La mise au point de programmes par simulation : réalisation d'un support conversationnel de mise au point : PILOTE

Bernard Peteul-Harmel

▶ To cite this version:

Bernard Peteul-Harmel. La mise au point de programmes par simulation : réalisation d'un support conversationnel de mise au point : PILOTE. Modélisation et simulation. Université Joseph-Fourier - Grenoble I, 1971. Français. NNT : . tel-00282878

HAL Id: tel-00282878
https://theses.hal.science/tel-00282878

Submitted on 28 May 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 :

TH 423

THESE

présentée à

L'UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

pour obtenir

LE GRADE DE DOCTEUR DE TROISIEME CYCLE.

INFORMATIQUE

par

Bernard PETEUL-HARMEL

LA MISE AU POINT DE PROGRAMMES PAR SIMULATION

REALISATION D'UN SUPPORT CONVERSATIONNEL DE MISE AU POINT : PILOTE

Thèse soutenue le 26 Juin 1971 devant la Commission d'Examen :

Monsieur	N. GASTINEL	Président
Messieurs	L. BOLLIET	Examinateurs
	M. GRIFFITHS	
	C. HANS	

UNIVERSITE SCIENTIFIQUE
ET MEDICALE DE GRENOBLE

LISTE DES PROFESSEURS

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

PROFESSEURS TITULAIRES

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

	FAU René	Clinique neuro-psychiatrique
	FELICI Noël	Electrostatique
	GAGNAIRE Didier	Chimie physique
	GALLISSOT François	Mathématiques pures
	GALVANI Octave	Mathématiques pures
	GASTINEL Noël	Analyse numérique
	GERBER Robert	Mathématiques pures
	GIRAUD Pierre	Géologie
	KLEIN Joseph	Mathématiques pures
Mme	KOFLER Lucie	Botanique et physiologie végétale
MM.	KOSZUL Jean-Louis	Mathématiques pures
	KRAVTCHENKO Julien	Mécanique
	KUNTZMANN Jean	Mathématiques appliquées
	LACAZE Albert	Thermodynamique
	LACHARME Jean	Biologie végétale
	LATURAZE Jean	Biochimie pharmaceutique
	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	Séméiologie médicale
	MASSEPORT Jean	Géographie
	MAZARE Yves	Clinique médicale A
	MICHEL Robert	Minéralogie et Pétrographie
	MOURIQUAND Claude	Histologie
	MOUSSA André	Chimie nucléaire
	NEEL Louis	Physique du Solide
	OZENDA Paul	Botanique
	PAUTHENET René	Electrotechnique
	PAYAN Jean-Jacques	Mathématiques pures
	PEBAY-PEYROULA Jean-Claude	Physique
	PERRET René	Servomécanismes
	PILLET Emile	Physique industrielle
	RASSAT André	Chimie systématique
	RENARD Michel	Thermodynamique
	REULOS René	Physique industrielle
	RINALDI Renaud	Physique
	ROGET Jean	Clinique de pédiatrie et de puériculture
	SANTON Lucien	Mécanique
	SEIGNEURIN Raymond	Microbiologie et Hygiène
	SENGEL Philippe	Zoologie
	SILBERT Robert	Mécanique des fluides
	SOUTIF Michel	Physique générale
	TANCHE Maurice	Physiologie
	TRAYNARD Philippe	Chimie générale
	VAILLAND François	Zoologie
	VAUQUOIS Bernard	Calcul électronique
Mme	VERAIN Alice	Pharmacie galénique
	VERAIN André	Physique
Mme	VEYRET Germaine	Géographie
MM.	VEYRET Paul	Géographie
	VIGNAIS Pierre	Biochimie médicale
	YOCCOZ Jean	Physique nucléaire théorique

PROFESSEURS ASSOCIES

MM.	BULLEMER Bernhard	Physique
	RADHAKRISHNA Pidatala	Thermodynamique

PROFESSEURS SANS CHAIRE

MM.	AUBERT Guy	Physique
Mme	BARBIER Marie-Jeanne	Electrochimie
MM.	BARRA Jean	Mathématiques appliquées
	BEAUDOING André	Pédiatrie
	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
	DESRE Pierre	Métallurgie
	DOLIQUE Jean-Michel	Physique des plasmas
	GAUTHIER Yves	Sciences biologiques
	GEINDRE Michel	Electroradiologie
	GIDON Paul	Géologie et Minéralogie
	GLENAT René	Chimie organique
	HACQUES Gérard	Calcul numérique
	JANIN Bernard	Géographie
Mme	KAHANE Josette	Physique
MM.	LATREILLE René	Chirurgie générale
	LAURENT Pierre	Mathématiques appliquées
	MULLER Jean-Michel	Thérapeutique
	PERRIAUX Jean-Jacques	Géologie et minéralogie
	POULOUJADOFF Michel	Electrotechnique
	REBECQ Jacques	Biologie (CUS)
	REVOL Michel	Urologie
	REYMOND Jean-Charles	Chirurgie générale
	ROBERT André	Chimie papetière
	SARRAZIN Roger	Anatomie et chirurgie
	SARROT-REYNAULD Jean	Géologie
	SIBILLE Robert	Construction Mécanique
	SIROT Louis	Chirurgie générale
Mme	SOUTIF Jeanne	Physique générale
M.	VALENTIN Jacques	Physique nucléaire

MAITRES DE CONFERENCES ET MAITRES DE CONFERENCES AGREGES

Mle	AGNIUS-DELORD Claudine	Physique pharmaceutique
	ALARY Josette	Chimie analytique
MM.	AMBLARD Pierre	Dermatologie
	AMBROISE-THOMAS Pierre	Parasitologie
	ARMAND Yves	Chimie

	BEGUIN Claude	Chimie organique
	BELORIZKY Elie	Physique
	BENZAKEN Claude	Mathématiques appliquées
Mme	BERTRANDIAS Françoise	Mathématiques pures
MM.	BLIMAN Samuel	Electronique (EIE)
	BLOCH Daniel	Electrotechnique
Mme	BOUCHE Liane	Mathématiques (CUS)
MM.	BOUCHET Yves	Anatomie
	BOUSSARD Jean-Claude	Mathématiques appliquées
	BOUVARD Maurice	Mécanique des Fluides
	BRIERE Georges	Physique expérimentale
	BRODEAU François	Mathématiques (IUT B)
	BRUGEL Lucien	Energétique
	BUISSON Roger	Physique
	BUTEL Jean	Orthopédie
	CHAMBAZ Edmond	Biochimie médicale
	CHAMPETIER Jean	Anatomie et organogénèse
	CHARACHON Robert	Oto-Rhino-Laryngologie
	CHIAVERINA Jean	Biologie appliquée (EFP)
	CHIBON Pierre	Biologie animale
	COHEN-ADDAD Jean-Pierre	Spectrométrie physique
	COLOMB Maurice	Biochimie médicale
	CONTE René	Physique
	CROUZET Guy	Radiologie
	DURAND Francis	Métallurgie
	DUSSAUD René	Mathématiques (CUS)
Mme	ETERRADOSSI Jacqueline	Physiologie
MM.	FAURE Jacques	Médecine légale
	GAVEND Michel	Pharmacologie
	GENSAC Pierre	Botanique
	GERMAIN Jean-Pierre	Mécanique
	GIDON Maurice	Géologie
	GRIFFITHS Michael	Mathématiques appliquées
	GROULADE Joseph	Biochimie médicale
	HOLLARD Daniel	Hématologie
	HUGONOT Robert	Hygiène et médecine préventive
	IDELMAN Simon	Physiologie animale
	IVANES Marcel	Electricité
	JALBERT Pierre	Histologie
	JOLY Jean-René	Mathématiques pures
	JOUBERT Jean-Claude	Physique du Solide
	JULLIEN Pierre	Mathématiques pures
	KAHANE André	Physique générale
	KUHN Gérard	Physique
Mme	LAJZEROWICZ Jeannine	Physique
MM.	LAJZEROWICZ Joseph	Physique
	LANCIA Roland	Physique atomique
	LE JUNTER Noël	Electronique
	LEROY Philippe	Mathématiques
	LOISEAUX Jean-Marie	Physique nucléaire
	LONGEQUEUE Jean-Pierre	Physique nucléaire
	LUU DUC Cuong	Chimie organique
	MACHE Régis	Physiologie végétale
	MAGNIN Robert	Hygiène et Médecine préventive
	MARECHAL Jean	Mécanique
	MARTIN-BOUYER Michel	Chimie (CUS)
	MAYNARD Roger	Physique du Solide
	MICOUD Max	Maladies infectieuses
	MOREAU René	Hydraulique (INP)

	NEGRE Robert	Mécanique
	PARAMELLE Bernard	Pneumologie
	PECCOUD François	Analyse (IUT B)
	PEFFEN René	Métallurgie
	PELMONT Jean	Physiologie animale
	PERRET Jean	Neurologie
	PERRIN Louis	Pathologie expérimentale
	PFISTER Jean-Claude	Physique du Solide
	PHELIP Xavier	Rhumatologie
Mle	PIERY Yvette	Biologie animale
	RACHAIL Michel	Médecine interne
	RACINET Claude	Gynécologie et obstétrique
	RICHARD Lucien	Botanique
Mme	RINAUDO Marguerite	Chimie macromoléculaire
MM.	ROMIER Guy	Mathématiques (IUT B)
	ROUGEMONT (DE) Jacques	Neuro-chirurgie
	STIEGLITZ Paul	Anesthésiologie
	STOEBNER Pierre	Anatomie pathologique
	VAN CUTSEM Bernard	Mathématiques appliquées
	VEILLON Gérard	Mathématiques appliquées (INP)
	VIALON Pierre	Géologie
	VOOG Robert	Médecine interne
	VROUSSOS Constantin	Radiologie
	ZADWORNY François	Electronique

MAITRES DE CONFERENCES ASSOCIES

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

CHARGES DE FONCTIONS DE MAITRES DE CONFERENCES

Mme	BERIEL Hélène	Physiologie
Mme	RENAUDET Jacqueline	Microbiologie

Je tiens à remercier:

Monsieur le Professeur Noël GASTINEL, Directeur du Laboratoire de Calcul, qui a bien voulu me faire l'honneur de présider le jury de cette thèse.

Monsieur Louis BOLLIET, Professeur à l'Institut Universitaire de Technologie de GRENOBLE, qui m'a orienté vers la recherche en programmation et a dirigé cette thèse.

Monsieur Michael GRIFFITHS, Maître de Conférence à la Faculté des Sciences de GRENOBLE, pour l'intérêt sympathique qu'il a porté à ce travail.

Monsieur Claude HANS, Ingénieur au Centre Scientifique IBM France, qui est à l'origine du sujet de thèse, qui a mis à ma disposition sa grande expérience, et n'a ménagé ni son temps, ni ses conseils précieux, pour me diriger et m'encourager dans mes travaux.

Tous mes collègues du Laboratoire et du Centre Scientifique, qui m'ont aidé dans la réalisation de ma tâche, grâce à leurs remarques, leurs conseils ou leur contribution active, en particulier Madame BERTHAUD et Messieurs BELLOT, LEFEBVRE et SAVARY.

Je remercie également le Centre Scientifique et les Services de Dactylographie et de Reproduction du Laboratoire auxquels je dois la réalisation matérielle de ce document.

T A B L E   D E S   M A T I E R E S

<><><><><>

Pages

CHAPITRE I: INTRODUCTION ... I.1

CHAPITRE II: LA MISE AU POINT DES PROGRAMMES ... II.1

1.LE COUT DE LA MISE AU POINT ... II.2
1.1.Les facteurs humains ... II.4
1.2.Les difficultés des programmes ... II.5

2.L'ENVIRONNEMENT DE MISE AU POINT ... II.7
2.1.La mise au point en "batch processing" ... II.7
2.2.La mise au point conversationnelle ... II.8
2.3. Les outils de mise au point ... II.10
2.3.1.Les outils statiques ... II.10
2.3.2.Les outils dynamiques ... II.11
2.4.La manipulation des outils de mise au point ... II.14
2.4.1.Le langage de commandes élémentaires ... II.15
2.4.2.Le macro-langage ... II.16

3.LE SUPPORT DE MISE AU POINT ... II.16

CHAPITRE III: LE SYSTEME REALISE: PILOTE ... III.1

1.L'ENVIRONNEMENT DE MISE AU POINT DEBUG ... III.2
1.1.Origine de DEBUG ... III.2
1.2.Utilisation de DEBUG ... III.2
1.3.Remarques sur DEBUG ... III.4

Pages

2.PRESENTATION DE PILOTE III.5

2.1.Le mode conversationnel III.6

2.2.Le mode simulation III.8

2.3.Le mode exécution III.10

3.MANUEL DE L'UTILISATEUR DE PILOTE SOUS CMS III.10

3.1.Activation de PILOTE III.10

3.2.Entrée dans le mode conversation III.13

3.3.Abandon du mode conversation III.15

3.4.Langage de requêtes III.16

add III.21

address III.22

at III.24

blip III.27

clear III.28

close III.30

collect III.31

dechex III.33

drop III.34

hexdec III.35

list III.36

monitor III.38

numbinst III.39

on III.40

print III.42

qualify III.44

read III.47

stop III.43

substract III.50

trace III.51

using III.53

*gprs III.55

*x III.56

Pages

4.LE MACRO-LANGAGE DE PILOTE. III.57
4.1.Les objectifs III.57
4.2.Les spécifications III.58
4.2.1.Les éléments du macro-langage III.59
4.2.1.1.Les symboles de base III.59
4.2.1.2.Les nombres et les chaînes III.60
4.2.1.3.Les identificateurs de variables III.60
4.2.2.L'instruction d'affectation III.61
4.2.3.Etiquettes et instruction de branchement III.61
4.2.4.Instruction conditionnelle III.62
4.2.5.Instruction composée III.63
4.2.6.Appel de procédures III.64

CHAPITRE IV: DESCRIPTION INTERNE ET LOGIQUE DE PILOTE IV.1

1.CHARGEMENT DE PILOTE DANS LA MEMOIRE IV.2

2.DEMON: PARTITION CONVERSATIONNELLE DE PILOTE IV.7
2.1.Communications entre DEMON et les modes Exécution et Simulation IV.7
2.1.1.Entrée à partir du mode exécution IV.8
2.1.2.Entrée à partir du mode simulation IV.9
2.1.3.Sorties du mode conversation IV.10
2.2.Analyse et traitement des requêtes IV.11

3.MONITEUR: SUPERVISEUR DU PROGRAMME TESTE IV.14
3.1.L'interpréteur IV.15
3.1.1.Simulation d'une instruction IV.16
3.1.2.Traitement des interruptions IV.18
3.2.Gestion des entrées-sorties IV.21
3.3.La trace et la collecte. IV.23

CHAPITRE V: CONCLUSION

APPENDICE A: UN EXEMPLE DE SESSION

APPENDICE B: QUELQUES EXEMPLES DE SORTIES

APPENDICE C: LA COMMANDE: "SYMBOL"

APPENDICE D: LA COMMANDE: "TAPPILOT".

BIBLIOGRAPHIE

CHAPITRE I

INTRODUCTION

Bien que la mise au point des programmes qui s'exécutent sur ordinateur ne soit pas un problème nouveau, on remarque qu'il n'existe encore que peu d'ouvrages sur la programmation qui abordent ce sujet. Deux explications peuvent être avancées.

La première est d'ordre formel. C'est la difficulté de traiter dans un cadre général, un sujet où le rôle joué par le programmeur a une importance aussi fondamentale. KNUTH souligne ceci dans son livre: "THE ART OF THE COMPUTER PROGRAMMING" [réf.17] et ajoute qu'un bon programmeur se distingue par sa créativité. Or il se révèle que c'est au cours de l'étape de mise au point que la liberté d'action est la plus grande donc les initiatives personnelles les plus nombreuses: trouver la marche à suivre qui met en évidence une faute dans l'exécution d'un programme demande en effet beaucoup de réflexion et d'intuition. Il est peu d'exemples où des moyens automatiques montrent le véritable point défaillant d'un programme; en général, ils soulignent une erreur qui est la conséquence d'anomalies antérieures et que les moyens automatiques (le plus souvent une interruption) n'avaient pas su découvrir, abandonnant ainsi cette tâche au programmeur.

Quant à la deuxième explication peut être faut-il l'attribuer à des motifs prioritaires. Dans "THE DEBUGGING EPOCH OPENS" [Ref 12], Halpern explique que si les problèmes de mise au point ne sont pas encore franchement abordés, c'est que d'autres plus urgents nécessitaient des solutions et mobilisaient les plus grands efforts. La mise au point, écrit Halpern, n'est que le troisième obstacle de la programmation.

Le premier était représenté par des difficultés physiques.

Elles avaient pour origine les petites tailles des mémoires centrales, la lenteur des premiers calculateurs et la synchronisation de l'unité centrale et des organes périphériques rapides comme les tambours.

Le second obstacle à vaincre fût le langage de programmation. Il est évident que l'écriture binaire des programmes ou même en langage machine ne facilitait pas la programmation et en tout cas limitait l'usage des ordinateurs à des spécialistes.

Mais à présent que les ordinateurs disposent de mémoires centrales vastes et rapides, que les langages de programmation permettent une formulation claire et agréable pour la plupart des algorithmes, l'opération finale pour achever un programme est sa mise au point.

On croit bien souvent que les problèmes de programmation et tout particulièrement de mise au point peuvent être facilités en grande part, grâce uniquement à l'usage d'un terminal à clavier de caractères, géré par un système dit conversationnel. Si cette opinion souvent empirique peut en effet se justifier, il convient néanmoins d'en faire la preuve et de montrer que cette preuve n'est pas dans les raisons généralement avancées.

A la vérité, on peut démontrer que le principal mérite des facilités de programmation revient essentiellement (dans l'état actuel des choses) au système de partage de temps et de ressources qui met "simultanément" l'ordinateur à la disposition de plusieurs utilisateurs connectés à l'aide d'un terminal. Ce système présente en effet deux intérêts importants:

- il permet de lire et de modifier aisément la mémoire où est chargé le programme.
- il autorise l'accès à des fonctions, généralement la simulation des clés du pupitre de l'ordinateur, pour tester et éventuellement agir sur l'exécution du programme.

Tous les familiers de ces systèmes s'entendent pour confirmer

qu'il s'agit bien là d'avantages appréciables. Mais ne pourrait-on pas bénéficier d'avantages comparables avec un coût réduit? c'est-à-dire éliminer les temps "d'overhead" parfois importants qu'introduisent ces systèmes.

Il est en effet facile de modifier un programme perforé sur cartes, et on peut très bien avoir recours à des procédures de mise au point et de tests sans utiliser de terminal. On s'aperçoit alors que ce qui est apprécié dans un système "multi-consoles", c'est le bref délai entre le moment où l'on "active" son travail et celui où l'on récupère les résultats (nous appelons ce temps, le temps de retenue). Si dans la plupart des systèmes conversationnels ce temps se compte en heures, on peut en concevoir certains où le temps de retenue se réduit à quelques dizaines de minutes [Ref 19]. Cette solution moins coûteuse donnerait certainement satisfaction à un grand nombre de personnes réclamant un système "multi-consoles" mais dont le désir véritable est de disposer d'un environnement à temps de retenue assez faible.

Supposons maintenant que nous disposions d'un système "multi-consoles", doté d'un moniteur et fournissant des temps de retenue très courts. Cet avantage devra t-il s'accompagner de nouvelles contraintes? -nous n'envisageons ici que la condition de l'utilisateur, sans s'occuper de celle de la machine- Dans cet environnement, la personne ne dispose comme moyen d'échange avec le calculateur que d'un terminal. Ce matériel assez lent (10 à 15 caractères/seconde) interdit les listes de programmes, les analyses de la mémoire et autres volumes d'informations dont les sorties sur le terminal font perdre tout le bénéfice du bref temps de retenue.

Si l'on veut un système "multi-consoles" efficace, on s'aperçoit qu'il faut le doter de possibilités nouvelles, exploitant au maximum l'interaction homme-machine.

En ce qui concerne la mise au point, il faut au moins fournir au programmeur la possibilité de formuler les requêtes et les réponses en termes

se rapprochant du langage source du programme assemblé ou compilé. Pour cela les modules conversationnels doivent avoir accès:

- à toutes les instructions et aux données du programme à tester lorsqu'il est chargé en mémoire.
- à l'ensemble des paramètres (adresses des points d'entrée, description symbolique du programme,...) qui ont permis de construire le programme en mémoire.

Mais si ceci est possible sans l'usage d'un terminal il n'en est plus de même lorsque l'on désire permettre aux utilisateurs d'être présents pendant l'exécution de leurs travaux et ainsi leur offrir un potentiel d'intervention. C'est là l'atout essentiel des systèmes "multi-consoles".

La méthode de mise au point la plus connue est l'examen d'une analyse de la mémoire obtenue sur l'occurrence d'une erreur fatale. Chacun connaît les désagréments de cette technique "post-mortem" en particulier lorsque l'erreur responsable n'est pas fatale et entraîne des modifications au programme. A partir de cette analyse, le programmeur doit retracer rétrospectivement l'exécution, et cet effort pénible n'est pas assuré d'une récompense, en particulier dans les programmes utilisant des structures de recouvrement. Pour remédier à ce délicat problème, l'interprétation dynamique des programmes pendant l'exécution semble intéressante. Elle permet en effet de tracer automatiquement les instructions qui se déroulent, de sortir les valeurs intermédiaires des variables, et de reconnaître si certaines conditions sont réalisées pour agir selon la demande du programmeur (on appellera ces demandes, des directives).

Les systèmes classiques de traitement par lots ("batch processing") excluent généralement de telles méthodes en raison du volume d'informations très impressionnant qui s'imprime et dont seulement une partie est

intéressante. Or à priori il est quasiment impossible dans de tels systèmes de savoir quelle sera cette portion intéressante.

Bien au contraire, les systèmes intéractifs conversationnels, permettent de choisir très sélectivement les parties de l'exécution du programme qui méritent un intérêt et sur celles-ci, intervenir grâce au terminal. L'exécution étant alors interrompue, le programmeur, dans un mode conversationnel, envoie ses requêtes, lit et modifie le contenu des variables, des registres, donne des directives, etc... puis redemande l'exécution au point d'interruption ou à tout autre point.

Le besoin senti par les programmeurs de tout genre, d'une assistance nécessaire pour mettre au point leurs travaux et l'intérêt qu'un terminal apporte à ce problème, encouragent les travaux dans ce domaine. Nous avons donc abordé le sujet et développé un environnement conversationnel de mise au point: PILOTE.

Pour des raisons justifiées par la suite, nous avons réalisé cet environnement en lui donnant le maximum d'indépendance vis à vis d'un système d'exploitation général, en particulier vis à vis des opérations d'entrées-sorties. Nous nommons support un tel environnement. L'objectif recherché est de fournir au programmeur une assistance qui soit à la fois automatique pour les cas généraux, mais aussi capable de s'adapter aux cas particuliers. Grâce à cette aide, nous espérons affranchir l'utilisateur de toutes les opérations exécutables par l'ordinateur afin de lui permettre une plus grande attention sur les véritables problèmes de mise au point.

Cette aide, particulièrement destinée aux programmeurs "assembleurs" se fait à deux niveaux:

. au niveau de l'exécution du programme testé, PILOTE dispose

d'un interpréteur qui simule le fonctionnement complet d'une unité centrale. De ce fait le programmeur peut contrôler dynamiquement et très minutieusement l'exécution et agir comme il le désire avant que l'erreur ne vienne perturber le programme. De cette interprétation, on peut obtenir l'impression sur terminal, machine à imprimer, ou bande magnétique, d'informations prélevées dynamiquement: trace, image - mémoire partielle ("snapshot"), collecte de renseignements,...En outre, l'unité centrale simulée étant tout à fait conforme à l'unité centrale réelle, aucune contrainte ne vient géner la mise au point.

. au niveau de la communication, le programmeur dispose d'un langage de commandes interprété par PILOTE, langage d'un niveau élémentaire où chaque instruction est une requête, et d'un macro-langage qui permet d'exécuter des procédures de mise au point.

Dans la version réalisée et actuellement disponible, PILOTE fonctionne avec le système CMS [Réf 15] .

CHAPITRE II

LA MISE AU POINT DES PROGRAMMES

Il est difficile de préciser quand commence et quand se termine la mise au point d'un programme. Déjà au niveau de la définition des spécifications (organisations des données, liens entre les sous-programmes, etc..) on peut faire intervenir des considérations permettant ultérieurement de faciliter la mise au point [Ref 25] (et ceci semble trop souvent oublié). Puis au niveau de la phase d'écriture et d'exécution la mise au point se poursuit. Enfin lorsque le produit est utilisable, l'effort n'est pas terminé. L'expérience montre en effet que malgrè les précautions que l'on peut prendre et le soin apporté, la plupart des programmes qui manipulent de nombreux objets (données de différents types, organes d'entrées-sorties,etc...) et en particulier les systèmes d'exploitation, contiennent encore un grand nombre d'erreurs. Pour preuve, on peut citer la remarque d'Hopkins [Réf 23, p 20] qui signale que chaque nouvelle version du système IBM OS/360 apporte la correction de 1000 erreurs environ, ce nombre étant sensiblement constant. Ce chiffre important, sans doute équivalent pour d'autres systèmes comparables, montre que la maintenance joue un rôle fondamental lorsqu'il s'agit d'un produit distribué sur une grande échelle.

Pour nos travaux, nous nous sommes limités à la mise au point, phase de la programmation. C'est-à-dire celle qui conduit le programmeur de l'instant où il fait exécuter individuellement et pour la première fois chaque sous-programme qui compose l'ensemble, jusqu'à l'instant où il procède aux tests de contrôle afin de s'assurer que dans tous les cas, les résultats fournis sont satisfaisants.(par exemple en faisant varier les paramètres d'entrées du programme). Pendant cette période, la manière d'opérer varie assez sensiblement. Ceci est dû au fait que dans les premiers temps peu d'instructions sont exécutées et qu'il s'agit surtout de corriger des erreurs dont l'incidence est

locale: erreurs dûes à des "fautes de plume", à une mauvaise connaissance du langage de programmation, ou à un oubli. Le programmeur peut à la limite suivre l'exécution de son programme instruction par instruction. Au contraire, lorsque l'écriture est presque achevée c'est-à-dire pendant la période des tests, le contrôle est donné à l'ensemble des composants du programme et ce qu'il est important de connaitre c'est l'enchaînement des différentes parties, quelles en sont les conditions d'entrées et de sorties. Le programmeur procède alors selon une stratégie de tests qu'il applique durant l'exécution. Il a donc une vue d'ensemble qui nécessite une information plus macroscopique.

Avant de montrer par quels moyens l'ordinateur peut soutenir les efforts du programmeur et l'assister efficacement, il est intéressant de souligner la part des opérations de mise au point dans le coût de réalisation d'un projet. Nous noterons également combien celui-ci peut être influencé par les facteurs humains et combien il faut le pondérer suivant la difficulté très variable des problèmes qui sont programmés.

Enfin nous décrirons dans ce chapitre comment la mise au point peut être réalisée et quelles sont les lignes directrices que nous nous sommes fixés pour développer le support PILOTE. Celui-ci est plus complètement décrit d'un point de vue externe (utilisation) et interne (fonctionnement) dans les deux chapitres qui suivent.

II.1. LE COUT DE LA MISE AU POINT

Le coût de la mise en oeuvre d'un projet, malgrè les études prévisionnelles qui peuvent être entreprises, se révèle presque toujours sous-estimé. On constate en effet que les efforts et les délais relatifs à cette étape sont souvent dépassés aussi bien pour les projets de grande envergure comme des systèmes d'exploitation que pour ceux plus modestes destinés à un usage plus limité. Les raisons sont multiples. Citons entre autres les modifications apportées, en cours de réalisation, aux spécifications de départ et les fluctuations impré-

visibles des moyens utilisés. Mais plus importantes encore sont les incertitudes sur le temps consacré à la correction des erreurs de programmation. Ces erreurs qui motivent la plus forte partie de la mise au point s'échelonnent sur différentes étapes que l'on peut distinguer, les difficultés soulevées n'étant pas de même nature.

a- à l'exécution de chaque sous-programme mis au point individuellement, les fautes sont nombreuses mais n'entrainent pas une mise au point compliquée. Elles proviennent d'oublis,de maladresses, ou d'une méconnaissance du langage. Généralement elles sont vite corrigées.

b- au moment des tests du programme fonctionnant avec tout ses sous-programmes, on détecte des erreurs au niveau de l'algorithme. Ces erreurs sont plus difficiles à mettre en évidence et nécessitent souvent la ré-écriture de certaines séquences. Elles entrainent une utilisation plus importante de l'ordinateur, puisqu'il faut exécuter plusieurs fois le programme.

c- lorsque le programme est inséré dans son contexte d'utilisation normale, les erreurs proviennent la plupart du temps d'une mauvaise connaissance du système superviseur. Là encore il faut exécuter plusieurs fois le programme.

La figure II.2 qui est empruntée aux travaux de M.PIETRASANTA sur le développement des systèmes de programmation, illustre entre autrès ces trois étapes et permet de lire le coût de la mise au point attribué à chacune d'elles. On peut également le comparer avec celui des spécifications (base) et de l'écriture du programme (code) [Réf 23, p70] .

Ce graphique attire deux remarques:

a- l'étude des temps montre que la mise au point individuelle des modules , la mise au point globale du programme et son intégration dans le système, représentent approximativement la moitié du planning. On note également que ce sont les corrections apportées à chaque unité qui sont les plus longues. Comme nous l'avons remarqué précédemment c'est la phase d'examen très minutieux des instructions.

b- L'ensemble des activités de mise au point utilise de 50 à presque 100 % des ressources allouées à la réalisation du projet. Le maximum de ce pourcentage est atteint pour les tests globaux. L'explication est le nombre important d'exécutions qui utilisent beaucoup de temps-machine.

Ces chiffres sont confirmés par d'autres études et J.D.ARON [Réf 23, p68] montre comment ils sont obtenus. Si cette documentation se rapporte à des systèmes importants (systèmes d'exploitation), on peut cependant projeter ces résultats sur des projets plus modestes. Notre but est surtout de donner un ordre de grandeur au coût de la mise au point.

II.1.1. LES FACTEURS HUMAINS.

Nous avons souligné dans l'introduction que la programmation et plus particulièrement la mise au point sont largement fonction du programmeur: certains par exemple écrivent soigneusement leur algorithme et ne tentent la première exécution que lorsqu'ils sont presque assurés du résultat, d'autres à l'inverse préfèrent commencer la mise au point dès qu'un minimum d'instructions permet un début d'exécution... Ces différences de méthodes, l'expérience de chacun, la qualification des programmeurs,provoquent des différences dans le temps d'écriture, de mise au point, d'exécution, le nombre d'instructions, etc...Si chacun a déjà pu constater ces différences il est néanmoins intéressant

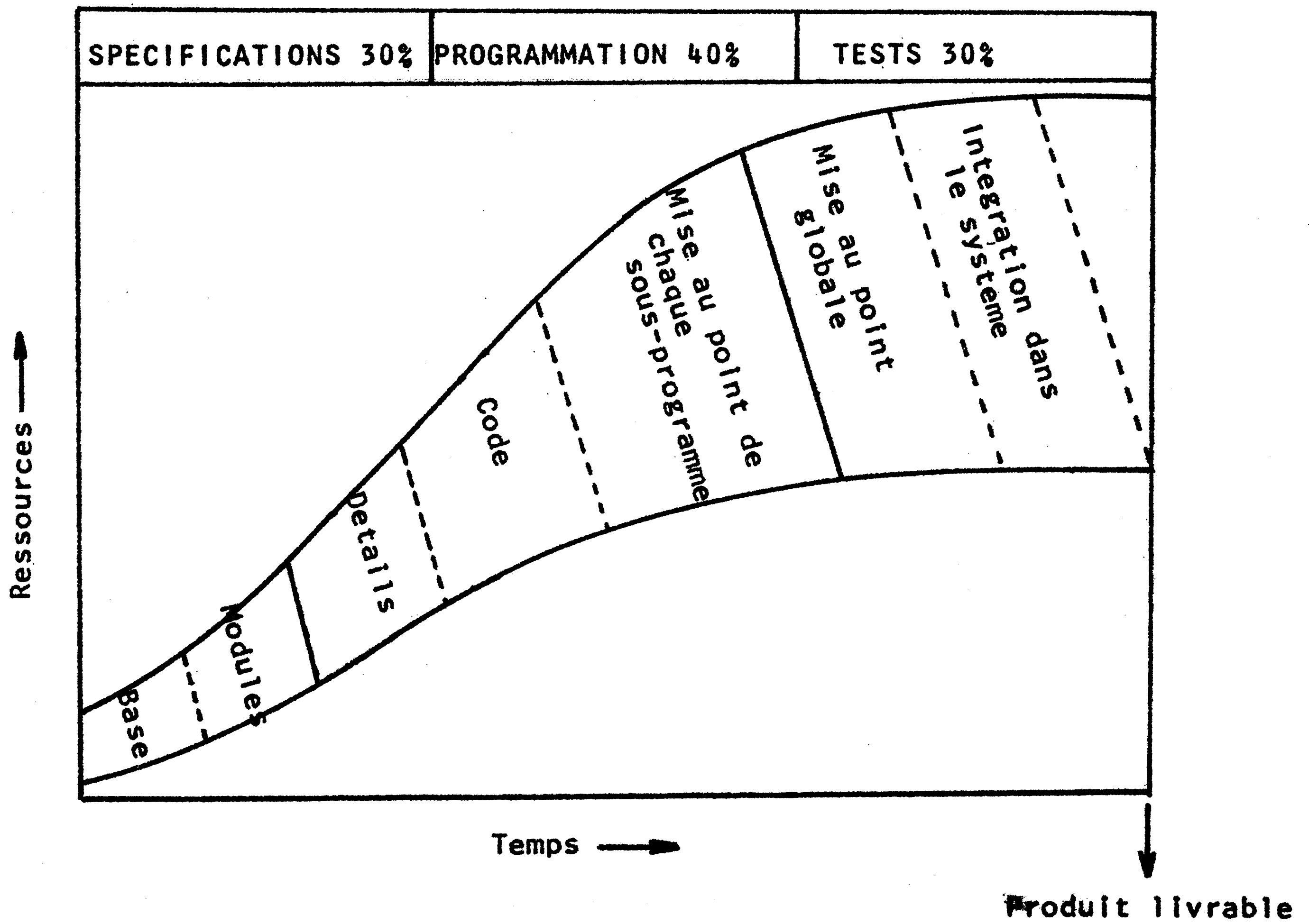


figure II.2 Estimations pour un projet de système.

(d'après A.M. Pietrasanta)

de les chiffrer. C'est ce qu'ont fait SACKMAN, ERICKSON et GRANT [Réf 20]. Les résultats présentés sur le tableau II.1 indiquent le rapport qu'il existe entre le meilleur programmeur et le moins bon pour des mesures portant sur des activités et résultats divers.

Mesure portant sur...	Rapport max.
le temps de mise au point	28/1
le temps-machine	10/1
le temps d'écriture	20/1
la taille du programme	6/1
le temps d'exécution	9/1

II.1 Ecarts de performances entre programmeurs.

L'écart est maximum pour la mise au point (28/1), ce qui justifie la remarque de l'introduction. Il montre en effet que la personnalité du programmeur, ses méthodes, son expérience, ont une influence capitale sur ses performances et particulièrement sur le temps qu'il consacre à la mise au point.

II.1.2. LES DIFFICULTES DE PROGRAMMES.

Tous les programmes ne représentent pas les mêmes difficultés de programmation et par conséquent le nombre d'instructions par homme-jour varie très sensiblement. Les principaux facteurs qui permettent d'estimer la complexité d'un programme sont le nombre d'objets différents qu'il manipule et le nombre d'intéraction qu'il possède. Plus ces nombres sont grands, plus les risques d'erreurs sont importants. Les prévisions distinguent trois

classes de programmes:

. Facile- Ce sont les problèmes d'applications classiques. Ils n'interagissent qu'avec des procédures du système-superviseur, par exemple pour gérer les données (organisation des fichiers) ou effectuer des entrées-sorties par méthodes d'accès.

. Moyen- Dans cette classe sont rangés les programmes utilitaires, tels que les compilateurs, les organisateurs de données, les méthodes d'accès, etc...

. Difficile- Les programmes jugés difficiles sont ceux qui sont très intéractifs. Tout particulièrement les modules qui partagent entre différentes tâches les diverses ressources d'un système, par exemple la mémoire ou l'unité centrale d'un ordinateur fonctionnant en "temps partagé". Dans cette partie sont également admis les superviseurs d'entrées-sorties sur les organes périphériques, et les problèmes concernés par l'occurrence aléatoire d'informations, tout précisément les systèmes de temps-réel.

Des méthodes quantitatives [Réf 23 p 74] permettent d'estimer selon la classe de chaque programme, le nombre moyen d'instructions délivrables par unité de temps. Remarquons qu'on appelle "instruction délivrable", une instruction non seulement écrite et mise au point, mais aussi parfaitement documentée dans une description logique du programme.

programme facile: 16 instructions-machine/homme-jour
programme moyen: 8 " "
programme difficile: 4 " "

En conclusion de ce premier paragraphe on peut donc remarquer que la recherche de l'algorithme d'un problème à traiter par ordinateur est une étape, mais que sa réalisation c'est-à-dire son écriture et sa mise au point en est un autre tout aussi difficile à franchir et certainement beaucoup plus coûteuse.

II.2 L'ENVIRONNEMENT DE MISE AU POINT.

On appelle généralement environnement, le milieu qui conditionne un individu, par le cadre de vie qu'il offre tant par les avantages et le côté agréable que par les contraintes. En programmation, nous donnons à ce terme le même sens: celui d'un cadre, regroupant dans celui-ci le système, le langage de programmation mais aussi et surtout des considérations telles que le niveau d'intéraction, l'assistance offerte.

L'environnement fixe souvent le genre de problèmes que l'on peut y aborder, le temps que l'on compte y consacrer. Si l'environnement est coopératif, le programmeur aura de "bonnes performances" et développera d'ambitieux projets, dans le sens inverse, il dépensera beaucoup de son temps inutilement et ses efforts seront souvent frustrés.

Nous distinguerons deux environnements de mise au point:

- l'un classique, que l'on appelle séquentiel ou "batch processing"

- l'autre plus récent qui fonctionne en mode conversationnel.

II.2.1 LA MISE AU POINT EN "BATCH PROCESSING"

Dans un environnement "batch", il faut prévoir avant l'exécution tous les cas de fonctionnement, normaux ou anormaux,

du programme et attacher à chacun d'eux des procédures de contrôle: contenu des variables, analyse de la mémoire, état du programme,etc... Ces résultats sont récupérés sur un organe de sortie, une machine à imprimer par exemple, après l'exécution. Entre ces deux instants: départ et arrêt de l'exécution,le programmeur ne peut rien modifier à ce qu'il a fait, et ce n'est que l'examen des résultats imprimés qui lui permettra de corriger son programme. En général, il devra se contenter de corriger la première erreur fatale rencontrée.

Le programmeur doit ensuite faire recompiler ou assembler son travail, le recharger et le faire réexécuter. Ce cycle de mise au point se renouvèle jusqu'à la disparition de toutes les fautes.

Comme exemple de mise au point "batch" citons TESTRAN [Réf 14] qui permet la correction des programmes écrits en assembleur et s'exécutant sous le système IBM OS/360.

II.2.2. LA MISE AU POINT CONVERSATIONNELLE.

A l'inverse du cas précédent, un système conversationnel de mise au point, permet au programmeur d'intervenir pendant l'exécution. La méthode consiste généralement à suspendre le déroulement des instructions grâce un point d'arrêt (il en existe de différentes sortes) et à permettre au programmeur, connecté grâce à un terminal, d'envoyer des requêtes qui examinent et modifient les paramètres du programme, les instructions, les variables intermédiaires, etc...

Finalement, alors que la mise au point "batch" oblige d'une part à prévoir toutes les circonstances de l'exécution et à y attacher des opérations de contrôle, d'autre part à stopper le programme sur l'occurrence de la première faute grave, la mise au point conversationnelle permet d'agir d'après les évènements et de poursuivre l'exécution même après une erreur fatale. Le bénéfice est très important.

Pourtant certains contestent le gain de la mise au point conversationnelle: ils reprochent essentiellement à cette technique d'encourager le programmeur à manquer de méthode et à considérer le terminal uniquement comme le moyen d'obtenir rapidement une exécution de leur travail. En effet, lorsqu'une erreur se produit il est tentant de faire les premières modifications qui viennent à l'esprit et de laisser à une nouvelle exécution le soin de confirmer la correction. Ce reproche met en évidence qu'un système disposant d'un terminal pour chaque utilisateur peut être inefficace s'il utilise des composants s'apparentant à un système "batch". Au contraire, si le terminal permet un véritable dialogue et fournit des réponses rapides à toutes les questions posées, on s'aperçoit que le système conversationnel permet d'optimiser la mise au point en temps et en efficacité.

Plusieurs expériences ont été menées pour comparer quantitativement ces deux méthodes. Parmi celles-ci, nous reproduisons les résultats obtenus par SACKMANN et ERICKSON, correspondants à la mise au point de deux programmes, l'un algébrique (programme 1), l'autre d'organisation de données (programme 2). Dans ces deux cas les programmeurs utilisaient les mêmes procédures de mise au point, à savoir celles du système IBM/TSS.(simulées évidemment dans le cas de mise au point "batch"). [Réf 20] .

Représentation de la mise au point en.	Programme 1		Programme 2	
	Convers.	"Batch"	Convers.	"Batch"
Nb.homme.heure	34,5	50,2	4,0	12,3
temps du calculateur(sec.)	1266	907	229	192

Les chiffres obtenus par cette expérience donnent une tendance

confirmée par l'ensemble des différentes enquêtes: le temps passé par l'homme pour mettre au point un programme est inférieur dans un environnement conversationnel, alors que le temps unité centrale augmente. On accroît ainsi l'efficacité de l'homme au détriment d'un temps machine légèrement accru; la raison est une aide plus efficace de la machine donc plus soutenue.

II.2.3. LES OUTILS DE MISE AU POINT.

Nous distinguons deux genres d'outils de mise au point: ceux qui sont utilisés pendant l'arrêt du programme, ou à son terme(outils statiques) et ceux qui fonctionnent pendant le déroulement des instructions(outils dynamiques). Il est assez facile de concevoir comment on peut fabriquer les premiers: sur la rencontre d'un point d'arrêt, l'exécution est suspendue et un environnement de mise au point est appelé. Ce programme sauve l'état du programme (registres, adresse de l'interruption, etc...) et exécute les opérations demandées; puis il restaure l'état du programme et lui rend le contrôle.

Au contraire les outils dynamiques sont plus difficiles à développer car ils nécessitent la présence d'un interpréteur qui simule l'exécution du programme testé.

II.2.3.1. Les outils statiques.

Dans ce cas l'exécution est suspendue par une interruption dûe soit à un point d'arrêt soit à une erreur de programmation.

Lorsque l'interruption est interrompue le programme est dans un certain état dont les caractéristiques sont l'adresse de l'interruption et la valeur des variables(mémoire, registres...).

Les opérations de mise au point sont donc l'examen, et la modification de cet état puis le relancement du programme.

PILOTE permet ces opérations par les requêtes:
X, GPRS, CSW, CAW, PSW: pour examiner la mémoire, les registres, l'état du programme,

STORE, SET : pour modifier la mémoire, les registres et l'état du programme.

GO : pour relancer l'exécution.

Pour faciliter ces opérations, il est intéressant qu'elles se fassent dans les termes avec lesquels le programme a été assemblé. Ainsi non seulement les variables sont désignées par leur nom symbolique (au lieu de leur adresse-mémoire), mais aussi le contenu est exprimé suivant le format défini dans le programme. Le format est constitué d'un facteur de répétition, d'un type (caractère, nombre entier, flottant, hexadécimal,...) et d'une longueur.

exemple: STORE VARIABLE 2F'4' modifie le contenu de la mémoire nommée variable. Sa valeur sera:
VARIABLE
↓
00000004 00000004.

Citons enfin dans la catégorie des outils statiques les images partielles ou entières de mémoires ("snapshot" ou "dump").

II.2.3.2. Les outils dynamiques.

Alors que les outils statiques permettaient l'examen du programme dans un état figé, les outils dynamiques offrent l'avantage de superviser l'exécution. Cette technique s'appuie sur l'interprétation.

Un interpréteur est un programme qui analyse les instructions testées et fournit un résultat identique à celui qui serait obtenu par une exécution normale. Après la phase d'analyse, l'interpréteur connaît toutes les caractéristiques d'une instruction: l'adresse, le code opération, les opérandes, les mémoires référencées, etc... Ces caractéristiques sont autant de précieux renseignements pour la mise au point.

Le prix de l'interprétation est le ralentissement de l'exécution qu'il entraîne. Aussi, il est nécessaire de pouvoir laisser le programme

s'exécuter à vitesse normale sur les parties qui ne nécessitent pas une interprétation, soit parce qu'elles sont déjà corrigées de toutes erreurs, soit parce que ce sont des routines dont le fonctionnement ne concerne pas le programmeur (programmes du système). Moyennant ceci, le coût de l'interprétation est très acceptable.

Les principaux outils fournis par l'interpréteur sont les points d'arrêt sur conditions(et même actions sur conditions), la trace des instructions, les historiques de l'exécution.

II.2.3.2.1.

Il est en général difficile de corriger une erreur qui a déjà pu perturber profondément le programme. Le moyen d'éviter ceci est d'arrêter l'exécution dès qu'on la soupçonne de présenter des conditions de fonctionnement, anormales ou seulement troubles. Pour stopper le déroulement des instructions, on utilise la technique des points d'arrêt sur conditions. Lorsque les conditions sont satisfaites, le programme est interrompu et le contrôle est donné à un module permettant l'examen conversationnel.

Quatre conditions permettent de stopper le programme:

. l'adresse de l'instruction . Un compteur peut être associé à une adresse, chaque fois que l'instruction est exécutée le compteur est décrémenté de un et son passage à zéro provoque l'arrêt du programme. La valeur du compteur est alors restauré et le programmeur peut exécuter automatiquement des requêtes qu'il a prédéfini ou/et envoyer de nouvelles requêtes sur le terminal.

C'est la requête "AT".

. le type de l'instruction. Certaines instructions peuvent interrompre le programme chaque fois qu'elles sont exécutées. Par exemple l'utilisateur peut demander de suspendre l'exécution sur les instructions de branchement.

C'est la requête "STOP INSTRUCTION".

. <u>la référence-mémoire</u>. Dans le cas, où le programmeur désire examiner plus minutieusement la façon dont est manipulé un tableau, une variable, etc... il peut demander l'arrêt du programme chaque fois qu'il y a référence à cette zone.

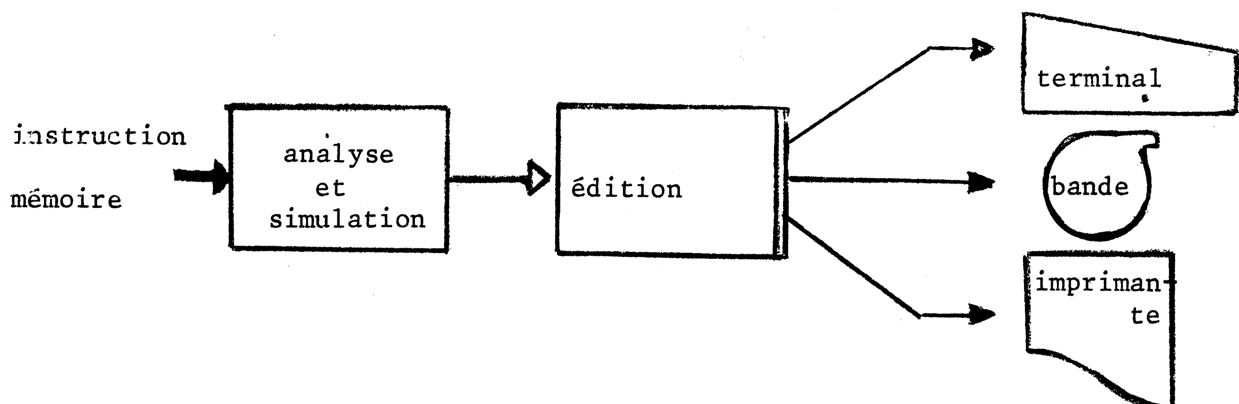
La requête est "STOP REFERENCE".

. <u>une interruption.</u> Enfin lorsqu'une interruption modifie le déroulement du programme, par exemple sur erreur de programmation, l'utilisateur peut demander l'arrêt du programme.

La requête est "ON" suivie du type de l'interruption.

II.2.3.2.2. <u>La trace des instructions</u> (Requête "TRACE") est le témoignage de l'exécution. Après la phase d'analyse de l'interpréteur, un programme d'édition reconstitue l'instruction telle qu'elle a été écrite par l'utilisateur et l'imprime soit sur le terminal, soit sur l'imprimante, soit sur bande magnétique . L'enregistrement ainsi fourni peut non seulement renseigner sur l'instruction exécutée, mais aussi sur les modifications qu'elle a entrainées: Registres, mémoires, état du programme, etc...

FIGURE II.3 <u>ANALYSE, SIMULATION ET EDITION D'UNE INSTRUCTION</u>



Cet outil de mise au point est très puissant car il fournit tous les renseignements sur l'exécution d'un programme. Son inconvénient est de ralentir considérablement l'interprétation et de produire un volume très important d'informations. C'est pour ces raisons que le format de la trace peut être choisi par le programmeur, ainsi que l'organe de sortie.

II.2.3.2.3.

Un outil de mise au point très utilisé est la collecte ou ramassage d'informations. (voir la requête "COLLECT"). La collecte est définie par un nombre d'instructions. Chaque fois qu'une instruction est exécutée, son code correspondant est rangé dans une zone de la mémoire avec éventuellement la valeur des registres à cet instant. Lorsque cette zone est pleine, l'instruction suivante viendra remplacer la plus ancienne. Grâce à cette technique, le programmeur conserve l'historique des n dernières instructions exécutées. Dans le cas d'une erreur de programmation, ou d'une fin anormale de l'exécution, l'utilisateur, par l'examen de cette collecte, peut facilement découvrir les motifs de l'erreur et effectuer les corrections nécessaires. La collecte est beaucoup plus agréable que l'image-mémoire sur papier, car elle contient des informations selon l'ordre chronologique d'exécution, ce que ne fait pas le "dump" post-mortem.

II.2.4. LA MANIPULATION DES OUTILS DE MISE AU POINT.

Suivant l'état plus ou moins avancé de la mise au point, le programmeur peut se contenter des outils qui lui sont proposés, ou construire à partir de ces dernièrs ses propres fonctions afin de répondre à des besoins particuliers.

Il est donc intéressant de distinguer deux niveaux: celui du langage de commandes élémentaires qui ne met en fonction qu'un outil à la fois, et celui du macro-langage qui permet d'assembler et d'enchaîner dans une même procédure plusieurs commandes élémentaires.

II.2.4.1. <u>Le langage de commandes élémentaires</u> est composé d'un ensemble de requêtes nommées par un mot-clé.

Exemples: STORE, TRACE, BREAK, DUMP,...

Le mot-clé est suivi d'une liste d'arguments dont la formulation est simple et peut être rendue familière à l'utilisateur par l'emploi de termes analogues à ceux du langage dans lequel a été écrit le programme testé. C'est ainsi que les mémoires d'un programme "qualifié" (voir la requête "QUALIFY") peuvent être désignées par le nom symbolique défini dans le langage source. De même son format (facteur de répétition, type et longueur), est formulé selon les conventions de l'assembleur 360.

Exemples:

STORE MOT F'4'

range dans la mémoire MOT l'entier 00000004.

X MOT+2 XL2

0004

On note également les requêtes "USING" et "DROP" qui ont la même signification que les instructions correspondantes en langage assembleur 360.

Exemples:

USING DSECTA,5

permet d'adresser la mémoire pointée par le registre 5 avec les noms symboliques définis formellement dans la "dummy section" DSECTA du programme.

DROP

libère le registre 5 de son rôle de registre de base.

II.2.4.2. Le macro-langage permet l'extension du langage de commandes élémentaires. Le langage de commandes, en raison de sa simplicité fournit des réponses rapides, mais ne permet pas de grouper plusieurs requêtes.

Pour effectuer un travail spécialisé, le programmeur est souvent amené à effectuer plusieurs fois les mêmes requêtes. C'est pour rendre ces opérations plus agréables que nous avons développé une forme de macro-langage . Grâce à ce macro-langage l'utilisateur peut créer une procédure de mise au point qui regroupe plusieurs requêtes, et qui est activable sur l'appel par nom, devenant ainsi une nouvelle requête du système.

Ce mécanisme de macro-langage permet:

- d'imbriguer plusieurs procédures entre elles.
- de paramétrer chaque procédure.
- de définir des variables globales ou locales
- de faire des tests sur des réponses aux requêtes.
- d'étiqueter des instructions et d'effectuer des ruptures de séquence soit sur le résultat d'un test soit pour exécuter une boucle.

Ce macro-langage qui permet une personnalisation des requêtes, donne lieu à une procédure qui est compilée et exécutable sur l'appel par nom. Le mécanisme de compilation et la définition du macro-langage sont décrits au chapitre III.4.

II.3. LE SUPPORT DE MISE AU POINT.

Les facilités de mise au point utilisent généralement des fonctions du système d'exploitation sous lequel elles sont disponibles -par exemple pour les opérations d'entrées-sorties- . Ces outils de mise au point se présentent sous forme d'instructions ou macro-instructions incorporées à des

endroits choisis du programme testé, et appellent des sous-programmes du système général. C'est ainsi que l'on peut demander d'imprimer la valeur des registres ou la représentation d'une partie de la mémoire chaque fois que telle instruction sera exécutée.

Exemple:

```
A       TEST     OPEN,ENTREE
        TEST     AT,MOUV
        DUMP     DATA,MEMO,MEMO+8
        ENTREE   SAVE(14,12)
                 ...
                 ...
MOUV    MVC      MEMO(8),0(5)
                 ...
                 ...
MEMO    DC       CL8'  '
                 ...
        END      A
```

Cet exemple d'utilisation de TESTRAN [Réf 14] montre comment l'on peut obtenir le contenu de la mémoire MEMO chaque fois que l'instruction MOUV est exécutée.

Le fait de définir ainsi les outils de mise au point impose des contraintes sur le programme testé, et le champ d'application.

L'insertion des outils dans les instructions oblige le programmeur d'une part à prévoir la mise au point avant l'assemblage ou la compilation, d'autre part à considérer les modifications que ces nouvelles instructions apportent à l'exécution.

Mais les contraintes sont encore plus gênantes pour l'ingénieur système qui veut soit développer un nouveau système, soit en tester des sous-programmes utilisés par les programmes de mise au point.

Dans le premier cas, l'absence de système entraîne l'absence des outils de mise au point, dans le second cas l'environnement de tests est inutilisable.

Ce sont ces inconvénients qui nous ont entrainés vers un nouvel outil général de mise au point, outil capable de "supporter" le système d'exploitation qui supervise l'exécution des programmes testés. C'est à cette catégorie de systèmes que nous donnons le nom de SUPPORT-SYSTEME ou plus simplement SUPPORT.

Dans un tel système, les fonctions de mise au point peuvent faire ou ne pas faire (dans ce cas l'indépendance est totale) parties du système supporté. Ce qu'il est important d'assurer c'est l'indépendance au niveau du fonctionnement.

La réalisation d'un tel SUPPORT conduit à résoudre des problèmes nouveaux et originaux par rapport à d'autres programmes de mise au point.

Le développement du langage de commandes d'un SUPPORT est guidé par des considérations sur la souplesse d'utilisation. Comme on ne peut anticiper ni sur la manière dont sera employé le SUPPORT, ni sur les informations que l'on en attend, ni sur le type de programmes testés, le langage de commande ne doit pas imposer des restrictions fonctionnelles en particulier du point de vue de la mémoire ou de la manipulation des organes périphériques.

Un autre aspect original d'un support est sa clandestinité vis à vis du système d'exploitation. Quand le support est utilisé, il doit sauver l'état de la machine et du programme et les restaurer lorsqu'il est abandonné.

Cette transparence nécessite l'acquisition d'une partie de la mémoire de l'ordinateur, zone qui est exclusivement réservée au SUPPORT de mise au point. Dans cette mémoire, sont implantés le programme de mise au point et les zones de travail nécessaires à son action. D'autres exemples montrant les différences entre un SUPPORT et un système ordinaire sont fournis par la gestion des interruptions et des organes périphériques.

Les interruptions qui provoquent des ruptures de séquence

de l'exécution sont générées sur les ordinateurs IBM/360 par l'ordinateur lui-même. Le SUPPORT ne pouvant contrôler ce phénomène, il lui appartient donc d'être prêt à en recevoir le signal et à le traiter avant le programme supporté. Dans le cas où l'interruption a été provoquée par le programme testé, le SUPPORT devra ensuite la lui réfléchir.(§IV.3.1.2)

Les organes périphériques créent également un problème complexe. Pour communiquer les données où les résultats de la mise au point , le SUPPORT utilise certaines unités (terminal, machine à imprimer, bande magnétique) qui sont aussi gérées par le système supporté. Il importe donc que les opérations d'entrées-sorties du SUPPORT ne modifient pas l'état du périphérique vis à vis du programme testé. Pour résoudre cette difficulté, il est indispensable que le SUPPORT exécute ces opérations grâce à un superviseur d'entrées-sorties indépendant du système supporté.(§ IV.3.2).

En contrepartie de ces difficultés, un SUPPORT de mise au point permet d'isoler tout à fait les fonctions du programme testé, de celles de la mise au point. Cette qualité offre le bénéfice de développer indépendamment la réalisation des unes sans modifier les autres, et ceci est un avantage capital pour l'ingénieur-système, avantage qui justifie la réalisation du SUPPORT PILOTE.

CHAPITRE III

LE SYSTEME REALISE: PILOTE

Le support de mise au point PILOTE a été développé sous le système de programmation CP-67/CMS [réf 15]. CP-67 est un système de partage de temps et de ressources, générant pour chaque utilisateur une machine virtuelle conforme à un modèle 360 IBM.CMS est le système-moniteur, conversationnel et mono-console, actif dans la machine virtuelle.

L'utilisation de cet environnement offre de nombreux avantages pour construire et développer de nouveaux systèmes. En particulier chaque utilisateur est libre d'user à sa guise d'un ordinateur et de plus CMS met à sa disposition un composant intéractif qui aide à la correction des erreurs de programmation: DEBUG.

Dans le désir de faciliter l'emploi de PILOTE nous avons inclus DEBUG, quelque peu modifié, dans le SUPPORT. Les avantages obtenus sont:

- les habitués de DEBUG conservent ce composant.
- nous pouvons utiliser les mêmes modules d'interrogation et de réponse pour PILOTE et DEBUG.
- nous conservons la presque totalité des requêtes de DEBUG.

Aussi avant de présenter PILOTE, nous nous proposons de faire quelques remarques sur cet environnement de CMS. Pour une présentation plus détaillée, nous renvoyons au manuel d'utilisation [Réf 15] .

III.1. L'ENVIRONNEMENT DE MISE AU POINT DEBUG.

III.1.1. ORIGINE DE DEBUG.

Cet environnement de CMS a été créé par le "Cambridge Scientific Center", avant la naissance du système lui-même. Son but était de faciliter la mise au point des premiers programmes qui allaient composer CMS. Cet objectif étant très précis et surtout immédiat, ses auteurs limitèrent volontairement leurs ambitions.

Malgrè cela il bénéficie de deux avantages importants:

- son autonomie vis à vis de CMS permet de l'utiliser dans la plupart des cas, y compris dans ceux où l'on corrige des composants appartenant au noyau du système.
- ses entrées-sorties sur le terminal, grâce à un programme indépendant du superviseur de CMS, ne modifient pas l'état de l'organe périphérique entre le moment où DEBUG prend et rend le contrôle.

Quand le système CMS fût totalement achevé, ses auteurs ont inclus DEBUG dans la partie résidente en mémoire centrale, mettant ainsi ce composant à la disposition des utilisateurs.

III.1.2. UTILISATION DE DEBUG.

Au moment où DEBUG a le contrôle, il sauvegarde l'état du programme qui s'exécutait ainsi que celui de la machine (registres, mots d'état, etc...). Cet état sera restauré à la fin des opérations de mise au point.

DEBUG peut être appelé:

a- de manière explicite par la commande "DEBUG" de CMS.

b- de manière implicite sur l'apparition de certaines conditions qui sont:

- l'occurrence d'un point d'arrêt posé par l'utilisateur
- l'occurrence d'une interruption-programme
- l'occurrence d'un interruption externe
- l'occurrence d'une erreur ou d'une situation impossible à rattraper automatiquement par le système.

Dans tous les cas, DEBUG permet alors à l'utilisateur d'envoyer à l'aide du terminal, des requêtes; celles-ci sont analysées et exécutées immédiatement. Certaines de ces requêtes permettent de relancer le programme interrompu ou de recharger dans la mémoire du calculateur un nouveau noyau de CMS.

Les requêtes de DEBUG sont :

BREAK	place à une adresse donnée un point d'arrêt.
CAW	imprime sur le terminal l'adresse du dernier programme-canal.
CSW	imprime sur le terminal l'état du dernier organe périphérique utilisé.
DEF	associe à un nom symbolique, une adresse et un nombre d'octets.
DUMP	liste sur l'imprimante l'image instantanée d'une partie de la mémoire centrale.
GO	relance l'exécution du programme à partir de l'adresse où il avait été interrompu, ou à partir d'un autre point.
GPR	imprime sur le terminal la valeur que possédaient les registres généraux avant l'interruption.
IPL	amène en mémoire la copie initiale de CMS.
ORIGIN	définit un facteur de translation pour les adresses mentionnées par la suite.
PSW	imprime le mot d'état du programme.
RESTART	a la même fonction que la requête IPL.
RETURN	permet de rendre le contrôle à CMS si DEBUG avait été appelé explicitement.

SET	modifie le contenu des registres ou des mots d'états spécifiés. (CSW, CAW, PSW).
STORE	modifie le contenu de la mémoire dont on précise l'adresse.
TIN	permet de choisir la façon dont se font les entrées-sorties sur le terminal:soit indépendantes de CMS soit par les sous-programmes du superviseur.
X	imprime sur le terminal le contenu de la mémoire dont on précise l'adresse.

III.1.3. REMARQUES SUR DEBUG.

L'utilisation de cet environnement de mise au point est pratique grâce à l'emploi de requêtes simples et fondamentales. Cette simplicité entraîne cependant des contraintes qui ralentissent l'activité du programmeur, et le laissent désarmé devant certains problèmes:

a- Dans le mode conversationnel (c'est-à-dire lorsque DEBUG a le contrôle), la simplicité des requêtes peut devenir une gêne. Par exemple il n'existe aucun mécanisme de construction de procédures qui regrouperaient plusieurs requêtes. De plus, chaque variable du programme doit être spécifiée par l'adresse hexadécimale où elle est représentée dans la mémoire. Ceci contraint le programmeur à avoir constamment sous les yeux deux listes: celle des adresses de chargement du programme et celle qui résulte de l'assemblage. De même le contenu de la mémoire est toujours exprimé en valeur hexadécimale, sans tenir compte de la façon dont elle a été originalement déclarée.(caractères, entier, etc...). Toutes ces restrictions ajoutées à l'absence des opérations arithmétiques élémentaires, gênent le programmeur et détournent son attention des problèmes véritables de mise au point.

b- Pour contrôler l'exécution du programme, on remarque que DEBUG fournit peu de moyens. S'il est vrai que sur l'apparition de chaque erreur

DEBUG est appelé, on ne se trouve pas moins dans une situation peu favorable. On ne peut alors et en général, que constater les faits; or contrôler l'exécution c'est avant tout la prévoir; pour ce faire DEBUG ne dispose que de points d'arrêts placés à des adresses fixes du programme. On comprend aisément qu'avec cet outil élémentaire, le programmeur doit prévoir tous les cas d'exécutions de son algorithme et insérer un point d'arrêt pour chacun d'eux. L'opération est longue et non triviale. Aussi il peut arriver de perdre la trace de l'exécution, la seule ressource consiste alors à demander une image-mémoire. Après l'étude de cette information qui permettra peut-être de corriger l'erreur, il sera nécessaire de recharger le programme dans la mémoire, de replacer de nouveaux points d'arrêt et de renouveler l'exécution.

Cette méthode s'apparente plus à un "batch" rapide qu'à une technique intéractive.

III.2. PRESENTATION DE PILOTE.

PILOTE appartient à la classe des supports définis au chapitre II.3. Orienté vers la mise au point et les tests de programmes, il met à la disposition de l'utilisateur un ensemble d'outils et de facilités pour contrôler l'exécution. Ces dispositifs sont manipulables au moyen d'un langage de requêtes de base, complété d'un macro-langage. Nous reviendrons sur celui-ci au chapitre III.4.

Les requêtes permettent " d'adresser la mémoire" de trois façons:

- symboliquement, grâce aux noms et aux étiquettes définis dans le programme écrit en langage source.
- avec l'adresse absolue de la mémoire occupée.
- avec l'adresse relative dont l'origine est choisie par le programmeur. Par exemple l'origine peut être le point de chargement.

Parmi les modules de PILOTE, un moniteur permet de simuler totalement le fonctionnement d'une machine sur elle-même. Grâce à ce moniteur, toutes les instructions du programme testé sont interprétées et l'utilisateur peut suivre très minutieusement le déroulement de son travail et activer des procédures particulières sur l'occurrence de conditions. (C'est ce qu'on appellera des directives).

Ce fonctionnement permet de contrôler et de prévoir l'exécution, d'en recueillir des traces et de collecter dynamiquement d'autres informations, sans modifier le programme testé.

Plus globalement, le support PILOTE constitue l'interface entre l'utilisateur qui travaille devant un terminal et les programmes qui s'exécutent dans la mémoire de l'ordinateur.

D'un côté, PILOTE dialogue avec l'utilisateur alors que le programme est interrompu: on est en mode conversationnel.

De l'autre côté, le programme s'exécute soit par simulation si le moniteur est en fonction, c'est le mode simulation, soit normalement si le moniteur est hors fonction, c'est le mode exécution. [fig III.1]

III.2.1. LE MODE CONVERSATIONNEL

Dans ce mode, l'exécution a été interrompue par un point d'arrêt (BREAK), par une erreur de programmation (interruption programme) ou par une directive (STOP). Le terminal est alors disponible pour permettre à l'utilisateur de taper une ligne. Celle-ci lue par PILOTE peut être traitée de trois façons:

a- elle est analysée et immédiatement exécutée: c'est le cas où l'état du programme et de la machine sont examinés et modifiés. C'est également celui où le programmeur donne des directives au moniteur pour lui indiquer comment réagir sur l'apparition de certaines conditions.

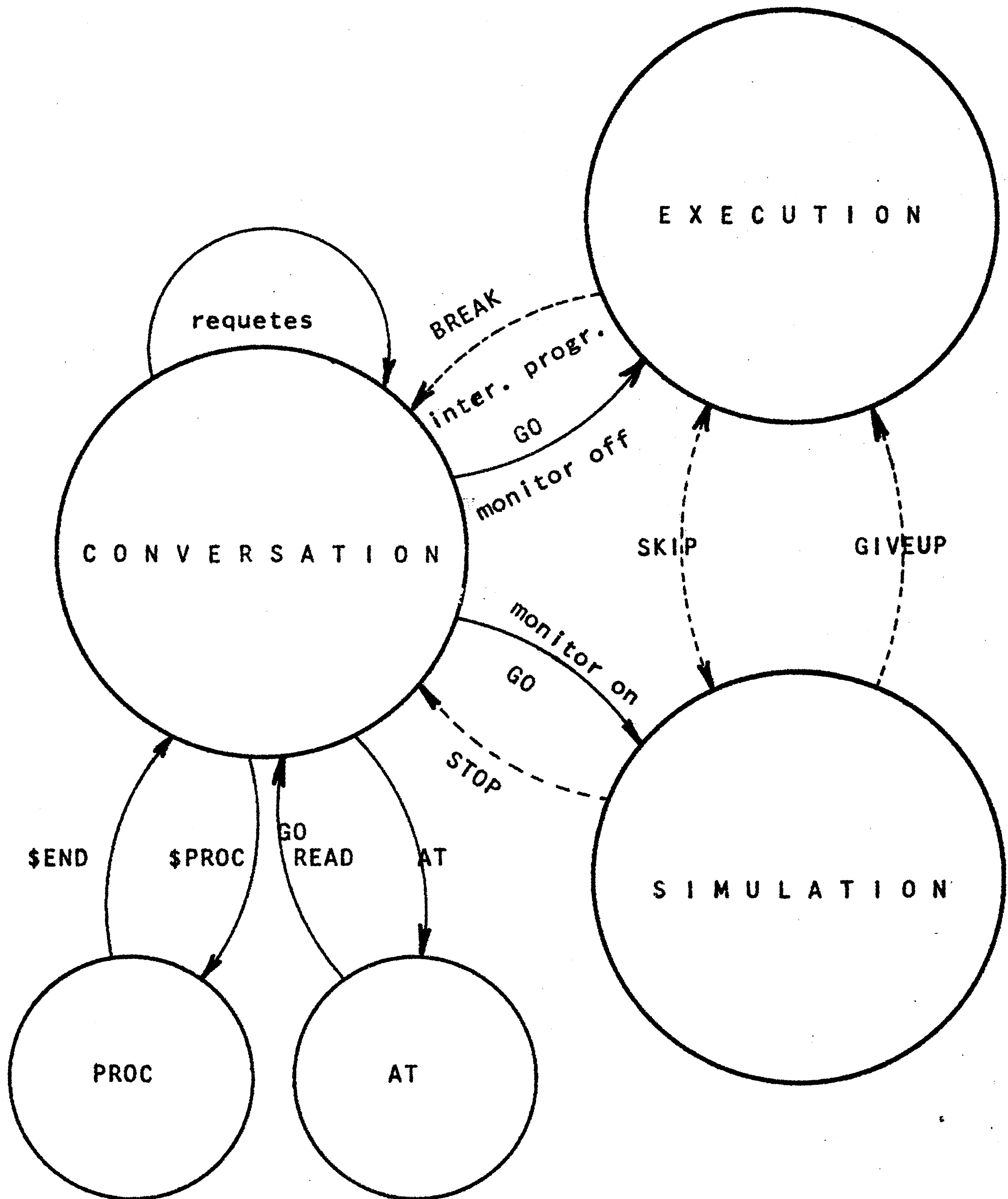
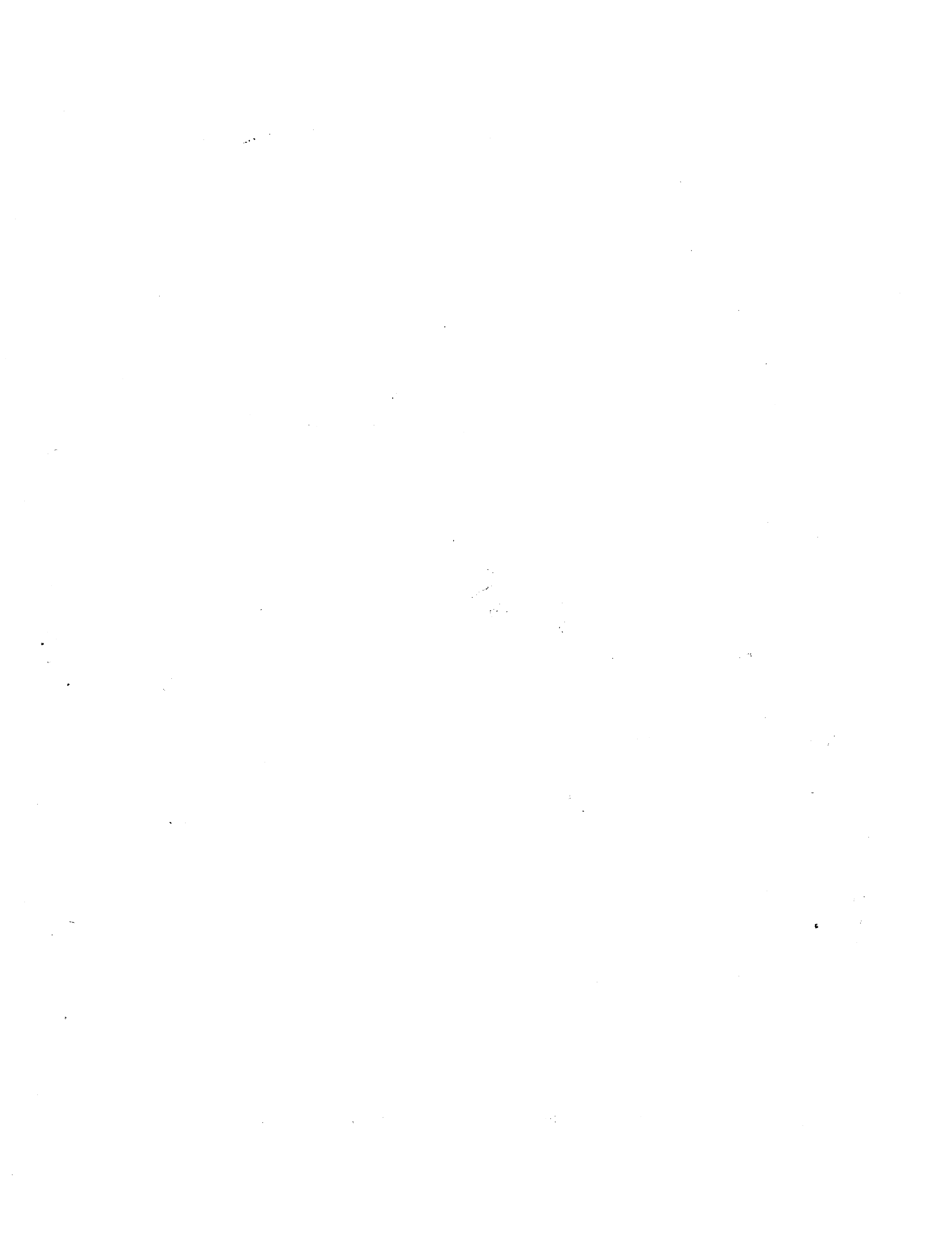


figure III.1 <u>Les modes de fonctionnement de PILOTE</u>.



exemples:

BREAKPOINT 02 AT 015002 → on passe en mode conversationnel sur l'occurrence d'un point d'arrêt (numéro 2) en 15002

gpr 2 ← l'utilisateur demande la valeur du registre 2

00015A30 → réponse

X element ← quelle est la valeur de la variable ELEMENT?

ADDR = 001204 EXEMPLE → ELEMENT est à l'adresse 1204 et contient la chaine de caractères 'EXEMPLE'

stop reference ligne ← c'est une directive: on demande d'interrompre l'exécution dès qu'il y aura une référence à la mémoire LIGNE

_ → ce signe(souligné) signifie que l'utilisateur peut envoyer une nouvelle requête .

b- elle est rangée dans une macro-requête. La requête AT permet de passer dans un sous-mode, le mode AT. Dans ce cas, la requête n'est pas immédiatement exécutée, mais rangée, avec les autres requêtes tapées en mode-AT, dans une macro-requête. L'exécution de celle-ci aura lieu avant la simulation de l'instruction qui est désignée par la requête AT(voir le paragraphe III.3.4).

exemple:

at boucle 3 provoque l'entrée dans le mode-AT
→ using entree 5
→ x element cl8 ces requêtes ne sont pas exécutées, mais rangées. Le "go"
→ go permet d'abandonner le mode-AT.
_

c- elle est compilée. Si l'utilisateur veut écrire une procédure de mise au point, il appelle le compilateur de PILOTE (cf paragraphe III.4). Les lignes qu'il tape ensuite sont compilées, au fur et à mesure et le code exécutable est placé dans une procédure baptisée par le programmeur. Ce nom devient alors celui d'une nouvelle requête élémentaire.

exemple:

```
$procedure test              ←   appel du compilateur pour créer la procédure
                                 TEST.
 >  εa = .gpr 2         )
 >  $if  εa  $eq  0  .read  )
 >  .go                 )        procédure
 >  $end                )
—                            ←   la procédure est terminée.
```

III.2.2. LE MODE SIMULATION.

On passe du mode conversationnel au mode simulation, en mettant le moniteur en fonction (voir la requête "MONITOR" III.3.4) et en demandant de relancer l'exécution (voir la requête "GO" III.1.2). Dans ce cas le programme est simulé et PILOTE fait fonctionner les outils de mise au point qui ont été activés.(traces, collectes,...). Si au cours de la simulation, certaines conditions sur le programme sont réalisées (référence-mémoire, interruption, etc...) PILOTE agira suivant l'une des quatre directives donnée par le programmeur:

NORMAL — l'occurrence de la condition ne modifie pas le déroulement de l'exécution. Celle-ci se poursuit donc normalement sous le contrôle du moniteur.

GIVEUP — la condition entraîne automatiquement l'abandon du contrôle par le moniteur. Ce contrôle est donné au programme lui-même; on passe donc du mode simulation au mode exécution.

SKIP — sur la condition, on passe en mode exécution jusqu'à ce que l'instruction qui suit la dernière (instruction) simulée soit exécutée. A ce moment on revient automatiquement en mode simulation. Ce passage provisoire d'un mode dans un autre permet d'éviter la simulation de sous-programmes qui sont appelés et qui rendent le contrôle au même point.

STOP — le programme est interrompu; son état ainsi que celui de la machine est préservé; on passe du mode simulation au mode conversationnel.

Ces directives ont été choisies pour prévenir à toutes les circonstances accidentelles du programme, et permettre au programmeur d'agir avant qu'une erreur ne vienne perturber l'exécution et compliquer la mise au point. De plus, l'utilisateur peut être tenu au courant de la façon dont s'exécute son programme, soit en temps réel (voir la requête TRACE III.3.4) soit à posteriori (requête COLLECTE III.3.4).

Enfin, en mode simulation, le programmeur peut contrôler son programme, en ne le faisant exécuter que n instructions à la fois; n allant de 1 (pas à pas) jusqu'à l'infini.(voir la requête NUMBINST III.3.4).

exemple : La trace de l'exécution est éditée sur le terminal au fur et à mesure du déroulement des instructions.

adresse de l'instruction

mémoire de l'instruction

```
009000  41782002  LA  R1 = 7   X = 8   B = 2   D = 002   (012992)
        * R7 = 00012992
INTERRUPT INPUT/OUTPUT OLDPSW=FF04000940009004 NEWPSW=00000000000005A4
0005A4 90FC0040 STM R1 = 15   R3 = 12   B = 0   D = 040 (000040)
```

description de l'instruction

III.2.3. LE MODE EXECUTION

Le programmeur peut juger que certaines parties (déjà mises au point, ou qu'il n'a pas à connaître) n'ont pas besoin d'être simulées et donc laisser le programme s'exécuter normalement: ceci évite le ralentissement produit par le moniteur. Dans ce mode exécution, le seul outil de contrôle est le point d'arrêt placé sur une instruction.(Voir la requête BREAK III.12). Si ce point d'arrêt est rencontré pendant l'exécution, ou pour toute erreur qui entraîne une interruption-programme, on revient en mode conversationnel.

III.3 MANUEL DE L'UTILISATEUR DE PILOTE SOUS CMS.

Dans le cadre de CMS, le support PILOTE se présente comme une nouvelle facilité du système, avec le fait particulier que l'utilisateur doit décider de son emploi dès le début de l'initialisation du superviseur du système. Pour résoudre les problèmes évoqués au chapitre IV, il est nécessaire en effet de charger le support dès le début de la session-CMS, et pour s'en débarrasser, de ramener en mémoire une nouvelle copie du système CMS.

Rappelons également, que l'environnement DEBUG est utilisé par PILOTE, et que si ce dernier est présent, DEBUG devient un composant du support, bien qu'ils occupent l'un et l'autre des places différentes en mémoire centrale. (Voir la figure III.2). Nous avons déjà vu les avantages de conserver DEBUG, en particulier le fait de ne rien modifier à ce qui existait déjà dans le passage entre les modes exécution et conversation.

III.3.1. ACTIVATION DE PILOTE.

PILOTE est amené dans la mémoire par la commande de CMS: "login"; il est chargé dans le bas de cette mémoire, c'est-à-dire celle

qui correspond aux plus grandes adresses. Cette opération achevée, la mémoire centrale est divisée en deux parties distinctes (figure III.2):

a- la zone non simulée dans laquelle résident tous les segments du support (à l'exception de DEBUG qui se trouve dans le noyau de CMS), y compris la mémoire libre dont il a la responsabilité exclusive et qu'il utilise pour ranger les informations à sauver: macros, collectes,....Cette zone est protégée par le moniteur et ne peut être référencée par aucune des instructions du programme simulé.

b- la zone simulée s'étend de l'adresse zéro de la mémoire centrale jusqu'à la zone non simulée. Cette partie représente la nouvelle mémoire de CMS, en particulier celle des programmes mis au point. Toute instruction simulée qui ferait référence au delà de cette limite provoquerait une erreur d'adressage.

Description de la commande login.

Objet:

Cette commande, en plus de sa signification habituelle, [Réf 15], permet de charger le support PILOTE, d'effectuer les modifications de DEBUG et de mettre à jour les constantes de CMS pour amputer la mémoire centrale de la partie occupée par PILOTE.

Format:

LOGIN < PILOTE < SLCxxxxx >>...

PILOTE est l'argument qui permet de charger PILOTE

SLCxxxxx permet de définir l'adresse de chargement de PILOTE. par défaut l'adresse (xxxxx) est calculée pour allouer 8Koctets de mémoire libre au support.

... Représentent les arguments habituels de la commande LOGIN. Rappelons que ceux-ci permettent de définir le catalogue des fichiers de l'utilisateur. [Réf 15] .

Utilisation:

L'argument PILOTE, éventuellement suivi de SLCxxxxx, doit précéder ceux qui sont utilisés pour construire le catalogue des fichiers utilisateurs. [Réf 15] .

Le résultat de cette commande est :

- Le chargement de PILOTE,
- La re-définition de la taille de la mémoire centrale,
- L'autorisation de l'emploi des requêtes de PILOTE sous DEBUG.(Apparition du signe "souligné"),
- l'exécution automatique de la requête:

  TIN DEBUG, pour effectuer les opérations d'entrées/sorties sur le terminal indépendamment de CMS.

Le support dispose d'une zone de mémoire libre qui lui est réservée en exclusivité au moment du chargement, et qu'il gère lui-même. Cette zone contient les macro-requêtes(AT) et la collecte d'informations (COLLECT). Cette mémoire libre prend place entre PILOTE et la fin physique de la mémoire centrale (voir figure III.2). Afin de conserver une certaine souplesse de fonctionnement, le programmeur peut définir la taille de la mémoire simulée au moyen de l'argument SLCxxxxx: xxxxx est l'adresse de chargement de support. Le calcul est simple:

xxxxx = TRMC - (TML - TP)

où TRMC est la taille réelle de la Mémoire Centrale (40000 en base 16 pour 256 K octets).

TML est la taille de Mémoire Libre de PILOTE (par défaut 2000 en base 16 - 8 K octets).

TP est la Taille de PILOTE fixée à 7000 en base 16.

Réponses:

PILOTE AT xxxxx - y K OF FREE STORAGE.

Ce message confirme que PILOTE est convenablement chargé dans la mémoire, à l'adresse xxxxx. La taille de la mémoire libre dont le support dispose est de y K octets.

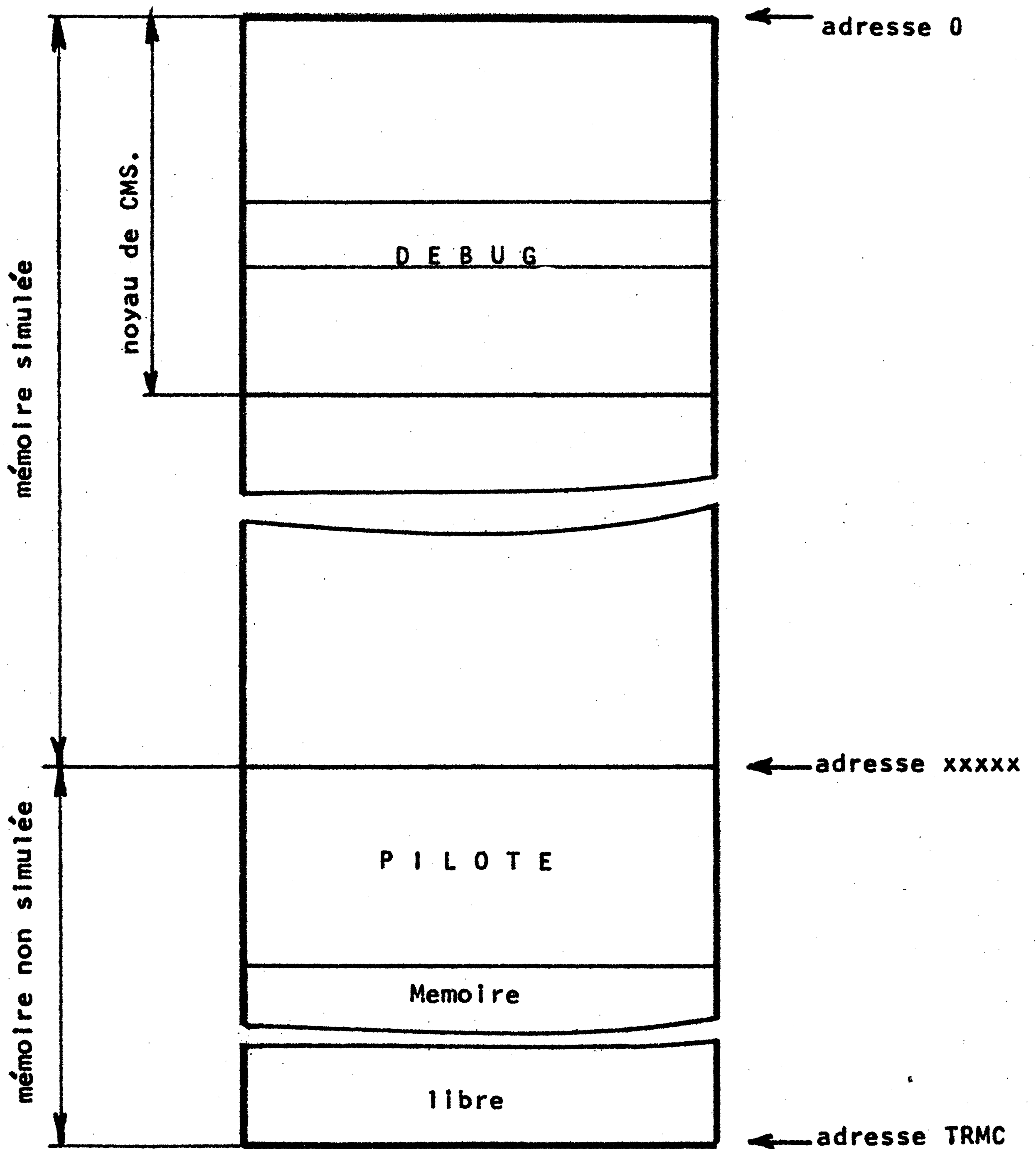


figure III.2 Organisation de la mémoire centrale.

PILOTE IS ALREADY IN CORE
Ce message d'erreur signale que la commande LOGIN PILOTE avait été précédemment envoyée.

III.3.2 ENTREES DANS LE MODE CONVERSATION.

Nous avons déjà vu que dans le mode conversationnel l'utilisateur envoie ses requêtes, et construit des procédures ou des macro - requêtes.(figure III.1).

Ce mode de fonctionnement devient disponible dans les deux cas ci-dessous:

Premier cas: le programme se déroulait en exécution pure et a été interrompu par une des conditions que nous rappelons simplement ci-dessous car elles sont aussi valables pour DEBUG (voir "CMS user's guide"Réf.15).

commande DEBUG	:	"DEBUG ENTERED..."
point d'arrêt	:	"BREAKPOINT n AT xxxxx"
erreur "hardware"	:	"HELP...HELP...MACHINE CHECK..."
interruption externe	:	"EXTERNAL INTERRUPT"
interruption-programme	:	"PROGRAM INTERRUPT PSW =..."
erreur inconnue de CMS	:	

A droite de chaque condition figure le message qui l'accompagne.

Second cas: le programme se déroulait en simulation: le moniteur de PILOTE stoppe le programme, et imprime sur le terminal la raison de cet arrêt.

"STOP REFERENCE x - y AT z "
L'exécution est stoppée car l'instruction résidant en z fait référence à la zone-mémoire d'adresses x et y. Cette référence peut être d'écriture, de lecture ou d'exécution. L'interruption est provoquée avant l'instruction. La requête qui permet d'effectuer de tels arrêts est STOP REFERENCE, suivie des deux adresses désignant la zone-mémoire.

"STOP INSTRUCTION inst.(mnémonique) AT adresse".
Le programme est interrompu sur l'instruction indiquée dans le message par son code (inst) et son mnémonique. L'adresse où figure cette instruction est également précisée. Comme précédemment, le programme est interrompu avant l'exécution de l'instruction.

Exemple: STOP INSTRUCTION D2 (MVC) AT 12000

"STOP INTERRUPT x "
Sur l'occurrence du type d'interruption x, le programmeur peut demander l'arrêt de la simulation pour passer en mode-conversation. Le message indique que l'interruption a eu lieu.

Exemple: STOP INTERRUPT SVC

"AT adresse compteur"
Ce message indique au programmeur que l'instruction à l'adresse indiquée va se réaliser pour la nième fois (n = compteur). Les requêtes contenues dans la macro-requête correspondante vont être automatiquement exécutées. Chaque requête entraîne l'écriture de la "requête" et des informations qui lui sont attachées. Si la liste des requêtes se termine par READ le programmeur peut taper de nouvelles lignes, sinon la simulation se poursuit.(voir la requête AT).

Exemple:

```
AT 15002 02      ← la 2ème fois que l'on exécute l'instruction
                   en 15002 la macro-requête va s'exécuter.
GPR 2
00000001         ← contenu du registre 2
SET GPR2 0001AB40 ← modification de ce registre
X BOUCLE         ← quel est le contenu de la mémoire boucle?
ADDR= 012000  000001A0  ← Réponse
READ             ← on désire envoyer de nouvelles lignes.
-                ← le support attend une ligne.
```

Exemple:

AT TEST 01
DUMP TABLEAU TABLEAU FINTABLE
GO

Dans cet exemple, chaque fois que l'on exécutera l'instruction TEST, on imprimera, sous le nom de TABLEAU, la mémoire de TABLEAU à FINTABLE. . L'exécution est automatiquement relancée après cette requête.

Enfin, le moniteur peut interrompre le programme, lorsqu'un quantum d'instructions est atteint. Ce quantum est fixé par la requête NUMBINST. Ce principe autorise aussi bien le pas à pas(une instruction exécutée à la fois) que la simulation continue. Lorsque le quantum est satisfait, un signe souligné, seul, indique que le terminal est disponible pour recevoir des requêtes.

III.3.3.ABANDON DU MODE-CONVERSATION.

Là encore plusieurs cas sont à envisager. On peut passer du mode-conversion soit pour réinitialiser la mémoire centrale, soit pour relancer le programme en exécution normale (mode-exécution), soit enfin pour simuler le programme (mode-simulation).

Pour revenir dans l'environnement de CMS ou pour réinitialiser la mémoire centrale on utilise les requêtes classiques de DEBUG, déjà citées en III.1.1 et détaillées dans le manuel de CMS [Réf 15] . Ce sont:

IPL
KX
RESTART
RETURN

La réactivation du programme est demandée par la requête GO,

suivie ou non d'une adresse selon que le programmeur désire commencer l'exécution à partir d'un point donné ou à partir du point d'interruption.

A partir de cette requête, le programme peut soit s'exécuter normalement (mode-exécution) si le moniteur est non-prêt (MONITOR OFF) soit s'exécuter par simulation (mode-simulation) si le moniteur est prêt (MONITOR ON). (Voir la figure III.1).

III.3.4 LE LANGAGE DE REQUETES.

Une requête est la totalité des informations contenues dans la ligne que tape le programmeur sur son terminal. Elle se compose d'un mot-clé (ordre) -la liste figure ci-dessous- qui est le nom de la requête et d'une suite d'arguments, le blanc étant le séparateur.

Deux types de requêtes sont à considérer:

a- les requêtes attachées au mode-conversation: l'utilisateur est averti que ce mode est actif par le signe "-", et les réponses sont immédiates.

b- les requêtes attachées au mode-AT: dans ce cas le signe " → " est imprimé et l'exécution de ces requêtes est différée.

Chaque mot-clé de la requête admet une abréviation minimum, une seule lettre peut suffire s'il n'y a pas de confusion possible; toutefois, les mots-clés étant rangés par ordre alphabétique, c'est le premier commençant par cette lettre qui sera sélectionné: Par exemple, la lettre "a" représente la requête "add" ; par contre les deux lettres "a" et "t" sont nécessaires pour le mot-clé "at".

Certaines requêtes admettent comme arguments une liste d'options. Dans cette liste, aucun ordre n'est défini et en cas de contradiction, c'est le dernier argument qui est pris en compte.

Exemple:

TRACE ON OFF est équivalent à TRACE OFF.

En outre, la vérification sur les arguments se fait toujours sur quatre caractères.

Dans la suite de ce paragraphe, nous détaillons les requêtes de PILOTE, de la manière dont peut le souhaiter un utilisateur: la fonction de la requête, son format syntaxique, son utilisation et les réponses qu'elle retourne (y compris les messages d'erreurs).Nous n'étudions pas ici les requêtes qui apparaissent déjà dans DEBUG, nous renvoyons pour celles-ci le lecteur au manuel d'utilisation de CMS [Réf 15] . Nous attirons simplement l'attention sur quatre points:

a- la requête "GO" a maintenant une double signification suivant l'état du moniteur (voir le paragraphe III.3.3). Signalons également que "taper une ligne vide" (simple retour de chariot) a le même effet que "GO" si le moniteur est prêt.

b- la requête "DEF" qui existait dans DEBUG a été supprimée et avantageusement remplacée par QUALIFY et USING.

c- grâce aux procédures de conversions de PILOTE, l'utilisateur peut préciser le format dans lequel il désire voir imprimer ou modifier la représentation de la mémoire:

Exemples:

STORE 12000 C'EXEMPLE'
Range en 12000 la chaîne de caractères indiquée.
X 15000 3F
Examen de la mémoire d'adresse 15000 sous forme de trois mots hexadécimaux.(nombres entiers).

d- si l'utilisateur ne désire pas ajouter aux adresses qu'il précise la valeur de l'origine, il doit faire suivre chacune de celles-ci de "A".

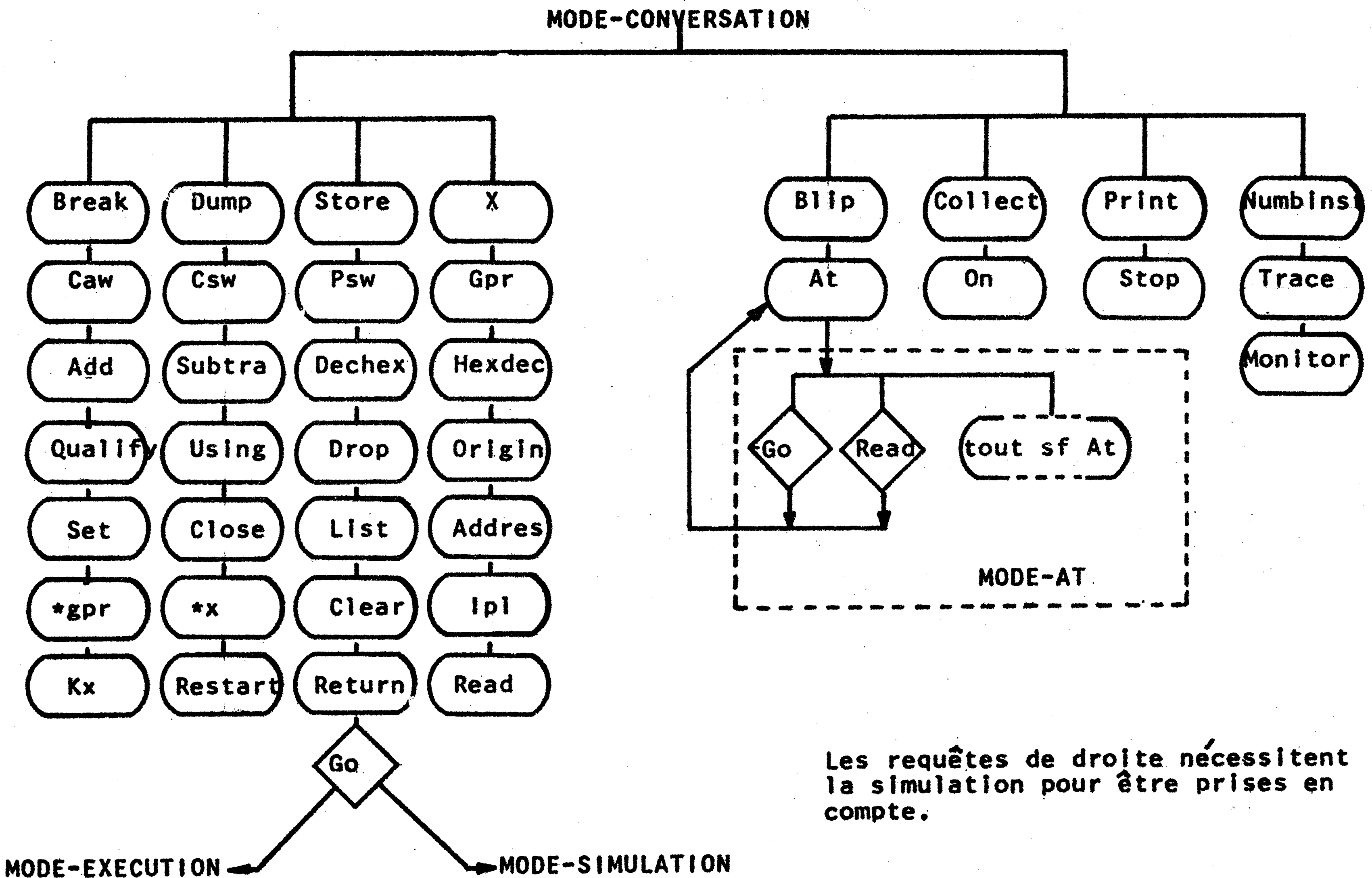
Exemples:

```
ORIGINE  12000
X  100.A     il s'agit de l'adresse absolue 100.
X 100        il s'agit de l'adresse absolue 12100.
```

Liste des requêtes de PILOTE

ADD	effectue l'addition de deux nombres.
ADDRESS	fournit l'adresse-mémoire d'une variable du programme.
AT(.)	permet la définition d'une macro-requête.
BREAK	voir DEBUG (III.1).
BLIP	compte les instructions.
CAW	voir DEBUG(III.1).
CLEAR	annule certaines requêtes.
CLOSE	ferme les unités de sortie.
COLLECT	permet de conserver des informations.
CSW	voir DEBUG(III.1)
DECHEX	assure les conversions de décimal en hexadécimal.
DROP	supprime l'effet de USING.
DUMP	voir DEBUG(III.1).
GO	voir DEBUG(III.1).
GPR	voir DEBUG(III.1).
HEXDEC	assure les conversions de l'hexadécimal en décimal .
IPL	voir DEBUG(III.1).
LIST	fournit la liste des requêtes tapées.
MONITOR	change l'état du moniteur.
NUMBINST	indique le nombre d'instructions à simuler.
ON	définit l'action du moniteur sur interruption.
ORIGINE	voir DEBUG(III.1).
PRINT	définit l'unité de sortie des informations.
PSW	voir DEBUG(III.1).
QUALIFY	qualifie les noms et étiquettes d'une section de contrôle.
READ	permet de lire une requête sur le terminal.
RESTART	voir DEBUG(III.1).
RETURN	voir DEBUG(III.1).
SET	voir DEBUG(III.1).
STORE	voir DEBUG(III.1).

SUBTRACT	effectue la soustraction de deux nombres.
TIN	voir DEBUG(III.1).
TRACE	permet de tracer les instructions exécutées.
X	voir DEBUG(III.1).
*GPR	imprime sur le terminal le contenu de la mémoire pointée par un registre.
*X	comme *GPR, mais la mémoire est adressée par un pointeur.



Les requêtes de droite nécessitent la simulation pour être prises en compte.

figure III.3 <u>Les requêtes de PILOTE</u>.

ADD

Format:

ADD	ophex1	ophex2

ophex1 et ophex2 sont deux mots hexadécimaux.

Utilisation:

Cet ordre exécute l'addition hexadécimale de deux nombres signés.

Réponse:

Le résultat apparaît sur le terminal sous la forme d'un mot.

Exemples:

add 10b 18	addition de 2 nombres positifs
00000223	c'est le résultat
add 80000abc b	addition d'un entier négatif avec un entier
80000ac1	positif.

ADDRESS

Objet

Cette requête retourne l'adresse absolue de l'argument qui est spécifié.

Format:

```
ADDRESS loc
```

loc est soit un nom symbolique du programme, soit une adresse hexadécimale relative ou absolue.

Utilisation:

Cette commande fournit l'adresse absolue hexadécimale de l'argument. Si c'est un nom qui est spécifié, celui-ci doit être qualifié.(voir "QUALIFY" et "USING") Dans le cas où il s'agit d'une adresse relative, la valeur de l'origine est ajoutée. S'il s'agit d'une adresse absolue, la réponse n'est évidemment qu'une confirmation.

Réponse:

NOT FOUND
le nom symbolique n'a pas été qualifié.

INVALID ARGUMENT
l'argument comporte, une erreur, par exemple une adresse relative non hexadécimale.

si aucune erreur n'est commise, la valeur absolue de l'adresse est retournée.

Exemple:

address elem
ADDR 012002
012002 est l'adresse absolue de la variable ELEM qui est définie dans le programme et qui a été précédemment qualifiée.

address 800.r

ADDR 000900

l'origine était 100, l'adresse absolue est donc 900

address 900.a

ADDR 000900

C'est la confirmation de l'adresse absolue 900.

address axde.r

INVALID ARGUMENT

AXDE n'étant pas une adresse hexadécimale, l'erreur est signalée.

AT

Objet

Cet ordre permet de grouper un certain nombre de requêtes dans une macro-requête. L'exécution de cette macro a lieu avant la simulation de l'instruction qui se trouve à l'adresse précisée.

Format:

```
AT        loc          < n >
```

loc est une adresse symbolique ou hexadécimale (absolue ou relative).

n est un entier décimal (n = 1 par défaut).

Utilisation:

Pour qu'une macro-requête soit prise en compte, il faut que le moniteur soit en fonction.("MONITOR ON").

Avant de simuler l'instruction qui se trouve en "loc", le moniteur examine si une macro lui correspond. Si oui, il décrémente de un le nombre décimal et s'il est nul, la macro est exécutée.
La macro est conservée et son compteur est restauré après son exécution. Grâce à ce compteur on peut choisir dans une boucle le numéro du passage pour exécuter la macro, évitant ainsi une manipulation fastidieuse des points d'arrêt.

Si l'on tape "AT.", les requêtes contenues dans la macro et le message avertissant que l'on exécute une macro ne seront pas imprimés au moment de l'exécution.

Une macro est un ensemble de requêtes de PILOTE. Lorsque le programmeur envoie la requête AT..., PILOTE ouvre une macro-requête et les prochaines lignes ne seront pas exécutées. Nous

en sommes avertis grâce à une petite flèche. Lorsque l'on est dans cette situation les lignes sont sauvées dans la mémoire libre de PILOTE sans aucune vérification de leur syntaxe. C'est au moment de leur exécution que l'on vérifiera si elles sont correctes ou non. Toutes les commandes de PILOTE sont alors valides sauf "AT".

Enfin pour fermer la macro, deux requêtes sont possibles:

-GO avec ou sans adresse qui a pour effet de relancer la simulation à l'adresse indiquée
-READ qui permet de repasser en mode conversationnel. Le clavier est alors déverrouillé, et l'utilisateur est invité, par une flèche, à envoyer de nouvelles requêtes, ou à relancer son programme.

Réponse:

NO MORE 8 STATEMENTS
signifie qu'il y a déjà 8 macros de définies. C'est le nombre maximum. Pour en créer une nouvelle, il faut en annuler une autre.

FREE STORAGE IS FULL
Ce message indique que la mémoire libre de PILOTE est pleine; la macro n'a pas été créée.

Exemples:

```
at. operat 50
→   psw
→   csw
→   caw
→   read
```

Toutes les 50 fois que l'instruction OPERAT sera exécutée, le PSW, le CSW et le CAW seront imprimés sur le terminal et le clavier sera déverrouillé.
La réponse pendant la simulation est:

```
FF0400CA04015048
000002080C000000
00000200
```

```
at  12038
→   gprl 11
→   trace on modg allinst tail
→   go
at  1203e
→   trace off
→   go
-
```

Ces deux macros permettent de montrer le registre 11 avant d'exécuter l'instruction en 12038, et de tracer toutes les instructions suivantes. Cette trace s'arrêtera lorsque l'opération en 1203e sera simulée. Cette macro est exécutée sans arrêt de la simulation.
Pendant la simulation, la réponse est:

```
AT 12038 01
GPR 11
000004d5
TRACE ON MODG ALLINST TAIL
GO

012038     1859        LR    R1=5    R2=9
      *R5 0000008
01203A     1C45        MR    R1=4    R2=5
      *R5 0000009
0123C      195B        CR    R1=5    R2=11
***PSW CHANGES                        CONDITION CODE=2
AT 12038  01
TRACE OFF
GO
```

BLIP

Objet

Cet ordre envoie périodiquement un message sur le terminal.

Format:

BLIP n

n est un nombre décimal.(n = 1000 par défaut).

Utilisation:

Chaque fois que le simulateur aura traité n instructions, il informera l'utilisateur par un message sur la console. Cette commande peut donc être utilisée pour compter les instructions qui sont simulées, ou encore pour faire prendre patience à l'utilisateur devant sa console.

Réponse:

Aucune réponse si la requête est correcte. Mais au moment où 1000 instructions, par exemple, ont été simulées, le message suivant sera imprimé sur la console:

1000 INSTRUCTIONS HAVE BEEN EXECUTED

CLEAR

Objet:

Cet ordre permet d'effacer certaines requêtes qui ont été précédemment définies.

Format:

CLEAR	REFERENCE	< loc1 < loc2 >>
	INSTRUCTION	< cophex >
	AT	< loc >
	COLLECT	

REFERENCE annule la requête STOP REFERENCE qui correspond aux adresses loc1 loc2. Si aucune adresse n'est indiquée tous les stops sur référence seront annulés.

INSTRUCTION annule la requête STOP INSTRUCTION sur le code opération indiqué. Si aucun code n'est indiqué toutes sont effacées.

AT annule la macro-requête indiquée, si l'adresse est absente, toutes les macros sont effacées.

COLLECT efface toute trace de la collecte déjà existante.

Utilisation:

Les commandes de PILOTE sont permanentes. On ne peut donc les annuler que par un ordre "CLEAR". Cet ordre permet aussi de libérer une partie de la mémoire libre pour définir de nouvelles macro -requêtes ou une nouvelle collecte.

Réponse:

NOT FOUND

Ce message apparaît si PILOTE n'a pas trouvé ce qu'on lui demandait.

Exemples:

Clear reference 12000 12003

Cette requête annule le stop sur référence d'adresses correspondantes.

clear at

Cette requête annule toutes les macros AT et libère la place qu'elles occupaient.

CLOSE

Objet

Cet ordre ferme les unités qui ont été utilisées pour indiquer les résultats de la simulation.

Format:

```
CLOSE   | PRINTER                 |
        | TAPE      < REWIND >    |
```

PRINTER représente l'imprimante d'adresse OOE
TAPE représente la bande d'adresse 181
REWIND permet de ré-embobiner la bande

Utilisation:

Lorsque l'utilisateur décide que l'ensemble des renseignements qu'il a obtenu constitue un fichier, il doit fermer celui-ci.

S'il s'agit de l'imprimante, on obtient alors la sortie.
S'il s'agit d'une bande, une marque de fin de fichier est écrite sur celle-ci, avec optionnellement un ré-embobinage.

Réponse:

UNIT NOT OPERATIVE
Ce message est envoyé sur la console, si PILOTE n'a pas trouvé l'unité.

COLLECT:

Objet:

Cet ordre permet de sauver un certain nombre d'informations sur le déroulement de l'exécution.

Format:

```
COLLECT | ON     < opt1    < opt2 >>    |
        | OUT                           |
```

OUT permet de sortir le contenu de la collecte sur l'unité définie par la requête "PRINT".

ON définit une zone de mémoire pour sauver les informations demandées. La taille de cette zone dépend des options qui suivent:

n est un nombre décimal indiquant le nombre d'instructions que l'on veut sauver. (par défaut n = 50).

GPRS Cette option permet de sauver avec l'instruction la valeur des registres généraux après l'exécution de celle-ci.

NOGPRS est l'option par défaut, seule l'instruction est sauvée.

Utilisation:

Dans une collecte on a les n dernières instructions simulées et éventuellement la valeur des registres après chacune de celles-ci. Le but de cette collecte estde fournir au programmeur une information sur l'exécution de son programme même au cas où il serait engagé dans une situation sans issue, sans retour arrière possible, c'est une sorte de trace post-mortem.

La collecte est représentée en mémoire sous la forme d'un ensemble de données ayant chacune le format suivant:
32 octets imprimables qui renseignent sur l'instruction (format "trace fast").
+64 octets si les registres ont été demandés.
De part l'occupation mémoire de la collecte, il ne peut en exister qu'une seule à un instant donné. Avant d'en créer une nouvelle, il faudra donc effacer celle qui pouvait déjà exister.
Lorsque la collecte est imprimée, ce sont les dernières instructions qui sortent les premières. On peut arrêter la liste sur la console, en appuyant sur la touche "attention", entre 2 instructions. (voir l'appendice B).

Réponse:

ALREADY DONE
Si on fait "COLLECT ON" avant d'avoir fait "CLEAR COLLECT"

NOT FOUND
Si on demande "COLLECT OUT" alors qu'aucune collecte n'a été définie.

FREE STORAGE IS FULL
Si la collecte demandée ne peut être contenue dans la mémoire libre. La collecte n'est pas créée. On peut redemander COLLECT ON avec des quantités plus modestes.

n INSTRUCTIONS HAVE BEEN PRINTED
Ce message indique que PILOTE a fini d'imprimer la collecte qui se composait de n instructions.

Exemples:

collect on 20 gprs
Cette requête va réserver de la place en mémoire libre pour contenir une collection de données de 20 instructions.

DECHEX

Format:

```
DECHEX    opdec
```

Utilisation:

Cette requête convertit un nombre entier positif décimal en hexadécimal.

Réponse:

Si la requête est acceptée, le résultat est imprimé par le terminal.

Exemple:

dechex 567
000001C8

DROP

Objet

Cette requête permet d'annuler l'effet du dernier USING envoyé.

Format:

```
DROP
```

Utilisation:

Lorsque le registre spécifié dans une précédente commande USING, ne pointe plus la zone-mémoire qui sert de référence, le programmeur peut supprimer cet adressage par la requête DROP. L'effet obtenu est donc identique à l'instruction d'assembleur de même nom.

Il est inutile d'indiquer le numéro du registre, puisque un seul registre peut servir de pointeur à un instant donné.

Réponse:

NOT FOUND
C'est le cas où aucune requête USING n'a été spécifiée antérieurement.

Si la commande est correcte, il n'y a pas de réponse.

HEXDEC

Format:

```
HEXDEC    ophex
```

Ophex est un entier hexadécimal sans signe.

Utilisation:

Cette requête effectue la conversion du nombre hexadécimal ophex en un entier décimal sans signe.

Réponse:

Le nombre converti est imprimé sur le terminal.

Exemple:

hexdec
00065524

LIST

Objet

Cette requête permet de rafraichir la mémoire du programmeur sur celles qui précèdent.

Format:

```
LIST    | AT          < loc > |
        | REFERENCE           |
        | INSTRUCTION         |
```

AT	permet de lister le nom et le compteur correspondant, des macro-requêtes qui ont été définies. Si une adresse est indiquée, c'est le contenu de la macro ainsi désignée qui est listé sur le terminal.
REFERENCE	liste tous les arrêts sur référence qui ont été spécifiés.
INSTRUCTION	liste les codes opérations qui provoqueront un arrêt de la simulation.

Utilisation:

Cette requête est souvent utile pour se souvenir des ordres précédemment donnés. Rappelons ici qu'une requête est valide tant qu'elle n'a pas été effacée avec l'ordre "clear".

Réponse:

NOT FOUND

si la (les) requête(s) n'existe(nt) pas

si les arrêts sur références ou instructions sont trouvés, ils sont listés sur le terminal.

Exemples:

```
list reference
STOP REFERENCE 12000-12003
               14000-15000

list  instruction
STOP INSTRUCTION D2 (MVC)
                 47 (BC)
                 9C (S10)

list at
AT 12000 01
AT 15038 08

list at 15038
GPR 0  2
TRACE ON ALLGPRS
READ
```

MONITOR

Objet:

Cette requête définit l'état du moniteur.

Format:

```
MONITOR  | ON  |
         | OFF |
```

ON   active le moniteur

OFF   le désactive

Utilisation:

Le fait de charger PILOTE en mémoire donne simplement accès aux commandes mais n'active pas le moniteur.
Pour pouvoir lancer la simulation de l'exécution au moment où l'on sortira de l'état-conversation, il est nécessaire que le moniteur soit prêt, il faut donc envoyer la ligne "MONITOR ON".
Pour éviter la simulation il faut désactiver le moniteur donc: "MONITOR OFF".

Note: Rappelons que PILOTE signale sa présence par le signe "souligné" lorsque l'on est dans l'état conversation. Si le moniteur est prêt, un retour de chariot suffit à l'activer à partir de l'adresse où l'on était arrivé. S'il était dormant on reste dans l'état-conversation.

<u>NUMBINST</u>

Format:

NUMBINST	FLOW
	n

FLOW indique un nombre illimité.

n est un nombre décimal indiquant le nombre d'instructions à simuler.n est égal à 1 par défaut (pas à pas).

Utilisation:

Cette requête permet d'indiquer à PILOTE le nombre d'instructions que l'on désire simuler. Lorsque ce nombre est atteint on revient dans l'état-conversation (signe "souligné").
Cette requête n'agit que lorsque le moniteur est en fonction. (MONITOR ON).

ON

Objet:

Cette requête permet d'indiquer au moniteur la manière de poursuivre après avoir réfléchi une interruption.

Format:

```
ON        | EXTERNAL        opt1   < opt2 >|
          | SVC                            |
          | PROGRAM                        |
          | MACHINE                        |
          | I/O                            |
```

Le premier argument indique le type de l'interruption.
opt1 et opt2 sont des options dont la liste est la suivante:

GIVEUP — permet d'abandonner la simulation. L'exécution au programme se poursuit alors normalement.

SKIP — permet d'exécuter normalement la routine de traitement de l'interruption et de reprendre la simulation lorsque l'on réexécutera l'instruction interrompue.

STOP — permet d'entrer dans l'état conversation si l'interruption indiquée se produit.

NORMAL — l'interruption est réfléchie normalement et la simulation se poursuit à l'adresse du nouveau PSW.

TRACE — l'interruption est tracée sur l'unité de sortie du moniteur.

NOTRACE — l'interruption n'est pas tracée.

Utilisation:

Cette requête n'a d'effet que lorsque la simulation est en fonction.
Ces différentes options ont pour but de donner au programmeur la plus grande souplesse sur le traitement des interruptions. En particulier il dispense celui-ci de la perte de temps que pourrait entrainer la simulation des routines de traitement d'interruptions qui sont déjà mises au point.

Les options par défaut sont:

ON EXTERNAL GIVEUP NOTRACE	ce qui permet, sous CMS, de revenir sous l'état-conversation.
ON SVC STOP TRACE	on peut ainsi contrôler les paramètres au moment de l'appel du superviseur.
ON PROGRAM GIVEUP NOTRACE	là encore, ceci permet, de revenir, sous CMS à l'état conversation ou à la routine (SPIE) de l'utilisateur.
ON MACHINE STOP NOTRACE	
ON I/O SKIP NOTRACE	ainsi, les routines de fin d'entrées/sorties de CMS n'apparaissent pas dans la simulation du programme utilisateur.

Réponse:

Aucune, mise à part les messages d'erreurs ordinaires.

Exemple:

ON PROGRAM TRACE STOP : si une interruption-programme arrive, on ne rendra pas le contrôle au système, mais directement à PILOTE. Les SPIES existantes ne seront pas prises.

PRINT

Objet:

Cet ordre définit l'unité de sortie des résultats de la simulation.

Format:

```
PRINT   | ONLINE  |
        | OFFLINE |
        | TAPE    |
```

ONLINE représente le terminal du programmeur.
OFFLINE représente l'imprimante d'adresse OOE.
TAPE représente la bande d'adresse 181.

Utilisation:

On peut sortir sur ces unités, la collecte, la trace des instructions ou des interruptions.
Ces 3 unités ont chacune leurs propriétés particulières:

La console permet de faire du pas à pas et d'avoir ainsi un contrôle très rigoureux de l'exécution de son programme. Elle est en général utilisée pour les parties de code qui demandent une mise au point méticuleuse.

L'imprimante permet une simulation plus rapide, mais le programmeur a alors un contrôle moins strict. Cette unité sert donc à tracer de longues séquences de code ou des boucles.

La bande permet une vitesse de simulation aussi rapide que

l'imprimante. Mais de plus, avec l'utilisation de l'utilitaire TAPPILOT, on peut créer avec une partie ou toute la bande, un fichier CMS. On aura donc intérêt à utiliser la bande dans les cas où l'on ignore le nombre d'instructions exécutées, et où l'on veut choisir à posteriori les instructions intéressantes.(appendice D).

Réponse:

Aucune, à part les erreurs standards.

QUALIFY

Objet:

Cette requête permet d'utiliser le nom des variables définies dans le programme source, comme adresse. De plus, le type et la longueur de celles-ci sont également accessibles.

Format:

```
QUALIFY   name
```

name est un nom du programme source qui est déclaré "externe".

Utilisation:

Pour utiliser comme adresses, les noms définis dans le programme source, ainsi que la longueur et le type des mémoires qu'ils désignent, il faut remplir deux conditions.

1) L'assemblage du programme doit être fait avec l'option "test". Ceci permet d'obtenir en plus du fichier TEXT habituel, un fichier SYM qui décrit tous les symboles du programme source.

2) Avant de charger le programme, un fichier SYMBOL devra avoir été créé (appendice C), pour organiser et condenser le fichier SYM.

La commande QUALIFY, permet alors à PILOTE, de créer en mémoire, à partir de la table de chargement du programme, et du fichier SYMBOL une table de références. Celle-ci permet de connaître les adresses absolues, les types et les longueurs des variables appartenant à la section de contrôle contenant

le nom spécifié dans la commande QUALIFY et qui ont une adresse supérieure à celui-ci.
Les variables qui peuvent s'adresser symboliquement sont dites qualifiées.
Si l'on désire qualifier toutes les variables d'une section de contrôle, on doit donc indiquer le nom de celle-ci.
Remarquons qu'une seule section de contrôle peut être qualifiée. Une deuxième commande QUALIFY annule donc l'effet de la précédente.

Réponse:

NOT EXTRN NAME
Ce nom n'a pas été déclaré "externe", ou n'existe pas.

FILE SYMBOL NOT FOUND
aucun fichier SYMBOL n'a été créé, il n'est donc pas possible d'utiliser cette commande.

si la commande est acceptée, il n'y a pas de réponse.

Exemples:

qualify entree
x element
ADDR 012000 PILOTE
la mémoire ELEMENT est de type caractère et contient la chaîne PILOTE.

x element x12
ADDR 012000 D7C9
on demande le contenu de ELEMENT sous forme de 2 octets hexadécimaux.

```
store element c 'support'
x element
ADDR 012000 SUPPORT
```

ELEMENT contient maintenant la chaîne SUPPORT

```
stop reference modif
```

le programme sera interrompu sur chaque référence à la mémoire MODIF

Ces exemples portaient sur le programme:

```
ENTREE    CSECT
          ...
          ...
          ...
MODIF     L          R15,VALEUR
          ...
          ...
          ...
ELEMENT   DC         CL8'PILOTE'
          ...
          ...
```

READ

Objet:

Cet ordre permet d'entrer des requêtes après l'exécution d'une macro-requête, ou de fermer une macro-requête.

Format:

READ

Utilisation:

Cette requête a deux fonctions suivant qu'elle est envoyée sous le mode AT de l'état-conversation ou exécutée dans une macro-requête.
Dans le premier cas, elle termine la macro-requête et permet ainsi de revenir à l'état-conversation. Dans le second cas, c'est à dire lorsqu'elle sera exécutée, elle replacera PILOTE dans l'état "attente de requêtes" afin qu'il puisse lire de nouvelles lignes.
Si cette requête est tapée dans l'état "attente de requête" elle n'a aucun effet.

Réponse:

Si l'on était dans le mode AT, le signe "souligné" remplace la petite flèche qui invitait à entrer des requêtes.

STOP

Objet:

L'ordre STOP permet de stopper la simulation si certaines conditions sont réalisées, et d'entrer dans le mode conversation.

Format:

STOP	INSTRUCTION	cophex	
	REFERENCE	hexloc1	< hexloc2>

INSTRUCTION permet de s'arrêter avant d'exécuter l'instruction dont le code opération hexadécimal est indiqué en paramètre (cophex).

REFERENCE permet de stopper la simulation si une instruction refère au bloc mémoire indiqué en paramètre. Si la 2ème adresse n'est pas indiquée, hexloc2 = hexloc1+3 est assuré.

Utilisation:

Cette requête ne peut agir que si la simulation est en fonction.

L'argument INSTRUCTION est intéressant dans le cas où l'on veut vérifier certains éléments avant d'exécuter une instruction particulière, par exemple le registre de contrôle d'un BCT, le CSW avant un SIO...

L'argument REFERENCE est très utile lorsque l'on désire s'arrêter chaque fois, que l'on fait référence à un bloc-mémoire,

que l'on exécute du code dans ce bloc, ou que l'on lit ou écrit une partie de ce bloc.

Ainsi, si une zone de données est modifiée, sans que l'on sache par quelles instructions, il suffit de mettre un STOP REFERENCE correspondant à ce bloc.

Si un stop instruction est rencontré, le message: "STOP INSTRUCTION XX(MMMM) AT XXXXXX" sera indiqué
XX est le code hexadécimal de l'opération.
MMMM est le mnémonique.
XXXXXX est son adresse.

Sur une STOP REFERENCE on aura:
STOP ON REFERENCE: AAAAAA-BBBBBB AT XXXXXX
AAAAAA-BBBBBB sont les adresses comprises dans un bloc défini, qui ont été référencées par l'instruction en XXXXXX.

Réponses:

NO MORE 8 STATEMENTS
signifie que l'on a défini le maximum de stops de l'un ou l'autre type. Il faut donc en supprimer un, pour en définir un nouveau.

INVALIDE CORE-ADDRESS:
Ce message indique qu'une adresse est supérieure à la taille de la mémoire "simulée", ou, dans le cas d'un stop sur référence que la deuxième adresse est inférieure à la première.

Aucune vérification n'est faite sur la double définition de ces commandes.

SUBTRACT

Objet:

Cette commande effectue la soustraction hexadécimale de 2 nombres signés.

Format:

SUBTRACT ophex1 ophex2

ophex1 et ophex2 sont les 2 opérandes hexadécimaux.

Utilisation:

Cette commande soustraie l'un à l'autre 2 nombres hexadécimaux signés. Il est évidemment plus sûr d'avoir recours à cette requête que de faire l'opération à la main.

Exemple:

```
subtract 1a0 b0
00000250
```

TRACE

Objet:

Cette requête permet de sortir et de définir le format de la trace sur l'unité précisée par l'ordre PRINT.

Format:

```
TRACE   opt1 < opt2  < opt3  < opt4 >>>
```

Il peut y avoir de 1 à 4 options. La liste de celles-ci est la suivante:

ON met la trace en fonction.

OFF la trace est mise hors fonction.

TAIL La ligne décrit complètement l'instruction en indiquant l'adresse de l'instruction, le code en mémoire, le registre de base, éventuellement l'index, le déplacement... également les modifications du code condition, du système masque, de la clé...(cf ex.)

FAST Seuls l'adresse, le code et le mnémonique sont imprimés.

ALLINST Toutes les instructions sont tracées.

BRONLY Seules les instructions de branchement sont tracées. Un "T" apparaît en tête de ligne lorsque le branchement a eu lieu.

BRTONLY Seules les instructions de branchement effectifs sont tracées, le "T" figure en tête de ligne.

MODGPRS Seuls les registres généraux modifiés par l'instruction apparaîtront dans la trace

ALLGPRS Chaque fois qu'une instruction est susceptible de modifier au moins un registre, les 16 apparaitront dans la trace. Le (ou les) registre(s) modifié(s) sera (seront) précédé(s) d'une étoile.

NOGPRS Les registres généraux n'apparaissent pas dans la trace.

NOFLTRS Les registres flottants n'apparaissent pas dans la trace.

FLTRS Si l'instruction tracée est suceptible de modifier les registres flottants, ils seront tous imprimés. Celui modifié sera précédé d'une étoile.

Utilisation:

La trace ne peut fonctionner que si la simulation est en fonction.
La trace est un outil puissant pour mettre au point un programme, surtout dans un environnement "online", ce qui évite de tracer inutilement certaines séquences d'instructions.
La trace de l'instruction est construite après la simulation de celle-ci; ceci permet d'indiquer les modifications des registres,du psw...etc...
Le problème causé par la trace est le ralentissement important de la simulation. Aussi nous avons donné au programmeur la plus grande souplesse dans le choix du format de sortie. (Appendice B).
Par défaut, les arguments sont les suivants:

OFF TAIL ALLINST MODGPRS NOFLTRS

Exemple:

trace on modgprs fltrs

USING

Objet:

Cette requête permet d'adresser symboliquement une zone du programme, pointée par un registre de base. En particulier cette zone peut être décrite par une DSECT.

Format:

```
USING    name    r
```

name   est un nom externe ou celui d'un DSECT
r   est le numéro d'un registre

Utilisation:

Comme pour la commande QUALIFY, il est nécessaire pour utiliser la requête USING, d'avoir assemblé le programme avec l'option TEST, et ensuite avoir créé un fichier SYMBOL (voir la requête QUALIFY).

La requête USING permet d'adresser symboliquement une partie du programme avec la technique: base/déplacement. La valeur de la base étant le contenu du registre de base spécifié:r. C'est évidemment le programmeur qui est responsable de la valeur correcte du registre de base.

Il n'existe qu'un registre de base au plus. Donc une seconde requête USING élimine l'effet de la première.

Exemples:

using format 5
le registre 5 étant supposé contenir l'adresse de LIGNE, les variables de celles-ci peuvent être désignées par leurs noms.

x numero

ADDR 015007 9

NUMERO contient le caractère 9

x numero2

ADDR 015009 54

NUMERO2 contient les caractères 54

Ces exemples portaient sur la séquence suivante:

```
                ...
                ...
                LA    R5,LIGNE
                USING FORMAT,R5
                ...
        LIGNE   DS    11C
                ...
        FORMAT  DSECT
        TETE    DS    CL5
        NUMERO  DS    C
                DS    C
        NUMERO2 DS    2C
                ...
```

***GPRS**

Objet

Cet ordre permet d'atteindre le contenu d'une mémoire avec un pointeur.

Format:

*GPR	r	lg

r est le numéro d'un registre.

lg est le nombre d'octets demandés.

Utilisation:

Cette requête permet de reférer lg octets de mémoire dont l'adresse est contenue dans le registre r.
Il n'est pas possible de mettre plusieurs étoiles.

Réponse:

Si la requête est correcte, le contenu de la mémoire adressée est imprimé sur le terminal avec la longueur demandée. Cette longueur est de 4 octets par défaut.

Exemple:

*gpr 13 8
00000767E41AABCE

***X**

Objet:

Adressage indirect de la mémoire

Format:

*X	loc	lg

hexloc est l'adresse du pointeur
lg est la longueur.

Réponse:

PILOTE répond en imprimant sur le terminal le contenu de la mémoire dont l'adresse est dans le mot indiqué.

Utilisation:

Cette requête permet d'avoir le contenu de la mémoire pointée par le mot qui est à l'adresse loc.
On ne peut mettre plusieurs étoiles.

III.4 LE MACRO-LANGAGE DE PILOTE.

Ce macro-langage n'a pas pour fonction de permettre l'écriture d'importants programmes, mais de fournir à l'utilisateur des mécanismes pour étendre le langage de requêtes primitives et de le particulariser afin de l'appliquer à certaines situations précises.

La réalisation de ce macro-langage est en cours d'achèvement, aussi nous d'indiquerons dans ce chapitre que les objectifs qui ont dirigé les spécifications et la description des éléments du langage. Une présentation plus complète apparaîtra dans le rapport de D.E.A de MM.BERTHAUD et SAVARY [Réf 3].

III.4.1. LES OBJECTIFS.

PILOTE fournit à l'utilisateur un ensemble d'outils de contrôle et de mise au point assez complet. Ces outils sont manipulables au moyen d'un langage de requêtes.

D'une façon générale les langages de requêtes sont agréables parcequ'ils sont simples. D'une part leur syntaxe permet de formuler sans trop d'erreurs ce que l'on désire, d'autre part, ils ont l'avantage de fournir des réponses rapides.

Malheureusement ces langages sont d'une utilisation rigide: une requête formant un tout, elle ne peut être conditionnée par une requête précédente. Cette impossibilité d'enchaînement automatique estun handicap pour l'utilisateur.

Par exemple, utilisons PILOTE pour tracer l'exécution d'un programme à partir de l'instruction étiquetée INSTR, si la variable VAR contient la valeur 5. Pour cela il faut:

a- mettre un point d'arrêt en INSTR(BREAK)

b- lorsque le programme est interrompu par ce point d'arrêt, demander la valeur de VAR (X).

c- Vérifier que cette valeur est égale à 5, si oui activer la trace (TRACE).

d- Relancer l'exécution (GO).

Chacune de ces étapes doit être répétée autant de fois que l'on exécute l'instruction en INSTR. Il serait beaucoup plus agréable d'exécuter la procédure suivante:

```
AT  INSTR  01
IF  VAR = 5  TRACE  ON
GO
```

De plus, nous avons étendu le problème du macro-langage en nous demandant s'il n'était pas possible de développer des mécanismes indépendants du langage de requêtes afin de lui donner la souplesse qui lui manque lorsqu'il opère seul. En atteignant cet objectif, on acquiert une technique non seulement applicable à PILOTE, mais aussi à un autre environnement, par exemple celui de l'édition de fichiers. Les objets manipulés, que ce soient les octets de la mémoire d'un programme, ou les lignes d'un fichier, n'ont aucune importance, puisqu'ils sont traités par les requêtes primitives. Aussi c'est en collaboration avec M.ADIBA [Réf 1] qui poursuit des travaux sur les éditeurs par contexte, que nous avons défini les spécifications du macro-langage.

III.4.2. LES SPECIFICATIONS.

Pour bénéficier d'une vitesse d'exécution rapide, et surtout pour ne pas encombrer la mémoire d'un interpréteur du macro-langage pendant l'exécution des procédures, nous avons choisi la solution de les compiler au moment où elles sont écrites sur le terminal.

La compilation se fait donc ligne à ligne grâce à une technique développée par messieurs GRIFFITHS et PELTIER [Réf 10] :

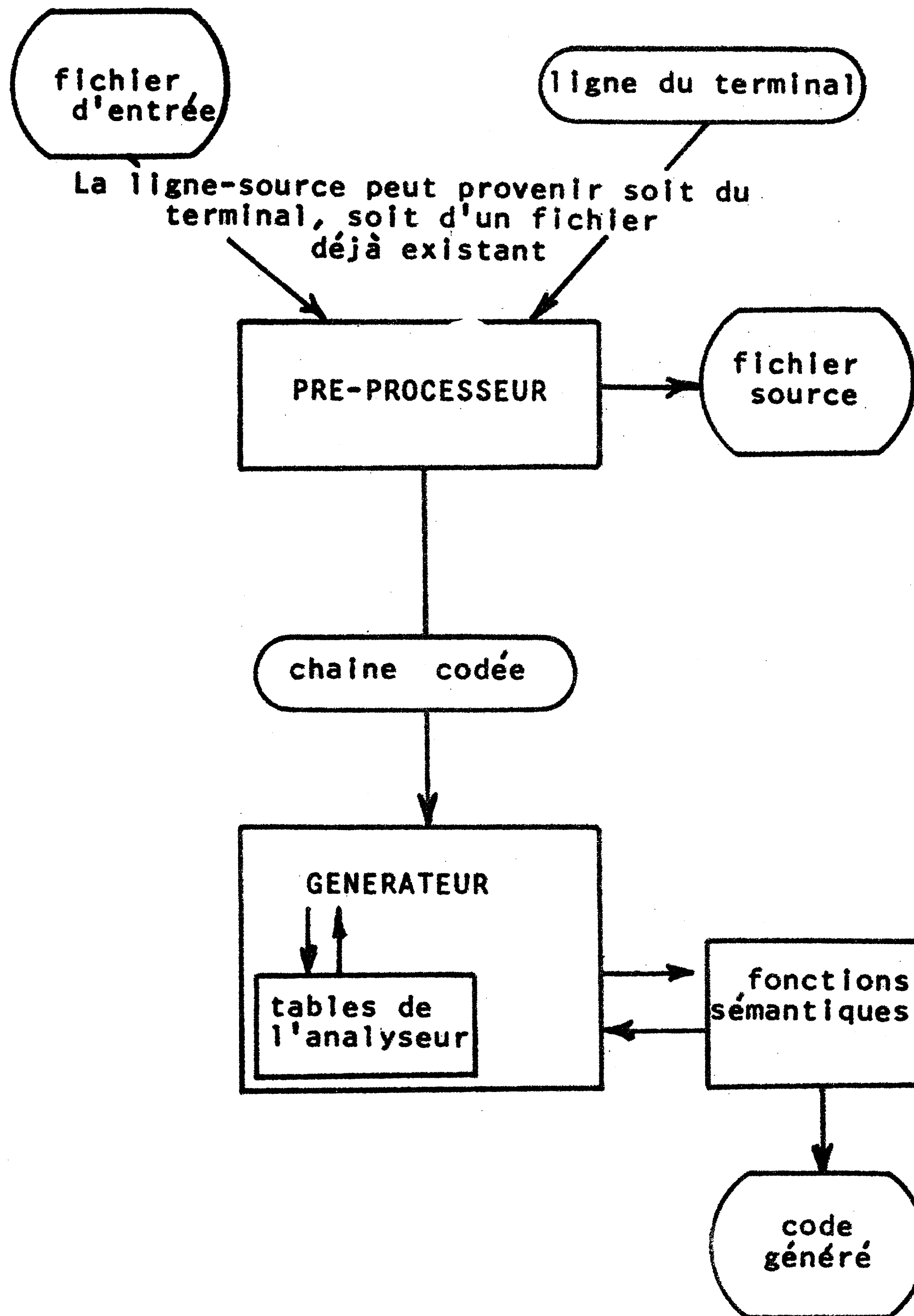


figure III.4 Macro-langage: génération du code-objet.

en fournissant à un transformateur, la grammaire de type LR1 qui décrit le macro-langage, on obtient un analyseur syntaxique. Ce programme fabriqué automatiquement vérifie la syntaxe de chaque instruction: à partir d'une chaîne codée, fournie par le pré-processeur, il décide si l'instruction est correctement écrite, sinon il indique l'erreur au programmeur et ce dernier est invité à corriger sa faute. Enfin un générateur fabrique le code exécutable à partir de la chaîne codée et de fonctions sémantiques.

Grâce à cette technique, on dispose très rapidement d'un vérificateur de grammaire et il suffit d'écrire les fonctions sémantiques et le générateur pour obtenir le code objet. L'organigramme général de la méthode est présenté en figure III.4.

Les programmes écrits dans ce macro-langage sont constitués d'une part d'instructions propres au macro-langage (ces instructions sont compilées), d'autre part de requêtes primitives du langage de requêtes.(Ces requêtes sont transmises à PILOTE au moment de l'exécution de la procédure). Pour faire la distinction, chaque requête devra être précédée d'un point, indiquant ainsi que la chaîne de caractères qui suit doit être transmise à l'analyseur de requêtes primitives.

III.4.2.1.Les éléments du macro-langage.

III.4.2.1.1.Les symboles de base

les chiffres décimaux	0 1 2 3 4 5 6 7 8 9
les chiffres hexadécimaux	0 1 2 3 4 5 6 7 8 9 A B C D E F
les caractères alphabétiques	A B C ... X Y Z
les opérateurs arithmétiques	+ - * /
les opérateurs de relation	$LT $EQ $GT $NLT $NEQ $NGT
les opérateurs séquentiels	$IF $GOTO $CALL
l'opérateur d'affectation	=
le déclarateur de requête primitive	.
les crochets	$( ) $DO $ENDO $PROC $END

III.4.2.1.2 Les nombres et les chaînes

Vu l'application de ce macro-langage à la mise au point des programmes s'exécutant sur un ordinateur IBM/360. Il y a deux genres de nombres:

a- les nombres décimaux entiers, précédés éventuellement d'un signe + ou -. Ces nombres sont composés de chiffres décimaux:
Exemples:
+132 -0140

b- les nombres hexadécimaux, c'est-à-dire selon la base de 16. Ces nombres sont représentés sous la forme de mots de quatre octets. Ils sont obligatoirement entiers, mais peuvent avoir un signe, représenté par le bit le plus à gauche du mot. [Réf 13].
Exemples:
X'19A' est un nombre hexadécimal positif
X'8000001B' est un nombre hexadécimal négatif.

c- les chaînes sont composées d'une suite de caractères alphanumériques encadrés par des quotes.
Exemples:
'ABC1' '232' '2AB'

III.4.2.1.3 Les identificateurs de variables.

Les identificateurs de variables sont représentés par une suite alphanumérique précédée du caractère "ε". Ces variables ne sont pas déclarées, mais leur genre (local ou global) et leur type (entièr ou caractère) sont déterminés par l'identificateur:

Les variables globales (c'est-à-dire accessibles à l'extérieur de la procédure) ont une chaîne de caractère commençant par la lettre G. Les autres sont locales.

Le type des variables est également défini par une lettre: si la variable est numérique son identificateur commence par la lettre "I". Les autres sont de type caractères.

Exemples:

εABC est une variable locale caractère
εGABC est une variable globale caractère
εI2 est une variable locale numérique
εGI2 est une variable globale numérique.

III.4.2.2. L'instruction d'affectation

L'affectation est définie par le signe "=". On peut affecter à une variable une autre variable de même type, ou une constante.

Exemples:

εA = 'EXEMPLE'	la variable εA contient 'EXEMPLE'
εI = εI + 1	la variable εI est incrémentée de 1
εB2 = .GPR4	la variable εB2 contient la valeur du registre 4, valeur qui est fournie par PILOTE.

III.4.2.3. Etiquette et instruction de branchement.

Dans une procédure, les instructions sont séquentiellement exécutées, sauf sur l'instruction $GOTO qui permet de se brancher à l'instruction étiquetée.

a- <u>Les étiquettes</u> sont composées d'une chaîne de caractères alphanumériques précédée d'un caractère "-".

Exemples:

-A -12A -B3

-&A n'est pas une étiquette.

Une étiquette identifie l'instruction en tête de laquelle elle est placée.

Exemple:

-A εI = εI + 1

Les étiquettes permettent de se renvoyer à l'instruction avec l'instruction $GOTO.

b- <u>L'instruction $GOTO</u> est formée de l'élément du macro-langage $GOTO suivi d'une étiquette.

Exemple:

```
...
$GOTO -A
  .
  .
...
...
-A  εI =  εI + 1
```

III.4.2.4. Instruction conditionnelle.

Pour tester une variable numérique ou caractère, on utilise l'instruction $IF. La structure de celle-ci est la suivante:

$IF test instruction inconditionnelle

Si le test est vérifié, l'instruction inconditionnelle est

exécutée, sinon on passe à l'instruction suivante. On peut tester deux variables ou une variable et une constante. Le test se fait avec les opérateurs de relation (III.4.2.1.1.).

Exemple:

On désire imprimer sous le nom DUMPTABL, la mémoire entre les valeurs TABLEAU et TABLEAU + 20 si la variable εA est égale à la valeur du registre 5, sinon on imprime la valeur du registre.

```
εB = .GPRS  5
 $IF εA  $EQ  εB  $GOTO  -A1
 .DUMP DUMPTABL TABLEAU TABLEAU+20
 $GOTO  -B1
 -A1 .GPR 5
 -B1 ...
```

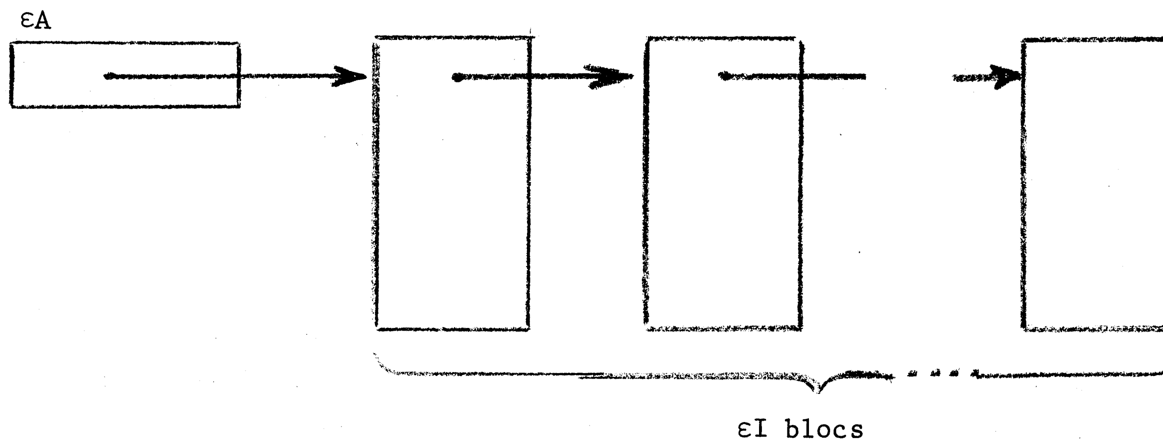
III.4.2.5 Instruction composée $DO

Cette instruction permet de grouper plusieurs instructions à exécuter une ou plusieurs fois. Ces dernières sont placées entre les crochets $DO et $ENDO. Si l'on désire exécuter plusieurs fois l'instruction composée, on indique ce nombre après le crochet $DO.

Exemple:

On désire imprimer le contenu de εI blocs chaînés en liste, Cette dernière étant pointée par ε A.

```
...
$DO   εI
.*X  εA  20  impression de 20 octets du bloc
εA = .*X  εA modification du pointeur
 $ENDO
 ...
```



III.4.2.6 <u>Appel de procédures</u>.

Dans une procédure, on peut appeler une autre procédure. L'appel se fait par l'instruction $CALL dont la structure est:

$CALL nom de la procédure paramètres

Dans la procédure appelée, les identificateurs ε1, ε2,...etc sont remplacés par les paramètres passés à l'appel.

Exemple:

Soit la procédure construite ainsi:

```
$PROC  XLIST
$DO  ε1
.*X  ε2  20
ε2 = . * X  ε2
$ENDO
$END
```

On peut l'appeler pour ε1 = 5 et ε2 = INIT :

```
$CALL  XLIST  5  INIT
```

```
NPUT SYNTAX=

5 PROG          'SYMBOL' 'PARENTG' LISTPARAM 'PARENTD' 'PV' LISTINST 'END' 'PV'
                'SYMBOL' 'PV' LISTINST 'END' 'PV'

6 LISTINST      INST 'PV' LISTINST
                INST 'PV'

7 INST          'ETIQ' INSTCOND
                INSTCOND
                'ETIQ' INSTBRANCH
                INSTBRANCH
                'ETIQ' INSTAFFECT
                INSTAFFECT
                'ETIQ' APPELPROC
                APPELPROC
                'ETIQ' COMPRIM
                COMPRIM
                'ETIQ' INSTDO
                INSTDO
                'ETIQ' 'EXIT'
                'EXIT'
                'ETIQ' 'NOP'
                'NOP'

8 INSTBRANCH    'GOTO' 'ETIQ'

9 INSTAFFECT    'VARNUM' 'EGAL' EXPRARITH
                VARIABLE 'EGAL' EXPRCAR
                'VARNUM' 'EGAL' COMPRIM
                VARIABLE 'EGAL' COMPRIM
                'VARNG' 'EGAL' COMPRIM
                VARG 'EGAL' COMPRIM
                VARG 'EGAL' EXPRCAR
                'VARNG' 'EGAL' EXPRARITH

0 INSTDO        'DO' 'VARNG' 'PV' LISTINST 'EDO'
                'DO' 'PV' LISTINST 'EDO'
                'DO' 'VARNUM' 'PV' LISTINST 'EDO'
                'DO' CSTNUM 'PV' LISTINST 'EDO'

1 EXPRCAR       VARG
                VARIABLE
                'CSTCAR'
                VARG 'CONC' EXPRCAR
                VARIABLE 'CONC' EXPRCAR
                'CSTCAR' 'CONC' EXPRCAR

2 APPELPROC     'CALL' 'SYMBOL' 'PARENTG' LISTPARAM 'PARENTD'
                'CALL' 'SYMBOL'
```

```
73 EXPRBOLL       VARIABLE 'OPRELATION' VARG
                  VARG 'OPRELATION' VARIABLE
                  'VARNG' 'OPRELATION' 'VARNUM'
                  'VARNUM' 'OPRELATION' 'VARNG'
                  'CSTCAR' 'OPRELATION' VARG
                  VARG 'OPRELATION' VARG
                  CSTNUM 'OPRELATION' 'VARNG'
                  'VARNG' 'OPRELATION' CSTNUM
                  VARG 'OPRELATION' 'CSTCAR'
                  'VARNG' 'OPRELATION' 'VARNG'
                  'CSTCAR' 'OPRELATION' VARIABLE
                  CSTNUM 'OPRELATION' CSTNUM
                  VARIABLE 'OPRELATION' VARIABLE
                  CSTNUM 'OPRELATION' 'VARNUM'
                  'VARNUM' 'OPRELATION' CSTNUM
                  VARIABLE 'OPRELATION' 'CSTCAR'
                  'VARNUM' 'OPRELATION' 'VARNUM'
                  'CSTCAR' 'OPRELATION' 'CSTCAR'

74 CSTNUM         'CSTH'
                  'CSTD'

75 INSTCOND       'IF' EXPRBOOL 'EXIT'
                  'IF' EXPRBOOL INSTBRANCH
                  'IF' EXPRBOOL INSTAFFECT
                  'IF' EXPRBOOL APPELPROC
                  'IF' EXPRBOOL COMPRIM
                  'IF' EXPRBOOL INSTDO

76 EXPRARITH      TERME
                  'SIGNE' TERME

77 TERME          FACTEUR
                  FACTEUR 'OP' TERME
                  FACTEUR 'SIGNE' TERME

78 FACTEUR        CSTNUM
                  'VARNUM'
                  'VARNG'
                  'PARENTG' EXPRARITH 'PARENTD'

79 LISTPARAM      'VARNUM'
                  VARIABLE
                  'CSTCAR'
                  CSTNUM
                  'VARNUM' LISTPARAM
                  VARIABLE LISTPARAM
                  CSTNUM LISTPARAM
                  'CSTCAR' LISTPARAM
```

```
80 VARIABLE      'VARIABLEN'
                 'VARIABLEN' SB

81 VARG          'VARGN'
                 'VARGN' SB

82 SB            'PARENTG' 'VARNG' VARNG' 'PARENTD'
                 'PARENTG' 'VARNG' 'CSTD' 'PARENTD'
                 'PARENTG' 'CSTD' 'VARNG' 'PARENTD'
                 'PARENTG' 'VARNG' 'VARNUM' 'PARENTD'
                 'PARENTG' 'VARNUM' 'VARNG' 'PARENTD'
                 'PARENTG' 'VARNUM' 'VARNUM' 'PARENTD'
                 'PARENTG' 'CSTD' 'VARNUM' 'PARENTD'
                 'PARENTG' 'CSTD' 'CSTD' 'PARENTD'
                 'PARENTG' 'VARNUM' 'CSTD' 'PARENTD'

83 COMPRIM       'NOMCOM'
                 'NOMCOM' LISTELEM
                 'NOMCOM' LISTELEM 'SEP' SB
                 'NOMCOM' 'SEP' SB

84 LISTELEM      'SMB'
                 'VARNUM'
                 'VARIABLEN'
                 'VARGN'
                 'VARNG'
                 'SMB' LISTELEM
                 'VARIABLEN' LISTELEM
                 'VARNUM' LISTELEM
                 'VARGN' LISTELEM
                 'VARNG' LISTELEM
```

CHAPITRE IV

DESCRIPTION INTERNE ET LOGIQUE DE PILOTE

PILOTE est écrit en langage d'assemblage IBM/360 [Réf 13]. Malgrè les reproches que l'on peut faire aux langages machine, il est le plus adapté à notre réalisation. La programmation du support PILOTE comprend en effet:

- des composants traitant les interruptions
- des composants de gestion d'entrées-sorties au niveau du canal.
- des composants simulant les instructions de la machine.

Pour ce faire, PILOTE est contraint de manipuler l'octet et le bit; de plus certains modules conversationnels exigent une fonctionnement rapide. Enfin, les problèmes de taille, et de mise au point du support confirment le choix du langage d'assemblage. D'une part il évite le chargement de bibliothèques, d'autre part, le support étant destiné principalement aux programmes en langage-machine, il nous a permis de poursuivre son propre développement, dès les premiers résultats.

Le support de mise au point peut être implanté en mémoire du calculateur à une place quelconque (sachant que toute la mémoire, de cet endroit à sa fin physique est inutilisable pour le programme testé -zone non simulée-) et s'exécuter indépendamment de tout système de programmes. Il dispose en effet de ses propres sous-programmes de traitement d'entrées-sorties et d'interruptions.

PILOTE se compose essentiellement de deux segments: DEMON et MONITEUR. La tâche principale du premier est d'assurer la partition conversationnelle entre PILOTE et le programmeur. Le second permet d'effectuer la

simulation des programmes, d'intervenir sur ceux-ci selon les directives de l'utilisateur, de collecter dynamiquement des informations et de les éditer, enfin de sortir ces renseignements sur des organes périphériques (imprimante, bande, terminal).

Deux segments annexes complètent cet ensemble: l'un GML, gère la mémoire libre de PILOTE, l'autre DATA analyse les instructions pour déterminer quelles sont les zones qu'elles affectent.

Enfin, notons que l'utilisation de PILOTE avec CMS, suivant l'emploi décrit au chapitre précédent (III.3), nécessite certaines modifications à trois modules du système:

NUCON	pour conserver l'adresse de résidence de DEBUG
DEBUG	pour accepter de nouvelles requêtes
LOGIN	pour implanter PILOTE en mémoire.

La figure IV.1 donne une vue générale des liaisons entre les différents segments, le programme testé et l'utilisateur qui dispose du terminal.

IV.1. CHARGEMENT DE PILOTE DANS LA MEMOIRE.

Dans le chapitre précédent, nous avons vu que le moniteur, grâce à l'interpréteur, fournit les outils de mise au point les plus avancés. Cet interpréteur simule dans toutes ses fonctions l'unité centrale d'un ordinateur IBM360 standard, c'est-à-dire d'un ordinateur dont la mémoire centrale n'est pas soumise à la translation dynamique d'adresses (cas du 360/67 par exemple). Cette simulation permet d'analyser et de contrôler tous les travaux qui utilisent l'unité centrale.

Le support PILOTE peut se représenter comme une machine de mise au point et une machine testée, produite par simulation. Si ces deux ordinateurs peuvent être conçus séparément (voir le chapitre V), il n'en est

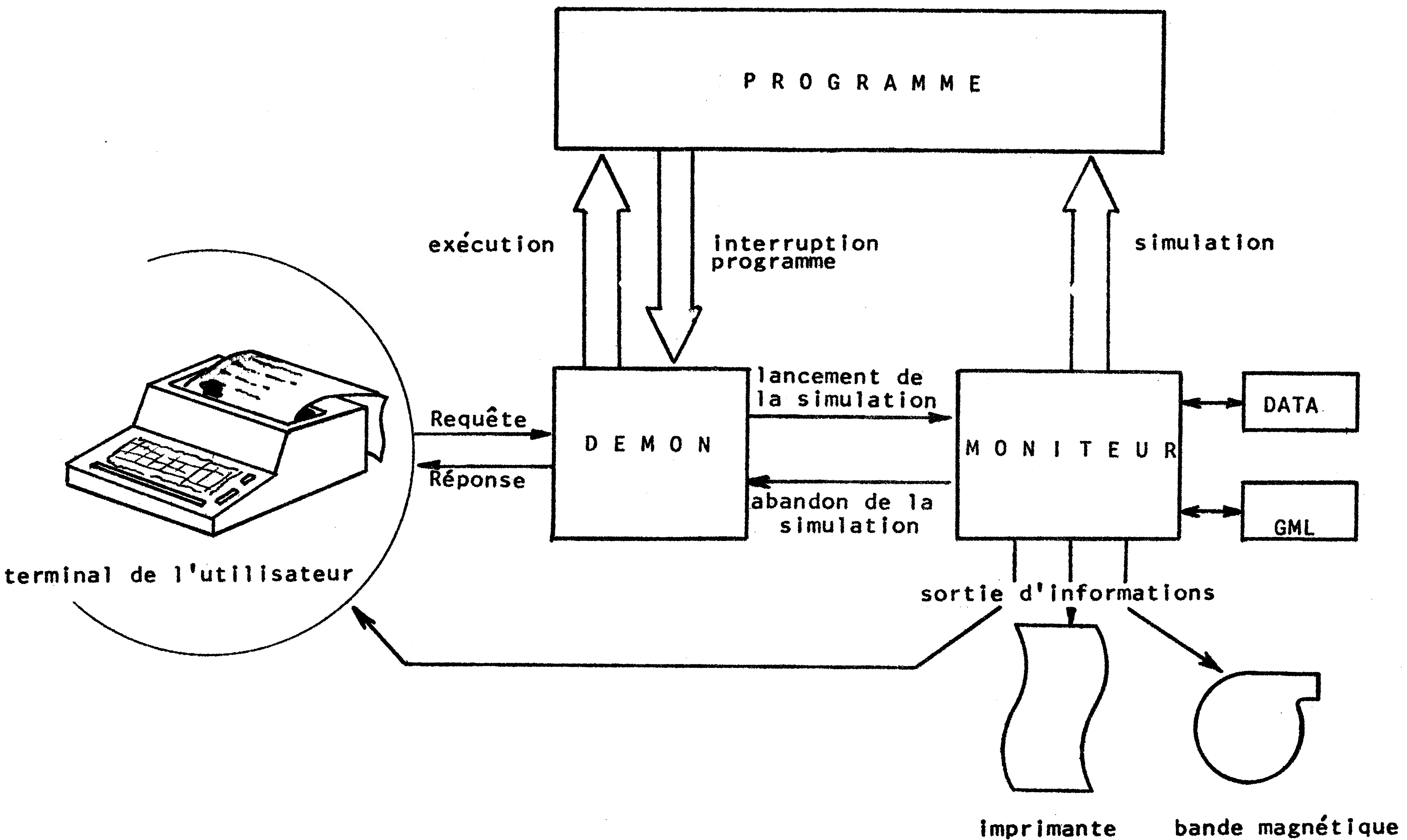


figure IV.1 Controle de l'exécution entre les 4 segments de PILOTE.

pas ainsi, et aucun moyen "hardware" ne permettant de les distinguer, il est nécessaire de définir les ressources de chacun d'eux pour éviter tout conflit.

Par convention, la machine simulatrice (PILOTE), dispose de toutes les ressources de l'ordinateur réel et peut ainsi exécuter toutes les instructions de l'unité centrale, adresser toute la mémoire et toutes les unités périphériques. Quant aux ressources de la machine simulée, il nous appartient de les définir, c'est-à-dire d'en dresser la liste. Durant toute la simulation, le moniteur vérifie si les ressources demandées par la machine simulée figurent dans cette liste. Si c'est le cas la ressource est rendue disponible, sinon une condition d'erreur interrompt le programme qui l'invoquait.

Pour autoriser la simulation de tous les programmes, nous nous sommes efforcés de minimiser les contraintes. Aussi les ressources de la machine simulée sont celles de l'ordinateur réel, excepté la partie de la mémoire centrale où réside PILOTE. Cette zone d'un seul tenant, située aux plus grandes adresses du calculateur, est la mémoire non simulée , par opposition à la mémoire simulée représentée par le reste.(Figure IV.2).

Grâce à ces conventions, les programmes s'exécutant sous le contrôle de PILOTE peuvent:

. disposer de toutes les instructions de l'ordinateur,
. fonctionner dans tous les modes, en particulier maître et esclave,
. effectuer des opérations sur toutes les unités périphériques,
. faire des références à la plus grande partie de la mémoire rapide.

Les deux "machines" fonctionnant sur le même ordinateur, seul le moniteur peut assurer la protection de la mémoire non-simulée; or celui-ci n'étant actif que pendant la simulation il faudrait convenir de la règle suivante: le programme testé ne peut jamais utiliser l'unité centrale

réelle, c'est-à-dire s'exécuter normalement.

Mais ceci s'oppose à l'un de nos objectifs qui est de fournir un outil de mise au point suffisamment souple pour n'être utilisé que sur certaines parties des programmes. Les autres doivent pouvoir utiliser l'unité centrale.

Il est donc nécessaire d'assouplir la règle énoncée, en minimisant les risques qui pourraient en résulter, mais sans les oublier. Pour cela, nous avertissons les programmes qui vont s'exécuter qu'une partie de la mémoire n'est plus adressable. Par cette opération, nous espérons qu'ils tiendront compte de l'avertissement et nous les autorisons à s'exécuter normalement.

L'idée générale qui a guidé la réalisation de cette opération est la suivante: dès que le superviseur des programmes testés (par exemple le système CMS) est appelé dans la mémoire de l'ordinateur, il détermine la taille de celle-ci (soit en la calculant, soit en consultant la valeur d'un paramètre) afin de l'organiser (mémoire résidente, mémoire de travail, etc..) Les informations résultant de cette opération sont alors conservées dans des zones de travail et sont utilisées par les différents programmes de gestion de mémoire. Si l'on parvient donc à faire savoir au système pendant son initialisation, qu'il ne peut disposer de toute la mémoire du calculateur, on peut raisonnablement penser que par la suite, ni lui, ni aucun autre programme n'adressera la mémoire interdite.

Une solution générale consiste à charger dans une première phase le support, en mémoire(opération "ipl" réel). Dans la seconde phase, le support contrôle toutes les autres opérations, en particulier:

- questionner au sujet de l'adresse de l'unité périphérique sur laquelle réside le système que l'on veut piloter (adresse de l'"ipl").
- charger à partir de cette adresse (simulation de "l'ipl" réel) le système dans la mémoire.

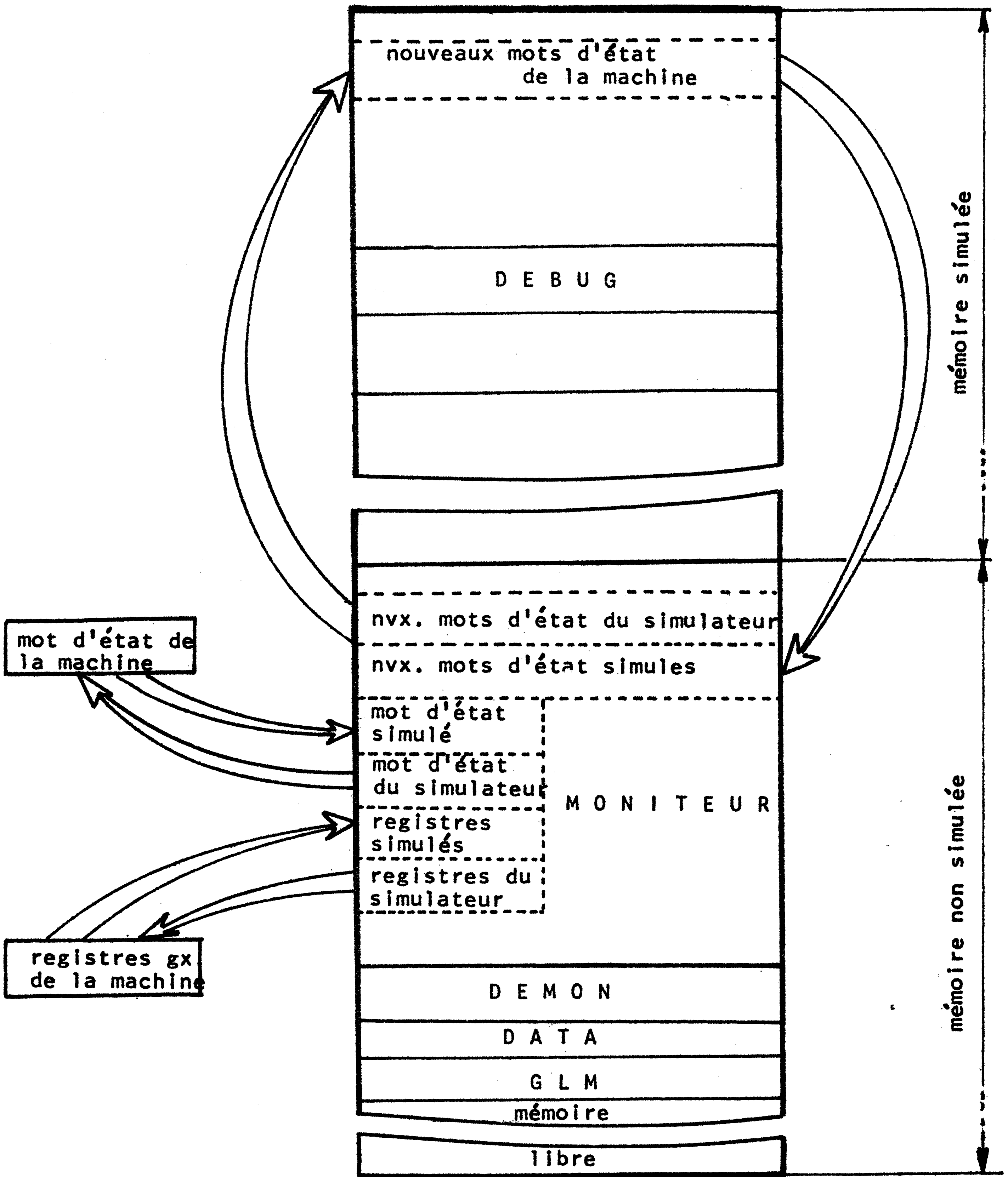


figure IV.2 <u>Organisation de la Mémoire Centrale et activation du simulateur</u>.

Par là-même, l'ensemble des initialisations, compte tenu du support, est réalisé. On a ainsi l'assurance que la taille de la mémoire simulée est bien calculée et que les risques de destructions de la mémoire non simulée sont réduits.

La solution employée pour le cas de CMS est différente. Pour un utilisateur de CMS il est en général sans intérêt de contrôler le chargement initial du système, opération qui augmente évidemment le temps d'exécution. Aussi ce n'est pas PILOTE qui charge CMS, mais l'inverse. Grâce à la commande "LOGIN", l'utilisateur peut invoquer le support.

Tant que CMS n'a pas construit le catalogue des fichiers de l'utilisateur |Réf 15| , la mémoire centrale n'est guère employée, en particulier la partie qui deviendra la zone non-simulée.

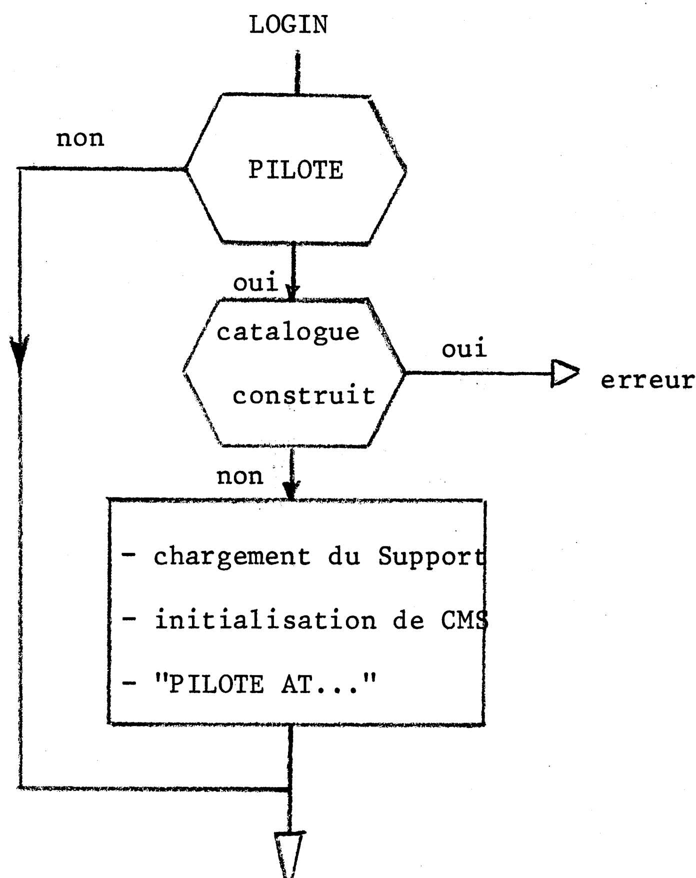
La première commande est implicitement ou explicitement LOGIN: elle construit en mémoire l'ensemble des informations qui décrit les fichiers personnels de l'utilisateur. C'est pour agir avant cette opération que nous avons ajouté un nouveau paramètre permettant de charger le support. (voir III.3.1).

En procédant ainsi, PILOTE a donc pris la place nécessaire à son fonctionnement et a modifié les paramètres de CMS relatifs aux adresses et à la taille de la mémoire centrale. Détaillons l'enchaînement des opérations:

La commande "LOGIN PILOTE" vérifie que le catalogue des fichiers de l'utilisateur n'est pas déjà construit, et donne le contrôle au programme PILOAD qui:

- amène en mémoire les quatre composants de PILOTE: MONITEUR, DEMON, DATA, GML.
- met à jour les paramètres qui décrivent la taille de la mémoire centrale.
- avertit le programme DEBUG de la présence de PILOTE
- calcule la taille de la mémoire de travail de PILOTE
- envoie une réponse sur le terminal

- rend le contrôle à la commande "LOGIN" qui construit le catalogue des fichiers.



De cette façon la mémoire rapide est partagée en deux parties (figure IV.2).

- mémoire simulée utilisable par CMS
- mémoire non simulée interdite à CMS et utilisée par PILOTE.

IV.2. DEMON - PARTITION CONVERSATIONNELLE DE PILOTE.

Ce composant est un des éléments essentiels dans l'organisation de PILOTE. Son rôle est double: d'une part il assure toutes les communications entre l'utilisateur, le programme testé, CMS, le moniteur; d'autre part il analyse, exécute et répond aux requêtes du programmeur.

Pour examiner et modifier l'état du programme ou de l'ordinateur,pour activer ou désactiver les outils de mise au point, l'utilisateur émet une requête à l'aide du terminal. Ce terminal est alors entièrement géré par DEMON. Pour faciliter l'utilisation de PILOTE, le mode-conversation est fonctionnellement identique à l'environnement DEBUG de CMS. Ce programme doit donc savoir:

- qu'il dispose de nouvelles requêtes
- qu'il peut donner le contrôle au moniteur (s'il est en fonction) sur la requête "GO" ou retour de chariot
- qu'il doit ranger les requêtes si une macro-requête est ouverte en écriture -MODE-AT-
- qu'il doit exécuter les requêtes rangées en mémoire si une macro-requête est ouverte en lecture.

IV.2.1. COMMUNICATIONS ENTRE DEMON ET LES MODES EXECUTION ET SIMULATION.

Il y a trois entrées à ce module, correspondant à trois cas:

- une demande explicite depuis l'environnement CMS, par la commande "DEBUG". Cette manière d'entrer dans le mode-conversation impose d'en sortir par la requête RETURN |Réf 15| .

- sur une condition anormale d'exécution au cours du déroulement d'une tâche quelconque.(mode-exécution).

- sur un arrêt de la simulation (mode-simulation).

IV.2.1.1. Entrée à partir du mode-exécution.

Quand on passe en mode-conversation à partir du mode-exécution il faut figer l'état du programme et de la machine. C'est pourquoi, les registres généraux, les mots d'états des canaux périphériques (CSW et CAW), le mot d'état du programme (PSW) et la partie des nouveaux mots d'état, utilisée par le "hardware", sont recopiés afin d'être restaurés lorsque les opérations de mise au point seront terminées et le programme relancé.

Les deux cas importants pour passer en mode conversation sont les interruptions externe et programme.

a- l'interruption externe

Une interruption externe peut être provoquée par l'horloge de l'unité centrale, par une autre unité centrale, par un signal déclenché par un poussoir se trouvant sur le pupitre de l'ordinateur. Seul le dernier évènement permet de passer en mode conversationnel.

Cette technique permettant un appel asynchrone, est souvent utilisée pour interrompre la tâche lorsque l'on soupçonne une exécution anormale. Mais pour que l'interruption soit prise en compte, il faut l'autoriser dans le mot d'état du programme (psw), c'est ce qui sera fait dès qu'on sort du mode conversation par la requête "GO".

b- L'interruption programme.

Cette interruption est provoquée par une instruction incorrecte, par exemple ayant un code opération invalide. Elle peut avoir deux significations:

- il s'agit une véritable erreur dûe au programme, l'utilisateur est alors invité à la corriger.

- il s'agit d'une erreur voulue, correspondant à un point

d'arrêt posé par le programmeur.(requête BREAK |Réf 15|). Dans ce cas, DEMON, corrige immédiatement l'instruction en replaçant le code opération en mémoire, et sur le terminal, indique au programmeur le numéro et l'adresse du point d'arrêt.

IV.2.1.2. Entrée à partir du mode-simulation.

Dans cette circonstance, c'est le moniteur qui appelle DEMON:

- soit parcequ'il a simulé le nombre d'instructions demandé,
- soit parcequ'un arrêt sur conditions (STOP et ON) a été rencontré,
- soit parcequ'une macro-requête est ouverte en lecture.

Grâce à un indicateur fourni par le moniteur, DEMON reconnaît la raison de l'appel et peut agir de trois façons différentes suivant les directives du programmeur.

a) Directive GIVEUP

Cette directive permet d'abandonner la simulation. DEMON restaure les éléments qui figurent l'état du programme et de la machine simulée, puis relance celle-ci avec l'aide de son mot d'état ("program status word" courant).

b) Directive SKIP

Cette directive correspond à une rupture de séquence. L'utilisateur désire abandonner la simulation, puis la reprendre lorsque l'exécution reviendra au point de rupture de séquence.

Pour faire ces opérations DEMON place un code invalide (X'DO') sur l'instruction qui suit celle de rupture de séquence, après avoir sauvé son code opération et son adresse dans une table (SKIPTBL). Lorsque

l'exécution sera interrompue par ce code invalide, DEMON, automatiquement restaurera le code opération et à partir de là, continuera la simulation du programme.

Après avoir posé le code invalide DEMON exécute le sous-programme GIVEUP.

c) Directive STOP.

Cette directive stoppe le programme et pemet à l'utilisateur d'examiner son état. Deux cas sont alors possibles:

Soit le contrôle est donné au terminal, celui-ci attend alors que l'utilisateur envoie des requêtes.

Soit une macro-requête est ouverte en lecture. Au lieu d'attendre les requêtes du terminal, DEMON lit celles-ci dans la mémoire libre de PILOTE, où elles ont été placées au moment de leurs définitions; et ainsi elles sont exécutées les unes après les autres. La dernière d'entre elles, READ ou GO, ferme la macro-requête et indique l'action à entreprendre: autoriser l'usage du terminal pour présenter de nouvelles requêtes, ou relancer l'exécution du programme. Cette exécution sera simulée ou non suivant l'état du moniteur, prêt ou non.

IV.2.1.3. Sorties du mode-conversation.

DEMON est abandonné sur la requête "GO" suivie éventuellement d'une adresse. Dans ce cas, l'adresse estplacée dans le mot d'état du programme.

Si le moniteur n'est pas en fonction, l'état de la machine est mis à jour avec les éléments qui avaient été copiés à l'entrée de DEMON: régistres généraux, mots d'état des canaux, mot d'état du programme, zone des nouveaux mots d'états. Puis le mot d'état du programme est activé.

Par contre si le moniteur est prêt, ces éléments sont passés comme paramètres à l'interpréteur et c'est le moniteur qui est activé;

la simulation commence alors à l'adresse indiquée par le compteur ordinal du programme. Toujours dans ce cas, notons que l'envoi sur le terminal d'une ligne vide a le même effet que la requête "GO" sans adresse.

IV.2.2. ANALYSE ET TRAITEMENT.

Les requêtes émises par le programmeur peuvent s'exécuter immédiatement (mode-conversation) ou en différé (mode-AT), si une macro-requête est ouverte (figure III.1).

IV.2.2.1 Analyse des requêtes.

Lorsque DEMON reçoit la ligne tapée sur le terminal, l'analyse est immédiatement entreprise; (une requête ne peut donc occuper plusieurs lignes). La première opération est la mise au propre de la ligne, compte tenu des caractères d'annulation de caractères - @ - et de ligne - ¢ -, ainsi que des blancs de tête.

Si l'on est en mode immédiat, un bloc-requête est construit. Le premier champ de ce bloc décrit la requête: nombre d'arguments, type des arguments. Les autres champs (de longueur 8 octets) contiennent chaque sous-chaîne de la ligne.(le délimiteur est le blanc).

- chaque sous-chaîne de type caractère est cadrée à gauche et complétée éventuellement par des blancs à droite,

- chaque sous-chaîne de type numérique (nombre décimal ou hexadécimal) est cadrée à droite et éventuellement complétée de zéro à gauche.

Le premier argument de la requête est pris comme mot-clé. Il est alors comparé avec les éléments de la liste des requêtes existantes dans DEBUG. Si la recherche est infructueuse, elle se poursuit avec la consultation de la liste des requêtes propres à PILOTE. Un échec de l'opération, provoque l'impression sur le terminal d'un message d'erreur et DEMON attend

une nouvelle requête, par contre un succès permet d'appeler le sous-programme spécialisé dans le traitement de la requête. Quand il a achevé sa tâche, il rend le contrôle à DEMON pour lire une nouvelle ligne.

Ce processus se poursuit jusqu'à ce qu'une requête qui permet de quitter le mode-conversation soit décodée. L'algorithme de ce programme est représenté par la figure IV.3.

IV.2.2.2 Traitement des requêtes.

La requête AT permet de passer du mode-conversation au mode-AT, où l'exécution des requêtes qui vont être émises est différée jusqu'au moment où le moniteur simulera l'instruction se trouvant à l'adresse indiquée dans la macro-requête.

Sur l'occurrence d'une requête AT, DEMON examine un bloc de contrôle: ATABLE (figure IV.4), et recherche dans celui-ci une entrée disponible. Dans celle-ci il place l'adresse d'exécution de la macro-requête et le compteur qui lui est associé. (Chapitre III.3.4).

La macro-requête est alors ouverte en écriture, jusqu'à l'occurrence d'une requête GO ou READ. Dans cette situation chaque ligne tapée sur le terminal est mise au propre (IV.2.2.1); mais au lieu d'analyser la requête comme précédemment, un bloc de mémoire est demandé dans la mémoire libre de PILOTE et chaîné à la ATABLE, ou au bloc précédent.

Sur le décodage d'une requête GO ou READ, le traitement est identique sauf s'il s'achève par la fermeture de la macro-requête et le retour au mode-conversation.

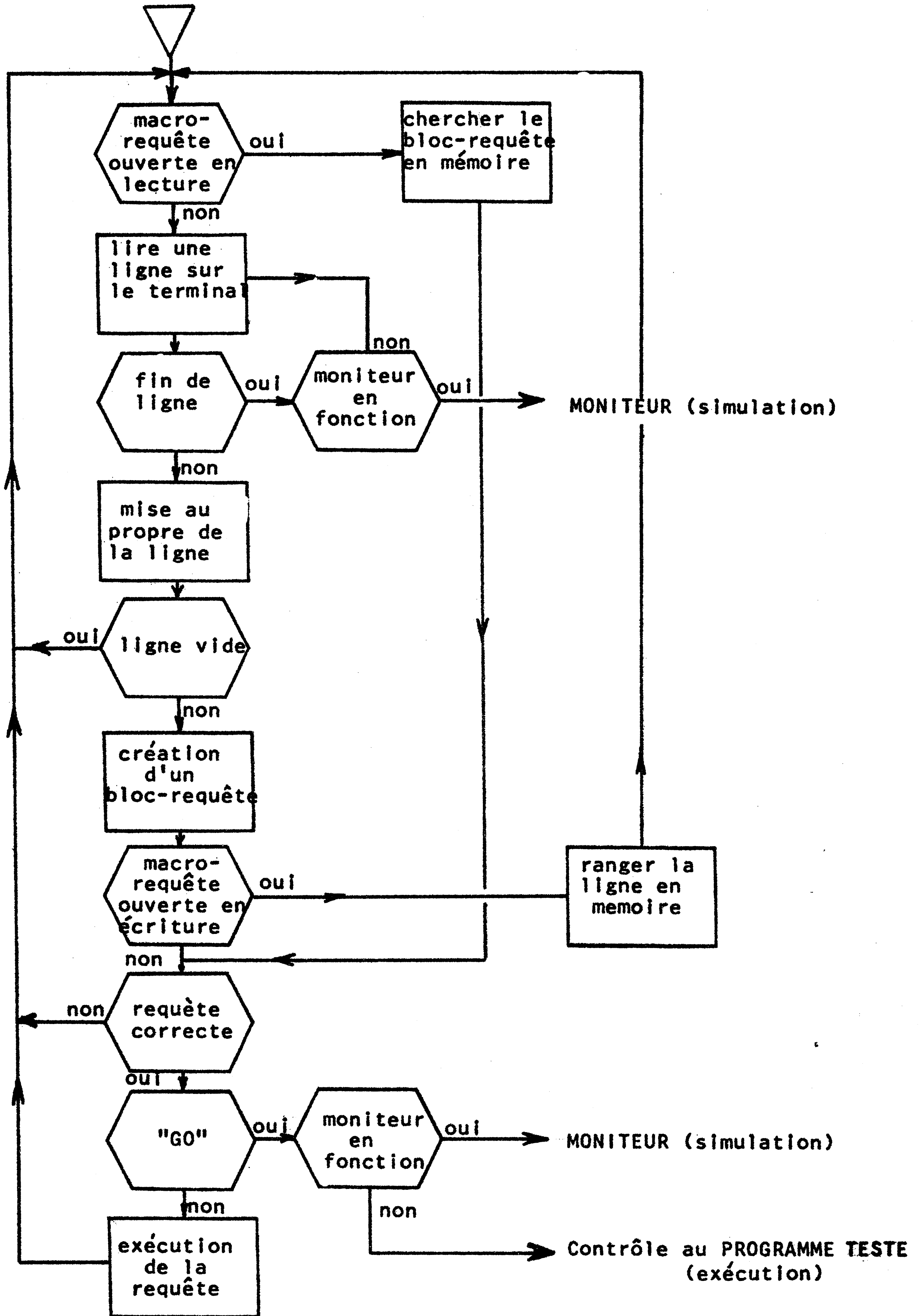


Figure IV.3 BEMON: traitement d'une ligne-requête.

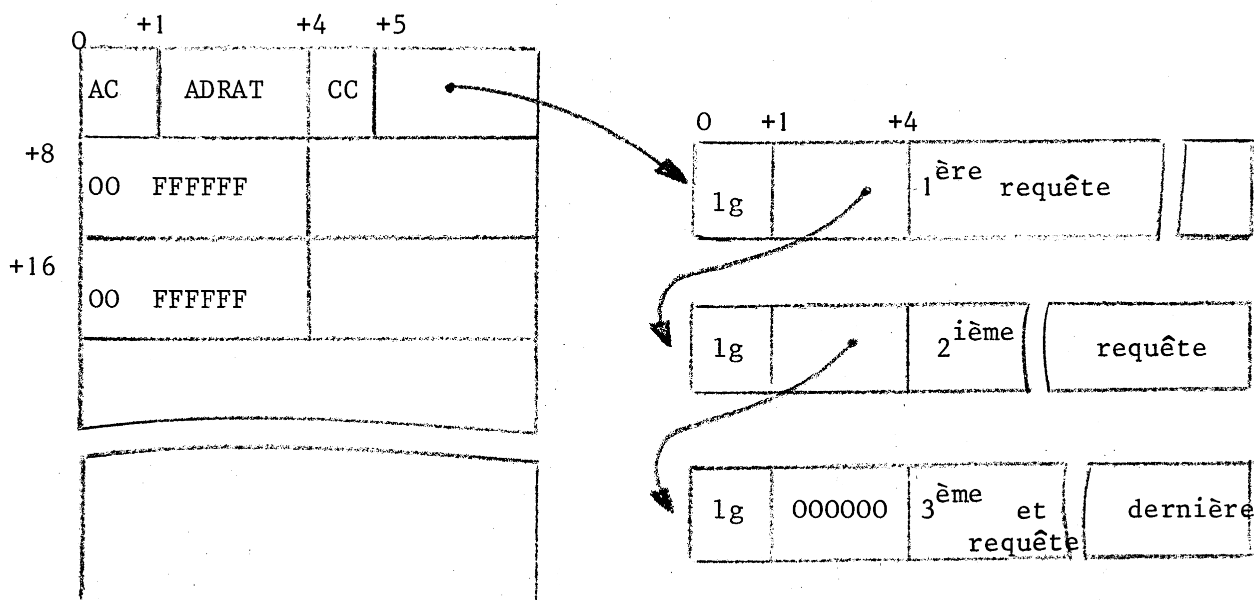


FIG.IV.4 ATABLE et chaînage des requêtes

il y a 3 entrées dans la ATABLE.

AC est le compteur absolu

ADRAT est l'adresse où sera exécutée la macro-requête. Si l'entrée est vide la valeur est -1.

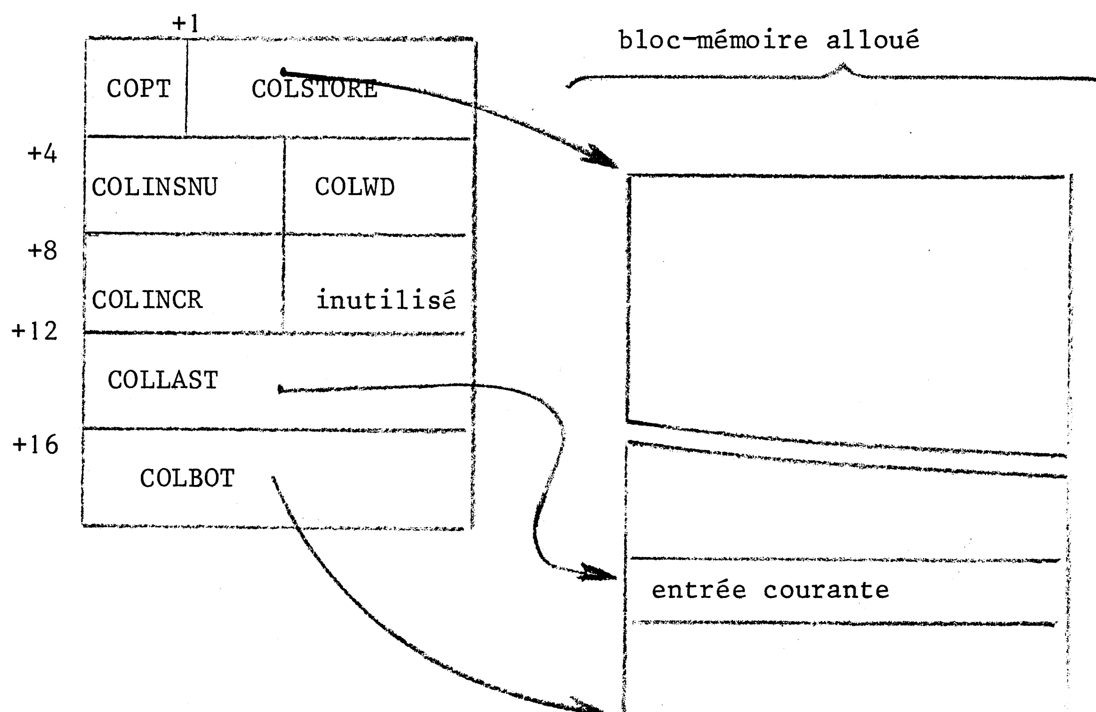
CE est le compteur courant

1g est la longueur de la requête

La requête COLLECTE permet d'obtenir un bloc pour que le moniteur puisse en cours de simulation ranger des informations. DEMON vérifie d'abord qu'aucune collecte n'a déjà été définie, et dans ce cas, envoie un message d'erreur.

Dans le cas où aucune collecte n'existe, DEMON, d'après le nombre d'instructions demandées, et l'option des registres (chapitre III.3.4), calcule la taille de mémoire libre nécessaire et en fait la demande. Si cette mémoire est accordée, il range dans la table COLTABLE (figure IV.5), son adresse. Puis il initialise avec la même valeur le pointeur vers l'instruction courante. C'est à partir de cette instruction que sera imprimée la collecte au moment de la requête COLLECT OUT.

Une instruction nécessite 32 octets, plus 64 octets pour les registres.



COPT	X'80' avec les reg.	COLINCR	taille de chaque entrée
	X'40' active	COLLAST	adresse de l'entrée courante
	X'20' déjà remplie		
COLSTORE	adresse du bloc-mémoire alloué	COLBOT	adresse de la fin du bloc alloué.
COLINSNU	nombre d'instructions max.		
COLWD	taille du bloc alloué		

FIGURE IV.5 COLTABLE

IV.3 MONITEUR - SUPERVISEUR DU PROGRAMME TESTE

Les principaux outils de mise au point de PILOTE sont fournis par le MONITEUR qui supervise l'exécution des programmes testés. Ce composant est formé de différents modules dont les rôles sont:

1- Simuler, par interprétation, les fonctions de l'unité centrale de l'ordinateur - interpréteur.

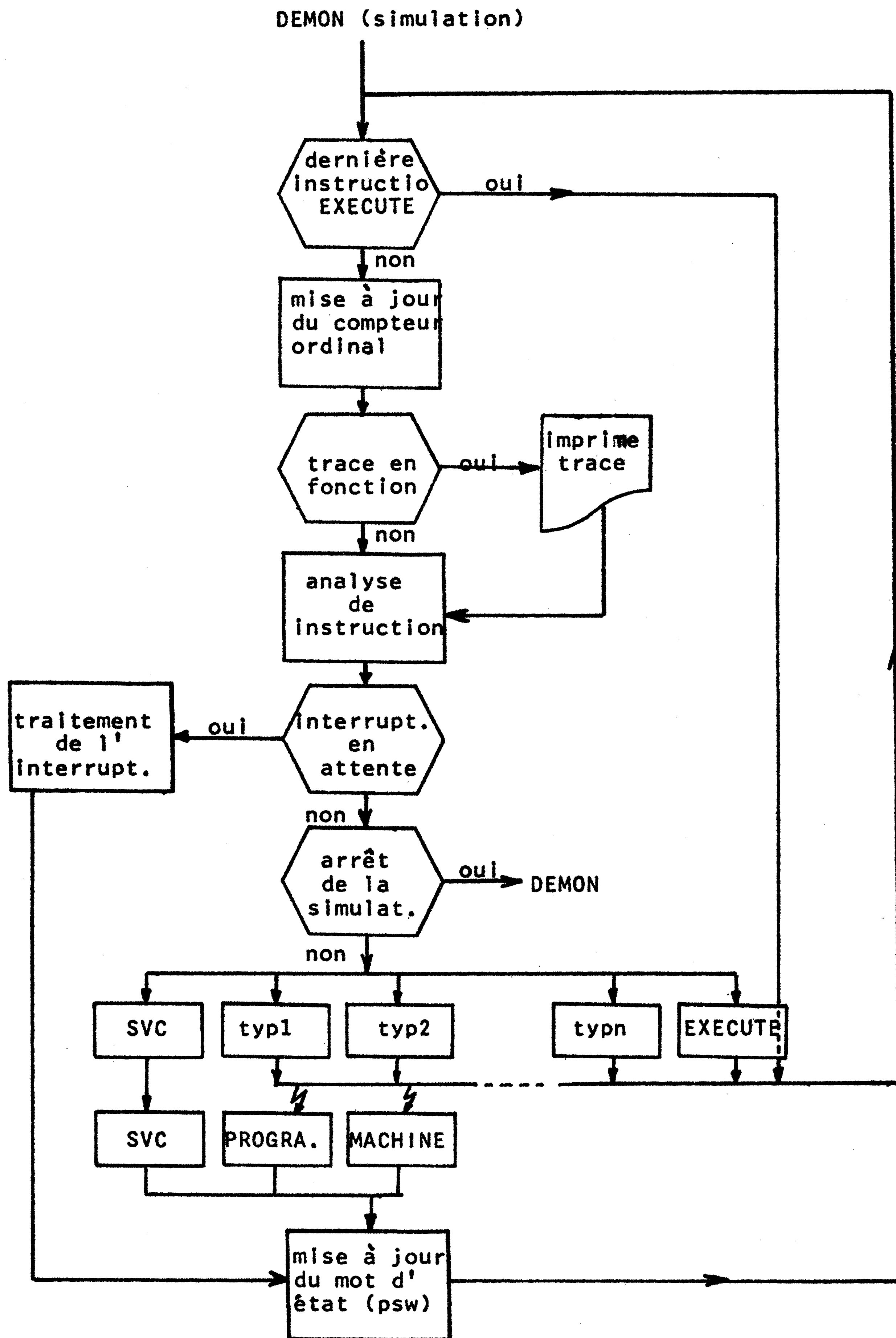


figure IV.6 <u>Boucle centrale du MONITEUR</u>.

2- Recueillir dynamiquement le long de l'exécution des informations à éditer -trace- ou à conserver en mémoire -collecte- pour être exploitées ultérieurement.

3- Tester l'occurence d'évènements indiqués par le programmeur et qui provoquent une action spéciale. -directives-

4- Gérer les opérations sur les organes périphériques de PILOTE (terminal, machine à imprimer, bande magnétique) pour respecter leurs indépendances vis à vis de celles émises par le programme testé.

IV.3.1. L'INTERPRETEUR.

L'interpréteur simule le fonctionnementcomplet de l'unité centrale d'un ordinateur IBM 360 standard. Rappelons qu'un modèle standard ne possède pas le mécanisme de traduction d'adresses virtuelles en adresses réelles (dynamic address translation), et n'utilise pas les registres de contrôle.

Cette restriction mise à part, l'interpréteur crée un environnement de simulation qui est invisible du programme testé (excepté le temps d'exécution). Deux entités de l'ordinateur doivent donc être traitées: les instructions et les interruptions.

L'interpréteur dispose de la mémoire non-simulée, interdite aux programmes et utilise l'unité centrale réelle, en particulier, les registres généraux et le mot d'état (program status word). De plus, pour garder le contrôle lors d'une interruption, le MONITEUR place dans la zone des nouveaux mots d'état de la machine, des mots d'état qui renvoient vers ses propres sous-programmes de traitement. Ces mêmes éléments (registres généraux, mot d'état, et zone des nouveaux mots d'état) correspondant au programme testé sont rangés dans la mémoire non simulée; nous les appellerons désormais registres simulés, mot d'état programme simulé, etc...(figure IV.2).

Bien que ces éléments soient constamment tenus à jour dans la mémoire, certains très fréquemment manipulés, sont aussi contenus dans les registres généraux afin d'accélérer l'exécution. En particulier le registre zéro contient le code-condition du mot d'état programme simulé, et le registre deux, le compteur ordinal simulé.

IV.3.1.1. Simulation d'une instruction.

Les instructions simulées sont réparties en groupes. Chaque groupe a ses propriétés caractéristiques: type de l'instruction, et modifications entraînées par les instructions qui le composent: registres, mémoire, code-condition, etc...Ce classement réduit le nombre de sous-programmes de simulation.

Le groupe de l'instruction est déterminé par le code opération qui sert d'index dans une table d'aiguillage, ceci s'effectuant dans la boucle principale du MONITEUR. Le sous-programme propre au groupe simule alors l'instruction testée en fournissant un résultat identique à celui qui aurait été normalement obtenu, et rend le contrôle à la boucle principale. Dans celle-ci le mot d'état programme simulé est mis à jour, et on commence à traiter l'instruction suivante (figure IV.6).

Nous étudierons la simulation des instructions classiques -qui ne sont ni des instructions de rupture de séquence, ni "EXECUTE"-, des instructions de ruptures de séquence, enfin de l'instruction "EXECUTE".

Les instructions classiques: Pour simuler une instruction classique on effectue plusieurs opérations:

a- analyse complète de l'instruction pour rechercher les registres qu'elle utilise.

b- chargement des registres qui serviront à la simulation avec la valeur des registres simulés utilisés.

```
MOV3   MONITEUR POUR LE SUPPORT 'PILOTE' ***29 AVRIL 1971***                                                    PAGE   41

FILE MONITEUR                                      CAMBRIDGE  SCIENTIFIC  CENTER

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                          F01JAN67   5/09/71

                                   1928 *TYPE53 - INSTRUCTIONS STH (40),ST (50) ** CC RX BD DD **          MON19280
00152E 4360 2001            00001  1929 TYPE53   IC    M6,1(,INST)          R1 ET RX DE L'INSTRUCTION      MON19290
001532 1876                        1930          LR    M7,M6                                               MON19300
001534 4280 E580            01580  1931          STC   M8,MOTY53            CODE OPERATION DANS INST. SIMULEE MON19310
001538 1B99                        1932          SR    M9,M9                                               MON19320
00153A 1474                        1933          NR    M7,M4                EFFACE R1                      MON19330
00153C 4780 E548            01548  1934          BC    8,*+12               S'IL N'Y A PAS DE RX           MON19340
001540 8970 0002            00002  1935          SLL   M7,2                 DEPLACEMENT DANS MOREG         MON19350
001544 5897 F280            00280  1936          L     M9,MOREG(M7)         M9= VALEUR DU RX               MON19360
001548 4370 2002            00002  1937          IC    M7,2(,INST)          REGISTRE DE BASE               MON19370
00154C 8860 0004            00004  1938          SRL   M6,4                 GARDE R1 SEULEMNT              MON19380
001550 8870 0004            00004  1939          SRL   M7,4                 GARDE LE REG DE BASE           MON19390
001554 8D60 0002            00002  1940          SLDL  M6,2                 DEPLACEMENT DANS MOREG         MON19400
001558 58C7 F280            00280  1941          L     M12,MOREG(M7)        M12=VALEUR DU REG DE BASE      MON19410
00155C 48B0 2002            00002  1942          LH    M11,2(,INST)         2NN HF DE L'INSTRUCTION        MON19420
001560 14B1                        1943          NR    M11,M1               MASQUE                         MON19430
001562 56B7 F2CC            002CC  1944          O     M11,MOD(M7)          ET REMASQUE                    MON19440
001566 58A6 F280            00280  1945          L     M10,MOREG(M6)        M10=INFORMATION A RANGER       MON19450
00156A 40B0 E570            01570  1946          STH   M11,*+6                                             MON19460
00156E 4199 0000            00000  1947          LA    M9,0(M9,0)           *** ADRESSE EFFECTIVE ***      MON19470
001572 5590 F268            00268  1948          CL    M9,MTP               ADRESSE DES NOUVEAUX PSWS      MON19480
001576 47C0 E58C            0158C  1949          BC    12,MO53LO            NON MODIFIES                   MON19490
00157A 159F                        1950          CLR   M9,M15               PROTECTION DU SUPPORT          MON19500
00157C 47A0 E83C            0183C  1951          BC    10,MODMP             CAUSE ERREUR D'ADRESSAGE       MON19510
001580 58A9 0000            00000  1952 MOTY53   L     M10,0(M9,0)          ******* INSTRUCTION POUR SIMULER *** MON19520
001584 4120 2004            00004  1953          LA    INST,4(,INST)        COMPTEUR ORDINAL               MON19530
001588 47F0 F3E8            003E8  1954          B     ML                   BOUCLE PRINCIPAL DU MONITEUR   MON19540
00158C 5590 F260            00260  1955 MO53LO   CL    M9,MOTTUM            TOUCHE AUX NOUVEAUX PSWS       MON19550
001590 47C0 E580            01580  1956          BC    12,MOTY53            NON, C'EST PARFAIT             MON19560
001594 4199 E720            01720  1957          LA    M9,MOPSW88-88(M9)    PSWS SIMULES                   MON19570
001598 47F0 E580            01580  1958          B     MOTY53               EXECUTION                      MON19580


                                   1960 *TYPE54 - INSTRUCTION EXECUTE (44) ** 44 RX BD DD **               MON19600
00159C 4380 2002            00002  1961 TYPE54   IC    M8,2(,INST)          BASE ET DEPLACEMENT            MON19610
0015A0 4370 2001            00001  1962          IC    M7,1(,INST)          R1 ET RX                       MON19620
0015A4 1867                        1963          LR    M6,M7                                               MON19630
0015A6 1B99                        1964          SR    M9,M9                                               MON19640
0015A8 1474                        1965          NR    M7,M4                OUBLI R1                       MON19650
0015AA 4780 E5B6            015B6  1966          BC    8,*+12               SI INDEX (RX) EGAL ZERO        MON19660
0015AE 8970 0002            00002  1967          SLL   M7,2                 DEPLACEMENT DANS MOREG         MON19670
0015B2 5897 F280            00280  1968          L     M9,MOREG(M7)         M9=RX                          MON19680
0015B6 8880 0004            00004  1969          SRL   M8,4                 GARDE BASE                     MON19690
0015BA 8980 0002            00002  1970          SLL   M8,2                 DEPLACEMENT DANS MOREG         MON19700
0015BE 58C8 F280            00280  1971          L     M12,MOREG(M8)        M12=REGISTRE DE BASE           MON19710
0015C2 48B0 2002            00002  1972          LH    M11,2(,INST)         BASE ET DEPLACEMENT            MON19720
0015C6 14B1                        1973          NR    M11,M1               GARDE LE DEPLACEMENT           MON19730
0015C8 56B8 F2CC            002CC  1974          O     M11,MOD(M8)          M12=CONTIENT L'ADRESSE DE BASE MON19740
0015CC 9200 E62B      0162B        1975          MVI   MOEXFL+1,X'00'       BRANCHEMENT ET FLAG            MON19750
0015D0 40B0 E5D6            015D6  1976          STH   M11,*+6              CONSTRUIT L'INSTRUCTION        MON19760
0015D4 41B9 0000            00000  1977          LA    M11,0(M9,0)          DETERMINE L' ADRESSE EFFECTIVE MON19770
```

instructions "store" -ST et STH-

instruction "execute"

EXTRAITS DE L'INTERPRETEUR .../...

-EX-

EXTRAITS DE L'INTERPRETEUR .../...


```
FILE MONITEUR                                          CAMBRIDGE  SCIENTIFIC  CENTER

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                                     F01JAN67   5/09/71

0015D8 9544 B000      00000        1978          CLI   0(M11),X'44'         EST-CE EXECUTE DE EXECUTE            MON19780
0015DC 4770 E5E4            015E4  1979          BNE   *+8                  NON,                                 MON19790
0015E0 4400 E5E0            015E0  1980          EX    0,*                  OUI,CAUSE EXECUTE EXCEPTION          MON19800
0015E4 D205 F232 B000 00232 00000  1981          MVC   MOI31+2(6),0(M11)    PLACE L'INSTRUCTION A EXECUTER       MON19810
0015EA 8860 0004            00004  1982          SRL   M6,4                 SHIFT RX                             MON19820
0015EE 4780 E602            01602  1983          BC    8,MOC2               SI LE RESULTAT EST NUL               MON19830
0015F2 8960 0002            00002  1984          SLL   M6,2                 DEPLACEMENT DANS MOREG               MON19840
0015F6 4396 F283            00283  1985          IC    M9,MOREG+3(M6)       DERNIER OCTET                        MON19850
0015FA 5690 F230            00230  1986          O     M9,MOI31             FAIT LE "OU"                         MON19860
0015FE 4290 F233            00233  1987          STC   M9,MOI31+3           PLACE LE RESULTAT                    MON19870
001602 5020 F27C            0027C  1988 MOC2     ST    INST,MOMLR           SAUVE L'ADRESSE                      MON19880
001606 960F E7B4      017B4        1989          OI    MO2CHNG,X'0F'        INDIQUE L'INSTR. DANS MOMLR          MON19890
00160A 4120 F232            00232  1990          LA    INST,MOI31+2         POINTE L'INSTRUCTION *************** MON19900
00160E 58B0 E654            01654  1991          L     M11,MOEX2                                                 MON19910
001612 50B0 F3E8            003E8  1992          ST    M11,ML               POUR REVENIR                         MON19920
001616 0700                        1993          CNOP  0,4                                                       MON19930
001618 4380 2000            00000  1994 MOBC8    IC    M8,0(,INST)          RECOMMENCE                           MON19940
00161C 4358 F0F0            000F0  1995          IC    M5,MOINDX(M8)                                             MON19950
001620 07F5                        1996          BR    M5                   GO SIMULATION ******                 MON19960

                                   1998 *ENTRER ICI POUR LE RETOUR APRES L'EXECUTION                             MON19980
001622 58B0 E618            01618  1999 MOTY54   L     M11,MOBC8            RESTAURE L'INSTRUCTION               MON19990
001626 50B0 F3E8            003E8  2000          ST    M11,ML               RESTAURE DANS LA BOUCLE PRINCIPALE   MON20000
00162A 9100 F257      00257        2001 MOEXFL   TM    MOIRG+15,X'00'       TESTE AIGUILLAGE                     MON20010
00162E 47E0 E63E            0163E  2002          BC    14,MOXFLE                                                 MON20020
001632 9200 E9D5      019D5        2003          MVI   MOXRST+1,X'00'       RESTAURE L'AIGUILLAGE                MON20030
001636 94F0 E7B4      017B4        2004          NI    MO2CHNG,X'F0'        LE REGISTRE INST EST A JOUR          MON20040
00163A 47F0 F3E8            003E8  2005          B     ML                   *** RETOUR A LA BOUCLE PRINCIPALE ** MON20050
00163E 5820 F27C            0027C  2006 MOXFLE   L     INST,MOMLR           RESTAURE L'INSTRUCTION               MON20060
001642 9200 E9D5      019D5        2007          MVI   MOXRST+1,X'00'       RESTAURE LE FLAG                     MON20070
001646 94F0 E7B4      017B4        2008          NI    MO2CHNG,X'F0'        REGISTRE 2 OK                        MON20080
00164A 4120 2004            00004  2009          LA    INST,4(,INST)        COMPTEUR ORDINAL                     MON20090
00164E 47F0 F3E8            003E8  2010          B     ML                   RETOUR BOUCLE PRINCIPALE             MON20100
001652 0700                        2011          CNOP  0,4                                                       MON20110
001654 47F0 E622            01622  2012 MOEX2    B     MOTY54                                                    MON20120
```

instruction de

```
001632 9200 E9D5      019D5        2003          MVI   MOXRST+1,X'00'       RESTAURE L'AIGUILLAGE                MON20030
                                   2014 *TYPE55 - INSTRUCTION LPSW (82) ** 82 .. BD DD **                        MON20140
001658 8000 EAC0      01AC0        2015 TYPE55   SSM   MFF                  INTERDICTION SUR LES INTERRUPTIONS   MON20150
00165C 91F0 F42B      0042B        2016          TM    MOINTALT-1,X'F0'     INTERRUPTION EN ATTENTE              MON20160
001660 4780 F42C            0042C  2017          BC    8,MOINTALT           OUI, ALLONS LA TRAITER               MON20170
001664 48B0 2002            00002  2018          LH    M11,2(,INST)         REG DE BASE ET DEPLACEMENT           MON20180
001668 4380 2002            00002  2019          IC    M8,2(,INST)          REG BASE                             MON20190
00166C 8880 0004            00004  2020          SRL   M8,4                 GARDE UNIQUEMENT LE REG DE BASE      MON20200
001670 4780 E682            01682  2021          BC    8,MOTY55+4           S'IL EST NUL                         MON20210
001674 8980 0002            00002  2022          SLL   M8,2                 DEPLACEMENT                          MON20220
001678 58C8 F280            00280  2023          L     M12,MOREG(M8)        M12=REGISTRE DE BASE                 MON20230
00167C 14B1                        2024          NR    M11,M1               MASQUE                               MON20240
00167E 41BB C000            00000  2025 MOTY55   LA    M11,0(M11,M12)       VALEUR DE LA BASE ET DU DEPLACEMENT  MON20250
001682 18CB                        2026          LR    M12,M11              CONTROLE DE L'ADRESSE VALIDE         MON20260
```

```
FILE MONITEUR                                   CAMBRIDGE  SCIENTIFIC  CENTER

  LOC  OBJECT CODE      ADDR1 ADDR2  STMT    SOURCE STATEMENT                                               F01JAN67    5/09/71

001684 54C0 E6D0              016D0  2027              N     M12,MOHEX7          EST-CE UN DOUBLE-MOT                 MON20270
001688 4770 E6B0              016B0  2028              BNZ   MOLPSWSP            NON,CAUSE UNE INTERRUPTION SPECIALE  MON20280
00168C 55B0 F268              00268  2029              CL    M11,MTP             ADRESSE INF AUX NPSW                 MON20290
001690 47C0 E6B4              016B4  2030              BC    12,MO55LO           NON                                  MON20300
001694 98AB B000              00000  2031 MOA82        LM    M10,M11,0(M11)      SIMULATION DE L'INSTRUCTION********  MON20310
001698 50A0 E6C8              016C8  2032              ST    M10,MOB82                                                MON20320
00169C 180B                          2033              LR    M0,M11              M0=CODE CONDITION                    MON20330
00169E 90AB EA58              01A58  2034              STM   M10,M11,SYSNVIRN    SIMULATION *****                     MON20340
0016A2 D200 E6CC EA5C 016CC 01A5C    2035              MVC   MOB82+4(1),MSKPRG   CC-PM                                MON20350
0016A8 940F E6C9        016C9        2036              NI    MOB82+1,X'0F'       PROTECTION DES CLES OFF              MON20360
0016AC 8200 E6C8        016C8        2037              LPSW  MOB82               CHARGE PSW-ALLER A MLINT             MON20370
0016B0 8200 1000        00000        2038 MOLPSWSP     LPSW  0(M1)               CAUSE UNE INTERRUPT. (ADR IMPAIRE)   MON20380
0016B4 55B0 F260              00260  2039 MO55LO       CL    M11,MOTTUM          SUPERIEUR AUX NOUV. PSW              MON20390
0016B8 4740 E694              01694  2040              BC    4,MOA82             OUI,C'EST BON                        MON20400
0016BC 41BB E720              01720  2041              LA    M11,MOPSW88-88(M11) ADRESSE DANS NPSW SIMULES            MON20410
0016C0 47F0 E694              01694  2042              B     MOA82               EXECUTION                            MON20420
0016C4 07000700                      2043              CNOP  0,8                                                      MON20430
0016C8 00000000                      2044 MOB82        DC    F'0'                                                     MON20440
0016CC 000003E0                      2045 MOC82        DC    A(MLINT)                                                 MON20450
0016D0 00000007                      2046 MOHEX7       DC    F'7'                                                     MON20460


                                     2048 *TYPE56 - INSTRUCTION TS (93) **93 .. BD DD **                              MON20480
0016D4 4380 2002              00002  2049 TYPE56       IC    M8,2(,INST)         CODE                                 MON20490
0016D8 48A0 2002              00002  2050              LH    M10,2(,INST)                                             MON20500
0016DC 8880 0004              00004  2051              SRL   M8,4                GARDE LA BASE                        MON20510
0016E0 4780 E6F2              016F2  2052              BC    8,MO56CM            SI NULLE                             MON20520
0016E4 8980 0002              00002  2053              SLL   M8,2                DEPLACEMENT                          MON20530
0016E8 58C8 F280              00280  2054              L     M12,MOREG(M8)       M12=VALEUR DE LA BASE                MON20540
0016EC 14A1                          2055              NR    M10,M1              MASQUE NOTRE BASE                    MON20550
0016EE 41AC A000              00000  2056              LA    M10,0(M12,M10)      ADRESSE EFFECTIVE                    MON20560
0016F2 55A0 F268              00268  2057 MO56CM       CL    M10,MTP             AU-DESSUS DES NPSW                   MON20570
0016F6 47C0 E706              01706  2058              BC    12,MO56LO           NON,VOIR EN DESSOUS                  MON20580
0016FA 9300 A000        00000        2059 MOTY56       TS    0(M10)              SIMULATION ************************  MON20590
0016FE 4120 2004              00004  2060              LA    INST,4(,INST)       MISE A JOUR DU COMPTEUR ORDINAL      MON20600
001702 4500 F3E8              003E8  2061              BAL   M0,ML               SAUVE LE CC ET VA A LA BOUCLE PRINCI MON20610
001706 55A0 F260              00260  2062 MO56LO       CL    M10,MOTTUM                                               MON20620
00170A 47C0 E6FA              016FA  2063              BC    12,MOTY56           OK POUR LES PSW                      MON20630
00170E 41AA E720              01720  2064              LA    M10,MOPSW88-88(M10) PSW SIMULES                          MON20640
001712 47F0 E6FA              016FA  2065              B     MOTY56              EXECUTION                            MON20650


                                     2067 *TYPE57 INSTRUCTION SPECIALE POUR LE SKIP (D0)                              MON20670
000000                               2068              USING SKIPDST,M10                                              MON20680
001716 58A0 E770              01770  2069 TYPE57       L     M10,ASKIPTB         TABLE DES SKIP                       MON20690
00171A 41C0 0008              00008  2070              LA    M12,SKIPCNT         NOMBRE D'ENTREES                     MON20700
00171E 5020 F230              00230  2071              ST    INST,MO131          RANGE LE COMPT. ORDINAL              MON20710
001722 D502 F231 A001 00231 00001    2072 TY57COMP     CLC   MO131+1(3),1(M10)   COMPARE L'ADRESSE                    MON20720
```

chargement du "psw" -LPSW-

instruction "test & set" -TS-

EXTRAITS DE L'INTERPRETEUR .../...

```
MOV3   MONITEUR POUR LE SUPPORT 'PILOTE' ***29 AVRIL 1971***                                                   PAGE  44

FILE MONITEUR                                     CAMBRIDGE  SCIENTIFIC  CENTER

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                          F01JAN67   5/09/71

001728 4780 E738            01738  2073            BE    SKIPFO              ON A TROUVE L'ENTREE             MON20730
00172C 41A0 A004            00004  2074            LA    M10,4(,M10)         ADRESSE SUIVANTE                 MON20740
001730 46C0 E722            01722  2075            BCT   M12,TY57COMP                                         MON20750
001734 47F0 E844            01844  2076            B     MODMP6              PROVOQUE UN PROGRAMME CHECK      MON20760
001738 D200 2000 A000 00000 00000  2077 SKIPFO     MVC   0(1,INST),0(M10)    RESTAURE LE CODE OPERATION       MON20770
00173E D203 A000 E76C 00000 0176C  2078            MVC   0(4,M10),MSFFFFFF   ENTREE LIBRE                     MON20780
001744 48C0 A030            00030  2079            LH    M12,SKNUMBER        NOMBRE DE SKIP                   MON20790
001748 46C0 E754            01754  2080            BCT   M12,*+12            SI NON NUL                       MON20800
00174C D207 E788 A028 01788 00028  2081            MVC   MOPSW88+16(8),PRCHK+8    RESTAURE LE NPSW PROG SIMULE MON20810
001752 40C0 A030            00030  2082            STH   M12,SKNUMBER        MET A JOUR LE COMPTE             MON20820
001756 D703 F438 F43C 00438 0043C  2083            XC    TRACECNS+4(4),TRACECNS+8 BUFFER DE LA TRACE          MON20830
00175C D703 F43C F438 0043C 00438  2084            XC    TRACECNS+8(4),TRACECNS+4 POUR LES                    MON20840
001762 D707 F438 F43C 00438 0043C  2085            XC    TRACECNS+4(8),TRACECNS+8 INTERVERTIR                 MON20850
001768 47F0 F3E8            003E8  2086            B     ML                  RETOUR A LA BCL PRINCIPALE       MON20860

00176C 00FFFFFF                    2088 MSFFFFFF DC    XL4'FFFFFF'         MASQUE DE L'ENTREE LIBRE         MON20880
001770 00000000                    2089 ASKIPTB  DC    A(SKIPTBL)          TABLE DES SKIP                   MON20890
                                   2090          DROP  M10                                                  MON20900


000000                             2092 SKIPDST  DSECT                                                      MON20920
000000                             2093 SKIPTAB  DS    8F                  TABLE DES SKIP                   MON20930
000008                             2094 SKIPCNT  EQU   (*-SKIPTAB)/4       NOMBRE D'ENTREES                 MON20940
000020                             2095 PRCHK    DS    4F                  ZONE DE SAUVEGARDE               MON20950
000030                             2096 SKNUMBER DS    H                   NOMBRE DE SKIP ACTIFS            MON20960
```

EXTRAITS DE L'INTERPRETEUR.

c- contrôle que l'instruction ne fait pas une référence à la mémoire non simulée. Dans ce cas une interruption est réfléchie au programme, et indique une erreur d'adressage.

d- si l'instruction modifie la zone des nouveaux mots d'état programme, correction de l'adresse effective pour faire la modification dans la partie de la mémoire où ils ont été sauvés.

e- exécution d'une instruction équivalente à l'instruction traitée.

f- mise à jour éventuelle du code condition simulé, du ou des registres généraux simulés et du mot d'état programme simulé.

g- retour à la boucle principale du MONITEUR.

L'exécution de l'instruction peut provoquer une interruption, dans ce cas c'est le sous-programme du MONITEUR qui se charge de la réfléchir au programme simulé (voir IV.3.1.2).

Les instructions de rupture de séquence. Si ces instructions étaient exécutées comme précédemment décrit, l'interpréteur perdrait le contrôle. C'est pourquoi arrivé au moment de l'exécution d'une instruction équivalente, on recherche avec le code condition simulé si l'instruction doit effectivement rompre la séquence d'exécution. Dans ce cas on modifie simplement le compteur ordinal simulé avant de passer à la phase suivante.

L'instruction "EXECUTE". Cette instruction nécessite également un traitement spécial car elle modifie temporairement la valeur du compteur ordinal |Réf 13| .

Au lieu d'exécuter une instruction équivalente, on procède ainsi:

a- vérification que l'instruction référée, n'est pas aussi une instruction "EXECUTE". Dans ce cas on génère une interruption.

b- rangement de l'instruction référée dans la mémoire non simulée et modification de cette instruction comme le réaliserait l'"EXECUTE".

c- mise à jour du compteur ordinal simulé vers cette instruction recopiée dans la mémoire non simulée.

d- retour à la boucle principale du MONITEUR en l'avertissant qu'il traite une instruction "EXECUTE".

Après avoir ainsi simulé l'instruction référée, au retour dans la boucle principale, le MONITEUR reviendra au sous-programme de l'"EXECUTE". Le compteur ordinal simulé sera alors remis à sa juste valeur avant que le contrôle ne soit rendu à la boucle principale (voir figure IV.6).

IV.3.1.2 Traitement des interruptions

La simulation de l'unité centrale ne peut être complète que si le MONITEUR rend également compte des interruptions. Rappelons brièvement le mécanisme des interruptions sur les ordinateurs IBM 360:

Il existe cinq types d'interruption: externe, appel du superviseur, programme, erreur de la machine, et entrée-sortie. A chacun de ces types est associé deux adresses: l'une contenant le vieux mot d'état, l'autre le futur ou nouveau mot d'état. Ainsi on trouve dans la partie de la mémoire centrale qui correspond aux plus petites adresses:

Adresse	Contenu
24	ancien mot d'état EXTERNE
32	ancien mot d'état SVC
40	ancien mot d'état PROGRAMME

48	ancien mot d'état MACHINE
56	ancien mot d'état ENTREE/SORTIE
...	
88	nouveau mot d'état EXTERNE
96	nouveau mot d'état SVC
104	nouveau mot d'état PROGRAMME
112	nouveau mot d'état MACHINE
120	nouveau mot d'état ENTREE/SORTIE

Lorsqu'une interruption se produit, le mot d'état courant est rangé dans la partie du vieux mot d'état correspondant au type de l'interruption, et le nouveau mot d'état de cette interruption devient mot d'état courant.

C'est donc en changeant les 5 nouveaux mots d'état que le MONITEUR peut conserver le contrôle sur interruption. Il copie dans la mémoire non simulée, les nouveaux mots d'état du programme et les remplace par les siens (figure IV.2).

Sur interruption à réfléchir, le MONITEUR, remplacera donc l'ancien mot d'état correspondant par le mot d'état simulé, et placera dans ce dernier le nouveau mot d'état simulé.

Notons aussi que pour ne pas modifier la synchronisation entre le programme et les interruptions, le MONITEUR fonctionne en autorisant les mêmes interruptions que le programme testé.

Dans les traitements particuliers, certaines différences existent tout de même entre les diverses interruptions.

Interruption appel du superviseur: Cette interruption estla seule provoquée par une instruction: "SVC". Elle est aussi la seule à avoir deux origines:

a- un "SVC" du MONITEUR. Celui-ci emploie cette instruction pour appeler un sous-programme qui permette d'interdire les interruptions d'entrée-sortie et externe, afin de ne pas être détourné dans une séquence d'instructions. Ces interruptions ne doivent évidemment pas être réfléchies au programme; pour l'éviter, le MONITEUR prend toujours garde d'émettre un avertissement au sous-programme de traitement des interruptions de type "SVC" avant d'exécuter l'opération. Celui-ci étant averti, l'interruption ne sera pas réfléchie.

b- un "SVC" du programme. Quant aux "SVC" du programme, elles sont réfléchies comme nous l'avons décrit précédemment. Le sous-programme du MONITEUR dont c'est la tâche se termine en rendant le contrôle à la boucle principale afin de traiter l'instruction suivante.

Interruption externe et d'entrée-sortie. Ces interruptions arrivent de façon asynchrone, pendant la simulation d'une instruction. Le MONITEUR enregistre l'interruption, mais ne la réfléchit pas immédiatement. Il termine la simulation de l'instruction en cours. Mais avant, dans la boucle principale, de donner le contrôle au sous-programme chargé de simuler l'instruction suivante, le MONITEUR va tester si une interruption est en attente de fin de traitement; dans ce cas il réfléchit alors l'interruption au programme testé et revient dans la boucle principale avec le mot d'état simulé mis à jour.

S'il s'agit d'une interruption d'entrée-sortie, le MONITEUR doit encore distinguer deux cas:

a- l'interruption correspond à une opération lancée par le programme, elle est donc normalement traitée.

b- l'interruption est provoquée par une pseudo-opération (voir IV.4.3.2) destinée à restaurer l'un des canaux périphériques. Il faut alors recopier l'état du canal (channel status word) à partir de la pile où il était rangé, et réfléchir ensuite l'interruption.

Interruption-programme et erreur machine: Ces deux interruptions sont immédiatement et complètement traitées par le MONITEUR, dès qu'elles surviennent. Le sous-programme correspondant rend ensuite le contrôle à la boucle principale du MONITEUR. On note donc que l'instruction en cours de simulation n'est pas achevée, ceci étant conforme au fonctionnement normal.

IV.3.2. GESTIONS DES ENTREES-SORTIES.

Pour que le support PILOTE ait une indépendance fonctionnelle vis à vis du système supervisant les programmes testés, il est en particulier indispensable que les opérations sur les organes périphériques soient réalisées par des sous-programmes propres à PILOTE. Ces sous-programmes doivent de plus ne pas modifier l'état de ces organes périphériques, vis à vis du programme simulé.

Les programmes simulés et le MONITEUR ont en commun: le terminal de l'utilisateur -adresse 009- , la machine à imprimer -adresse 00E- et une bande magnétique -adresse 181-.Ces unités servent à produire des sorties d'informations: traces, collectes, images de la mémoire, etc...

Avant d'utiliser une unité d'entrée-sortie le MONITEUR teste donc si elle est déjà occupée par le programme simulé.

Si l'unité est libre l'opération est envoyée au canal, sa fin est attendue, et le contrôle est rendu.

Si l'unité est occupée, le MONITEUR sait qu'il s'agit d'une opération dûe au programme et attend qu'elle s'achève. Lorsqu'il reçoit le signal de la fin d'opération il conserve dans une pile tous les renseignements concernant le canal: son adresse, son état ("channel status word") et éventuellement ses octets d'états("sense bytes") si la condition de fin d'opération est accompagnée d'une erreur.

L'unité étant alors disponible, le MONITEUR exécute l'opération et attend le signal qui marque sa terminaison. Il s'agit à présent de remettre l'unité dans l'état où le MONITEUR l'avait trouvée: malheureusement

n'étant pas maître du canal et les phénomènes étant évidemment liés au temps, il est impossible de les reproduire exactement. Il suffit heureusement de faire en sorte que le programme testé retrouve dans les mots d'état du canal, celui de l'unité. Aussi le MONITEUR envoie une opération spéciale sur l'organe périphérique, une pseudo-opération, qui consiste à occuper le canal et éventuellement à provoquer une interruption. Sans attendre la réponse du canal, le contrôle est rendu.

Si le mot d'état programme simulé permet de recevoir des signaux du canal (interruptions autorisées), la simulation est interrompue par la terminaison de la pseudo-opération, et le sous-programme de traitement des entrées-sorties prend le contrôle. Si au contraire les interruptions d'entrée-sortie sont interdites, c'est le sous-programme chargé de la simulation des instructions telles que "TIO", "SIO", "TCH" et "HIO" qui, averti qu'une pseudo-opération a été lancée, saura en tenir compte.

Dans les deux cas, la marche à suivre est la même:

a- la pseudo-opération est reconnue.

b- le mot d'état du canal ("channel status word") est remplacé par celui qui avait été sauvegardé dans la pile, au moment où la fin de l'opération réelle a été enregistrée.

c- l'élément de la pile est effacé.

De cette façon, le programme testé retrouve les résultats de son entrée-sortie sans s'apercevoir qu'entre ce moment et celui où elle a été initialisée, d'autres ont pu s'exécuter.

Cette méthode risque cependant de provoquer la confusion dans le programme si le MONITEUR abandonne la simulation avant la fin de la pseudo-opération. Dans ce cas, en effet, c'est le programme testé qui récupère le signal du canal. Il reçoit donc des informations qui n'ont aucune relation avec l'opération qu'il a envoyée. Le MONITEUR ne peut donc abandonner le contrôle tant qu'une pseudo-opération n'est pas complètement terminée.

IV.3.3. LA TRACE ET LA COLLECTE.

L'interprétation des programmes pendant leur exécution permet d'obtenir dynamiquement des informations. Celles-ci peuvent être éditées et envoyées sur une unité de sortie-trace- ou conservées en mémoire-collecte-, Grâce aux requêtes (III.3.4), l'utilisateur peut réclamer ou refuser ces informations; elles fournissent une source de renseignements très intéressants, mais accroissent sensiblement le temps de simulation.

IV.3.3.1. La trace

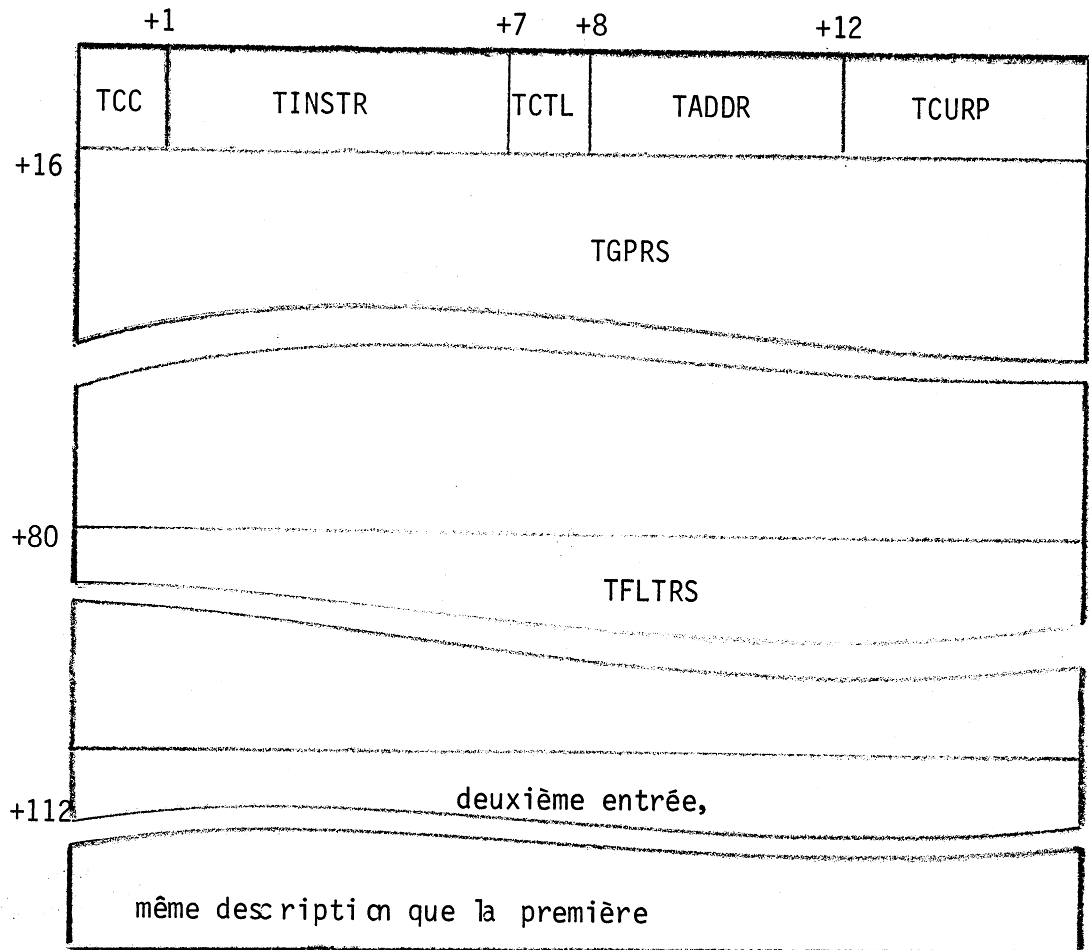
La trace est obtenue à partir de la boucle principale du MONITEUR (figure IV.6). Deux instructions sont traitées à la fois.

La première est celle qui vient d'être simulée et qui a donc modifié éventuellement les registres généraux, flottants et le mot d'état programme.

La seconde va être simulée par le MONITEUR: son adresse est déjà connue.

A chacune de ces instructions correspond deux zones (figure IV.7). Dans la première, le code condition simulé, les registres généraux et flottants, au cas où ils auraient été modifiés, sont copiés. Puis on effectue un travail d'édition dont le résultat est une ou plusieurs lignes, suivant le format désiré, qui sont imprimées. Dans la seconde on copie la mémoire contenant l'instruction, son adresse, et on indique si les registres généraux ou flottants peuvent être modifiés.

Le sous-programme de trace des interruptions est appelé sur l'occurrence de celles-ci. Il dispose d'une table dans laquelle il rânge l'ancien et le nouveau mot d'état programme simulé, de l'interruption, ainsi que son type. Dans la zone de la trace de l'instruction interrompue, il mentionne ce fait.



TCC	: code condition	TINSTR:	instruction en mémoire
TCTL	: bits de contrôle	TADDR :	adresse de l'instruction
	X'00' entrée disponible	TGPRS :	emplacement des 16 registres généraux
	X'10' instruction interrompue		
	X'03' registres généraux	TFLTRS:	emplacement des 16 registres flottants.
	X'08' registres flottants		
TRCURP :	1er mot du "psw" simulé		

FIG. IV.7 TRACINST

IV.3.3.2. La Collecte.

La collecte permet de conserver les n dernières instructions simulées. C'est une pile circulaire qui se remplit sans cesse: lorsqu'elle est complète, c'est la plus ancienne instruction qui laisse sa place (figure IV.5).

On tient continuellement à jour un pointeur vers la plus ancienne instruction et un autre vers la place où viendra l'instruction suivante; si on fait au moins un tour complet on l'indique (COLUSED figure IV.5).

CHAPITRE V

CONCLUSION

L'objectif initial du travail que nous venons de présenter, fut de réaliser un système général de contrôle d'un ordinateur en activité. Ce système, mettant à profit l'environnement CP-67, se compose de deux machines virtuelles [Réf 15] :

- l'une est la machine-objet (ordinateur fonctionnant sans conditions particulières) sur laquelle on désire recueillir une certaine quantité d'informations; celles-ci pouvant aussi bien concerner la mise au point que la description fonctionnelle du travail exécuté par la machine: nombre d'opérations sur les canaux périphériques, taille mémoire utilisée pendant un intervalle de temps, charge des modules, etc...

- l'autre est la machine-laboratoire, qui espionne la machine-objet, pour acquérir les informations demandées.

Ce projet est réalisé en deux étapes: la première est l'élaboration des programmes contenus dans la machine-laboratoire, pour assurer les fonctions de tests, de contrôles et d'entrées-sorties; la seconde est la mise en place, entre les deux ordinateurs, de liaisons qui permettent à la machine testée d'ignorer les mesures de performances dont elle est l'objet.

Ce que nous avons présenté dans cette thèse concerne la première étape, la seconde est développée par P. LEFEBVRE.

Ceci explique pourquoi nous avons accordé tant d'efforts

à l'indépendance entre le support PILOTE et les programmes pilotés:

-La mémoire centrale est partagée en deux régions,
- l'unité centrale appartient à PILOTE, le système piloté n'en a que la simulation,
- les organes périphériques sont restaurés dans leurs états initiaux si PILOTE les utilise.

Grâce à ce principe, PILOTE peut s'employer sur différents systèmes: soit il est amené en mémoire par les fonctions classiques de chargement, soit encore il est implanté en mémoire centrale avant tout autre programme afin d'en avoir le contrôle dès la première instruction. L'utilisation du support exige:

1-De résoudre le problème du chargement en mémoire, problème que nous avons résolu dans le cadre de CMS.

2-D'accepter une augmentation du temps d'exécution du programme piloté, entrainée par la simulation: cet accroissement du temps d'unité centrale dépend d'une part des demandes d'informations qui sont exigées du moniteur trace, collecte,...- et d'autre part du temps d'unité centrale qu'utiliserait le programme s'il s'exécutait normalement: l'interpréteur compense en effet les temps d'attente provenant des opérations sur les canaux. Pour un programme, sans sorties d'informations (c'est-à-dire en simulation seule), l'exécution devient 5 à 50 fois plus lente.

3-De se satisfaire d'une unité centrale standard. Dans l'état actuel de l'interpréteur, on ne peut simuler des programmes utilisant les possibilités "d'un ordinateur à mémoire paginée". Pour permettre ceci, il faut ajouter à l'interpréteur, la simulation des dispositifs spéciaux; entre autres:

a. le mécanisme de traduction des adresses virtuelles en adresses réelles.
b. l'extension du mot d'état programme
c. les clés de protection-mémoire, plus complètes.
d. une nouvelle description de la haute mémoire (page 0)
e. de nouveaux types d'interruptions: faute de page, faute de table de pages.

La réalisation de ces extensions permettrait de piloter des programmes utilisant la pagination et la segmentation, tel que CP-67 [Réf 15] .

La méthode d'interprétation, instruction machine par instruction machine, semble limiter l'utilisation de PILOTE aux programmes écrits en langage d'assembleur: une instruction d'assembleur étant équivalente à une instruction machine. Pour mettre au point un programme écrit en langage symbolique, il faut donc en connaître la liste équivalente en instructions d'assembleur, ceci est permis par la plupart des compilateurs.

Si cette opération peut présenter un intérêt pour les langages tels que FORTRAN, PL360 et ceux qui sont destinés à l'écriture de systèmes, il n'en est pas de même pour ceux qui sont beaucoup plus éloignés de l'instruction assembleur, c'est le cas d'ALGOL, PL/1, etc...

Pour ces derniers, les techniques de mise au point sont très différentes de celles que nous avons employées. Il faut soit faire une compilation incrémentielle et intéractive -ce qui nécessite une interprétation continue du programme -[Ref 27] soit faire une compilation classique, mais qui inclut dans le module objet généré des instructions spéciales pour appeler le programme de mise au point; cette technique a été adoptée pour le composant FORTBUG, destiné aux programmes FORTRAN [Réf 16] .

Quant à l'interface entre l'utilisateur et PILOTE, nous l'avons voulu agréable et intéractif:

- le langage de requêtes permet de spécifier en termes proches des instructions d'assembleur, les opérations de mise au point.

- le support évite au programmeur les manipulations qui peuvent être automatisées: exécution automatique des macro-requêtes, opérations arithmétiques, conversions...

- il fournit, par la simulation, les informations qui permettent d'anticiper les fautes de programmation, d'imposer des conditions de fonctionnement, de collecter des données.

- enfin le support est véritablement intéractif:

  * en outre du dialogue qui s'établit dans le mode conversationnel,

  * la simulation peut être abandonnée (et le programme s'exécute à grande vitesse) ou reprise au grè du programmeur, si les circonstances le demandent.

Il n'est cependant pas de conclusion définitive à une recherche, nous pensons maintenant qu'elle appartient à ceux qui emploieront le support et aux extensions qui seront développées à partir de celui-ci.

APPENDICE A

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

000000                              1 PRIME    CSECT
000000                              2          USING *,R15                ADRESSAGE
                                    3          SAVE  (14,12)              SAUVEGARDE
000000                              4+         DS    0H
000000 90EC D00C            0000C   5+         STM   14,12,12(13) SAVE REGISTERS
000004 F270 F0F0 0008 000F0 00008   6 PACK     PACK  ENTREE(8),8(R1)      SOUS FORME PACKED
00000A DE03 F151 F0F6 00151 000F6   7          ED    DONNEE(4),ENTREE+6
000010 4FF0 F0F0            000F0   8          CVB   R15,ENTREE           SOUS FORME BINAIRE
000014 12FF                         9          LTR   R15,R15              X=NUL OU NEGATIF ?
000016 4730 F024            00024  10          BC    3,OK                 NON
                                   11          RETURN (14,12),RC=4              ERREUR
00001A 98EC D00C            0000C  12+         LM    14,12,12(13) RESTORE THE REGISTERS
00001E 41F0 0004            00004  13+         LA    15,4(0,0) LOAD RETURN CODE
000022 07FE                        14+         BR    14 RETURN
000024 17BB                        15 OK       XR    R11,R11              REMISE A ZERO
000026 8CA0 0001            00001  16          SRDL  R10,1                DERNIER BIT DANS R11
00002A 12BB                        17          LTR   R11,R11              X PAIR OU IMPAIR
00002C 4770 F03C            0003C  18          BNZ   IMPAIR               X EST IMPAIR
000030 8CA0 001F            0001F  19          SRDL  R10,31               R11= X
000034 4AB0 F120            00120  20          AH    R11,H1               ON AJOUTE 1
000038 47F0 F044            00044  21          B     DEBUT
00003C 8CA0 001F            0001F  22 IMPAIR   SRDL  R10,31               R11= X
000040 4AB0 F122            00122  23          AH    R11,H2               ON AJOUTE 2
000044 4190 0003            00003  24 DEBUT    LA    R9,3                 R9=Y
000048 1859                        25 ENCORE   LR    R5,R9
00004A 1C45                        26          MR    R4,R5                R5= Y**2
                                   27 *
                                   28 *
00004C 4EB0 F100            00100  29          CVD   R11,ZONE             DECIMAL
000050 F377 F110 F100 00110 00100  30          UNPK  ZONE1(8),ZONE       ZONE
000056 96F0 F117      00117        31          OI    ZONE1+7,X'F0'        SIGNE
00005A 4E50 F108            00108  32          CVD   R5,ZONE2            DECIMAL
00005E F377 F118 F108 00118 00108  33          UNPK  ZONE21(8),ZONE2      ZONE
000064 96F0 F11F      0011F        34          OI    ZONE21+7,X'F0'        SIGNE
                                   35 *
                                   36 *
000068 195B                        37 TEST     CR    R5,R11               EST CE QUE Y**2 GT X
00006A 4720 F090            00090  38          BH    BON                  OUI C'EST X LE NOMBRE CHERCHE
00006E 4780 F088            00088  39          BE    INCR                 X NON PREMIER
000072 182B                        40          LR    R2,R11               PREPARE LA DIVISION
000074 8C20 0020            00020  41          SRDL  R2,32
000078 1D29                        42          DR    R2,R9                X/Y
00007A 1222                        43          LTR   R2,R2                DIVISIBLE
00007C 4780 F088            00088  44          BZ    INCR                 OUI, X NON PREMIER
000080 4A90 F122            00122  45          AH    R9,H2                PLUS 2
000084 47F0 F048            00048  46          B     ENCORE               ESSAYE AVEC LE NOUVEL Y
000088 4AB0 F122            00122  47 INCR     AH    R11,H2               PLUS 2
00008C 47F0 F044            00044  48          B     DEBUT
000090 4EB0 F100            00100  49 BON      CVD   R11,ZONE             DECIMAL
000094 DE03 F15A F106 0015A 00106  50          ED    RESULT(4),ZONE+6
00009A 4120 F0D8            000D8  51          LA    R2,CCWCONS           ADRESSE DU PROG CANAL
00009E 5020 0048            00048  52          ST    R2,CAW               DANS LE CAW
0000A2 4830 F124            00124  53          LH    R3,ADRCONS           ADRESSE DE LA CONSOLE
0000A6 9D00 3000      00000        54          TIO   0(R3)                TEST
```

```
 LOC   OBJECT CODE      ADDR1 ADDR2  STMT   SOURCE STATEMENT

0000AA 4770 F0A6              000A6   55           BC    7,*-4               BOUCLE SI OCCUPEE
0000AE 9C00 3000        00000         56           SIO   0(R3)               ECIT LE RESULTAT
0000B2 4770 F0AE              000AE   57           BC    7,*-4
0000B6 9D00 3000        00000         58           TIO   0(R3)
0000BA 4770 F0B6              000B6   59           BC    7,*-4
0000BE 4110 F0E8              000E8   60           LA    R1,DESLIST          FONCTION DESBUF
0000C2 0ACA                           61           SVC   202
0000C4 000000C8                       62           DC    AL4(*+4)
                                      63           RETURN (14,12),RC=0
0000C8 98EC D00C              0000C   64+          LM    14,12,12(13) RESTORE THE REGISTERS
0000CC 41F0 0000              00000   65+          LA    15,0(0,0) LOAD RETURN CODE
0000D0 07FE                           66+          BR    14 RETURN


0000D2 000000000000
0000D8 0100012660000038               68 CCWCONS   CCW   X'01',MSGCONS,SILI+CC,FINMSG
0000E0 0300000040000001               69           CCW   X'03',0,SILI,1
0000E8 C4C5E2C2E4C64040               70 DESLIST   DC    CL8'DESBUF'
0000F0 000000000000000C               71 ENTREE    DC    PL8'0'
0000F8 0000000000000000               72 SORTIE    DC    FL8'0'
000100 0000000000000000               73 ZONE      DC    FL8'0'
000108 0000000000000000               74 ZONE2     DC    FL8'0'
000110 0000000000000000               75 ZONE1     DC    FL8'0'
000118 0000000000000000               76 ZONE21    DC    FL8'0'
000120 0001                           77 H1        DC    H'1'
000122 0002                           78 H2        DC    H'2'
000124 0009                           79 ADRCONS   DC    H'9'                ADRESSE DE LA CONSOLE
000126 D3C540D5D6D4C2D9               80 MSGCONS   DC    C'LE NOMBRE PREMIER IMMEDIATEMENT SUPERIEUR A' '
000151 40202020                       81 DONNEE    DC    XL4'40202020'
000155 40C5E2E340                     82           DC    CL5' EST '
00015A 40202020                       83 RESULT    DC    XL4'40202020'
000038                                84 FINMSG    EQU   *-MSGCONS
000048                                85 CAW       EQU   X'48'
```

```
IPL 190

CMS..VERSION 3.0 (avril 71)

login pilote slc70000
PILOTE  AT 070000-  28K OF FREE STORAGE
R; T=0.31/0.83 11.35.02
```

"ipl" du système CMS et chargement du support PILOTE à l'adresse 70000.

```
load prime (type
 PRIME    AT 12000
R; T=0.03/0.11 11.35.16
```

chargement du programme à tester

```
debug
DEBUG   ENTERED...
break 1 12000
return
R; T=0.01/0.03 11.35.33
```

entrée dans le mode-conversation pour poser un point d'arrêt et retour à CMS

```
start prime 19
EXECUTION BEGINS...
DEBUG   ENTERED
BREAKPOINT 01 AT 012000
origin prime
INVALID ARGUMENT
qualify prime
addr prime
ADR     =  00012000
origin prime
trace on allg
numbinst 2
monitor on
-
```

demande de l'exécution

arrêt sur le point d'arrêt

définition des symboles du programme

définition de l'origine

le moniteur devient prêt
exécution et trace sur 2 instructions...

```
  012000   90ECD00C      STM    R1=14   R3=12    B2=13 D2=00C (00D6C4)

  012004   F270F0F00008  PACK   L1=7    L2=0     B1=15 D1=0F0 (0120F0)   B2=0  D2=008 (000008)

store pack x16'f277f0f01008'
num 1
go pack
```

La trace montre que l'instruction a été mal écrite : correction PACK ENTREE(8), 8(8,R1)
Ré-exécution de l'instruction.

```
  012004   F277F0F01008  PACK   L1=7    L2=7     B1=15 D1=0F0 (0120F0)   B2=1  D2=008 (00A488)

x entree
 ADR   =    00120F0       +000000019000000
```

le contrôle du résultat montre une erreur.

```
*gpr 1 x116
 ADR   =    000A480       D7D9C9D4 C5404040 F1F94040 40404040
*gpr 1 c116
 ADR   =    000A480       PRIME    19
sto a488,a cl8'00000019'
*g 1 cl16
 ADR   =    000A480       PRIME    00000019
go pack
```

mémoire pointée par le reg. 1 en hexadécimal, puis sous forme de caractères.
correction.
contrôle
ré-exécution

```
  012004   F277F0F01008  PACK   L1=7    L2=7     B1=15 D1=0F0 (0120F0)   B2=1  D2=008 (00A488)

x entree
 ADR   =    00120F0       +000000000000019
```

cette fois le résultat est correct.

```
-

  01200A   DE03F151F0F6  ED     L1=3             B1=15 D1=151 (012151)   B2=15 D2=0F6 (0120F6)
 ***PSW CHANGES***                                          CONDITION CODE=2
```

```
x donnee
 ADR   =    0012151        4040F1F9      examen de la mémoire avec son type implicite (hexadec)
x donnee c
 ADR   =    0012151          19          puis sous forme de caractères.
at test
->x zone1 cl8                            } définition d'une macro-requête.
->x zone21 cl8
->go
at 120be                                 puis d'une seconde.
->read
list at                                  } liste des macro-Requêtes
AT 0120BE    01
AT TEST      01
list at test                             liste d'une macro-requête particulière
X ZONE1 CL8
X ZONE21 CL8
GO
stop instruction de                      pose d'un stop sur l'instruction X'DE' (edit)
stop inst 44                               "     "   "    "        "       X'44' (execute)
list inst

STOP INSTRUCTION: DE (ED  )              } liste des stops sur instruction.
                  44 (EX  )
-                                        exécution avec trace (indication de tous les registres)

  012010   4FF0F0F0      CVB    R1=15  X2=0    B2=15 D2=0F0 (0120F0)
        -R0 00000121 -R1 0000A480 -R2 00007EBE -R3 6000D652 -R4 00000000 -R5 00000006 -R6 00000000 -R7 000083C6
        -R8 00000070 -R9 00000100 -10 0000A480 -11 5000D5F2 -12 0006FFD0 -13 0000D6B8 -14 00002776 *15 00000013

nu flow                                  interprétation continue.          R15 a été modifié
trace modg                               exécution
-

  012014   12FF          LTR    R1=15  R2=15

  012016   4730F024      BC            X2=0    B2=15 D2=024 (000037)   TEST CONDITION CODE=3 2

  000037   00FF          OP00
 DEBUG   ENTERED                         on passe en mode-conversation sur la faute de programmation
 PROGRAM INT.    PSW = 8100000660000039
st 10,r 4fa0                             correction.
x 10,r                                   vérification.
 ADR   =    0012010        4FA0F0F0
st 14,r 12aa                             modification de la mémoire pour corriger
gpr 15                                   valeur du registre 15
00000013
set gpr 15 00012000                      modification de celle-ci
gpr 15                                   vérification
00012000
go 10,r                                  exécution

  012010   4FA0F0F0      CVB    R1=10  X2=0    B2=15 D2=0F0 (0120F0)
        *10 00000013

  012014   12AA          LTR    R1=10  R2=10

  012016   4730F024      BC            X2=0    B2=15 D2=024 (012024)   TEST CONDITION CODE=3 2

  012024   17BB          XR     R1=11  R2=11
        *11 00000000
 ***PSW CHANGES***                                  CONDITION CODE=0

  012026   8CA00001      SRDL   R1=10          B2=0  D2=001 (000001)
        *10 00000009 *11 80000000
```

```
 01202A   12BB          LTR    R1=11  R2=11
***PSW CHANGES***                                    CONDITION CODE=1

 01202C   4770F03C      BC            X2=0    B2=15 D2=03C (01203C)   TEST CONDITION CODE=3 2 1

 01203C   8CA0001F      SRDL   R1=10          B2=0  D2=01F (00001F)
       *10 00000000 *11 00000013

 012040   4AB0F122      AH     R1=11  X2=0    B2=15 D2=122 (012122)
       *11 00000015
***PSW CHANGES***                                    CONDITION CODE=2

 012044   41900003      LA     R1=9   X2=0    B2=0  D2=003 (000003)
       *R9 00000003

 012048   1859          LR     R1=5   R2=9
       *R5 00000003

  CP
ex
```

passage sous CP pour générer une interruption externe et passer en mode-conversation

```
DEBUG   ENTERED, EXTERNAL INT.
collect on 50 gprs
trace fast
```

définition d'une collecte de 50 instructions.

exécution.

```
  01204C   4EB0F100      CVD
  012050   F377F110F100  UNPK
  012056   96F0F117      OI
  01205A   4E50F108      CVD
  01205E   F377F118F108  UNPK
  012064   96F0F11F      OI

AT TEST     01
X ZONE1 CL8
 ADR   =    0012110      00000021
X ZONE21 CL8
 ADR   =    0012118      00000009
GO
```

} exécution d'une macro-requête

```
  012068   195B          CR
  01206A   4720F090      BC
  01206E   4780F088      BC
   CP
ex
DEBUG   ENTERED, EXTERNAL INT.
trace bronly
```

passage sous CP pour générer une interruption externe

trace des instructions de rupture de séquence uniquement

exécution.

```
T 01207C   4780F088      BC
T 01208C   47F0F044      BC
```

] 2 ruptures de séquence

```
AT TEST     01
X ZONE1 CL8
 ADR   =    0012110      00000023
X ZONE21 CL8
 ADR   =    0012118      00000009
GO
```

} exécution d'une macro-requête

```
  [illegible]   [illegible]   BC
  01206E   4780F088      BC
  01207C   4780F088      BC
T 012084   47F0F048      BC
```

} ces 3 instructions n'ont pas provoqué de rupture de séquence,

→ il n'en est pas de même pour celle-ci

```
AT TEST     01
X ZONE1 CL8
 ADR   =    0012110      00000023
X ZONE21 CL8
 ADR   =    0012118      00000025
```

} exécution d'une macro-requête

```
go

T 01206A    4720F090       BC              arrêt de l'exécution sur la rencontre de l'instruction
STOP INSTRUCTION: DE (ED   ) AT 012094     édit en 12094.
n 1
trace tail allinst                          changement de format de la trace
—

  012094    DE03F15AF106   ED     L1=3.            B1=15 D1=15A (01215A)    B2=15 D2=106 (012106)

x msgcons
 ADR   =    0012126        LE NOMBRE PREMIER IMMEDIATEMENT SUPERIEUR A     affichage de msgcons.
x msgcons c156
 ADR   =    0012126        LE NOMBRE PREMIER IMMEDIATEMENT SUPERIEUR A  19 EST    23   idem mais sur 56
n flow                     simulation continue                                          caractères
—                          exécution

  01209A    4120F0D8       LA     R1=2    X2=0     B2=15 D2=0D8 (0120D8)
         *R2 000120D8

  01209E    50200048       ST     R1=2    X2=0     B2=0  D2=048 (000048)

  0120A2    4830F124       LH     R1=3    X2=0     B2=15 D2=124 (012124)
         *R3 00000009

  0120A6    9D003000       TIO                     B1=3  D1=000 (000009)
 ***PSW CHANGES***                                        CONDITION CODE=0

  0120AA    4770F0A6       BC             X2=0     B2=15 D2=0A6 (0120A6)    TEST CONDITION CODE=3 2 1

LE NOMBRE PREMIER IMMEDIATEMENT SUPERIEUR A  19 EST   23     message du SIO qui suit.

0120AE    9C003000       SIO                     B1=3  D1=000 (000009)

  0120B2    4770F0AE       BC             X2=0     B2=15 D2=0AE (0120AE)    TEST CONDITION CODE=3 2 1

  0120B6    9D003000       TIO                     B1=3  D1=000 (000009)

  0120BA    4770F0B6       BC             X2=0     B2=15 D2=0B6 (0120B6)    TEST CONDITION CODE=3 2 1

AT 0120BE    01              } exécution d'une macro-requête avec arrêt (READ).
READ                         }
psw
810000400001 20BE              mot d'état du programme.
—

  0120BE    4110F0E8       LA     R1=1    X2=0     B2=15 D2=0E8 (0120E8)
         *R1 000120E8

  0120C2    0ACA           SVC    CODE(DECIMAL)=202

 ***INTERRUPT**** SUPERVISOR CALL OLD PSW 810000CA400120C4 - NEW PSW 0004000000002330
 ***PSW CHANGES*** SYSTEM MASK=00000000          AMWP=0100
STOP ON INTERRUPT SVC          arrêt sur SVC.
on svc normal                  simulation sans arrêt sur les SVC suivants
trace off                      suppression de la trace
—                              exécution

 ***INTERRUPT**** SUPERVISOR CALL OLD PSW 000400CA40005884 - NEW PSW 0004000000002330
STOP INSTRUCTION: DE (ED   ) AT 00A908     arrêt sur l'instruction EDIT.
clear inst de                              annulation de cet arrêt.
ll inst

STOP INSTRUCTION: 44 (EX   )
clear inst                                 annulation de tous les arrêts sur instruction.
collect out                                sortie de la collecte.
```

```
00A902    D2081000C154  MVC
   GR 0-7 00000121  000120E8  000120D8  00000009  00000000  00000019  00000000  000083C6
   GR 8-F 00000070  00000005  00000000  00000017  0006FFD0  0000D6B8  00002776  00012000
00A8FE    5810C19C      L
   GR 0-7 00000121  000120E8  000120D8  00000009  00000000  00000019  00000000  000083C6
   GR 8-F 00000070  00000005  00000000  00000017  0006FFD0  0000D6B8  00002776  00012000
00A8FA    41000009      LA
   GR 0-7 00000121  000120E8  000120D8  00000009  00000000  00000019  00000000  000083C6
   GR 8-F 00000070  00000005  00000000  00000017  0006FFD0  0000D6B8  00002776  00012000
00A8F6    4E10C180      CVD
   GR 0-7 00000121  000120E8  000120D8  00000009  00000000  00000019  00000000  000083C6
   GR 8-F 00000070  00000005  00000000  00000017  0006FFD0  0000D6B8  00002776  00012000
00A8F2    5D00C14C      D
   GR 0-7 00000121  000120E8  000120D8  00000009  00000000  00000019  00000000  000083C6
   GR 8-F 00000070  00000005  00000000  00000017  0006FFD0  0000D6B8  00002776  00012000
cp

0005 INSTRUCTIONS HAVE BEEN PRINTED
monitor off

DEBUG
go
R; T=1.49/8.25 11.58.18
```

la collecte est imprimée sur le terminal, dans le sens contraire de l'exécution

la collecte est stoppée par la touche "attention". abandon de la simulation. confirmation du mode-conversation. exécution normale

fin du programme.

A P P E N D I C E   A

<><><><>

UN EXEMPLE DE SESSION

Dans cette session, nous nous proposons d'utiliser PILOTE pour tester l'exécution du programme PRIME. Celui-ci recherche le nombre premier immédiatement supérieur à un nombre donné: a, puis imprime le résultat sur le terminal, grâce aux instructions T10 et S10.

Après avoir chargé CMS à partir du disque d'adresse 190 -IPL 190-, on amène le support PILOTE à l'adresse hexadécimale 70000 -login pilote slc70000-. Un message confirme l'opération et précise la taille de la mémoire libre du support: 28 Koctets.

Le programme PRIME est ensuite chargé à l'adresse hexadécimale 12000 -load prime-.(Désormais nous indiquerons toujours les adresses en hexédécimale).

A partir de l'environnement CMS, on passe en mode conversationnel -debug- pour poser un point d'arrêt -break 1 12000- sur la première instruction du programme; on revient ensuite à CMS -return-.

On lance le programme PRIME, en lui demandant de rechercher le nombre premier immédiatement supérieur à 19 -start prime 19-. Sur le point d'arrêt, l'exécution est interrompue, le programmeur est invité à taper ses requêtes.

On se définit une origine avec le nom du programme -origine prime-. Cette requête est rejetée car le nom est inconnu. Par la requête suivante, on qualifie tous les symboles du programme -qualify prime-. On demande l'adresse absolue de PRIME -addr prime-, la réponse est 12000. On renouvelle la requête"origin ", cette fois avec succès. Dorénavant et jusqu'à la prochaine origine, celle-ci est fixée à 12000, début du programme.

On active la trace, en demandant tous les registres -trace on allg-.

On fixe le nombre d'instructions à simuler à 2 -numbinst 2-.
On active le moniteur -monitor on-.
On lance l'exécution à partir de l'adresse courante 12000 -"ligne vide"-...

PILOTE imprime sur le terminal la trace de 2 instructions. Grâce à celles-ci, on s'aperçoit que l'instruction PACK était incorrectement écrite:

PACK ENTREE(8),8(R1)   au lieu de PACK ENTREE(8),8(8,R1)

La correction est effectuée -store pack x16'f277f01008'-. On demande le pas à pas -num 1- et on ré-exécute l'instruction PACK -go pack-...

Le résultat est vérifié, c'est-à-dire que l'on demande la valeur de la variable ENTREE -x entree-. Elle est imprimée suivant le format défini dans le fichier assemblé: 8 octets sous forme "packed". Le résultat est incorrect, la faute en est au deuxième opérande. On demande le contenu de la mémoire pointée par le registre 1, sous forme hexadécimale -*gpr 1 x116- puis sous forme caractère -*gpr 1 c116-. Ce contenu est modifié par la requête suivante -store a488.a c18'00000019'-. On renouvelle l'instruction...

ENTREE est cette fois correcte. Le résultat étant satisfaisant, on demande l'exécution de l'instruction suivante, puis on examine la variable DONNEE sous sa forme pré-définie -x donnee- et sous forme de caractères -x donnee c-.

Par la requête suivante -at test- on passe en mode-AT, pour ranger dans une macro-requête, trois requêtes:

x zone1 c18
x zone21 c18
go

Après être revenu en mode conversationnel par la requête GO, on crée une nouvelle macro-requête. Celle-ci, contenant simplement READ,

permettra à son exécution de stopper le programme et de passer en mode conversationnel.

On demande la liste des "ats" -list at-.

On demande le contenu de la macro-requête TEST -list at test-

On définit un stop sur l'instruction EDIT (X'DE') -stop instruction de- et sur EXECUTE (X'44') -stop inst 44- puis on liste les stops instructions existants -list inst-.

On trace une nouvelle instruction; le registre 15, repéré par une astérisque, a été modifié.

De nouveau, en mode-conversation, on demande une simulation continue -nu flow- et une trace avec les seuls registres modifiés -trace modg-.

On relance l'exécution...

Sur une interruption programme, causée par un branchement à une adresse impaire (spécification), le mot d'état du programme estimprimé: PSW 8100000660000039, et on passe en mode conversation.

L'erreur provient de l'emploi du registre 15 comme registre de travail, alors qu'il est déjà utilisé comme registre de base.

L'instruction en adresse relative 10 (absolue 12010) est corrigée -st 10.r 4fa0- le registre 15 est remplacé par le registre 10.

On vérifie cette modification -x 10.r- et on corrige également l'instruction en 14 relatif -st 14.r 12aa-.

On demande la valeur du registre 15 -gpr 15- et on la modifie -set gpr 15 00012000-.

Puis on relance l'exécution, qui est toujours simulée, à partir de l'adresse relative 10: -go 10.r-.

Le programme testé s'exécute...

Après quelques instructions, on désire passer en mode conversation. Pour ce faire, on génère une interruption externe: passage sous CP-67, et commande "external" -ex-.

On définit alors une collecte de 50 instructions avec leurs registres -collect on 50 gprs- et on réclame la trace rapide des instructions -trace fast-.

L'exécution est relancée.

On repasse en mode-conversation avec l'interruption externe (commande "external" sous CP-67); on définit la trace pour les instructions de rupture de séquence seulement -trace bronly- et on repart...

La trace est imprimée, la macro-requête TEST s'exécute chaque fois que l'on passe sur l'instruction TEST. De nouveau on se trouve en mode-conversation sur l'occurrence de l'instruction EDIT: STOP INSTRUCTION: DE (ED)AT 12P94.

On demande le pas à pas -n 1- et une trace de toutes les instructions en format édité entier -trace tail allinst-. On examine la mémoire MSGCONS selon son type (caractères) et sa longueur pré-définie (43) -x msgcons-. La longueur étant insuffisante, on renouvelle la requête -x msgcons c156-.

Puis on relance le programme en simulation continue -n flow- par "retour de chariot".

Le programme est tracé...

A l'instruction 120BE la macro-requête définie s'exécute et débloque le clavier du terminal (READ).

On examine le mot d'état du programme -psw- et on relance l'exécution...

L'instruction SVC provoque une interruption pour appeler le superviseur de CMS. Sur ce type d'interruption, on passe en mode-conversationnel.

On demande la simulation normale sur les SVC suivants -on svc normal-.

On désactive la trace -trace off-, et on repart.

Cette fois, on est stoppé par une instruction EDIT: STOP INSTRUCTION: DE (ED) AT 00A908.

On efface ce stop -clear stop de-.

On liste tous les stops instructions restants -li inst-.

On les annule tous -clear inst-.

On demande l'impression de la collecte -collect out-. Celle-ci s'imprime sur le terminal en commençant par la dernière instruction exécutée (MVC), puis l'avant-dernière, etc...

Au bout de 5 instructions, on stoppe l'impression par la touche "attention" (passage sous CP-67 et touche "ATTN").

On désactive le moniteur -monitor off-.

Sur l'envoi d'une ligne vide ("retour de chariot seul"), le message DEBUG confirme que l'on est en mode-conversation, mais ne relance pas l'exécution. Il faut taper : "go".

Le programme s'exécute alors normalement, c'est-à-dire sans simulation, et se termine par le message: R; T=1.49/8.25 11.58.18.

A P P E N D I C E   B

```
***INTERRUPT**** INPUT/OUTPUT    OLD PSW 8100000980012000 - NEW PSW 00040000000005A8
***PSW CHANGES***                                         AMWP=0100

 0005A8   50C00000       ST      R1=12  X2=0    B2=0  D2=000 (000000)

 0005AC   05C0           BALR    R1=12  R2=0
        -R0 00000121 -R1 0000A480 -R2 00007EBE -R3 6000D6BA -R4 00000000 -R5 00000006 -R6 00000000 -R7 000083C6
        -R8 00000070 -R9 00000100 -10 0000A480 -11 5000D65A *12 400005AE -13 0000D720 -14 00002776 -15 00012000

 0005AE   900FC162       STM     R1=0   R3=15   B2=12 D2=162 (000710)

 0005B2   D203C1920000   MVC     L1=3           B1=12 D1=192 (000740)   B2=0  D2=000 (000000)

 0005B8   98030038       LM      R1=0   R3=3    B2=0  D2=038 (000038)
        *R0 81000009 *R1 80012000 *R2 0000B5A0 *R3 08000000 -R4 00000000 -R5 00000006 -R6 00000000 -R7 000083C6
        -R8 00000070 -R9 00000100 -10 0000A480 -11 5000D65A -12 400005AE -13 0000D720 -14 00002776 -15 00012000

 0005BC   41E0C0A0       LA      R1=14  X2=0    B2=12 D2=0A0 (00064E)
        -R0 81000009 -R1 80012000 -R2 0000B5A0 -R3 08000000 -R4 00000000 -R5 00000006 -R6 00000000 -R7 000083C6
        -R8 00000070 -R9 00000100 -10 0000A480 -11 5000D65A -12 400005AE -13 0000D720 *14 0000064E -15 00012000

 0005C0   984BC0F2       LM      R1=4   R3=11   B2=12 D2=0F2 (0006A0)
        -R0 81000009 -R1 80012000 -R2 0000B5A0 -R3 08000000 *R4 000007FF *R5 00000400 *R6 0000000C *R7 00000490
        *R8 00000000 *R9 00000000 *10 0000001C *11 00000000 -12 400005AE -13 0000D720 -14 0000064E -15 00012000

 0005C4   1440           NR      R1=4   R2=0
        -R0 81000009 -R1 80012000 -R2 0000B5A0 -R3 08000000 *R4 00000009 -R5 00000400 -R6 0000000C -R7 00000490
        -R8 00000000 -R9 00000000 -10 0000001C -11 00000000 -12 400005AE -13 0000D720 -14 0000064E -15 00012000
***PSW CHANGES***                                                  CONDITION CODE=1

 0005C6   1299           LTR     R1=9   R2=9
        -R0 81000009 -R1 80012000 -R2 0000B5A0 -R3 08000000 -R4 00000009 -R5 00000400 -R6 0000000C -R7 00000490
        -R8 00000000 -R9 00000000 -10 0000001C -11 00000000 -12 400005AE -13 0000D720 -14 0000064E -15 00012000
***PSW CHANGES***                                                  CONDITION CODE=0

 0005C8   4780C02C       BC             X2=0    B2=12 D2=02C (0005DA)   TEST CONDITION CODE=0

 0005DA   41F0C088       LA      R1=15  X2=0    B2=12 D2=088 (000636)
        -R0 81000009 -R1 80012000 -R2 0000B5A0 -R3 08000000 -R4 00000009 -R5 00000400 -R6 0000000C -R7 00000490
        -R8 00000000 -R9 00000000 -10 0000001C -11 00000000 -12 400005AE -13 0000D720 -14 0000064E *15 00000636

 0005DE   4190C040       LA      R1=9   X2=0    B2=12 D2=040 (0005EE)
        -R0 81000009 -R1 80012000 -R2 0000B5A0 -R3 08000000 -R4 00000009 -R5 00000400 -R6 0000000C -R7 00000490
        -R8 00000000 *R9 000005EE -10 0000001C -11 00000000 -12 400005AE -13 0000D720 -14 0000064E -15 00000636

 0005E2   91805002       TM      I2=128         B1=5  D1=002 (000402)   BIT I2=10000000

 0005E6   0789           BCR            R2=9                            TEST CONDITION CODE=0

 0005EE   8756C034       BXLE    R1=5   R3=6    B2=12 D2=034 (0005E2)
        -R0 81000009 -R1 80012000 -R2 0000B5A0 -R3 08000000 -R4 00000009 *R5 0000040C -R6 0000000C -R7 00000490
        -R8 00000000 -R9 000005EE -10 0000001C -11 00000000 -12 400005AE -13 0000D720 -14 0000064E -15 00000636

 0005E2   91805002       TM      I2=128         B1=5  D1=002 (00040E)   BIT I2=10000000

 0005E6   0789           BCR            R2=9                            TEST CONDITION CODE=0

 0005EE   8756C034       BXLE    R1=5   R3=6    B2=12 D2=034 (0005E2)
        -R0 81000009 -R1 80012000 -R2 0000B5A0 -R3 08000000 -R4 00000009 *R5 00000418 -R6 0000000C -R7 00000490
        -R8 00000000 -R9 000005EE -10 0000001C -11 00000000 -12 400005AE -13 0000D720 -14 0000064E -15 00000636
```

```
***INTERRUPT**** INPUT/OUTPUT    OLD PSW 8100000980012000 - NEW PSW 00040000000005A8

 0005AC   05C0          BALR   R1=12  R2=0
       -R0 00000121 -R1 0000A480 -R2 00007E8E -R3 60000D6BA -R4 00000000 -R5 00000006 -R6 00000000 -R7 000083C6
       -R8 00000070 -R9 00000100 -10 0000A480 -11 5000D65A *12 400005AE -13 0000D720 -14 00002776 -15 00012000

T 0005C8   4780C02C      BC            X2=0    B2=12 D2=02C (0005DA)   TEST CONDITION CODE=0

T 0005E6   0789          BCR           R2=9                            TEST CONDITION CODE=0

T 0005EE   8756C034      BXLE   R1=5   R3=6    B2=12 D2=034 (0005E2)
       -R0 81000009 -R1 80012000 -R2 0000B5A0 -R3 08000000 -R4 00000009 *R5 0000040C -R6 0000000C -R7 00000490
       -R8 00000000 -R9 000005EE -10 0000001C -11 00000000 -12 400005AE -13 0000D720 -14 0000064E -15 00000636

T 0005E6   0789          BCR           R2=9                            TEST CONDITION CODE=0

T 0005EE   8756C034      BXLE   R1=5   R3=6    B2=12 D2=034 (0005E2)
       -R0 81000009 -R1 80012000 -R2 0000B5A0 -R3 08000000 -R4 00000009 *R5 00000418 -R6 0000000C -R7 00000490
       -R8 00000000 -R9 000005EE -10 0000001C -11 00000000 -12 400005AE -13 0000D720 -14 0000064E -15 00000636

T 0005E6   0789          BCR           R2=9                            TEST CONDITION CODE=0

T 0005EE   8756C034      BXLE   R1=5   R3=6    B2=12 D2=034 (0005E2)
       -R0 81000009 -R1 80012000 -R2 0000B5A0 -R3 08000000 -R4 00000009 *R5 00000424 -R6 0000000C -R7 00000490
       -R8 00000000 -R9 000005EE -10 0000001C -11 00000000 -12 400005AE -13 0000D720 -14 0000064E -15 00000636

T 0005E6   0789          BCR           R2=9                            TEST CONDITION CODE=0

T 0005EE   8756C034      BXLE   R1=5   R3=6    B2=12 D2=034 (0005E2)
       -R0 81000009 -R1 80012000 -R2 0000B5A0 -R3 08000000 -R4 00000009 *R5 00000430 -R6 0000000C -R7 00000490
       -R8 00000000 -R9 000005EE -10 0000001C -11 00000000 -12 400005AE -13 0000D720 -14 0000064E -15 00000636

T 0005E6   0789          BCR           R2=9                            TEST CONDITION CODE=0

T 0005EE   8756C034      BXLE   R1=5   R3=6    B2=12 D2=034 (0005E2)
       -R0 81000009 -R1 80012000 -R2 0000B5A0 -R3 08000000 -R4 00000009 *R5 0000043C -R6 0000000C -R7 00000490
       -R8 00000000 -R9 000005EE -10 0000001C -11 00000000 -12 400005AE -13 0000D720 -14 0000064E -15 00000636

T 0005E6   0789          BCR           R2=9                            TEST CONDITION CODE=0

T 0005EE   8756C034      BXLE   R1=5   R3=6    B2=12 D2=034 (0005E2)
       -R0 81000009 -R1 80012000 -R2 0000B5A0 -R3 08000000 -R4 00000009 *R5 00000448 -R6 0000000C -R7 00000490
       -R8 00000000 -R9 000005EE -10 0000001C -11 00000000 -12 400005AE -13 0000D720 -14 0000064E -15 00000636

T 0005E6   0789          BCR           R2=9                            TEST CONDITION CODE=0

T 0005EE   8756C034      BXLE   R1=5   R3=6    B2=12 D2=034 (0005E2)
       -R0 81000009 -R1 80012000 -R2 0000B5A0 -R3 08000000 -R4 00000009 *R5 00000454 -R6 0000000C -R7 00000490
       -R8 00000000 -R9 000005EE -10 0000001C -11 00000000 -12 400005AE -13 0000D720 -14 0000064E -15 00000636

T 0005E6   0789          BCR           R2=9                            TEST CONDITION CODE=0

T 0005EE   8756C034      BXLE   R1=5   R3=6    B2=12 D2=034 (0005E2)
       -R0 81000009 -R1 80012000 -R2 0000B5A0 -R3 08000000 -R4 00000009 *R5 00000460 -R6 0000000C -R7 00000490
       -R8 00000000 -R9 000005EE -10 0000001C -11 00000000 -12 400005AE -13 0000D720 -14 0000064E -15 00000636

T 0005E6   0789          BCR           R2=9                            TEST CONDITION CODE=0

T 0005EE   8756C034      BXLE   R1=5   R3=6    B2=12 D2=034 (0005E2)
       -R0 81000009 -R1 80012000 -R2 0000B5A0 -R3 08000000 -R4 00000009 *R5 0000046C -R6 0000000C -R7 00000490
       -R8 00000000 -R9 000005EE -10 0000001C -11 00000000 -12 400005AE -13 0000D720 -14 0000064E -15 00000636
```

```
0005EE   8756C034      BXLE  R1=5   R3=6    B2=12 D2=034 (0005E2)
      *R5 0000049C

0005F2   5850C0F6      L     R1=5   X2=0    B2=12 D2=0F6 (0006A4)
      *R5 00000400

0005F6   49405000      CH    R1=4   X2=0    B2=5  D2=000 (000400)

0005FA   078F          BCR          R2=15                           TEST CONDITION CODE=0

000636   58F05008      L     R1=15  X2=0    B2=5  D2=008 (000408)
      *15 0000CCB0

00063A   5050C112      ST    R1=5   X2=0    B2=12 D2=112 (0006C0)

00063E   D21FC132C142  MVC   L1=31          B1=12 D1=132 (0006E0)   B2=12 D2=142 (0006F0)

000644   9003C152      STM   R1=0   R3=3    B2=12 D2=152 (000700)

000648   12FF          LTR   R1=15  R2=15
***PSW CHANGES***                                   CONDITION CODE=2

00064A   078E          BCR          R2=14                           TEST CONDITION CODE=0

00064C   05EF          BALR  R1=14  R2=15
      *14 6000064E

00CCB0   18DF          LR    R1=13  R2=15
      *13 0000CCB0

00CCB2   58B0D358      L     R1=11  X2=0    B2=13 D2=358 (00D008)
      *11 000001E8

00CCB6   48905000      LH    R1=9   X2=0    B2=5  D2=000 (000400)
      *R9 00000009

00CCBA   4090D282      STH   R1=9   X2=0    B2=13 D2=282 (00CF32)

00CCBE   18CE          LR    R1=12  R2=14
      *12 6000064E

00CCC0   91040044      TM    I2=4           B1=0  D1=044 (000044)   BIT I2=00000100
***PSW CHANGES***                                   CONDITION CODE=0

00CCC4   4710D036      BC           X2=0    B2=13 D2=036 (00CCE6)   TEST CONDITION CODE=3

00CCC8   91800044      TM    I2=128         B1=0  D1=044 (000044)   BIT I2=10000000

00CCCC   4710D1E6      BC           X2=0    B2=13 D2=1E6 (00CE96)   TEST CONDITION CODE=3

00CCD0   91080044      TM    I2=8           B1=0  D1=044 (000044)   BIT I2=00001000
***PSW CHANGES***                                   CONDITION CODE=3

00CCD4   4710D030      BC           X2=0    B2=13 D2=030 (00CCE0)   TEST CONDITION CODE=3

00CCE0   41F00008      LA    R1=15  X2=0    B2=0  D2=008 (000008)
      *15 00000008

00CCE4   07FE          BCR          R2=14                           TEST CONDITION CODE=3 2 1 0

00064E   9859E072      LM    R1=5   R3=9    B2=14 D2=072 (0006C0)
      *R6 000004A4 *R7 00000400
```

```
0022E6    05F0            BALR
0022E8    D20310008000    MVC
0022EE    1254            LTR
0022F0    4780F014        BC
0022FC    5890F03C        L
002300    947F9002        NI
002304    879AF018        BXLE
002300    947F9002        NI
002304    879AF018        BXLE
002300    947F9002        NI
002304    879AF018        BXLE
002300    947F9002        NI
002304    879AF018        BXLE
002300    947F9002        NI
002304    879AF018        BXLE
002300    947F9002        NI
002304    879AF018        BXLE
002300    947F9002        NI
002304    879AF018        BXLE
002300    947F9002        NI
002304    879AF018        BXLE
002300    947F9002        NI
002304    879AF018        BXLE
002300    947F9002        NI
002304    879AF018        BXLE
002300    947F9002        NI
002304    879AF018        BXLE
002300    947F9002        NI
002304    879AF018        BXLE
002300    947F9002        NI
002304    879AF018        BXLE
002308    18F0            LR
00230A    980B3000        LM
00230E    07FE            BCR
00D112    47F0D052        BC
00D07A    4820B03E        LH
00D07E    1222            LTR
00D080    4780D0A0        BC
00D0C8    9500B037        CLI
00D0CC    4770D0E0        BC
00D0D0    9500B035        CLI
00D0D4    4770D0E0        BC
00D0D8    9201B037        MVI
00D0DC    4120B050        LA
00D0E0    5020B038        ST
00D0E4    4860B03C        LH
00D0E8    4110B054        LA
00D0EC    58F0D524        L
00D0F0    05EF            BALR
00CED2    41701002        LA
00CED6    43801000        IC
00CEDA    5070B010        ST
00CEDE    4280B010        STC
00CEE2    940FB010        NI
00CEE6    43801001        IC
00CEEA    4280B017        STC
00CEEE    4180B010        LA
00CEF2    50800048        ST
```

```
002300    947F9002       NI
 GR 0-7 00000000  00000218  C3D6D5F1  00000750  00000000  00000000  0000001C  00000000
 GR 8-F 000004A4  00000400  0000000C  00000490  0000D028  0000D028  4000D112  4000224C
002304    879AF018       BXLE
 GR 0-7 00000000  00000218  C3D6D5F1  00000750  00000000  00000000  0000001C  00000000
 GR 8-F 000004A4  0000040C  0000000C  00000490  0000D028  0000D028  4000D112  400022E8
002300    947F9002       NI
 GR 0-7 00000000  00000218  C3D6D5F1  00000750  00000000  00000000  0000001C  00000000
 GR 8-F 000004A4  00000400  0000000C  00000490  0000D028  0000D028  4000D112  4000224C
0022FC    5890F03C       L
 GR 0-7 00000000  00000218  C3D6D5F1  00000750  00000000  00000000  0000001C  00000000
 GR 8-F 000004A4  00000400  0000000C  00000490  0000D028  0000D028  4000D112  400022E8
0022F0    4780F014       BC
 GR 0-7 00000000  00000218  C3D6D5F1  00000750  00000000  00000000  0000001C  00000000
 GR 8-F 000004A4  00000400  0000000C  00000490  0000D028  0000D028  4000D112  4000224C
0022EE    1254           LTR
 GR 0-7 00000000  00000218  C3D6D5F1  00000750  00000000  00000000  0000001C  00000000
 GR 8-F 000004A4  00000400  0000000C  00000490  0000D028  0000D028  4000D112  400022E8
0022E8    D20310008000   MVC
 GR 0-7 00000000  00000218  C3D6D5F1  00000750  00000000  00000000  0000001C  00000000
 GR 8-F 000004A4  00000400  0000000C  00000490  0000D028  0000D028  4000D112  4000224C
0022AC    8200F0C4       LPSW
 GR 0-7 00000000  00000214  C3D6D5F1  00000750  00000000  00000000  0000001C  00000000
 GR 8-F 000004A4  00000400  0000000C  00000490  0000D028  0000D028  4000D112  4000224C
0022A8    41101004       LA
 GR 0-7 00000000  00000218  C3D6D5F1  00000750  00000000  00000000  0000001C  00000000
 GR 8-F 000004A4  00000400  0000000C  00000490  0000D028  0000D028  4000D112  4000224C
0022A4    4770F010       BC
 GR 0-7 00000000  00000214  C3D6D5F1  00000750  00000000  00000000  0000001C  00000000
 GR 8-F 000004A4  00000400  0000000C  00000490  0000D028  0000D028  4000D112  4000224C
0022A0    59001000       C
 GR 0-7 00000000  00000210  C3D6D5F1  00000750  00000000  00000000  0000001C  00000000
 GR 8-F 000004A4  00000400  0000000C  00000490  0000D028  0000D028  4000D112  4000224C
00229C    41101004       LA
 GR 0-7 00000000  00000214  C3D6D5F1  00000750  00000000  00000000  0000001C  00000000
 GR 8-F 000004A4  00000400  0000000C  00000490  0000D028  0000D028  4000D112  4000224C
002298    4710F064       BC
 GR 0-7 00000000  00000210  C3D6D5F1  00000750  00000000  00000000  0000001C  00000000
 GR 8-F 000004A4  00000400  0000000C  00000490  0000D028  0000D028  4000D112  4000224C
002294    91209002       TM
 GR 0-7 00000000  00000210  C3D6D5F1  00000750  00000000  00000000  0000001C  00000000
 GR 8-F 000004A4  00000490  0000000C  00000490  0000D028  0000D028  4000D112  4000224C
002290    96809002       OI
 GR 0-7 00000000  00000210  C3D6D5F1  00000750  00000000  00000000  0000001C  00000000
 GR 8-F 000004A4  00000400  0000000C  00000490  0000D028  0000D028  4000D112  4000224C
00227A    4780F044       BC
 GR 0-7 00000000  00000210  C3D6D5F1  00000750  00000000  00000000  0000001C  00000000
 GR 8-F 000004A4  00000400  0000000C  00000490  0000D028  0000D028  4000D112  4000224C
002276    55209004       CL
 GR 0-7 00000000  00000210  C3D6D5F1  00000750  00000000  00000000  0000001C  00000000
 GR 8-F 000004A4  00000400  0000000C  00000490  0000D028  0000D028  4000D112  4000224C
002272    5890F0D8       L
 GR 0-7 00000000  00000210  C3D6D5F1  00000750  00000000  00000000  0000001C  00000000
 GR 8-F 000004A4  00000400  0000000C  00000490  0000D028  0000D028  4000D112  4000224C
002262    4780F026       BC
 GR 0-7 00000000  00000210  C3D6D5F1  00000750  00000000  00000000  0000001C  00000000
 GR 8-F 000004A4  00000400  0000000C  00000490  0000D028  0000D028  4000D112  4000224C
002260    1254           LTR
 GR 0-7 00000000  00000210  C3D6D5F1  00000750  00000000  00000000  0000001C  00000000
 GR 8-F 000004A4  00000400  0000000C  00000490  0000D028  0000D028  4000D112  4000224C
```

Appendice 8- COLLECTE avec les registres generaux.

A P P E N D I C E   C

APPENDICE C

<><><><>

LA COMMANDE SYMBOL

Objet:

Cette commande de CMS crée un fichier SYMBOL DEBUGGIN P5 à partir d'un ou plusieurs fichiers de type SYM.

Format:

```
SYMBOL   filename1   <...filenameN >
```

filename est le nom du fichier de type SYM.

Utilisation:

Pour utiliser les requêtes "QUALIFY" et "USING" de PILOTE (III.3.4), le programmeur doit avoir assemblé son travail avec l'option "test". Ceci provoque la création d'un fichier de type SYM ayant le nom du programme.
La commande SYMBOL crée à partir de un ou plusieurs de ces fichiers, un autre qui se nomme SYMBOL DEBUGGIN et qui éventuellement remplace l'ancien déjà existant. C'est à partir de ce fichier, contenant la description symbolique du programme que "QUALIFY" et "USING", pourront reconnaître les noms tapés ultérieurement.

Réponse:

FILE filename1
FILE filename2
...
ces messages s'impriment à mesure que sont traités les fichiers "filename" SYM.

Messages d'erreur:

filename NOT FOUND
si un des fichiers de type SYM n'existe pas, ce message est imprimé et on passe au suivant.

LAST EXTRN NAME IS X
le maximum de noms externes et de "dsect" étant 144, les suivants ne sont pas pris comme tels et ne peuvent donc pas faire l'objet d'une requête "QUALIFY" ou "USING". Par contre ils seront accessibles comme un nom ordinaire.

Exemple:

```
asmg prime données (test)  ← assemblage avec l'option
R;T=4.48/11.42 17.12.35      "test" de prime et de données.

asmg prime2                ← assemblage sans l'option
R;T=1.07/4.09  17.15.02      "test" de prime2

symbol prime prime2 données
FILE PRIME
FILE PRIME2
PRIME2 NOT FOUND
FILE DONNEES
R;T=0.19/0.87 17.20.55
```

APPENDICE D

A P P E N D I C E D

<><><><>

LA COMMANDE TAPPILOT

Objet:

Cette commande de CMS crée un fichier de type SCRIPT à partir d'une bande magnétique contenant les sorties de PILOTE (requête "PRINT" III.3.4).

Format:

```
TAPPILOT  filename    < number >
```

filename est le nom du fichier créé
number est le nombre de lignes que l'on veut copier.

Utilisation:

A partir d'une bande magnétique d'adresse 181, la commande permet de créer un fichier CMS dont le nom est le premier argument, le type SCRIPT et le mode P1.
La copie commence de l'endroit où est positionnée la bande (voir la commande TAPEIO dans le manuel d'utilisation de CMS [Réf 15]) jusqu'à la rencontre d'une marque de fin de bande où jusqu'à ce que le nombre de lignes -optionnel- soit atteint.

Le fichier étant créé, l'utilisateur peut le consulter en l'imprimant ou le traiter en l'éditant, etc...

Exemple:

```
tapeio fsf       ←   positionnement sur le fichier suivant
R;T=0.01/0.01 18.16.32
tappilot tracout 50  ← création d'un fichier script
R;T=0.02/0.07  18.20.05  de 50 lignes.
```

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

(1) ADIBA M. — Edition par contexte dans un environnement de temps partagé. Thèse de 3ème cycle. Faculté des Sciences de Grenoble. 1971.

(2) BERNSTEIN W.A. and OWENS J.T. — Debugging in a time sharing environment.-FJCC 1968.

(3) BERTHAUD M. et SAVARY — Un macro-langage pour un langage de commandes. Application au support de mise au point PILOTE. Rapport de DEA Institut de Mathématiques Appliquées de Grenoble- à paraître.

(4) BOILEN S. — A time-sharing Debugging System for a small computer. proc SJCC p 51 1963.

(5) BRADY P.T. — Writing an online debugging program for the experienced user. Communications of the ACM volume 11/number 6/june 68.

(6) CORBATO F.J. — Description of the FAPBUG Symbolic Debugging Program. MIT february 1964.

(7) CRISMAN, P.A. — (editor) FAPBUG et MADBUG dans CTSS: a programmer's guide 2nd edition the MIT Press.1965.

(8) DIEHM I.C. — "Computer Aids to Code Checking" proc of the ACM meeting at Toronto 1952 p 19.

(9) EVANS T.G.and DARLEY D.L. — Online debugging techniques: a survey. FJCC 1966.

(10) GRIFFITHS M. Analyse déterministe et compilateurs. Thèse d'état. Faculté des Sciences de Grenoble.

(11) GRISHMAN R. AIDS All Purpose Interactive Debugging System User's manual. Institute of math.science, New York University.

(12) HALPERN M. "Computer Programming: The debugging Epoch opens" Computers and Automation, volume 14, november 1965 p 28.

(13) IBM System/360 Principles of operation A22-6821-7

(14) IBM System/360 Operating system. TESTRAN C28-6648-1

(15) IBM Cambridge Scientific Center CP-67/CMS.User's guide ref.320-215.

(16) JACOLIN FORTBUG un accompagnateur fortran. Centre Scientifique IBM et Institut de Mathématiques Appliquées de Grenoble. 1970.

(17) KNUTH D.E. The art of computer programming. Vol 1, fundamental algorithms, Addison-Wesley publishing co, 1968.

(18) KULSRUD H.E. HELPER an interactive extensible debugging system.

(19) LYNCH W.C. Description of a high capacity fast turnaround university computing center, communications of the ACM volume 9/p 117 1966.

(20) SACKMAN H. ERIKSON W.J.and GRANT E.E. Exploratory experimental studies comparing online and offline programming performance. Communications of the ACM volume 11/number 1/january 1968.

(21) SDS Reference manual. Computing Center.memo 12.

(22) SOFTWARE ENGINEERING Report on a conference sponsored by the NATO SCIENCE COMMITTEE. Garmisch, Germany 7th to 11th october 1968.

(23) SOFTWARE ENGINEERING Report on a conference sponsored by the NATO SCIENCE COMMITTEE. Rome, Italie 27th to 31th October 1969.

(24) STOCKTON GAINES The debugging of computer programs. Institute for defense analyses. Von Neumann Hall.Princeton.august 1969.

(25) VAN HORN E.C. Three criteria for designing computing systems to facilitate debugging. Communications of the ACM volume 11/number 5/may 1968.

(26) LEFEBVRE P. Adressage Symbolique de PILOTE. Note interne. Institut de Mathématiques Appliquées de GRENOBLE. Mai 1971.

(27) PECCOUD, GRIFFITHS PELTIER Incremential Interactive Compilation. University of Grenoble. IBM France Scientific Centre. Proc of the IFIP Congress 1968, p 384.