTEME CMS.

II LE COMPILATEUR PL/1.

III IMPLANTATION DU COMPILATEUR-OS DANS LE SYSTEME CMS.

III-1 Liste de paramètres en CMS.

III-2 Fonctions-système nécessaires au compilateur.

III-2-1 Acquisition et libération dynamique de mémoire.
III-2-2 Chargeur.
III-2-3 Méthodes d'accès.

III-3 L'interface entre le superviseur et le compilateur.

III-3-1 Liste de paramètres.
III-3-2 Appel du compilateur.
III-3-3 Initialisation de la mémoire libre.
III-3-4 Intéraction entre le compilateur et l'utilisateur.
III-3-5 Traitement des options propres à CMS
III-3-6 Itération sur plusieurs fichiers-source.
III-3-7 Simulation des méthodes d'accès pour le fichier utilitaire SYSUT1.

IV GENERATION DU COMPILATEUR.

V CONTROLE DE LA VALIDITE DE LA STRUCTURE DE RECOUVREMENT.

VI EXECUTION DE PROGRAMMES PL/1.

VI-1 La bibliothèque d'exécution .

VI-2 L'interface d'exécution.

VI-3 Activation d'un programme PL/1.

VI-4 Problèmes posés par l'exécution d'un programme PL/1.

I - INTRODUCTION AU SYSTEME CMS - [I1] [I3]

CMS est un système conversationnel mono-utilisateur qui s'exécute, soit sur un ordinateur réel, soit plus généralement, dans une machine virtuelle générée par le système CP/67 [I2,I3]. Les deux systèmes, CMS et CP/67, ont été développés dans les années 1966, au Centre Scientifique IBM de Cambridge (Massachussetts - USA).

Le système CMS possède, entre autres :

1. Un noyau résident qui contient des fonctions telles que :
   - gestion des interruptions,
   - gestion de mémoire (attribution et libération),
   - chargement soit absolu, soit translatable,
   - gestion des opérations d'entrée-sortie,
   - etc...

2. Un environnement de mise au point (DEBUG) pour les programmes écrits en langage d'assemblage.

3. Un mécanisme de gestion de fichiers définissant une méthode d'accès direct; l'identification d'un fichier-CMS résidant sur disque est constituée du triplet "nom, type, mode" et chaque enregistrement du fichier est accessible directement, à l'aide de son numéro d'ordre.

4. Un éditeur conversationnel (EDIT) permettant la création, la modification et la consultation de fichiers. [A1].

5. Un mécanisme de macro-langage (EXEC) qui permet de créer de nouvelles commandes, à partir de commandes primitives de CMS et de les enregistrer dans un fichier de type EXEC.

6. Différents assembleurs et compilateurs (PL/1, Algol, Fortran, AlgolW). Il est à noter que ces assembleurs ou compilateurs n'ont pas été écrits pour le système CMS et, de ce fait, ils ne tiennent pas compte de la structure du système, ou du fait que celui-ci s'exécute sur une machine virtuell c'est-à-dire dans un contexte de mémoire paginée. Les assembleurs ou compilateurs sont généralement ceux fournis avec les différentes versions du système OS/360.

La structure de la mémoire d'une machine utilisant le système CMS est illustrée par la figure I.a. Tous les assembleurs et compilateurs s'exécutent dans la zone utilisateur.

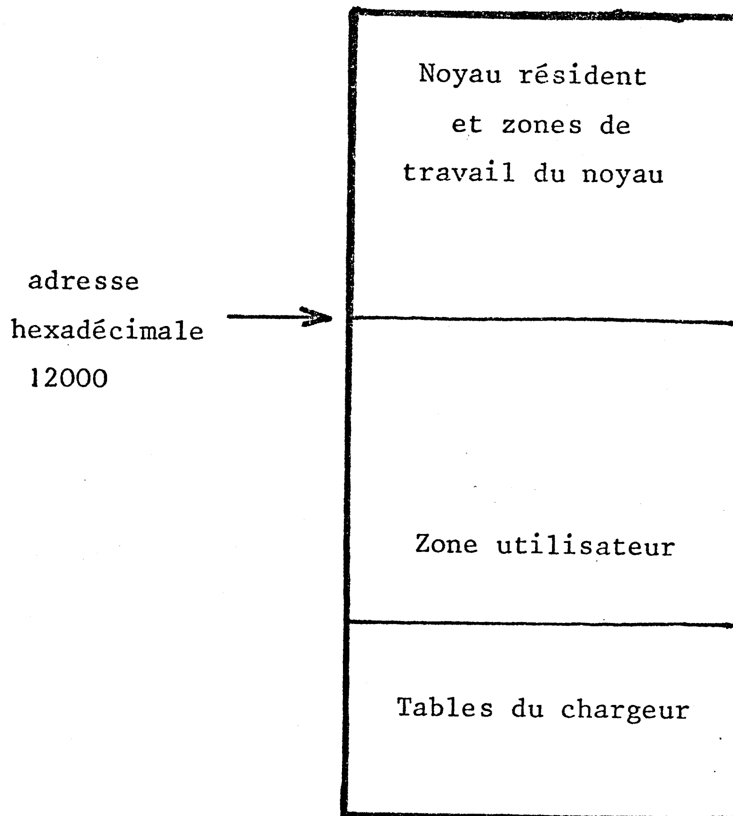


FIGURE I.a - La mémoire d'une machine CMS

II - LE COMPILATEUR PL/1 - [I4]

Ce compilateur de niveau F (version 5) correspond à la version 20.1 du système OS/360.

Comme pour tout compilateur, le programme source est constitué soit par un paquet de cartes, soit par un fichier sur disque ou sur bande contenant des images de cartes.

Le résultat de la compilation est essentiellement deux fichiers : un fichier contenant le "code-machine" généré et un fichier contenant la liste du programme PL/1, suivie éventuellement de la liste des erreurs et de la table des références.

Pour réaliser la traduction, le compilateur utilise un fichier utilitaire appelé SYSUT1 dans lequel il range des résultats intermédiaires.

Le compilateur est constitué de 253 programmes que nous appelons phases; chaque phase a une fonction logique.

Etant donné sa taille très importante, le compilateur ne peut évidemment pas prendre place en un seul bloc en mémoire et, de ce fait, il est organisé suivant une structure de recouvrement (OVERLAY) qui peut être schématisée par la figure II.a.

Pour faciliter la représentation, la figure est divisée en quatre tranches verticales; les noms des phases sont notés symboliquement, par exemple les deux caractères AA symbolisent la phase IEMAA.

La structure de recouvrement est telle que :

- les phases dont les noms figurent sur une même horizontale sont présentes en mémoire en même temps,

- les phases dont les noms figurent sur une même verticale ne peuvent théoriquement, pas être en mémoire en même temps.

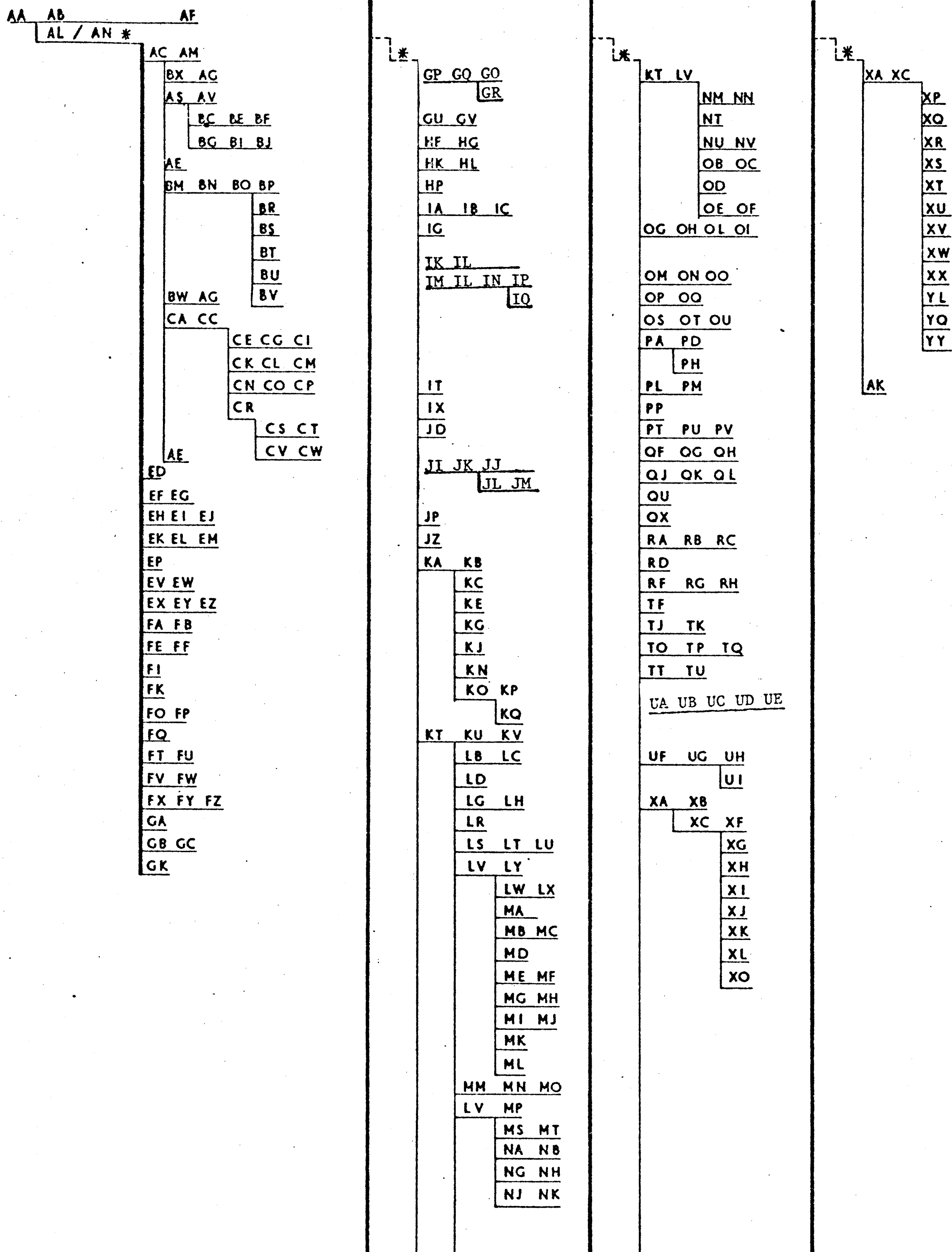


FIGURE II.a

Structure de recouvrement du compilateur

Le compilateur PL/1 étant défini pour être utilisé dans le système OS/360, sa structure de recouvrement est construite par l'éditeur de liens (LINK EDIT) du système. Pour cela, les programmes sont chargés en mémoire, les liaisons entre programmes sont établies (résolution des adresses externes) et un module translatable est créé pour chaque phase du compilateur; un tel module pourra par la suite, être implanté n'importe où en mémoire.

Ainsi, lors de l'exécution du compilateur, les différents modules, qui correspondent aux phases, se chargent dynamiquement en mémoire par l'intermédiaire des fonctions LINK, XCTL, RETURN, LOAD, DELETE. Ces fonctions sont exécutées à la suite d'appels privilégiés (SVC) au superviseur [I8,I9].

LINK (SVC 6) charge une phase et lui donne le contrôle.

XCTL (SVC 7) charge une phase à la place de la phase précédente et lui donne le contrôle.

RETURN (SVC 3) indique que la phase en cours est terminée et doit disparaître de la mémoire. Cette phase a été précédemment chargée soit par LINK soit par XCTL.

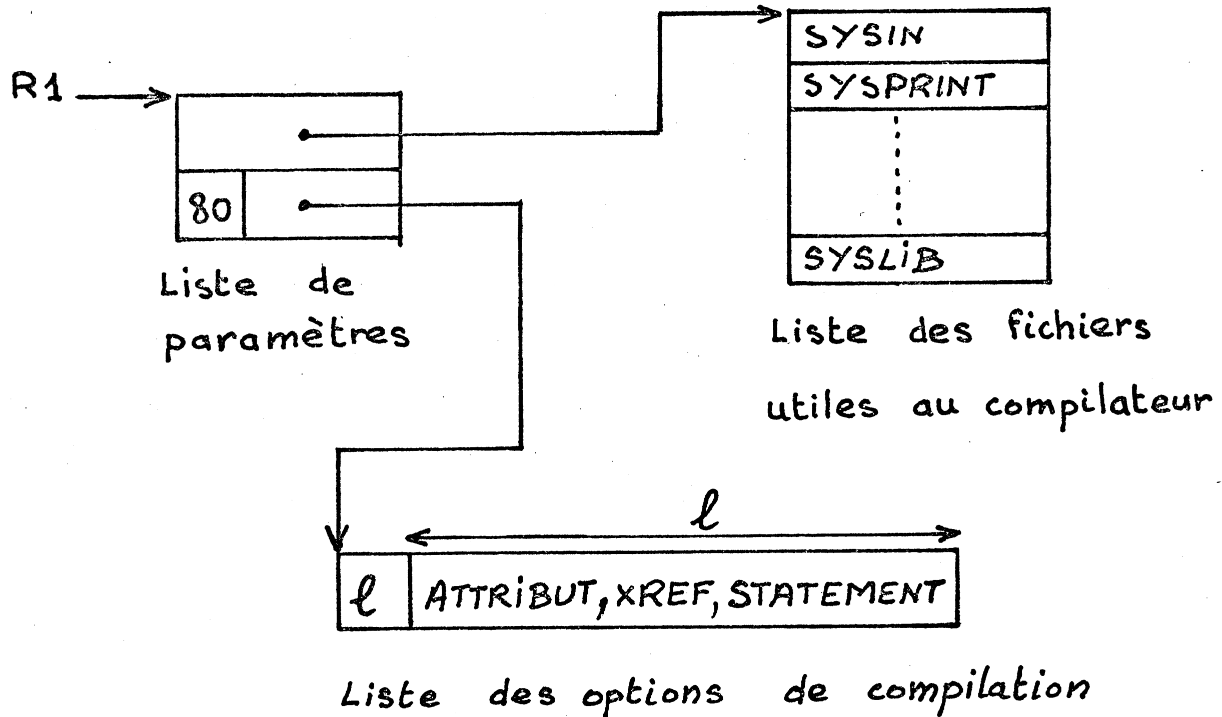
LOAD (SVC 8) charge une phase et retourne dans un registre l'adresse d'implantation de la phase chargée.

DELETE (SVC 9) permet de détruire une phase précédemment chargée par LOAD.

Les fonctions LINK, XCTL, LOAD acquièrent dynamiquement la zone de mémoire qui est nécessaire pour contenir la phase; les fonctions RETURN et DELETE libèrent la zone de mémoire qui contient la phase à détruire.

Ce procédé permet d'assurer que les zones de travail acquises et libérées dynamiquement par les différentes phases, n'interfèrent jamais avec les zones contenant les phases.

Le compilateur reçoit dans le registre 1 l'adresse de la liste des paramètres de compilation. Cette liste, construite par le superviseur du système OS/360, a le format suivant :



Pour gérer les fichiers, le compilateur utilise différentes méthodes d'accès définies par le système.

Par exemple :

- Le traitement des fichiers SYSIN (entrée) et SYSPRINT (sortie) est effectué à l'aide de la méthode QSAM (Queued Sequential Access Method) [I8]. A cette méthode d'accès séquentiel correspondent les macro-instructions GET (lecture) et PUT (écriture)[I9].

- Le traitement du fichier utilitaire SYSUT1 est réalisé à l'aide des méthodes BSAM (Basic Sequential Access Method) et XDAP (eXecute Direct Access Program) [I8]. A la méthode d'accès BSAM correspondent les macro-instructions READ, WRITE, NOTE, POINT, CHECK; à la méthode XDAP, la macro-instruction XDAP [I9].

III - IMPLANTATION DU COMPILATEUR-OS DANS LE SYSTEME CMS -

Pour permettre cette opération, le système CMS doit respecter les règles d'appel du compilateur ainsi que les conventions de liaison; il doit, de plus, lui fournir des fonctions-système identiques à celles offertes par le système OS.

III-1 Liste de paramètres en CMS.

La commande CMS qui indique qu'une compilation de programme PL/1 est demandée a la forme suivante :

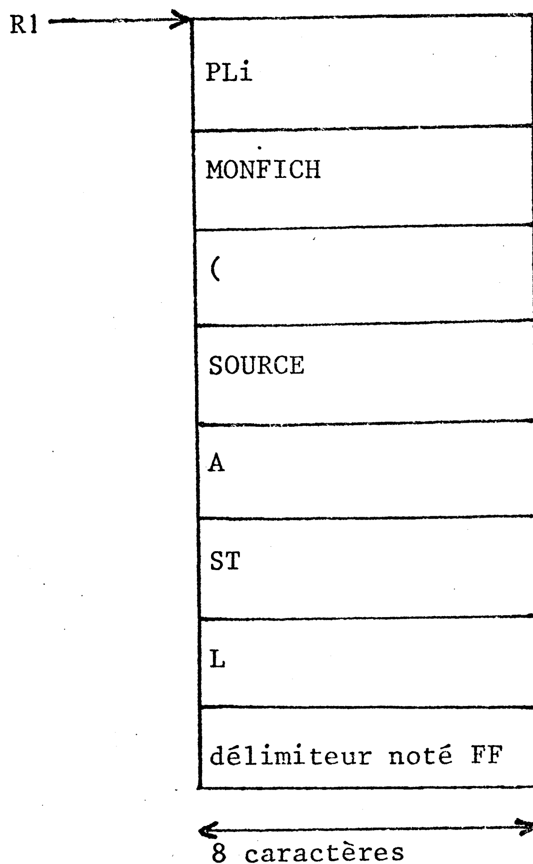
PLi fichier-1 ... fichier-n (liste d'options séparées par un ou plusieurs blancs)

où fichier-1...fichier-n sont les noms des fichiers à compiler qui ont tous le type PLI. A la réception d'une telle commande, le système CMS découpe la commande, en zones de 8 caractères. Ces zones constituent la liste de paramètres qui est transmise au programme chargé de satisfaire la commande.

Notons que les mots-clés de longueur supérieure à 8 caractères sont tronqués; par contre ceux qui ont une longueur inférieure à 8, sont étendus sur la droite par des blancs.
Ainsi, pour la commande

PLi MONFICH (SOURCE A ST L

le système CMS charge en mémoire le module PLI (programme de liaison entre le système et le compilateur) et lui donne le contrôle avec dans le registre l'adresse de la liste de paramètres ainsi construite :



On constate qu'un des rôles du programme de liaison sera de construire la liste de paramètres attendue par le compilateur.

III-2 Fonctions-système nécessaires au compilateur.

Dans ce qui suit, nous appellerons fonctions-système les demandes présentées au système, par le compilateur. Une fonction-système est donc la simulation, par le système CMS, de la fonction correspondante du système OS.

Nous appellerons fonction-cms une possibilité supplémentaire offert par le système CMS. Dans ce cas, la demande est présentée soit à l'aide d'une commande-cms, soit à l'aide d'un appel-superviseur (SVC) de numéro 202.

Ainsi, par exemple, LOAD peut être :

- La fonction-système qui charge dynamiquement un programme. La demande est alors effectuée au moyen d'un appel-superviseur de numéro 8.

- La fonction-cms qui effectue le chargement d'un programme en début de la zone utilisateur. La demande est dans ce cas, présentée grâce à la commande-cms LOAD.

III-2-1 <u>Acquisition et libération dynamique de mémoire</u>.

Nous avons déjà vu que CMS possède ce mécanisme; précisons qu'il fonctionne de manière analogue au mécanisme correspondant du système OS.

III-2-2 <u>Chargeur</u>

Le chargeur de CMS simule correctement, dans les cas simples d'utilisation, les fonctions LINK, XCTL, RETURN, LOAD et DELETE. Deux possibilités sont offertes.

a- Le chargeur travaille sur des fichiers de type TEXT, résultats d'assemblage ou de compilation. Dans ce cas, le chargeur acquiert (ou libère) dynamiquement la zone de mémoire nécessaire pour contenir le programme, installe ce programme en mémoire et réalise les liaisons (résolution des adresses externes) avec les programmes déjà présents. Cet algorithme définit le mode: <u>chargement dynamique translatable</u>.

Ce procédé est extrêmement long. En effet, il y a, à la fois, chargement avec translation et construction des liens; ce qui, en OS, correspondrait à ne pas faire d'édition de liens avant le chargement.

Cette méthode n'est donc pas raisonnablement envisageable pendant une compilation.

b- Le chargeur travaille sur des fichiers de type MODULE. Ces fichiers contiennent l'image de la partie de la mémoire dans laquelle ont été précédemment chargés (et liés) un ou plusieurs programmes. Dans ce cas, le chargeur utilise le mode chargement dynamique absolu. Il n'y a pas acquisition dynamique de mémoire et, aucune translation ni aucune résolution d'adresses n'est à effectuer.

C'est la méthode que nous avons adoptée.

III-2-3 Méthodes d'accès

CMS simule les méthodes d'accès séquentiel BSAM et QSAM . Au cours des opérations d'entrée-sortie sur les fichiers, (lectures ou écritures), CMS incrémente automatiquement les numéros des enregistrements-cms avant que ceux-ci soient traités par l'unique méthode d'accès du système (accès direct par numéro d'enregistrement). [I1].

La méthode d'accès XDAP [I9] n'est pas simulée par CMS. Il faudra donc installer en mémoire, en même temps que le compilateur, un programme de simulation pour cette méthode d'accès.

III-3 L'interface entre le superviseur et le compilateur.

C'est ce que nous avons appelé précédemment programme de liaison.

Les différences entre les systèmes OS et CMS nous conduisent à insérer un tel programme entre le système CMS et le compilateur PL/1. Ainsi, le compilateur ne sera pas appelé directement par le superviseur du système CMS mais par l'interface chargé de simuler les fonctions qui, par rapport à OS, sont mal ou pas définies dans CMS.

III-3-1 Liste de paramètres.

L'interface doit transformer la liste de paramètres fournie par CMS pour construire la liste attendue par le compilateur. De plus, l'interface vérifie les options de compilation et isole celles qui sont destinées au seul système CMS.

III-3-2 Appel du compilateur.

L'interface charge en mémoire, à l'aide de la fonction-cms LOADMOD, le module IEMAA qui est la racine du compilateur. Il lui donne ensuite le contrôle en respectant les conventions suivantes :

- le registre 1 repère la liste des paramètres,
- le registre 13 repère la zone de sauvegarde des registres,
- le registre 14 contient l'adresse de retour dans l'interface,
- le registre 15 contient l'adresse du point d'entrée du module IEMAA.

III-3-3 Initialisation de la mémoire libre.

Nous savons que pendant toute la compilation, le chargeur de CMS fonctionne suivant le mode chargement dynamique absolu. Les phases sont ainsi amenées en mémoire à l'adresse définie lors de la génération du compilateur (Cf §IV). Toutefois, chaque phase a besoin de zones de travail au cours de son exécution; il y a donc lieu de partager logiquement la zone utilisateur en deux parties :

- une partie, zone des instructions, qui sera réservée à l'implantation des phases;
- une partie, zones de travail, qui est gérée dynamiquement au cours de l'exécution du compilateur.

Pour qu'il n'y ait pas interférence entre ces deux zones, l'interface protège la partie contenant les phases: pour cela, il réserve avant l'appel du compilateur, une zone de mémoire de longueur égale à celle de la plus longue branche de l'arbre de la structure de recouvrement. Cette zone commence obligatoirement au début de la zone utilisateur et la longueur de cette branche est déterminée manuellement lors de la génération du compilateur.

III-3-4 Intéraction entre le compilateur et l'utilisateur.

Il n'est pas question de simuler ici, un compilateur conversationnel ou incrémental. Le compilateur utilisé étant de type "batch", le dialogue se fera uniquement dans le sens compilateur vers utilisateur. Il est réalisé par la sortie immédiate des diagnostics sur le terminal d'intéraction; les diagnostics sont aussi écrits dans le fichier de type LISTING qui recueille la liste du programme source.

III-3-5 Traitement des options propres à CMS.

Ces options ne sont pas transmises au compilateur mais sont traitées par l'interface. Elles concernent :

- l'indication d'impression des diagnostics sur la console,
- l'impression du fichier LISTING sur l'imprimante ou la création de ce fichier sur le disque privé de l'utilisateur. Dans ce dernier cas, en plus du fichier de type TEXT contenant le code-machine généré, l'utilisateur obtient aussi sur disque, un fichier de type LISTING.

III-3-6 Itération pour plusieurs compilations successives.

Nous avons vu que le format de la commande PLI permet d'indiquer que plusieurs fichiers sont à compiler. Ainsi, après chaque compilation, l'interface effectue la mise à jour de la partie "liste des fichiers" de la liste de paramètres, puis appelle à nouveau le compilateur. Cet appel est précédé du chargement du module IEMAA car la copie qui est en mémoire a été modifiée et n'est pas réutilisable.

III-3-7 Simulation des méthodes d'accès pour le fichier utilitaire SYSUT1.

Ce fichier utilitaire est créé quand le compilateur n'a pas suffisamment d'espace de travail en mémoire.

L'espace de travail "SYSUT1" réside donc en mémoire puis partiellement sur disque en cas de saturation de la mémoire. Cet espace, découpé en blocs de taille fixe est accessible à l'aide d'une table d'adressage contenue dans le module IEMAL (ou IEMAN) du compilateur (figure III.3.7.a). La taille du bloc est le minimum entre la valeur d'une fonction de la longueur de l'espace mémoire disponible et la capacité d'une piste du disque qui contiendra éventuellement le fichier utilitaire. La fonction de la taille de la mémoire libre est définie par une table de constantes contenue dans le compilateur; la capacité de la piste est obtenue à l'aide de la fonction-système DEYTYPE (SVC 24).

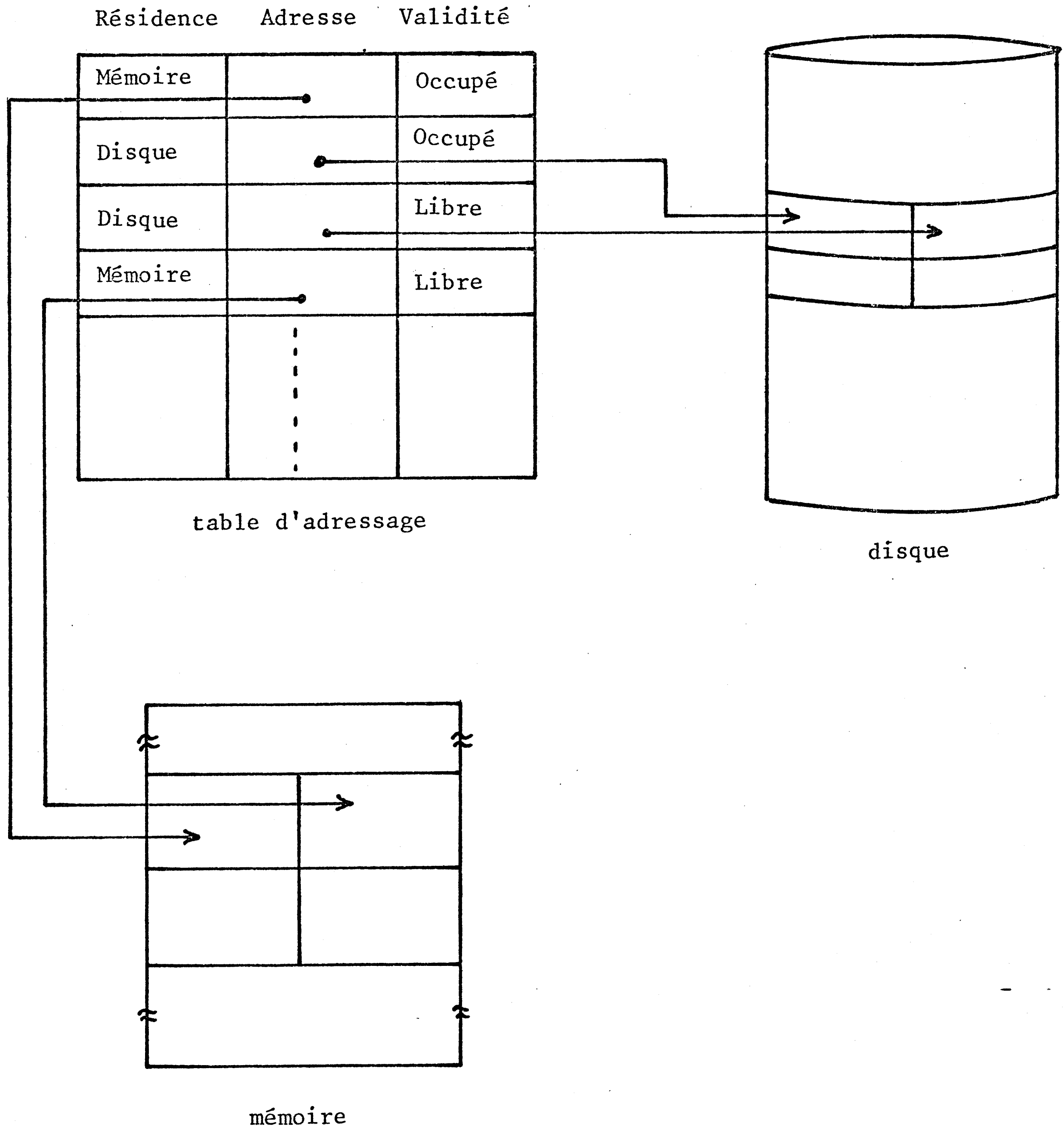


FIGURE III.3.7.a - Structure de l'espace de travail SYSUT1 -

Le fichier SYSUT1 est initialisé lorsque le compilateur s'aperçoit que la mémoire est saturée. Ce fichier est contrôlé par un bloc de contrôle DCB (Data Control Block) [I10] dont le champ le plus important pour nous est celui nommé DCBFDAD. Cette zone contient après chaque entrée-sortie, l'adresse sous forme MBBCCHHR de l'enregistrement traité.

Précisons qu'il existe dans le système OS/360, trois manières d'addresser un enregistrement de fichier résidant sur disque [I9] :

- adresse absolue, notée
  MBBCCHHR
  avec
  MBB non significatif pour une unité de disque,
  CC numéro de cylindre
  HH numéro de piste
  R numéro d'enregistrement sur la piste.

- adresse de piste, relative au début du fichier, notée
  TTR
  avec
  TT numéro relatif de piste
  R numéro d'enregistrement sur la piste.

- adresse d'enregistrement relative au début du fichier, notée
  AAA
  avec
  AAA : numéro d'enregistrement dans le fichier.

Les différentes opérations effectuées, par le compilateur, sur le fichier SYSUT1 se résument ainsi :

1. Création ou addition d'un enregistrement en fin de fichier à l'aide de la méthode séquentielle BSAM. La séquence de traitement est :

WRITE (écriture de l'enregistrement et mise à jour de DCBFDAD)

CHECK (attente de la fin de l'entrée sortie et traitement des erreurs par le compilateur).

NOTE (connaissance de l'adresse de l'enregistrement sous la forme TTR).

2. Mise à jour du fichier effectuée par accès direct, en utilisant la méthode XDAP. La séquence de traitement est :

Sauvegarde de DCBFDAD

POINT (positionnement dans le fichier et modification de DCBFDAD à l'aide du paramètre TTRZ)

XDAP (lecture ou écriture à partir de l'adresse MBBCCHHR contenue dans DCBFDAD)

CHECK (attente de la fin de l'entrée-sortie et traitement des erreurs par le compilateur)

Restauration de DCBFDAD

Précisons que :

- La fonction-système POINT (positionnement dans le fichier) admet comme paramètre une adresse de la forme TTRZ où Z vaut soit 0 si l'on désire accéder à l'enregistrement d'adresse TTR, soit 1 si l'on désire accéder à l'enregistrement suivant.

- La fonction-système NOTE fournit l'adresse de l'enregistrement traité, sous la forme TTRO. Cette adresse est la transformée de l'adresse (MBBCCHHR) contenue dans la zone DCBFDAD.

- L'appel des fonctions-système OPEN, XDAP s'effectue à l'aide des instructions SVC 19 et SVC 0; les adresses des fonctions-système POINT, NOTE, CHECK et WRITE sont placées dans le DCB, lors de l'initialisation du fichier.

- Nous ne parlerons pas de la fonction READ (lecture d'un enregistrement) car elle n'est pas utilisée par le compilateur.

Simuler les méthodes d'accès du fichier SYSUT1 revient donc à transformer des adresses MBBCCHHR ou TTR en numéro d'enregistrement-cms et à lire ou écrire l'enregistrement-cms. Ainsi, l'interface doit donc effectuer la simulation, pour le traitement de SYSUT1, des fonctions-système :

DEVTYPE		(SVC 24)
OPEN		(SVC 19)
XDAP		(SVC 0)
WRITE	(BSAM)	
POINT		
NOTE		
CHECK		

Pour des raisons de compatibilité, toutes ces fonctions-système sont installées dans l'interface, bien que certaines d'entre elles existent dans le système CMS.

Avant de présenter la simulation de ces fonctions, examinons tout d'abord le traitement des interruptions de type SVC et la réalisation des entrées-sorties sur disque par le système CMS.

Traitement des interruptions de type SVC par CMS [I1]

Le mécanisme de traitement des interruptions de type SVC prévoit la possibilité de donner le contrôle, suivant le code de l'interruption, soit à un sous-programme de l'utilisateur, soit à un sous-programme du système CMS.

Pour associer à chaque numéro de SVC, l'adresse d'un sous-programme de traitement, CMS dispose de deux tables :

- une table des fonctions-SVC définies par l'utilisateur,
- une table des fonctions-SVC définies par le système CMS (figure III.3.7.b)

Par construction, la table des fonctions-SVC définies par l'utilisateur est explorée la première; ce qui permet à l'utilisateur de simuler localement dans son programme, des fonctions-SVC définies par ailleurs dans le système. La table de l'utilisateur est créée ou mise à jour par la fonction-cms HNDSVC qui admet en paramètres :

- Un opérateur, tel que SET pour addition, CLEAR pour soustraction,
- Une suite de définitions qui, chacune, associent à un numéro de SVC, l'adresse du sous-programme de traitement relatif à ce numéro.

La figure III.3.7.b montre l'état des tables des fonctions-SVC lorsque l'utilisateur a indiqué que la fonction SVC 00 devait être traitée par le sous-programme nommé XDAP, la fonction SVC 19 par le sous-programme MYOPEN et la fonction SVC 24 par le sous-programme MYDEVTYP.

table des fonctions-SVC définies par l'utilisateur

nb d'entrées: 3	
00	XDAP
19	MYOPEN
24	MYDEVTYP
numéro de SVC	adresse du sous-programme de traitement

table des fonctions-SVC définies par le système

nb d'entrées: n	
19	OPENCMS
24	DVTYPCMS
numéro de SVC	adresse du sous-programme de traitement

FIGURE III.3.7.b. -Les tables des fonctions SVC-

Les entrées-sorties sur disque du système CMS [I2, I3, L3]

Trois fonctions essentielles réalisent les opérations d'entrée-sortie sur disque :

- la fonction-cms RDBUF qui lit un ou plusieurs enregistrements-cms,
- la fonction-cms WRBUF qui écrit un ou plusieurs enregistrements-cms,
- la fonction-cms FINIS qui indique qu'un fichier n'est plus utilisé (fermeture).

Ces fonctions, comme la plupart des fonctions-cms, sont initialisée à l'aide d'une instruction SVC de numéro 202 et reçoivent une liste de paramètres construite avant l'appel.

La liste de paramètres transmise soit à la fonction RDBUF, soit à la fonction WRBUF comporte entre autres :

- l'identification du fichier
- le numéro du premier enregistrement-cms à traiter,
- l'adresse de la zone tampon à utiliser pour l'entrée-sortie,
- la longueur de cette zone et la longueur d'un enregistrement-cms, c'est-à-dire le nombre d'enregistrements à traiter.

La liste de paramètres transmise à la fonction FINIS ne comporte que l'identification du fichier.

Il n'existe pas de fonction-cms spécialisée pour initialiser un fichier (ouverture). Le fichier est ouvert lors de la première entrée-sortie soit en mode lecture par la fonction RDBUF, soit en mode écriture par la fonction WRBUF. Il est alors nécessaire de former un fichier actif en lecture avant de l'écrire et inversement.

Etudions maintenant les fonctions-système mises en jeu lors du traitement du fichier utilitaire SYSUT1 .

Simulation de DEVTYPE (SVC 24)

Cette fonction indique que la longueur d'une piste est identique à la taille du bloc imposé par l'espace mémoire disponible. Il n'y aura donc qu'un enregistrement par piste, c'est-à-dire que le numéro d'enregistrement R est constant et égal à 1 dans les adresses de la forme MBBCCHHR et TTR.

Simulation de OPEN (SVC 19) (Figure III.3.7.c.)

Cette fonction effectue l'initialisation du bloc de contrôle DCB dans lequel sont placées les adresses des sous-programmes de l'interface chargés de simuler les fonctions WRITE(BSAM), POINT, NOTE et CHECK. Elle construit ensuite la liste de paramètres des fonctions-cms RDBUF et WRBUF et note l'adresse de cette liste dans une zone du DCB. Enfin elle met à zéro la zone DCBFDAD du DCB.

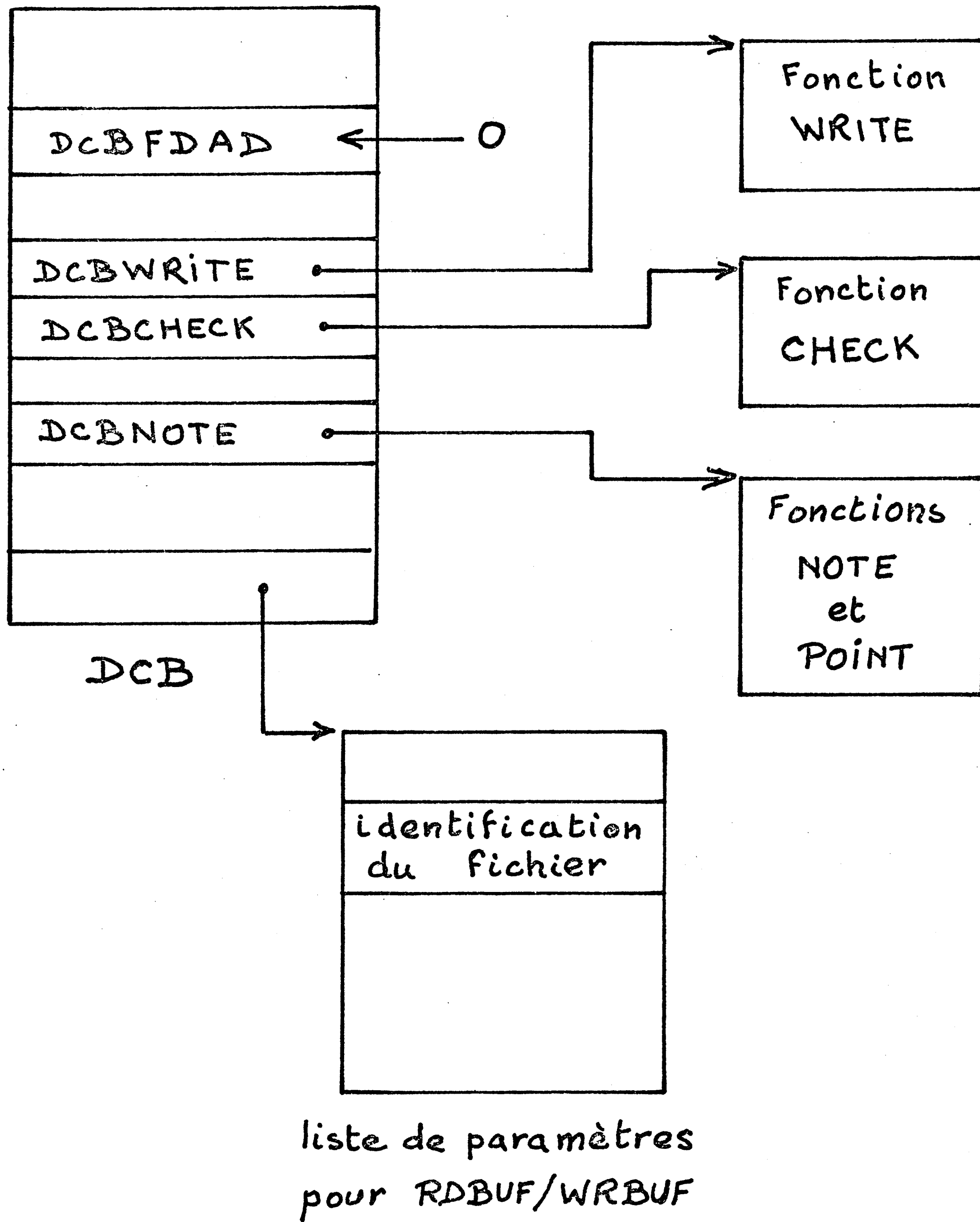


Figure III.3.7.c. Simulation de OPEN

Simulation de WRITE (BSAM) (Figure III.3.7.d.)

Cette fonction a pour objet de :

- Incrémenter de 1 l'adresse MBBCCHHR contenue dans la zone DCBFDAD du DCB.
- Transformer ensuite cette nouvelle adresse en un numéro d'enregistrement-cms.
- Placer dans la liste de paramètres de la fonction WRBUF, ce numéro d'enregistrement-cms ainsi que l'adresse de la zone tampon associée.
- Effectuer l'opération d'écriture en appenant la fonction WRBUF et noter dans le bloc DECB (Data Event Control Block) qui décrit l'entrée-sortie, les conditions de fin d'entrée-sortie [I8,I10].

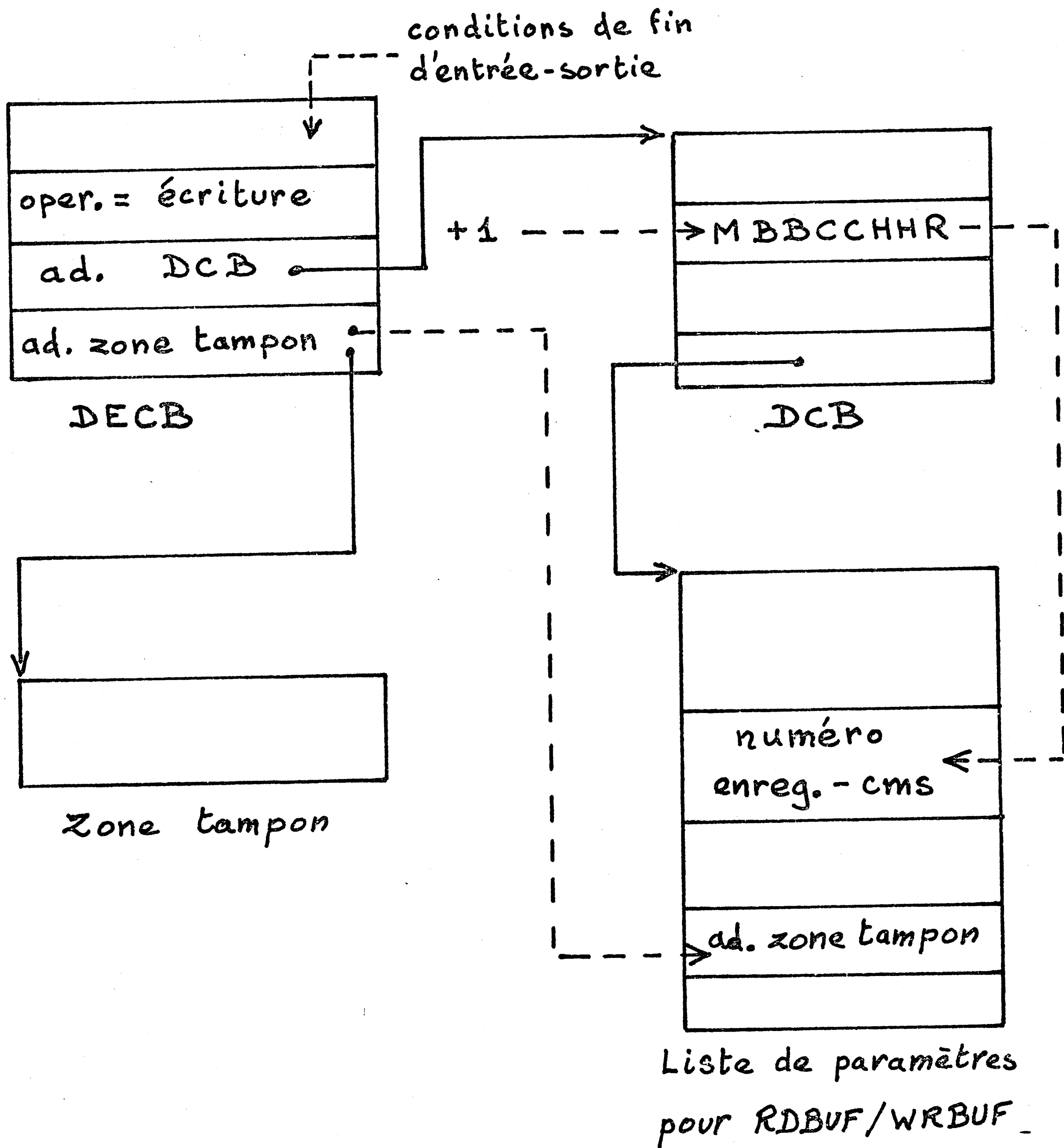


Figure III.3.7.d. Simulation de WRITE (BSAM)

Simulation de CHECK (Figure III.3.7.e.)

Cette fonction vérifie comment s'est effectuée l'entrée-sortie. Si une erreur est détectée, la fonction donne le contrôle au sous-programme du compilateur, chargé de traiter ces erreurs. L'adresse de ce sous-programme est contenue dans la zone DCBSYNAD du DCB.

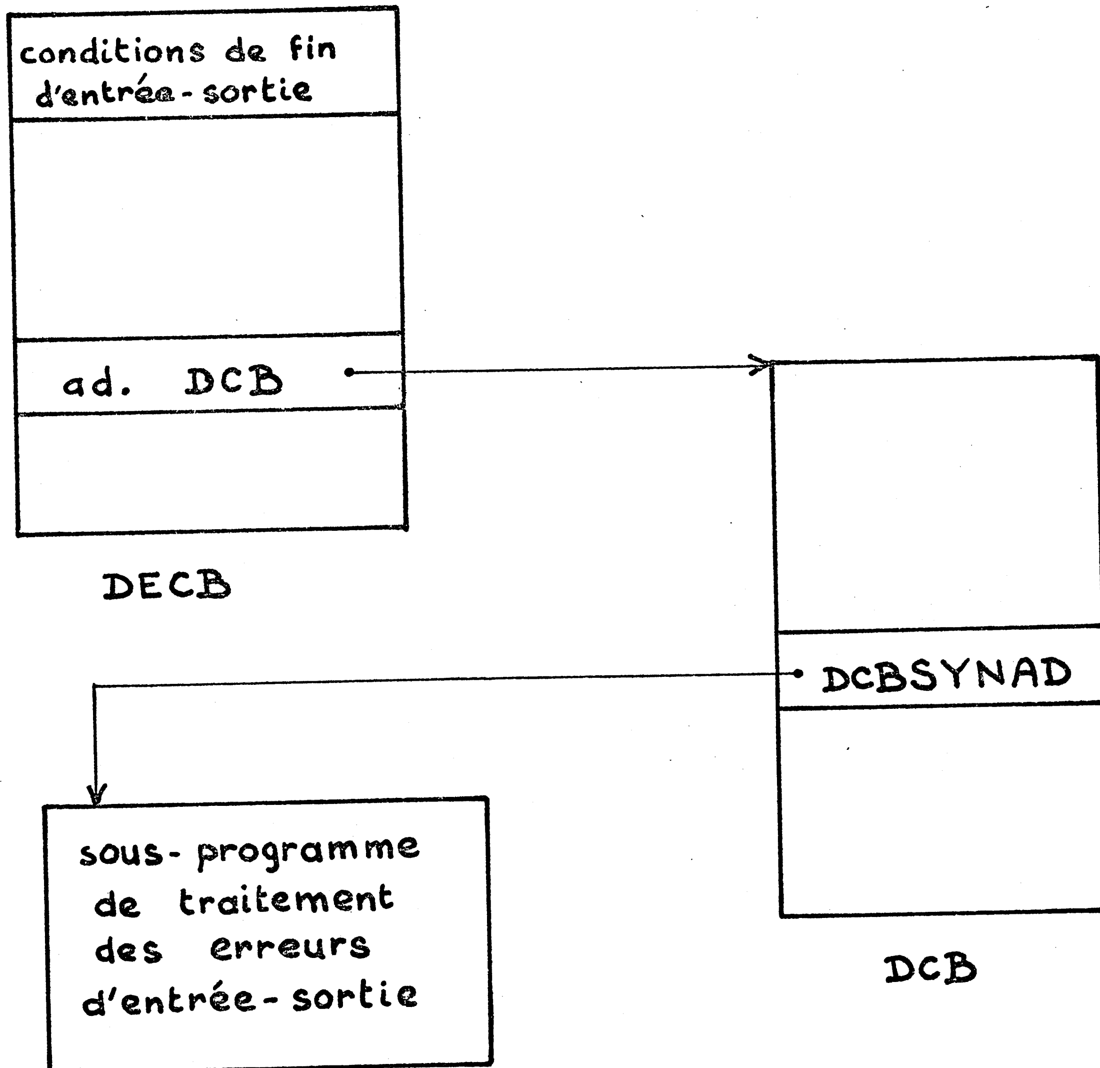


Figure III.3.7.e. Simulation de CHECK -

Simulation de NOTE (figure III.3.7.f.)

Cette fonction calcule l'adresse du dernier enregistrement traité sous la forme TTR, à partir du numéro d'enregistrement-cms pris dans la liste de paramètres de WRBUF/RDBUF; c'est-à-dire que :

TT est égal au numéro -1 de l'enregistrement-cms

et

R est égal à 1.

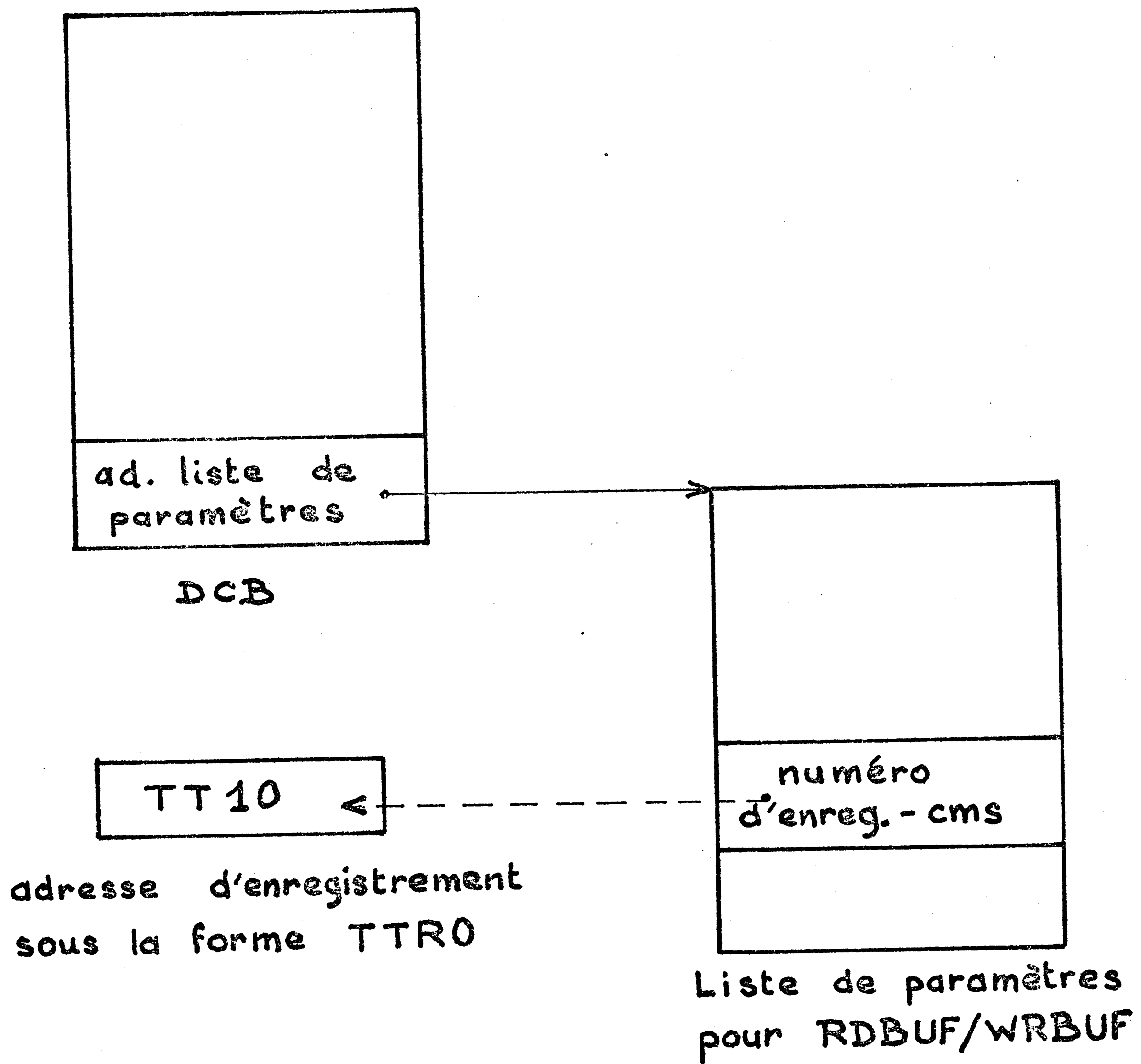


FIGURE III.3.7.f. Simulation de NOTE.

Simulation de POINT (figure III.3.7.g).

Cette fonction calcule un numéro d'enregistrement-cms à partir de l'adresse TT1z fournie. Ce numéro d'enregistrement-cms, égal à TT+1+Z, est placé dans la liste de paramètres pour WRBUF, RDBUF. De plus l'adresse absolue d'enregistrement est évaluée et placée dans la zone DCBFDAD du DCB.

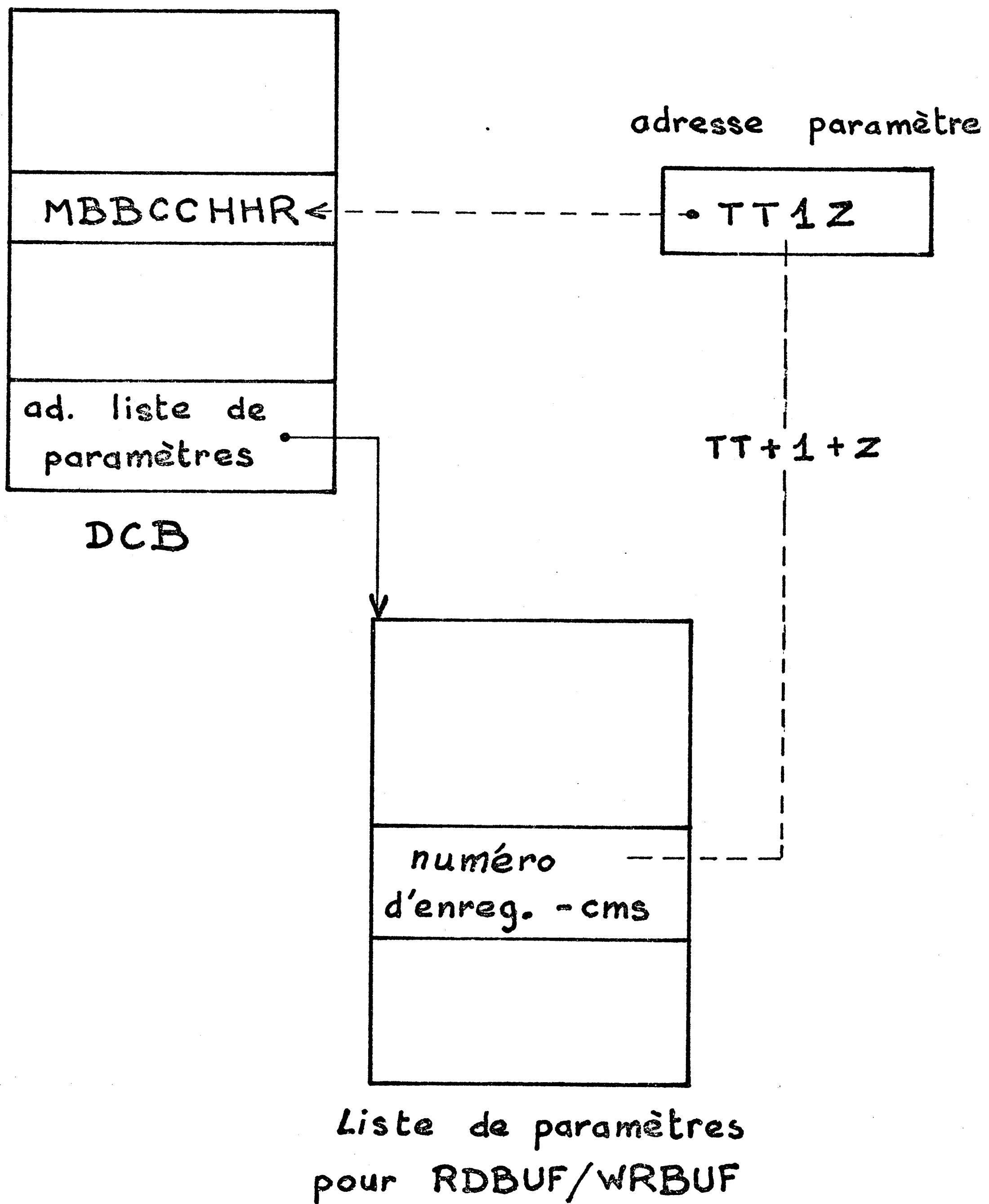


FIGURE II.3.7.g. Simulation de POINT

Simulation de XDAP (SVC 0) (Figure III.3.7.b.)

Cette fonction reçoit en paramètre l'adresse d'un bloc de contrôle "XDAP Control Block" qui décrit l'opération d'entrée-sortie. Ce bloc est formé par la concaténation de trois blocs de contrôle :

- un ECB (Event Control Block) qui contiendra les conditions de fin d'entrée-sortie,
- un IOB (Input-Output Block) qui contient en outre les adresses du DCB et de l'enregistrement à traiter,
- un bloc qui contient un programme canal formé de trois instructions-canal (SEARCH, TIC, READ ou WRITE). L'adresse de la zone tampon associée à l'entrée-sortie se trouve dans la dernière instruction-canal.

La fonction XDAP a pour objet de :

- Placer dans la liste de paramètres pour WRBUF/RDBUF, l'adresse de la zone tampon.
- Fermer éventuellement le fichier-cms à l'aide de la fonction FINIS, si le mode d'activité du fichier est incompatible avec l'entrée-sortie demandée.
- Effectuer l'entrée-sortie en appelant soit la fonction WRBUF, soit la fonction RDBUF. Le choix est déterminé à l'aide du code opération (READ ou WRITE) de la dernière instruction-canal.

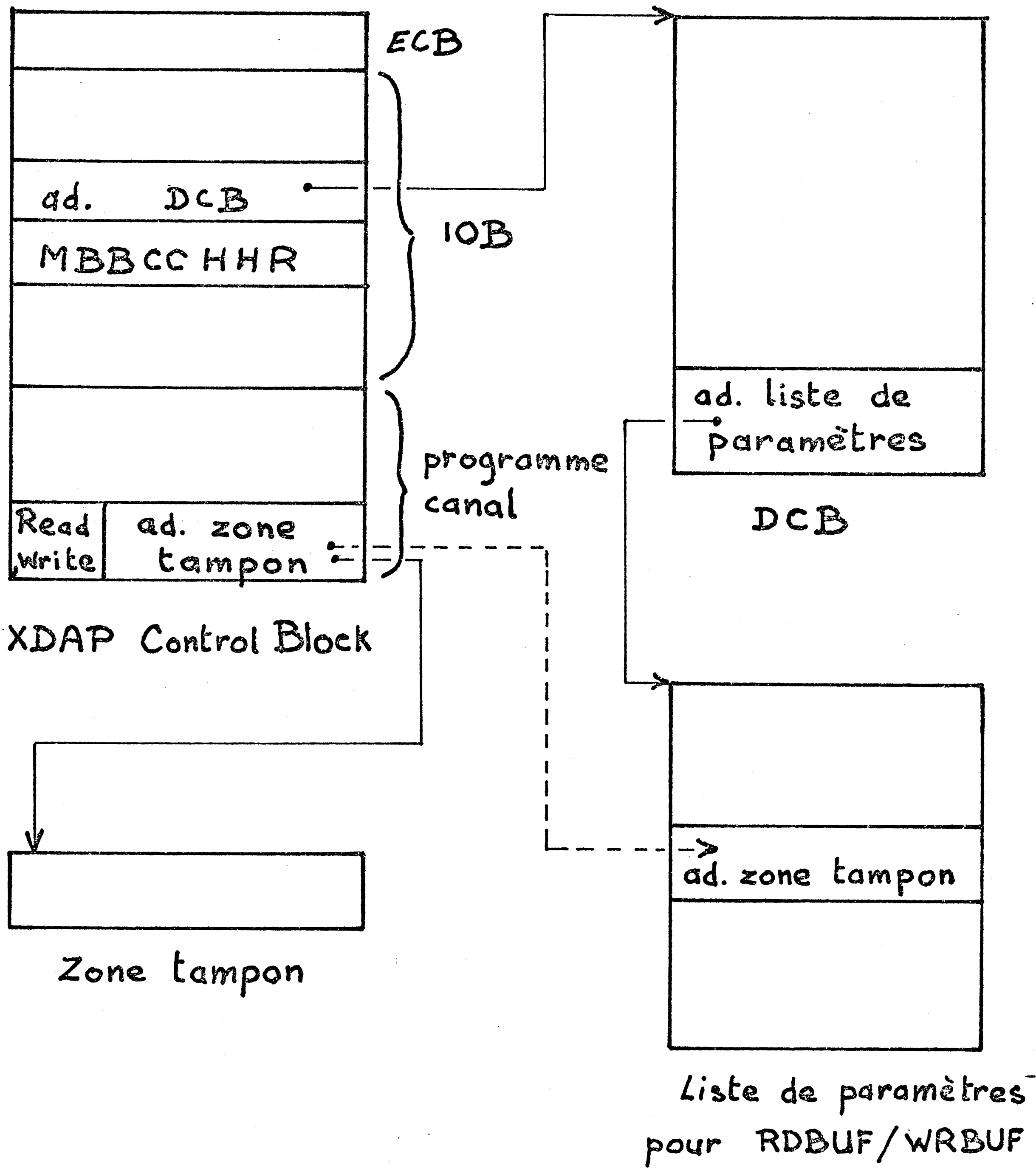


FIGURE III.3.7.b. Simulation de XDAP

IV - GENERATION DU COMPILATEUR

Nous appelons génération l'opération qui consiste à créer un fichier de type MODULE pour chacun des 253 programmes qui constituent le compilateur PL/1.

Cette opération nous permet d'optimiser les temps de compilation: en effet, durant toute compilation, l'enchainement des phases est assuré par le chargeur qui fonctionne en mode absolu et non en mode translatable et qui traite alors des fichiers de type MODULE au lieu de fichiers de type TEXT.

Pour autoriser ce mode de fonctionnement, il faut donc préparer au préalable les images-mémoire des différentes phases du compilateur; les relations entre ces phases sont matérialisées, du point de vue de l'implantation en mémoire, par l'arbre qui décrit la structure de recouvrement.

Vu le nombre important des programmes qui constituent le compilateur, la génération de celui-ci est effectuée à l'aide d'une procédure EXEC (cf.§I). Nous ne fournirons pas une liste du contenu de cette procédure mais, nous nous attacherons à décrire le principe qui est appliqué pour toute génération.

Examinons tout d'abord quelles sont les opérations nécessaires pour créer le fichier de type MODULE qui contient l'image-mémoire d'une phase :

- le programme qui correspond à la phase est installé en mémoire; le chargeur qui fonctionne alors en mode translatable, résoud les adresses.

- L'image de la partie de la mémoire qui a été utilisée pour l'implantation est alors copiée dans un fichier de type MODULE.

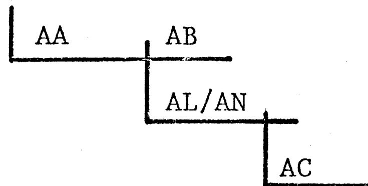
Détaillons ensuite les fonctions-cms qui participent à la génération :

- LOAD et REUSE qui permettent le chargement d'une phase à partir du fichier TEXT qui a été produit par l'assemblage du programme. Le chargeur fonctionne dans ce cas en mode translatable, les adresses externes sont résolues et les liens entre programmes présents en mémoire sont établis. Tout chargement commandé par LOAD s'effectue normalement en début de la zone utilisateur; tout chargement commandé par REUSE s'effectue derrière la phase précédemment installée en mémoire soit par LOAD, soit par REUSE, soit par LOADMOD.

- GENMOD qui copie dans un fichier de type MODULE, l'image de la partie de la mémoire dans laquelle nous avons installé une phase à l'aide de LOAD ou de REUSE. Cette fonction-cms permet de créer un module-cms qui, contrairement aux modules-OS créés par l'éditeur de liens du système OS/360, est non translatable.

- LOADMOD qui effectue le chargement en mémoire à partir d'un fichier de type MODULE. Le chargeur fonctionne alors en mode absolu, ce qui permet un chargement plus rapide que celui effectué par LOAD ou REUSE. L'adresse d'implantation est fixe et a été définie lors de la création du module.

Pour illustrer le mécanisme de génération, considérons la structure de recouvrement qui décrit les liens entre les cinq programmes AA, AB, AL, AN et AC.



Par convention, les phases dont les noms se trouvent immédiatement à droite d'une même horizontale ne peuvent être présentes en même temps en mémoire et sont donc générées à la même adresse; les phases dont les noms figurent sur une même horizontale se trouvent en même temps en mémoire. La notation AL/AN signale que ces deux phases sont logiquement identiques; le compilateur utilise l'une ou l'autre de ces phases.

En supposant que la longueur du programme AL soit supérieure à celle du programme AN, la séquence de génération de la structure de recouvrement est :

```
LOAD      AA
GENMOD    AA
REUSE     AB
GENMOD    AB
LOADMOD   AA
REUSE     AN
GENMOD    AN
LOADMOD   AA
REUSE     AL
GENMOD    AL
REUSE     AC
GENMOD    AC
```

Cette suite de commandes-cms peut être conservée dans un fichier de type EXEC qui constitue alors la procédure de génération de cette

structure de recouvrement.

Les différents états de la mémoire au cours de cette génération sont schématisés par la figure IV-a :

. L'état 1 est obtenu après les commandes :

LOAD AA	chargement en début de la zone utilisateur, à partir du fichier AA TEXT.
GENMOD AA	création du fichier AA MODULE et définition de l'adresse d'implantation de la phase AA (adresse E1).

. L'état 2 est obtenu après :

REUSE AB	chargement, après la phase AA, à partir du fichier AB TEXT
GENMOD AB	création du fichier AB MODULE et définition de l'adresse d'implantation de la phase AB (adresse E2).

. L'état 3 est obtenu après :

LOADMOD AA	chargement à l'adresse E1 de la phase AA, à partir du fichier AA MODULE.
REUSE AN	chargement, après la phase AA, à partir du fichier AN TEXT
GENMOD AN	création du fichier AN MODULE et définition de l'adresse d'implantation de la phase AN (adresse E2).

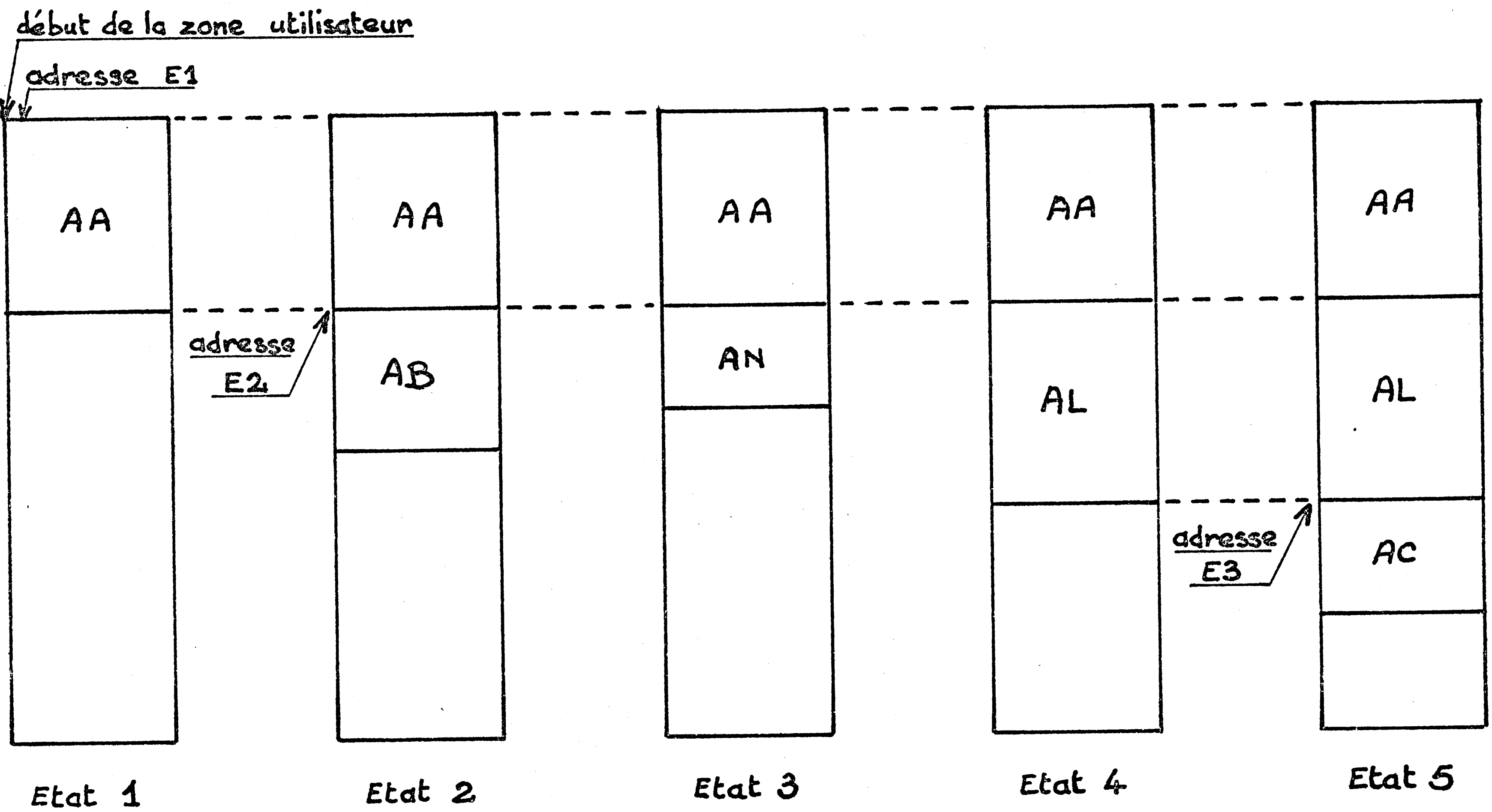


FIGURE IV-a -Etats de la mémoire au cours de la génération-

. L'état 4 est obtenu après :

LOADMOD AA	chargement à l'adresse E1 de la phase AA, à partir du fichier AA MODULE
REUSE AL	chargement, après la phase AA, à partir du fichier AL TEXT
GENMOD AL	création du fichier AL MODULE et définitio, de l'adresse d'implantation de la phase AL (adresse E2).

. L'état 5 est obtenu après :

REUSE AC	chargement, après la phase AL, à partir du fichier AC TEXT
GENMOD AC	création du fichier AC MODULE et définition de l'adresse d'implantation de la phase AC (adresse E3).

V - CONTROLE DE LA VALIDITE DE LA STRUCTURE DE RECOUVREMENT

Il est peu pensable de contrôler "manuellement" la validité de cette structure, en dessinant le graphe des appels des différentes phases. Il semble plus intéressant de compiler un programme de test, suffisamment complexe pour que toutes les phases du compilateur soient utilisées.

Si la compilation de ce programme se déroule normalement, on peut dire que la structure de recouvrement est correcte.

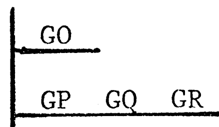
Si cette compilation est interrompue pour une cause d'erreur du compilateur, la structure est certainement incorrecte.

Une première méthode de recherche consisterait à étudier une image mémoire indiquant les phases chargées et à vérifier à l'aide des listes d'assemblage, la validité des liens entre ces phases.

Une deuxième méthode plus élégante et plus rapide, que nous avons adoptée, consiste à obtenir, grâce à l'interface de compilation, une trace des appels dynamiques des phases.

Cette trace indique le nom et l'adresse de la phase qui émet la demande de chargement ou d'effacement d'une phase, le type de la demande (LINK, LOAD, XCTL, DELETE, RETURN), le nom de la phaseà charger ou à détruire, ainsi que son adresse d'implantation en mémoire. Ainsi, en cas d'erreur, l'étude des liens des dernières phases chargées est immédiate et ne nécessite plus la consultation d'une image de la mémoire.

Cette méthode s'est révélée très efficace; par exemple, une partie de la structure initiale indique le recouvrement :



L'étude du "code source" de ces phases montre que la phase GO contient les demandes :

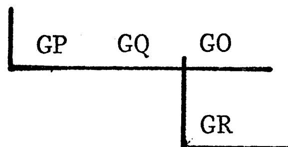
```
LOAD      GP
:
LOAD      GQ
:
GOTO      GP
```

et la phase GP, les demandes :

```
RELEASE   GO
:
LOAD      GR
```

Il y a incompatibilité entre ces demandes et la structure primitivement adoptée. En effet, les phases GO et GP ont été implantées à la même adresse; le chargement de la phase GP, demande émise par la phase GO, détruit donc la phase GO. De même, lorsque la phase GP émet la demande d'effacement de la phase GO, la phase GP est détruite.

La trace dynamique des appels nous montre que la structure correcte est :



VI - EXECUTION D'UN PROGRAMME PL/1

L'exécution d'un "code objet" provenant de la compilation d'un programme PL/1 nécessite la présence d'un ensemble de sous-programmes regroupés dans une bibliothèque.

VI-1 La bibliothèque d'exécution

Elle est constituée par un fichier de type TXTLIB qui simule une organisation de fichiers dite partitionnée [19]. Ce fichier contient le "code objet" des sous-programmes utilisés lors de l'exécution d'un programme PL/1.

Les sous-programmes de la bibliothèque sont soit chargés en mémoire en même temps que le programme à exécuter, soit installés dynamiquement pendant l'exécution du programme. Contrairement à la compilation, le chargeur de CMS utilise le mode chargement dynamique translatable puisque les sous-programmes de la bibliothèque doivent être installés n'importe où en mémoire.

VI-2 Interface d'exécution

L'interface d'exécution doit assurer la transmission des paramètres éventuels et la protection de la zone mémoire qui contient à la fois le programme et les sous-programmes de la bibliothèque utilisés par ce programme.

VI-3 Activation d'un programme PL/1

La compilation de tout programme PL/1, c'est-à-dire de toute procédure possédant l'attribut MAIN, genère en plus des instructions machine correspondant aux instructions PL/1, une section de contrôle appelée IHENTRY et une constante nommée IHEMAIN. De plus, le compilateur définit IHENTRY comme le point d'entrée qui recevra le contrôle lors de l'activation du programme.

La constante IHEMAIN contient l'adresse du prologue du programme; la section de contrôle IHENTRY contient les instructions qui permettent de transférer le contrôle à l'un des points d'entrée IHESAP de la bibliothèque, chargé d'initialiser l'exécution du programme.

Pour que l'interface d'exécution IHECMS prenne automatiquement le contrôle à la place du sous-programme initial, nous avons forcé le compilateur à générer l'adresse externe de IHECMS au lieu de celle de l'un des points d'entrée de IHESAP. L'interface d'exécution transmettra le contrôle soit au point d'entrée IHESAPA, soit au point d'entrée IHESAPB suivant qu'il y a ou non transmission de paramètres à la procédure ayant l'attribut MAIN. Nous obtenons alors les liaisons entre programme et sous-programmes de la bibliothèque qui sont schématisés par la figure VI.3.a.

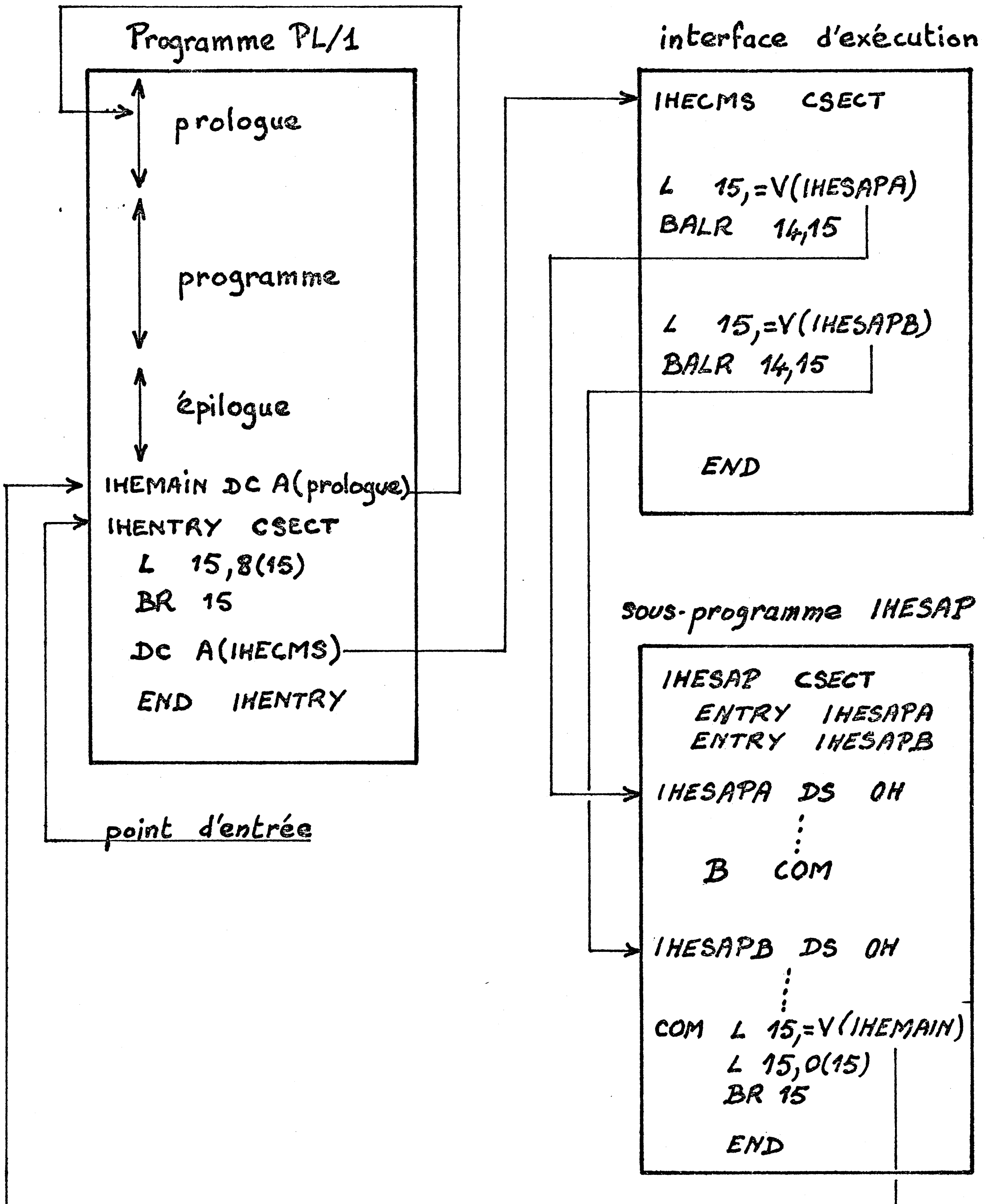


FIGURE VI-3-a -Liaisons entre programme et sous-programmes de la bibliothèque.

VI-4 Problèmes posés par l'exécution d'un programme PL/1.

Les problèmes posés se situent principalement au niveau des entrées-sorties.

Nous avons vu que CMS simule les méthodes d'accès séquentiel BSAM (Basic Sequential Access Method) et QSAM (Queued Sequential Access Method). La simulation des autres méthodes d'accès n'étant pas faite, il est donc illusoire de penser exécuter sous le système CMS, un programme utilisant soit la méthode BDAM (Basic Direct Access Method), soit une méthode d'accès indexé séquentiel: BISAM (Basic Indexed Sequential Access Method) ou QISAM (Queued Indexed Sequential Access Method).

D'autres problèmes sont dûs au système CMS lui-même. En effet, toutes les possibilités du chargement dynamique du système OS, ne sont pas fournies par CMS. Par exemple, lors d'une première demande de chargement d'un module (LOAD EP=A), le système OS installe en mémoire ce module. D'autres demandes de chargement relatives à ce même module ne provoquent aucun nouveau chargement et le système OS note simplement le nombre de ces demandes. Par la suite, le module ne sera effacé en mémoire que lorsque le nombre des demandes de destruction (DELETE EP=A) sera égal au nombre des demandes de chargement, relatives à ce module.

Cette possibilité n'est pas offerte par le système CMS, ce qui provoque quelquefois des erreurs d'exécution qui ne sont pas dûes à l'écriture du programme PL/1. Nos travaux ne nous ont pas permis de faire les modifications nécessaires au système CMS, mais nous notons toutefois que les possibilités du chargeur dynamique doivent être étendues dans les prochaines versions du système.

Un autre problème était le suivant: lors de la simulation de la fonction XCTL, le système CMS utilisait le contenu d'une zone mémoire après que cette zone ait été rendue à la mémoire libre. Sur une machine réelle ou dans une machine virtuelle générée par les anciennes versions de CP, cette

faute de logique n'entrainait pas une erreur de programmation. En effet, le contenu de la zone mémoire n'était pas alors altéré par l'opération de libération. Toutefois, après la mise en service d'une version de CP, contenant le mécanisme de libération de pages, nous nous sommes aperçus que le déroulement de certains programmes PL/1 était devenu dépendant de la charge du système CP.

En effet, dans certains cas, la libération d'une zone de mémoire est équivalent à dire qu'une page est libre et n'est plus utilisée. Si cette libération survient alors que la page de mémoire virtuelle réside sur support externe, CP substitue à l'image de cette page dont le contenu est logiquement devenu non significatif, l'image d'une unique page pleine de zéros [12].

Une modification a été apportée au système CMS pour pallier cette erreur.

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

UN ENVIRONNEMENT DE MISE AU POINT POUR PL/1:

BUGPLI ET SES REQUETES

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

UN ENVIRONNEMENT DE MISE AU POINT POUR PL/1:

B U G P L I   ET SES REQUETES

I LES BUTS DE L'ACCOMPAGNATEUR BUGPLI.

II COMPILATION. CREATION D'UN PROGRAMME A MISE AU POINT INTERACTIVE.

III UTILISATION DE L'ENVIRONNEMENT BUGPLI.

- III-1 Demande d'exécution d'un programme à mise au point intéractive.
- III-2 Activation asynchrone de BUGPLI.
- III-3 Activation de BUGPLI sur erreur d'exécution.

IV LES REQUETES DE L'ENVIRONNEMENT BUGPLI.

- IV-1 Requêtes utilitaires.
- IV-2 Requêtes permettant un accès à l'environnement généré par PL/1.
- IV-3 Requêtes permettant l'accès aux variables PL/1 de types arithmétique ou chaîne.
- IV-4 Requêtes permettant l'accès aux variables PL/1 de type fichier
- IV-5 Requête permettant l'examen du contenu d'un fichier: EXAMINE.

V LES REQUETES DE L'ENVIRONNEMENT EXAMINE.

VI RECAPITULATIF DES REQUETES.

- VI-1 Requêtes de BUGPLI.
- VI-2 Requêtes de EXAMINE.

I - LES BUTS DE L'ACCOMPAGNATEUR BUGPLI.

Le système CMS possède un environnement de mise au point (DEBUG) pour programmes écrits en langage machine. En utilisant comme référence la liste d'assemblage, l'utilisateur place des points d'arrêt sur des instructions. Au cours de l'exécution, la rencontre d'un point d'arrêt, matérialisé par une instruction de code opération invalide, donne automatiquement le contrôle à DEBUG. A cet instant, l'utilisateur peut :

- Examiner et modifier le contenu de la mémoire, des registres et du mot d'état programme.
- Placer de nouveaux points d'arrêt.
- Reprendre l'exécution soit séquentiellement soit sur une instruction quelconque.

Cet outil de mise au point, extrêmement simple, s'est révélé très efficace. Dans la version standard de CMS, l'accès à la mémoire s'effectue en fournissant une adresse hexadécimale car DEBUG ne connait pas les variables définies dans le programme. Toutefois, les outils de mise au point PILOTE [P1] et SPY [L1] autorisent la mise au point de façon symbolique: l'utilisateur fournit les noms des variables de son programme qu'il désire consulter ou modifier.

Notre objectif, avec l'accompagnateur BUGPLI, est de proposer un mécanisme de mise au point pour les programmes PL/1, qui offre des possibilités analogues; c'est-à-dire :

- Accès symbolique aux variables.
- Reprise de l'exécution soit séquentiellement soit à une autre instruction.
- Mise en place de points d'arrêt.

Toutefois il existe une différence fondamentale entre le langage PL/1 et le langage d'assemblage: PL/1 est un langage qui possède une notion de bloc. Cette notion est importante car :

- Dans chaque bloc, l'utilisateur peut définir ou redéfinir de nouvelles variables.
- La reprise de l'exécution ne peut pas se faire à n'importe quelle instruction. En effet, il est toujours possible de quitter un bloc mais il est interdit d'entrer dans un bloc autrement que par le début de ce bloc.

L'accès symbolique aux variables peut être obtenu de manière simple car le langage PL/1 possède deux instructions GET DATA et PUT DATA qui permettent d'associer, lors de l'exécution, les noms et les adresses-mémoire des variables. Pour réaliser cette association, le compilateur génère la table des symboles définis dans le programme; chaque entrée de cette table contient le nom, l'adresse d'une variable ainsi qu'une indication sur le type de cette variable.

Aussi, pour permettre à notre environnement BUGPLI de connaître symboliquement les variables d'un programme,il suffit de lui fournir l'adresse de la table des symboles.

Il est par contre plus difficile de résoudre le problème posé par la reprise non séquentielle de l'exécution ou par la mise en place de points d'arrêt. Pour effectuer de telles opérations, il est nécessaire de nommer l'instruction PL/1 à laquelle on veut se transférer ou sur laquelle on veut s'arrêter. Une première méthode serait de nommer l'instruction en donnant son adresse d'implantation. Cette alternative nécessite que l'utilisateur connaisse, grâce à la liste des instructions-machine générées, l'adresse relative de chaque instruction PL/1 par rapport au début de la procédure qui la contient.

Une telle méthode est peu réaliste. Par contre, il est plus satisfaisant de définir le nom symbolique d'une instruction comme étant le numéro d'instruction qui lui est affecté par le compilateur. Aussi, si BUGPLI dispose d'une table contenant le numéro et l'adresse relative de chaque instruction de la procédure, il sera à même d'évaluer l'adresse-mémoire de toute instruction PL/1 de cette procédure.

En résumé, BUGPLI doit connaître :

- Pour chaque bloc, l'adresse de la table des symboles.
- Pour chaque instruction PL/1 exécutable, le numéro et l'adresse relative de l'instruction.

Ces informations seront captées lors de l'exécution du programme PL/1, par contre les mécanismes de transmission doivent être mis en place lors de la compilation.

De ce fait, la réalisation de notre environnement de mise au point fait apparaître deux points essentiels pour l'utilisateur :

- Le programme source est unique et il n'y a pas à lui faire subir de modification.
- Seuls les numéros d'instruction affectés par le compilateur doivent être connus pour se référer aux instructions PL/1.

Nous avons choisi d'exposer dans cette deuxième partie, BUGPLI vu et employé par l'utilisateur et de réserver la présentation de la structure interne à la troisième partie.

II - COMPILATION - CREATION D'UN PROGRAMME A MISE AU POINT INTERACTIVE

Nous appelons programme à mise au point intéractive, le résultat de la compilation d'un programme PL/1, avec l'option spécifique DEBUG.

EXEMPLE

PLI FACT ESSBUGI (DEBUG

Comme il a été dit dans la première partie, l'exécution de cette commande a pour but de compiler les fichiers FACT et ESSBUGI et donc de créer les fichiers FACT TEXT et ESSBUGI TEXT. Puisque l'option DEBUG est spécifiée, ces deux programmes peuvent être mis au point de manière intéractive.

Le compilateur PL/1 [I7,I4] n'a pas cette option, aussi nous l'avons modifié de façon à produire, pour chaque programme compilé avec l'option DEBUG, un fichier supplémentaire de type STPLI. Ce fichier contient pour chaque instruction PL/1 exécutable :

- Le numéro d'instruction affecté par le compilateur.
- L'adresse relative de la première des instructions-machine générées pour l'instruction PL/1.

Une seconde modification permet au compilateur d'insérer automatiquement, après toute instruction PROCEDURE, ENTRY ou BEGIN (c'est-à-dire au début de chaque bloc) la séquence :

CALL BUGPLID; PUT DATA; GET DATA;

Les instructions PUT DATA et GET DATA ne sont pas exécutées mais permettent d'une part de mettre en place le mécanisme d'adressage symbolique, lors de l'exécution, de l'ensemble des variables pouvant être traitées par ces instructions; et d'autre part, leur seule présence provoque automatiquement le chargement des sous-programmes de conversion (contenus dans la bibliothèque d'exécution de PL/1) nécessaires à l'édition et à la modification des contenus des variables.

L'instruction CALL BUGPLID matérialise pour nous le début de chaque bloc: le numéro de cette instruction sert d'identification au bloc dans lequel elle est contenue. De plus, à l'exécution, cette instruction permet à BUGPLI de repérer symboliquement les variables qui peuvent être traitées par les deux instructions (PUT DATA; GET DATA;) qui la suivent.

Précisons toutefois que les paramètres formels d'une procédure et les variables ayant la classe BASED ou les types POINTER, OFFSET, AREA, LABEL ne peuvent être connues symboliquement. En effet, les instructions PUT DATA et GET DATA ne permettent pas de traiter ces variables.

REMARQUES

1- Tous les messages énoncés dans la suite sont écrits en anglais; nos travaux sont destinés non seulement à une communauté d'utilisateurs français mais aussi une communauté d'utilisateurs européens dans le cadre du comité VM de l'organisation SEAS (Share European Association).

2- Dans la présentation des requêtes de BUGPLI, nous notons entre crochets les paramètres facultatifs et entre accolades un choix possible entre plusieurs paramètres; les valeurs par défaut, si elles existent, sont soulignées.

III - UTILISATION DE L'ENVIRONNEMENT BUGPLI.

III-1 Demande d'exécution d'un programme à mise au point intéractive.

Cette demande s'effectue en donnant le contrôle au point d'entrée IHECMS, à l'aide de la commande START du système CMS, dont le format est dans ce cas :

START IHECMS [Paramètres pour la procédure ayant l'attribut MAIN] (options....

Les deux seules options reconnues sont :

- TYPE Cette option permet à l'utilisateur d'effectuer, sur son terminal, les entrées-sorties relatives aux fichiers PL/1 SYSIN et SYSPRINT.

- {DEBUG | DE} Cette option indique que l'utilisateur désire contrôler l'exécution de son programme à l'aide de l'environnement BUGPLI.

Dans ce cas, avant d'activer la procédure possédant l'attribut MAIN, le contrôle est donné à BUGPLI qui émet le message :

BUGPLI : QUALIFY, PLACE STOP AND RESUME.

L'utilisateur peut alors placer un ou plusieurs arrêts sur instruction, en utilisant les requêtes QUALIFY et STOP. Il quitte ensuite l'environnement BUGPLI à l'aide de la requête RESUME. Le contrôle est alors donné à la procédure possédant l'attribut MAIN; l'exécution du programme PL/1 est maintenant supervisée par BUGPLI.

III-2 Activation asynchrone de BUGPLI.

BUGPLI prend automatiquement le contrôle sur interruption externe provoquée par l'utilisateur et émet, dans ce cas, les messages :

EXTERNAL INTERRUPTION
BUGPLI ENTERED DURING nom-de-procédure numéro-d'instruction

L'interruption qui vient d'arriver à entrainer la suspension de l'exécution du programme quelque part à l'intérieur d'une instruction PL/1. L'utilisateur doit obligatoirement placer un arrêt sur l'instruction PL/1 suivant celle qui est en cours d'exécution, puis reprendre l'exécution du programme supervisé par l'environnement de mise au point.

NOTE : Une interruption externe peut être générée en cours d'exécution d'un programme en appuyant sur la touche "Attention" du terminal IBM-2741 (ou sur la touche équivalente sur un terminal d'un autre type) et en émettant la fonction-console EXTERNAL.

III-3 Activation de BUGPLI sur erreur d'exécution

BUGPLI prend automatiquement le contrôle lorsqu'une erreur est détectée; généralement le sous-programme de traitement des erreurs de la bibliothèque d'exécution PL/1 rend le contrôle au superviseur à l'aide d'une instruction SVC de code 13.

L'utilisateur est alors averti par les messages :

BUGPLI : SVC 13
BUGPLI ENTERED DURING nom-de-procédure numéro-d'instruction

L'utilisateur se trouve alors dans l'environnement BUGPLI et dispose des requêtes pour essayer de déterminer la cause de l'erreur. Les requêtes GOTO et RESUME qui permettent une reprise de l'exécution, sont cependant interdites: le déroulement du programme n'ayant pû se faire correctement, il est aberrant de reprendre l'exécution après détection de l'erreur.

IV - LES REQUETES DE L'ACCOMPAGNATEUR BUGPLI

Précisons que le dialogue entre l'utilisateur et BUGPLI s'effectue entièrement à l'aide du terminal et que toute erreur détectée dans le décodage des paramètres d'une requête est signalée par le message '???'.

NOTE : Pour faciliter la lecture des exemples fournis, nous préfixerons par le signe → toutes les lignes frappées par l'utilisateur.

IV-1 Requêtes utilitaires

IV-1-1 Requête nulle ou Retour-Chariot (RC).

La réponse est "BUGPLI", message qui rappelle à l'utilisateur

que l'environnement BUGPLI est dans l'état "attente et lecture de requêtes".

EXEMPLE

→ (RC)
BUGPLI

IV-1-2 DUMP

{ DUMP | D } adresse-hexadécimale longueur-décimale

Cette requête fournit une image du contenu hexadécimal de la mémoire à partir de l'adresse spécifiée et sur la longueur indiquée.

EXEMPLE

→ DUMP 12A30 6
F1F2F3F4F5F6

Messages d'erreur :

. '2 ARGUMENTS'
Indique que l'utilisateur n'a pas fournit les deux arguments nécessaires.

. 'ARG 1'
Indique que l'adresse est invalide.

. 'ARG 2'
Indique que la longueur décimale est invalide.

IV-1-3 DO

DO nombre-hexadécimal opérateur nombre-hexadécimal

Cette requête effectue une opération arithmétique entre deux nombres hexadécimaux et imprime le résultat.

L'opérateur peut être +, -, *, /.
Un nombre hexadécimal est formé de un à huit chiffres hexadécimaux.

EXEMPLE

→ DO A3B6 * 5
3328E

Message d'erreur : ???

IV-1-4 DEBUG

{DEBUG}
{DE}

Cette requête permet d'accéder à l'environnement DEBUG de CMS. On reviendra à l'environnement BUGPLI à l'aide de la requête RETURN (ou RET).

EXEMPLE

→ DEBUG
DEBUG ENTERED
⋮ utilisation de
⋮ DEBUG de CMS
→ RETURN
→ (RC)
BUGPLI

NOTE : En permettant un accès à l'environnement DEBUG de CMS, cette requête offre à l'utilisateur, la possibilité de contrôler le fonctionnement de procédures écrites en langage d'assemblage.

IV-1-5 SPIE

SPIE { ON | OFF | DEBUG }

La fonction SPIE [I8,I9] du système CMS permet de signaler au système que l'utilisateur veut traiter les interruptions de type programme qui peuvent se produire pendant l'exécution de son programme. Aussi, tout programme PL/1 appelle la fonction SPIE et lui fournit l'adresse d'un sous-programme de traitement des erreurs provoquant des interruptions de type programme. Ce sous-programme est soit le sous-programme standard de la bibliothèque d'exécution PL/1, soit un bloc du programme PL/1 si l'utilisateur a programmé des instructions "ON-condition" [I6].

Parmi les interruptions de type programme, l'environnement DEBUG du système CMS traite normalement celles pour code opération invalide; en effet, un point d'arrêt de DEBUG est matérialisé par une instruction ayant un code opération invalide.

Aussi, l'exécution d'une instruction sur laquelle a été placée un point d'arrêt de DEBUG provoque une interruption programme pour code opération invalide. Le contrôle est alors donné soit à DEBUG s'il n'y a pas eu d'appel à la fonction SPIE, soit au sous-programme de traitement des erreurs mentionné lors du dernier appel de la fonction SPIE.

Par définition de la requête SPIE :

- SPIE OFF permet de désactiver la fonction SPIE du système. Toutes les interruptions de type programme seront traitées par DEBUG.

- SPIE DEBUG signale que toutes les interruptions de type programme sauf celles provoquées par la rencontre d'un code opération invalide, donnent le contrôle au sous-programme de traitement

d'erreur. Les interruptions de type programme pour code invalide sont traitées par DEBUG. En d'autres termes, les poins d'arrêt de DEBUG sont à nouveau autorisés.

- SPIE ON permet de restaurer la fonction SPIE c'est-à-dire annule les effets des deux requêtes SPIE OFF et SPIE DEBUG.

NOTE : Les requêtes SPIE et DEBUG permettent à l'utilisateur de placer des points d'arrêts à l'intérieur de procédures écrites en langage d'assemblage.

IV-1-6 CMS

CMS fonction-cms [paramètre-1...paramètre-n]

Cette requête permet d'exécuter une fonction-cms qui réside dans le noyau du système.

EXEMPLE

→ CMS OFFLINE PRINT LOAD MAP

Le code retour de la fonction sera, s'il est non nul, transmis à l'utilisateur à l'aide du message :

RETURN CODE = code-retour.

Messages d'erreur :

- INVALID FUNCTION NAME

L'utilisateur n'a pas fourni le nom d'une fonction-cms.

- NON RESIDENT FUNCTION

  La fonction que l'utilisateur désire exécuter ne réside pas dans le noyau du système. La requête est abandonnée: en effet, le chargement du module correspondant à la fonction détruirait partiellement le programme PL/1.

IV-1-7 IMPLANT

{ IMPLANT | IMP } symbole-de-type-externe

Cette requête fournit les renseignements sur ce symbole.

Si le symbole de type externe est un nom de point d'entrée (primair ou secondaire), l'utilisateur obtient les valeurs hexadécimales de l'adresse d'implantation en mémoire de ce point d'entrée et le facteur de translation qui lui a été appliqué.

EXEMPLE

```
→ IMP   BUGPLI1
  BUGPLI1  000157A0     000032C0
```

Si le symbole de type externe est un nom de pseudo-registre [I5,I7], l'utilisateur obtient les valeurs hexadécimales de la longueur de ce pseudo-registre et de son déplacement dans le vecteur des pseudo-registres (PRV).

EXEMPLE

```
→ IMP       IHEQSLA
  IHEQSLA       00000004      0000002C
```

Message d'erreur :

- NOT FOUND

  Le symbole de type externe n'existe pas.

IV-1-8 CVHEX

CVHEX nombre-décimal

Cette requête convertit en hexadécimal le nombre décimal donné en paramètre.

EXEMPLE

```
→ CVHEX    4108
  0000100c
```

IV-1-9 CVDEC

CVDEC nombre-hexadécimal

Cette requête convertit en décimal, le nombre hexadécimal fourni en paramètre.

EXEMPLE

```
→ CVDEC    100C
  4108
```

IV-2 Requêtes permettant un accès à l'environnement généré par PL/1.

Nous appelons environnement généré par PL/1, l'ensemble des blocs de contrôle et des informations qui sont gérés par PL/1 pour contrôler le déroulement du programme.

IV-2-1 TRACE

TRACE [n]

Cette requête fournit un historique du déroulement du programme en imprimant, pour les n dernières procédures actives, le nom de la procédure et le numéro de la dernière instruction exécutée dans cette procédure.

Lorsque le paramètre n est omis, l'historique décrit le déroulement entre la procédure possédant l'attribut MAIN et la dernière procédure active.

L'ordre d'impression est inverse de l'ordre d'exécution.

EXEMPLE

```
→ TRACE  3

    STATEMENT   13   PROCEDURE   FACT
    STATEMENT   23   PROCEDURE   FACT
    STATEMENT   23   PROCEDURE   FACT
```

Dans cet exemple, FACT est une procédure récursive qui calcule la factorielle d'un nombre.

IV-2-2 PLIST

PLIST nom-de-procedure [RECURS numéro d'appel récursif]

Cette requête fournit l'adresse et l'image du contenu de la liste des paramètres transmise à la procédure nommée, lors de son nième appel récursif (ou lors du premier appel, si le numéro d'appel récursif est omis).

Le paramètre nom-de-procédure représente le nom d'une procédure interne ou externe.

Par convention :

- En PL/1, la transmission des paramètres se fait par adresse.
- Lors de l'appel d'une procédure, la liste des paramètres est repérée par le registre général 1;la fin de la liste est matérialisée par la valeur exadécimale 80 dans l'octet de gauche du dernier mot.

EXEMPLE

```
→ PLIST      FACT     RECURS    2
  PARMLIST    AT      0001E270
  0001DE3C
  8001E1BC

  PLIST      FACT
  PARMLIST       AT   0001DF40
  0001DE3C
  8001DE40
```

PLIST. nom-de-procédure [RECURS numéro-d'appel-récursif]

Cette deuxième forme ne fournit que l'adresse de la liste des paramètres.

EXEMPLE

```
→ PLIST.   FACT   RECURS   2
  PARMLIST    AT    0001E270
→ PLIST.   FACT
  PARMLIST  AT  0001DF40
```

Messages d'erreur :

- NOT FOUND

  La procédure nommée n'est pas active à cet instant ou le $n^{ième}$ appel récursif n'a pas été effectué.

- NO PARMLIST

  La procédure nommée a été appelée sans paramètres.

IV-2-3 <u>RECURS</u>

RECURS   nom-de-procédure

Cette requête fournit le nombre d'appels récursifs de la procédure interne ou externe nommée.

EXEMPLE

```
→ RECURS        FACT
  5
```

Message d'erreur :

- NOT FOUND

  La procédure nommée n'est pas active.

IV-2-4 <u>PLIREG</u>

PLIREG

Cette requête imprime le contenu des seize registres généraux tels qu'ils existaient avant que l'environnement BUGPLI soit activé.

Rappelons que, par convention, les registres possèdent les propriétés suivantes :

R1	Repère la liste des paramètres
R10	Est le registre de base de la procédure PL/1.
R11	Repère la section de contrôle qui contient les variables de type STATIC (STATIC INTERNAL STORAGE CSECT).[I5]
R12	Repère le vecteur des pseudo-registres.
R13	Repère la zone de sauvegarde courante.
R14 et R15	Sont utilisés pour l'appel des procédures et des sous-programmes de la librairie d'exécution.

EXEMPLE

→ PLIREG
00000018
⋮
000203C4

IV-2-5 PRVPT

PRVPT

Cette requête imprime l'adresse du vecteur des pseudo-registres, c'est-à-dire le contenu du registre général 12.

EXEMPLE

→ PRVPT
00019C18

IV-2-6 <u>DSAPT</u>

DSAPT

Cette requête imprime le contenu du pseudo-registre IHEQSLA qui repère le dernier bloc de contrôle DSA ou VDA actif (Cf.Troisième partie, §V-1 - Chaînage des blocs: la liste des DSA et VDA).

EXEMPLE

→ DSAPT
0001A4D0

IV-2-7 <u>PREG</u>

{ PREG / PR } nom-pseudo-registre

Cette requête imprime l'adresse et le contenu du pseudo-registre nommé.

EXEMPLE

→ PR IHEQSLA
AT 00019C64
0001A4D0

La forme :

{ PREG. / PR. } nom-pseudo-registre

Ne fournit que l'adresse du pseudo-registre.

EXEMPLE

→ PREG. IHEQSLA
AT 00019C64

Message d'erreur :

NOT FOUND
Le pseudo-registre n'existe pas.

IV-2-8 QUALIFY

{ QUALIFY | QUAL } { nom-de-procédure-externe | ? }

a- La forme "QUALIFY nom-de-procédure-externe' signale à l'environnement BUGPLI que l'on se réfère désormais, à la procédure externe de nom "nom-de-procédure-externe". BUGPLI utilise alors le fichier "nom-de-procédure-externe STPLI" pour évaluer les adresses d'implantation des instructions PL/1 contenues dans la procédure nommée.

EXEMPLE

→ QUAL FACT

Messages d'erreur :

- PROC nom-de-procédure-externe NOT FOUND
  La procédure n'existe pas en mémoire.

- FILE nom-de-procédure-externe NOT FOUND

  Il n'existe aucun fichier de nom "nom-de-procédure-externe STPLI"

- ERROR DURING RDBUF

  Erreur pendant la lecture d'un fichier de type STPLI.

b- La forme "QUALIFY␣?" imprime le nom du fichier de type STPLI qui est actuellement utilisé pour le calcul des adresses des instructions PL/1.

EXEMPLE

→ QUAL FACT
→ QUAL ?
FACT

NOTE : La requête "QUAL nom-de-procédure-externe" doit être employée avant d'utiliser les requêtes STOP et GOTO.

IV-2-9 ORG

ORG [Nom-de-procédure--externe BLOCK Identification de bloc [RECURS Numéro-d'appel récursi]]

Cette requête a deux fonctions :

a- Avec paramètres, elle permet d'accéder à l'environnement PL/1 qui existait lors du $n^{ième}$ appel récursif du bloc identifié par le doublet (nom-de-procédure-externe, identification-de-bloc). Notons que le bloc ainsi repéré doit être actif lorsque l'utilisateur émet cette requête.

On peut, de cette façon, accéder aux variables définies dans un bloc disjoint ou englobant le bloc courant.

EXEMPLE

→ ORG ESSBUGI BLOCK 2

L'utilisateur demande l'accès aux variables du bloc "2" de la procédure ESSBUGI.

→ ORG FACT BLOCK 14 RECURS 5

L'utilisateur demande l'accès aux variables de la cinquième génération du bloc "14" de la procédure FACT.

b- <u>Sans paramètre</u>, cette requête restaure l'environnement du bloc courant de façon à accéder aux variables de ce bloc.

EXEMPLE

→ ORG

On a maintenant à accéder aux variables du bloc courant.

Messages d'erreur :

- BLOCK NOT FOUND

  Le bloc défini par le doublet (nom-de-procédure-externe, identification-de-bloc) ou le triplet (nom-de-procédure-externe, identification-de-bloc, numéro-d'appel-récursif) n'existe pas dans la pile d'exécution.

- NON-RECURSIVE BLOCK

  On a émis la requête en fournissant les paramètres RECURS numéro-d'appel-récursif, alors que le bloc concerné n'est pas récursif.

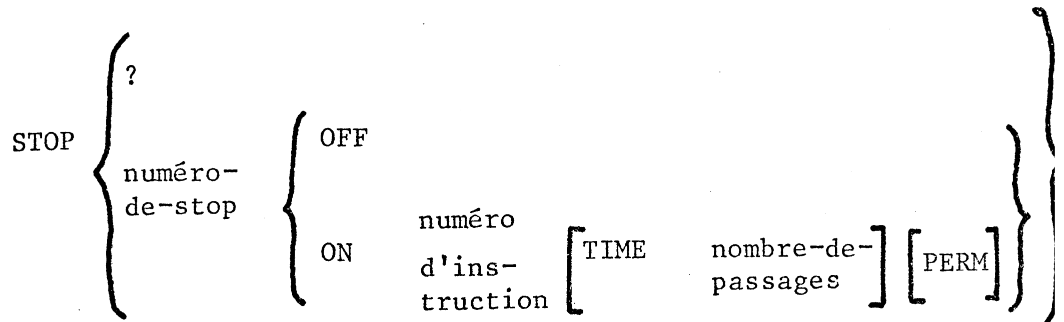
NOTE : Rappelons que l'identification d'un bloc est le numéro de l'instruction PL/1 CALL BUGPLID qui se trouve en tête de ce bloc.

REMARQUES

- Toute erreur détectée soit pendant l'analyse des paramètres, soit pendant l'exécution de la requête, restaure automatiquement l'accès aux variables du bloc courant.

- Les deux requêtes GOTO et RESUME qui permettent de reprendre l'exécution du programme, restaurent l'accès aux variables du bloc courant.

IV-2-10 STOP



a- La forme "STOP numéro-de-stop ON....." permet de placer un arrêt, identifié par "numéro-de-stop", sur l'instruction repérée par "numéro-d'instruction",et appartenant à la procédure précédemment qualifiée à l'aide de la requête QUALIFY.

La valeur n du paramètre "nombre-de-passages" (opérande du mot clé TIME) indique à BUGPLI qu'il ne devra marquer l'arrêt que lors de la $n^{ième}$ exécution de l'instruction concernée.

Le mot clé PERM signale que l'arrêt que l'on place possède l'attribut permanent.

Précisons que "numéro-de-stop" peut prendre les valeurs de 1 à 15.

EXEMPLE

```
→QUAL  FACT
→STOP  1  ON  25
→STOP  2  ON  10  PERM
→STOP  3  ON  21  TIME  6
→STOP  4  ON  22  TIME  5  PERM
```

On qualifie la procédure FACT, puis on place :

- un arrêt de numéro 1 sur l'instruction PL/1 de numéro 25 de la procédure FACT.
- un arrêt permanent de numéro 2 sur l'instruction 10.
- un arrêt de numéro 3, à ne prendre en compte que lors de la sixième exécution exécution de l'instruction 21.
- un arrêt permanent de numéro 4, à ne prendre en compte que toutes les cinq exécutions de l'instruction 22.

b- La forme "STOP numéo-de-stop OFF" permet d'annuler l'arrêt "numéro-de-stop".

EXEMPLE

```
→QUAL  FACT
→STOP 6 ON 18
→STOP 6 OFF
```

c- La forme "STOP ?" fournit la liste des arrêts actifs et l'identification des instructions PL/1 correspondantes.

Le message envoyé à l'utilisateur a la forme suivante :

STOP numéro-de-stop ON nom-de-procédure-externe numéro-d'instruction

[TIME nombre-de-passages, nombre-de-passages restant à faire] [PERM]

EXEMPLE

```
→STOP  ?
 STOP  1  ON  FACT  25
 STOP  2  ON  FACT  10  PERM
 STOP  3  ON  FACT  21  TIME  6, 1
 STOP  4  ON  FACT  22  TIME  5, 0  PERM
```

<u>NOTE</u> : Précisons qu'il existe seulement quatre façons de désactiver un arrêt sur instruction :

- Emettre la requête "STOP numéro-de-stop OFF".
- Exécuter l'instruction PL/1 sur laquelle se trouve l'arrêt, s'il ne possède pas l'attribut permanent.
- Placer un arrêt de même numéro sur une autre instruction PL/1 exécutable.
- Placer un arrêt de même numéro sur une instruction PL/1 qui n'existe pas.

Messages d'erreur :

- PLI STATEMENT numéro-d'instruction NOT FOUND

  L'instruction "numéro-d'instruction" n'existe pas dans la procédure PL/1 qualifiée.

- ERROR DURING RDBUF

  Erreur pendant la lecture du fichier de type STPLI et de nom égal au nom de la procédure qualifiée.

- REPLACE STOP numéro-de-stop ON nom-de-procédure-externe numéro-d'instruct

  L'arrêt "numéro-de-stop" qui existait sur l'instruction identidiée par("nom-de-procédure-externe", "numéro-d'instruction") a été annulé. Cet arrêt se trouve maintenant sur l'instruction nommée dans la requête
  "STOP numéro-de-stop ON numéro-d'instruction..."

- DELETE STOP numéro-de-stop ON nom-de-procédure-externe numéro-d'instruction

L'utilisateur reçoit ce message lorsqu'il place un arrêt sur une instruction qui en contenait déjà un. L'ancien arrêt est annulé et le nouvel arrêt est pris en compte.

- NON-PERMANENT STOP

BUGPLI s'attendait à trouver le mot clé PERM; ne l'ayant pas reconnu, il signale que l'arrêt qu'il vient de placer, ne possèd pas l'attribut permanent.

IV-2-11 RESUME

RESUME

Cette requête permet la reprise séquentielle de l'exécution du programme PL/1.

REMARQUE

Avant de reprendre l'exécution, l'accès aux variables du bloc courant est automatiquement restauré, si nécessaire.

EXEMPLE

BUGPLI ENTERED: STOP 1 ON FACT 25

⋮

→ RESUME

L'utilisation de cette requête est interdite lorsque l'environnement BUGPLI a reçu le contrôle après détection d'une erreur d'exécution. Dans ce cas, l'utilisateur reçoit le message d'erreur :

IT'S IMPOSSIBLE TO PERFORM THIS REQUEST.

IV-2-12 GOTO

GOTO numéro-d'instruction BLOCK identification-de-bloc

Cette requête permet de reprendre l'exécution du programme à l'instruction "numéro-d'instruction" contenue dans le bloc désigné par "identification-de-bloc", de la procédure précédemment nommée à l'aide de la requête QUALIFY.

Cette requête n'est acceptée que si l'environnement BUGPLI a été activé par la rencontre d'un arrêt sur instruction.

REMARQUE

Avant de reprendre l'exécution, l'accès aux variables du bloc courant est automatiquement restauré, si nécessaire.

EXEMPLE

```
BUGPLI ENTERED: STOP 1 ON FACT 25
.
.
.
→QUALIFY  ESSBUGI
→GOTO 19 BLOCK 11
```

Messages d'erreur :

- STATEMENT NOT FOUND

  L'instruction PL/1 définie par le couple ("numéro-d'instruction" "identification-de-bloc") ne peut être trouvée dans la procédure qualifiée.

  Il existe deux causes à cette erreur :
  - Soit le paramètre "numéro-d'instruction" est erroné.
  - Soit le bloc repéré par "identification-de-bloc" n'est pas actif.

- IT'S IMPOSSIBLE TO PERFORM THIS REQUEST

Ce message signale que la requête ne peut être satisfaite car BUGPLI n'a pas été activé par la rencontre d'un arrêt sur instruction.

IV-2-13 BLOCK

BLOCK [n]

Cette requête fournit une image de la pile d'exécution constituée par les blocs de contrôle DSA et VDA, le vecteur des pseudo-registres (PRV) et la zone de sauvegarde externe.

Le paramètre n permet de limiter l'impression aux n dernières structures de la pile.

L'utilisateur obtient pour chaque structure, l'adresse et une indication sur la nature de la structure, ainsi que l'image des deux ou trois premiers mots qu'elle contient. De plus, si la structure est une DSA, on obtient le "INVOCATION COUNT" et le numéro (en décimal) de la dernière instruction exécutée dans le bloc correspondant à cette DSA. Si la DSA est relative à une procédure, on obtient aussi l'adresse d'implantation et le nom de cette procédure.

EXEMPLE

```
→ BLOCK
  VDA  AT  0001DF88 : 200001980001DEB8
  DSA  AT  0001DEB8 : D00000D0 0001DD90 00000000
   INVOCATION COUNT : 0000000000000002 STATEMENT:  15
  DSA  AT  0001DD90 : C00001280001D610 0001DEB8
   AT  00013070 PROCEDURE ESSBUGI
   INVOCATION COUNT : 0000000000000001 STATEMENT:   9
  PRV  AT   0001D610 : 29000078000122E8
  SA EXT AT 000122E8 : 00000120 0000DB080001DD90
```

IV-3 Requêtes permettant l'accès aux variables PL/1 de type arithmétique ou chaîne.

Seules sont accessibles les variables connues de manière symbolique, c'est-à-dire les variables dont le nom et le contenu peuvent être imprimés à l'aide d'une instruction PUT DATA. De ce fait, les paramètres formels d'une procédure et les variables de type POINTER, OFFSET, AREA ou de classe BASED, ne peuvent être traités par BUGPLI.

Les requêtes présentées dans la suite ne traitent que les variables connues symboliquement, qui appartiennent soit au bloc PL/1 courant, soit à un bloc dont on a obtenu l'environnement à l'aide d'une requête ORG. De plus, les variables doivent être entièrement qualifiées. Dans l'exposé qui suit, nous ne ferons plus mention de ces conditions.

IV-3-1 NAMES

NAMES

Cette requête fournit le nom de toutes les variables connues dans le bloc.

EXEMPLE

```
→ NAMES
    O
    P
    Q
    T
→   ORG  ESSBUGI  BLOCK  2
→   NAMES
     I
     J
     K
     N
    BUF
```

IV-3-2 SYMTAB

SYMTAB   nom-de-variable

Cette requête imprime l'adresse et le contenu de l'entrée de la table des symboles qui définit la variable nommée.

EXEMPLE

```
→ SYMTAB  T
  AT   00016D04
  00016CB801E3404001016D1801000090008C0000
```

La forme :

SYMTAB.   nom-de-variable

n'imprime que l'adresse de l'entrée de la table des symboles qui définit la variable.

EXEMPLE

```
→ SYMTAB.  T
   AT  00016D04
```

Message d'erreur :

- NOT FOUND

La variable nommée n'existe pas.

IV-3-3 DED

DED   nom-de-variable

Cette requête imprime l'adresse, la longueur et le contenu du "Data Element Descriptor" associé à la variable.

EXEMPLE

```
→ DED   T
  A(DED) = 01016D18   LN(DED) = 00000003
  8C1F80
```

La forme

DED.   nom-de-variable

n'imprime que l'adresse et la longueur du "Data Element descriptor".

EXEMPLE

```
→ DED.   T
  A(DED) = 01016D18    LN(DED) = 00000003
```

Message d'erreur :

- NOT FOUND

  La variable nommée n'existe pas.

IV-3-4 DV

DV nom-de-variable

Cette requête imprime l'adresse, la longueur et le contenu du "Dope Vector" décrivant la variable.

EXEMPLE

```
→ DV   T
  A(DV) = 000206A0     LN(DV) = 0000000C
  010206E400000004 003C0001
```

La forme

DV.   nom-de-variable

n'imprime que l'adresse et la longueur du "Dope Vector" de la variable.

EXEMPLE

```
→ DV   T
  A(DV) = 000206A0     LN(DV) = 0000000C
```

Messages d'erreur :

- NOT FOUND

  La variable n'existe pas

- NO DOPE VECTOR

  La variable ne possède pas de "Dope Vector".

IV-3-5 ADDR

ADDR   nom-de-variable [(indice,...)]

Cette requête imprime l'adresse, la longueur et le contenu (sous forme hexa-décimale) de la variable.

Si la variable est dimensionnée et que l'on ne fournisse pas de liste d'indices, BUGPLI prend pour indices les valeurs des bornes inférieures de la variable.

EXEMPLE

```
→ ADDR  T
  A(VAR) = 000206E8   LN(VAR) = 00000004
  00000006
→ ADDR  T(1)
  A(VAR) = 000206E8    LN(VAR = 00000004
  00000006
```

La forme

ADDR. nom-de-variable [(indice,...)]

n'imprime que l'adresse et la longueur de la variable.

EXEMPLE

```
→ ADDR. T
  A(VAR) = 000206E8    LN(VAR) = 0000004
```

Messages d'erreur :

- NOT FOUND

  La variable n'existe pas.

- SUBSCRIPT ERROR

  Un des indices fournis est erroné.

- DIMENSION ERROR

Le nombre d'indices donnés n'est pas égal au nombre de dimensions de la variable.

IV-3-6 ATTR

```
ATTR   nom-de-variable
```

Cette requête interprète le contenu des descripteurs de la variable et fournit la liste des attributs de la variable.

EXEMPLES

- Pour la variable CHARIOV définie par :

```
DCL  CHARIOV  CHAR(10)  VARYING;

→ ATTR  CHARIOV
  CHARIOV
  CHAR ( 10) VARYING
```

- Pour la variable I définie par :

```
DCL  I  FIXED  BIN  (31,8)  STATIC;

→ ATTR  I
  I
  STATIC
  BIN   FIXED  (31,8)
```

- Pour la variable AFB310 définie par :

```
N = 2;
M = 4;
DCL  AFB310 (-N:M, -M:N)  FIXED BIN (31,0);
```

```
→ ATTR   AFB310
     AFB310
      BIN  FIXED  (31, 0)
     2 DIMENSIONS
     LOWER  BOUND        UPPER BOUND
         -2                   4
         -4                   2
```

Message d'erreur :

- NOT FOUND

La variable n'existe pas.

IV-3-7 LENGTH

LENGTH   nom-de-variable [(indice,...)]

Cette requête imprime la longueur de la variable. Si la variable nommée est de longueur variable, l'utilisateur obtient les longueurs courante et maximale.

EXEMPLES

- Pour la variable BIT8 telle que :

```
DCL  BIT8  BIT(8);

→ LENGTH  BIT8
  LENGTH=    8
```

- Pour la variable CHARIOV telle que :

```
DCL  CHARIOV  CHAR(10)  VARYING;
CHARIOV='CHARIOV';

→ LENGTH  CHARIOV
  CURRENT LENGTH = 7     MAX. LENGTH = 10
```

Messages d'erreur :

- NOT FOUND

  La variable n'existe pas.

- SUBSCRIPT ERROR

  Un des indices fournis est erroné.

- DIMENSION ERROR

  Le nombre d'indices donnés n'est pas égal au nombre de dimensions de la variable.

IV-3-8 DISPLAY

$\left\{ \begin{array}{l} \text{DISPLAY} \\ \text{DIS} \end{array} \right\}$ nom-de-variable [(indice,...)]

Cette requête imprime, sous forme éditée, le contenu de la variable.

EXEMPLE

```
→ DIS   Q
        60
→ DIS   T (40)
        6
```

Messages d'erreur :

- NOT FOUND

  La variable n'existe pas.

- SUBSCRIPT ERROR

  Un des indices donnés est erroné.

- DIMENSION ERROR

  Le nombre d'indices donnés n'est pas égal au nombre de dimensions de la variable.

IV-3-9 <u>SET</u>

SET nom-de-variable [(indice,...)] valeur

Cette requête permet de modifier la valeur de la variable à l'aide de la valeur fournie.

EXEMPLE

```
→ DIS    X
    'ESSBUGC'
→ SET  X  'MODIF'
→ DIS  X
   'MODIF  '
→ DIS  T(40)
        6
→ SET  T(40)  40
→ DIS  T(40)
       40
```

Messages d'erreur :

- NOT FOUND

  La variable n'existe pas.

- SUBSCRIPT ERROR

  Un des indices donnés est erroné.

- DIMENSION ERROR

  Le nombre d'indices donnés n'est pas égal au nombre de dimensions de la variable.

NOTES :

- Les variables doivent être complètement qualifiées.
- Les requêtes SET et DISPLAY utilisent les sous-programmes de conversion de la librairie d'exécution PL/1.
- L'utilisation des requêtes SET et DISPLAY n'est pas possible si BUGPLI a été activé par interruption externe. L'utilisateur est averti de cette restriction, par le message :
  IT'S IMPOSSIBLE TO PERFORM THIS REQUEST.

IV-4 Requêtes permettant l'accès aux variables PL/1 de type fichier.

IV-4-1 FILES

FILES

Cette requête imprime le nom de chaque fichier PL/1 actif,soit en écriture, soit en lecture.

Message d'erreur :

- NO OPENED FILES
  Il n'existe aucun fichier PL/1 actif.

IV-4-2 DCLCB

DCLCB nom-de-fichier-PL/1

Cette requête imprime l'adresse et le contenu du "Declare Control Block" associé au fichier PL/1 actif.

La forme

DCLCB. nom-de-fichier-PL/1

n'imprime que l'adresse du "Declare Control Block".

Message d'erreur :

- NOT FOUND
  Le fichier n'existe pas ou n'est pas actif.

IV-4-3 FCB

FCB nom-de-fichier-PL/1

Cette requête imprime l'adresse et le contenu du "File Control Block" associé au fichier PL/1 actif.

La forme

FCB. nom-de-fichier-PL/1

n'imprime que l'adresse du "File Control Block".

Message d'erreur :

- NOT FOUND
  Le fichier n'existe pas ou n'est pas actif.

IV-4-4 PRFILE

PRFILE nom-de-fichier-PL/1

Cette requête imprime l'adresse et le contenu du pseudo registr

associé au fichier PL/1 ouvert.

La forme

PRFILE. nom-de-fichier-PL/1

n'imprime que l'adresse du pseudo-registre.

Message d'erreur :

- NOT FOUND
  Le fichier n'existe pas ou n'est pas actif.

IV-5 Requête permettant l'examen du contenu d'un fichier: EXAMINE

{ EXAMINE | EXAM } nom type { mode | __P1__ }

Cette requête donne le contrôle à l'environnement EXAMINE qui, comme nous le verrons au paragraphe suivant, permet d'examiner le contenu d'un fichier.

Les paramètres "nom, "type", "mode" constituent l'identification d'un fichier-cms résidant sur disque.

Messages d'erreur :

- NO FILENAME AND FILETYPE
  L'utilisateur n'a pas fourni d'identification de fichier.

- NO FILETYPE
  L'identification du fichier est incomplète.

- FILE NOT FOUND

  Le fichier n'existe pas.

- FILE EMPTY

  Le fichier ne contient aucun enregistrement.

V - LES REQUETES DE L'ENVIRONNEMENT EXAMINE.

L'entrée dans cet environnement est signalée par le message :

EXAMINE nom type mode

L'environnement EXAMINE ne permet que la consultation du fichier; à l'entrée dans cet environnement, le premier enregistrement du fichier devient l'enregistrement courant pour EXAMINE. Les requêtes disponibles ont un fonctionnement identique à celui des requêtes correspondantes de l'environnement EDIT de CMS; elles sont :

V-1 Impression: PRINT.

$$\left\{ \begin{matrix} \text{PRINT} \\ \text{P} \end{matrix} \right\} \quad \left\{ \begin{matrix} \text{n} \\ \underline{1} \end{matrix} \right\}$$

Cette requête imprime le contenu de n enregistrements à partir de l'enregistrement courant; le dernier enregistrement traité devient l'enregistrement courant.

Si la fin du fichier est atteinte avant d'avoir traité n enregistrements, l'utilisateur en est averti par le message: EOF.

V-2 Positionnement en début du fichier: TOP

$$\left\{ \begin{array}{l} \text{TOP} \\ \text{T} \end{array} \right\}$$

Le premier enregistrement du fichier devient l'enregistrement courant.

V-3 Positionnement en fin du fichier : BOTTOM .

$$\left\{ \begin{array}{l} \text{BOTTOM} \\ \text{B} \end{array} \right\}$$

Le dernier enregistrement du fichier est imprimé et devient l'enregistrement courant.

V-4 Positionnement sur un enregistrement : GOTO

$$\left\{ \begin{array}{l} \text{GOTO} \\ \text{GO} \end{array} \right\} \quad n$$

L'enregistrement de numéro n est imprimé et devient l'enregistrement courant. Si cet enregistrement n'existe pas, l'utilisateur en est averti par le message: EOF.

V-5 Déplacement vers le début du fichier: UP

$$\begin{Bmatrix} \text{UP} \\ \text{U} \end{Bmatrix} \quad \begin{Bmatrix} n \\ \underline{1} \end{Bmatrix}$$

Le n^{ième} enregistrement qui précède l'enregistrement courant est imprimé et devient le nouvel enregistrement courant. Si le début du fichier est dépassé au cours du déplacement, l'utilisateur en est averti par le message: TOP; le premier enregistrement du fichier peut alors être atteint à l'aide des requêtes PRINT ou NEXT.

V-6 Déplacement vers la fin du fichier: NEXT.

$$\begin{Bmatrix} \text{NEXT} \\ \text{N} \end{Bmatrix} \quad \begin{Bmatrix} n \\ \underline{1} \end{Bmatrix}$$

Le n^{ième} enregistrement qui suit l'enregistrement courant est imprimé et devient le nouvel enregistrement courant.

V-7 Identification de l'enregistrement courant: LINENO

$$\begin{Bmatrix} \text{LINENO} \\ \text{LI} \end{Bmatrix}$$

Cette requête imprime le numéro d'ordre de l'enregistrement courant.

V-8 Sortie de l'environnement EXAMINE: QUIT.

```
{ QUIT }
{      }
{ Q    }
```

Cette requête permet de quitter l'environnement de consultation de fichier et de revenir dans l'environnement BUGPLI.

NOTES :
- La requête nulle (ou retour chariot) imprime le message: EXAMINE.
- Toute erreur détectée lors de l'analyse des paramètres d'une requête est signalée par le message: ???
- Le retour à l'environnement BUGPLI est effectué sur :
  . émission de la requête QUIT,
  . détection d'une erreur d'entrée-sortie pendant la consultation du fichier.

VI-- RECAPITULATIF DES REQUETES

VI-1 Requêtes de BUGPLI

Requête		Section
{ ADDR } / { ADDR. }	nom-de-variable [(indice,...)]	IV-3-5
ATTR	nom-de-variable	IV-3-6
BLOCK	[n]	IV-2-3

Commande	Paramètres	Référence
CMS	fonctions-cms [paramètre-1...paramètre-n]	IV-1-6
CVDEC	nombre-hexadécimal	IV-1-9
CVHEX	nombre-décimal	IV-1-8
{ DCLCB DCLCB. }	nom-de-fichier PL/1	IV-4-2
{ DEBUG DE }		IV-1-4
{ DED DED. }	nom-de-variable	IV-3-3
{ DISPLAY DIS }	nom-de-variable [(indice,...)]	IV-3-8
DO	nombre-hexadécimal opérateur nombre-hexadécimal	IV-1-3
DSAPT		IV-2-6
{ DUMP D }	adresse-hexadécimale longueur-décimale	IV-1-2
{ DV DV. }	nom-de-variable	IV-3-4
{ EXAMINE EXAM }	nom type [mode]	IV-5
{ FCB FCB. }	nom-de-fichier-PL/1	IV-4-3

FILES — IV-4-1

GOTO numéro-d'instruction BLOCK identification-de-bloc — IV-2-12

{IMPLANT | IMP} symbole-de-type-externe — IV-1-7

LENGTH nom-de-variable [(indice,...)] — IV-3-7

NAMES — IV-3-1

ORG [nom-de-procédure-externe BLOCK identification-de-bloc [RECURS numéro-d'appel-récursif]] — IV-2-9

PLIREG — IV-2-4

{PLIST | PLIST.} nom-de-procédure [RECURS numéro-d'appel-récursif] — IV-2-2

{PREG | PR | PREG. | PR.} nom-de-pseudo-registre — IV-2-7-

PRFILE nom-de-fichier-PL/1 — IV-4-4

PRVPT — IV-2-5

{QUALIFY / QUAL}	{nom-de-procédure-externe / ?}	IV-2-8
RECURS	nom-de-procédure	IV-2-3
RESUME		IV-2-11
SET	nom-de-variable [(indice,..)] valeur	IV-3-9
SPIE	{ON / OFF / DEBUG}	IV-1-5
STOP	{? / numéro-de-stop} {OFF / ON numéro-d'instruction [TIME nombre-de-passages] [PERM]}	IV-2-
{SYMTAB / SYMTAB.}	nom-de-variable	IV-3-2
TRACE	[n]	IV-2-1
(RC)		IV-1-1

VI-2 Requêtes de EXAMINE

{BOTTOM / B}	V-3

Command	Operand	Section
GOTO / GO	n	V-4
LINENO / LI		V-7
NEXT / N	n / $\underline{1}$	V-6
PRINT / P	n / 1	V-1
QUIT / Q		V-8
TOP / T		V-2
UP / U	n / $\underline{1}$	V-5

T R O I S I E M E   P A R T I E

STRUCTURE INTERNE DE BUGPLI

T R O I S I E M E   P A R T I E

STRUCTURE INTERNE DE BUGPLI

I   STRUCTURE GENERALE DE BUGPLI.

II   NOM SYMBOLIQUE D'UNE INSTRUCTION.

III   NOMS DES VARIABLES DU PROGRAMME.

- III-1 La table des symboles.
- III-2 Accès à la table des symboles.

IV   ACCES AUX VARIABLES. ADRESSAGE.

- IV-1 Les descripteurs de variables: "Dope Vectors".
- IV-2 Adressage d'une variable.

V   NOTION DE BLOC PL/1.

- V-1 Chaînage de blocs: la liste des DSA et VDA.
- V-2 Initialisation d'un bloc.
- V-3 Fermeture d'un bloc.
- V-4 Blocs contenus dans une procédure récursive.
- V-5 Intéraction entre BUGPLI et le mécanisme de gestion des blocs PL/1.
- V-6 La pile de sauvegarde des blocs.
- V-7 Traitement de la pile de sauvegarde des blocs.
- V-8 Que permet la pile de sauvegarde ?

VI   IDENTIFICATION DE PROCEDURE.

- VI-1 Prologue de procédure.
- VI-2 Recherche par BUGPLI du nom de la dernière procédure externe activée.

VII ETUDE DES MECANISMES DE L'ACCOMPAGNATEUR BUGPLI CONCERNANT LES ARRETS SUR INSTRUCTIONS.

VII-1 La table des arrêts sur instruction.
VII-2 Annulation d'un arrêt sur instruction.
VII-3 Mise en place d'un arrêt sur instruction.
VII-4 Suspension de l'exécution lors de la rencontre d'un arrêt.
VII-5 Reprise de l'exécution après un arrêt.

VII-5-1 Reprise séquentielle.
VII-5-2 Reprise non séquentielle.

I - STRUCTURE GENERALE DE BUGPLI.

L'environnement de mise au point que nous appelons aussi l'accompagnateur d'exécution PL/1 est un ensemble de programmes qui sont chargés en mémoire en même temps que le programme de l'utilisateur.

L'adressage symbolique des instructions PL/1 est résolu à l'aide des informations contenues dans les fichiers de type STPLI (cf.Partie 2, §II); l'accès symbolique aux variables est possible grâce à l'insertion des instructions CALL BUGPLID; PUT DATA; GET DATA;.

Les problèmes liés aux notions de bloc et de récursivité sont résolus par échange d'informations entre BUGPLI et le mécanisme de gestion de la pile d'exécution.

II - NOM SYMBOLIQUE D'UNE INSTRUCTION.

Nous avons défini le nom symbolique d'une instruction PL/1 exécutable comme le numéro qui est affecté à cette instruction par le compilateur. Nous savons que l'adresse-mémoire d'une instruction PL/1 peut être calculée pendant l'exécution, si l'on connait :

- l'adresse de chargement de la procédure,
- le numéro de cette instruction et son adresse relative dans la procédure.

Pour acquérir ces informations, l'interface de compilation transmet systématiquement au compilateur, l'option STMT, lorsqu'il détecte la présence de l'option DEBUG. De plus, nous avons modifié le compilateur pour qu'il transmette à l'interface du compilateur, pour chaque instruction PL/1 exécutable, le numéro de cette instruction et l'adresse relative de la première des instructions-machine générées. Ces informations sont sauvegardées dans un fichier-cms de type "STPLI" si la compilation a lieu

avec l'option DEBUG. Un enregistrement de ce fichier est formé de deux mots-mémoire, soit huit octets; le premier mot contient le numéro d'une instruction PL/1, le deuxième mot l'adresse relative de cette instruction. Les enregistrements de ce fichier sont rangés dans l'ordre croissant des numéros d'instructions.

III - NOMS DES VARIABLES DU PROGRAMME .

Pour que l'accompagnateur d'exécution puisse accéder aux noms et aux adresses des variables du programme PL/1, nous utilisons une instruction PUT DATA (ou GET DATA) qui, rappelons-le, force le compilateur à créer la table des symboles pour identifier toutes les variables connues dans le bloc qui contient cette instruction.

III-1 La table des symboles.

La table des symboles, appelée aussi SYMTAB, est formée d'entrées; les entrées sont chaînées les unes aux autres et chaque entrée définit une variable.

Par exemple, considérons un programme dont la structure peut se résumer ainsi :

```
bloc1......PROG:    PROC ;
    :            DCL   A,B,C...
    :             :
    :             :
    :bloc 2....BEGIN;
    :         :
    :         :        DCL    A,D,E...
    :         :         :
    :         :         :
    :         :...... END;
    :              :
    :              :
    :.......... END    PROG;
```

La table des symboles de ce programme est schématisée par la figure III-1-a.

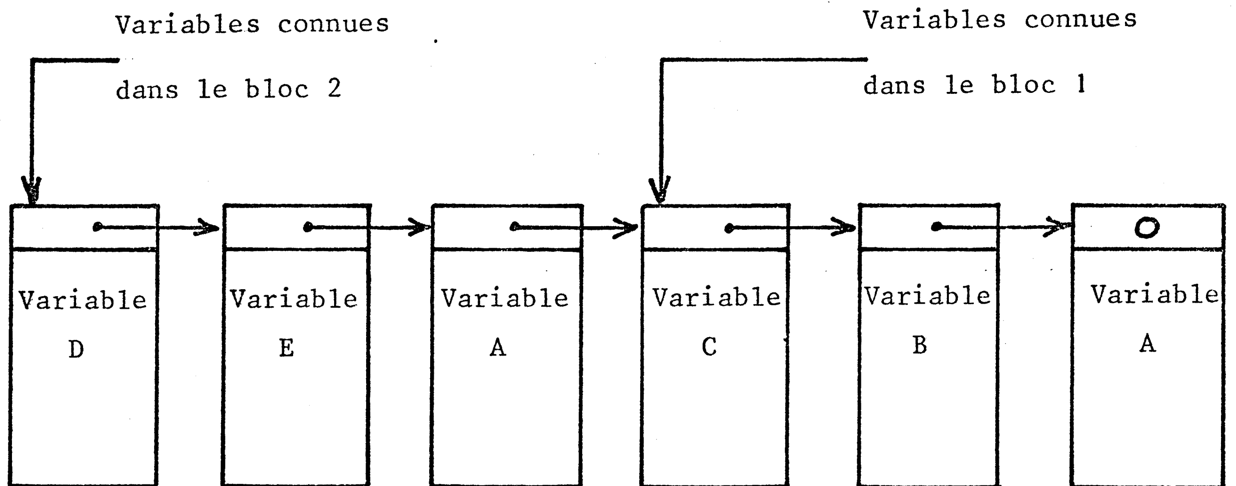


FIGURE III.1.a
La table des symboles d'un programme

Chaque entrée de la table des symboles définit une variable qui appartient à l'une des trois classes suivantes :

- Classe des variables statiques (STATIC)

  Les variables de cette classe sont globales au programme et sont contenues dans une section de contrôle appelée "STATIC INTERNAL CONTROL SECTION".

- Classe des variables automatique (AUTOMATIC)

  Les variables sont allouées automatiquement lors de l'initialisation du bloc où elles sont déclarées. Elles se trouvent dans la DSA (Dynamic Storage Area) associée à ce bloc.

- Classe des variables contrôlées

  Les variables de cette classe sont allouées sur demande du programmeur et seule la dernière génération de chaque variable est accessible. Par contre,dans la table des symboles,il n'existe aucune définition ni pour les variables de classe BASED,ni pour les variables de type AREA,POINTER, LABEL,OFFSET,ni pour les paramètres formels de procédure.

Chaque entrée de la table des symboles contient (Figure III.1.b

- Le lien avec l'entrée suivante.
- Le nom symbolique de la variable précédé de la longueur de l'identificateur.
- Le nombre de dimensions de la variable.
- L'adresse d'un descripteur (DED ou Data Element Descriptor) qui précise quelle est la nature du contenu de la variable (valeur numérique, chaîne de caractères,...).
- Des indicateurs de classe.
- Deux zones (A et B) qui permettent d'adresser la variable.

<table>
<tr><td colspan="2">Adresse de l'entrée suivante</td></tr>
<tr><td>L</td><td></td></tr>
<tr><td colspan="2">Nom de
la variable</td></tr>
<tr><td>D</td><td>Ad - DED</td></tr>
<tr><td>Ind</td><td>Zone A</td></tr>
<tr><td>Zone B</td><td></td></tr>
</table>

L = Longueur de l'identificateur

D = Nombre de dimensions de la variable (0 pour une variable scalaire)

Ind = Indicateur de classe de la variable :
- variable STATIC,
- variable AUTOMATIC ou CONTROLLED non structurée,
- variable AUTOMATIC ou CONTROLLED structurée

FIGURE III.1.b - Format d'une entrée de la table des symboles SYMTAB -

Précisons quels sont les contenus des deux zones A et B en fonction de la classe de la variable :

a- Variable de classe STATIC

Zone B : inutilisée.

Zone A : adresse de la variable ou de son "dope vector". Celui-ci est un descripteur supplémentaire pour affiner la connaissance de la variable (cf §IV-1).

b- Variable de classe AUTOMATIC

Zone B : adresse relative, dans le vecteur des pseudo-registres (PRV), du pseudo-registre qui repère la zone DSA qui contient la variable.

Zone A : -Variable n'appartenant pas à une structure :
Adresse relative, à l'intérieur de la zone DSA, de la variable ou de son descripteur (dope vector).

-Variable appartenant à une structure :
Adresse relative, à l'intérieur de la zone DSA, du descripteur (dope vector) de la variable.

c- Variable de classe CONTROLLED

Zone B : adresse relative, à l'intérieur du vecteur des pseudo-registres, du pseudo-registre qui repère l'allocation courante de la variable.

Zone A : -Variable n'appartenant pas à une structure :
Adresse relative, à l'intérieur de la dernière zone d'allocation, de la variable ou de son descripteur (dope vector).

- Variable appartenant à une structure :
  Adresse relative du descripteur de la variable à l'intérieur du descripteur de la structure.

NOTE : Le descripteur ou "dope vector" d'une variable arithmétique contenue dans une structure est un mot qui contient l'adresse de cette variable arithmétique.

III-2 Accès à la table des symboles

Parmi les instructions-machine correspondant à l'instruction PL/1 PUT DATA, il existe une instruction qui permet de recueillir l'adresse de la table des symboles du bloc.

Pour que l'accompagnateur BUGPLI obtienne cette adresse, nous utilisons la séquence :

```
CALL  BUGPLID;
PUT   DATA;
```

Cette séquence est transformée lors de la compilation en une suite d'instructions-machine donnée par la figure III.2.a.

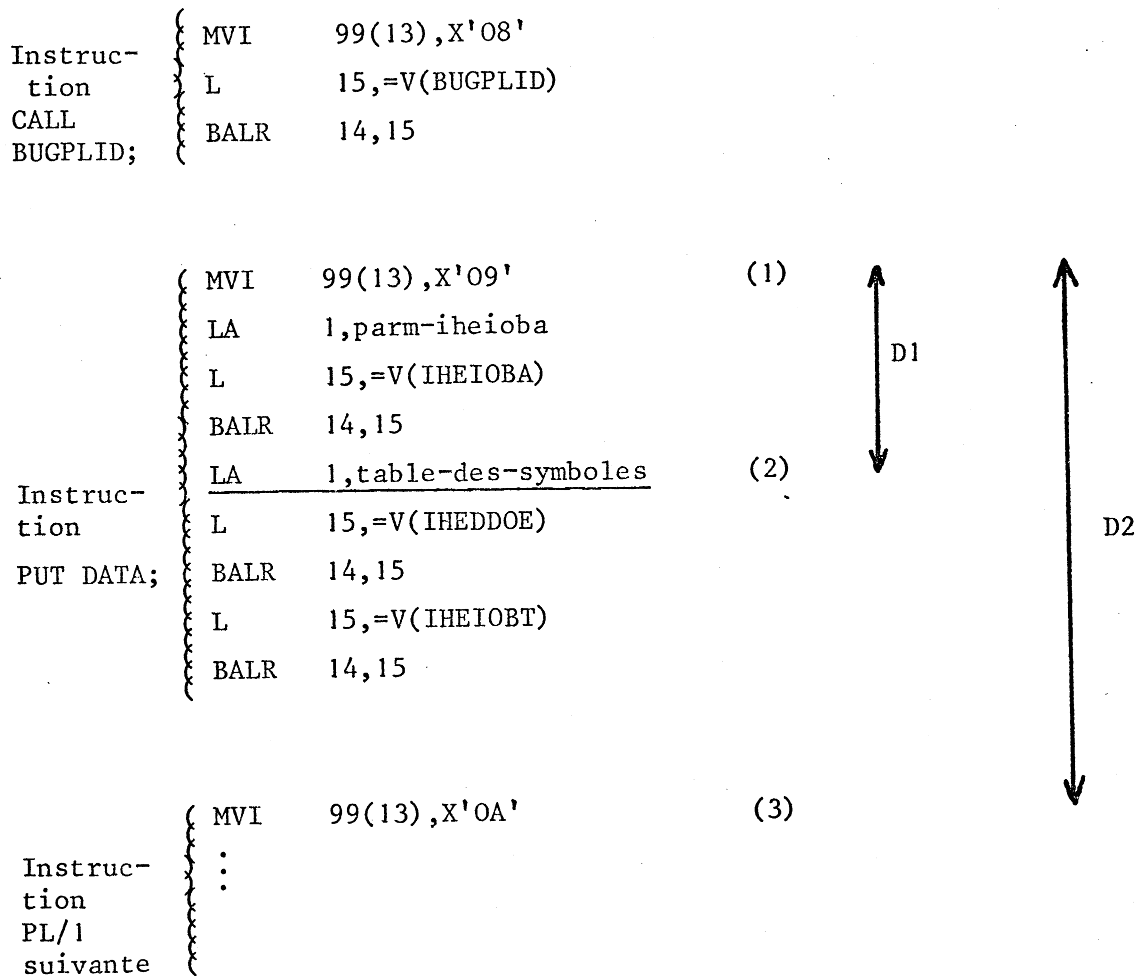


FIGURE III.2.a - Instructions-machine générées par les instructions PL/1: CALL BUGPLID et PUT DATA.

Dans cette séquence :

- IHEIOBA, IHEDDOE, IHEIOBT sont les noms des fonctions de la bibliothèque d'exécution PL/1, qui sont chargées d'initialiser, d'effectuer et de terminer l'impression des contenus de toutes les variables connues dans le bloc.

- Les longueurs D1 et D2 sont fixes et connues.

- L'exécution de l'instruction (2) charge dans le registre 1, l'adresse de la table des symboles.

L'instruction PL/1 CALL BUGPLID donne le contrôle au point d'entrée BUGPLID de l'accompagnateur BUGPLI. A l'entrée de BUGPLID, le registre 14 contient l'adresse de l'instruction (1). Le déplacement D1, déplacement de l'instruction (2) par rapport au début de la séquence générée par PUT DATA, étant fixe, BUGPLI peut donc en exécutant la seule instruction (2), obtenir l'adresse de la table des symboles. Pour éviter des opérations d'entrée-sortie inutiles, l'exécution du PUT DATA ne sera pas effectuée; pour cela, BUGPLI incrémente le contenu du registre 14 de la quantité fixe D2. Le contrôle est donc rendu à l'instruction PL/1 qui suit l'instruction PUT DATA, c'est-à-dire à l'instruction-machine (3).

IV - ACCES AUX VARIABLES - ADRESSAGE

Nous avons vu qu'une variable scalaire de type arithmétique est entièrement décrite par une entrée de la table des symboles et un DED. Les variables indicées, les variables de type chaîne et les variables contenues dans une structure possèdent un descripteur supplémentaire appelé "Dope Vector".

IV-1 Les descripteurs de variables: "Dope Vectors".

Il existe quatre types de descripteurs de variables :

- Descripteur de chaîne: SDV (String Dope Vector),
- Descripteur de tableau: ADV (Array Dope Vector),
- Descripteur de tableau de chaînes: SADV(String Array Dope Vector)
- Descripteur de structure: STDV (Structure Dope Vector).

IV-1-1 Descripteur de chaîne: SDV (Figure IV.1.1.a)

Il décrit une chaîne de caractères ou de bits et contient :

- L'adresse de la chaîne
- Les dimensions maximale et courante de la chaîne. Ces deux dimensions sont identiques pour une chaîne de longueur fixe.

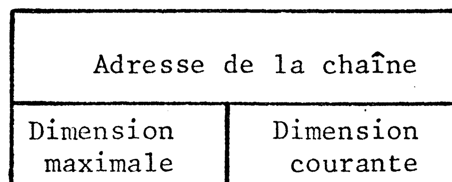


FIGURE IV.1.1.a -Descripteur de chaîne:SDV-

IV-1-2 Descripteur de tableau: ADV (Figure IV.1.2.a).

Il décrit un tableau de variables de type arithmétique et contient :

- L'adresse de l'origine virtuelle du tableau c'est-à-dire l'adresse de l'élément (qui n'existe peut-être pas) dont les indices sont tous nuls.

- Une zone de multiplicateurs qui permettent d'optimiser le calcul de l'adresse d'un élément du tableau. Il existe un multiplicateur pour chaque dimension du tableau.

- Une zone de paires de bornes qui permettent de détecter les dépassements de tableau par indices erronés. Pour chaque dimension, une paire de bornes contient les bornes supérieure et inférieure relatives à cette dimension.

Adresse origine virtuelle	
multiplicateur 1	
⋮	
multiplicateur n	
Borne supérieure 1	Borne Inférieure 1
⋮	
Borne supérieure n	Borne inférieure n

FIGURE IV.1.2.a - Descripteur de tableau: ADV -

Un multiplicateur est évalué à l'aide de la formule

$$M_i = \ell \times \prod_{r=i+1}^{n} (BS_r - BI_r + 1)$$

pour i = 2 à n

et $$M_1 = \ell$$

où M_i est le multiplicateur de rang i,
ℓ la longueur d'un élément du tableau,
BS_r la borne supérieure de la $r^{ième}$ dimension,
BI_r la borne inférieure de la $r^{ième}$ dimension.

L'adresse d'un élément du tableau est alors :

$$\text{adresse} = O.V + \sum_{i=1}^{n} IND_i \times M_i$$

où O.V est l'adresse de l'origine virtuelle,
IND_i la valeur du $i^{ème}$ indice,
M_i le ième multiplicateur.

IV-1-3 Descripteur de tableau de chaînes: SADV (Figure IV.1.3.a)

Il décrit un tableau de variables de type chaîne de caractères ou chaîne de bits.

Il est formé par la concaténation d'un descripteur de tableau (ADV) et d'un mot contenant les dimensions maximale et courante d'un élément du tableau (tous les éléments du tableau ont la même longueur).

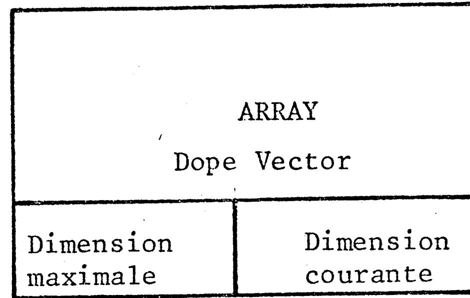


FIGURE IV.1.3.a - Descripteur de tableau de chaînes: SADV

IV-1-4 Descripteur de structure: STDV

Il décrit une variable structurée de classe AUTOMATIC ou CONTROLLED.

Il est formé par la concaténation des différents descripteurs associés à chaque élément de la structure.

IV-2 Adressage d'une variable

Pour accéder au contenu d'une variable dont on fournit le nom entièrement qualifié, BUGPLI doit nécessairement rechercher dans la table des symboles, l'entrée qui définit cette variable. Les zones A et B de cette entrée permettent de calculer l'adresse de la variable.

Le calcul est différent suivant la classe de la variable :

- La zone A donne directement l'adresse de la variable ou de son descripteur, si cette variable est de classe STATIC.

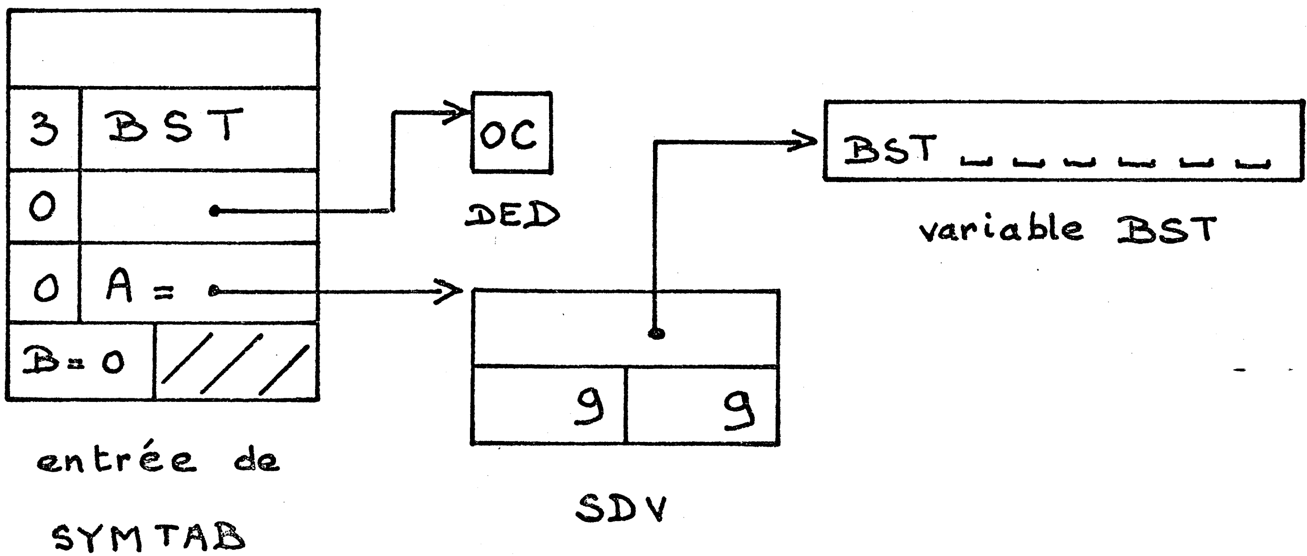
- Pour les variables de classe AUTOMATIC ou CONTROLLED, il faut d'abord accéder au pseudo-registre dont le déplacement est égal au contenu de

la zone B; l'adresse de la variable ou de son descripteur est ensuite obtenue par addition des contenus de ce pseudo-registre et de la zone A.

Illustrons à l'aide de quelques exemples, le calcul de l'adresse d'une variable.

IV-2-1 Variable de type caractère et de classe STATIC

```
DCL      BST      CHAR(9)    STATIC;
BST='BST';
```



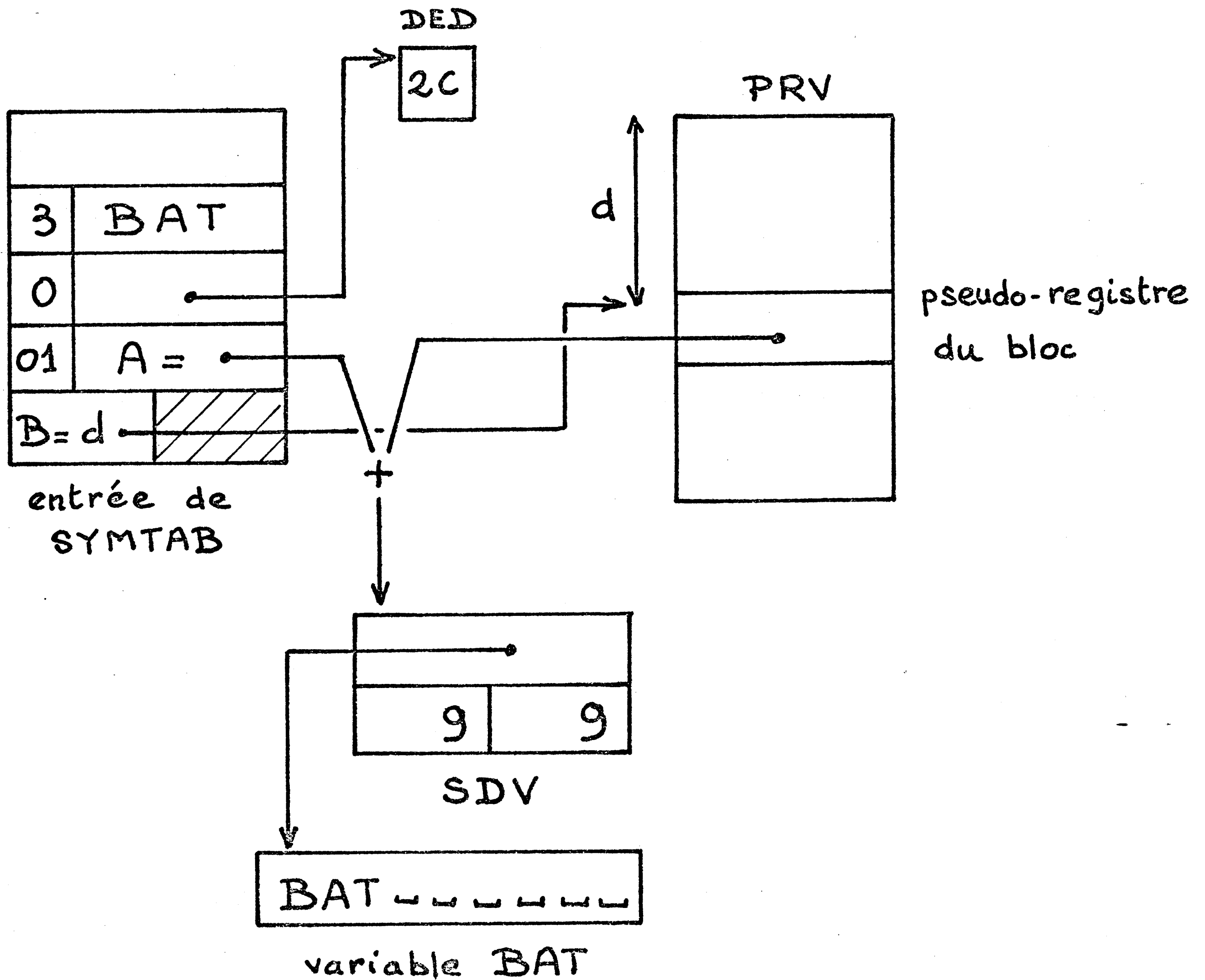
zone A = adresse du SDV

zone B = 0

La zone A de l'entrée de la table des symboles contient l'adresse du descripteur de la variable; le descripteur contient l'adresse de la variable.

IV-2-2 Variable de type caractère et de classe AUTOMATIC.

```
DCL  BAT   CHAR(9);
BAT='BAT';
```



Zone A = adresse relative du SDV

Zone B = adresse relative du pseudo-registre.

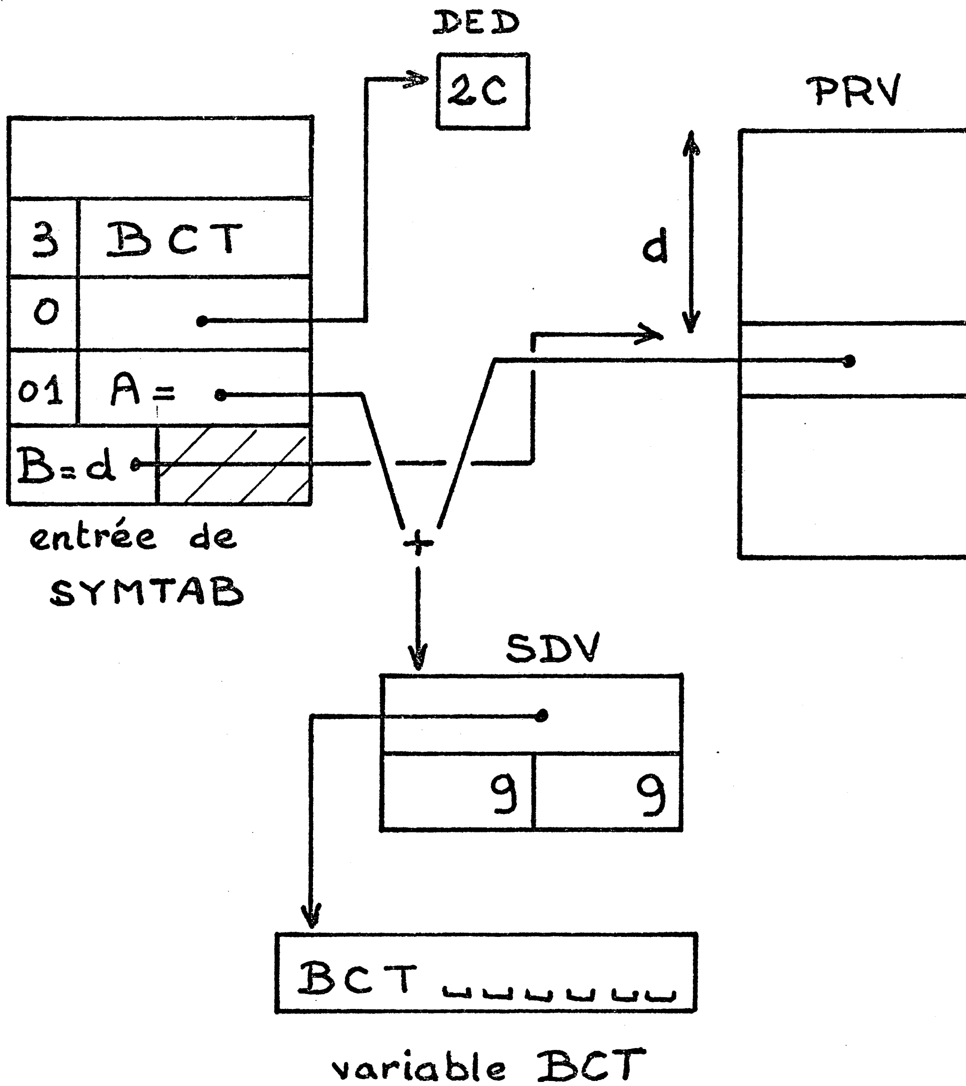
La zone B contient l'adresse relative du pseudo-registre associé au bloc contenant la variable. Ce pseudo-registre repère la DSA qui contient la variable et son descripteur (SDV).

La zone A fournit l'adresse relative du descripteur de la variable. L'adresse absolue de ce descripteur est obtenue par addition des contenus du pseudo-registre du bloc et de la zone A.

Le descripteur de la variable fournit ensuite l'adresse de la variable.

IV-2-3 Variable de type caractère et de classe CONTROLLED

```
DCL      BCT    CHAR(9)    CTL;
ALLOCATE        BCT;
BCT='BCT';
```



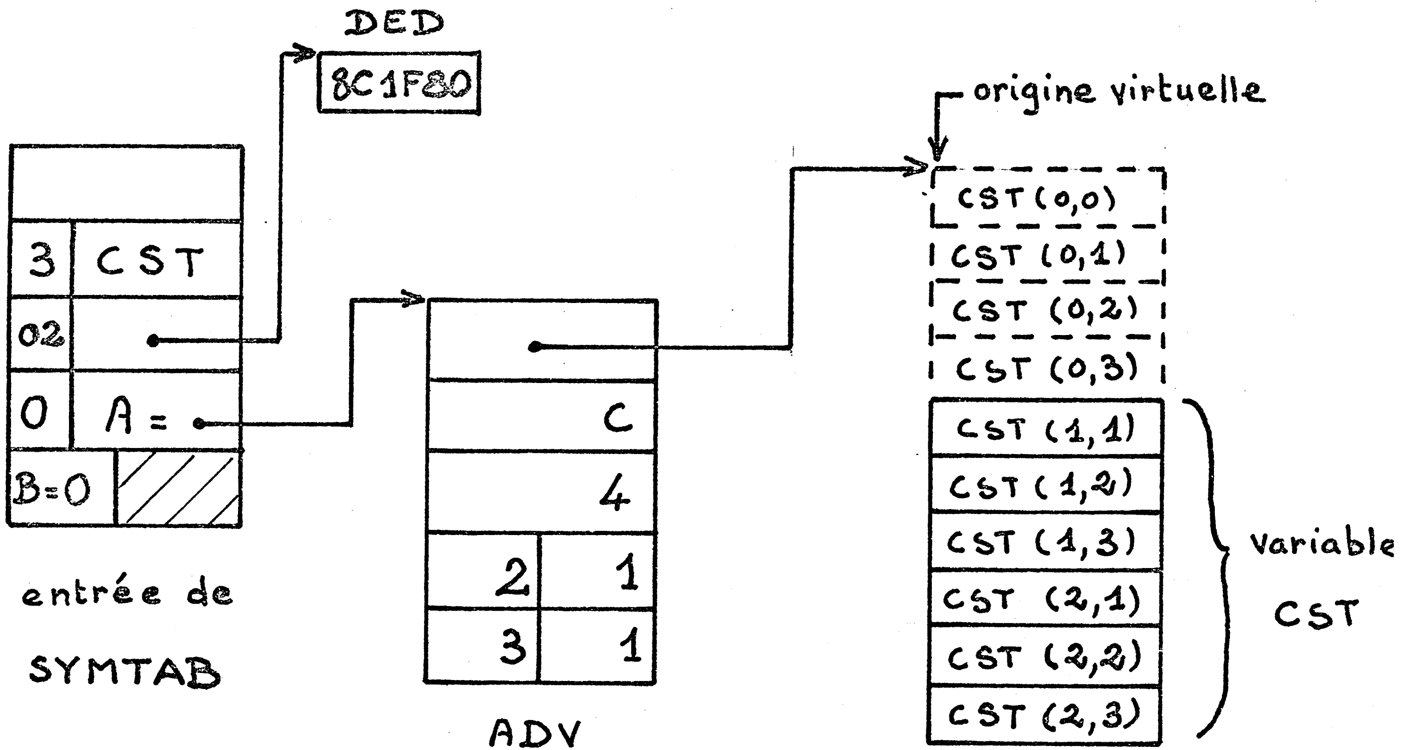
Zone A = adresse relative du SDV

Zone B = adresse relative du pseudo-registre.

Le calcul d'adresse est similaire à celui présenté au paragraphe IV-2-2. La seule différence est que le pseudo-registre adressé à l'aide de la zone B, ne repère pas une DSA mais la dernière zone d'allocation de la variable.

IV-2-4 Tableau de variables de type arithmétique et de classe STATIC.

DCL CST (2,3) BIN FIXED(31,0) STATIC;



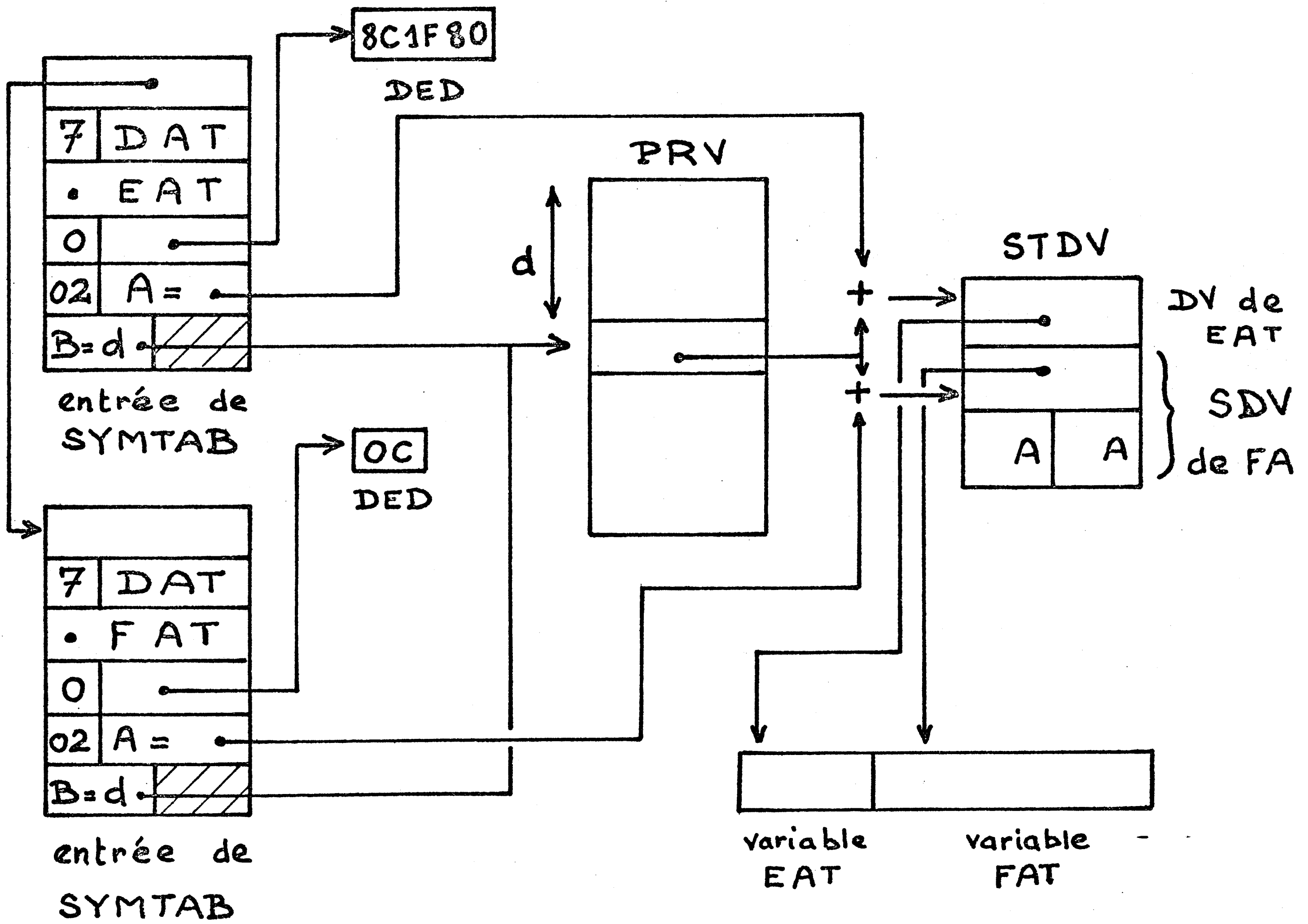
Zone A = adresse du ADV

Zone B = 0

La zone A repère le descripteur (ADV) de la variable; le descripteur contient l'adresse de l'origine virtuelle du tableau. L'adresse d'un élément du tableau est obtenue en ajoutant à l'adresse de l'origine virtuelle la zomme des produits des indices de l'élément par les multiplicateurs.

IV-2-5 Variable structurée de classe AUTOMATIC.

```
DCL 1 DAT,
      2 EAT  BIN FIXED(31,0) ;
      2 FAT  CHAR(10);
```



Zone A = adresse relative du Dope Vector

Zone B = adresse relative du pseudo-registre

Les zones B des entrées de la table des symboles qui définissent les variables de la structure, contiennent toutes l'adresse relative du pseudo-registre associé au bloc qui contient la structure. Ce pseudo-registre repère la DSA qui contient le descripteur de la structure (STDV) et les variables qui forment la structure.

La zone A d'une entrée définissant une variable de la structure contient l'adresse relative du descripteur de la variable; l'adresse absolue de ce descripteur est obtenue par addition des contenus du pseudo-registre du bloc et de la zone A.

Le descripteur de la variable fournit ensuite l'adresse de la variable.

Remarquons que le descripteur de la variable scalaire arithmétique DAT.EAT est un mot qui contient l'adresse de cette variable.

V - NOTION DE BLOC PL/1

Un bloc PL/1 est délimité par une instruction PROCEDURE ou BEGIN et une instruction END.

Lors de l'exécution, un bloc actif est matérialisé par une zone DSA (Dynamic Storage Area) à laquelle peut être attachée une zone VDA (Variable Data Area). Une VDA est créée pour contenir les variables de classe AUTOMATIC de longueur inconnue à la compilation; une DSA contient les autres variables AUTOMATIC définies dans le bloc.

Plus précisément, une DSA contient :

- Une zone de sauvegarde de registres.
- L'identification interne du bloc ou "Invocation Count". La 1ère DSA contient la valeur 1, la 2ième DSA la valeur 2, etc.. Cette identification est utilisée pour l'exécution d'une instruction GOTO relative soit à une variable de type LABEL, soit à une étiquette définie dans une procédure ayant l'attribut récursif.
- Une zone nommée DISPLAY, utilisée pour tous les blocs contenus dans une procédure récursive.
- Une zone qui contient à chaque instant, le numéro de la dernière instruction PL/1 exécutée dans le bloc.
- Les variables AUTOMATIC de longueur connue lors de la compilation, les zones de travail, les descripteurs des variables, les listes de paramètres.

REMARQUE

> Pour une meilleure compréhension, nous allons tout d'abord exposer différentes notions qui ont trait aux blocs du langage PL/1; nous présenterons après, au paragraphe V-5, l'intéraction que nous avons définie entre l'accompagnateur BUGPLI et le mécanisme de gestions des blocs.

V-1 Chaînage de blocs: la liste des DSA et VDA

Les DSA et les VDA forment une liste dont le premier élément est la zone de sauvegarde externe; la fin de la liste est repérée par le pseudo-registre IHEQSLA. Les DSA possèdent un double chaînage (pointeurs avant et arrière) alors que les VDA n'ont qu'un seul pointeur de chaînage.

Supposons un enchaînement de programmes tels que :

- Une procédure principale P appelle une procédure Q.
- Un bloc, interne à la procédure Q, contient la déclaration d'un tableau ayant une borne variable.

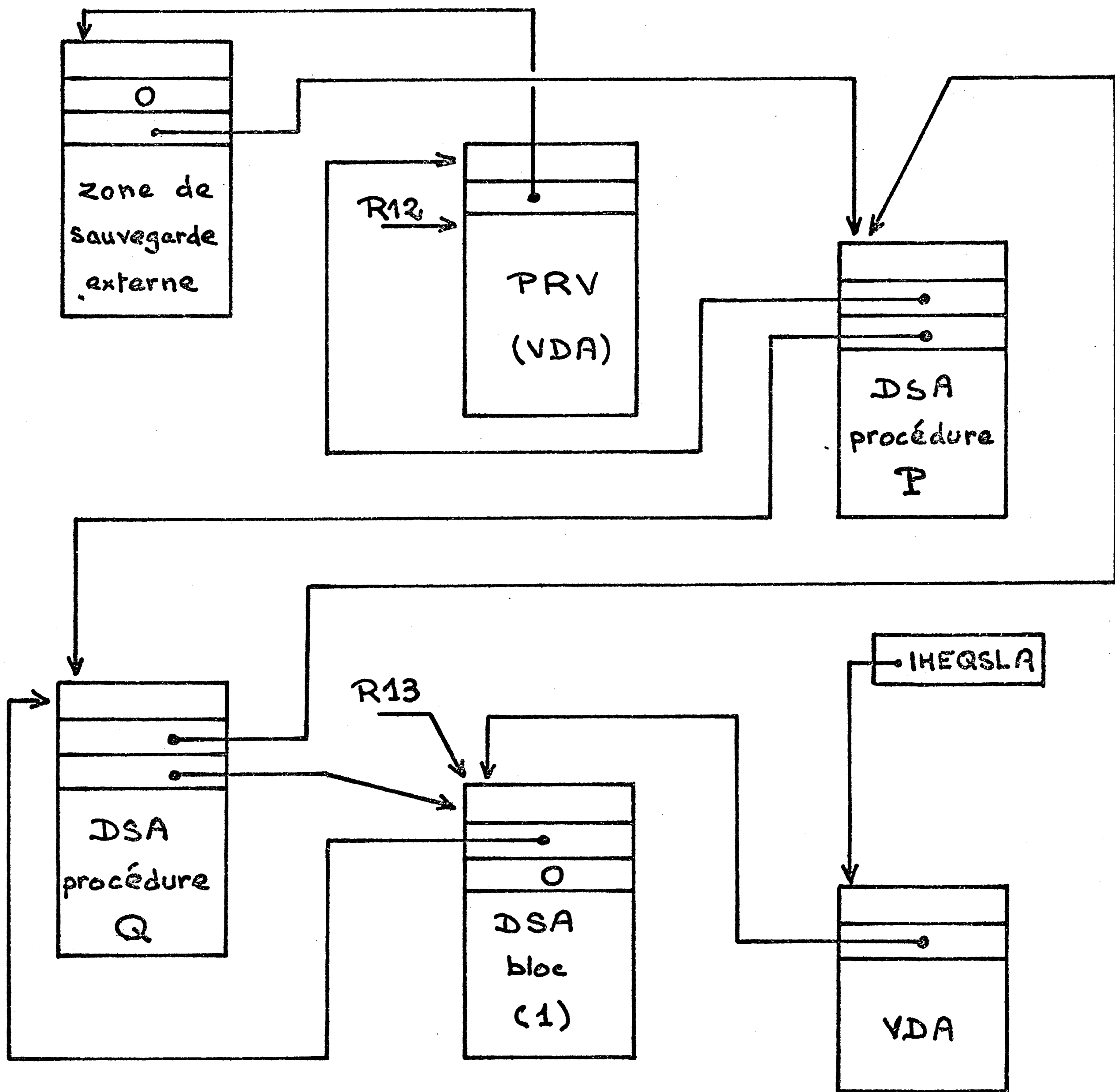
Soit par exemple :

```
P:  PROC   OPTIONS(MAIN);
 DCL (I,J)   FIXED   BIN;
 :
 CALL Q;
 :
 END  P;


Q:  PROC;
  DCL  N FIXED BIN;
  N=10;
  :
bloc (1)  ┌── BEGIN;
          │     DCL   T(N)  FIXED   BIN;
          │     :
          └──── END;      <------ instruction (2)
                :
  L: ......
                :
  END     Q;
```

Lorsqu'à l'exécution, le bloc interne (1) devient actif, la liste des DSA et des VDA est montrée par la figure V.1.a. Le bloc interne (1) est représenté physiquement par une DSA et par une VDA qui est repérée par le pseudo-registre IHEQSLA.

L'exécution de l'instruction END (2) qui ferme le bloc interne (1), provoque la suppression des deux derniers éléments de la liste (figure V.1.b). Le pseudo-registre IHEQSLA repère alors la DSA associée à la procédure Q.



Le registre général 12 (R12) repère le vecteur des pseudo-registres; le registre général 13 (R13) repère la zone courante de sauvegarde des registres.

FIGURE V.1.a - DSA et VDA après initialisation du bloc (1).

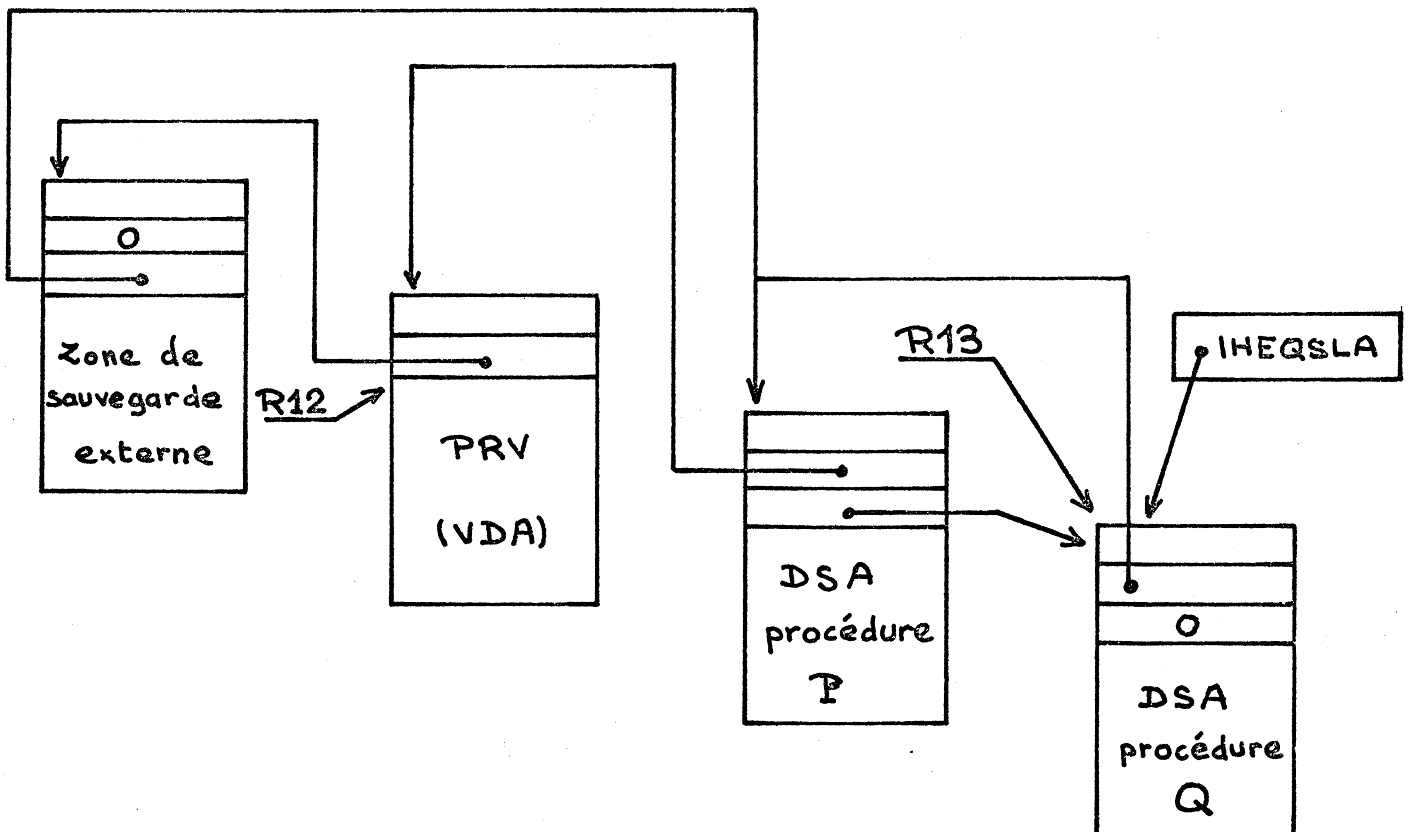


FIGURE V.1.b - DSA et VDA après fermeture du bloc (1).

V-2 Initialisation d'un bloc

Lors de l'initialisation d'un bloc, la fonction IHESADA de la bibliothèque d'exécution PL/1 acquiert dynamiquement une zone de mémoire pour y construire la DSA associée au bloc. De plus, si le bloc est le premier bloc de la procédure ayant l'attribut MAIN, il y a au préalable, acquisition d'une zone VDA pour construire le vecteur des pseudo-registres.

V-3 Fermeture d'un bloc

La "fermeture" d'un bloc est provoquée par l'exécution d'une instruction END, RETURN ou d'une instruction GOTO qui donne le contrôle à une instruction contenue dans un bloc englobant le bloc que l'on quitte. La conséquence de cette "fermeture" est la libération d'une ou plusieurs zones DSA ou VDA par les fonctions IHESAFA (instruction END), IHESAFB (instruction RETURN) ou IHESAFC (instruction GOTO) de la bibliothèque d'exécution.

Soit par exemple :

```
P:   PROC;
 DCL   LAB  LABEL;
 .
 .
 .
 LAB=L;
 CALL   Q(LAB);
 .
 .
 .
 L:
 .
 .
 .
 END   P;
```

```
Q:  PROC (ETIQ);
DCL  ETIQ   LABEL;
:
  BEGIN;
    :
    GOTO  ETIQ;       <---------- instruction (1)
    :
  END;
  :
  END  Q;
```

L'exécution de l'instruction (1) GOTO ETIQ permet de quitter la procédure Q et de revenir dans la procédure P, à l'instruction étiquetée L.

La fonction IHESAFC, chargée d'exécuter l'instruction GOTO, doit connaître le nombre de zones (DSA ou VDA) à libérer et l'adresse de l'instruction à laquelle elle doit donner le contrôle. Ces informations sont contenues dans la variable de type LABEL nommée LAB.

De manière générale, une variable de type LABEL contient :

- L'adresse de l'instruction repérée par l'étiquette qui est affectée à la variable.

- L'identification interne du bloc qui contient cette instruction étiquetée. Cette identification est placée dans la variable de type LABEL, par le prologue du bloc, lors de l'initialisation de celui-ci, alors que l'adresse de l'instruction est generée lors de la compilation.

V-4 Blocs contenus dans une procédure récursive

Lors de chaque initialisation d'un bloc contenu dans une procédure récursive, les variables de classe AUTOMATIC déclarées dans le bloc sont allouées: une nouvelle génération de ces variables est ainsi créée. Ouvrir (ou fermer) un bloc contenu dans une procédure récursive correspond donc à placer en tête d'une pile d'exécution (ou à retirer de cette pile) l'ensemble des variables de classe AUTOMATIC définies dans ce bloc.

Nous savons que les variables de classe AUTOMATIC sont contenues dans une zone DSA. D'autre part, il existe pour chaque bloc défini à la compilation, un pseudo-registre appelé DISPLAY, qui est utilisé pour adresser les variables du bloc.

Dans le cas d'un bloc contenu dans une procédure récursive, ce pseudo-registre contient à tout instant l'adresse de la zone DSA associée à la dernière activation du bloc.

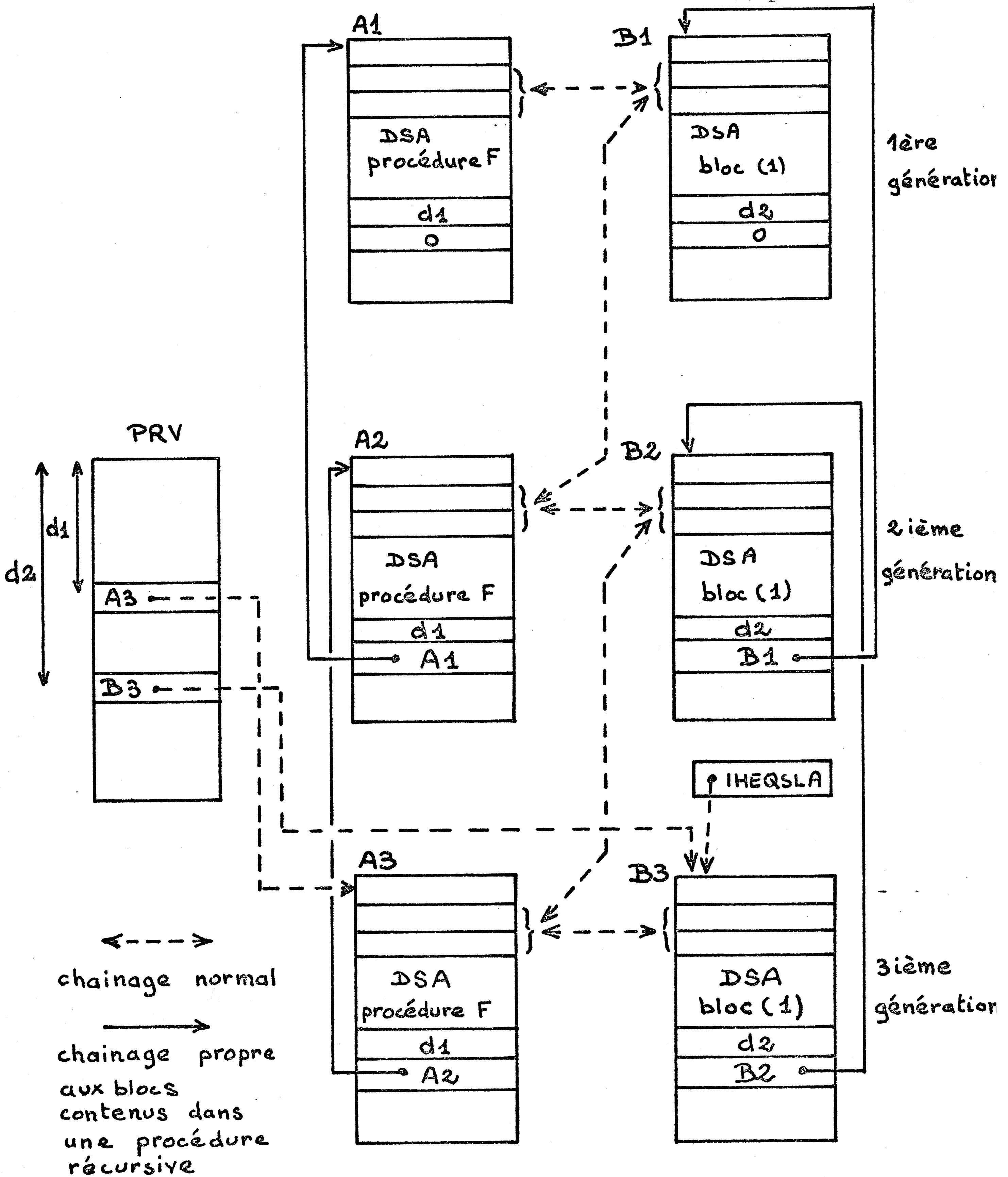
Aussi, pour reprendre l'exécution d'un niveau inférieur de récursion, lors de chaque sortie d'un tel bloc, il est nécessaire de mettre à jour le pseudo-registre associé à ce bloc. Pour cela, les informations nécessaires sont conservées dans chaque zone DSA associée à un bloc contenu dans une procédure récursive et sont formées de l'adresse relative du pseudo-registre dans le vecteur des pseudo-registres (PRV) et de la valeur à placer dans ce pseudo-registre.

Soit par exemple la procédure récursive :

```
               F:   PROC (N)   RECURS;
               DCL     N  BIN  FIXED;
                IF  N=0   THEN   RETURN;
bloc (1)  ....BEGIN;
          :      DCL   M   BIN   FIXED;
          :      M=N-1;
          :      CALL   F(M);
          :...... END;
                END    F;
```

Après le troisième appel de cette procédure, nous avons trois générations pour chacun des deux blocs définis. Les zones DSA associées à ces blocs sont alors liées comme le montre la figure V.4.a.

Dans cette figure nous avons fait figurer en pointillé, le chaînage normal tel qu'il a été présenté au paragraphe V-1; en trait plein, le chaînage propre aux blocs contenus dans une procédure récursive. Le pseudo-registre d'adresse relative d1 repère la DSA associée à la troisième activation de la procédure P; le pseudo-registre d'adresse relative d2 repère la DSA associée à la troisième activation du bloc (1). Chaque DSA associée soit à la procédure F, soit au bloc (1) contient les informations nécessaires pour accéder à la DSA associée à la génération précédente de la procédure ou du bloc.



<u>FIGURE V.4.a</u> - DSA associées à des blocs participant à une récursion -

V-5 Intéraction entre l'accompagnateur BUGPLI et le mécanisme de gestion des blocs PL/1.

L'accompagnateur doit être averti, par les fonctions de la bibliothèque qui gèrent les blocs (ouverture et fermeture), du cheminement du contrôle dans le programme. Ceci est nécessaire d'une part pour avoir accès aux variables connues dans le bloc en cours, d'autre part pour satisfaire les requêtes GOTO de BUGPLI.

BUGPLI doit aussi réaliser une association entre l'identification interne d'un bloc (Invocation Count) et son identification externe fournie par l'utilisateur.

Nous avons vu que la séquence :

```
CALL    BUGPLID;
PUT     DATA;
GET     DATE;
```

est placée automatiquement en tête de chaque bloc PL/1 et permet à BUGPLI de connaître l'adresse de la table des symboles définis dans le bloc.
A l'aide de cette séquence, nous définissons l'identification externe d'un bloc comme le numéro de l'instruction CALL BUGPLID qui est en tête de ce bloc. Ce numéro d'instruction permet de localiser le bloc à l'intérieur d'une procédure, la procédure étant localisée par le nom de la procédure externe qui la contient.

Les informations ainsi recueillies par l'accompagnateur sont conservées dans la pile de sauvegarde des blocs qui décrit les blocs actifs du programme.

V-6 La pile de sauvegarde des blocs (Figure V.6.a).

Elle est construite par BUGPLI et recueille des informations sur les blocs actifs, pour connaître le cheminement de l'exécution à l'intérieur du programme. Elle constitue en fait, le pivot central de BUGPLI pour lui permettre de traiter tout ce qui se rapporte aux blocs d'un programme PL/1. Cette pile est indispensable à l'accompagnateur pour satisfaire les requêtes telles que ORG, GOTO et RECURS.

Une entrée de la pile est formée de six mots et contient :

- l'adresse de la table des symboles du bloc,
- l'adresse de la zone DSA associée au bloc,
- l'identification interne (Invocation Count) du bloc,
- l'identification externe du bloc,
- le nom de la procédure externe qui contient le bloc.

La pile de sauvegarde des blocs, constituée de 16 parties acquises et libérées dynamiquement, est décrite par une table de 16 entrées; une entrée de cette table contient l'adresse de la partie correspondante de la pile, si cette partie existe.

Chaque partie de la pile contient 256 entrées; l'ensemble de la pile permet donc d'avoir 4096 blocs actifs simultanément (Figure V.6.b.).

Adresse de SYMTAB	Adresse DSA	Identification interne	Identification externe	Nom de procédure externe

FIGURE V.6.a - Une entrée de la pile de sauvegarde des blocs -

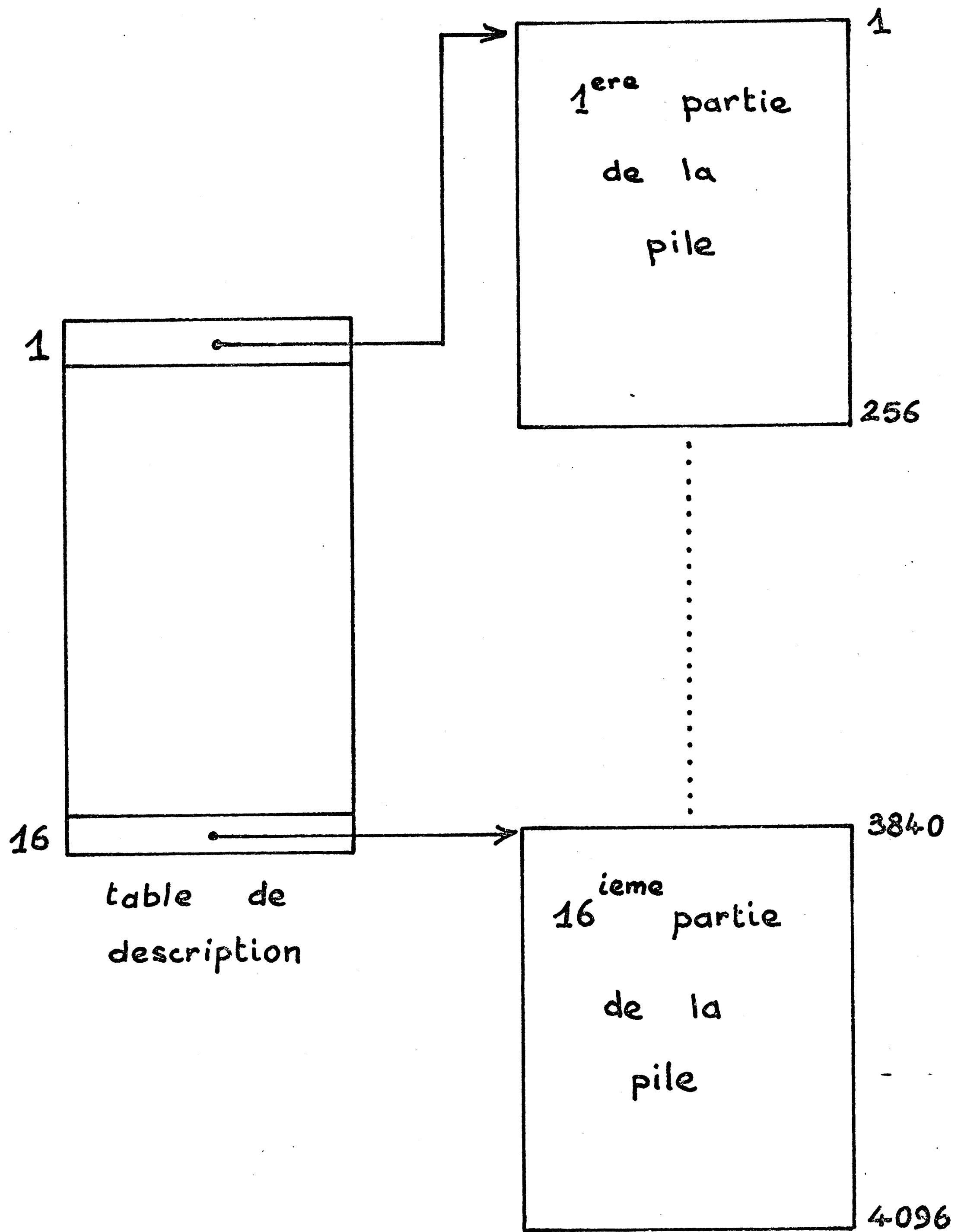


FIGURE V.6.b - Structure de la pile de sauvegarde des blocs -

V-7 Traitement de la pile de sauvegarde des blocs par l'accompagnateur d'exécution.

Pour optimiser le traitement de la pile, deux constantes locales à BUGPLI - AENTCOUR et BLOCOUR contiennent respectivement l'adresse et le numéro de l'entrée décrivant le dernier bloc actif.

L'accompagnateur est appelé par le module IHESAP qui contient les fonctions qui gèrent les blocs PL/1 :

- Lors de l'acquisition d'une zone DSA (ouverture d'un bloc), BUGPLI sauvegarde dans la pile, l'adresse de la DSA et l'identification interne du bloc.

- Lors de la libération d'une zone DSA (fermeture d'un bloc), BUGPLI invalide la dernière entrée significative de la pile; ceci est effectué en mettant à jour les deux constantes locales AENTCOUR et BLOCOUR.

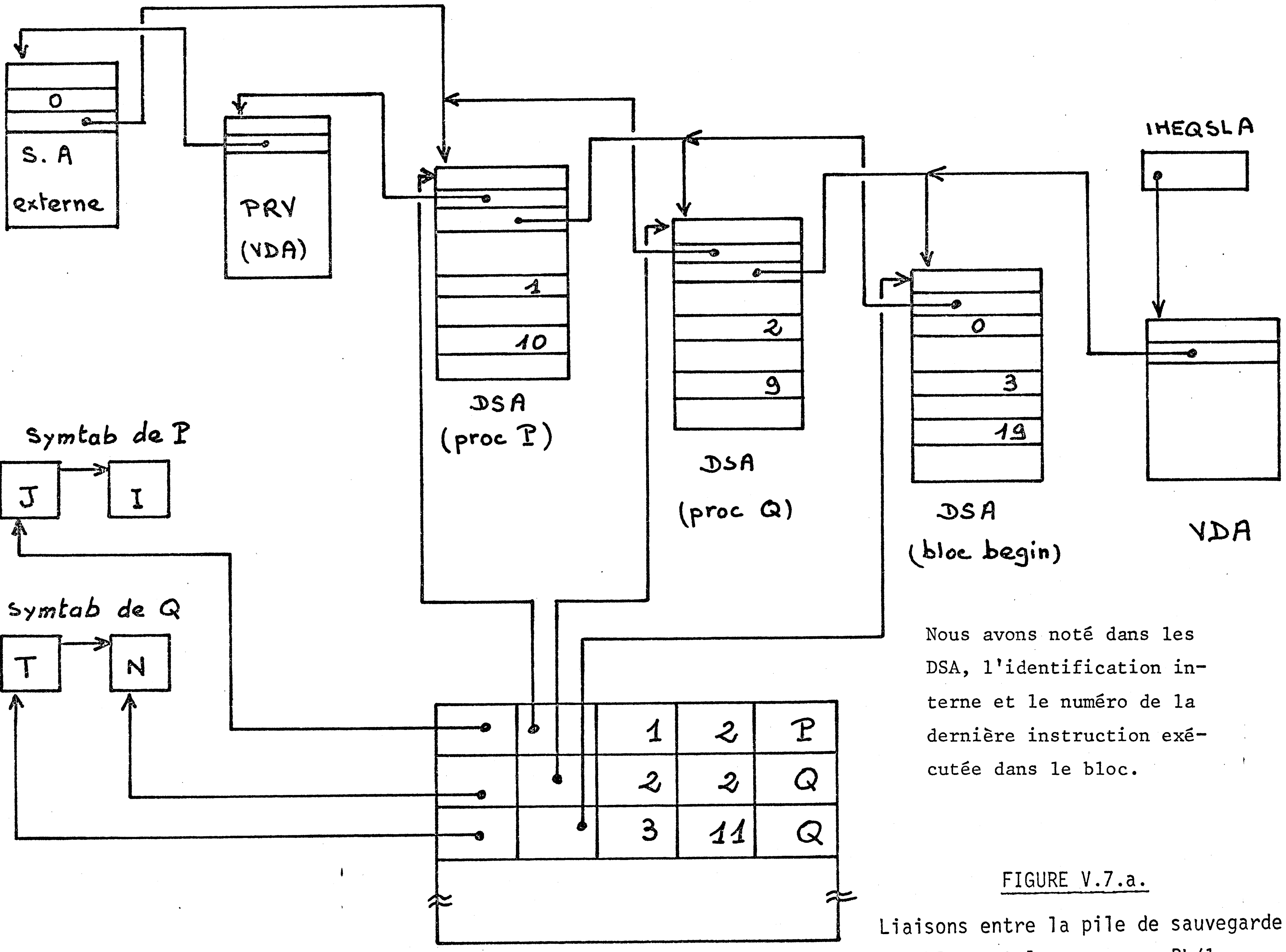
Pour actualiser les autres zones d'une entrée de la pile, l'accompagnateur utilise la première instruction exécutée dans un bloc; celle-ci est l'instruction CALL BUGPLID. Elle permet à BUGPLI de placer dans la pile, l'adresse de la table des symboles du bloc, et l'identification externe du bloc, puis après recherche, le nom de la procédure externe qui contient ce bloc (cf paragraphe VI.2).

Pour illustrer ce traitement, reprenons l'exemple que nous avons utilisé au paragraphe V.1. Après compilation du programme avec l'option DEBUG, nous obtenons deux listes telles que :

```
1    P:   PROC     OPTIONS(MAIN);
2    CALL  BUGPLID;   PUT DATA;  GET   DATA;
5    DCL  (I,J)    FIXED   BIN;
     :
10   CALL   Q;
     :
20   END    P;
```

```
1    Q:  PROC;
2    CALL  BUGPLID;  PUT DATA; GET  DATA;
5    DCL   N   FIXED   BIN;
6    N = 10;
     :
10   BEGIN;
11   CALL BUGPLID; PUT DATA;  GET DATA;
14   DCL     T(N)   FIXED   BIN;
     :
20   GOTO   L;
     :
30   END;
     :
40   L:
     :
50   END   Q;
```

Après l'activation du bloc interne de la procédure Q, nous avons en mémoire la structure donnée par la figure V.7.a. Il existe trois blocs PL/1 actifs et par suite trois entrées valides dans la pile. La pile de sauvegarde nous permet de savoir que :



Nous avons noté dans les DSA, l'identification interne et le numéro de la dernière instruction exécutée dans le bloc.

FIGURE V.7.a.

Liaisons entre la pile de sauvegarde des blocs et le programme PL/1.

- Le premier bloc PL/1 qui a été activé appartient à la procédure P et a pour identifications interne 1 et externe 2; les variables connues dans ce bloc sont J et I; en consultant la DSA associée, nous savons que la dernière instruction exécutée dans ce bloc a pour numéro 10.

- Le second bloc PL/1 activé appartient à la procédure Q et a pour identifications interne 2 et externe 2; l'unique variable connue dans ce bloc est N; la dernière instruction exécutée est celle de numéro 9.

- Le troisième et dernier bloc PL/1 actif appartient aussi à la procédure P et a pour identifications interne 3 et externe 11; les variables connues dans ce bloc sont T et N; la dernière instruction exécutée dans ce bloc est, par exemple, l'instruction de numéro 19.

V-8 Que permet la pile de sauvegarde ?

L'accompagnateur va utiliser la pile de sauvegarde pour satisfaire les requêtes de mise au point. Ainsi, les deux informations -identification externe de bloc et nom de procédure externe- correspondent aux paramètres de la requête ORG qui autorise l'accès aux variables d'un bloc englobant le bloc courant. L'entrée de la pile qui contient l'identification du bloc fournit l'adresse de la table des symboles du bloc, donc l'accès aux noms des variables, et l'adresse de la zone DSA associée au bloc, donc l'accès au contenu des variables.

La présence de l'identification interne d'un bloc permet de satisfaire les requêtes GOTO. En effet, l'exécution d'une requête GOTO est identique à l'exécution d'une instruction GOTO de PL/1. Le transfert est réalisé à l'aide de la fonction IHESAFC de la bibliothèque d'exécution, qui admet comme paramètres l'adresse de l'instruction sur laquelle on veut se transférer et l'identification interne du bloc qui contient cette instruction.

Pour cela, le deuxième paramètre de la requête GOTO (identification externe du bloc) et le paramètre de la requête QUALIFY (nom de la procédure externe qui contient ce bloc) permettant, après sélection de l'entrée de la pile qui décrit le bloc, d'obtenir l'identification interne du bloc.

Le numéro d'instruction, premier paramètre de la requête GOTO, est transformé en adresse relative à l'aide d'un fichier de type STPLI, puis en adresse absolue par addition de l'adresse d'implantation de la procédure externe dont le nom est le paramètre de QUALIFY.

VI - IDENTIFICATION DE PROCEDURE

Pour répondre aux besoins des requêtes QUALIFY, l'accompagnateur d'exécution doit reconnaître à quelle procédure s'applique la requête. Il va utiliser, pour ce faire, la structure même d'une procédure PL/1 qui se compose de trois parties :

- Le prologue qui est une suite de constantes et d'instructions pour initialiser la procédure,
- La procédure elle-même,
- L'épilogue qui est une suite d'instructions destinée à désactiver la procédure.

Pour l'accompagnateur, seul le prologue doit être connu puisque c'est par son intermédiaire qu'il pourra satisfaire les requêtes qui portent sur la procédure.

VI-1 Prologue de procédure (Figure VI.1.a).

Cette suite de constantes et d'instructions contient le nom de la procédure, la longueur de la zone DSA qu'il contient de lui associer, les instructions de sauvegarde des registres et d'acquisition de la zone DSA.

Les registres sont placés soit dans la zone de sauvegarde externe, soit dans la zone DSA précédente. Une particularité des conventions d'appel de la procédure fait que chaque DSA précédant la DSA associée à une procédure, repère le nom de cette procédure: ceci provient de la sauvegarde du registre général 15 qui contient l'adresse du point d'entrée de la procédure.

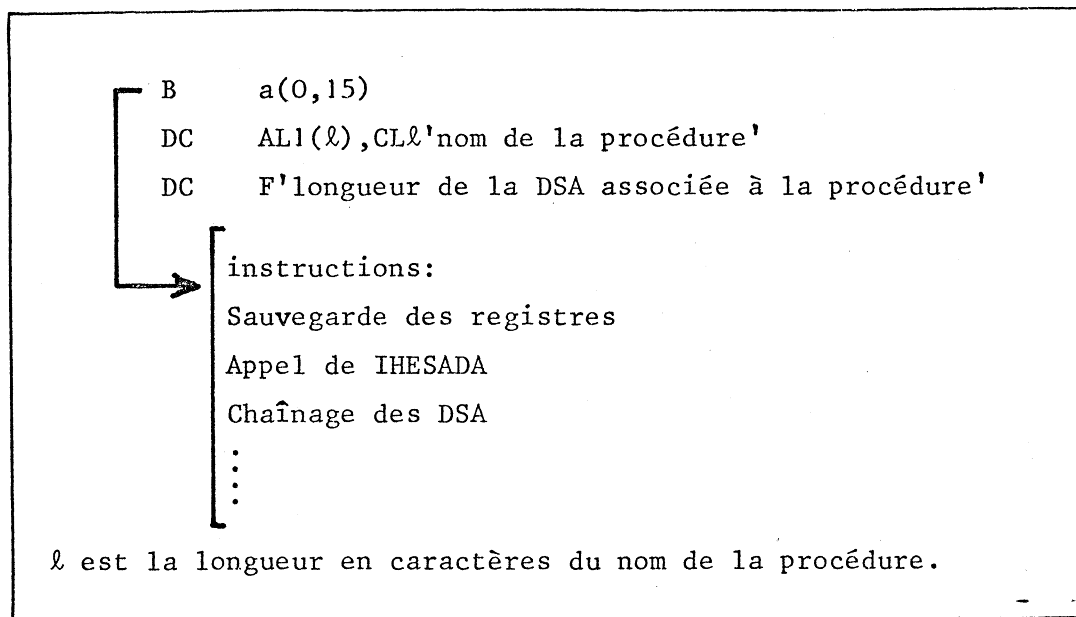


FIGURE VI.1.a - Prologue de procédure en son point d'entrée principal -

Une procédure peut avoir, en plus de son point d'entrée principal, un ou plusieurs points d'entrée secondaire. Le prologue d'une procédure en un point d'entrée secondaire (figure VI.1.b) contient le nom de ce point d'entrée, l'adresse du point d'entrée principal, les instructions de sauvegarde des registres et d'exécution du prologue de la procédure en son point d'entrée principal.

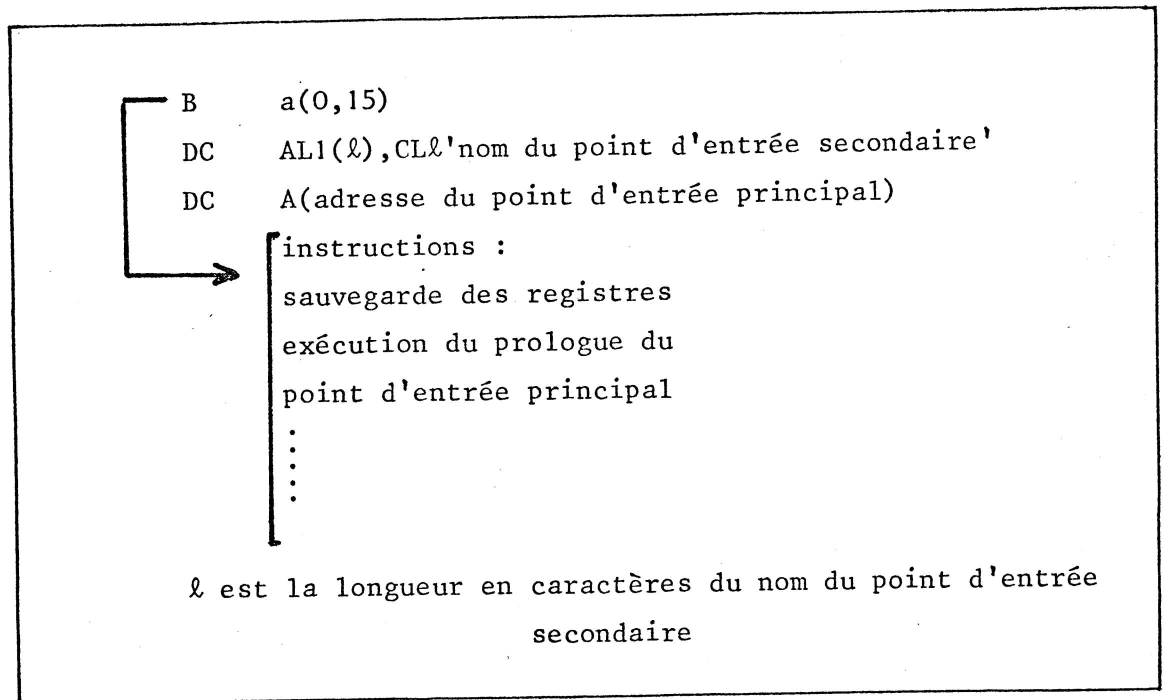


FIGURE VI.1.b -Prologue de procédure en un point d'entrée secondaire-

Connaissant ces deux sortes de prologue, l'accompagnateur d'exécution analyse la séquence des instructions-machine qui forment un prologue et reconnait quel est le type du point d'entrée (point d'entrée principal ou point d'entrée secondaire) et quelle est la nature de la procédure (procédure interne ou procédure externe).

De plus, grâce aux prologues des procédures, BUGPLI peut connaître à tout instant, les noms des procédures PL/1 actives.

VI-2 Recherche par BUGPLI du nom de la dernière procédure externe activée.

Cette recherche est effectuée lors de l'initialisation d'une entrée de la pile de sauvegarde des blocs et, lors du traitement de la requête STOP dont la réponse comporte un nom de procédure.

Pour cela, l'accompagnateur recherche la première zone DSA qui corresponde à une procédure puis, obtient l'adresse du point d'entrée de cette procédure à l'aide de la valeur que contenait le registre général 15 lorsqu'il a été sauvegardé dans la zone DSA précédente. L'analyse de la structure du prologue permet de déterminer le type du point d'entrée qui peut être :

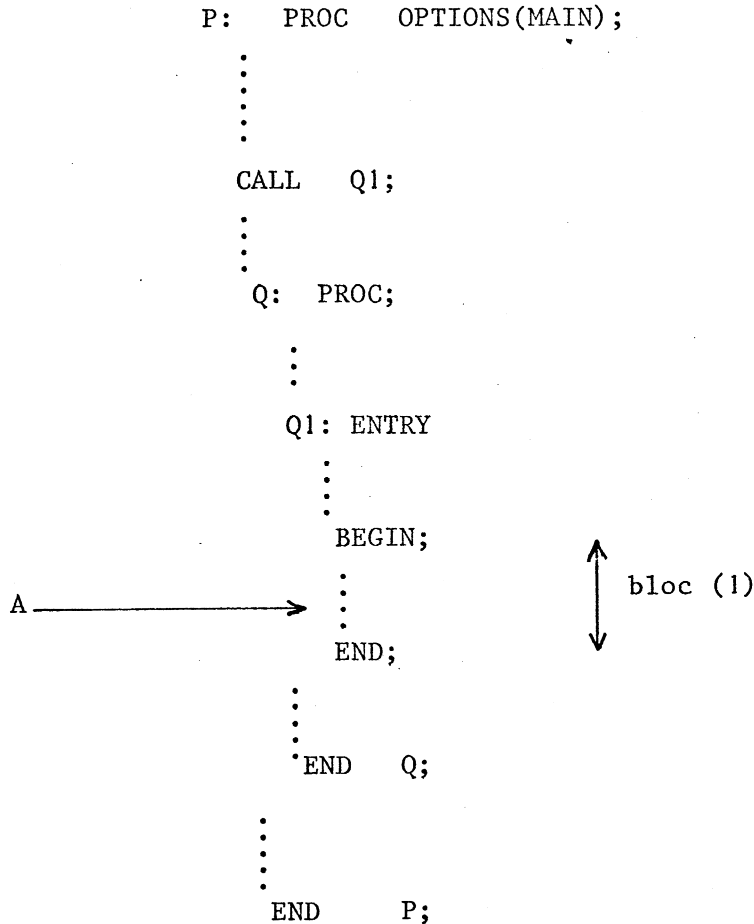
- Un point d'entrée principal de procédure externe: le prologue contient le nom de la procédure externe et la recherche est alors terminée.

- Un point d'entrée secondaire de procédure externe. L'accompagnateur utilise dans ce cas, le repère sur le point d'entrée principal, pour accéder au nom de la procédure.

- Un point d'entrée de procédure interne. L'accompagnateur doit alors poursuivre la recherche en explorant la liste des DSA.

Une partie de cet algorithme est aussi utilisé lors de la recherche du nom d'une procédure interne. Dans ce cas, la recherche est effectuée lors du traitement des requêtes PLIST, RECURS et TRACE qui admettent en paramètre un nom de procédure ou dont la réponse comporte un nom de procédure.

Considérons à titre d'exemple de recherche de nom de procédure externe, le programme suivant :



En supposant que l'exécution se soit déroulée jusqu'au point noté A, la structure qui existe, à cet instant en mémoire est représentée par la figure VI.2.a.

A cet instant, le pseudo-registre IHEQSLA repère la DSA associée au bloc (1); en utilisant le chaînage des DSA,BUGPLI sélecte la DSA associée à la procédure Q. Il adresse ensuite la DSA précédente, c'est-à-dire la DSA associée à la procédure P; la valeur que contenait le registre général 15 (R15) lors de la sauvegarde, lui fournit l'adresse du prologue du point d'entrée Q1. L'analyse du prologue signale que le point d'entrée Q1 est un point d'entrée secondaire. BUGPLI utilise alors le repère sur le point d'entrée principal Q; l'analyse du prologue de ce point d'entrée signale alors que la procédure Q est une procédure interne.

L'accompagnateur parcourt à nouveau la liste des zones DSA et VDA : partant de la DSA associée à la procédure P, il repère la zone de sauvegarde externe et obtient l'adresse du point d'entrée de la procédure L'analyse du prologue signale que P est un point d'entrée principal de procédure externe. La recherche est terminée: le nom de la procédure est contenue dans le prologue de cette procédure.

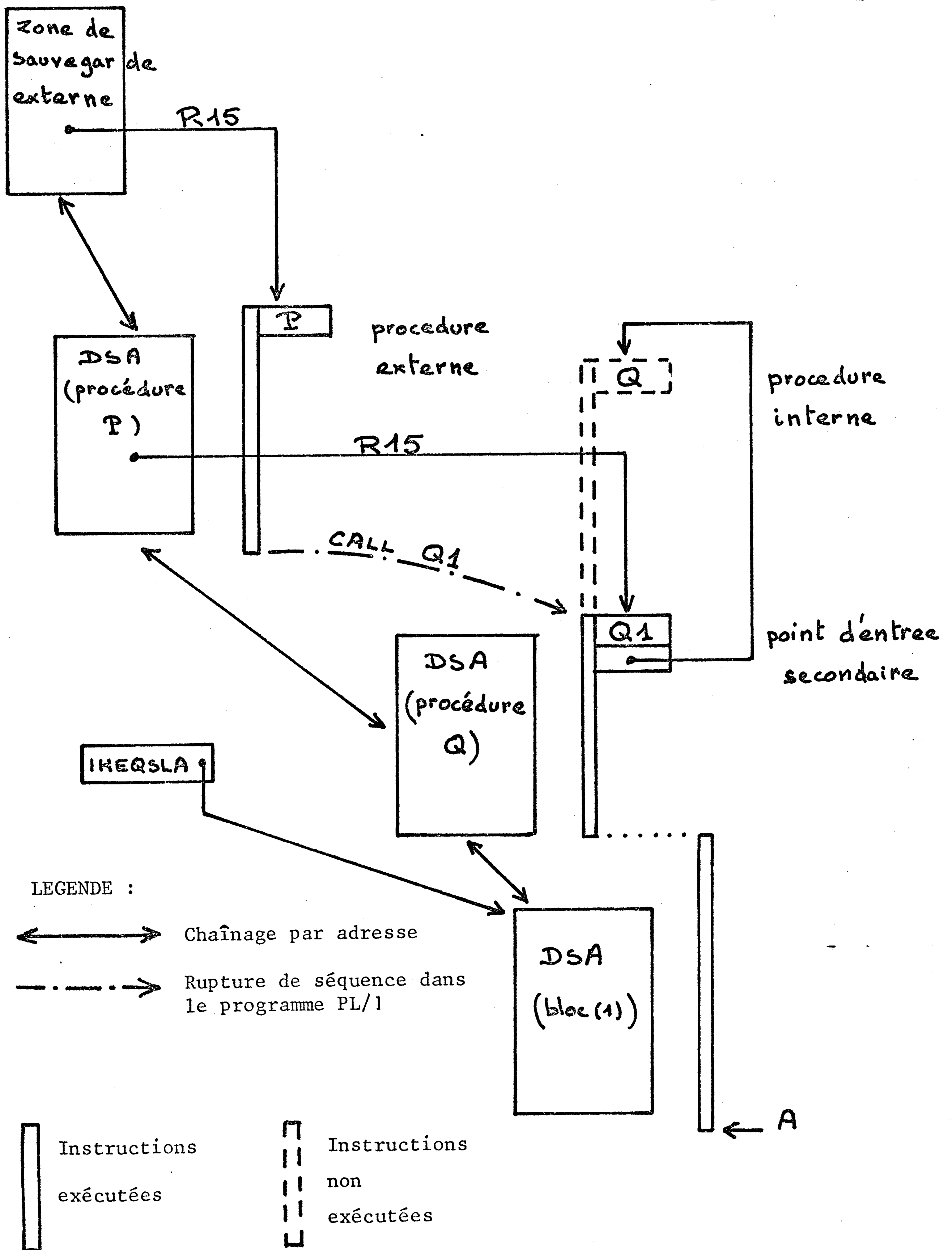


FIGURE VI.2.a.

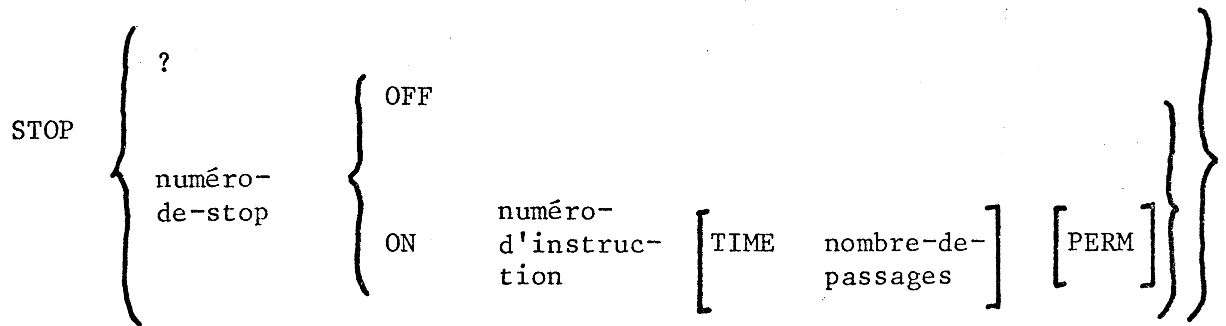
Liaisons entre DSA et prologues de procédures

VII - ETUDE DES MECANISMES DE L'ACCOMPAGNATEUR BUGPLI CONCERNANT LES ARRETS SUR INSTRUCTION.

Un des aspects les plus importants de BUGPLI se rapporte aux arrêts sur instruction. Nous connaissons les requêtes qui donnent à l'utilisateur la possibilité de placer des points d'arrêt et de les enlever; nous nous intéressons maintenant aux mécanismes internes de l'accompagnateur qui permettent de satisfaire ces requêtes et, au processus qui se déroule lorsqu'un arrêt est rencontré au cours de l'exécution d'un programme.

Pour résoudre les divers problèmes, nous avons adjoint à l'accompagnateur une table des arrêts qui recueille l'ensemble des informations nécessaires.

Rappelons la forme générale d'une requête STOP que nous avons énoncée au paragraphe IV.2.10 de la deuxième partie.



VII-1 La table des arrêts sur instruction

Cette table est formée de seize entrées; chaque entrée (figure VII.1.a) caractérise un arrêt et il y a identité entre le numéro d'ordre d'une entrée et le numéro de l'arrêt associé à cette entrée.

Chaque entrée contient :

- un indicateur qui prend les valeurs hexadécimales :
  - FF si l'arrêt correspondant n'est pas positionné,
  - 80 si l'arrêt possède l'attribut permanent,
  - 00 dans les autres cas;

- le numéro de l'instruction PL/1 sur laquelle se trouve l'arrêt associé à l'entrée;

- l'adresse-mémoire de la première des instructions-machine correspondant à l'instruction PL/1 sur laquelle se trouve l'arrêt;

- le nom de la procédure externe qui contient cette instruction;

- une zone de six octets utilisée pour sauvegarder l'instruction-machine qui est modifiée lors de la mise en place de l'arrêt;

- deux zones utilisées pour traiter l'option TIME. La première zone contient la valeur du paramètre associé au mot clé TIME (ou par défaut la valeur 1), la seconde zone est utilisée pour compter le nombre de fois que l'instruction PL/1 est exécutée.

Indicateur	Adresse instruction	Nom de procédure	Sauvegarde instruction	Numéro d'instruction	Paramètre TIME	Compteur du nombre d'exécution

FIGURE VII.1.a - Une entrée de la table des arrêts -

VII-2 Annulation d'un arrêt sur instruction

Le numéro de l'arrêt permet de sélecter l'entrée de la table qui décrit cet arrêt; cette entrée contient l'adresse et la forme originale de l'instruction qui avait été modifiée.

L'annulation de l'arrêt consiste donc à restaurer cette instruction et à invalider l'entrée en plaçant dans l'indicateur la valeur hexadécimale FF.

VII-3 Mise en place d'un arrêt

Cette opération s'effectue sur demande de l'utilisateur qui fournit le numéro de l'arrêt et le numéro de l'instruction PL/1 sur laquelle il désire placer cet arrêt. Le nom de la procédure externe a été indiqué précédemment à l'aide d'une requête QUALIFY.

Appelons i le numéro de l'arrêt et A le numéro de l'instruction PL/1; à cette instruction correspond une suite d'instructions-machine que nous notons a_i.

Placer un arrêt de numéro i sur l'instruction PL/1 de numéro A consiste à :

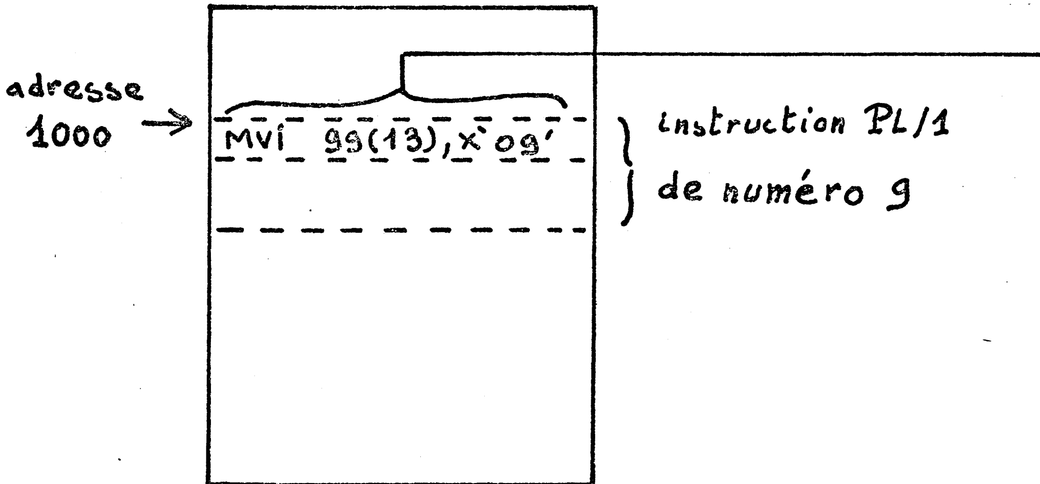
- Rechercher l'adresse de l'instruction PL/1 A;
- Vérifier qu'il n'existe pas déjà un arrêt de numéro j ($j \neq i$) sur cette instruction;
- Vérifier qu'il n'existe pas déjà un arrêt de numéro i sur une autre instruction;
- Prendre une copie de l'instruction-machine a_1;
- Insérer à sa place une <u>instruction spéciale</u>.

Pour rechercher l'adresse de l'instruction PL/1 mise en cause, BUGPLI consulte le fichier de type STPLI correspondant à la procédure qualifiée (Cf deuxième partie, paragraphe IV.2.8) et détermine l'adresse relative de l'instruction A; cette adresse est ensuite transformée en adresse absolue que nous appelons ad(A). Cette adresse est aussi celle de la première des instructions-machine correspondant à l'instruction PL/1; nous la noterons aussi ad(a_1).

L'accompagnateur consulte ensuite l'entrée de rang i de la table des arrêts sur instruction pour vérifier qu'il n'existe pas déjà un arrêt de numéro i sur une autre instruction; lorsque cet arrêt existe, BUGPLI l'annule.

A l'aide de l'adresse absolue ad(a_1), l'accompagnateur s'assure qu'il n'existe pas déjà un arrêt sur l'instruction PL/1 A. Pour cela, il consulte la table des arrêts; si une entrée de la table correspond à cette instruction, il existe donc un arrêt de numéro j ($j \neq i$) sur l'instruction A; l'accompagnateur annule donc cet arrêt.

Lorsque ces vérifications sont effectuées, BUGPLI crée l'entrée correspondant à l'arrêt de numéro i; il copie dans cette entrée la première des instructions-machine correspondant à l'instruction PL/1 A, c'est-à-dire l'instruction-machine a_1; il insère son adresse ad(a_1) dans la table. Il récupère la valeur du nombre-de-passages dans la requête, la place dans la zone réservée au compteur du nombre d'exécution de l'instruction PL/1. Il analyse l'attribut PERM et donne à l'indicateur la valeur hexadécimale 80 si cet attribut existe ou la valeur hexadécimale 00 si cet attribut n'existe pas. Enfin, l'accompagnateur insère le numéro-d'instruction pris dans la requête ainsi que le nom de procédure qui a été sauvegardé lors du traitement de la dernière requête QUALIFY.



Requètes:

QUALIFY P

STOP 2 ON 9

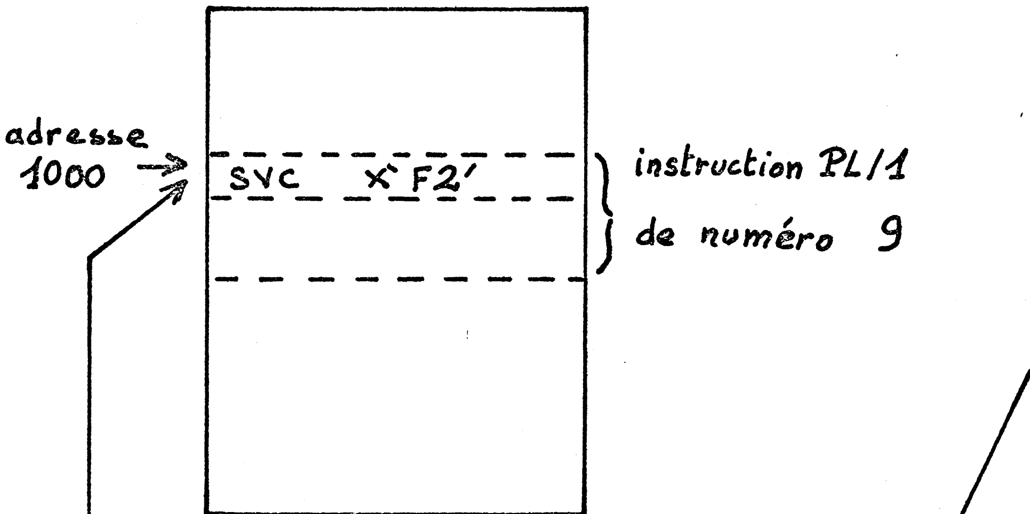


Table des arrêts sur instruction

entrée de rang 2

00	1000	P	MVI 99(13),X'09'	9	1	0

FIGURE VII.2.a.

Mise en place d'un arrêt

La mise en place de l'arrêt se termine en plaçant à l'adresse ad(al) une instruction spéciale d'appel au superviseur SVC de code hexadécimal Fi.

La figure VII.3.a schématise la mise en place d'un arrêt de numéro 2 sur l'instruction PL/1 de numéro 9 dans la procédure P.

VII-4 Suspension de l'exécution lors de la rencontre d'un arrêt.

Lors de l'exécution de l'instruction SVC de code hexadécimal Fi qui matérialise l'arrêt de numéro i, il se produit une interruption de type appel au superviseur et de code hexadécimal Fi; le mécanisme de traitement des interruptions donne le contrôle à l'accompagnateur BUGPLI.

A l'aide du code l'interruption, BUGPLI sélecte l'entrée i de la table des arrêts et incrémente de 1 le compteur d'exécution qui se trouve dans cette entrée. BUGPLI compare ensuite le contenu de ce compteur à la valeur du paramètre TIME et détermine si l'arrêt à prendre en compte (égalité) ou non (inégalité) (Figure VII.4.a).

Dans le cas d'une inégalité, l'exécution du programme doit se poursuivre. L'arrêt ne sera effectif que lors d'une prochaine exécution de l'instruction SVC qui le matérialise; il importe donc que cet arrêt ne soit pas effacé. Pour cela, BUGPLI ne restaure pas l'instruction-machine à sa place mais exécute l'instruction-machine sauvegardée dans l'entrée de la table des arrêts et redonne le contrôle au programme. (L'exécution de l'instruction sauvegardée est effectuée en utilisant l'instruction EXECUTE ; ceci peut se faire sans danger car l'instruction à exécuter n'est jamais une instruction de transfert).

Lorsque le compteur d'exécution est égal à la valeur du paramètre TIME, l'exécution du programme est suspendue et l'arrêt est annulé

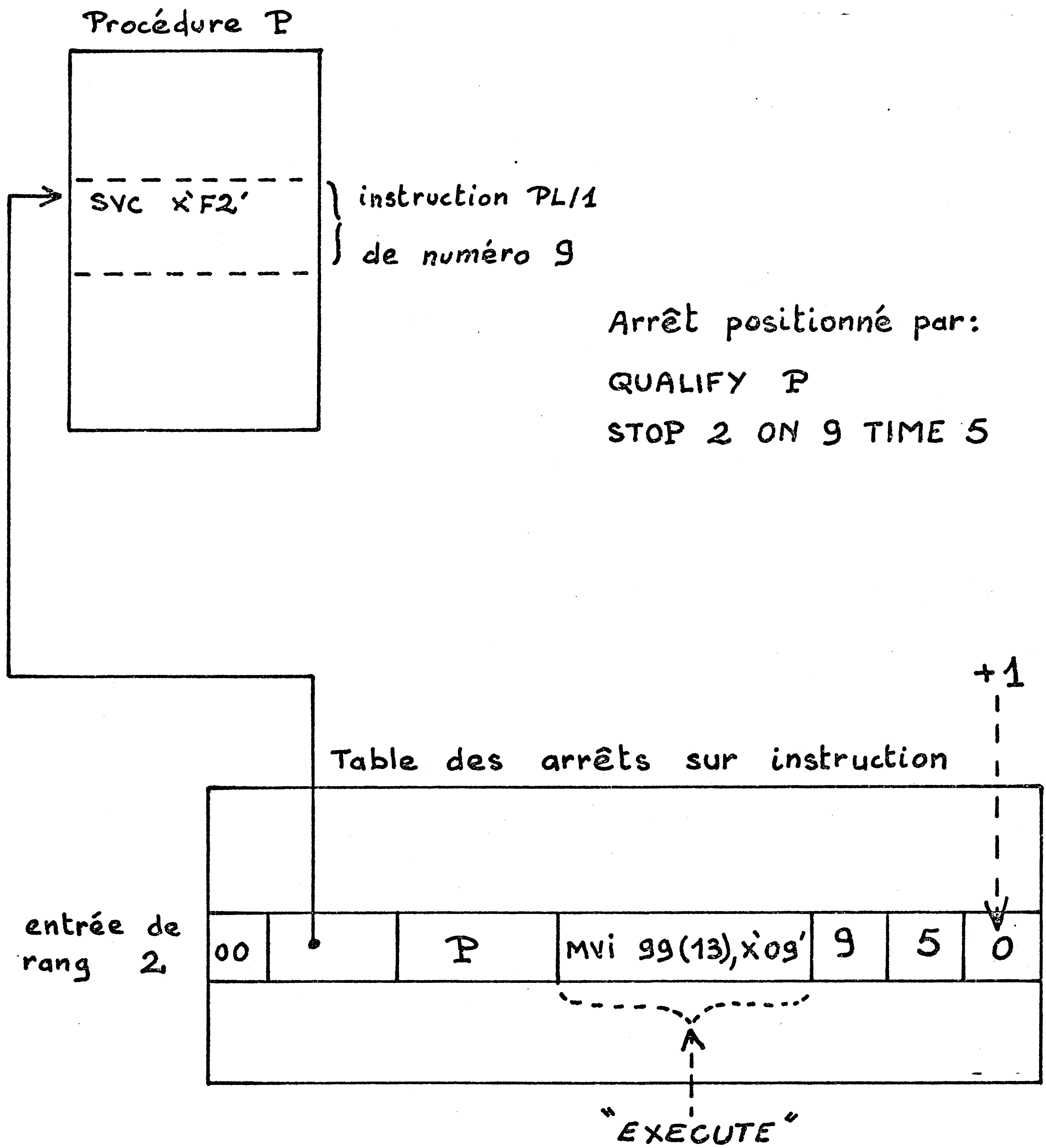


FIGURE VII.4.a.

Rencontre d'un arrêt

s'il ne possède pas l'attribut permanent. Après avoir averti l'utilisateur par le message :

BUGPLI ENTERED: STOP numéro-de-stop ON nom-de-procédure-externe-numéro-d'instruction

BUGPLI se met en attente de requêtes.

VII-5 Reprise de l'exécution après un arrêt

Un second aspect de l'accompagnateur concerne la reprise de l'exécution après un arrêt. Celle-ci a été interrompue par la rencontre de l'arrêt de numéro i placé sur l'instruction PL/1 de numéro A.

Rappelons que nous notons a_1 la première des instructions-machine générées par l'instruction PL/1 de numéro A et que $ad(a_1)$ est l'adresse de l'instruction-machine a_1 et de l'instruction PL/1 A.

VII-5-1 Reprise séquentielle

Si l'arrêt i qui a interrompu l'exécution du programme possède l'attribut permanent, l'instruction SVC de code hexadécimal Fi qui matérialise cet arrêt se trouve toujours en mémoire à l'adresse ad(a). L'accompagnateur BUGPLI exécute alors l'instruction a_1 qui a été sauvegardée dans l'entrée i de la table des arrêts. Il calcule ensuite la longueur $\ell(a_1)$ de l'instruction a_1 puis l'adresse mémoire $ad(a_2) = ad(a_1) + \ell(a_1)$ de la deuxième instruction-machine de la séquence correspondant à l'instruction PL/1 A. Le contrôle est alors donné au programme PL/1 à l'adresse $ad(a_2)$.

Si l'arrêt i ne possédait pas l'attribut permanent, il a été annulé et l'instruction-machine a_1 a été restaurée. Le contrôle est dans ce cas donné au programme PL/1 à l'adresse ad(a_1).

VII-5-2 <u>Reprise non séquentielle</u>.

Une telle reprise est demandée à l'aide de la requête GOTO dont les paramètres sont le numéro de l'instruction PL/1 sur laquelle on désire reprendre l'exécution et l'identification externe du bloc qui contient cette instruction.

L'accompagnateur BUGPLI recherche, dans la pile de sauvegarde des blocs, si le bloc nommé est actif (Cf §V-6 et V-7).

Si le bloc n'est pas actif, le transfert demandé est impossible: on ne peut entrer dans un bloc qu'en passant par le début de ce bloc. La requête est rejetée.

Si le bloc est actif, BUGPLI obtient dans l'entrée correspondante de la pile de sauvegarde l'identification interne de ce bloc, puis évalue l'adresse de l'instruction PL/1 sur laquelle l'utilisateur désire se transférer. Pour donner le contrôle au programme PL/1, BUGPLI appelle la fonction IHESAFC (Cf §V-8) de la bibliothèque d'exécution, en lui fournissant comme paramètres l'adresse de l'instruction PL/1 et l'identification interne du bloc qui contient cette instruction.

C O N C L U S I O N

Le travail que nous venons d'exposer nous a conduit à effectuer une étude approfondie du compilateur et de la bibliothèque d'exécution PL/1; ceci nous a permis d'apporter un certain nombre d'améliorations tant en compilation qu'en exécution.

En compilation, tous les problèmes posés par la méthode d'accès utilisée pour le traitement du fichier utilitaire SYSUT1 ont été résolus: nous avons ainsi rendu possible la compilation de gros programmes dans une machine virtuelle ayant une taille mémoire normale.

Lors de la réalisation de cette méthode d'accès, nous avons fait un choix: en effet, il aurait été possible de ne jamais utiliser d'espace de travail sur disque, à condition de toujours disposer de suffisamment d'espace mémoire. Mais un tel fonctionnement nous aurait vraisemblablement amené à définir des machines virtuelles de taille mémoire de plus en plus importantes. Pour éviter de telles contraintes, nous préférons utiliser aussi l'espace de travail sur disque.

En conséquence, l'espace de travail utilisé par le compilateur réside en partie en mémoire. La partie excédentaire se trouve physiquement sur disque; la zone de disque nécessaire pour ranger le fichier utilitaire est acquise dynamiquement dans un espace temporaire fourni par le système CP.

En exécution, les erreurs de chargement dynamique détectées lors de l'initialisation d'un fichier ont disparu grâce à une légère modification du système CMS.

Lors de la mise en oeuvre des améliorations, nous avons pris garde de modifier le moins possible le compilateur et la bibliothèque d'exécution; ceci nous permet d'envisager une adaptation rapide aux nouvelles versions de CMS et de PL/1. En effet, seuls les interfaces de compilation et d'exécution seront à adapter.

La réalisation de l'accompagnateur d'exécution nous permet de présenter une méthode souple et agréable pour la mise au point de programmes écrits dans un langage évolué.

Les utilisateurs peuvent désormais contrôler le déroulement de leurs programmes au cours de l'exécution, il leur est possible d'arrêter l'exécution en plaçant des points d'arrêt sur les instructions, de consulter le contenu des variables, d'altérer ce contenu, ce qui conduit à modifier à volonté le déroulement des programmes.

En offrant cette nouvelle possibilité de contrôle, nous pensons diminuer de manière appréciable les temps de mise au point de programmes et fournir aux utilisateurs un moyen de mieux utiliser leur temps de calcul.

Vis à vis d'autres environnements de mise au point pour langages évolués, tels que FORTBUG [J1] pour programmes FORTRAN et COBUG [B1] pour programmes COBOL, l'accompagnateur BUGPLI présente une particularité: il permet la mise au point de programmes écrits dans un langage qui possède une notion de bloc.

Nous notons que certaines améliorations et extensions peuvent encore être apportées à BUGPLI. Il serait en effet souhaitable de pouvoir consulter et modifier les contenus des variables de classe BASED ou de type POINTER et OFFSET. Ceci ne peut se faire que si l'on modifie profondément le compilateur pour le forcer à créer pour chaque variable de cette classe ou de ce type, une entrée dans la table des symboles. Il faudrait, bien entendu, modifier aussi les fonctions GET DATA et PUT DATA de la bibliothèque d'exécution, qui utilisent la table des symboles du programme.

Nous pouvons aussi envisager la création d'un mécanisme de macro-requêtes de mise au point qui donnerait à l'utilisateur la possibilité de définir ses propres requêtes à partir des requêtes primitives de l'accompagnateur.

L'étude effectuée lors de la réalisation de notre accompagnateur d'exécution nous a permis de cerner les problèmes que posent d'autres langages évolués tels que ALGOL 60, ALGOL 68, ALGOL W. Nous pensons que les solutions proposées et mises en oeuvre pour PL/1, sont aussi valables pour ces langages.

Toutefois, devant les difficultés de mise en oeuvre, il semble plus séduisant de prévoir et de construire l'outil de mise au point dès la définition du compilateur. En effet, une partie des informations contenues dans le code source est perdue durant la compilation; ces informations pourraient alors être conservées et transmises à l'accompagnateur d'exécution. Une telle possibilité devrait être prévue lors de la réalisation de tout compilateur destiné à s'exécuter dans un système conversationnel.

Nous pensons que les outils de mise au point pour programmes écrits en langages évolués sont appelés à se développer dans les systèmes conversationnels. En effet, l'utilisation de tels outils facilite grandement le travail du programmeur et lui permet d'optimiser l'utilisation de son temps de calcul.

B I B L I O G R A P H I E

[A1] ADIBA M.
Editeurs par contexte pour systèmes conversationnels à temps partagés.
Université Scientifique et Médicale de Grenoble.
Thèse de 3e Cycle - 1971 -

[B1] BASSO P.
COBUG. Accompagnateur pour l'exécution d'un programme COBOL-ANS.
C.I.C.G.
Note Technique - 1972 -

[B2] BERTHAUD
Variables Based et Tâches en PL/1.
Etude FF2-0104.
Centre Scientifique IBM de Grenoble.

[J1] JACOLIN H.
Mise au point conversationnelle de programmes FORTRAN : FORTBUG.
Etude n°FF2-0136 Juin 1972
Centre Scientifique IBM de Grenoble.

[L1] LEFEBVRE P.
SPY : Un système de contrôle pour la mise au point de systèmes de programmation grâce à un couplage de machines virtuelles.
Université et Scientifique et Médicale de Grenoble.
Thèse de 3ème Cycle - 1972 -

[L2] LE HEIGET J.P.
Généralisation de la notion d'espace virtuel sous les systèmes CP-67/CMS.
Thèse CNAM - 1972 -

[L3] LE HEIGET J.P.
Gestion de fichiers du système CMS.
Note interne C.I.C.G. - 1971 -

[L4] de LAMBERTERIE X.

Espaces virtuels et gestion de fichiers.
Université Scientifique et Médicale de Grenoble.
Thèse 3ème Cycle - 1973 -

[L5] LEROUDIER J.

Une analyse de Système.
Université Scientifique et Médicale de Grenoble.
Thèse de 3ème Cycle - 1973 -

[N1] NGUYEN-THANH THI

XCP. Un environnement graphique conversationnel pour l'examen d'un système. Application à CP/67.
Université Scientifique et Médicale de Grenoble.
Thèse de 3ème Cycle - 1973 -

[P1] PETEUL-HARMEL B.

La mise au point de programmes par simulation.
Réalisation d'un support conversationnel de mise au point: PILOTE.
Université Scientifique et Médicale de Grenoble.
Thèse de 3ème Cycle - 1971 -

[S1] SAVARY H.

Outils de mise au point pour langages de haut niveau: association de modules et contrôle de l'exécution.
Université Scientifique et Médicale de Grenoble.
Thèse de 3ème Cycle - 1973 -

NOTICES TECHNIQUES

[I1] IBM
CMS Program Logic Manual

[I2] IBM
CP/67 Program Logic Manual.

[I3] IBM
CP/67-CMS User's Guide.

[I4] IBM OS/360
PL1(F) Compiler
Program Logic Manual.

[I5] IBM OS/360
PL/1 Subroutine Library
Program Logic Manual.

[I6] IBM OS/360
PL/1 (F)
Language Reference Manual.

[I7] IBM OS/360
PL/1 (F)
Programmer's Guide.

[I8] IBM OS/360
Supervisor and Data Management Services.

[I9] IBM OS/360
Supervisor and Data Management Macro Instructions.

[I10] IBM OS/360
System Control Blocks.

[I11] IBM OS/360-TSO
PL/1 Checkout Compiler.
General Information.

[I12] IBM OS/360-TSO
PL/1 Checkout Compiler.
User's Guide.