Machine PASC-HLL : réalisation avec des micro-processeurs en tranches d'une unité centrale multiprocesseur adaptée au langage PASCAL

Gérard Baille

THESE

présentée à

l' Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR DE L'INPG
"Génie Informatique"

par

Gérard BAILLE

MACHINE PASC-HLL : REALISATION AVEC DES MICROPROCESSEURS EN TRANCHES D'UNE UNITE CENTRALE MULTIPROCESSEUR ADAPTEE AU LANGAGE PASCAL.

Thèse soutenue le 24 octobre 1983 devant la Commission d'Examen :

Monsieur	Louis BOLLIET	: Président
Messieurs	François ANCEAU Vincent CORDONNIER Gérard NOGUEZ	} Examinateurs

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année universitaire 1982-1983

Président de l'Université : D. BLOCH

Vice-Président : René CARRE
Hervé CHERADAME
Marcel IVANES

PROFESSEURS DES UNIVERSITES :

ANCEAU François	E.N.S.I.M.A.G.
BARRAUD Alain	E.N.S.I.E.G.
BAUDELET Bernard	E.N.S.I.E.G.
BESSON Jean	E.N.S.E.E.G.
BLIMAN Samuel	E.N.S.E.R.G.
BLOCH Daniel	E.N.S.I.E.G.
BOIS Philippe	E.N.S.H.G.
BONNETAIN Lucien	E.N.S.E.E.G.
BONNIER Etienne	E.N.S.E.E.G.
BOUVARD Maurice	E.N.S.H.G.
BRISSONNEAU Pierre	E.N.S.I.E.G.
BUYLE BODIN Maurice	E.N.S.E.R.G.
CAVAIGNAC Jean-François	E.N.S.I.E.G.
CHARTIER Germain	E.N.S.I.E.G.
CHENEVIER Pierre	E.N.S.E.R.G.
CHERADAME Hervé	U.E.R.M.C.P.P.
CHERUY Arlette	E.N.S.I.E.G.
CHIAVERINA Jean	U.E.R.M.C.P.P.
COHEN Joseph	E.N.S.E.R.G.
COUMES André	E.N.S.E.R.G.
DURAND Francis	E.N.S.E.E.G.
DURAND Jean-Louis	E.N.S.I.E.G.
FELICI Noël	E.N.S.I.E.G.
FOULARD Claude	E.N.S.I.E.G.
GENTIL Pierre	E.N.S.E.R.G.
GUERIN Bernard	E.N.S.E.R.G.
GUYOT Pierre	E.N.S.E.E.G.
IVANES Marcel	E.N.S.I.E.G.
JAUSSAUD Pierre	E.N.S.I.E.G.
JOUBERT Jean-Claude	E.N.S.I.E.G.
JOURDAIN Geneviève	E.N.S.I.E.G.
LACOUME Jean-Louis	E.N.S.I.E.G.
LATOMBE Jean-Claude	E.N.S.I.M.A.G.

.../...

LESSIEUR Marcel	E.N.S.H.G.
LESPINARD Georges	E.N.S.H.G.
LONGEQUEUE Jean-Pierre	E.N.S.I.E.G.
MAZARE Guy	E.N.S.I.M.A.G.
MOREAU René	E.N.S.H.G.
MORET Roger	E.N.S.I.E.G.
MOSSIERE Jacques	E.N.S.I.M.A.G.
PARIAUD Jean-Charles	E.N.S.E.E.G.
PAUTHENET René	E.N.S.I.E.G.
PERRET René	E.N.S.I.E.G.
PERRET Robert	E.N.S.I.E.G.
PIAU Jean-Michel	E.N.S.H.G.
POLOUJADOFF Michel	E.N.S.I.E.G.
POUPOT Christian	E.N.S.E.R.G.
RAMEAU Jean-Jacques	E.N.S.E.E.G.
RENAUD Maurice	U.E.R.M.C.P.P.
ROBERT André	U.E.R.M.C.P.P.
ROBERT François	E.N.S.I.M.A.G.
SABONNADIERE Jean-Claude	E.N.S.I.E.G.
SAUCIER Gabrielle	E.N.S.I.M.A.G.
SCHLENKER Claire	E.N.S.I.E.G.
SCHLENKER Michel	E.N.S.I.E.G.
SERMET Pierre	E.N.S.E.R.G.
SILVY Jacques	U.E.R.M.C.P.P.
SOHM Jean-Claude	E.N.S.E.E.G.
SOUQUET Jean-Louis	E.N.S.E.E.G.
VEILLON Gérard	E.N.S.I.M.A.G.
ZADWORNY François	E.N.S.E.R.G.

PROFESSEURS ASSOCIES

BASTIN Georges	E.N.S.H.G.
BERRIL John	E.N.S.H.G.
CARREAU Pierre	E.N.S.H.G.
GANDINI Alessandro	U.E.R.M.C.P.P.
HAYASHI Hirashi	E.N.S.I.E.G.

PROFESSEURS UNIVERSITE DES SCIENCES SOCIALES (Grenoble II)

BOLLIET Louis
Chatelin Françoise

PROFESSEURS E.N.S. Mines de Saint-Etienne

RIEU Jean
SOUSTELLE Michel

CHERCHEURS DU C.N.R.S.

FRUCHART Robert	Directeur de Recherche
VACHAUD Georges	Directeur de Recherche

.../...

ALLIBERT Michel	Maître de Recherche
ANSARA Ibrahim	Maître de Recherche
ARMAND Michel	Maître de Recherche
BINDER Gilbert	
CARRE René	Maître de Recherche
DAVID René	Maître de Recherche
DEPORTES Jacques	
DRIOLE Jean	Maître de Recherche
GIGNOUX Damien	
GIVORD Dominique	
GUELIN Pierre	
HOPFINGER Emil	Maître de Recherche
JOUD Jean-Charles	Maître de Recherche
KAMARINOS Georges	Maître de Recherche
KLEITZ Michel	Maître de Recherche
LANDAU Ioan-Dore	Maître de Recherche
LASJAUNIAS J.C.	
MERMET Jean	Maître de Recherche
MUNIER Jacques	Maître de Recherche
PIAU Monique	
PORTESEIL Jean-Louis	
THOLENCE Jean-Louis	
VERDILLON André	

CHERCHEURS du MINISTERE de la RECHERCHE et de la TECHNOLOGIRE (Directeurs et Maîtres de Recherches, ENS Mines de St. Etienne)

LESBATS Pierre	Directeur de Recherche
BISCONDI Michel	Maître de Recherche
KOBYLANSKI André	Maître de Recherche
LE COZE Jean	Maître de Recherche
LALAUZE René	Maître de Recherche
LANCELOT Francis	Maître de Recherche
THEVENOT François	Maître de Recherche
TRAN MINH Canh	Maître de Recherche

PERSONNALITES HABILITEES à DIRIGER des TRAVAUX de RECHERCHE (Décision du Conseil Scientifique)

ALLIBERT Colette	E.N.S.E.E.G.
BERNARD Claude	E.N.S.E.E.G.
BONNET Rolland	E.N.S.E.E.G.
CAILLET Marcel	E.N.S.E.E.G.
CHATILLON Catherine	E.N.S.E.E.G.
CHATILLON Christian	E.N.S.E.E.G.
COULON Michel	E.N.S.E.E.G.
DIARD Jean-Paul	E.N.S.E.E.G.
EUSTAPOPOULOS Nicolas	E.N.S.E.E.G.
FOSTER Panayotis	E.N.S.E.E.G.

.../...

GALERIE Alain	E.N.S.E.E.G.
HAMMOU Abdelkader	E.N.S.E.E.G.
MALMEJAC Yves	E.N.S.E.E.G. (CENG)
MARTIN GARIN Régina	E.N.S.E.E.G.
NGUYEN TRUONG Bernadette	E.N.S.E.E.G.
RAVAINE Denis	E.N.S.E.E.G.
SAINFORT	E.N.S.E.E.G. (CENG)
SARRAZIN Pierre	E.N.S.E.E.G.
SIMON Jean-Paul	E.N.S.E.E.G.
TOUZAIN Philippe	E.N.S.E.E.G.
URBAIN Georges	E.N.S.E.E.G. (Laboratoire des ultra-réfractaires ODEILLON)
GUILHOT Bernard	E.N.S. Mines Saint Etienne
THOMAS Gérard	E.N.S. Mines Saint Etienne
DRIVER Julien	E.N.S. Mines Saint Etienne
BARIBAUD Michel	E.N.S.E.R.G.
BOREL Joseph	E.N.S.E.R.G.
CHOVET Alain	E.N.S.E.R.G.
CHEHIKIAN Alain	E.N.S.E.R.G.
DOLMAZON Jean-Marc	E.N.S.E.R.G.
HERAULT Jeanny	E.N.S.E.R.G.
MONLLOR Christian	E.N.S.E.R.G.
BORNARD Guy	E.N.S.I.E.G.
DESCHIZEAU Pierre	E.N.S.I.E.G.
GLANGEAUD François	E.N.S.I.E.G.
KOFMAN Walter	E.N.S.I.E.G.
LEJEUNE Gérard	E.N.S.I.E.G.
MAZUER Jean	E.N.S.I.E.G.
PERARD Jacques	E.N.S.I.E.G.
REINISCH Raymond	E.N.S.I.E.G.
ALEMANY Antoine	E.N.S.H.G.
BOIS Daniel	E.N.S.H.G.
DARVE Félix	E.N.S.H.G.
MICHEL Jean-Marie	E.N.S.H.G.
OBLED Charles	E.N.S.H.G.
ROWE Alain	E.N.S.H.G.
VAUCLIN Michel	E.N.S.H.G.
WACK Bernard	E.N.S.H.G.
BERT Didier	E.N.S.I.M.A.G.
CALMET Jacques	E.N.S.I.M.A.G.
COURTIN Jacques	E.N.S.I.M.A.G.
COURTOIS Bernard	E.N.S.I.M.A.G.
DELLA DORA Jean	E.N.S.I.M.A.G.
FONLUPT Jean	E.N.S.I.M.A.G.
SIFAKIS Joseph	E.N.S.I.M.A.G.
CHARUEL Robert	U.E.R.M.C.P.P.
CADET Jean	C.E.N.G.
COEURE Philippe	C.E.N.G. (LETI)

.../...

DELHAYE Jean-Marc	C.E.N.G. (STT)
DUPUY Michel	C.E.N.G. (LETI)
JOUVE Hubert	C.E.N.G. (LETI)
NICOLAU Yvan	C.E.N.G. (LETI)
NIFENECKER Hervé	C.E.N.G.
PERROUD Paul	C.E.N.G.
PEUZIN Jean-Claude	C.E.N.G. (LETI)
TAIEB Maurice	C.E.N.G.
VINCENDON Marc	C.E.N.G.

LABORATOIRES EXTERIEURS

DEMOULIN Eric	C.N.E.T.
DEVINE	C.N.E.T. (R.A.B.)
GERBER Roland	C.N.E.T.
MERCKEL Gérard	C.N.E.T.
PAULEAU Yves	C.N.E.T.
GAUBERT C.	I.N.S.A. Lyon

Je tiens à remercier

Monsieur BOLLIET, professeur à l'IUT d'informatique, qui a bien voulu me faire l'honneur de présider le jury de cette thèse

Monsieur CORDONNIER, professeur à l'université de LILLE, qui a bien voulu juger ce travail

Monsieur NOGUEZ, professeur à l'Institut de programmation, qui a accepté de faire partie de ce jury

Monsieur ANCEAU, professeur à l'INPG, qui a tout mis en œuvre pour que cette réalisation voit le jour

Monsieur SCHOELLKOPF, mon coéquipier avec qui je partage la paternité de PASC-HLL et qui a contribué à l'élaboration de ce document

Monsieur LAURENT qui nous a fortement aidé pour la réalisation matérielle de la machine et la conception du processeur PME

Les membres de l'équipe d'architecture des ordinateurs

Les membres de l'atelier d'électronique

Madame DIAZ, qui a assuré la dactylographie de ce texte dans des conditions très difficiles

et enfin le service de reprographie qui a assuré le tirage de ce document avec sa compétence habituelle

G. BAILLE

A mon épouse Anne-Marie
et à ma fille Delphine

TABLE DES MATIERES

DEUXIEME PARTIE

INTRODUCTION

Il est nécessaire de présenter le contexte dans lequel ce travail se situe.
Il représente le troisième volet d'un vaste projet de conception et de réalisation d'une machine adaptée au langage PASCAL et devant illustrer une approche méthodologique "descendante".

Dans sa thèse R.FORTIER a donné une ébauche de définition d'un langage intermédiaire (ce qui constitue la première étape d'une conception descendante), et une architecture adaptée au problème.

J.P SCHOELLKOPF a concrétisé cette approche par la description d'une machine "pipe-line" et la présentation des aspects système de PASC-HLL.

Cette troisième thèse traite de la réalisation et de la mise au point de la machine physique. Cette étude est donc le fruit d'un travail d'équipe épaulée par l'équipe de recherche d'architecture des ordinateurs dirigée par F. ANCEAU au sein du laboratoire IMAG et soutenue financièrement par l'INRIA et le CNRS.

L'ouvrage se décompose en deux parties :

Dans la première sont abordé les aspects méthodologique de conception d'une machine informatique.

Le chapitre I est consacré à une présentation succinte de la méthodologie de conception descendante d'une unité centrale d'ordinateur et à quelques réflexions sur la définition d'un langage intermédiare interprétable, à la décomposition fonctionnelle du travail d'une unité centrale conduisant à l'architecture "pipe-line" proposée.

Le chapitre II décrit de manière fonctionnelle les cinq processeurs de PASC-HLL.
Pour chacun d'eux la même démarche de conception descendante a été suivie et a conduit à des architectures très spécialisées.
Le chapitre III illustre cette démarche et les réflexions qu'elle suscite pour l'un des processeurs, le processeur de gestion des files.

La deuxième partie est consacrée à la description technique et à la

réalisation de chacun des quatre autres processeurs. Les problèmes de conception rencontrés et les choix qu'ils ont fait naître sont commentés.

En annexe on trouvera la présentation du système MADAM (Matériel d'Aide au Développement d'Application Microprogrammée) conçu et réalisé pour mettre au point les microprogrammes, ainsi que les microprogrammes symboliques des cinq processeurs.

chapitre I

ASPECTS METHODOLOGIQUES

I.1. METHODOLOGIE DE CONCEPTION DESCENDANTE [1] [2]

Quel que soit l'objet qu'il doit concevoir, tout créateur doit avoir présent à l'esprit la finalité de cet objet. Celui-ci doit répondre aux spécifications initiales qui en ont motivé l'étude conformément au principe de Peter:

> "Si vous ne savez pas où vous allez, vous avez de fortes chances de vous retrouver ailleurs" [L.J.PETER, R.HULL, 1969].

L'architecte d'ordinateur qui doit définir une machine spécialisée n'échappe pas à la règle. Il doit être guidé par la classe de problème que la machine doit traiter (types de programmes des utilisateurs futurs) et se poser sans arrêt les deux questions suivantes:

- quel est le projet à réaliser (projet)?

- de quels moyens dispose-t-il pour atteindre le but fixé (moyens)?

Deux "écoles" s'affrontent pour la conception d'un calculateur:

- une méthode dite ascendante qui consiste à regrouper des fonctions de plus en plus complexes susceptibles de satisfaire les désirs des utilisateurs.
  Nous pensons que cette technique classique conduit à des calculateurs pouvant être universels, c'est-à-dire capables de résoudre de nombreux problèmes et entre autres, ceux pour lesquels il a été conçu, mais de façon non optimale. Il ne sera pas forcément adapté aux besoins de ces futurs utilisateurs humains.

> "Pour aller quelque part, en général, le plus simple était encore de partir de là où l'on voulait aller, et avec du temps et un peu de chance, on y arrivait effectivement" [J.ROUXEL, Les Shadocks].

- une méthode dite descendante qui, à partir des spécifications des utilisateurs futurs (projet) définit progressivement une architecture adaptée, jusqu'à la réalisation physique qui constitue le but de la démarche.
L'évolution technologique (circuits LSI et VLSI en particulier) autorise une telle approche pour définir des architectures nouvelles spécialisées et donc mieux adaptées à leur utilisation.

I.1.1. Niveaux d'interprétation

Généralement, les ordinateurs sont organisés de manière "verticale": la complexité du traitement va en décroissant depuis les langages de haut niveau, jusqu'aux langages directement interprétés par un assemblage de circuits électroniques.

> "Une machine électronique peut être vue comme un empilement de niveaux d'interprétation de langages de plus en plus simples qui constitue une hiérarchie: chaque niveau est tel qu'il émule les niveaux qui le précèdent dans la hiérarchie" [SCHOELLKOPF 77].

Si l'on représente un calculateur sous forme de graphe dans le formalisme des systèmes hiérarchisés [3] on voit apparaître de façon claire ces niveaux d'interprétation (figure 1).

> "D'un point de vue intuitif, une interprétation peut être vue comme l'exécution et l'enchaînement de primitives (instructions) soumises par un élément (le processus) à une machine décrite par un algorithme (le processeur) en vue de leur exécution. Cet algorithme peut soit s'exécuter localement, soit soumettre de nouvelles primitives à son propre processeur et ainsi de suite jusqu'à ce qu'il n'y ait plus de sous-traitance" [ANCEAU 74].

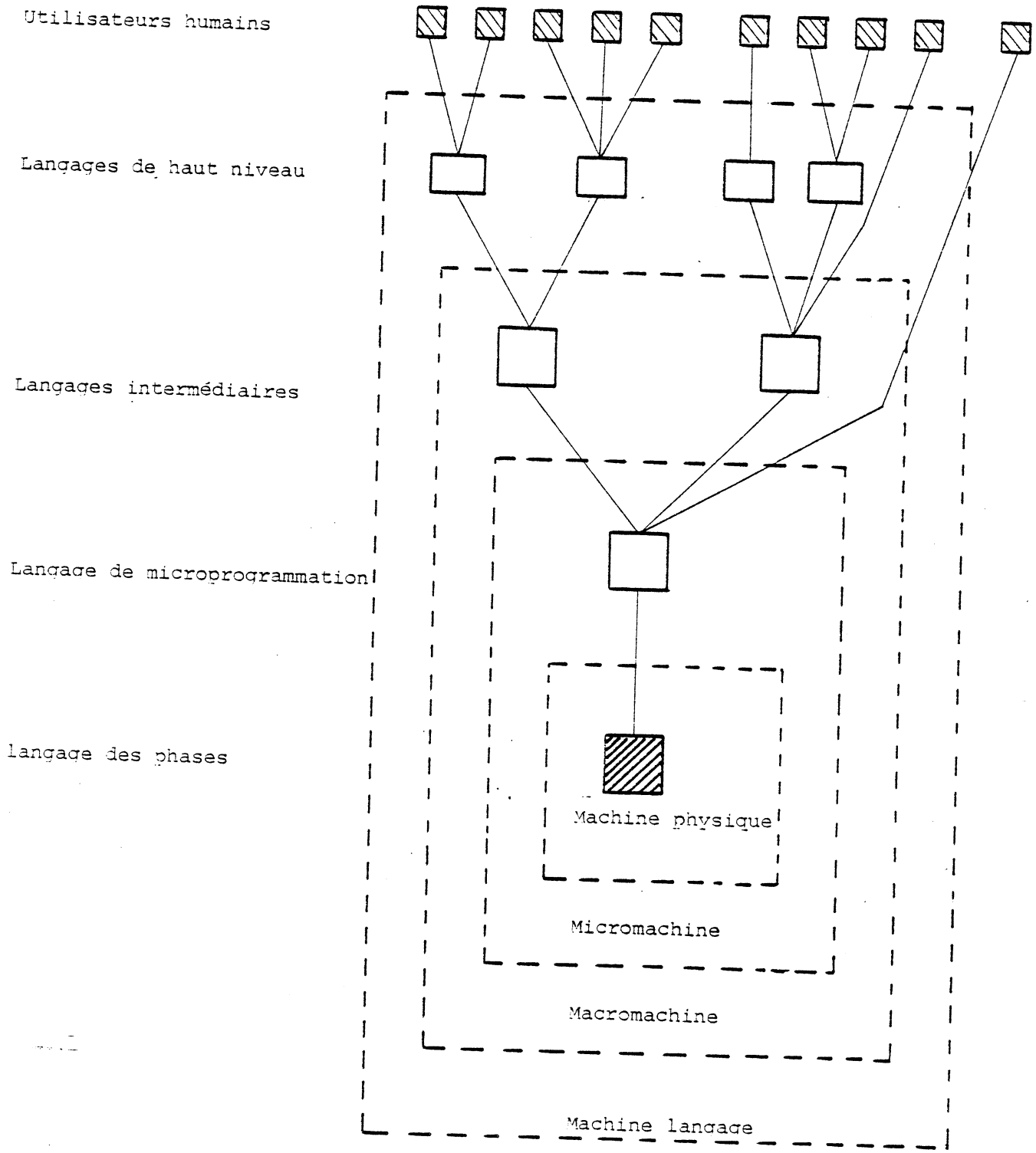


Figure I.1 Organisation verticale d'un ordinateur

Compte tenu de la finalité d'un calculateur (répondre aux besoins des utilisateurs en respectant certaines contraintes de coût, performances, fiabilité...), une conception progressive guidée par les spécifications initiales des utilisateurs (programmes) et tenant compte de cette hiérarchie, semble être envisageable. Le problème de l'architecte est alors ramené à la construction d'un système hiérarchisé à partir de fonctions devant être réalisées au niveau des feuilles d'une manière descendante depuis les feuilles utilisateurs jusqu'à la machine physique.

Considérons une machine adaptée au traitement d'un langage de haut niveau (PASCAL par exemple).

L'utilisateur communique avec la machine langage par l'intermédiaire de programmes écrits en PASCAL.

La première étape d'une démarche descendante va consister à interpréter de manière optimale ces programmes.

Cette étude des programmes peut conduire à la définition d'un langage intermédiaire (LI) directement issu du langage de haut niveau et qui soit interprétable. C'est l'étape de traduction des chaînes du langage évolué en des chaînes d'un langage d'instructions adapté et interprétable.

Le langage LI est par définition interprétable et l'on peut définir son algorithme d'interprétation. Celui-ci est conditionné par les utilisateurs du langage de haut niveau dont les programmes types écrits en LI permettent de définir les caractéristiques de leur interprétation et de mesurer le comportement dynamique de l'interpréteur de LI.

Une réalisation de l'algorithme d'interprétation de LI qui tienne compte de l'influence du but à atteindre peut donc être envisagée.

On choisira par exemple un ensemble de primitives qui deviendront des microinstructions, et l'algorithme d'interprétation du langage LI deviendra un microprogramme.

L'ensemble de ces primitives définit un langage de microprogrammation dont on peut définir l'algorithme d'interprétation.

La réalisation de cet algorithme d'interprétation pourra être un assemblage de circuits électroniques commandés par un langage de phase (horlogerie).

Si tel n'est pas le cas, il faut ajouter des étapes supplémentaires dans la démarche...

L'intérêt d'une telle approche est d'aborder chaque niveau d'interprétation en connaissant les caractéristiques qu'il doit remplir. Sa spécialisation est donc parfaitement déterminée.

En résumé, la conduite d'une étape de conception entre deux niveaux d'interprétation consiste à choisir une réalisation de l'algorithme d'interprétation d'un langage dont on connait une description formelle. Cette réalisation sera:

- soit la machine électronique (ultime étape),

- soit un ensemble de primitives qui deviendra le langage à interpréter à la prochaine étape.

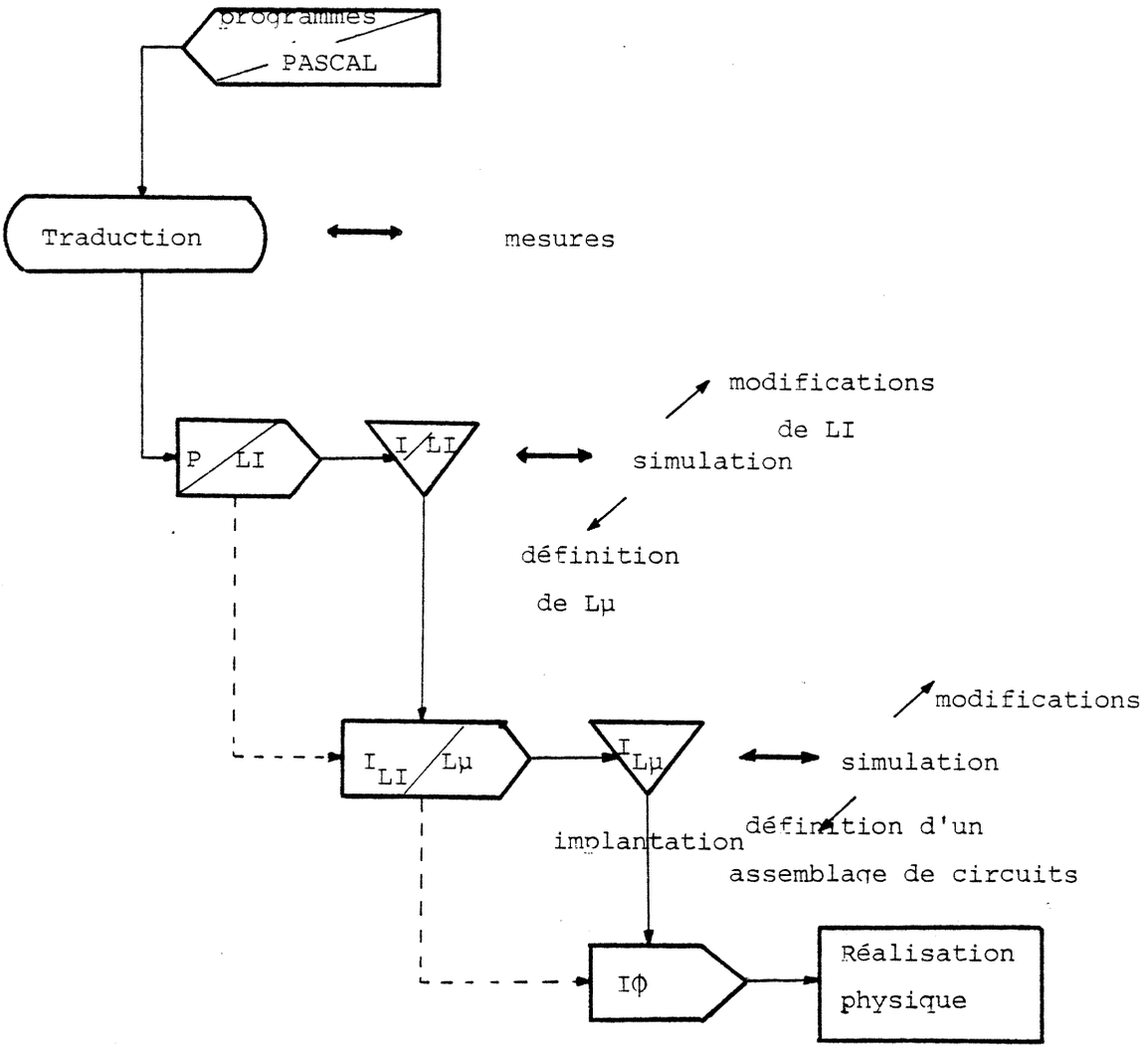


Figure I.2 - Hiérarchie des niveaux d'interprétation et conduite de la démarche descendante.

I.1.2. Niveaux de mémorisation

Comme nous avons défini des niveaux d'interprétation, nous pouvons définir des niveaux de mémorisation suivant des critères de taille, de coût ou de temps d'accès.

Le coût d'un élément de mémorisation croît quand son temps d'accès diminue. Les niveaux de mémorisation constituent eux aussi une hiérarchie:

- mémoire secondaire ou mémoire de masse (disques, bandes magnétiques) caractérisée par un faible coût au bit et un temps d'accès long.

- mémoire centrale (tores ou semiconducteurs) caractérisée par un temps d'accès moyen et un coût au bit moyen.

- mémoire locale et registres généraux caractérisés par un temps d'accès court et un coût au bit élevé.

- mémoires bloc-notes au temps d'accès très court et au coût au bit très élevé.

A chacune des étapes de la démarche, des entités sont définies. Elles sont constituées par les données et les variables manipulées à chaque niveau.

- les fichiers au niveau utilisateurs

- les variables au niveau du langage de haut niveau

- un compteur ordinal et des variables de travail au niveau de la macromachine

- un microcompteur ordinal et d'autres variables de travail au niveau de la micromachine.

Une méthode de conception descendante devrait permettre de s'assurer que l'accès à un être mémorisé est d'autant plus aisé et rapide qu'il est plus fréquemment référencé, et par conséquent d'optimiser les critères de coût/performances.

De façon plus générale, une telle méthode devrait garantir une utilisation rationnelle des ressources de la machine par une organisation adaptée des éléments de mémorisation.

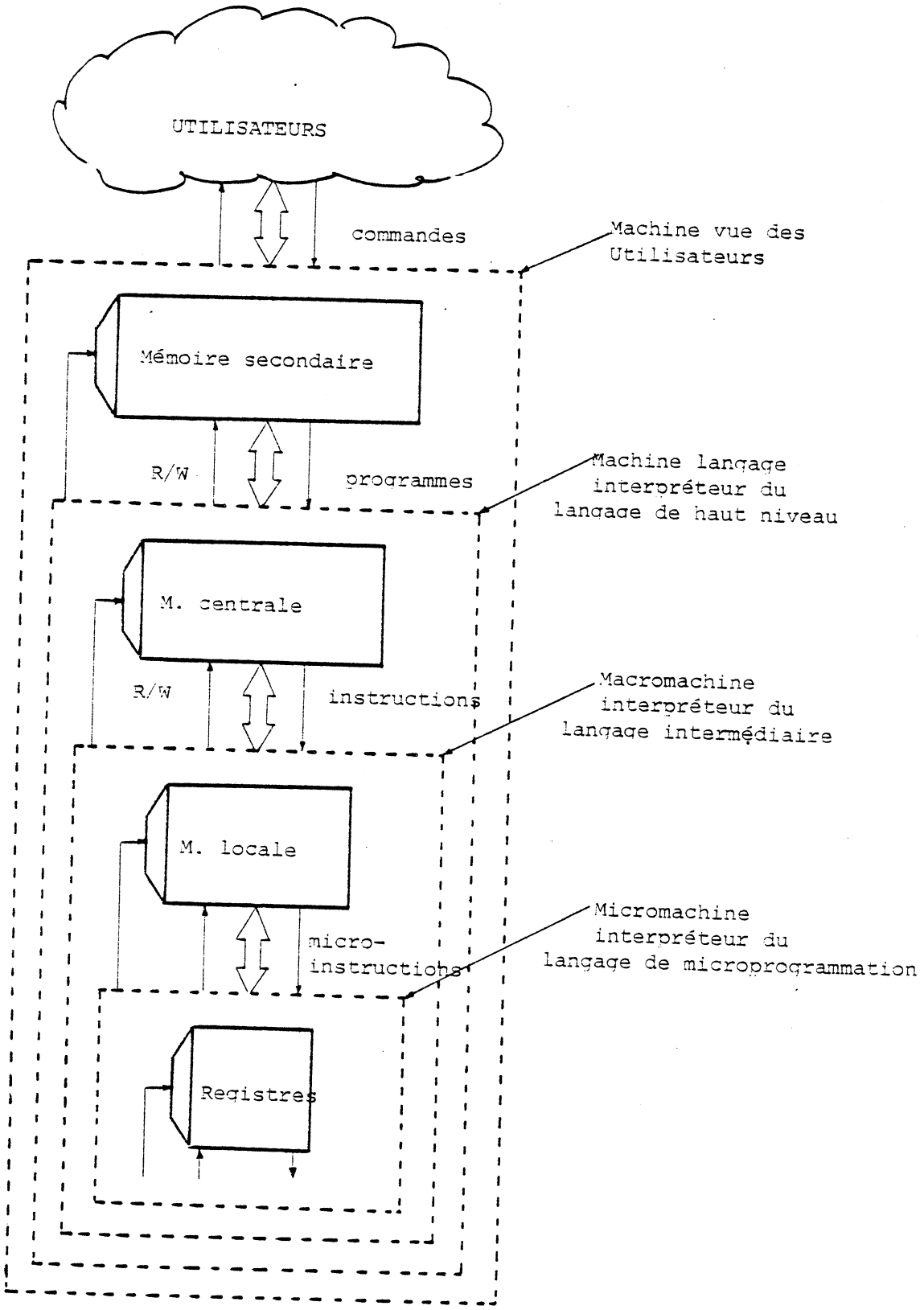


Figure I.3 - Organisation hiérarchisée des niveaux de mémorisation

I.2.LANGAGE INTERMEDIAIRE

Les performances d'un ordinateur sont directement influencées par son code machine, en effet, la rapidité d'exécution des programmes est fortement liée à leur concision et à leur facilité d'écriture.

La définition de ce code machine doit donc être menée avec le plus grand soin.

I.2.1. Définition d'un langage intermédiaire

La première étape d'une conception descendante d'une machine "langage de haut niveau" consiste à définir un langage intermédiaire interprétable par la machine envisagée . Ce langage intermédiaire constitue le langage machine.

R.FORTIER a fait une première ébauche de définition d'un langage intermédiaire pour une machine orientée vers le langage PASCAL en s'appuyant sur la méthode mise au point par GREBERT [4] et dont le principe de base était de faire subir le minimum de transformations au langage source en éliminant la syntaxe et en réordonnant éventuellement (préfixage ou postfixage) les symboles du langage source [5].

Des mesures statiques et dynamiques sur un lot de programmes écrits en PASCAL lui ont permis de tirer les conclusions quant au choix des primitives du langage.

Grossièrement, celui-ci est de type postfixé et est comparable au langage des machines BURROUGHS B6500, interprétant ALGOL 60 [6] [7] et à celui défini par WORTMAN pour la machine PL [8].

Au fur et à mesure de la conception de la machine PASCHLL, ce langage a été optimisé en fonction des performances espérées pour la réalisation envisagée. Cette remarque met en évidence l'influence des moyens sur le projet dans la méthode descendante.

I.2.2. Amélioration d'un code machine

La notion de compaction du code généré est très importante dans la définition d'un langage machine. En effet, un "programme" ne passe qu'une infime partie de sa vie dans la mémoire centrale de l'ordinateur. Le reste de son existence il "réside" en mémoire secondaire sur des supports d'information coûteux (disques ou bandes).

Pour diminuer d'une part les coûts de résidence en mémoire secondaire (prix), et d'autre part, les temps de transfert en mémoire centrale (performances), on a tout intérêt à réduire la taille des programmes. Il est à noter que la compaction de code peut aussi améliorer les temps d'exécution.

La question à se poser est la suivante:

- comment compacter un programme?

Deux approches complémentaires sont possibles :

-la réduction de la taille et du nombre de ses opérandes

Une instruction est généralement constituée par un code opération et un ou des opérandes.

Sachant que les instructions les plus fréquentes référencent une variable (instruction d'accès), on cherchera à définir un espace d'adressage minimal pour ces variables.

Par contre, il semble difficile de compacter des instructions de type "valeur immédiate". Cependant des mesures ont montré que 95% des valeurs immédiates d'un programme pouvaient être codées sur 8 bits et que les constantes "0" et "1" sont très fréquemment utilisées.

On peut trouver une solution à la compaction du code généré en introduisant plusieurs primitives du langage intermédiaire. Par exemple :

instructions	taille
ZERO (préfixe)	1 octet
UN (préfixe)	1 octet
LIT 8 (préfixe, valeur)	2 octets
LIT 16 (préfixe, valeur)	4 octets

Ces instructions sont telles que leur encombrement est inversement proportionnel à leur fréquence d'utilisation.

-l'augmentation de la puissance des instructions

Une deuxième notion importante pour la compaction d'un code machine concerne la puissance des instructions c'est à dire leur aptitude à exprimer des notions sémantiques complexes contenues dans le langage de haut niveau. L'idée est de substituer à la programmation classique (type IBM 360) la microprogrammation d'instructions plus puissantes.

Par exemple, pour l'indexation d'un tableau à deux dimensions, on obtient un facteur de compaction du code généré de 30 pour une machine de type PASCHLL par rapport à une machine classique et universelle du type IBM 360 [8].

L'introduction d'instructions spécialisées puissantes (comme l'opérateur INDEX par exemple) n'introduirait pas une telle diminution de code sans une structure de données adéquates.

Les variables manipulées par la machine PASCHLL possèdent un descripteur de variables pouvant faire référence à un descripteur de type (pour les variables structurées de type ARRAY par exemple) qui contient toutes les informations nécessaires au calcul d'indexation et au contrôle de dépassement de bornes pour un tableau par exemple.

I.3. NOTION DE PROCESSEUR FONCTIONNEL

Le but du projet PASCHLL est la réalisation d'un processeur de traitement spécialisé dans l'exécution de programmes manipulant des données situées dans une mémoire centrale.
Il n'est donc concerné ni par le transfert d'informations entre les périphériques et la mémoire centrale, ni par la gestion globale des programmes soumis par les utilisateurs au système d'exploitation.

Cette approche illustre la notion de processeurs fonctionnels [9] et fait apparaître une répartition des fonctions:

- PASCHLL réalise la fonction d'exécution tandis que

- la machine "hôte" ou Machine Système exécute les fonctions système d'exploitation.

Cette décomposition fonctionnelle peut être généralisée et une approche hiérarchisée de l'architecture d'un ordinateur peut être envisagée.

Par exemple, la machine système peut se concevoir comme étant constituée de plusieurs processeurs spécialisés se répartissant les fonctions système [11], et parmi lesquelles, par exemple, un processeur de gestion des fichiers [10].

Le processeur de traitement peut lui-même être décomposé fonctionnellement.
En effet, que fait une unité centrale pendant l'exécution d'un programme?

- Intuitivement son travail se décompose en quatre étapes:

  - elle acccède à des instructions,

  - elle accède à des données,

  - elle exécute des opérations,

  - elle range des résultats.

Généralement, ces quatre tâches fonctionnelles se déroulent séquentiellement sur le même processeur physique.

On peut imaginer un processeur de traitement constitué de quatre processeurs fonctionnels se répartissant ces quatre tâches et travaillant en parallèle pour accroître les performances globales de la machine.

Cette réflexion sur l'architecture de la machine PASCHLL a été menée en parallèle avec la première étape de sa conception descendante c'est-à-dire la définition de son langage intermédiaire interprétable.

Les résultats de ces études semblent contradictoires:

- D'une part, une architecture fonctionnelle multiprocesseurs devant permettre une désynchronisation entre les quatre fonctions naturelles d'une unité centrale et donc une possibilité de parallélisme (pipe-line).

- D'autre part, un langage intermédiaire de type post-fixé qui est généralement exécuté sur une machine à pile (BURROUGHS B 6500 par exemple) dans laquelle on accède aux instructions, puis on range les opérandes dans une pile d'évaluation, ensuite on exécute les opérateurs qui s'appliquent aux derniers opérandes rangés dans la pile (sommet et sous-sommet pour un opérateur diadique), puis on range les résultats.

Dans une telle machine, aucune désynchronisation n'est possible. Nous avons donc cherché une méthode d'évaluation d'une expression post-fixée (ou sous forme polonaise) autorisant un certain parallélisme dans l'exécution.

Nous avons pensé remplacer la pile d'évaluation par une file (FIFO queue) [cette idée originale est due à F.ANCEAU]:

- le processeur d'ACCES l'alimenterait,

- le processeur OPERATIF viendrait chercher ses opérandes,

- le processeur d'INSTRUCTION lirait le code et aiguillerait les instructions vers les deux autres à travers deux files d'attente.

Remarque:
La fonction de rangement des résultats peut être vue comme un opérateur diadique d'affectation ayant pour premier opérande le résultat évalué et comme second opérande l'adresse à laquelle ce résultat doit être rangé. Cette fonction peut donc être prise en charge par le processeur opératif.

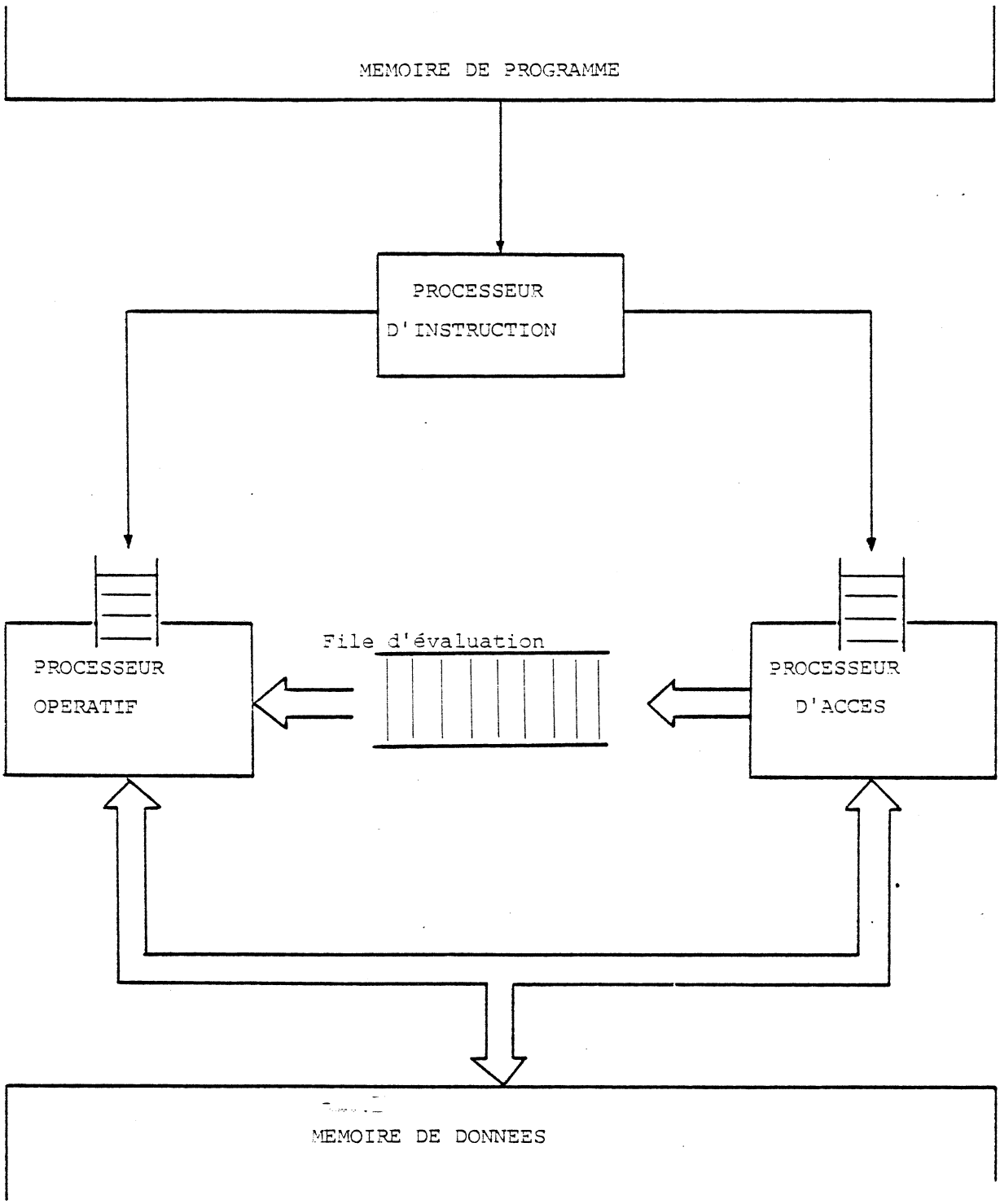


Figure I.4 - Architecture multiprocesseurs fonctionnels permettant la désynchronisation des trois tâches naturelles d'une Unité Centrale

I.4. EVALUATION D'UNE EXPRESSION POST-FIXEE SUR UNE FILE D'ATTENTE. [12], [13], [14], [15]

Si l'emplacement des opérandes ne pose aucun problème dans le cas où l'évaluation se fait sur une pile (machine BURROUGHS B 6500, HEWLETT PACKARD HP 3000 par exemple), il n'en est pas de même si elle se fait sur une file d'attente.
Un mécanisme simple a cependant été trouvé et a fait l'objet d'un brevet ANVAR [14].

L'organisation générale décrite par la figure 4 fait apparaître trois processeurs spécialisés:

- le processeur d'accès aux instructions que nous appellerons PINS,

- le processeur d'accès aux variables que nous appellerons PAC,

- le processeur d'exécution des opérateurs que nous appellerons POP.

PINS accède aux instructions en mémoire centrale et les analyse pour envoyer les instructions d'accès à PAC à travers une file d'attente BIPAC et les opérateurs à travers une file d'attente BIPOP. Ces deux files d'attente ont la particularité d'indiquer l'état VIDE pour le lecteur et l'état PLEIN pour l'écrivain, permettant ainsi d'en assurer la gestion automatique.

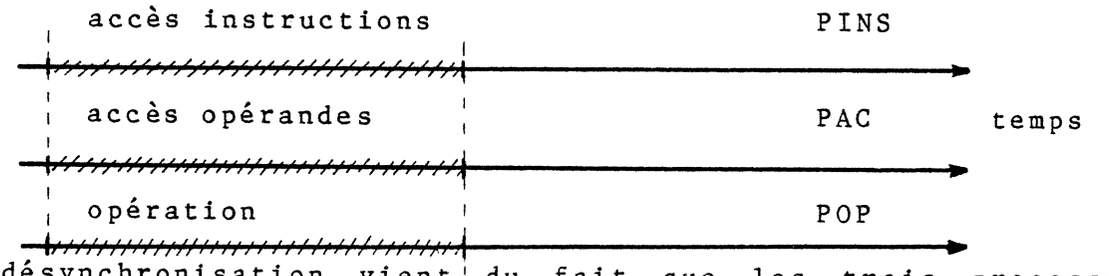
Cette structure globale permet d'illustrer les possibilités de parallélisme d'une telle architecture.
Au même instant :

- un processeur (PINS) exécute une étape de l'accès aux instructions,

- un second (PAC) exécute une étape de l'accès à un opérande,

- un troisième (POP) exécute une étape d'une opération arithmétique ou logique.

Le diagramme séquentiel classique d'un processeur unique est remplacé dans une telle architecture pipe-line par le diagramme parallèle suivant:



La désynchronisation vient du fait que les trois processeurs

opèrent au même instant sur des données indépendantes: l'instruction d'accès lue en mémoire par PINS à l'heure H ne sera exécutée par le processeur PAC qu'à l'heure ($H + d_{PAC}$), et l'opérateur lu en mémoire par PINS à l'heure (H+1) ne sera exécuté par le processeur POP qu'à l'heure ($H+1+d_{POP}$).

Bien entendu, il faut que l'opérande nécessaire à POP pour exécuter son opérateur soit présent dans la file d'attente, c'est-à-dire qu'il ait été accédé par PAC. En d'autres termes, il faut que

$$(d_{POP}+1) \geq d_{PAC}.$$

Nous allons décrire succintement le mécanisme d'évaluation sur FILE et le comparer à l'évaluation sur PILE à l'aide d'un exemple.

Soit à évaluer l'expression suivante:

$$A + \frac{B*C}{D-E}$$

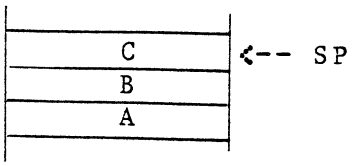
Après post-fixage, on obtient:

ABC * DE - ÷ +

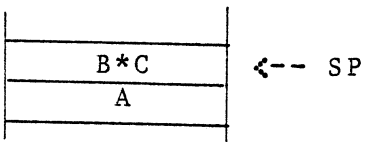
Pour la clarté de la démonstration, nous supposerons qu'au début la PILE et la FILE sont toutes les deux vides et que les temps d'exécution sont les même pour un accès et un opérateur.
Nous allons représenter les états successifs de la PILE et de la FILE.

Dans les schémas suivants SP représente le sommet de la PILE, ECR l'entrée dans la FILE (pointeur d'écriture), P1 et P2 donnent respectivement la position du premier et du second opérande.

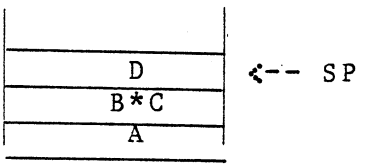
évaluation sur PILE



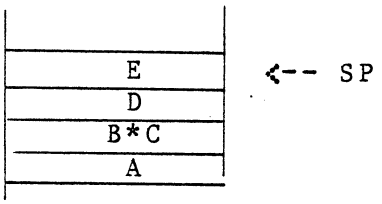
après l'opérateur *



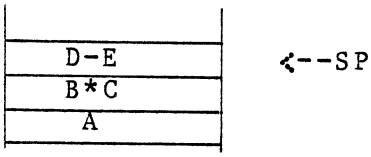
après 1 opération PUSH



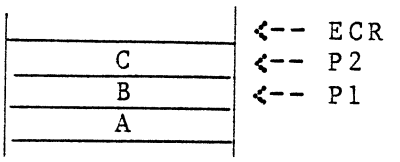
après 1 opération PUSH



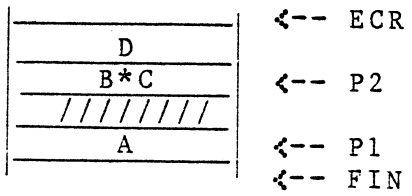
après l'opérateur -



évaluation sur FILE

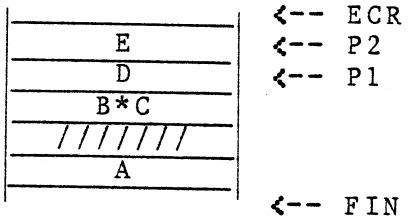


après l'opérateur *

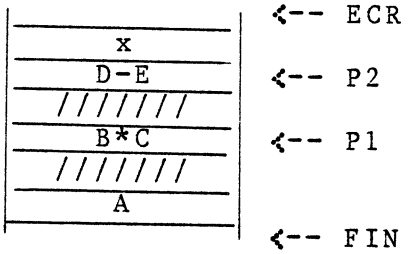


l'accès à D se fait en parallèle avec l'exécution de *

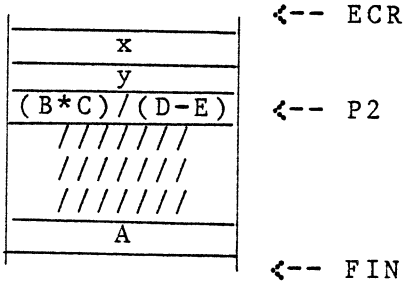
après un nouvel accès



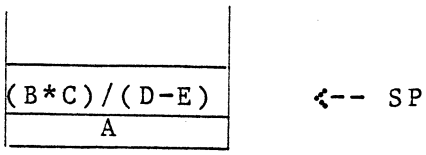
après l'opérateur -



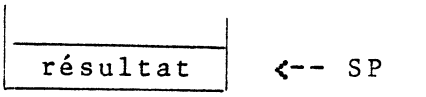
après l'opérateur -



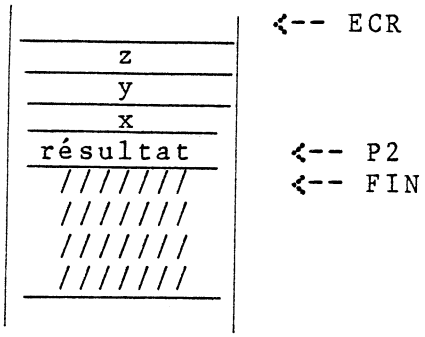
après l'opérateur ÷



après l'opèration +



après l'opérateur +



pendant ce nouveau cycle :
- soit un nouvel accès pour l'expression suivante est réalisé
- soit un opérateur de cette expression est exécuté sur les opérandes z et y.

Cet exemple illustre le gain de temps possible du fait de la désynchronisation entre "accès aux opérandes" et "exécution des opérateurs".

Il appelle quelques remarques:

1 - La position des opérandes est triviale si l'évaluation se fait sur PILE:
ils se trouvent en sommet (pointé par SP) et sous sommet (position SP-1).
Le résultat est rangé en position SP-1 qui devient le nouveau sommet (SP <-SP-1):
la PILE se vide et se remplit par le haut. L'accès à un nouvel opérande se fait par une opération PUSH.

-> écriture en SP+1 et SP <- SP+1

En résumé, un seul pointeur est nécessaire à la gestion de la PILE.

2 - Pour ce qui est de la FILE, nous avons besoin d'au moins trois pointeurs:

- ECR pour l'écriture d'un nouvel opérande,
- P2 pour indiquer la position du deuxième opérande,
- P1 celle du premier opérande.

Si la file d'évaluation n'est pas infinie, mais réalisée par une mémoire à accès alléatoire adressée circulairement, il faut un

quatrième pointeur (FIN).

I.4.1. Mouvement des pointeurs P2 et P1

Le mouvement des pointeurs P2 et P1 n'est pas trivial: il est nécessaire de comptabiliser les accès adjacents dans la chaîne post-fixée pour positionner P2 et P1 sur les bons opérandes au moment de la transition accès-opérateur de la chaîne post-fixée.

Après chaque opérateur diadique P2 ne bouge pas, mais P1 est décrémenté pour pointer sur l'opérande suivant.
Il peut se faire que la case indiquée soit vide (comme après (-) dans l'exemple précédent) et il faut aller chercher la première case non vide en aval.
Pour cela, les "trous" adjacents créés par chaque opérateur diadique sont comptabilisés pour pouvoir être "enjambés" le moment venu.

Notons que quand P1 devient égal à FIN, la file d'évaluation a été vidée par le bas et FIN peut venir se positionner derrière P2:

si P1 = FIN alors FIN <- P2-1

Une analyse de la chaîne post-fixée des instructions est suffisante pour prévoir une image théorique de la file d'évaluation et donc le positionnement de P2 après une transition instruction d'accès-opérateur et le saut de P1 par dessus les "trous" pour aller chercher un opérande en attente.
Seul le processeur d'instruction PINS voit passer toutes les instructions: il lui appartient donc de gérer l'image théorique de l'évolution de la file d'évaluation et d'intercaler des extraordres

AVANCE (d) et RECULE (t)

dans la file d'attente des instructions de POP.

I.4.2. Image théorique de la file d'évaluation

A un instant donné de l'exécution, la file d'évaluation se présente comme une suite d'opérandes adjacents (dk) entrecoupée de cases vides (tk).

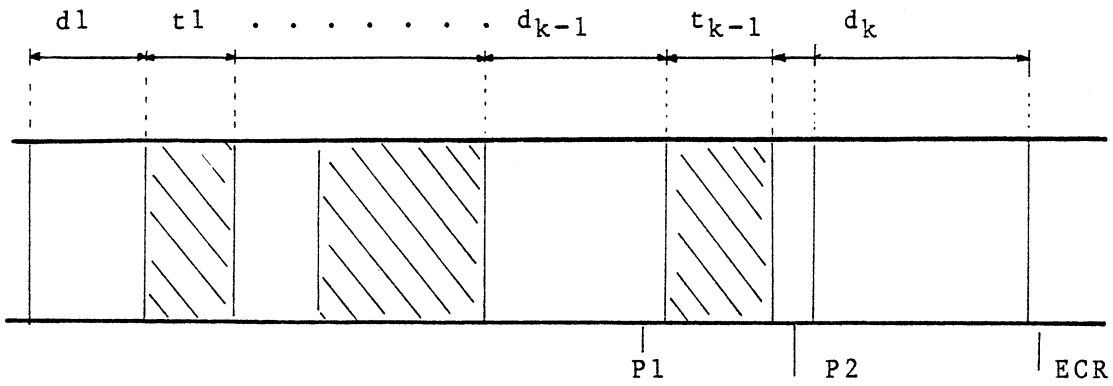


Figure I.5 Image théorique de la file d'évaluation.

Cette suite de couples (d_k, t_k) représente l'image théorique de la file. Elle est construite pendant l'analyse de la chaîne post-fixée et elle anticipe l'état qu'aura la file après l'exécution de l'instruction analysée.
Elle est construite par l'opérateur de contrôle de la façon suivante :

1 - à l'initialisation, l'indice k est nul (aucun groupe)

2 - la première instruction d'accès rencontrée (transition "fin d'expression" -> accès) provoque la création d'un élément de la suite :

$$k=1 \; ; \; d_k=1 \quad \text{et} \quad t_k=0$$

3 - chaque transition accès -> accés entraîne l'incrémentation de d_k:

$$d_k \leftarrow d_k+1 \quad \text{et} \quad t_k=0$$

4 - C'est l'occurence d'une transition accès -> opérateur qui entraîne la génération d'un ordre AVANCE (d_k) .

Un opérateur diadique va créer un trou dans la file à l'emplacement qu'occupait son premier opérande.

Pour chaque opérateur de ce type, l'image théorique va être modifiée par décrémentation de d_k et incrémentation de t_k.
Une transition

(accès -> opérateur) ou (opérateur -> opérateur)

entraîne:

$$d_k \leftarrow d_k-1 \quad \text{et} \quad t_k \leftarrow t_k+1$$

et ce jusqu'à l'annulation de d_k.

($d_k = 0$) indique que tous les opérandes de rang k ont été utilisé et qu'il est nécessaire d'aller chercher le premier opérande en aval en enjambant le trou de rang k-1 (génération d'un ordre RECUL (t_{k-1}). Cette opération consiste à concaténer deux trous, donc à supprimer un élément de la suite (d_k, t_k):

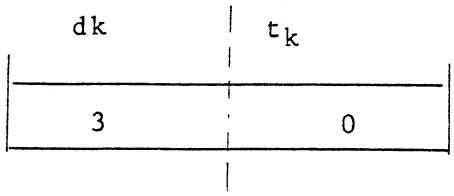
$$t_k \leftarrow t_k + t_{k-1}$$

$$k \leftarrow k-1$$

Si nous reprenons l'expression de l'exemple précédent,

A B C * D E - / +

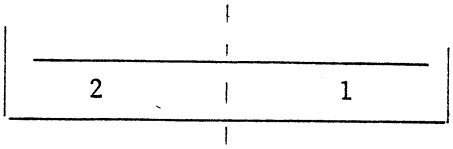
nous pouvons représenter les images théoriques successives de la file d'évaluation sur une pile de profondeur k:



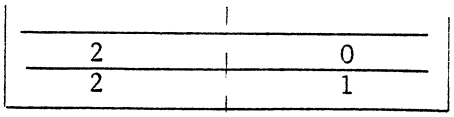
- après C

- transition C -> *
---> génération de AVANCE(3)

- après *

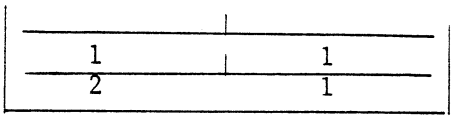


- après E

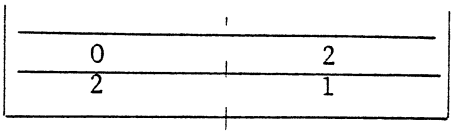


- transition E -> -
---> génération de AVANCE(2)

- après -

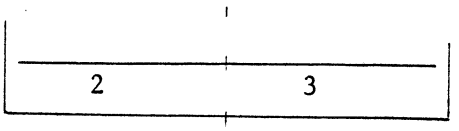


- après /

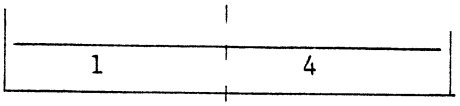


d_k = 0 ---> génération de RECUL(t_{k-1}=1)
compaction de deux trous
suppression d'un élément de la pile

d'où une nouvelle image



- après +



Remarque:
d_k=1 pour k=1 indique la fin de l'expression.
En effet, l'élément restant est le résultat de l'évaluation de l'expression précédente. Il ne peut être considéré comme un opérande. La file peut alors être vidée par le bas:

FIN <- P2-1

I.4.3. Evaluation du taux d'extra-ordres

Soit une expression comptant N opérandes répartis en p groupes. Nous appellerons groupe de rang i, le couple a_i, o_i constitué par a_i instructions d'accès successives suivies de o_i opérateurs diadiques.

Il y a génération d'un extra-ordre AVANCE pour chaque groupe donc au total p extra-ordres AVANCE pour l'expression.

On ne peut évaluer précisément le nombre de RECUL à générer pour une expression quelconque, mais on peut le borner.

Un ordre RECUL est généré pour aller récupérer des opérandes laissés en attente par le groupe précédent.
Pour l'image théorique de la file , l'ordre RECUL a pour effet la concaténation de deux trous adjacents et donc la suppression d'un couple (d_k,t_k)

Pour que le groupe i laisse au moins un opérande en attente, il faut que l'inégalité suivante soit vérifiée:

$$a_i > o_i$$

Sachant que pour une expression donnée, il y a un opérateur diadique de moins que d'opérandes

$$a_i = o_i + 1$$

il y a forcément un groupe parmi les p groupes de l'expression qui ne vérifie pas l'inégalité précédente.

Par conséquent le nombre maximal d'extra-ordres RECUL à générer sera:

$$\max(\text{RECUL}) = p-1$$

D'autre part l'inégalité précédente peut n'être vérifiée par aucun des p groupes de l'expression , auquel cas aucun ordre RECUL ne sera généré:

$$\min(\text{RECUL}) = 0$$

Remarque:
Une expression de N opérandes entraîne N-1 opérateurs diadiques et la répartition en p groupes entraîne p opérateurs pour lesquels un ordre AVANCE doit être généré.

Il reste donc N-1-p opérateurs susceptibles d'entraîner un saut de P1. Donc on peut écrire :

$$p-1 < N-1-p$$

c'est-à- dire

$$N \geq 2p$$

Pour une expression de N opérandes le maximum d'extra-ordres ne pourra être atteint que si la relation

$$N = 2p$$

est vérifiée.
Considérons la fonction T = "nombre d'extra-ordres" sur "nombre d'instructions" :

$$T = \frac{Nb(avance)+Nb(recul)}{a_i + o_i}$$

$$\frac{p+0}{N+N-1} < T < \frac{p+p-1}{N+N-1}$$

$$\frac{p}{2N-1} < T < \frac{2p-1}{2N-1}$$

Le taux maximum d'extra-ordres peut être atteint seulement si N=2p et sa valeur sera :

$$\frac{N-1}{2N-1}$$

donc inférieure à 50%.

Remarque:
Cette évaluation ne tient pas compte des opérateurs monadiques dont la présence peut éventuellement impliquer des ordres AVANCE, mais aucun ordre RECUL.
La fonction T sera donc généralement inférieure à l'estimation précédente.

Exemple
Soit l'expression :

$$A - \frac{(B+C)(D+E)(F+G)(H+I)}{J}$$

Mise sous forme polonaise, elle devient:

ABC + DE + FG + HI + * * * J / -

Elle comporte 10 opérandes répartis en 5 groupes, 3 groupes vérifiant l'inégalité

$$a_i \geq o_i.$$

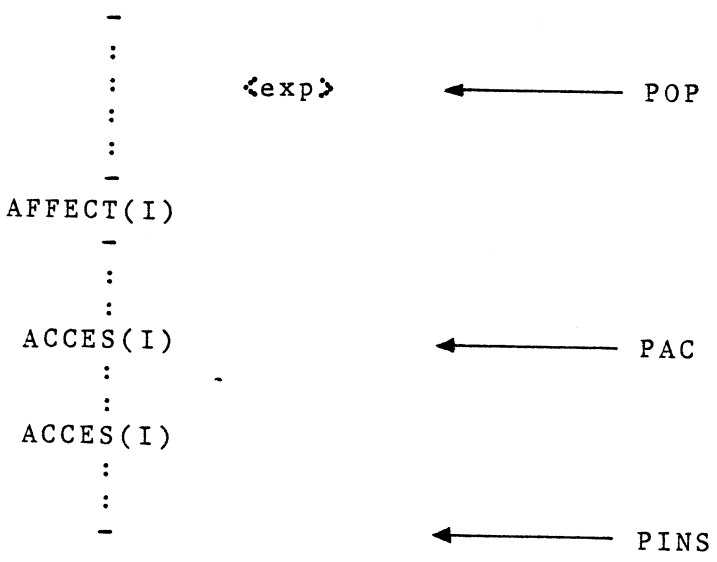
I.5. PROBLEMES DUS AUX ARCHITECTURES "PIPE-LINE"

La décomposition d'une unité centrale d'ordinateur en processeurs fonctionnels travaillant en parallèle semble très intéressante à première vue, mais elle entraîne des problèmes dont la résolution peut fortement diminuer les performances globales espérées pour la machine. Ces problèmes sont de deux ordres: les dépendances et l'anticipation.

I.5.1. Le problème des dépendances

I.5.1.1. Position du probléme:

Prenons par exemple un fragment de la suite d'instructions à exécuter par la machine schématisée de la façon suivante:



Supposons qu'à un instant donné, les positions des processeurs fonctionnels soient comme indiqués sur le schéma.

On constate que PAC doit ranger dans la file d'évaluation le descripteur de la variable I.
Or, POP est en train d'évaluer une expression <exp> dont le résultat doit être affecté à la variable I.
Si PAC va chercher le descripteur de I dans la pile de contexte en mémoire centrale, il rangera dans la file d'évaluation une valeur erronée pour I.
On dira que I est sous dépendance tant que l'instruction AFFECT(I) n'aura pas été exécutée par le processeur opératif.

Une solution triviale consisterait à bloquer PAC jusqu'à ce que la dépendance soit résolue.
Elle enlèverait beaucoup d'intérêt à l'architecture pipe-line où

les stations passeraient leur temps à s'attendre mutuellement. Cette situation est trop fréquente pour être négligée: en effet si l'on modifie une variable dans un programme, c'est en général pour l'utiliser peu de temps après.

I.5.1.2. Solution au problème des dépendances:

Pour ne pas bloquer le pipe-line, PAC va construire un descripteur de dépendances dans la file d'évaluation avant de passer à l'accés suivant. Quand POP exécutera l'affectation, il modifiera ce descripteur de dépendance dans FILEVAL avec les nouvelles valeurs de I, c'est-à-dire qu'il rangera le résultat là où il sera utile et il lèvera la dépendance.

Si une nouvelle occurence de l' accès à la même variable I sous dépendance a lieu, un descripteur de dépendance chaîné avec le premier sera construit pour que POP puisse remonter cette chaîne de reprise le moment venu.

Pour résoudre ce problème, il est nécessaire d'introduire une nouvelle file que nous appellerons "file de dépendances ou FILEDEP".
Cette file a la même profondeur que la file d'évaluation (par exemple 16 mots) et elle présente la particularité d'avoir une partie adressable par son contenu dans laquelle sont consignés les "noms internes" des variables sous dépendance.

- initialisation d'une dépendance

L'algorithme d'interprétation de PINS est modifié de manière à ce que les instructions de type AFFECT(I) soient envoyées dans BIPOP et BIPAC.
Pour PAC AFFECT(I) veut dire:
"initialisation d'une dépendance sur la variable I".
Son travail consiste à ranger dans la partie associative de FILEDEP le nom interne de I:

$$FILEDEP(NIN).ASS \leftarrow S_i, D_i$$

où NIN est le pointeur d'écriture dans FILEDEP.

détection d'une dépendance

Pour toute occurrence d'une instruction d'accès de type ACCES(I), PAC consulte la table des dépendances par une recherche associative sur le nom de la variable I :

Est-ce que ce nom existe?

Deux cas peuvent se présenter:

1 - il n'a pas été trouvé :
PAC va alors chercher en mémoire le descripteur de I qu'il range dans FILEVAL(ECR).

2 - il est en FILEDEP.ASS

Cela signifie que la variable est en cours de modification, il n'est donc pas nécessaire d'accéder à la mèmoire (d'où un gain de temps potentiel); mais il faut construire un descripteur de dépendances dans FILEVAL(ECR) et indiquer la position de ce descripteur en inscrivant son adresse dans le champ (.PEVAL) de FILEDEP à l'adresse associée au nom de la variale I.

FILEVAL(ECR) <- descripteur de dépendance

FILEDEP(adresse associée).PEVAL <- ECR

ECR <- ECR+1.

Remarque

Si ce n'est pas la première occurrence d'un accés à une variable sous dépendance, il faut créer un nouveau maillon de la chaîne de reprise.
Pour ce faire, le descripteur de dépendance est construit de la manière suivante:

- le champ préfixe (PF) indique un descripteur de dépendances:

FILEVAL(ECR).PF = DEP

- le champ reprise (REP) donne l'adresse dans FILEVAL du précédent descripteur de dépendances sur la variable I:

FILEVAL(ECR).PF = FILEDEP(adresse associée).PEVAL

- enfin l'adresse du premier maillon est modifiée:

FILEDEP(adresse associée).PEVAL <- ECR.

Cet algorithme peut être toujours appliqué. Il suffit qu'à l'initialisation de la dépendance, le champ PEVAL soit marqué à "NIL".

- résolution d'une dépendance

Quand il exécute l'affectation, POP range

- d'une part le nouveau descripteur de I en mémoire et,

- d'autre part, le range dans toutes les cases de FILEVAL dans lesquelles un descripteur de dépendances sur la variable I a été construit en remontant la chaîne de reprise à partir du pointeur PEVAL de FILEDEP(NOUT) (NOUT est le pointeur de lecture dans FILEDEP).

Il ne lui reste plus qu'à marquer cette dépendance comme étant résolue.

Remarque

Il parait astucieux de conserver dans FILEDEP les descripteurs de dernières variables modifiés de façon à réaliser un "cache" su celles-ci.

Ainsi, après résolution de la dépendance sur la variable I, ell devient directement accessible dans la file des dépendances.

Des accès mémoire sont donc supprimés, d'où une augmentatio sensible des performances globales de la machine. La variable est disponible dans FILEDEP jusqu'à son écrasement dû à l limitation de la taille de la file et de son fonctionnement en mod premier rentrè, premier sorti (FI-FO).

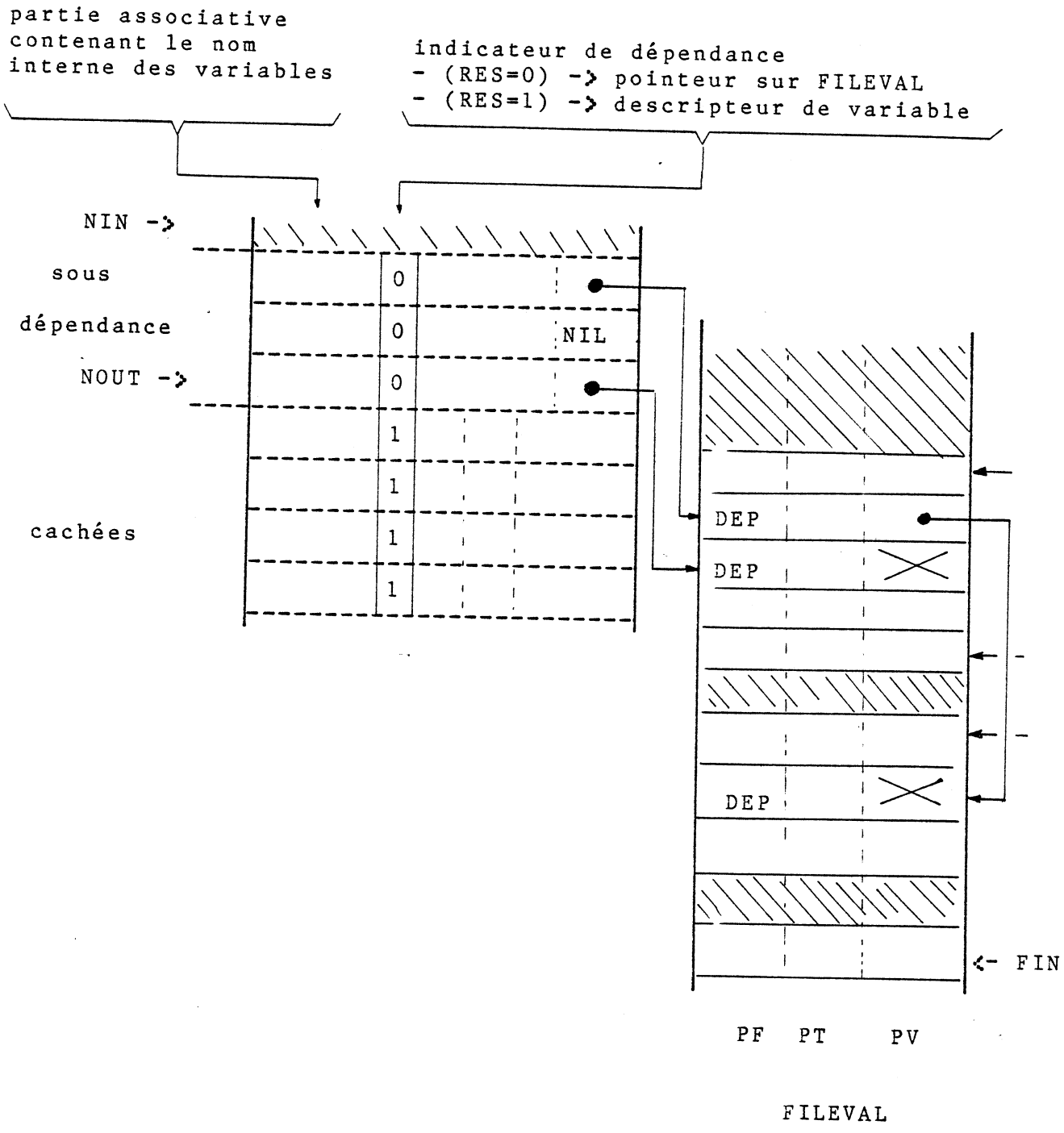


Figure I.6 - Exemple d'état de la file d'évaluation et de la file des dépendances.

I.5.1.3. Introduction du processeur FILE

La gestion de FILEVAL demande 4 pointeurs, celle de FILEDEP deux. Ces deux files sont utilisées alternativement par PAC et par POP qui tous deux doivent connaître la position des pointeurs.
Pour ne pas surcharger les algorithmes de ces deux processeurs fonctionnels avec des dialogues au sujet des pointeurs, un processeur de service a été introduit. Il s'agit du processeur FILE, ressource partagée du processeur PAC et du processeur POP.

Son rôle est de gérer l'adressage de FILEVAL et de FILEDEP et de contrôler le flot des descripteurs sur un bus interne à la machine, le bus FILE.

Les processeurs fonctionnels posent des requêtes au processeur de service. Celles-ci seront satisfaites suivant un protocole donnant par exemple la priorité à la station vidant le pipe-line.

Elles sont du genre:

- rangement d'un descripteur pour PAC,

- demande d'un opérande ou rangement d'un résultat pour POP,

ou encore:

- initialisation d'une dépendance,

- résolution d'une dépendance.

I.5.2. Le problème de l'anticipation

Les performances d'une machine à structure pipe-line dépendent de sa faculté d'anticiper le sèquencement du programme :
de nombreuses machines de ce type voient leurs performances théoriques cruellement diminuées par les mauvais choix vidant le pipe-line.

Dans une machine langage, la structure des programmes à traiter est connue et permet de choisir l'alternative la plus probable. Par exemple, pour une instruction FOR on choisit de rentrer dans la boucle, ainsi si le programme tourne N fois dans la boucle, le taux d'erreur sera de 1/N et la reprise quasi immédiate.

Lorsque le processeur PINS accède à une instruction de branchement conditionnel, il ne connait pas la valeur booléenne qui lui permettrait de choisir à coup sûr la bonne alternative.
En effet, ce prédicat sera évalué par le processeur opératif qui est en retard par rapport à PINS.

Ce dernier ne peut pas attendre, il choisit donc l'alternative la plus probable et se place dans un état dit conditionnel, dans

dans lequel il s'interdit d'exécuter des opérations irréversibles ou d'envoyer à PAC des instructions dont l'exécution serait irréversible: il se limite en fait à un travail de préparation.

Quand POP connait la valeur du prédicat, il est en mesure de juger si le choix fait par PINS était le bon:

- il était bon : POP lève l'état conditionnel de PINS et l'autorise à continuer en le faisant sortir d'une phase d'inaction éventuelle.

- il était mauvais : POP se rend compte que PINS a fait un travail de préparation inutile et que PAC a exécuté des accès inutiles. Il émet donc une interruption qui doit avoir pour effet:

  - de vider les files d'attente BIPAC et BIPOP qui ne contiennent que des instructions relatives à la mauvaise alternative,

  - d'annuler le travail de PAC en supprimant les opérandes rangés dans la file d'évaluation en modifiant simplement le pointeur d'écriture dans FILEVAL:

    ECR <- P2+1

    En effet, P2 pointe sur la valeur du prédicat, donc tout ce qui est en amont doit être supprimé.

  - de reprendre l'exécution au début de la bonne alternative: pour ce faire, PINS, lorsqu'il fait un choix, calcule toujours l'adresse de l'autre alternative et la range au sommet de sa pile de contrôle.

Une telle organisation n'est performante que s'il est possible de connaître l'alternative la plus probable. Dans PASCHLL, cette notion de probabilité est présente dans la diversité des instructions de branchement..

Exemple de structure de contrôle :

- boucle WHILE

  syntaxe PASCAL = WHILE <exp> DO <inst>

  syntaxe PASCHLL = LOOP(m) <exp> WHILE <inst> ENDLOOP

- boucle REPEAT

  syntaxe PASCAL = REPEAT <inst> UNTIL <exp>

  syntaxe PASCHLL = LOOP(m) <inst><exp> UNTIL

- boucle EXITIF

syntaxe PASCAL = LOOP <inst1> EXITIF <exp><inst2> END

syntaxe PASCHLL = LOOP(m) <inst1><exp> EXITIF <inst2> ENDLOOP

- boucle FOR

syntaxe PASCAL = FOR := <exp1> TO <exp2> DO <inst>

syntaxe PASCHLL = <exp1><exp2> FOR(m) AFFECT(v) <inst> ROF

Les différentes syntaxes des boucles de PASCHLL font apparaître une mise en préfixé des instructions :

LOOP(m) et FOR(m)

qui définissent le début de la boucle et donnent en paramètre l'adresse de fin :

(CO+m).

PINS choisit de rentrer dans la boucle et s'il a fait un mauvais choix, il dispose de l'adresse de sortie.

Les opérateurs de contrôle WHILE (sort si FAUX), UNTIL et EXITIF (sort si VRAI) sont post-fixés derrière le prédicat.

PINS choisit l'enchaînement le plus probable, il passe en séquence pour WHILE et EXITIF et il remonte au début de la boucle pour UNTIL et ROF.

L'instruction ENDLOOP quant à elle, est inconditionnelle: PINS remonte au début de la boucle.

- aiguillage à deux directions :

a/ syntaxe PASCAL = IF<exp> THEN <inst>

syntaxe PASCHLL = <exp> IF(m) <inst> FI

b/ syntaxe PASCAL = IF<exp> THEN <inst1> ELSE<inst2>

syntaxe PASCHLL = <exp> IF(m) <inst1> NETH(n) ELSE <inst2> FI

NETH(n) permet de sortir du segment de contrôle <inst1> et de sauter par dessus la deuxième alternative.

IF(m) donne en paramètre l'adresse de la deuxième alternative.

- aiguillage multidirectionnel :

Il permet de faire un choix parmi un nombre quelconque d'alternatives en fonction de la valeur scalaire d'une variable:

syntaxe PASCAL =

```
CASE <exp> OF
<Io> : <insto>
     :
     :
     :
<Ip> : <instp>
 END
```

syntaxe PASCHLL =

```
<exp> CASE(m)
<Io> OF (n1) <insto> FO
   :
   :
   :
<Ip> OF (np+1) <instp> FO
ESAC
```

Le paramètre m de CASE(m) définit l'adresse de l'instruction qui suit ESAC: elle est utilisée par FO pour sortir du segment de contrôle.

La probabilité pour que <exp> soit égale à l'une des valeurs spécifiées dans la liste de valeurs <Ii> dépend du nombre total de valeurs possibles et du nombre d'étiquettes de <Ii>. Elle dépend du programmeur qui devra placer les cas les plus fréquents en tête de liste pour que son aiguillage soit réalisé dans les meilleurs délais.
En effet, dans PASCHLL nous comparons séquentiellement la valeur de <exp> avec celle de <Io>, puis <I1>... jusqu'à trouver l'égalité.

Cette recherche séquentielle est optimisée par le fait que les trois processeurs y participent:

- PAC range la valeur des listes <Ii> dans FILEVAL,

- POP effectue les comparaisons,

- PINS anticipe en supposant que le cas précédent ne soit pas le bon et prépare l'accès aux valeurs du cas suivant.

Le paramètre de OF(n_i) donne l'adresse de la liste d'étiquette <Ii> permettant de sauter par-dessus le segment <$inst_{i-1}$>.

I.6. Architecture globale de PASCHLL

Les trois processeurs fonctionnels PINS, PAC et POP ont accès à la mémoire centrale. Pour régler les conflits d'accès et faire la liaison avec la machine système, un processeur de service PME a été introduit.

On obtient le schéma complet de l'architecture de PASCHLL:

Une description fonctionnelle de ces cinq processeurs est donnée dans le chapître suivant. Leurs réalisations seront ensuite détaillées.

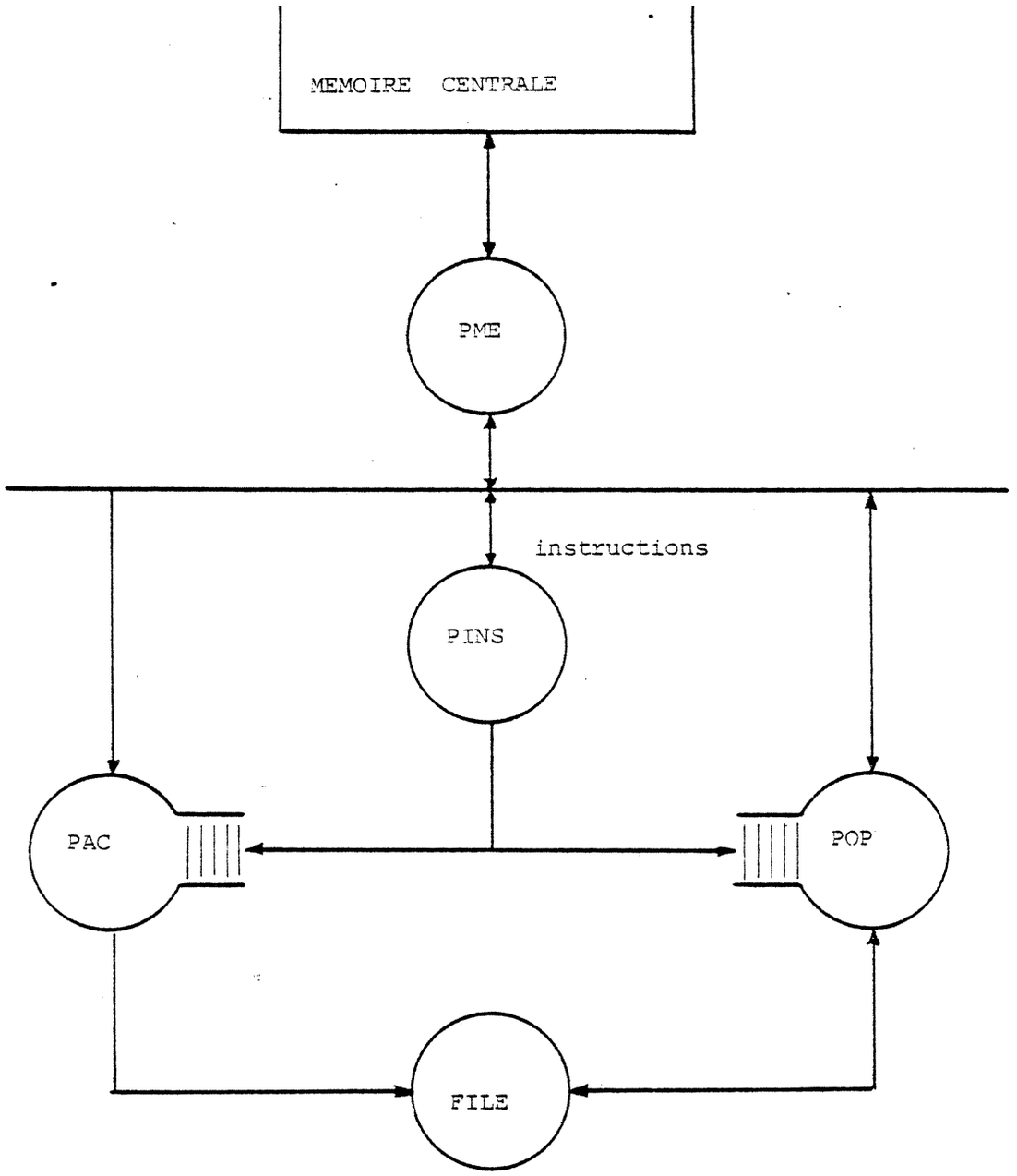


Figure I.6 - Architecture globale de PASC-HLL

Chapître II -

DESCRIPTION FONCTIONNELLE DES CINQ PROCESSEURS

II.1. LE PROCESSEUR D'ACCES AUX INSTRUCTIONS PINS

II.1.1. Rôle de PINS dans PASCHLL

C'est la station principale du pipe-line qui constitue la machine PASCHLL.
C'est le processeur maître en ce sens qu'il analyse le programme à exécuter se présentant à lui sous la forme d'une chaîne post-fixée d'instructions (notation polonaise) pour opérer un tri de façon à :

- distribuer les instructions d'accès aux opérandes et de modification de la pile de contexte en mémoire centrale au processeur d'accès PAC.

- distribuer les opérateurs au processeur opératif POP. Pour PASCHLL, l'ensemble des opérateurs regroupe non seulement tous les opérateurs arithmétiques et logiques monadiques ou diadiques du langage PASCAL, mais aussi les opérateurs d'accès comme :

  - l'indexation d'un tableau (opérateur INDEX) ou

  - l'accès à un élément d'une structure record de PASCAL (opérateur CHAMP) et

  - les opérateurs d'affectation (AFFECT ou PARAM) etc...

- générer des "extra-ordres" de mouvement de pointeurs pour le processeur FILE quand c'est nécessaire.

- exécuter les instructions de contrôle en anticipant éventuellement sur le résultat d'un prédicat pour choisir une alternative pour la suite du programme (par exemple PINS choisit systématiquement l'entrée dans une boucle FOR).

Pour schématiser, on peut dire que PINS a un rôle de démultiplexage du flot d'instructions constituant le programme à exécuter par PASCHLL.

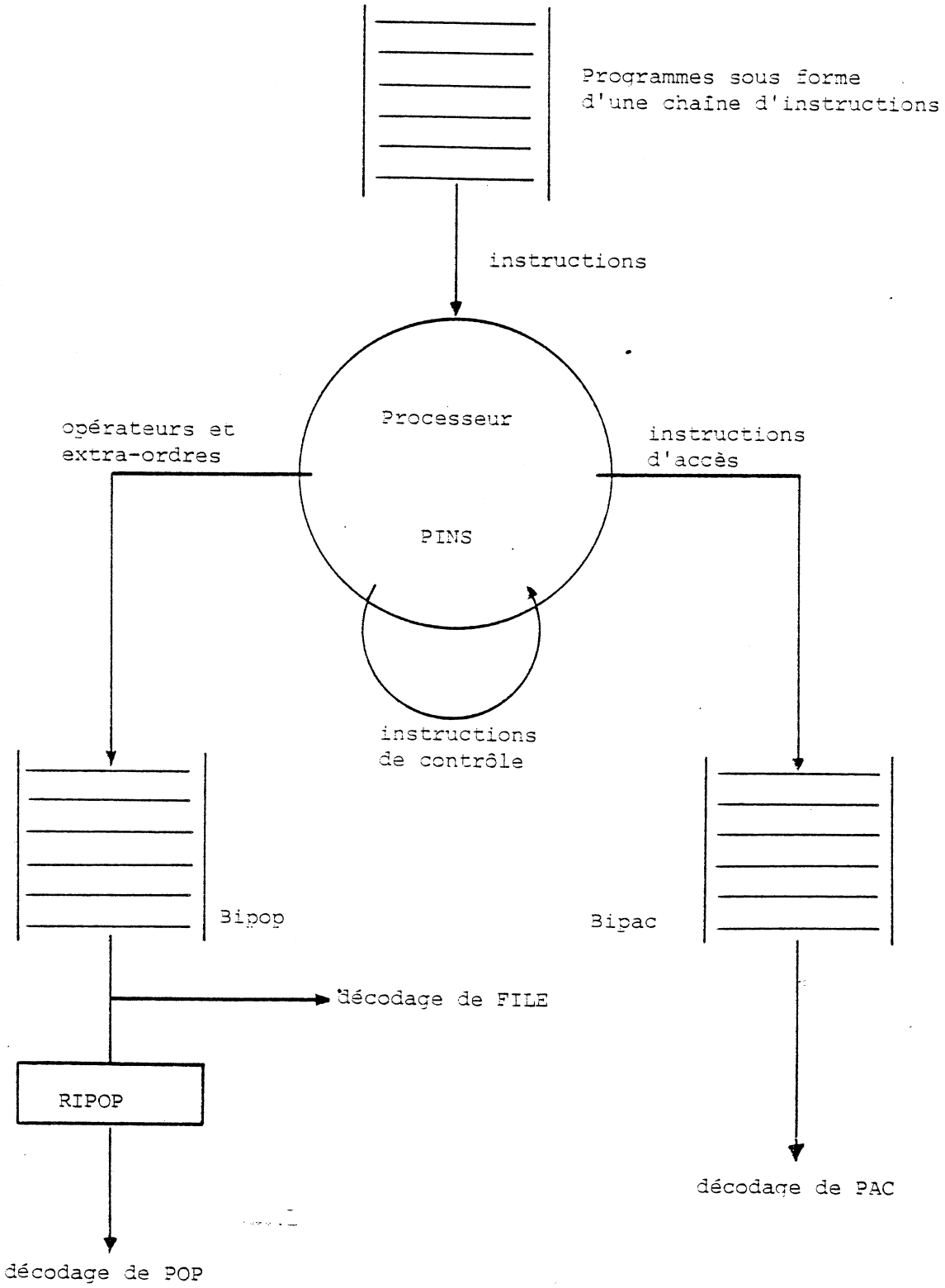


Figure II.1 - PINS: démultiplexage du flot des instructions

L'algorithme d'interprétation de PINS n'est pas classique et comporte plusieurs étapes :

1 - l'accès à une nouvelle instruction.

2 - le traitement des transitions entre quatre grands types d'instructions qui sont :

- les instructions d'accès,
- les opérateurs,
- les instructions de fin d'expression,
- les instructions de contrôle.

Cette étape permet la gestion de l'image théorique de la file d'évaluation donc la génération éventuelle d'extra-ordres de mouvement de pointeurs sur celle-ci.

3 - le traitement des caractéristiques des instructions.

Au niveau de PINS une instruction se caractérise par:

- son type parmi les 4 précités,
- sa destination (PAC ou POP ou les deux),
- l'entrée dans l'état conditionnel,
- la sortie de l'état conditionnel,
- l'attente d'un événement extérieur à PINS permettant la levée de cet état conditionnel.

Cette étape permet la gestion du bus instruction interne à la machine.

4 - L'exécution de l'algorithme associé à l'instruction de contrôle.

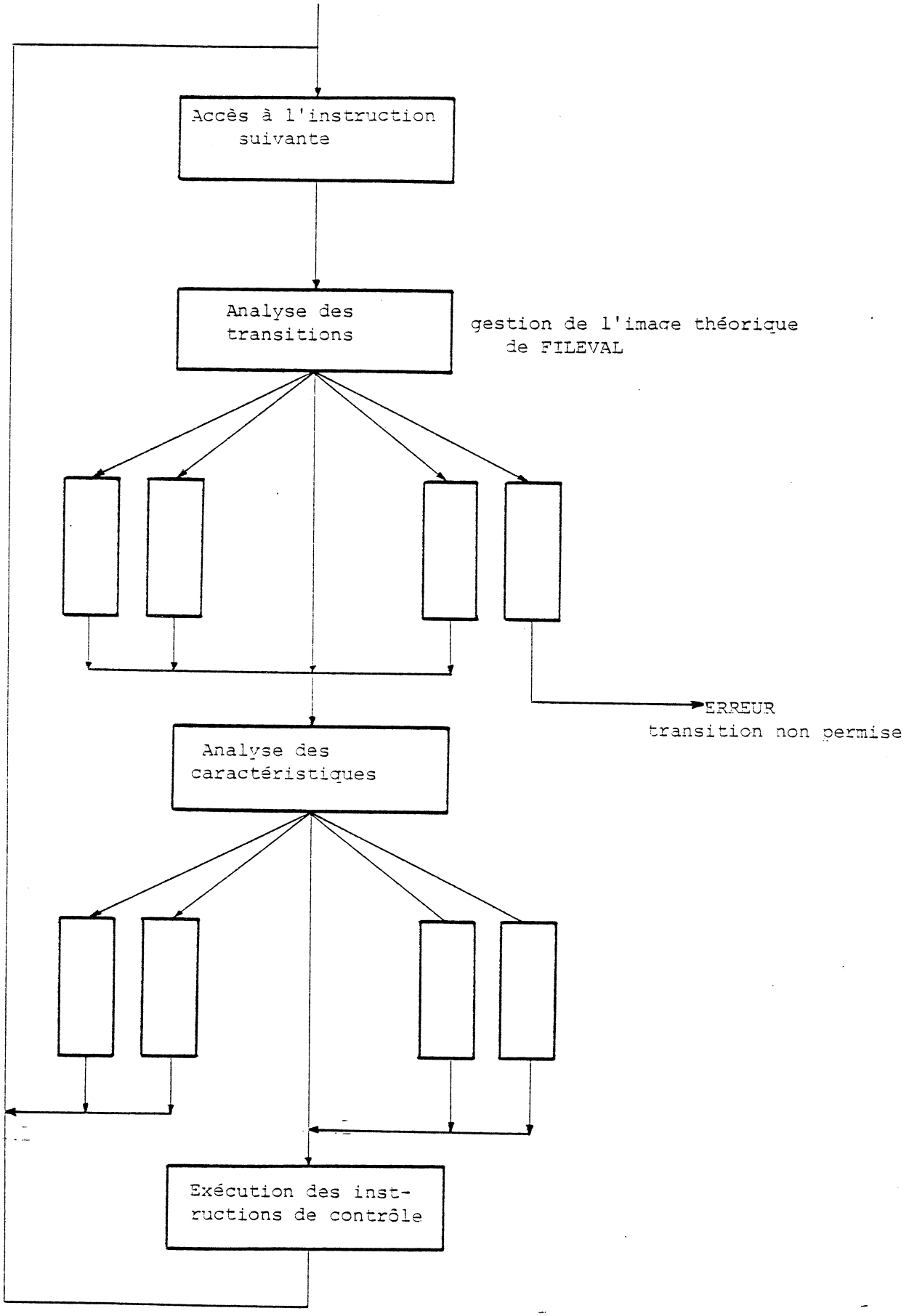


Figure II. 2 - Algorithme d'interpr2tation de PINS

L'architecture de PINS a été définie à partir d'une réflexion sur cet algorithme selon la méthodologie de conception descendante dont ce projet se voulait une illustration. Ce processeur est constitué par l'interconnexion d'un certain nombre de modules ayant chacun une tâche bien définie.
Ce sont:

- l'automate d'accès aux instructions en mémoire centrale. Il se présente comme un interlocuteur de PME (PINSI) indépendamment de PINS,

- le module de gestion de l'image théorique de FILEVAL,

- le module de gestion du bus instructions,

- le module d'exécution des instructions de contrôle.

II.1.2. Le codage des instructions

Du fait de la structure "pipe-line" de PASCHLL, la même instruction peut être décodée par plusieurs processeurs qui auront tous des algorithmes d'interprétation différents.

Considérons par exemple l'opérateur ADD.

Il sera successivement décodé par PINS, FILE et POP :

- pour PINS, il s'agit d'un opérateur diadique entrainant une modification de l'image théorique de la file d'évaluation, et suivant l'état de cette image, la génération éventuelle d'un extra-ordre AVANCE, INIT ou RECULE, et enfin le chargement de BIPOP.

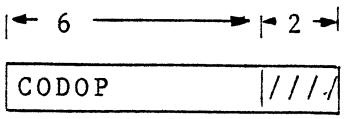
-pour FILE cela signifie le transfert de l'opérande pointé par P1 dans le registre de communication avec POP.

-pour POP il s'agit d'additionner deux opérandes.

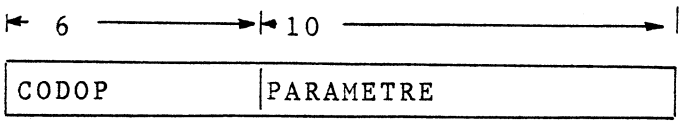
Le codage des instructions essaie de tenir compte de cette complexité due à l'architecture de la machine.

La méthode descendante nous a permis de définir le format des instructions suivant :

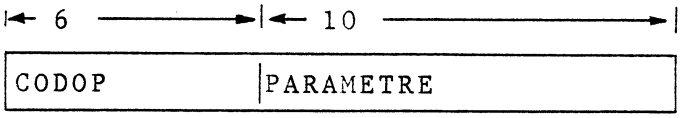
- instructions de contrôle sans paramètre



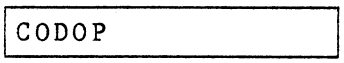
- instructions de contrôle avec paramètres



- instructions d'accès



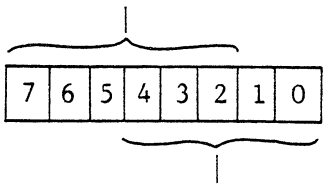
- opérateur



Il n'est pas nécessaire que PINS fasse un décodage fin de toutes les instructions. Pour lui, la reconnaissance de 64 classes d'instructions est suffisante. Il ne décode que les 6 bits poids fort de l'instruction (6 = taille du code opération des instructions de contrôle).

POP a besoin de reconaître 32 opérateurs monadiques et 32 opérateurs diadiques. Cette distinction 1 parmi 32 se fera sur les cinq bits poids faible.

choix de 1 classe parmi 64 (PINS)



choix de 1 opérateur parmi 32

Il y a donc recouvrement des bits 2, 3 et 4 du code opération ce qui implique qu'opérateurs monadiques et opérateurs diadiques occupent chacun 8 classes d'instructions pour PINS avec strictement le même traitement.
Ceci peut sembler être du gâchis mais (64 - 2*8) = 48 classes sont suffisantes pour les instructions autres que les opérateurs.

II.1.2.1. Instructions de contrôle

Structure IF THEN ELSE

a - if <exp> then <stat>

<exp> THEN (↑) <stat> EXIT

b - if <exp> then <stat1> else <stat2>

<exp> THEN(↑) <stat1> NETH(↑) ELSE<stat2>EXIT

THEN(m)	000001	m
NETH(m)	110101	m
ELSE	11011000	
EXIT	10101100	

Structure FOR

a - for I:= <exp1> to <exp2> do <stat>

<exp1><exp2> FORUP(m) ASS(I) <stat> ROFUP

b - for I:= <exp1> downto <exp2> do <stat>

<exp1><exp2> FORDOWN(m) ASS(I) <stat> ROFDOWN

FORUP(m)	010100	m
FORDOWN(m)	010101	m
ROFUP	00010100	
ROFDOWN	00010110	
ASS(s,d)	100000	s,d

Structures WHILE, REPEAT, LOOP

a - while <exp> do <stat>

LOOP(m) <exp> WHILE <stat> ENDLOOP

b - repeat <stat> until <exp>

LOOP(m) <stat> <exp> UNTIL

c - loop <stat1> exitif <exp><stat2> end

LOOP(m) <stat1><exp> EXITIF <stat2> ENDLOOP

LOOP(m)	111110 | m
ENDLOOP	11001000
EXITIF	00001000
WHILE	00001010
UNTIL	00001100

Structure CASE

case<exp> of $v^1,\ldots,v^{n1}$: <stat1> ;

$v^1,\ldots,v^{n2}$: <stat2> ;

:

:

:

$v^1,\ldots,v^{np}$: <statp> end ;

<exp> CASE(m) $I8(v^1)\ldots I8(v^{n1})$ OF(n1) <stat1> FO

$I8(v^1)\ldots I8(v^{n2})$ OF(n2) <stat2> FO

:

:

:

$I8(v^2)\ldots I8(v^{np})$ OF(np) <statp> FO

ESAC

CASE(m)	000100 | m
OF(m)	101000 | m
FO	01000100
ESAC	01001000

Branchements GOTO

L'instruction GOTO <label> pouvant conduire à sortir d'un ou de plusieurs segments de contrôle, il faut utiliser deux instructions:

EXITFOR(NFOR,NSEG) et GOTO(DEPLACEMENT)

EXITFOR(n,m)	10010100	n	m
GOTO(m)	111101	m	

Remarque:
- NFOR indique le nombre de segments FOR que l'on quitte,

- NSEG indique le nombre total de segments que l'on quitte.

Appel de procédure ou de fonction

a - proc ;

CALL(PROC) ; ENTER ;

b - proc(<expl>,...,<expn>) ;

CALL(PROC) ; <expl> ; PARAM (SP,-3) ;

:
:

<expn> ; PARAM(SP,-(2+n)) ; ENTER

Remarque:

Lorsque <exp> est réduit à une variable et que le paramètre a été déclaré var (par référence), il faut générer l'instruction INDD au lieu de REF.

CALL(s,d)	101010	(s,d)	
CALLF(s,d)	101100	(s,d)	
ENTER	10011000		
PARAM(s,d)	100001	1111	d

Retour de procédure ou de fonction

RETURN	10011100
RETURNF	10011110

Structure WITH

a - with <record-var> do <inst>

b - with <R1>, <R2>,...,<Rn> do <inst>

A l'intérieur de chaque procédure, les <record-variables> mis en facteur peuvent être adressés par rapport à un registre W.
Le premier a pour déplacement ZERO.
Après avoir rangé ces variables par une instruction AFFECT(w,i), on génère l'instruction WITH(+i) qui permet à la machine d'incrémenter le pointeur de pile

(SP:=SP+i).

Lorsqu'on quitte un segment WITH, on libère la place occupée par les <record-variables> en générant l'instruction WITH(-i) qui permet de faire

SP := SP-i.

a - nom(Record-Var) ; affect(W,0) ; with(+1) ;

<inst> ; WITH(-1) ;

b - nom(R1) ; affect(W,0) ;

NOM(R2) ; AFFECT(W,1) ;
:
:
:
NOM(Rn) ; AFFECT(W,n-1) ;
WITH(+n) ;

NOM(w,i)
<inst> ------> dans <inst> on fait
CHAMP(p)

WITH(n)	10100100		n
AFFECT(W,i)	100000	1110	i
NOM(W,i)	110000	1110	i

II.1.2.2. Affectations

Affectation à une variable simple

V := <exp> ;

<exp> ; AFFECT(V) ;

AFFECT(s,d) | 100000 | (s,d) |

Affectation à une variable structurée

<address-exp> := <value-exp> ;

<address-exp> ; <value-exp> ; ASSA

ASSA | 01011000 |

Cas particuliers pour une variable simple

V := V+1 -> INCV(V)

V := V-1 -> DECV(V)

V := 0 -> CLRV(V)

V := 1 -> SETV(V)

INCV(s,d) | 100010 | (s,d) |

DECV(s,d) | 100011 | (s,d) |

CLRV(s,d) | 101110 | (s,d) |

SETV(s,d) | 101111 | (s,d) |

Cas particuliers pour une variable structurée

<EA> := <EA>+1; -> <EA>;INCA;

<EA> := <EA>-1; -> <EA>;DECA;

<EA> := 0; -> <EA>;CLRA;

<EA> := 1; -> <EA>;SETA;

<EA> := n; -> <EA>;ASSI(n);

<EA> := <EA>+n -> <EA>;INCAI(n);

<EA> := <EA>-n -> <EA>;DECAI(n);

<EA>[n] := <exp> -> <EA>;<exp>;ASSXI(n)

INCA	00011100	ASSI(n)	00011011	n
DECA	00011101	INCAI(n)	00011000	n
CLRA	00011110	DECCAI(n)	00011001	n
SETA	00011111	ASSXI(n)	00011010	n

II.1.2.3. Références et valeurs immédiates

Références à une variable

NOM(s,d)	110000	(s,d)
INDD(s,d)	110001	(s,d)

Remarque
Pour V := <exp>

INDD(V) ; <exp> ; ASSA = <exp> ; AFFECT(V)

Valeurs immédiates

ZERO-INT	11010000	
ZERO-SET	11010001	
ONE-INT	11110000	

ne sont générés que pour la construction d'un POWERSET, instructions ELEM et INTER

INT8(v)	11100000	v
SET8(v)	11100001	v
CHAR(v)	11100010	CHAR-CODE
INT16(v)	11100100	v :
SET16(v)	11100101	v :
INT32(ad)	11101000	ad:
SET32(ad)	11101001	ad:
REAL32(ad)	11101011	ad:
INT64(ad)	11101100	ad:
REAL64(ad)	11101111	ad:
SETV(L,ad)	11111100	L
CS(L,ad)	11111010	L

1.2.4. Opérateurs

Opérateurs arithmétiques

ADD	01100000
SUB	01000001
MULT	01100010
DIV	01100011
IDIV	01100100
MOD	01100110
ODD	00101000
INC	00100000
DEC	00100001
INCI(n)	00100100 | n
DECI(n)	00100110 | n
NEG	01010000
ABS	00101001
TRUNC	00101010
INT	00100010
CHR	00100011

Opérateurs d'accès

INDEX	01000000
INXI(p)	00000000 | p
CHAMP(p)	00111100 | p

Opérateurs de comparaison

EQ	01110000
NEQ	01110001
LT	01110100
LE	01110101
GT	01110110
GE	01110111
EQo	00110000
LTo	00110101
LEo	00110100
NEQo	00110001
GEo	00110110
GTo	00110111
EQ1	00110010
LT1	00111001
LE1	00111000
NEQ1	00110011
GE1	00111010
GT1	00111011

Opérateurs logiques

AND	01101000
OR	01101001
XOR	01101010
XAND	01101011
NAND	01101100
NOR	01101101
XNOR	01101110
XNAND	01101111
NOT	00101100

Opérateurs ensemblistes

IN	01111100
ELEM	01111000
INTER	01001100

```
S := [1, I...J,3] ;
ZERO-SET ; ONE-INT ; ELEM ;
NOM(I) ; NOM(J) ; INTER ; OR ;
INT8(3) ; ELEM ; AFFECT(S) ;
```

TABLEAU RECAPITULATIF DU CODAGE DES INSTRUCTIONS

IF(m)	04	CALL(s,d)	A8	ADD	60
				SUB	61
NEHT(m)	D4	CALLF(s,d)	B0	MULT	62
				DIV	63
ELSE	D8	ENTER	98	IDIV	64
				MOD	65
FI	AC	RETURN	9C		
		RETURNF	9E	AND	68
				OR	69
				XOR	6A
				XAND	6B
FORUP(m)	50	ASS(s,d)	80	NAND	6C
				NOR	6D
FORDOWN(m)	54	PARAM(s,d)	84	XNOR	6E
				XNAND	6F
EXITFOR	94	INCV(s,d)	88	EQ	70
				NEQ	71
ROFUP	14	DECV(s,d)	8C	LT	74
				LE	75
ROFDOWN	16	CLRV(s,d)	B8	GT	76
				GE	77
		SETV(s,d)	BC		
				IN	7C
LOOP(m)	F8			ELEM	78
				INTER	4C
ENDLOOP	C8	NOM(s,d)	C0	EQ0	30
				LT0	35
EXITIF	08	INDD(s,d)	C4	LE0	34
				NEQ0	31
WHILE	0A			GE0	36
				GT0	37
UNTIL	0C	ASSA	58		
				EQ1	32
		ASSX	5A	LT1	39
				LE1	38
CASE(m)	10	INCA	1C	NEQ1	33
				GE1	3A
OF(m)	A0	DECA	1D	GT1	3B
FO	44	CLRA	1E	ODD	28
ESAC	48	SETA	1F	INDX	40
				INXI(p)	00
		ASSI(p)	1B	FIELD(p)	3C
GOTO(m)	F4	ASSXI(p)	1A	INC	20
				DEC	21
EXIT	AC	INCAI(p)	18	INT	22
				CHR	23
HALT	90	DECAI(p)	19	INCI(n)	24
				DECI(n)	226
EXITFOR	94			NEG	28
				ABS	29
WITH(n)	A4			TRUNC	2A

		NOT	2C
ZERO-INT	D0		
ZERO-SET	D1		
ONE-INT	F0		
INT8(v)	E0		
SET8(v)	E1		
CHAR(v)	E2		
INT16(V)	E4		
SET16(V)	E5		
INT32(ad)	E8		
SET32(ad)	E9		
REAL32(ad)	EB		
INT64(ad)	EC		
REAL64(ad)	EF		
SETV(L,ad)	FD		
CHARS(L,ad)	FF		

II.2. Le processeur d'accès PAC

II.2.1. Rôle de PAC dans PASCHLL

PAC doit alimenter la file d'évaluation avec des descripteurs de variables pour leur utilisation ultérieure comme opérandes par le processeur opératif.

Il doit donc soit:

- ACCEDER aux descripteurs,

- soit les CONSTRUIRE avant de les RECOPIER dans FILEVAL.

Ces deux alternatives illustrent deux types d'instructions d'accès traitées par PAC:

1/ les instructions du type REF(S,D),

2/ les instructions du type LITERAL.

Le doublet (S,D) représente le nom interne de la variable.

Il permet d'en calculer son adresse en mémoire et plus exactement dans la zone pile de contexte (CTX).

"S" est le niveau lexicographique et il représente le numéro d'une base à la valeur de laquelle il faut ajouter le déplacement D pour obtenir l'adresse dans CTX du descripteur de variable.

a - aucune référence n'a jamais été faite à cette variable, auquel cas il faudra aller la lire en mémoire à l'adresse calculée à partir du (S,D) pour l'écrire dans FILEVAL.

b - cette variable est en cours de modification, on dira qu'elle est sous dépendance (voir problème des dépendances dans la thèse de J.P.SCHOELLKOPF). PAC doit alors fournir un "descripteur de dépendance" à FILE en construisant éventuellement un maillon de la chaîne de reprise..

c - la variable a déjà été référencée et la dépendance a été résolue: son descripteur à jour se trouve dans la partie "mémoire cache" de la file des dépendances et il faut en avertir FILE pour qu'il le recopie dans FILEVAL.

Pour distinguer ces trois possiblités, PAC doit pouvoir consulter une table qui donne pour chaque (S,D) rencontré la position du descripteur correspondant. C'est la partie associative de FILEDEP.

Les instructions d'accès du deuxième type provoquent la construction d'un descripteur de variables à partir de l'instruction.

PAC est aussi impliqué dans l'interprétation des instructions

AFFECT(s,d) et PARAM(s,d)

qui sont en fait issues de la compaction d'une instruction d'accès (donc entraînant l'écriture d'un opérande dans FILEVAL) avec un opérateur diadique d'affectation.
C'est l'interprétation par le processeur FILE de ces instructions qui entraîne l'initialisation d'une dépendance se traduisant par la construction d'un nouveau point d'entrée dans le tableau des dépendances.

PAC a un autre rôle très important:

Il est chargé de la gestion de la zone "pile de contexte" (CTX) de la mémoire centrale.
Pour ce faire, il reçoit les instructions :

- d'appel (CALL s,d),
- d'entrée (ENTER) et
- de retour (RETURN)

de procédure ou de fonction.

L'instruction CALL provoque :

- la lecture du bloc de description du segment,
- la préparation de la zone valeur de ces paramètres,
- la recopie des descripteurs de paramètre et
- la préparation de la marque du bloc.

L'occurrence de l'instruction ENTER entraîne :

- la vérification du nombre des paramètres,
- la mise à jour des chaînages statiques et dynamiques,
- la recopie des descripteurs de variables locales,
- la mise à jour des registres SP et W avec vérification du débordement.

Accessoirement, elle permet à FILE de mettre à jour la file des dépendances.

Le retour de la procédure doit remettre

- CTX, les registres de base, le registce W

dans l'état où ils se trouvaient avant l'appel.

Les instructions manipulées par PAC sont les suivantes:

- instructions d'accès:
  - de type REF(s,d):
    - NOM(s,d)
    - INDD(s,d)
  - de type LITERAL:
    - ZERO
    - ONE
    - LIT(8)
    - LIT(16)
    - LIT(32)
  - concaténées avec un opérateur d'affectation:
    - AFFECT(s,d)
    - PARAM(s,d)
- instructions de gestion de la pile de contexte:
  - CALL(s,d)
  - CALLF(s,d)
  - ENTER
  - ENTEREXT
  - RETURN
  - RETURNEXT

II.2.2 Le chemin de données.

Ces quelques considérations préliminaires au sujet du rôle du processeur d'accès dans l'ensemble PASCHLL renseignent sur les besoins d'échange avec les autres processeurs et fait apparaître la nécessité de traitements parallèles.
Il ne faut pas que la recherche des noms internes des descripteurs dans la file des dépendances soit pénalisante pour l'accès aux opérandes, cette recherche doit donc être menée en parallèle avec le calcul d'adresse).

PAC reçoit ses instructions venant de PINS à travers une file d'attente BIPAC de 16 mots de 16 bits. La sortie de BIPAC est connectée à la partie contrôle pour le décodage des instructions et à la partie opérative pour permettre les calculs d'adresse mémoires à partir des 10 bits (S,D) de l'instruction.

Les dialogues avec la mémoire centrale (via PME) demandent :

- deux tampons de 32 bits :
  - l'un en lecture (RML),
  - l'autre en écriture (RME) et
- un registre d'adresse mémoire de 16 bits (RADM).

Les descripteurs accédés ou construits sont destinés à la file d'évaluation: un registre tampon de 32 bits (PACE) en émission sur le bus FILE est nécessaire.

La file des dépendances (FILEDEP) est une ressource partagée entre FILE et PAC:
PAC la consulte et FILE la modifie.
Elle est composée de deux parties physiquement dissociées :

- l'une associative (table adressée par les S,D), câblée sur la carte de PAC,
- l'autre contenant des descripteurs et située sur FILE.

La communication entre les deux se fait par le bus "pointeur" bidirectionnel.

La figure II.3 montre l'architecture globale de PAC et son insertion dans PASCHLL.

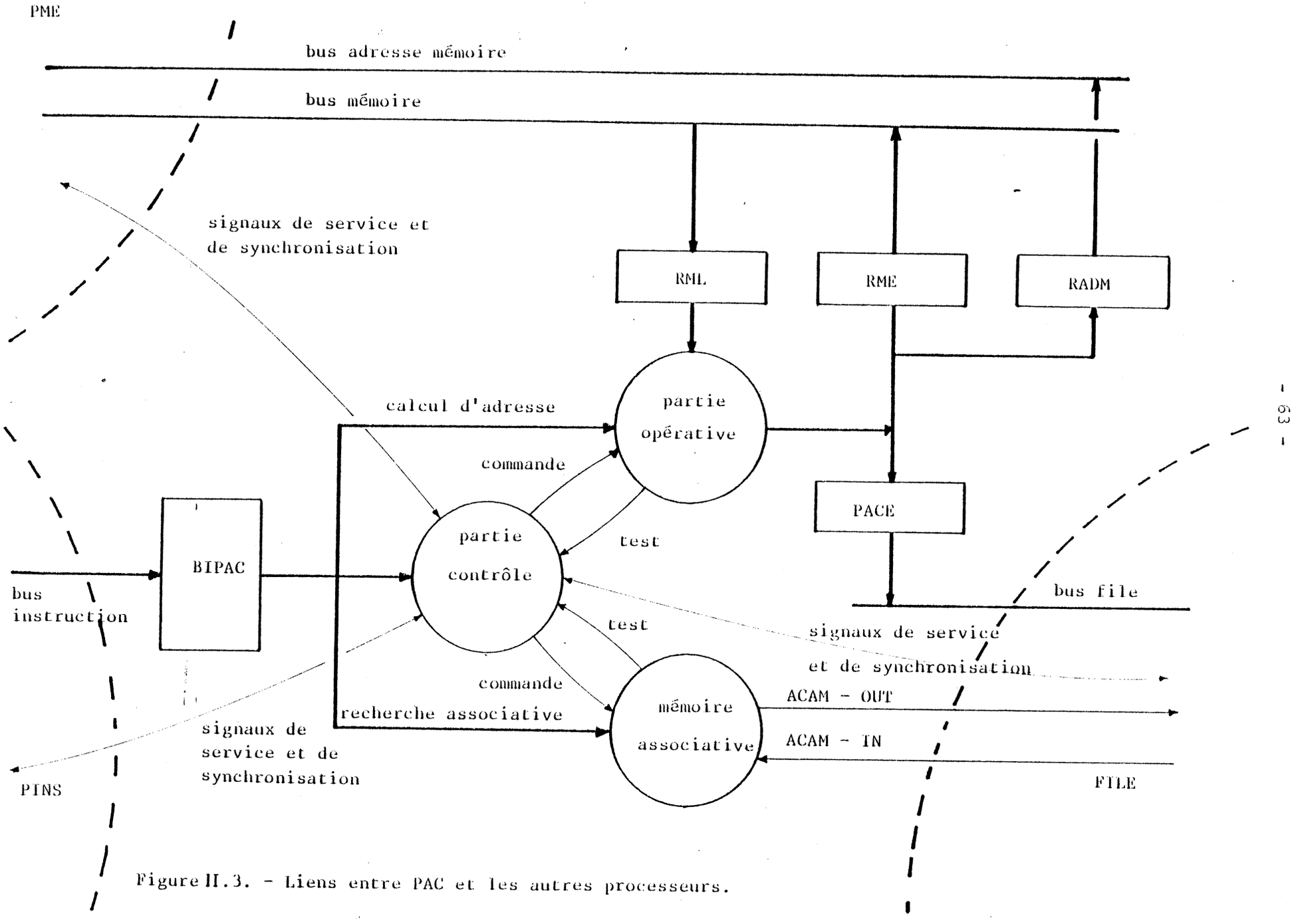


Figure II.3. - Liens entre PAC et les autres processeurs.

II.2.3. Rappels sur les modes d'adressage

Au cours de l'exécution d'un programme, l'adresse d'une variable est donnée par son nom interne qui est un doublet codé sur 10 bits:

(niveau lexicographique S, déplacement D).

Dans un souci de minimisation de la taille des instructions d'accés (très fréquentes), il a été retenu le codage suivant du nom d'une variable:

adressage d'une variable globale:

```
9 8 7 ..................0

0 0           D
```

D est un nombre positif permettant d'adresser jusqu'à 256 variables globales.

adressage d'une variable non globale:

```
9 8 7  ..................0

   s            d
```

d est un entier signé compris entre -64 et +63, permettant d'adresser jusqu'à 64 variables locales et 62 paramètres internes à une procédure.

Les bits 7 à 9 permettent 6 niveaux d'imbrication lexicographique (les deux configurations 000 et 001 étant exclues), ce qui est raisonnable sachant que la décomposition en module est toujours possible (et même souhaitable).

adressage relatif au sommet de pile

```
9 8 7 6 5  ..............  0

1 1 1 1          d
```

Entre le début du passage des paramètres défini par l'instruction CALL et l'entrée effective dans une procédure définie par l'instruction ENTER, les paramètres sont adressés par rapport au sommet de pile SP qui pointe sur la future base du bloc alloué à la procédure.

Au moment de l'entrée dans la procédure, le NOM des paramètres doit être changé.

adressage relatif au registre W:

9 8 7 6 5 0

1 1 1 0 d

L'instruction WITH du langage PASCAL permet de définir un environnement d'adressage particulier à l'intérieur d'une procédure pour accélérer l'accès au champ d'une structure (Record de PASCAL):

Le descripteur d'une ou de plusieurs variables structurées est mis en facteur et il est adressé directement par rapport à un registre de base appelé W.

II.3. Le processeur opératif POP

II.3.1. Rôle de POP dans PASCHLL

Le processeur opératif est le processeur de calcul de PASCHLL. Les instructions qu'il doit exécuter sont issues d'un registre (RIPOP) chargé à partir d'une file d'attente (BIPOP) remplie avec les opérateurs et les extra-ordres par le processeur d'instructions PINS.

Les opérateurs peuvent être les opérateurs arithmétiques et logiques du langage PASCAL mais aussi des instructions de calcul d'indexation de tableau (INDEX) ou d'accès à un élément dans une structure "record" (CHAMP) ou d'affectation (AFFECT)...

PASCHLL est une machine à préfixe :
toutes les variables sont décrites et elles sont rangées dans la zone "pile de contexte" (CTX) en mémoire centrale sous forme de descripteurs de 32 bits.
Les opérateurs manipulés par POP s'appliquent généralement sur le champ valeur (ou adresse de la valeur) PV de ces descripteurs, cependant l'information contenue dans le champ préfixe (PF) est nécessaire pour préciser l'opération et générer un point d'entrée dans le microprogramme.
En effet, il n'existe par exemple qu'un seul opérateur d'addition (ADD) mais son exécution sera différente s'il s'applique

- à deux entiers courts, ou

- à deux réels (Il faudra commencer par aller chercher les valeurs en mémoire, puis s'occuper des mantisses et des exposants) , ou

- entre un entier et un réel (on commence par faire une conversion entier -> réel avant de faire l'addition entre deux réels).

Le code opération de POP sera donc constitué par le contenu du registre instructions (BIPOP) et les préfixes des opérandes sur lesquels l'opération s'applique.

Ceci permet en outre des vérifications: on peut facilement déceler des incompatibilités entre tel opérateur et tel type d'opérande.

D'autre part, POP contrôle le déroulement algorithmique du programme: il est le seul à connaître le résultat des prédicats décidant du séquencement du programme et il doit avertir PINS du bien-fondé de l'anticipation qu'il a pu faire.

II.3.2. Organisation du chemin des données

Les spécifications externes du processeur POP définissent ses relations avec les autres processeurs en termes de synchronisations élémentaires et d'échange de données.

- Les relations entre PINS et POP ont déjà été définies et demandent un registre de 8 bits (RIPOP) chargé à partir d'une file (BIPOP) de 16 mots de 8 bits, remplie depuis le bus instruction sous contrôle de PINS.

- POP a besoin d'échanger des descripteurs de variables ou de type ou des valeurs longues (plus de 16 bits) avec la mémoire centrale via PME.

- FILE envoie à POP et sur sa demande les opérandes sous forme de descripteurs et reçoit les résultats intermédiaires ou définitifs à ranger dans FILEVAL ou dans FILEDEP.

Ces besoins d'échanges de données avec l'extérieur déterminent les éléments d'entrée et de sortie:

- avec PME:
  - un port d'entrée/sortie de 32 bits avec un registre d'entrée (RML) et un registre de sortie (RME).
  - un registre d'adresses de 16 bits (RADM).
- avec FILE:
  - un port d'entrée/sortie de 32 bits avec un registre d'entrée POPL et un registre de sortie POPE.

Les adresses comme les champs valeurs ou adresse de la valeur des descripteurs manipulés par POP ont une largeur de 16 bits.
Nous avons donc choisi un chemin de données de 16 bits et en particulier l'opérateur est constitué de 4 tranches Am 2903 ("super slice").
Les registres d'entrée et de sortie en communication avec les bus mémoire et FILE seront divisés en deux (avec indice 0 pour les poids faibles et indice 1 pour les poids forts).

Outre les registres d'entrée/sortie, d'autres éléments de mémorisation sont connectés sur le chemin de données. Les 16 registres internes de l'opérateur contiennent les variables de travail comme :

- SP (sommet de pile),
- BT (base des descripteurs de type),
- BC (base des constantes),
- $R2^0$ et $R2^1$ (valeur du deuxième opérande),
- $R1^0$ et $R1^1$ (valeur du premier opérande),
- AL et AE (adresse de lecture et adresse d'écriture et
- des registres banalisés utilisés pour le traitement d'instructions complexes.

POP a besoin d'une pile pour stocker les résultats intermédiaires longs (plus de 16 bits) qui ne peuvent être rangés dans FILEVAL et pour recevoir le contenu de FILEVAL à sauvegarder à toute occurrence d'un appel de fonction.

Cette pile de POP est en mémoire centrale, mais il a semblé intéressant d'implanter les 16 derniers èlèments dans une mémoire locale de 16 mots de 32 bits adressés par un compteur K et dont l'occupation est contrôlée par un compeur N (voir gestion de cette pile dans la thèse de J.P.SCHOELLKOPF).

Il est noter que le registre SP contient l'adresse du premier élément de la pile en mémoire centrale. On distinguera PILE0 et PILE1 sur le chemin de données.

D'autres éléments viennent se greffer sur le chemin de données: ce sont des opérateurs spécialisés comme :

- un générateur de profils binaires(ROMSET) utilisé pour les opérations ensemblistes,
- un dispositif de masquage permettant la modification d'un seul octet dans un mot avant une écriture mémoire,
- un système permettant des extensions de signe de 8 à 16 bits ainsi que des croisements d'octets (data path switch).

L'organisation du chemin de données autour de ces éléments répond à des critères algorithmiques (transferts et opérations à faire entre les différents éléments de mémorisation) et à des critères de performance (introduction d'opérateurs spécialisés, possibilité de parallélisme, choix entre traitement câblé et traitement microprogrammé).
Ils peuvent être classés de la façon suivante:

éléments source poids faibles

POPL0, RML0, PILE0, BIPOP, ROMSET, RA et RB (registres internes à l'opérateur pouvant contenir les champs valeurs du premier et du second opérande par exemple).

éléments source poids forts

POPL1, RML1, PILE1, (PF1, PT1), (PF2, PT2) contiennent les champs préfixes et pointeurs sur type des descripteurs de variables dont les champs PV sont dans R_0 et R_1).

éléments destination poids faibles

POPE0, RME0, PILE0, RADM, ROMSET, RB

éléments destination poids forts

POPE1, RME1, PILE1, (PF1, PT1), (PF2, PT2), RPF (registre de travail sur les préfixes).

Il parait souhaitable et même indispensable qu'il y ait une certaine autonomie entre ce que l'on pourrait appeler chemin de données poids fort et chemin de données poids faible. Il est par exemple logique que les transferts de la PILE vers la mémoire, ou de la mémoire vers la PILE, puissent se faire sur 32 bits à la fois.
Il en est de même pour l'acquisition d'un descripteur quelle qu'en soit sa source: FILEVAL ou mémoire.
Il est donc nécessaire d'avoir le parallélisme suivant:

- transfert mémoire -> PILE

  RML0 -> PILE0 // RML1 -> PILE1

- transfert PILE -> mémoire

  PILE0 -> RME0 // PILE1 -> RME1

- passage d'un descripteur par FILE

  POPL0 -> RB // POPL1 -> (PF1, PT1)
  ou
  POPL1 -> (PF2, PT2)

- accès à un descripteur en mémoire (indirection)

  RML0 -> RB // RML1 -> (PF1, PT1)
  ou
  RML1 -> (PF2, PT2)

L'indépendance entre les poids faibles et les poids forts et les manipulations différentes entre les préfixes et les valeurs (ou adresses) nous ont conduit à définir deux bus d'entrée:

- BUS0 (pour les poids faibles) et
- BUS1 (pour les poids forts) avec possibilité de communication BUS1 vers BUS0, et

et un bus de sortie unique Y dont les octets peuvent être tout d'abord croisés puis multiplexés avec le BUS1 pour charger les registres de sortie poids forts.

L'organisation générale des chemins des données est montrée dans la figure II.4

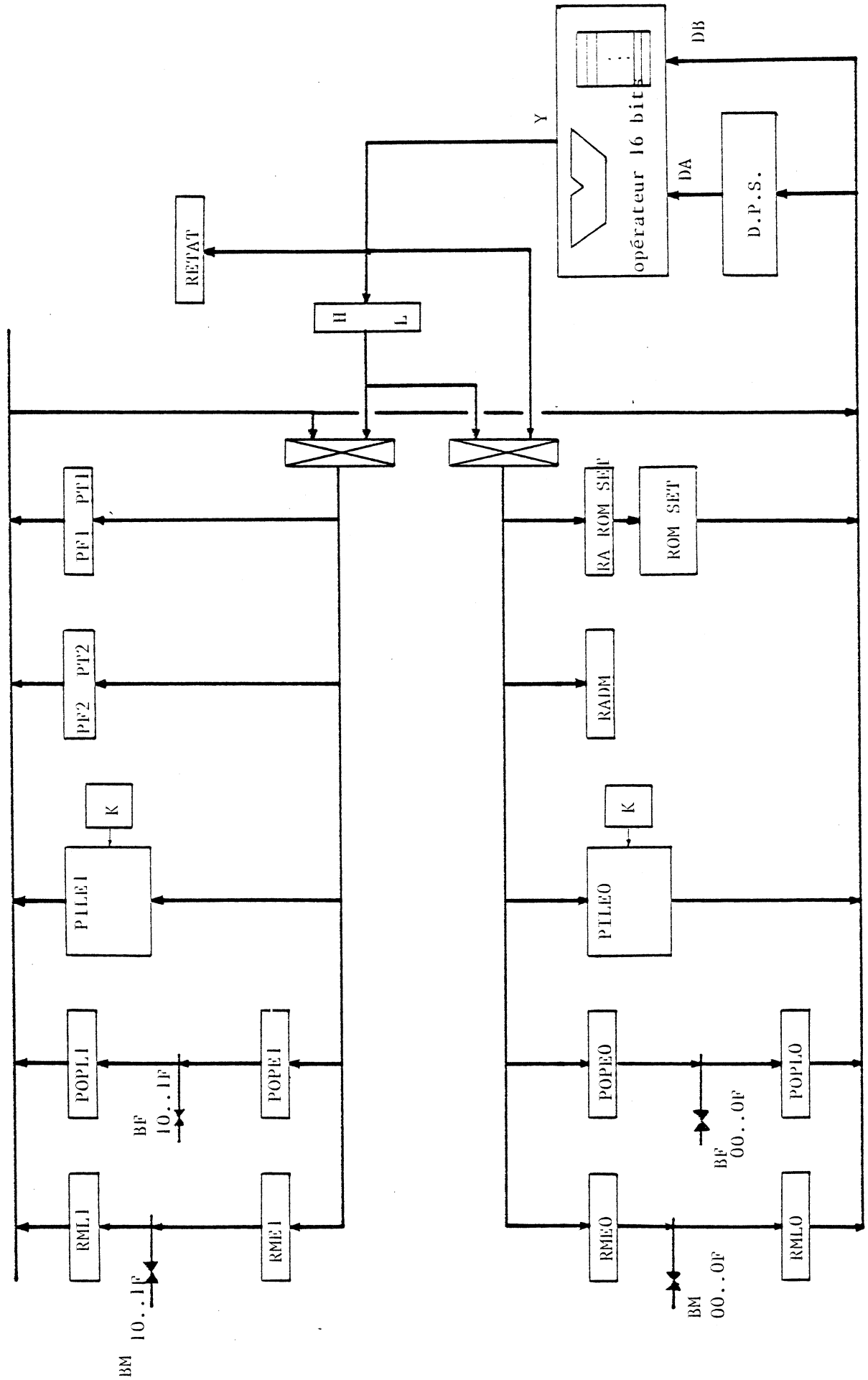


Figure II.4. - Processeur POP: organisation du chemin de données

II.4. Le processeur de gestion des Files d'EVALuation et des DEPendances: FILE

II.4.1. Rôle de FILE dans PASCHLL

Contrairement aux trois processeurs précédents, FILE ne découle pas naturellement de la décomposition du travail d'une unité centrale d'ordinateurs

- en accès aux instructions,
- accès aux oprérandes,
- exécution des opérateurs et
- rangement des résultats.

Il a été introduit pour contrôler le flot des données entre PAC et POP à travers les mémoires de travail organisées en files d'attente FILEVAL et FILEDEP et le bus interne des descripteurs (bus FILE).

Il ne reçoit pas directement les instructions qu'il doit exécuter du processeur d'accès aux instructions PINS, mais répond aux ordres générés par PAC et POP en même temps que leur requête.

Ordres passés par PAC

- rangement du descripteur de variable se trouvant dans le tampon PACE,
- construction d'un descripteur de dépendance et d'un maillon de chaîne de reprise,
- transfert d'un élément de FILEDEP dans la file d'évaluation,
- initialisation d'une dépendance,
- initialisation d'une dépendance après marquage à VIDE d'une position de FILEDEP.

Ordres passés par POP .

- passage d'un opérande pointé par P2,
- passage d'un opérande pointé par P1,
- stockage d'un résultat intermédiaire,
- résolution d'une dépendance,
- AVANCE (n) ou INIT (n)
- RECUL (n)

En fait, FILE a vis-à-vis de POP un rôle de préparation et il analyse toutes les instructions à la sortie de BIPOP avant le processeur opératif.

Les instructions AVANCE, INIT et RECULE sont les "extra-ordres" de mouvement des pointeurs générés par le processeur PINS après examen de la chaîne postfixée des instructions qu'il analyse.

II.4.2. Chemin de données

Le processeur FILE peut être décomposé en plusieurs modules:

- les mémoires de travail FILEVAL et FILEDEP,
- un opérateur de 4 bits qui travaille sur les pointeurs vers ses mémoires,
- un automate allocateur permettant de prendre en compte les demandes de PAC et de POP sous certaines conditions,
- un module de commande du bus interne de description.

Le chemin de données du processeur FILE est représenté sur la figure II.5.

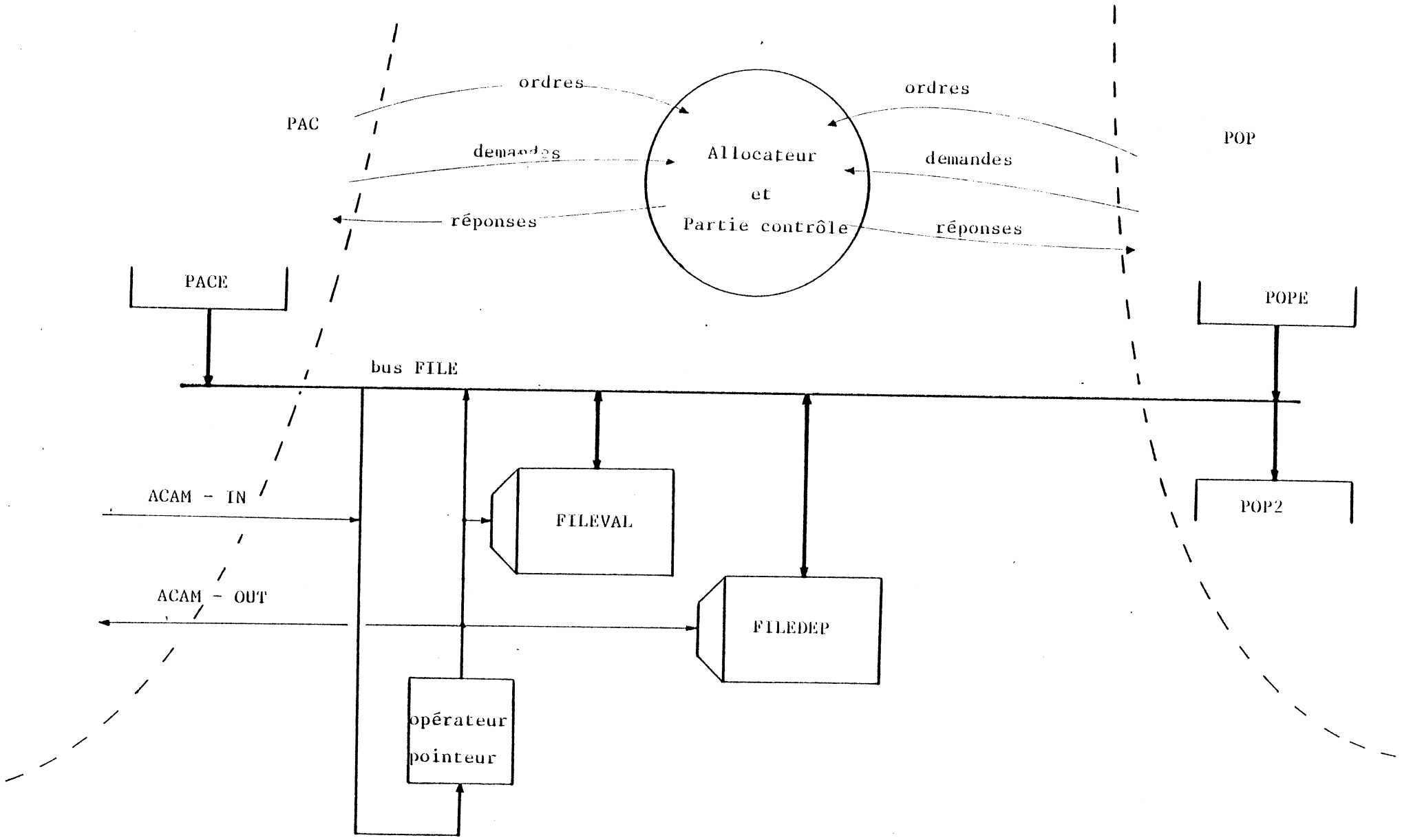


Figure II.5 - Chemin de données de processeur FILE.

II.5. Le processeur de dialogue avec la mémoire centrale PME

II.5.1. Rôle de PME dans PASCHLL

PASCHLL a été définie comme un processeur fonctionnel de traitement spécialisé dans l'exécution de programmes manipulant des données se trouvant dans la mémoire centrale. Les fonctions "systèmes d'exploitation" (gestion des programmes dans un système de type "traitement par bits", gestion de la mémoire centrale, dialogue avec les organes d'entrée/sortie) sont laissées à la charge d'une autre unité centrale hôte: la machine système.

Cette décomposition fonctionnelle suppose un transfert des programmes entre la fonction d'exécution et la fonction système.

Pour minimiser les temps de transfert, nous avons choisi d'utiliser une mémoire centrale unique partagée par les deux processeurs (PASCHLL et Machine système) et physiquement composée de plusieurs blocs indépendants accessibles aux deux processeurs.

Toutes les données relatives à l'exécution d'un programme devront se trouver dans le même bloc.
Chaque bloc est alternativement accédé

- par PASC-HLL (il contient un programme en exécution) et

- par la machine système (lorsqu'une opération système est exécutée).

Pour rendre plus aisé et plus souple la connexion de PASC-HLL avec la Mémoire Centrale géré par le système hôte, le processeur PME a été introduit.

Il réalise l'interfaçage PASC-HLL -> Mémoire Centrale.

II.5.2. Rappel sur l'espace d'adressage de la machine

PINS, PAC et POP adressent les informations dans un espace d'adressage virtuel composé de quatre zones :

- la zone CODE contenant des informations en lecture seule générées par le compilateur : table des types, table des constantes, code des procédures.

- la zone EXTERNE contenant le même type d'informations, mais relatives à des modules externes issus de compilations séparées.

- la zone CTX contenant la pile de contexte de l'exécution du programme et reflétant l'imbrication dynamique des procédures.

- la zone DYN réservée à l'allocation dynamique des données.

Les adresses générées par chacun des processeurs occupent 16 bits:

- les deux bits poids forts indiquent le numéro de la zone concernée,

- les 14 bits poids faibles indiquent le déplacement dans cette zone.

Chacune des zones peut donc contenir jusqu'à 16 Kmots.

Le processeur PME est chargé de fournir une adresse réelle à la mémoire centrale. Pour cela son algorithme d'interprétation est divisé en deux étapes:

- une étape d'initialisation pendant laquelle le système d'exploitation lui fournit une table décrivant l'implantation réelle et l'encombrement des quatre zones en mémoire centrale.

BASE CTX	LIMITE CTX
BASE DYN	LIMITE DYN
BASE CODE	LIMITE CODE
BASE EXTERNE	LIMITE EXTERNE

- une étape de calcul pendant laquelle il opère la translation d'adresse:

BASE(no. de zone) + déplacement

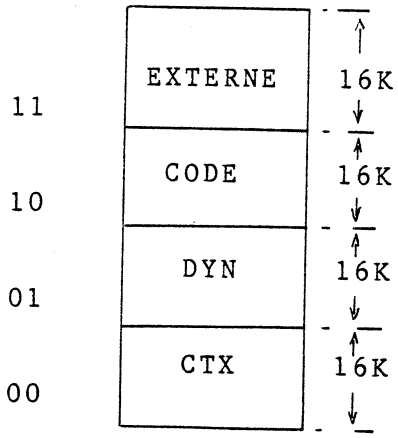
tout en vérifiant que la relation:

[BASE(no. de zone) + déplacement] < LIMITE (no. de zone)

et que les protections en écriture pour CODE et EXTERNE sont bien satisfaites.

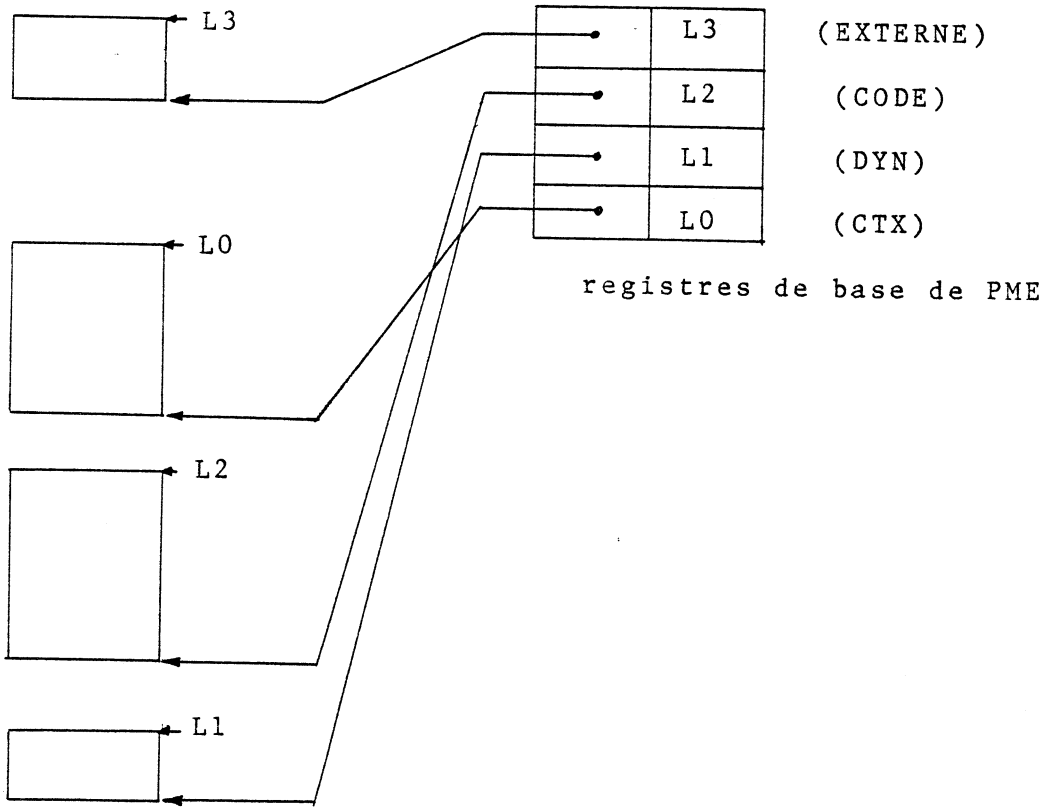
Il est à noter que l'adressage virtuel de la part de PASCHLL laisse entière liberté au système d'exploitation quant à l'implantation physique des 4 zones dans la mémoire centrale. De plus, il peut n'allouer à chaque zone que la taille qui lui est nécessaire (l'encombrement de CODE et EXTERNE est donné par le compilateur, quant à celui de CTX ou de DYN, il est fonction de la complexité du programme et peut être définie par le programmeur comme paramètre d'une instruction système).

Espace virtuel



mémoire virtuelle de PASC-HLL

Espace réel



registres de base de PME

mémoire réelle du sytème "h0te"

Après initialisation par la machine système, le processeur de dialogue avec la mémoire centrale doit remplir deux fonctions:

- une fonction d'allocation du bus mémoire à l'un des demandeurs,

- une fonction opérative consistant à faire la translation adresse virtuelle -> adresse réelle et le contrôle de dépassement des bornes.

La figure II.6 montre l'insertion de PME dans l'architecture globale de PASC-HLL et ses liens avec les autres processeurs.

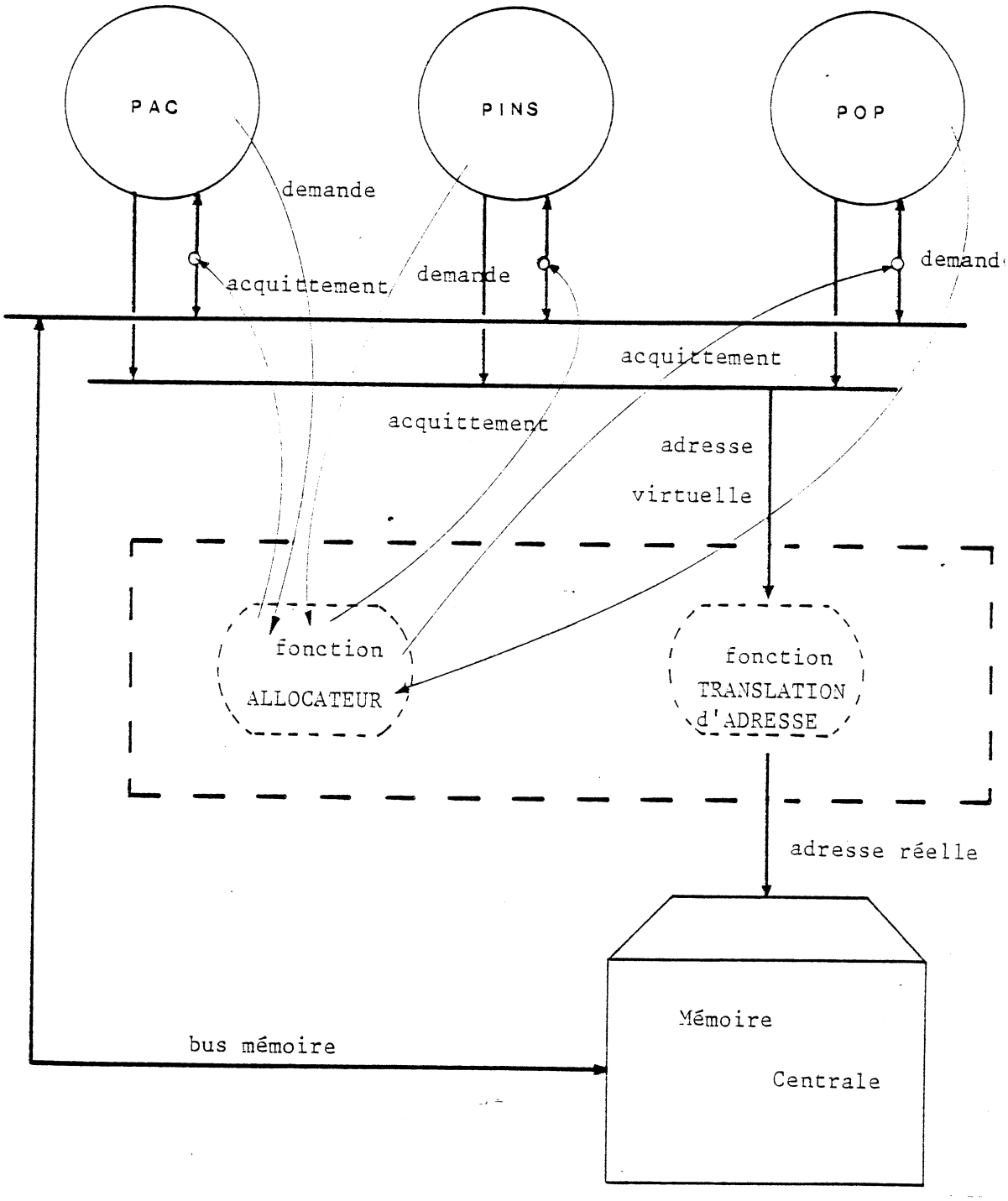


Figure II.6. - PME: liens avec les autres processeurs.

chapitre III.

APPLICATION DE LA METHODE DE CONCEPTION DESCENDANTE AU PROCESSEUR FILE

On a vu la nécessité d'introduire un processeur spécialisé dans la gestion des files d'évaluation et des dépendances utilisées à la fois par le processeur d'accès (remplissage) et le processeur opératif (extraction des opérandes et rangement des résultats intermédiaires).
FILE est une ressource partagée par PAC et POP.

III.1. Cahier des charges

III.1.1. Contrôle du bus FILE

Le rôle du processeur FILE est en fait de contrôler le flot des descripteurs de variables transitant entre PAC et POP par l'intermédiaire des files d'évaluation et des dépendances via un bus interne à PASCHLL que nous appellerons bus FILE.
FILE doit :

- d'une part lire dans un registre de PAC un descripteur de variables (ou de dépendances) ou une adresse par celui-ci calculée pour le ranger dans une des deux files , et

- d'autre part, fournir à POP les opérandes qui lui sont nécessaires (chargement d'un registre de POP) et lire les descripteurs modifiés pour les stocker dans une des files (lecture d'un registre de POP en vue du rangement d'un résultat intermédiaire).

III.1.2. Gestion des pointeurs

Quatre pointeurs sont nécessaires pour l'adressage de FILEVAL:

- ECR le pointeur d'écriture utilisé par PAC pour le rangement des descripteurs,

- P1 et P2 indiquant les positions respectives du premier et du second opérandes utilisés par POP,

- FIN qui marque la fin de l'espace de travail de POP.

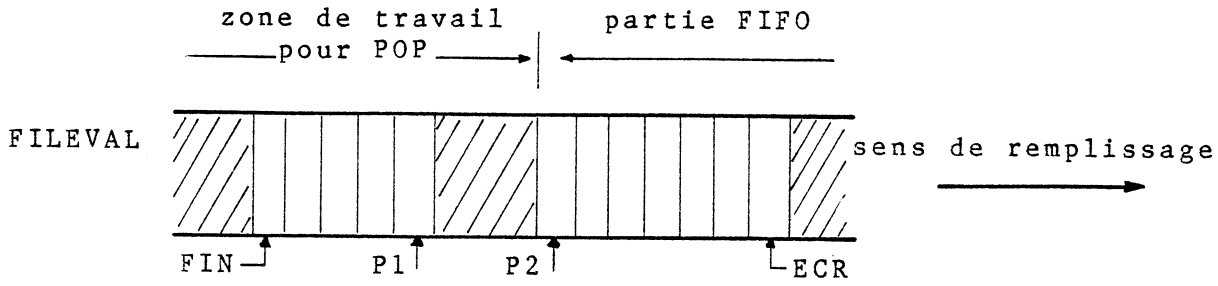


Figure III.1 - FILEVAL

(Pour plus de détails , se reporter à la description du mécanisme d'évaluation sur FILE).
La gestion de FILEDEP ne nécessite que deux pointeurs (voir le problème des dépendances):

- NIN le pointeur d'écriture,

- NOUT le pointeur de lecture donnant la frontière entre dépendances résolues et dépendances non résolues.

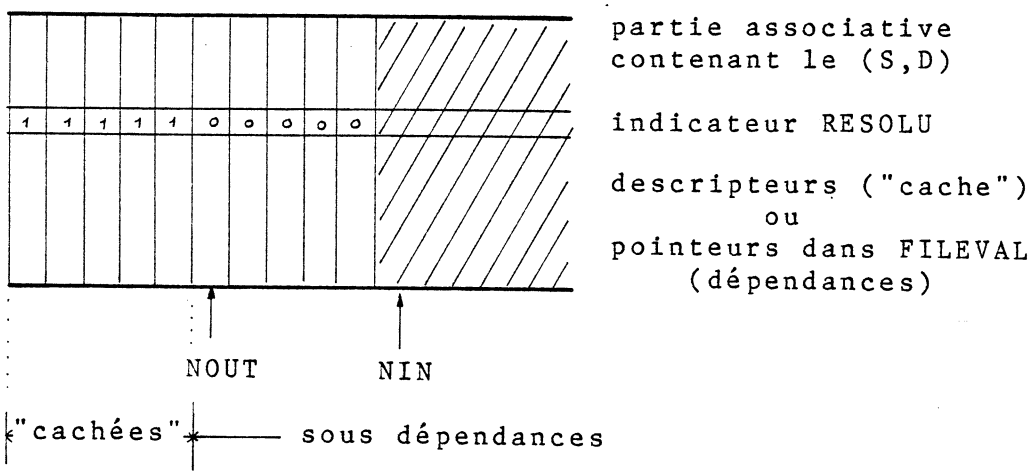


Figure III.2 - FILEDEP

Ces six pointeurs doivent être rangés dans des registres, on doit pouvoir :

- leur affecter une valeur,

- les incrémenter ou les décrémenter,

- leur ajouter ou leur retrancher une valeur entière inférieure à la taille en mots de ces mémoires,

- les comparer.

Toutes ces opérations doivent se faire modulo la taille des mémoires.

III.1.3. Allocation

Le processeur FILE est une ressource partagée par PAC et POP pour lesquels il doit alternativement travailler. Ceux-ci formulent une requête et FILE s'alloue soit à l'un, soit à l'autre.
Pour faire un choix il doit être muni d'un mécanisme d'allocation prenant en compte les demandes selon un ordre de priorité à définir, cet ordre pouvant être inversé si un déséquilibre notoire du pipe-line est détecté, en imposant en quelque sorte une attente au processeur trop en avance.

Cet automate d'allocation pourra être totalement ou en partie câblé.

III.1.4. Travail à exécuter pour PAC

PAC sollicite FILE pour deux grandes classes d'instructions qui sont le remplissage de FILEVAL et le remplissage de FILEDEP.

III.1.4.1. Remplissage de FILEVAL

Cette classe regroupe les instructions du type REF(S,D) et les instructions du type LITERAL.
Dans les deux cas, PAC aura chargé son registre de communication avec soit un descripteur de variable lu en mémoire (instruction NOM(S,D) ou construit (instruction LITERAL), soit un descripteur de dépendances (la variable de niveau lexicographique S et de déplacement D est sous dépendance) avant de demander à FILE le rangement dans FILEVAL (ECR) si toutefois cette écriture est possible.

Remarque:
Le fait que la variable soit sous dépendance peut se traduire par la création d'un nouveau maillon dans la chaîne de reprise.
Si le descripteur se trouve dans la partie "cache" de FILEDEP (RESOLU=1) PAC ne doit ni accéder en mémoire, ni construire de descripteur.
FILE fera simplement le transfert:

FILEDEP(adresse associée au S,D) -> FILEVAL(ECR)

III.1.4.2. Remplissage de FILEDEP

Ce sont les instructions du type AFFECT(S,D) ou PARAM(S,D).
Il s'agit d'initialiser une dépendance sur ce couple (S,D).

PAC construit une adresse d'octet

((base S + déplacement D)*4)

qu'il range dans son registre de communication et demande à FILE de ranger en FILEVAL(ECR) un mot constitué par cette adresse en poids faible et les champs PF et PT du descripteur de la variable en poids fort. Ce mot sera récupéré par POP et servira de deuxième opérande à l'opérateur AFFECT. Ensuite FILE doit inscrire ce S,D en FILEDEP(NIN) après avoir marqué l'indicateur RESOLU à zéro et le pointeur sur FILEVAL à "NIL".

Remarque:
- Si la variable était déjà sous dépendance, FILE devrait s'en rendre compte et mettre PAC en attente jusqu'à ce que cette dépendance soit résolue.

- Si elle se trouvait dans la partie "cache" il faudrait d'abord l'extraire de façon à ce qu'il n'y ait pas de confusion possible avant d'initialiser une nouvelle dépendance.

III.1.4.3. Recherche associative

La file des dépendances est divisée en deux parties:

- une première partie avec possibilité d'adressage par le contenu dans laquelle est mémorisé le couple S,D (niveau lexicographique et déplacement) et un bit indicateur de résolution de dépendance.

- une seconde partie contenant :

  - soit un descripteur de variable (dépendance résolue),
  - soit un pointeur sur FILEVAL(début d'une chaîne de reprise) ce pointeur pouvant être nul (NIL).

Chaque fois que PAC extrait une instruction de type NOM(S,D), ID(S,D), AFFECT(S,D), PARAM(S,D), il doit examiner si la variable associée au couple (S,D) est ou n'est pas sous dépendance avant de faire une lecture mémoire.
Afin de ne pas pénaliser l'exécution il semble préférable que la partie de FILEDEP adressable par son contenu, soit située sur la carte du processeur PAC et que celui-ci soit prioritaire quant à son utilisation.
En même temps qu'il calcule une adresse à partir de (S,D) il peut la présenter à l'entrée de la mémoire associative et suivant le résultat , demander ou non une lecture mémoire centrale, plutôt que de demander au processeur FILE de se charger de cette recherche.
Cette dernière méthode augmenterait le temps d'allocation à PAC, donc pénaliserait POP et rendrait plus complexe l'algorithme de FILE qui devrait faire la distinction entre une instruction NOM(S,D) d'une instruction LITERAL.

D'un point de vue matériel la première méthode permet de diminuer

le nombre de connexions entre PAC et FILE.
Au lieu de 11 fils en fond de panier :
(10 pour le couple (S,D) + le bit RESOLU)
on n'en a que 4 :
(adresse codée dans FILEDEP = ACAM-OUT).

En outre, comme FILE ne connait que des instructions de remplissage de FILEVAL et des instructions de remplissage de FILEDEP, un seul bit issu de BIPAC lui est nécessaire. Il faut lui ajouter le résultat de la recherche (indicateur TROUVE) et le bit RESOLU.

Ces trois bits donnent six points d'entrée dans le microprogramme:

- ACCES, $\overline{\text{TROUVE}}$
- ACCES, TROUVE, NON RESOLU
- ACCES, TROUVE, RESOLU
- AFFECT, $\overline{\text{TROUVE}}$
- AFFECT, TROUVE, NON RESOLU
- AFFECT, TROUVE, RESOLU.

III.1.5. Travail à exécuter pour POP

III.1.5.1. Opérateurs simples

FILE doit fournir à POP le (ou les) descripteur(s) d'opérandes dont il a besoin pour exécuter un opérateur monadique ou diadique par l'intermédiaire du bus FILE.

Quand, dans une expression polonaise, on a une suite d'opérandes suivie d'une suite d'opérateurs diadiques, le passage du deuxième opérande (pointé par P2) ne se fait qu'une seule fois, au moment de la transition opérande-opérateur, tandis que le premier opérande (pointé par P1) doit être passé à chaque occurrence d'un opérateur diadique.

Un opérateur monadique s'applique toujours au deuxième opérande: il n'y aura donc pas de passage de descripteur.

La transition opérateur-opérande dans la chaîne postfixée entraîne le rangement du résultat intermédiaire dans le mot de FILEVAL pointé par P2.

On remarque que FILE a un travail de préparation vis-à-vis de POP pour l'exécution d'opérateurs simples.
Il parait judicieux que POP puisse disposer des opérandes qui lui sont nécessaires quand il exécute un opérateur, c'est-à-dire que FILE les lui ait fournis avant.

Ceci peut être réalisé simplement en introduisant un niveau de pipe-line supplémentaire entre eux deux:

- pendant que POP exécute un opérateur, FILE passe les opérandes pour l'opérateur suivant.

Il suffit d'ajouter un élément à la FILE d'attente BIPOP en interposant un registre RIPOP entre sa sortie et la partie contrôle de POP. Ainsi, pendant que POP traite l'instruction de RIPOP, FILE fait le travail de préparation relatif à l'instruction se trouvant à la sortie de BIPOP.

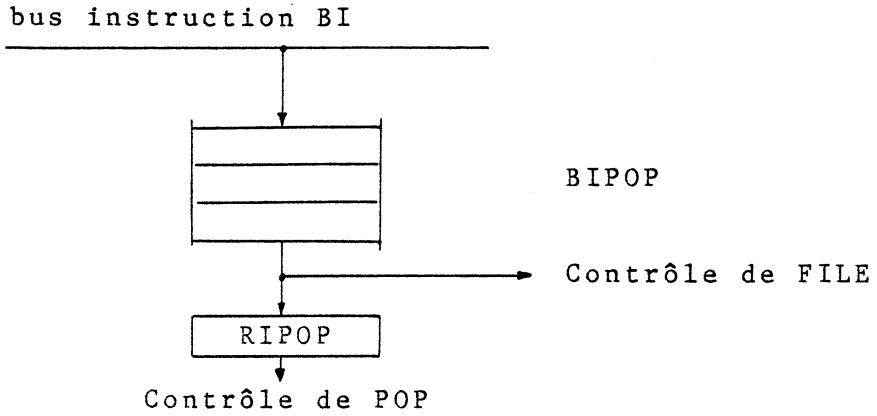


Figure III.3 - BIPOP

III.1.5.2. Extra-ordres

La gestion des pointeurs dans FILEVAL nécessite de la part de PINS une analyse de la chaîne postfixée et la génération éventuelle d'extra-ordres (INIT(n), AVANCE(n), RECUL(n)) (voir la description du mécanisme d'évaluation sur FILE).
Ceux-ci sont envoyés dans BIPOP via le bus d'instruction BI en vue de leur analyse par FILE.

Le deuxième opérande (celui pointé par P2) n'est envoyé à POP qu'après une transition opérande-opérateur de la chaîne postfixée.
Or cette transition est détectée par PINS et a pour effet la génération

- soit de l'extra-ordre AVANCE(n)

- soit de l'extra-ordre INIT(n).

Après avoir ajouté à P2 le paramètre n, FILE devra mettre FILEVAL(P2) dans le registre de communication avec POP (POPE).

Ainsi, il n'aura pas à faire la distinction entre le premier opérateur après une transition opérande-opérateur et les suivants, son algorithme sera identique dans les deux cas.

Le stockage d'un résultat intermédiaire, doit se faire, après que POP ait exécuté le dernier opérateur avant une transition opérateur-opérande. Elle sera nécessairement suivie de n opérandes ($0<n<16$) dans la chaîne postfixée puis par une transition opérande-opérateur.

Ce dernier opérateur se trouve juste avant l'extra-ordre AVANCE(n) dans BIPOP, c'est-à-dire qu'il est exécuté par POP pendant que FILE analyse AVANCE(n).
En conséquence, la valeur qu'avait P2 avant de lui ajouter n sera sauvegardée par FILE dans un registre de travail (RT) pour le rangement du résultat intermédiaire quand POP aura exécuté cet opérateur, c'est-à-dire quand il traitera AVANCE(n).

III.1.5.3. instruction AFFECTPOP

FILE doit fournir l'adresse à laquelle POP doit ranger le résultat pour le stocker dans la partie "cache" de FILEDEP et résoudre éventuellement des dépendances. Comme pour AVANCE cette instruction se déroulera en deux temps:

1/ préparation: envoi à POP de l'adresse mémoire

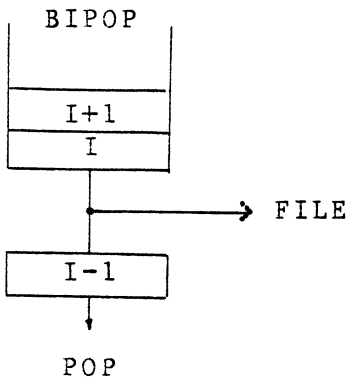
2/ stockage du résultat et résolution des dépendances.
Cette deuxième phase ne pourra être faite que lorsqque POP analysera à son tour l'instruction AFFECTPOP.

A ce stade de la description du cahier des charges, on voit apparaître le fait que les dialogues entre POP et FILE seront moins évidents qu'entre PAC et FILE du fait de ces instructions en deux étapes.

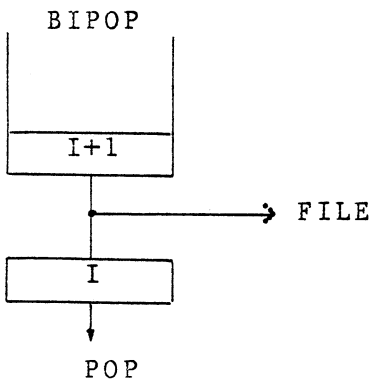
Ce type d'instruction remet en cause l'anticipation faite par FILE sur l'instruction que doit exécuter POP.

Soit I l'instruction nécessitant un traitement en deux temps par FILE.
Pendant que POP exécute l'opérateur (I-1) FILE peut faire la première phase de I mais la deuxième (rangement d'un résultat intermédiaire ou définitif) ne pourra se faire que lorsque POP l'aura préparé, c'est-à-dire aura décodé I.

C'est en effet I qui indique à POP l'emplacement de (I-1) dans la chaîne postfixée des instructions.



première phase



deuxième phase

Une solution serait qu'en fin d'exécution de chaque opérateur, POP aille décoder l'instruction suivante pour savoir s'il doit transmettre ou non un résultat intermédiaire à FILE.
Cette méthode oblige à mobiliser FILE pendant toute l'exécution des opérations (ce qui pénalise fortement PAC) et le fait de décoder l'instruction suivante enlève tout son intérêt au pipe-line.

On propose de résoudre ce problème d'une autre manière:

POP positionne une demande pendant l'exécution de I-1, FILE décode I et transfère FILEVAL (P2) dans le registre POPE , puis il acquitte la demande mais il positionne une bascule indiquant qu'il n'a pas terminé son algorithme.
Quand POP décode une nouvelle instruction (I), il demande à FILE la préparation de I+1. Celui-ci se rend compte qu'il n'a pas terminé le traitement de I.
Il va commencer par finir l'algorithme relatif à cette instruction avant de préparer l'opérande pour I+1.

III.1.5.4. instructions répétitives

Si, dans une expression, un opérande est le résultat d'une fonction, PINS peut avoir à demander à POP la sauvegarde de FILEVAL en mémoire (ordre(s) SAUVE(N)) avant d'initialiser l'évaluation du résultat de cette fonction.
Quand celle-ci sera terminée la restauration de FILEVAL sera demandée (ordre RESTAURE) pour poursuivre l'évaluation de l'expression.

Une troisième instruction répétitive existe. Il s'agit de OF(n) permettant la recherche d'une alternative dans une structure CASE.

Ces trois instructions ont un point commun : le transfert sur le

bus FILE de n descripteurs un à un au rythme imposé par POP.

L'utilisation de la bascule "pas terminé" est insuffisante.
Il faut que FILE se rende compte de la fin des transferts.
Pour cela il doit décrémenter le paramètre N à chaque pas.
Le passage de N à zéro indique la fin du transfert et FILE peut donc aller décoder la sortie de BIPOP afin de préparer l'opérande suivant.

III.2. Structure générale du processeur FILE

III.2.1. chemin de données

De la description du cahier des charges on peut déduire une architecture globale du processeur FILE.

Il est organisé autour de deux files FILEVAL et FILEDEP connectées sur le bus bidirectionnel de 32 bits interne à PASCHLL et servant à véhiculer les descripteurs de variables que nous appellerons bus-file (BF).
Il a la maîtrise totale du transit sur ce bus, c'est-à-dire que les commandes d'ouverture de "portes" vers ce bus et de chargement de registres à partir de ce bus sont générés par FILE, même s'ils sont géographiquement localisés sur d'autres processeurs.

La gestion de l'adressage dans les FIFO nécessite un opérateur pourvu de :

- registres internes permettant de stocker les valeurs des pointeurs nécessaires et un registre de travail,

- d'une Unité Arithmétique et Logique simplifiée permettant :

  - le transfert de registre à registre,
  - l'addition ou la soustraction d'une valeur entière inférieure à 16,
  - l'incrémentation ou la décrémentation

et pouvant fournir un certain nombre d'indicateurs d'état :

- passage à zéro,
- retenue sortante,
- signe du résultat...

qui seront testés par la partie contrôle.

Cet opérateur recevra sur son entrée, appelée bus pointeur d'entrée (BPE):

- soit le paramètre N issu de BIPOP pour les instructions répétitives de type SAUVE(n) ou les extra-ordres comme

AVANCE(n).

- soit l'adresse associée (ACAM-OUT) venant de la parttie associative de la file des dépendances pour pouvoir, par exemple, faire le transfert d'un descripteur de variable se trouvant dans la partie "cache" de FILEDEP vers FILEVAL:

FILEVAL(ECR) <- FILEDEP(ACAM-OUT)

- soit 4 bits du bus FILE permettant de remonter une chaîne de reprise dans FILEVAL pour résoudre les dépendances sur une variable après l'exécution par POP d'une affectation.

La sortie de l'opérateur (bus pointeur sortie (BPS)) doit pouvoir être transférée:

- dans le registre d'adresse de FILEVAL (ADEVAL)

- dans le registre d'adresse de FILEDEP (ADDEP)

- vers le bus FILE pour former un nouveau maillon de la chaîne de reprise

- vers la partie associative de FILEDEP (ACAM-IN) pour marquer à RESOLUE une dépendance.

La structure générale du chemin de données est représentée sur la figure III.5.

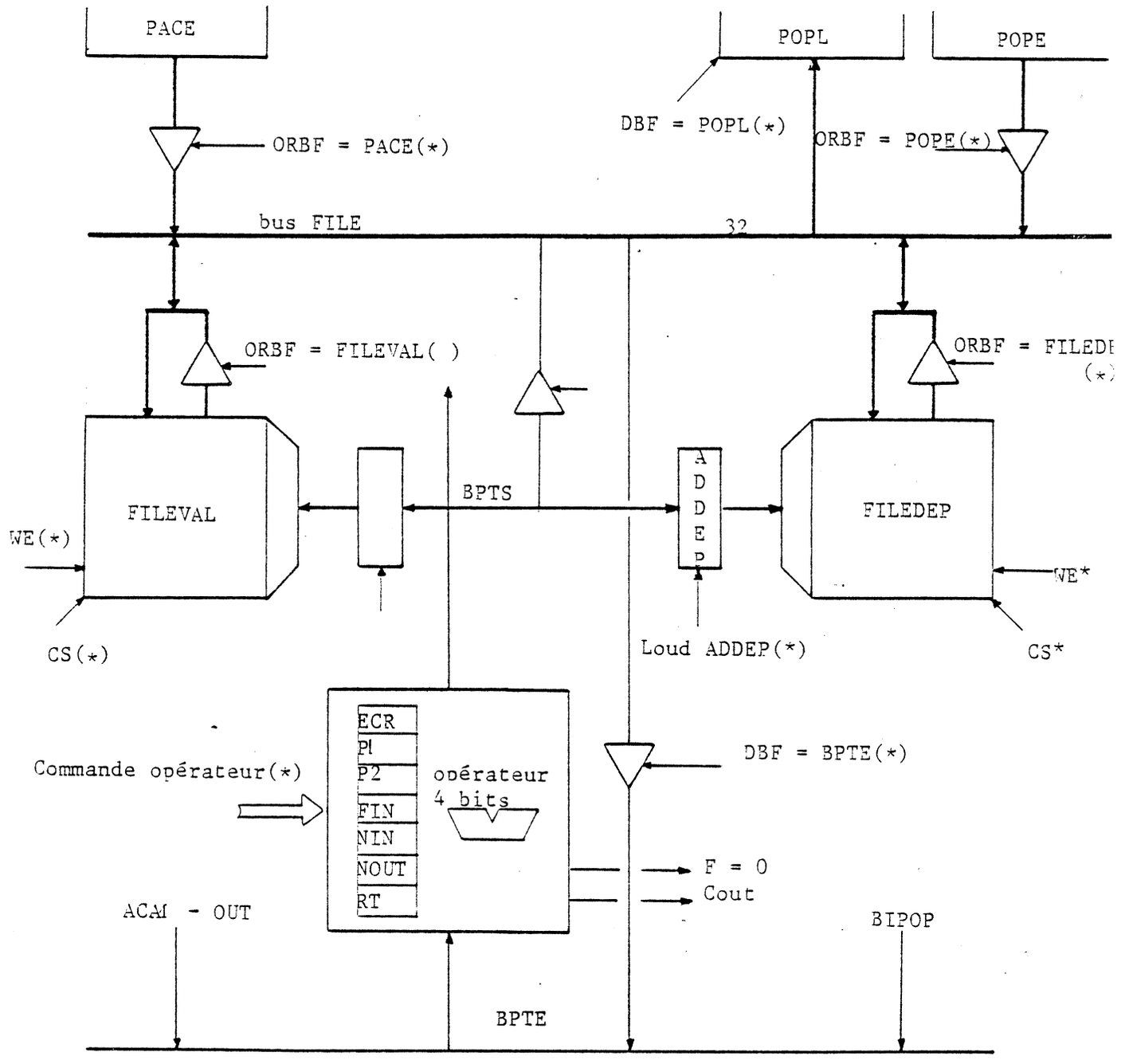


Figure III.5 - Structure du chemin de données de FILE

(*) Tous ces signaux sont contrôlés par le microprogramme de FILE.

III.2.2. partie contrôle

III.2.2.1. Structure

Comme les autres processeurs de PASCHLL, FILE est microprogrammé. Sa partie contrôle est donc bâtie autour :

- d'une mémoire morte contenant son microprogramme (CROM : Control Read Only Memory),

- d'un registre microinstruction (RmicroI) contenant la microinstruction en cours d'exécution et

- d'un dispositif permettant de calculer l'adresse de la microinstruction suivante (Séquenceur) en fonction :

  - du contenu de RmicroI,

  - d'événements extérieurs (conditions et attentes),

  - de l'état de l'automate d'allocation et

  - de l'instruction à exécuter pour PAC ou pour POP.

Sa structure globale est classique:

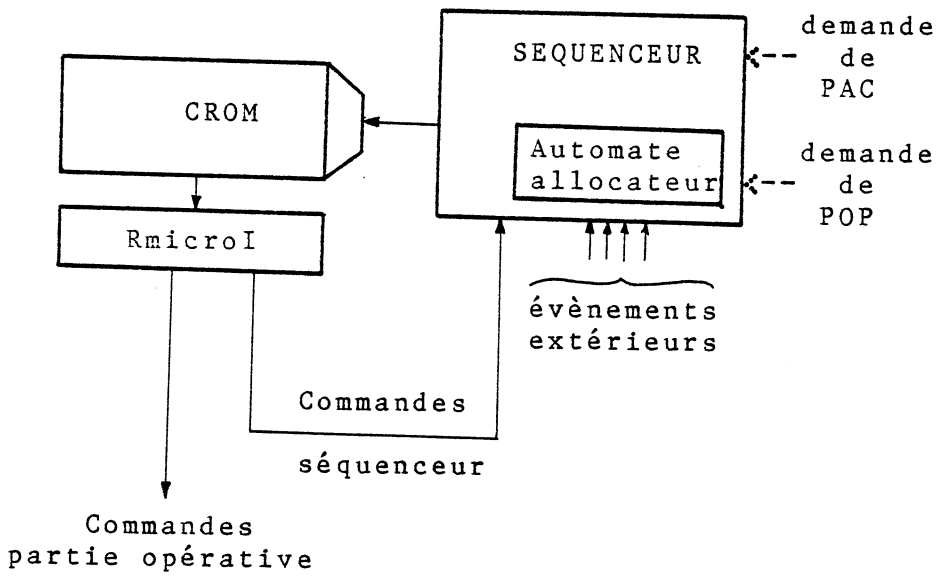


Figure III.6 - FILE : Partie contrOle

III.2.2.2. séquenceur

Il doit fournir l'adresse de la microinstruction suivante qui peut être soit:

- l'adresse de la microinstruction en cours d'exécution (se trouvant dans RmicroI) augmentée de un (passage en séquence).

- une adresse donnée par la microinstruction elle-même (possibilité de branchement ou d'appel d'une routine (dans ce cas, il faut avoir la possibilité de stocker l'adresse de retour et prévoir plusieurs niveaux d'appel (imbrication de procédure)).

- un point d'entrée fourni par l'un ou l'autre des demandeurs suivant l'état de l'automate allocateur.

Cette adresse doit pouvoir être conditionnée ou modifiée par :

- un évènement extérieur comme le résultat d'un test sur les pointeurs (signaux (F=0) ou (Cout) issus de l'opérateur ou bien l'état d'une bascule,

- une demande d'interruption venant de POP (signal TROMPE si PINS avait fait une mauvaise anticipation).

- un bit d'instruction permettant de choisir entre deux séquences de microprogramme.

Le séquenceur doit aussi être doté d'un mécanisme de gestion des attentes :
FILE boucle sur une microinstruction jusqu'à l'arrivée d'un évènement extérieur (la demande de l'un des processeurs par exemple).

Il doit assurer le multiplexage entre ces différentes sources d'adresse, en fonction de la commande de séquencement venant de RmicroI.
Il aura la structure suivante :

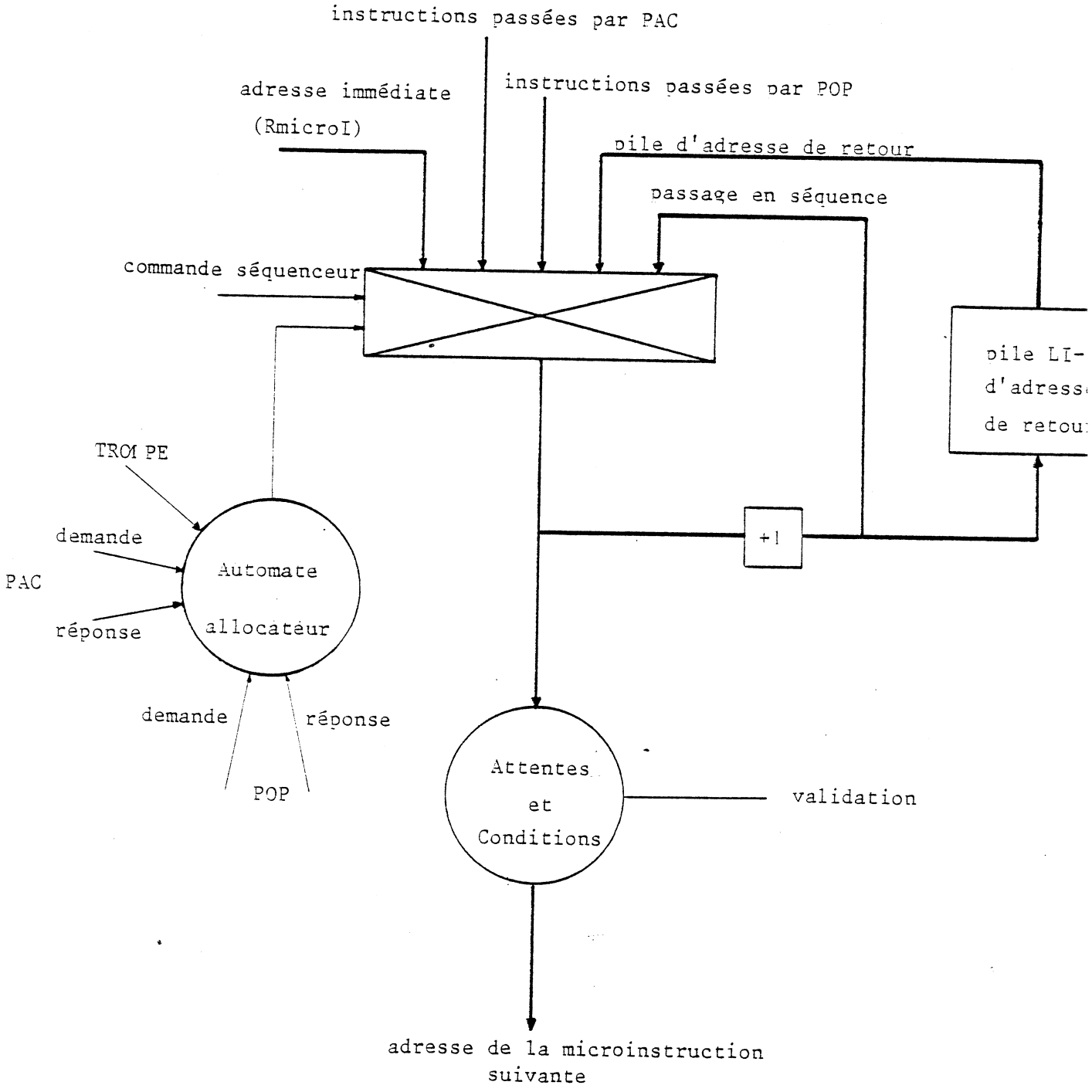


Figure III.7. - Structure du séquenceur

III.2.3. Structure générale du microprogramme

Au démarrage FILE doit se mettre dans un état "propre".
Pour cela il doit :

- initialiser la valeur de ses six pointeurs,

- positionner ses bascules d'état et de dialogue avec ses interlocuteurs.

C'est la phase d'initialisation qui doit être activée une fois pour toutes à la mise sous tension.

Ensuite il y a un cycle d'allocation pendant lequel FILE décide pour qui il doit travailler. Il peut être sollicité de trois manières différentes:

- par le signal TROMPE venant de POP

C'est la demande la plus prioritaire.
Elle est générée quand le prédicat évalué par POP ne correspond pas à l'alternative choisie par PINS.
Elle déroute le microprogramme vers une séquence de remise à jour de FILEVAL et de FILEDEP.

- par la demande de POP

Elle a la priorité 2 et correspond soit :

- à la demande d'un opérande
- au rangement d'un résultat intermédiaire ou définitif
- à un extra-ordre de mouvement de pointeur
- à une instruction répétitive.

L'opération demandée peut être impossible à réaliser: par exemple AVANCE(n) est impossible si la valeur de P2 augmentée de n est supérieure (modulo la taille de FILEVAL) à la valeur de ECR. Un déséquilibre du pipe-line est alors détecté et la demande de POP doit être masquée.
On dira que FILE met POP en attente volontaire.

- par la demande de PAC

Elle est la moins prioritaire. Elle est utilisée pour demander le rangement d'un nouveau descripteur dans FILEVAL ou pour initialiser une dépendance.
FILE doit aussi avoir la possibilité de la masquer si par exemple la valeur de ECR incrémentée de 1 devient égale à la valeur de FIN (modulo la taille de FILEVAL).

Le choix de l'ordre des priorités n'est pas dû au hasard:
Il est en effet logique que TROMPE soit la demande la plus prioritaire puisqu'elle correspond à un mauvais choix et qu'il faut vider les deux files.

Pour éviter que celles-ci ne deviennent un goulot d'étranglement nous avons donné la priorité la plus forte à l'extraction d'opérandes plutôt qu'au remplissage , c'est-à-dire à POP plutôt qu'à PAC.

Pendant cette phase d'allocation, on a un éclatement à trois directions (TROMPE, Allocation à POP, Allocation à PAC), les deux dernières donnant lieu à un nouveau branchement multidirectionnel suivant les instructions de PAC (CODEPAC) ou de POP (BIPOP) pour la phase d'exécution.

La structure générale du microprogramme peut être schématisée de la façon suivante:

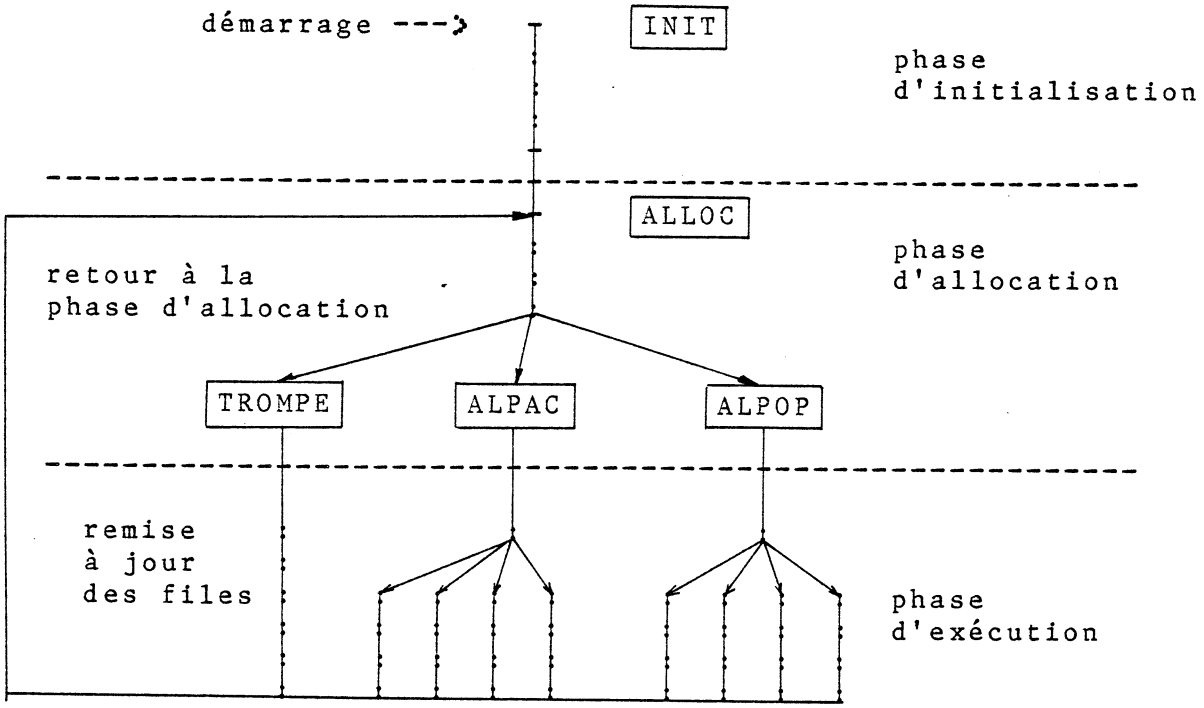


Figure III.8 - Structure générale du microprogramme.

III.3. REALISATION

III.3.1. choix matériel

La famille des circuits intégrés Am2900 semble tout à fait adaptée au processeur FILE.

Le composant Am 2901 regroupe dans un seul boîtier un ensemble de circuits MSI comprenant :

- une Unité Arithmétique et Logique de 4 bits,
- 16 registres généraux de 4 bits à double accés,
- un registre accumulateur de 4 bits,
- des multiplexeurs sur chacune des entrées et en sortie de l'UAL,
- une entrée D de 4 bits,
- une sortie Y de 4 bits,
- un système de décalage droite ou gauche du résultat d'une opération, avant son chargement dans un des registres ou dans un des registres et dans l'accumulateur,
- une matrice de décodage générant toutes les commandes d'opérations , de chargement, de décalage, d'origine des données en entrée d'UAL à partir de 9 bits de microinstructions.

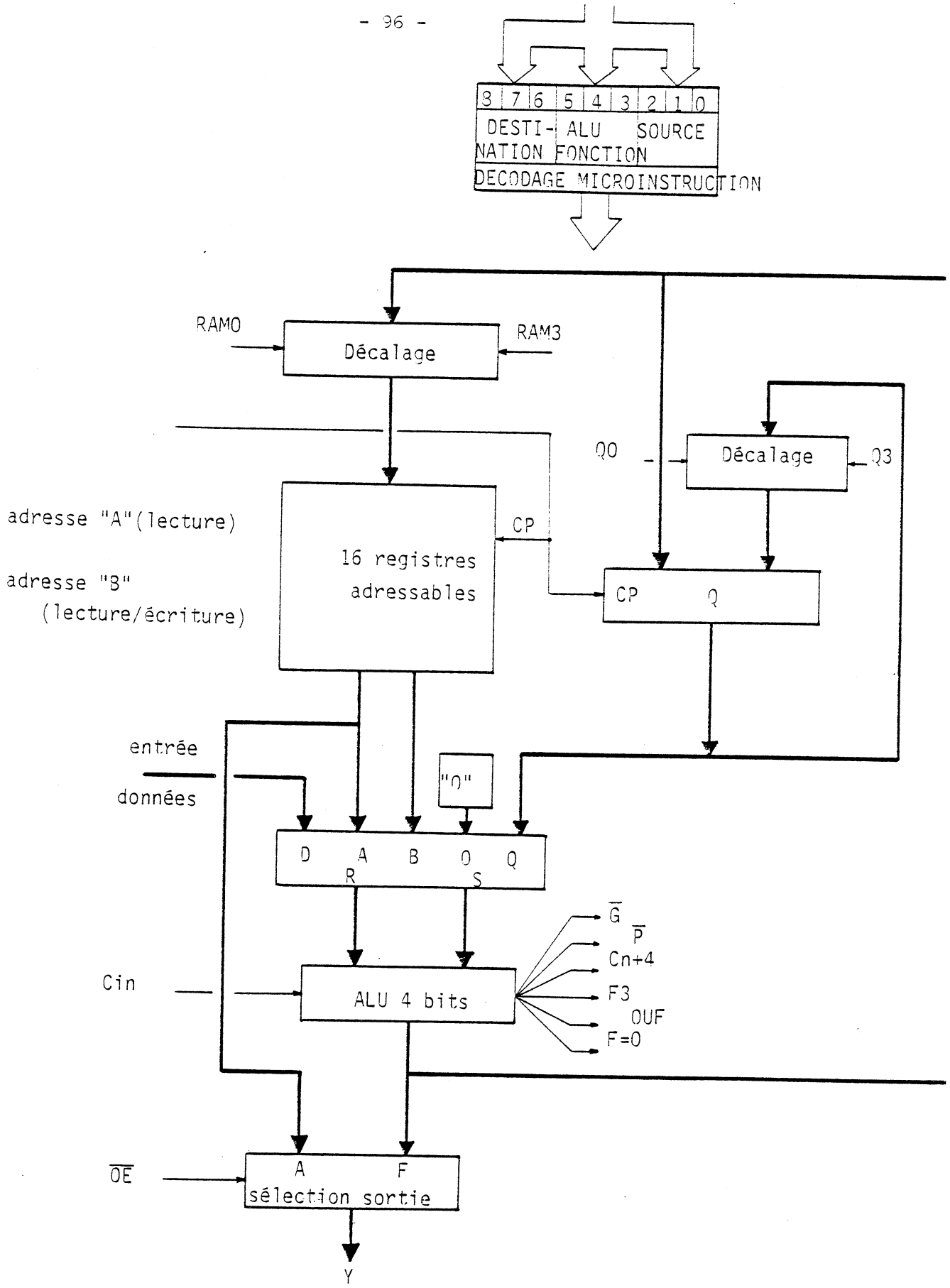


Figure III.9 - Synoptique du 2901.

Cet opérateur a été choisi pour la gestion des pointeurs.
R.FORTIER [5] a montré par ses mesures sur de nombreux programmes écrits en PASCAL que 16 opérandes consécutifs dans une expression mise sous forme postfixée est un maximum (d'ailleurs jamais atteint).
Une taille de 16 mots pour FILEVAL semble être tout à fait raisonnable.
Ces mesures ont aussi montré que de nombreux programmes n'utilisent pas plus de 16 variables au total, ce qui signifie que toutes ces variables se trouveront dans FILEDEP (si elle contient 16 mots) après leur initialisation.
En d'autres termes, pendant toute la durée d'exécution de ces programmes, chaque variable ne sera accédée qu'une seule fois en mémoire centrale.

Les fournisseurs de circuits intégrés proposent de petites mémoires RAM de 16 mots de 4 bits cascadables dans les deux sens. Elles sont adressables par 4 bits, ce qui est compatible avec la largeur du chemin de données de l'opérateur Am2901.
FILEVAL et FILEDEP seront donc réalisées avec 8 boîtiers chacune, agencés de manière à faire des RAM de 16 mots de 32 bits (32 étant la largeur des descripteurs manipulés par PASCHLL).

Remarque
FILEDEP est en fait divisée en deux parties:

- la partie RAM contenant les descripteurs (ou les pointeurs) de largeur 32 bits,

- la partie associative, adressable par son contenu, de 11 bits (10 bits pour le nom interne de la variable + l'indicateur RESOLU) partagée entre PAC et FILE réalisée par d'autres types de composants et située sur la carte du processeur PAC (pour des raisons de nombre de fils en fond de panier).

Le composant Am 2909 regroupe dans un boîtier:

- un multiplexeur entre 4 sources possibles:

  - une entrée directe D
  - un registre interne R chargeable à partir d'une entrée externe (AR)
  - une pile de profondeur 4 chargeable à partir d'un incrémenteur sur la sortie du composant,
  - la sortie de cet incrémenteur.

- un opérateur OU logique à deux entrées sur chaque bit de sortie du multiplexeur, l'autre entrée étant une broche externe du circuit (ORi).

- un opérateur ET logique permettant de mettre à zéro toutes les sorties du circuit en positionnant un niveau bas sur l'entrée Zéro du composant.

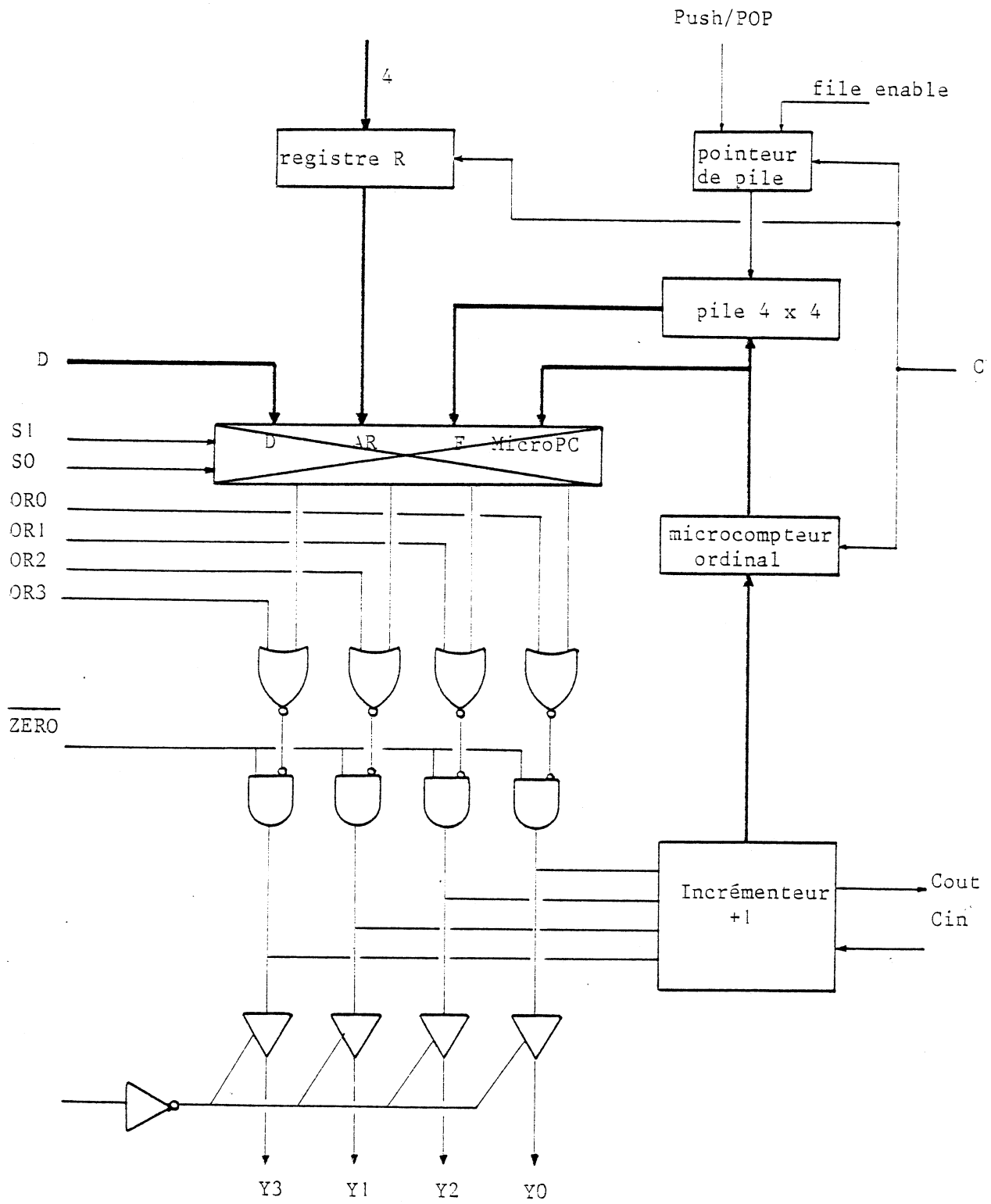


Figure III.10 - Synoptique du 2909

L'étude de cette famille de composants permet de connaître ce qui peut être fait à chaque cycle machine et donc de décomposer les algorithmes en fonction de l'horlogerie. Elle en montre aussi les limites. Ces "microprocesseurs en tranches" résultent du découpage en tranches horizontales de largeur 4 bits d'une Unité centrale classique et il est tentant de réaliser un processeur universel en les juxtaposant, puis de la particulariser pour notre problème par microprogrammation.
Cette méthode de travail n'est pas satisfaisante car une machine universelle sera forcément inadaptée à un problème particulier soigneusement défini par un cahier des charges précis.
Il existe une autre manière d'aborder ces macrocomposants en les considérant comme l'assemblage de circuits MSI (Medium Stage Integration) pouvant réduire le nombre de boîtiers nécessaires à la réalisation physique d'une partie de la machine , même si toute la puissance de calcul offerte n'est pas utilisée.

III.3.2. Réalisation de la partie opérative

III.3.2.1. gestion des mémoires et opérations pointeurs

Les FIFO (FILEVAL et FILEDEP) sont réalisées par des mémoires RAM ordinaires de 16 mots adressées circulairement à l'aide de 6 pointeurs stockés dans les registres d'une tranche opérative Am2901 capables d'exécuter toutes les opérations et comparaisons nécessaires à leur gestion.

- opérations nécessaires à la gestion des pointeurs

  - incrémentation

    X <- X+1

  - décrémentation

    X <- X-1

  - addition avec une donnée externe

    X <- X+D

  - soustraction avec une donnée externe

    X <- X-D

  - transfert avec ou sans incrémentation/décrémentation/ addition/soustraction

    X1 <- X2
    X1 <- X2+1
    X1 <- X2-1
    X1 <- X2+X3
    X2 <- X2-X3

- comparaisons

(X1 = X2) ?
(X1 = X2+1) ?
(X1 > X2) ?

- initialisation

X1 <- "0"

Remarques
Les opérations arithmétiques et les transferts se font en un cycle tandis que les comparaisons demandent 2 cycles:

1/ l'Am 2901 exécute une soustraction,

2/ on teste soit l'indicatteur F=0 (égalité), soit le signe du résultat (relation d'ordre).

L'initialisation à zéro de la valeur d'un pointeur peut être réalisée par une soustraction:

X1 <- "0" <=> X1 <- (X1-X1)

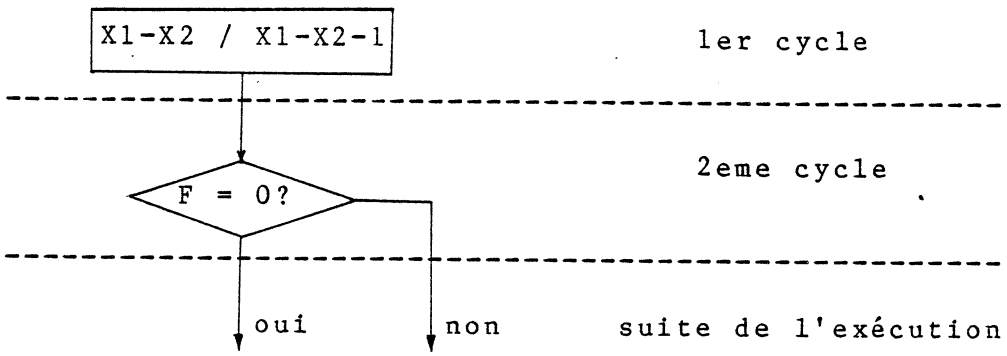
- problèmes posés par la comparaison module 16

La vérification de l'égalité entre deux pointeurs (ou entre le pointeur X1 et le pointeur X2 incrémenté de 1) est simple:

- dans un premier cycle une soustraction est programmée

X1-X2 (ou X1-X2-1)

A la fin de ce cycle la sortie F=0 est mémorisée dans une bascule d'état testée au cours du cycle suivant.



Le fait de vérifier que la valeur d'un pointeur est supérieure à la valeur d'un autre pointeur (et ceci modulo 16) pose plus de problème.
Il faut en effet tenir compte de la position relative de ces deux pointeurs :

(sont-ils ou non dans le même tour?)

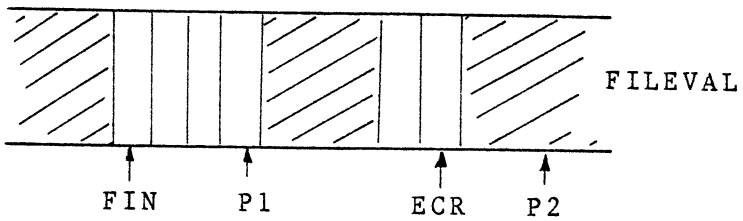
pour déterminer le sens de la soustraction à effectuer :

(X1-X2) ou (X2-X1).

Exemple

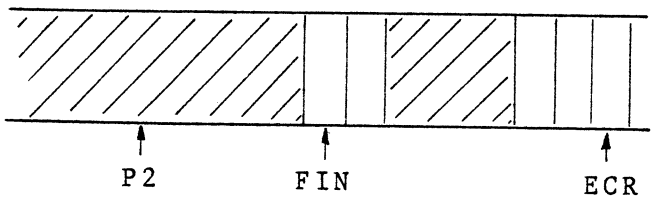
Comparaison P2 > ECR

1er cas : P2 et ECR sont dans le même tour :



Si (P2-ECR) est positif, alors P2 est en avance sur ECR.

2eme cas : P2 et ECR ne sont pas dans le même tour:



Si (ECR-P2) est positif alors P2 est en avance sur ECR.

Le pointeur P2 est le seul à faire des bonds en avant de n pas (n>1) et la seule relation d'ordre à vérifier est entre P2 et ECR.

On doit disposer d'un indicateur donnant la position relative de ces deux pointeurs (même tour ou non).

Le changement de tour pour l'un de ces pointeurs se traduit au niveau de l'opérateur par la génération d'une retenue sortante Cout lors d'une opération d'incrémentation ou d'addition avec une donnée externe.

La solution retenue utilise une bascule T (en pratique une bascule D montée en diviseur par deux) changeant d'état à chaque occurrence d'une retenue sortante quand la tranche opérative exécute une opération du type

ECR <- ECR+1 ou P2 < P2+n.

La partie contrôle de FILE teste la valeur de cette bascule pour

déterminer le sens de la soustraction.

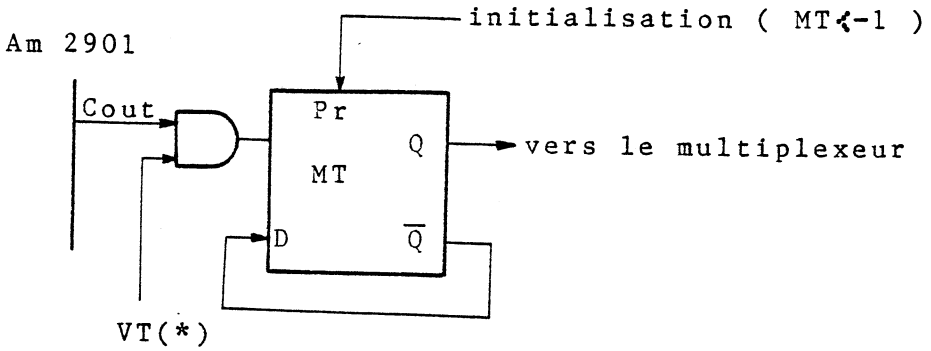


Figure III.11 - Bascule de "tour".

(*) le signal VT permet de valider la commande de la bascule MT. Il est issu de la microinstruction et vaut un pour les opérations susceptibles de faire changer de tour soit le pointeur ECR, soit le pointeur P2.

Le sens de la soustraction étant déterminée, il faut l'effectuer et ensuite tester le signe du résultat.

A l'intérieur de l'opérateur la soustraction A-B s'opère par addition du complément restreint de l'opérande B à l'opérande A. Le signe de la différence est lié à la valeur de la retenue sortante Cout.

Sur les entrées de l'opérateur on a:

$$A = \sum_{i=0}^{3} A_i 2^i \quad \text{et} \quad B = \sum_{i=0}^{3} B_i \, 2^i \; ; \; Cin = 1$$

Le complément restreint de B sera:

$$CR(B) = \sum_{i=0}^{3} \overline{B_i} \; 2^i$$

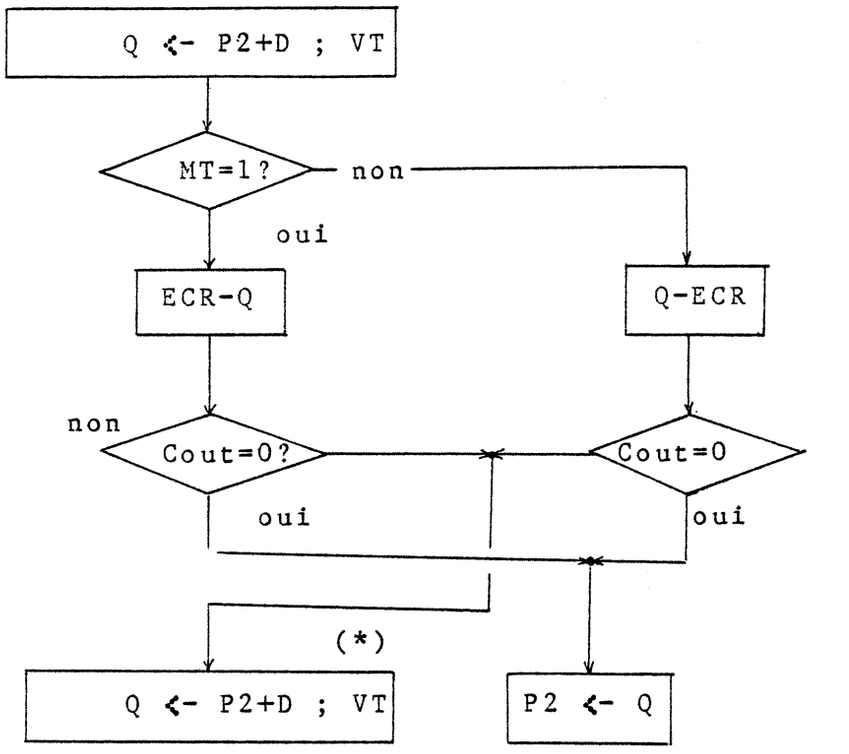
et la différence:

$$A-B = A+CR(B)+2^0$$

et son signe:

$A-B > 0$ si Cout = 1
$A-B < 0$ si Cout = 0

En résumé, le test du dépassement éventuel de P2 incrémenté de n par rapport à ECR sera exécuté de la manière suivante:



(*) Cette opération est nécesssaire pour remettre MT dans son état initial. Si la première occurrence de l'opération n'a pas changé la valeur de MT, la deuxième ne la changera pas non plus. Par contre, si la première fois Cout a été généré, il le sera une nouvelle fois et MT reprendra sa valeur initiale.

- problème posé par le pointeur NIL

La résolution des dépendances conduit à remonter une chaîne de reprise jusqu'à ce que soit trouvé le pointeur vide (NIL).
L'adressage de FILEVAL nécessite 4 bits.
Le champ pointeur des descripteurs de dépendances doit avoir un cinquième bit permettant de valider les quatre autres. Il sera mémorisé dans une bascule NIL à chaque transfert depuis le bus FILE vers le bus pointeur par l'entrée D de l'opérateur.

> NIL = 1 indiquera que les 4 bits valeurs de pointeur n'ont aucune signifiation (on est donc arrivé au dernier maillon de la chaîne de reprises).

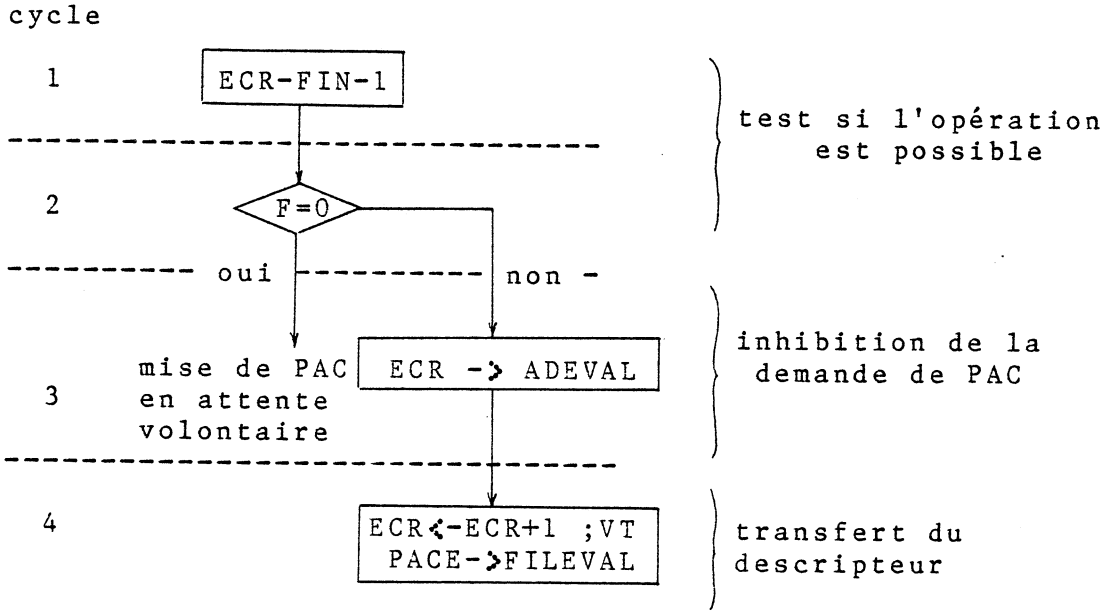
III.3.2.2. Décomposition des algorithmes en fonction de l'horlogerie

L'opérateur Am 2901 étant choisi et le chemin de données étant défini, on peut décomposer les algorithmes à réaliser pour chaque

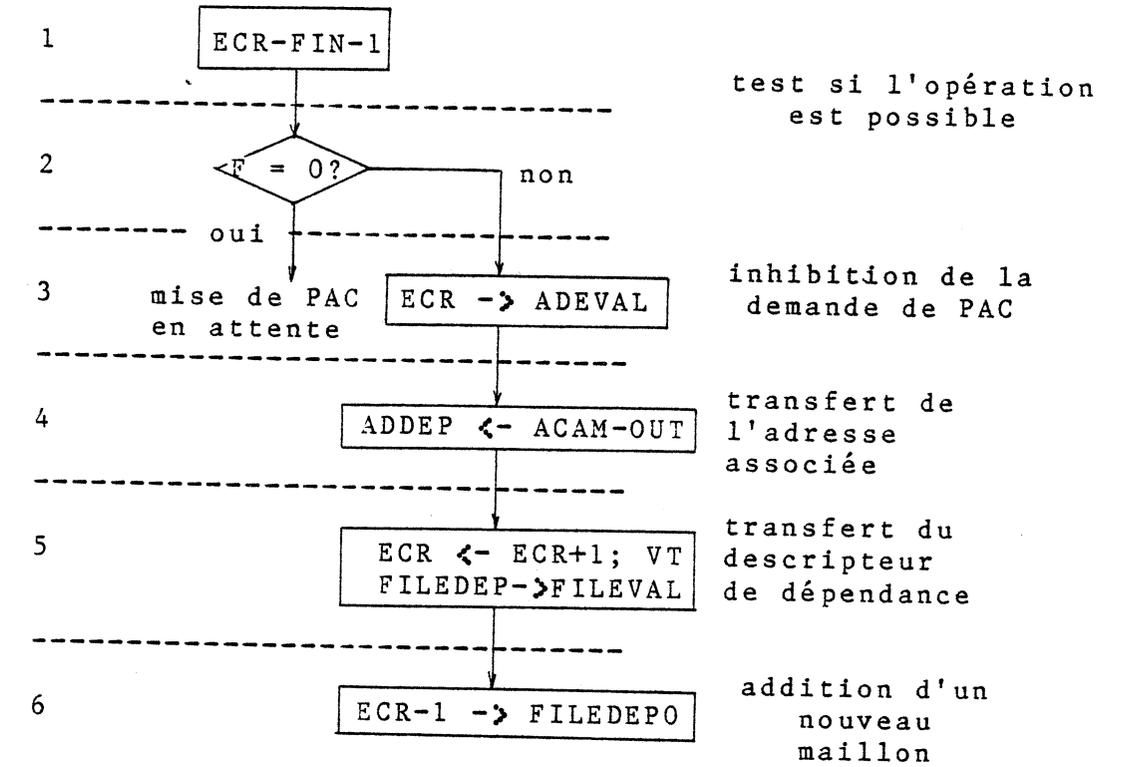
instruction en des séquences de microinstructions élémentaires exécutables en un cycle machine.

- instrutions passées par PAC

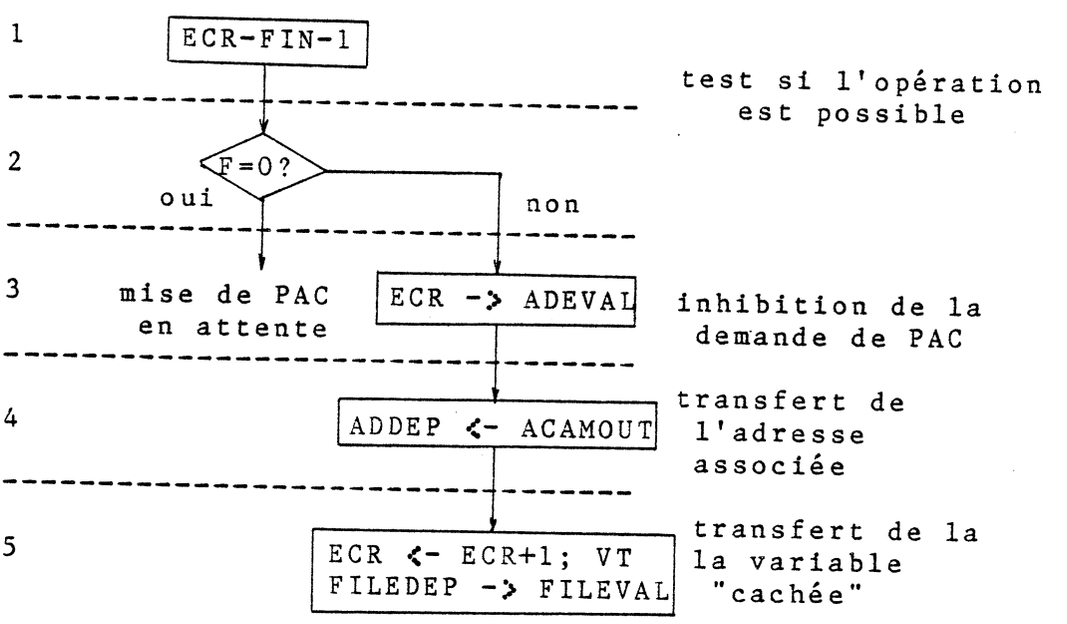
ACC, $\overline{TR}$ Accés à un descripteur se trouvant dans PACE



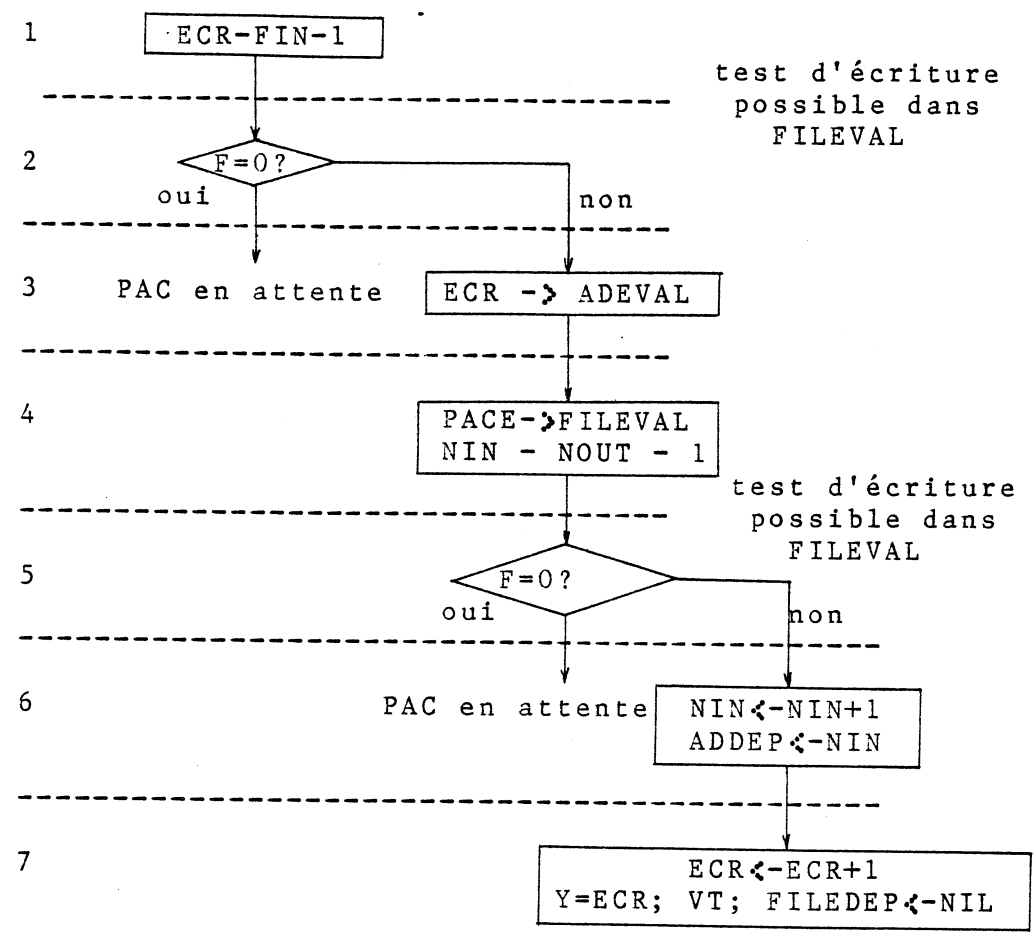
ACC, TR, $\overline{RES}$ construction d'un maillon de chaîne de reprise



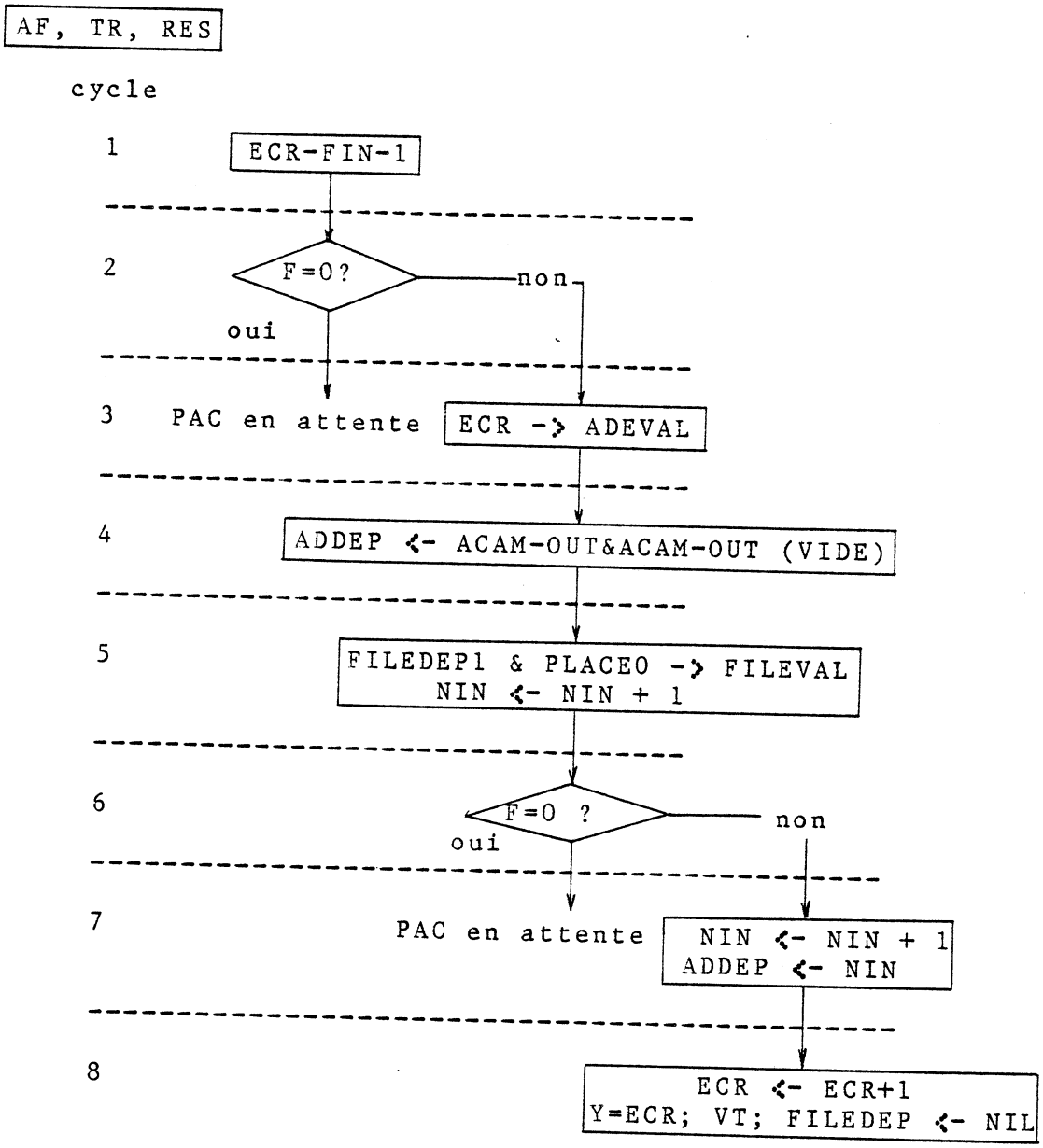
AC, TR, RES Accés à une variable cachée



AF, $\overline{TR}$ Initialisation d'une dépendance



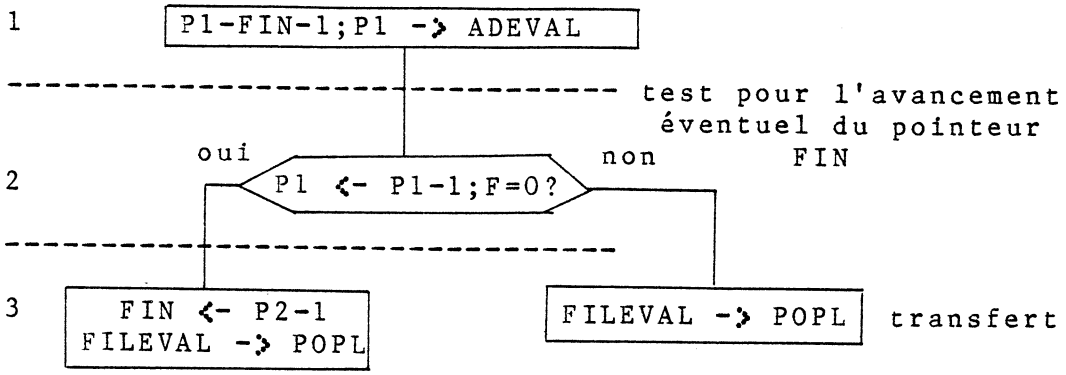
AF, TR, $\overline{RES}$ Mise de PAC en attente

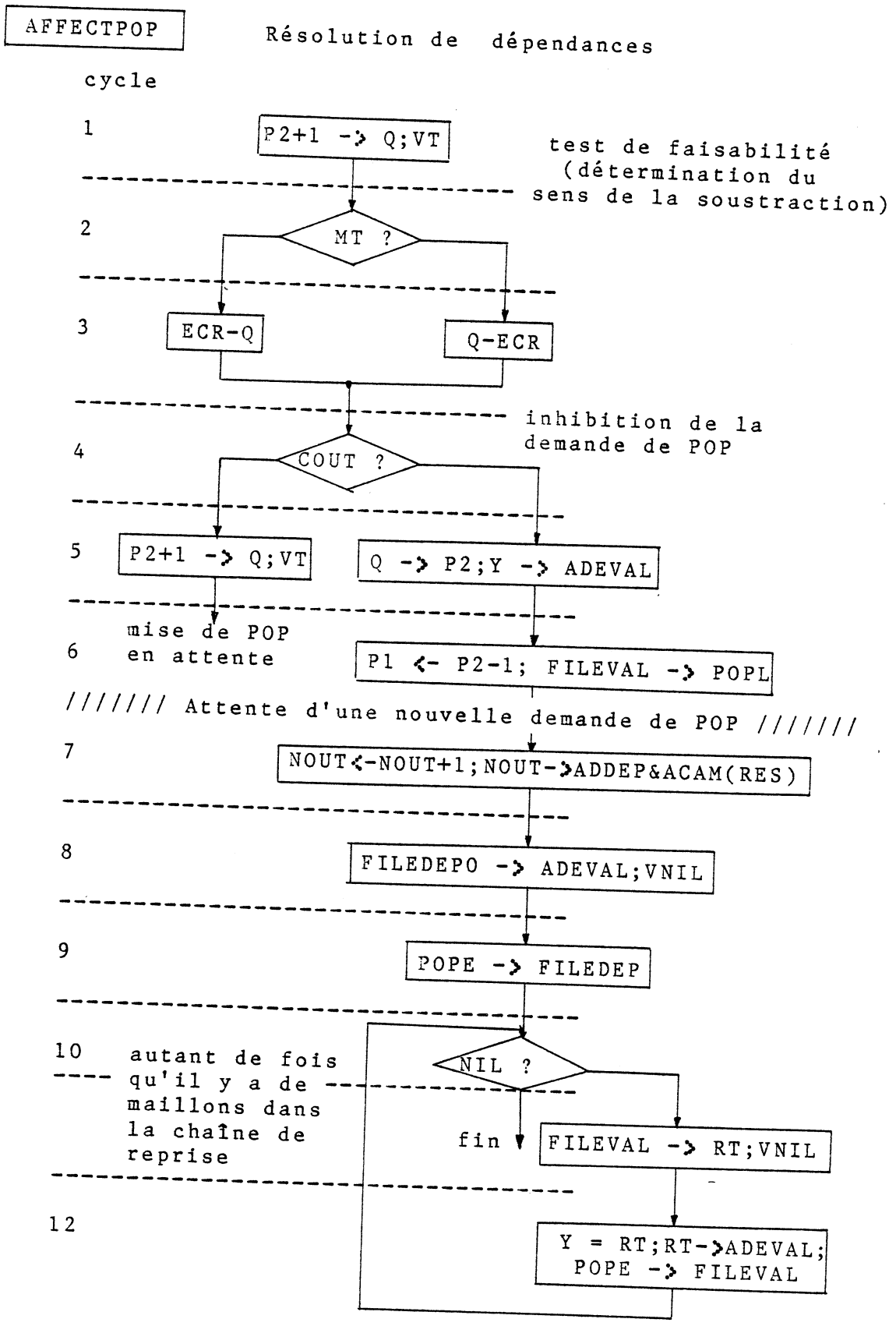


instructions passées par POP:

UNAIRE et CTRL1 : opérateurs et instructions de contrôle monadiques. POP posséde déjà son opérande, FILE n'a rien à faire.

BINAIRE et CTRL2 : opérateurs et instructions de contrôle diadiques (passage d'un opérande).
cycle




AFFECTPOP
Résolution de dépendances
cycle
1
P2+1 -> Q;VT
test de faisabilité (détermination du sens de la soustraction)
2
MT ?
3
ECR-Q
Q-ECR
inhibition de la demande de POP
4
COUT ?
5
P2+1 -> Q;VT
Q -> P2;Y -> ADEVAL
mise de POP en attente
6
P1 <- P2-1; FILEVAL -> POPL
/////// Attente d'une nouvelle demande de POP ///////
7
NOUT<-NOUT+1;NOUT->ADDEP&ACAM(RES)
8
FILEDEPO -> ADEVAL;VNIL
9
POPE -> FILEDEP
10
autant de fois qu'il y a de maillons dans la chaîne de reprise
NIL ?
fin
FILEVAL -> RT;VNIL
12
Y = RT;RT->ADEVAL;
POPE -> FILEVAL

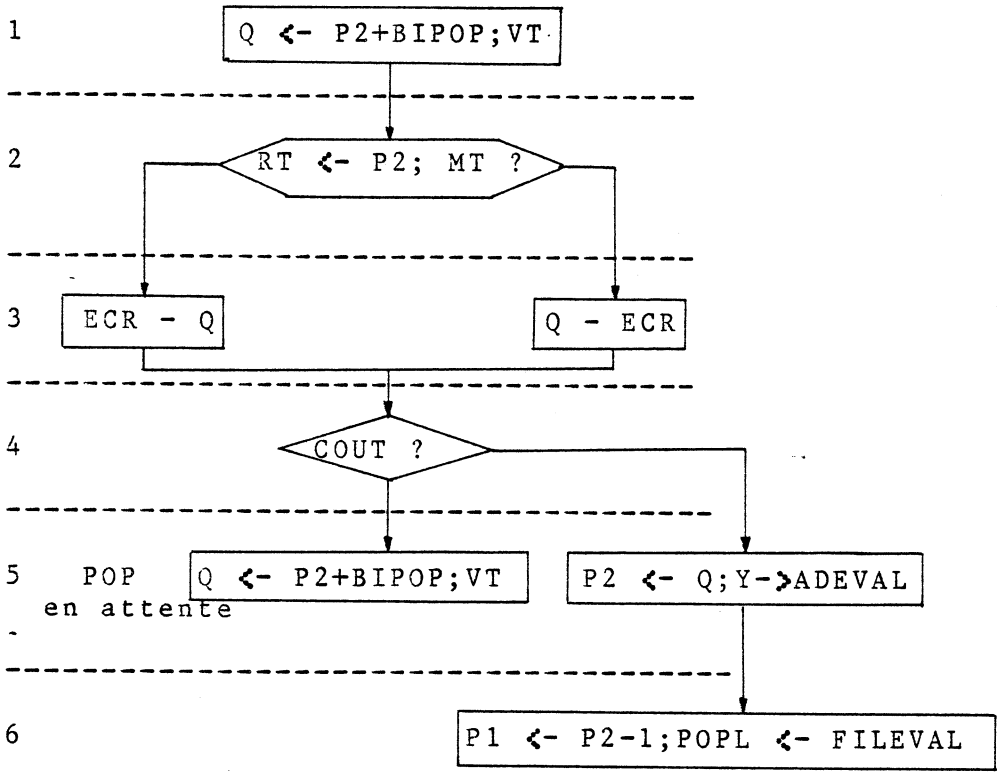
Instructions de mouvements de pointeurs

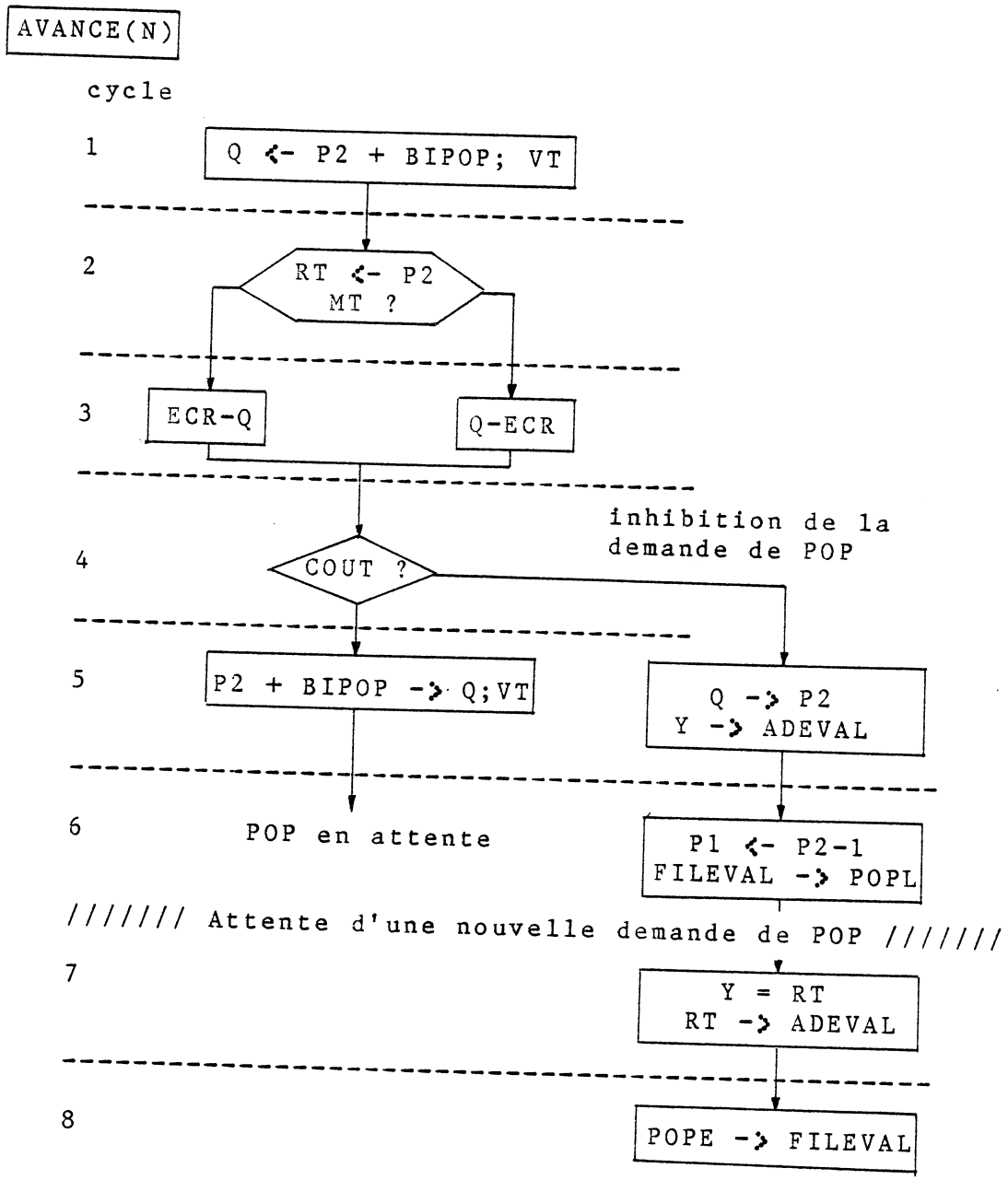
RECUL(N)

1 P1 <- P1-BIPOP

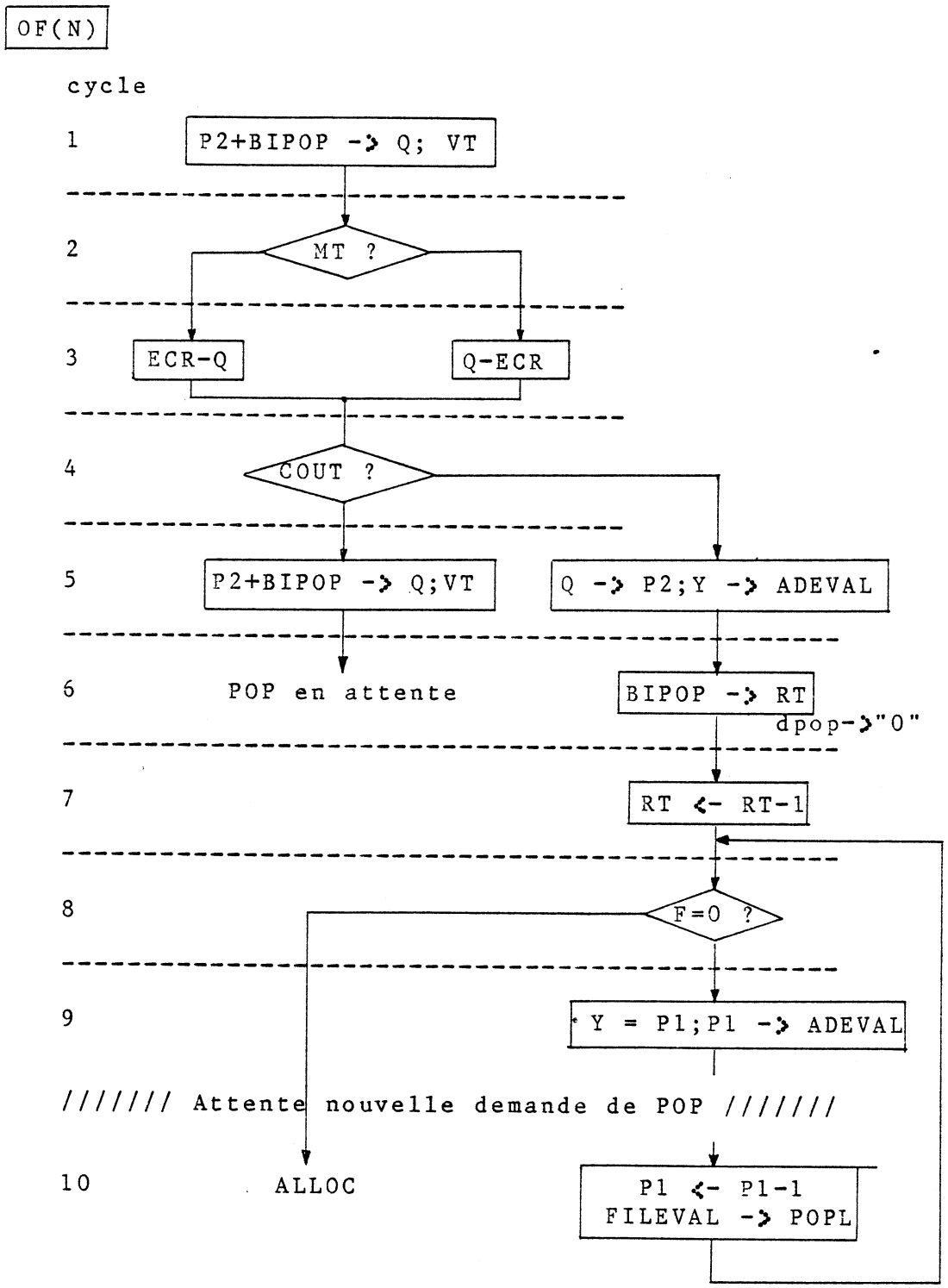
INIT(N)

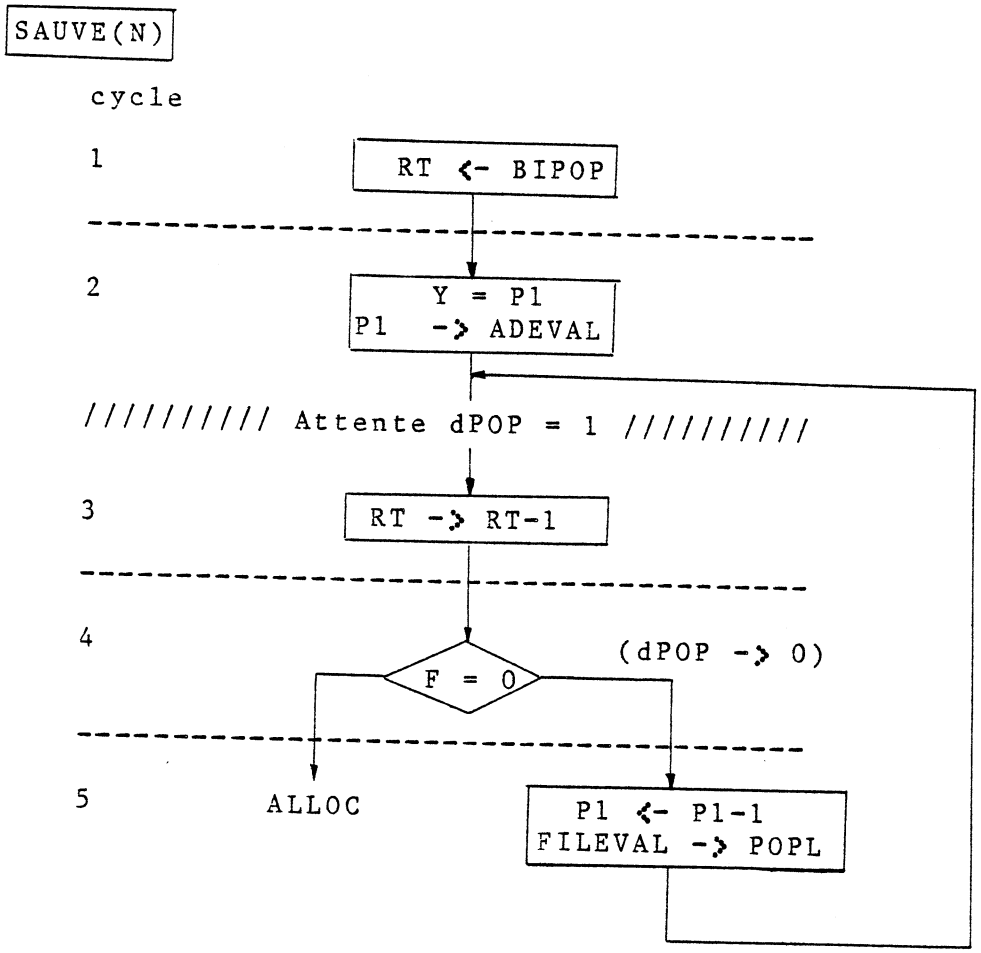
cycle




AVANCE(N)
cycle
1
Q <- P2 + BIPOP; VT
2
RT <- P2
MT ?
3
ECR-Q
Q-ECR
4
COUT ?
inhibition de la
demande de POP
5
P2 + BIPOP -> Q;VT
Q -> P2
Y -> ADEVAL
6
POP en attente
P1 <- P2-1
FILEVAL -> POPL
/////// Attente d'une nouvelle demande de POP ///////
7
Y = RT
RT -> ADEVAL
8
POPE -> FILEVAL

Instructions répétitives





III.2.3. Codage des commandes de la partie opérative Choix des primitives

Le microprogramme symbolique étant défini, le problème du codage des microinstructions en fonction des commandes élémentaires du matériel se pose.

On ne s'intéresse dans ce paragraphe qu'à la partie de la microinstruction relative au chemin de données "pointeurs".

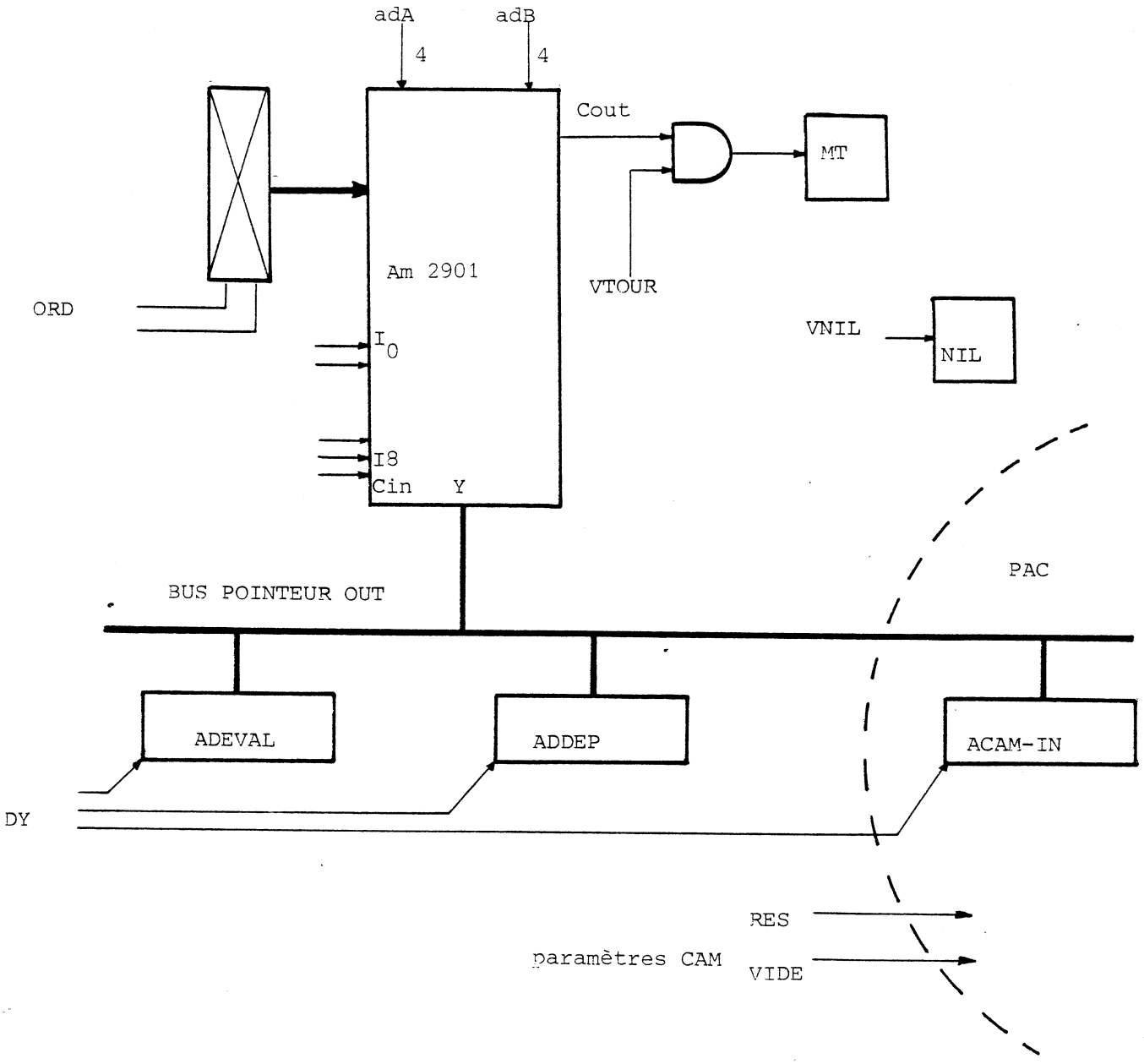


Figure III.12 - Chemin de données pointeur.

Cette partie est décomposable en un certain nombre de champs:

- CODOP : champ de 10 bits (I0...I8,Cin) servant à commander l'opérateur Am2901.

- adA - adB : champs de 4 bits permettant d'adresser les registres internes de l'opérateur.

- ORD : champ de 2 bits servant à sélectionner une source parmi 4 possibles sur le bus d'entrée de l'opérateur.

- DY : champ de 3 bits qui sont les commandes de chargement des registres ADEVAL, ADDEP et ACAM-IN branchés sur le bus de sortie de l'opérateur.

- un champ de 2 bits permettant une écriture de paramètres dans la mémoire associative (VIDE, RESOLU).

- un champ de validation des bascules MT (VTOUR) et NIL (VNIL).

Soit 27 bits différents pour commander l'environnement pointeur. Ces 27 bits peuvent être ramenés à 23. En effet :

- on n'utilise que 7 registres internes de l'opérateur, les champs adA et adB se réduisent à 3 bits chacun.

- Toutes les possibilités de l'opérateur Am2901 ne sont pas utilisées :

  - on ne fait que des opérations arithmétiques (I5 = cst = 0)

  - on n'utilise pas les fonctions de décalage (I8 = cst = 0)

Cette largeur de 23 bits, pour la seule commande de l'environnement pointeur, semble démesurée par rapport à la simplicité et au nombre réduit des opérations à réaliser. Le but de ce paragraphe est d'essayer de trouver des méthodes de réduction de la largeur des microinstructions en les codant de manière appropriée.

Le fait de coder les microinstructions oblige à ajouter une couche supplémentaire de matériel entre le registre de microinstructions et les organes à commander.
Cette couche supplémentaire va avoir une certaine influence sur le temps de cycle du processeur et sur son coût en matériel. Il s'agira donc de trouver un compromis.

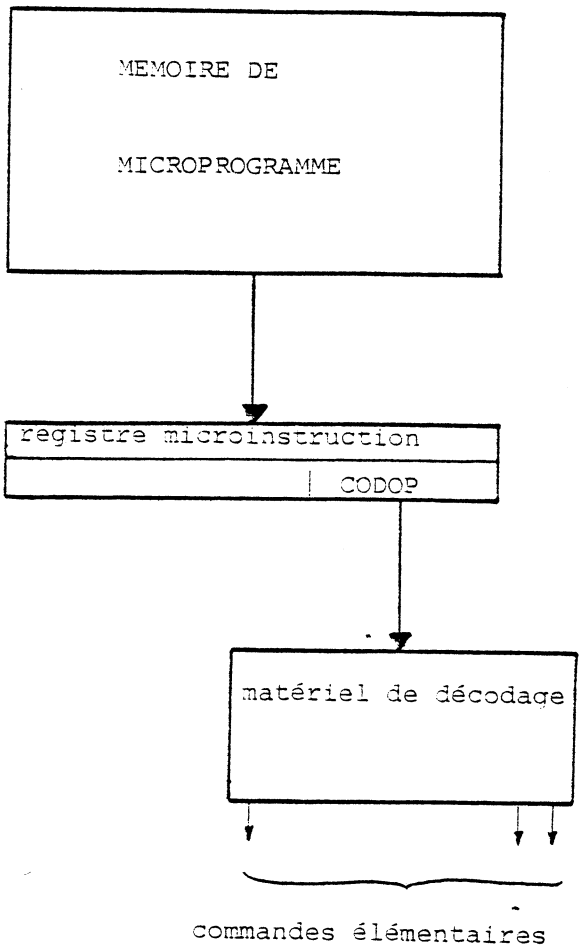


Figure III. 13 - décodage de la microinstruction

Plusieurs possibilités existent pour ce matériel de décodage. Il peut en effet être réalisé à l'aide de circuits logiques SSI classiques, en utilisant autant que faire se peut des circuits un peu plus complexes comme les décodeurs , ou en utilisant une matrice de décodage (PLA ou mémoires mortes).

Les solutions SSI ou MSI impliquent un nombre trop important de circuits.

On trouve chez les constructeurs de circuits intégrés de petites mémoires mortes à temps de lecture très bref (de l'ordre de 40 ns), permettant de réaliser des matrices de décodage performantes, appelées dans la littérature mémoires d'éxécutif (MITRA 15 par exemple).
Leur emploi nécessite la définition d'une nouvelle couche de langage (nanoprogrammation).

Le principe de l'emploi de mémoires d'exécutif et de la nanoprogrammation étant acccepté, le problème du choix des primitives se pose. Ce problème est étroitement lié au codage de la microinstruction. Dans le cas qui nous intéresse, il s'agit de coder les 27 bits nécessaires à la commande de l'environnement pointeur.

On peut tout d'abord remarquer que le fait d'avoir les 27 bits (ou les 23 après une optimisation immédiate) dans le registre microinstruction, correspond à un choix de primitives à la sémantique très pauvre.

On peut en effet dire, de façon schématique, pour la commande de l'opérateur, que les bits I3 à I5 qui codent l'opération à réaliser par l'unité arithmétique et logique interne à l'opérateur, définissent la primitive. Les autres bits étant les paramètres nécessaires à son exécution:

- I0 à I2 indiquent les origines des entrées R et S tandis que

- I6 à I9 permettent de savoir la destination du résultat de l'opérateur.

Ainsi la microinstruction symbolique

ECR <- ECR+1

pourrait être mise sous la forme

R + S; (RAM(A),0; Cin=1; RAM(B) <- F)

avec RAM(A) = ECR et RAM(B) = ECR.

Pour augmenter la puissance des primitives , on peut considérer le champ CODOP tout entier et donc définir des fonctions élémentaires de la forme:

RAM(B) <- RAM(A)+1 ou Q <- RAM(B))+D... etc.

Les champs adA et adB deviennent des paramètres pour ces primitives.

Si l'on fait l'inventaire de toutes les opérations de ce type demandées à l'opérateur 2901, on obtient la liste suivante:

1 RAM(B) <- RAM(A) - RAM(B) ; Y=F

2 RAM(B) <- RAM(A)+1 ; Y=F

3 Q <- RAM(A) ; Y=F

4 Q <- Q - RAM(A) ; Y=F

5 Y = RAM(A)+1

6 RAM(B) <- Q ; Y=F

7 RAM(B) <- RAM(A)-1 ; Y=F

8 Q <- RAM(A)+D ; Y=F

9 RAM(A)-RAM(B)-1 ; Y=F

10 RAM(B) <- D ; Y=F

11 Y = RAM(A)

12 RAM(B) <- RAM(A)+1 ; Y=RAM(A)

13 Y = D

14 RAM(A)-RAM(B)-1 ; Y=RAM(A)

15 RAM(B) <- RAM(B)-D ; Y=F

On voit que 15 fonctions différentes sont demandées à l'opérateur.
C'est-à-dire que 4 bits suffisent pour les coder.
L'emploi d'une matrice de décodage de 15 lignes de 10 colonnes permet d'économiser une largeur de 6 bits dans toutes les microinstructions.

Les plus petites mémoires mortes disponibles dans le commerce possèdent 32 mots, on peut esssayer de mieux les remplir en augmentant la puissance sémantique des primitives.
Pour ce faire, on peut se rapprocher des microinstructions symboliques, en nommant de manière explicite les variables utilisées, ce qui revient à considérer les champs adA et adB comme faisant partie de la primitive. C'est-à-dire définir des fonctions du type

ECR <- ECR+1 ou bien Q <- P2+D

Faisons l'inventaire de toutes les fonctions de ce type utilisées par le processeur FILE:

1 P2 <- P2 - P2

2 ECR <- P2+1

3 FIN <- FIN - FIN

4 NIN <- NIN - NIN

5 ECR - Q

6 Q - ECR

7 Q <- P2+1

8 P2 <- Q

9 P1 <- P2-1

10 Q <- P2+D

11 NIN - NOUT - 1

12 NIN <- NIN - 1

13 RT <- D

14 Y <- RT

15 NIN <- NIN+1

16 ECR <- ECR+1

17 Y <- NIN-1

18 Y <- D

19 ECR - FIN - 1

20 Y <- ECR

21 Y <- ECR - 1

22 P1 - FIN - 1 et Y <- P1

23 P1 <- P1-1

24 NOUT <- NOUT+1

25 RT <- RT-1

26 RT <- P2

27 P1 <- P1-D

On obtient donc 27 primitives codables sur 5 bits. La mémoire de nanoprogramme associée doit contenir 27 mots de 18 bits (14 après optimisation).

Deux boîtiers de ROM (32x8) permettent donc une économie de 9 bits sur la largeur des microinstructions et du registre de microinstructions.

Ces primitives auront pour paramètres:

- l'origine sur l'entrée D,

- la destination du bus pointeur avec éventuellement les paramètres d'écriture dans la mémoire associative,

- les signaux de validation des bascules de tour (MT) et NIL.

On peut essayer d'aller plus loin dans cette approche en considérant comme primitive toute opération sur l'environnement pointeur, c'est-à-dire en ne faisant subsister dans la microinstruction qu'un numéro d'opération sur les pointeurs sans aucun autre paramètre.

On obtient 31 opérations différentes comme le montre le tableau suivant:

1 P2 <- P2-P2

2 ECR <- P2+1

3 FIN <- FIN-FIN

4 NIN <- NIN-NIN

5 ECR - Q

6 Q - ECR

7 Q <- P2+1 ; VTOUR

8 P2 <- Q ; ADEVAL <- Y

9 P1 <- P2-1

10 Q <- P2+BIPOP ; VTOUR

11 NIN - NOUT - 1

12 NIN <- NIN - 1 ; ACAM <- Y ; VIDE

13 RT <- FILEVALO

```
14   ADEVAL <- RT
15   NIN    <-  NIN+1 ; ADDEP <- NIN
16   ECR    <-  ECR+1 ; VTOUR
17   ACAM - IN <- NIN - 1 ; VIDE, RESOL
18   ADDEP <-  ACAM - OUT
19   ECR - FIN - 1
20   ADEVAL <- ECR
21   ADDEP <-ACAM - OUT ; ACAM-IN <- ACAM-OUT ; VIDE
22   P1 - FIN - 1 ; ADEVAL <- P1
23   P1     <-  P1 - 1
24   FIN    <-  P2 - 1
25   NOUT   <-  NOUT + 1 ; ADDEP <- Y ; ACAM - IN <- Y ; RESOL
26   ADEVAL <- FILEDEPO ; VNIL
27   RT     <-  BIPOP
28   RT     <-  RT - 1
29   RT     <-  P2
30   P1     <-  P1 - D ; D=BIPOP
31   Y      <-  ECR - 1
```

Soit 31 opérations différentes pouvant être codées par un numéro sur 5 bits.

Le temps de maintien des entrées de l'opérateur 2901 pouvant être nul, le registre microinstruction sera chargé sur le front montant de l'horloge de base CP.
Avec la solution adoptée consistant à utiliser des boîtiers AM 2918 pour le registre microinstructions, des boîtiers 74S488 pour les PROM de décodage et des boîtiers 74S154 pour le multiplexeur de l'entrée D, on peut évaluer les temps de stabilisation pour les différentes entrées de l'opérateur.

Si on appelle:

- T1 le temps de propagation (clock to output) pour RmicrI
- T2 le temps de lecture des PROM (address to output)
- T3 le temps de propagation (select to output) pour le multiplexeur,

on obtient:

- pour le temps de stabilisation des entrées A,B,I,Cin :

$$T1 + T2 = 13+40 = 53ns,$$

- pour le temps de stabilisation des entrées :

$$D = T1 + T2 + T3 = 13+40+23 = 76ns.$$

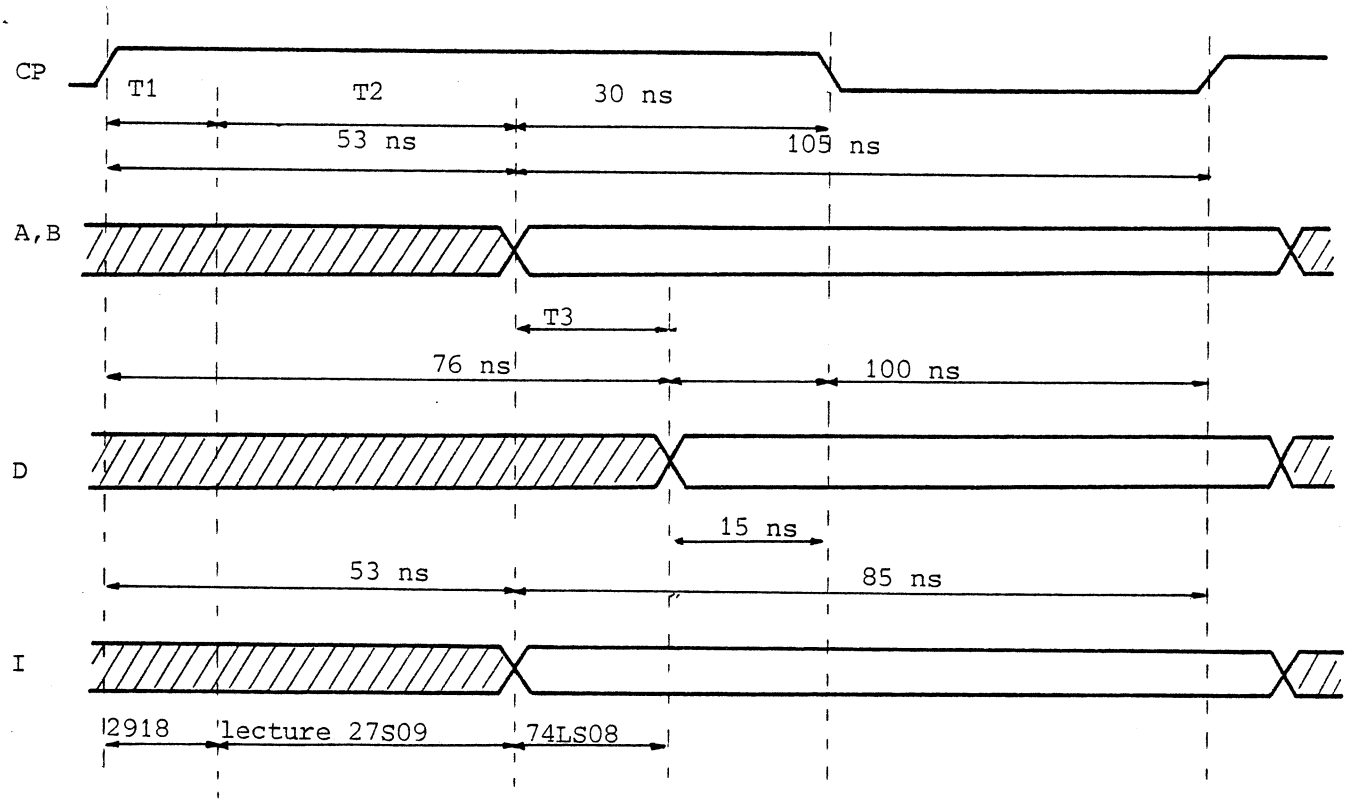


Figure III.15 - Diagramme des temps pour un microcycle.

Influence d'un tel codage sur les performances de la machine

Il est évident qu'il faut ajouter au temps de cycle minimal donné par le constructeur, le temps de lecture des petites PROM de décodage.

Si on regarde les chronogrammes du 2901, on peut remarquer un certain nombre de choses:

- les entrées A et B de l'opérateur doivent être stables au moins 30ns avant la descente du signal d'horloge et au moins 105ns avant sa remontée.

- les entrées D doivent être stables au moins 100ns avant la remontée de CP.

-les entrées commandes I doivent être stables au moins 15ns avant la descente du signal et au moins 85ns avant sa remontée.

- pour la retenue entrante CIN, ce temps se ramène à 15ns.

- les temps de maintien après le front montant des horloges peut être nul.

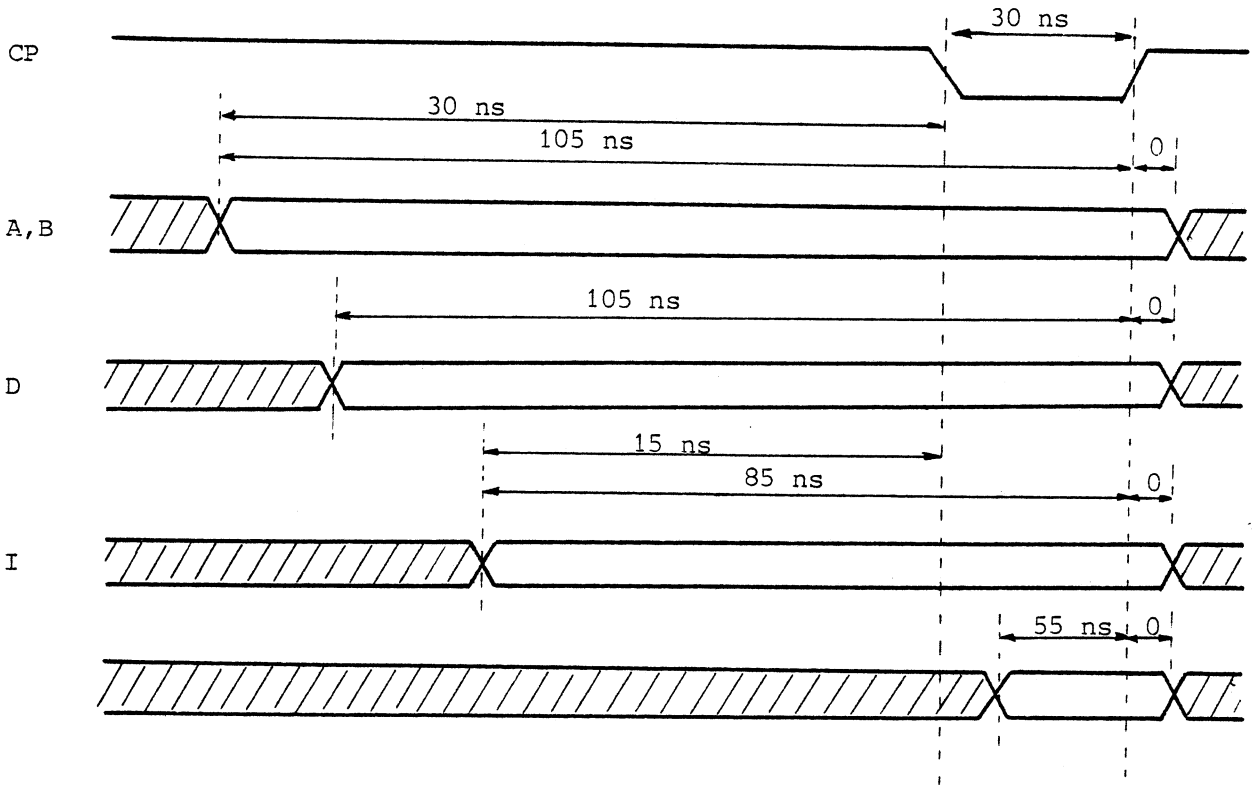


Figure III.16 -Diagramme des temps pour un microcycle.

On va s'intéresser aux temps hachurés sur le chronogramme précédent, c'est-à-dire aux temps de stabilisation des données.

On constate que le temps de cycle minimum que l'on peut espérer est de 176ns.

Avec la solution de la microprogrammation horizontale, c'est-à-dire le registre microinstructions contenant toutes les commandes de l'opérateur, on aurait pu tourner avec un temps de cycle de:

T1 + T3 + 100 = 136ns.

On peut trouver un compromis pour gagner une vingtaine de nanosecondes en faisant raccourcir les temps T2 et T3, c'est-à-dire en sortant le sous-champ origine D de la mémoire d'exécutif et en le faisant apparaître dans la microinstruction.
Cette solution augmente de 2 bits la largeur de toutes les microinstructions.

Le temps de cycle minimal sera alors:

Max [(T1+T2+105), (T1+T3+100)] = 13+40+105 = 158ns.

On pourrait aussi sortir les champs adA et adB (augmentation de 6 bits de la largeur de la microinstruction)
Le temps de cycle minimal serait alors de :

Max [(T1+105) , (T1+T2+85) , (T1+T3+100)]

c'est à dire 136 nanosecondes, temps que l'on ne peut pas réduire.

Un compromis devra être trouvé au vu des performances globales escomptées pour la machine PASC-HLL. Il faudra tenir compte des temps de cycle minimaux des autres processeurs.

III.3.3. Réalisation de la partie contrôle

III.3.3.1. Fonctionnement du séquenceur Am 2909

Parallèlement à l'exécution de la partie opérative de la microinstruction, les séquenceurs Am 2909 calculent l'adresse de la microinstruction suivante en fonction du code présent sur leurs entrées de commande S0, S1, FE , PUP et Cin.

S0 et S1 sélectionne la source de l'adresse de la microinstruction suivante:

S1	S0	source
0	0	compteur de microcyle (micrPC)
0	1	registre AR
1	0	sommet de la pile interne (STK0)
1	1	entrée directe D

FE et Pup codent les opérations de modification de la pile.

FE	Pup	
1	x	pas de changement
0	1	incrémentation du pointeur de pile, stockage de la valeur courante du microcompteur ordinal dans STK0
0	0	décrémentation du pointeur de pile (dépilage)

La retenue entrante quand elle est au niveau haut permet à l'incrémenteur de jouer son rôle et de stocker la valeur de l'adresse de la microinstruction courante augmentée de 1 dans micrPC.

Le fait de la forcer à zero empêche cette incrémentation et donc permet le bouclage sur une microinstruction.

Ce phénomène est utilisé pour les attentes:
pour cela on relie l'entrée Cin à la sortie d'un multiplexeur d'attente validé par un bit de la microinstruction.
Tant que l'évènement attendu n'est pas présent on empêche l'incrémentation du micrPC.
Si l'ordre de séquencement "continue" a été sélectionné (c'est à dire si micrPC est l'adresse de la microinstruction suivante) on boucle sur la même microinstruction.
Cependant si l'on regarde les diagrammes temporels du séquenceur on s'aperçoit que l'incrémentation de micrPC se fait en tout début de cycle, c'est-à-dire avant que la condition n'ait eu le temps de s'établir.

L'utilisation de l'entrée Cin pour lever les attentes, demande d'avoir positionné Cin à la microinstruction qui précède.

Pour avoir plus de souplesse on a préféré utiliser une autre technique.
On interpose un registre adresse des microinstructioons à la sortie du séquenceur. Ce registre est chargé en début de cycle (front montant de CP) avec l'adresse de la microinstruction courante.

Si c'est une microinstruction d'attente, il faut empêcher le chargement du registre jusqu'à ce qu'elle soit résolue et sélectionner ce registre comme adresse de la microinstruction suivante.
Le multiplexage entre l'adresse générée par le Am 2909 et le registre se fait en utilisant les commandes OE des deux boîtiers.

Non seulement cette méthode facilite l'écriture et la compréhension du microprogramme, mais elle donne plus de souplesse:
on peut en effet faire une attente quelle que soit la source de l'adresse de la microinstruction suivante.

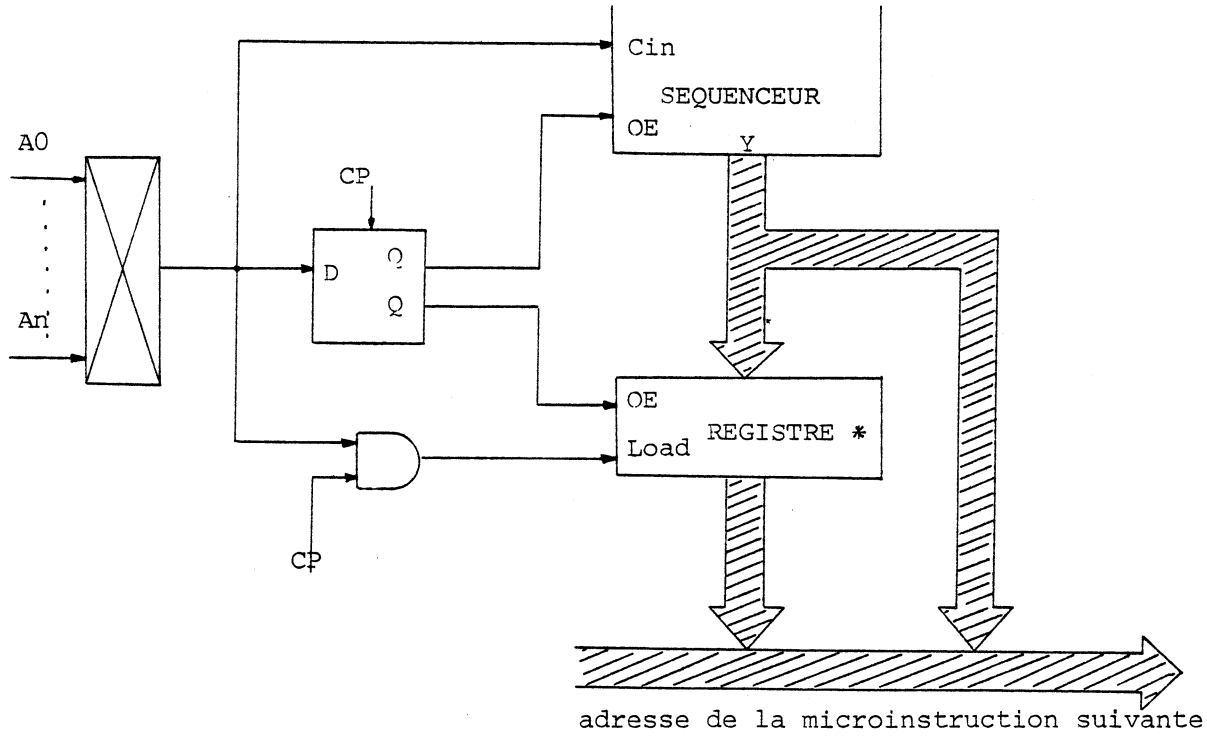


Figure III.16 - Réalisation des attentes.

Deux conditions d'attente sont utilisées par FILE:

- (VIDE) testée avant d'aller lire BIPOP

- DPOP pour la synchronisation entre POP et FILE lors d'une sauvegarde de FILEVAL par exemple.

III.3.3.2. Branchements multidirectionnels

Les séquenceurs Am 2909 possèdent des entrées ORi permettant le forçage à "1" du bit numéro i de l'adresse de la microinstruction suivante quelle qu'en soit sa source (microcompteur, registre AR, entrée D ou sommet de pile STKO).

Moyennant certaines contraintes d'implantation du microprogramme, ces entrées peuvent être utilisées pour des branchements multidirectionnels.

microinstructions conditionnelles

L'entrée OR1 des séquenceurs est utilisée pour des brancheeents à deux directions suivant la valeur d'une condition.

Soit ad l'adresse de la microinstruction conditionnelle sélectionnée par l'ordre arrivant sur SO, S1, FE et Pup.

Suivant la valeur de la condition présente sur l'entrée OR1, la prochaine microinstruction à exécuter est à l'adresse:

ad si la condition est fausse (OR1=0)

ad+2 si la condition est vraie (OR1=1)

Les conditions testées par le microprogramme de FILE sont au nombre de 7:

F=0 F3 sorties de l'opérateur Am 2901 mémorisées à la fin du cycle précédent et utilisées pour les comparaisons de pointeur.

MT sortie d'une bascule MT (même tour) permettant de réaliser des comparaisons modulo 16.

PT bascule "pas terminé" qui indique au processeur FILE que l'algorithme relatif à l'instruction précédente venant de POP n'est pas terminée (voir chap. Allocation).

NIL utilisé pour la résolution des dépendances (voir microprogramme correspondant à l'instruction AFFECTPOP).

BIPAC 10 bit venant du processeur PAC qui permet de distinguer une affectation d'un passage de paramètres dans la classe AFFECT.

VRAI condition câblée à "1" (toujours vraie) permettant le forçage de OR1. Cette condition est utilisée pour des problèmes d'implantation des microinstructions.

Les numéros de ces conditions sont codés sur trois bits de la microinstruction et sont câblés en entrée sélection d'un multiplexeur 8 vers 1 (74 LS 151) dont la sortie Y est reliée à l'entrée OR1.

allocation

Le processeur FILE est une ressource partagée entre les processeurs PAC et POP.

Ceux-ci, quand ils en ont besoin, positionnent une demande qui sera prise en compte par FILE sous certaines conditions:

- il n'y a pas de demande plus prioritaire,
- la demande n'est pas masquée.

L'allocation du processeur FILE se traduit par un branchement multidirentionnel dans son microprogramme:
les entrées OR0, OR6 et OR7 de son séquenceur sont utilisées à cette fin. Elles sont validées par microprogramme pendant le cycle d'allocation (DEM).

La fin de toute séquence de microprogramme correspondant à l'algorithme de traitement d'une instruction pour POP ou pour PAC se traduit par un ordre de séquencement "branchement à l'adresse ALLOC et validation de OR0, OR6 et OR7"

(JMPR(ALLLOC),DEM).

OR0 est utilisée pour le forçage à l'adresse

TROMPE = (ALLOC+1)

qui est le point d'entrée de l'algorithme de remise à jour des files d'EVALuation et des DEPendances.

On a vu que PINS fait un choix quand il rencontre une instruction conditionnelle ou de boucle et commence à envoyer à BIPOP et BIPAC les instructions correspondant à l'alternative choisie.
Il anticipe donc sur la valeur du prédicat qui sera évalué par POP. Deux cas sont possibles:

- bon choix: l'état conditionnel de PINS est levé et le traitement continue.
- mauvais choix: POP positionne une bascule TROMPE pour en avertir PINS qui doit faire vider BIPOP et BIPAC, se brancher à l'adresse correspondant à l'autre alternative et avertir FILE qui doit remettre à jour FILEVAL et FILEDEP.

C'est la sortie de cette bascule qui arrive sur OR0: le miroprogramme de FILE est dans sa phase d'allocation. De plus, elle force OR6 et OR7 à zéro en masquant les demandes de POP et de PAC.

OR6 reçoit la demande de PAC pendant le cycle d'allocation, si cette demande n'est pas masquée.

Outre le masquage de la demande de PAC par le signal TROMPE, FILE a pu mettre PAC en attente volontaire s'il a pris trop d'avance par rapport à POP: par exemple une écriture dans FILEVAL est impossible parce que le pointeur ECR a rattrapé le pointeur FIN.

OR7 reçoit la demande de POP pendant le cyle d'allocation si celle-ci n'est pas masquée.

Le masquage de la demande de POP peut se faire

- soit par le signal TROMPE,

- soit par la mise en attente volontaire de POP. (Si par exemple l'instruction AVANCE(N) ne peut être exécutée du fait qu'après incrémentation le pointeur P2 dépasse le pointeur ECR.

Une priorité entre les demandes de POP et de PAC a été établie: POP est le processeur le plus prioritaire.

Les états de masquage des demandes (mise en attente volontaire) sont mémorisés dans deux bascules ATVPAC et ATVPOP.

Ces deux bascules sont mises à "1" soit

- par microordres de FILE,

- soit par la sortie de la bascule TROMPE,

Leur remise à zero se fait par microordres.

Les valeurs des entrées OR0, OR6 et OR7 des séquenceurs Am 2909 en fonction de l'état des demandes et des bascules ATVPOP, ATVPAC et TROMPE sont données dans la table de vérité suivante:

DPAC	DPOP	ATVPAC	ATVPOP	TROMPE	DEM*	OR0	OR6	OR7
x	x	x	x	x	1	0	0	0
0	0	0	0	0	0	0	0	0
x	x	x	x	1	0	1	0	0
0	1	x	0	0	0	0	0	1
1	1	x	0	0	0	0	0	1
1	0	0	x	0	0	0	1	0
1	1	0	1	0	0	0	1	0
1	0	1	0	0	0	0	0	0

*DEM est le signal de validation du cycle d'allocation.

Pour réaliser ces trois fonctions (OR0, OR6, OR7) on a choisi d'utiliser deux boîtiers double multiplexeur 4 vers 1. Ils seront validés par le signal DEM.

Le premier reçoit sur ses quatre entrées, soit:

- le couple DPAC, DPOP
- le couple 0, DPOP (PAC est en attente volontaire)
- le couple DPAC, 0 (POP est en attente volontaire)
- le couple 0, 0 (TROMPE a été positionné à "1").

La sélection d'un de ces quatre couples (entrées A et B du multiplexeur) se fait par les sorties des bascules ATVPAC et ATVPOP:
C'est donc le multiplexeur de masquage des demandes.

Le deuxième multiplexeur sert pour la mutuelle exclusion entre les demandes de PAC et de POP (priorité) et pour le forçage de l'adresse TROMPE (début de l'algorithme de remise jour des files).

On obtient le câblage suivant:

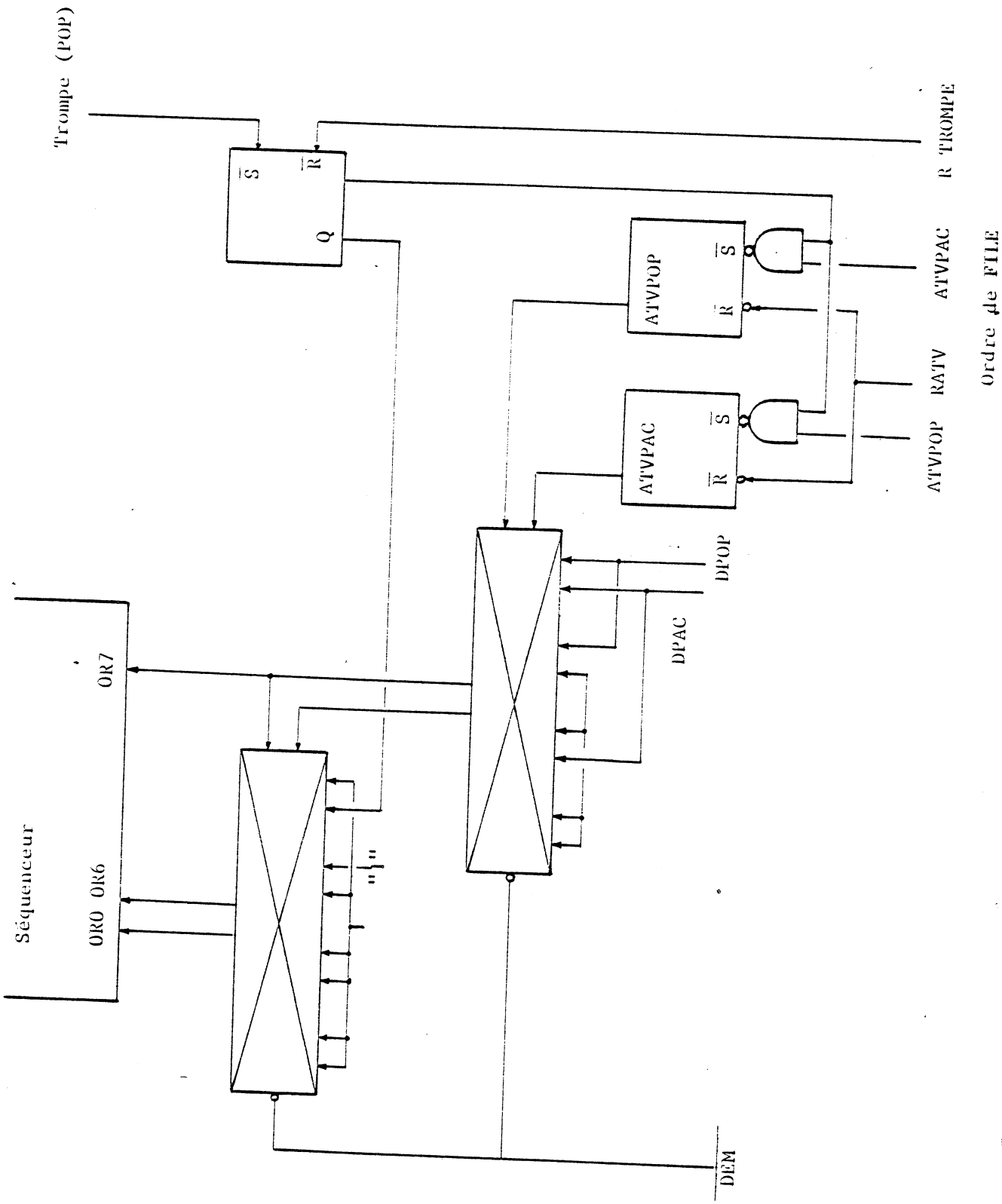


Figure III.17 : FILE : Allocateur.

Remarques à propos de l'allocation de POP

On a vu que FILE avait un rôle de préparation des opérandes pour POP (introduction d'un registre RIPOP).

Cependant certaines instructions comme AVANCE(N) ou AFFECTPOP demandent une sauvegarde d'un résultat intermédiaire ou définitif dans FILEVAL, sauvegarde qui ne peut évidemment avoir lieu qu'à la fin de l'exécution de l'opérateur traité par POP.
Cet algorithme d'exécution peut être relativement long par rapport à l'algorithme de FILE correspondant à l'instruction suivante. Pour ne pas pénaliser PAC, on décompose l'exécution de ces instructions en deux phases:

- la première phase peut être exécutée immédiatement.

- la deuxième phase qui ne sera exécutée que lorsque POP aura terminé son travail.

Entre temps, FILE pourra éventuellement ranger de nouveaux opérandes passés par PAC s'il en manifeste la demande.
Pour ce faire, une bascule PT (Pas Terminée) a été introduite. Elle est mise à "1" en fin de première phase par mircordre.
Quand POP a fini l'exécution de son instruction il demande la suivante et positionne sa requête DPOP. Quand FILE la prendra en compte, il testera PT:

* "PT=1" lui indique qu'avant l'exécution de l'instruction venant de BIPOP il doit terminer l'algorithme relatif à l'instruction précédente (sauvegarde d'un résultat dans FILEVAL).

Prenons par exemple l'instruction AVANCE(N):

FILE doit dans la première phase

- tester si après addition de N au pointeur P2, celui-ci ne dépasse pas le pointeur ECR,

- réaliser cette addition,

- fournir à POP l'opérande se trouvant en FILEVAL(P2).

Pendant la deuxième phase, il doit sauvegarder dans l'ancien FILE(P2) le résultat que lui communique POP à travers son buffer d'écriture POPE. Ceci se traduit par le microprogramme suivant :

Adresses	microinstruction et commentaire
AVANCE	P2+D → Q, D=BIPOP, VT
AVANCE+1	RT ← P2 Si MT JSR(TOURN)
	/stockage de P2 dans RT pour la deuxième phase de l'instructio
AVANCE+2	ordre PT ← 1, JSR(ALLOC)
	/mise à "1" de la bascule "pas terminé" et branchement au cycle d'allocation avec empilage de l'adresse de retour pour finir la séquence de microinstruction avant l'exécution de l'algorithme relatif à l'opérateur suivant/
AVANCE+3	Y=RT, RT → ADEVAL, Attendre ($\overline{\text{VIDE}}$)
AVANCE+4	POPE ← FILEVAL, JMPD(BIPOP)
	/ces deux microinstructions constituent la deuxième phase: rangement dans FILEVAL du résultat évalué par POP/
TOURN	ECR - Q
	/MT=0, ECR et P2 sont dans le même tour/
TOURN+1	Si F3 JMPR(P2+N)
TOURN+2	Q-ECR
	/MT=1, ECR et P2 ne sont pas dans le même tour/
TOURN+3	Si F3 JMPR(P2+N)
P2+N	P2+BIPOP → Q, VT, ordre(ATVPOP), JMPR, D(ALLOC)
	/F3=0 ⇒ P2 dépasse ECR, on réexécute l'opération précédente (P2+BIPOP) pour le cas où elle aurait entraîné un changement d'état de MT de façon à se replacer dans l'état initial(2 basculements sont équivalents à aucun changement) et on met PAC en attente volontaire avant de revenir au cycle d'allocation/

(P2+N)+2	Q → P2, Y → ADEVAL, ordre(DPOP ← 0)
(P2+N)+3	P1 ← P2-1, FILEVAL → POPL, RTS
	/il n'y a pas eu de dépassement: on passe à POP l'opérande nécessaire à l'exécution de l'opérateur suivant/

III.3.3 3 Utilisation de l'entrée D

L'entrée D du séquenceur sert exclusivement à donner l'adresse de la première microinstruction d'une séquence correspondant à une macroinstruction ou plus exactement, une classe de macroinstruction venant de POP ou de PAC.

Les classes d'instruction venant de POP sont:

- contrôle unaire (CTRL1)
- opérateur unaire (UNAIRE)
- contrôle binaire (CTRL2)
- opérateur binaire (BINAIRE)
- appel et retour de procédures externes (RET/ENT)
- affectation correspondant à AFFECT(S,D) (l'affectation correspondant à AFFECT sans paramètre étant un opérateur binaire)(AFFECTPOP)
- initialisation d'une sauvegarde (INITSAUVE(N))
- sauvegarde (SAUVE(N))
- comparaisons multiplles (OF(N))
- positionnement de P2 sans sauvegarde de résultat (INIT(N))
- positionnement de P2 avec sauvegarde d'un résultat (AVANCE(N))
- positionnement de P1 (RECUL(N)).

Soit 12 classes d'instructions qui sont codées sur les 4 bits poids fort de BIPOP, les quatre autres contenant éventuellement un paramètre (N pour les 6 dernières) envoyés via un multiplexeur sur les entrées D0 à D3 de l'opérateur Am 2901.

De PAC arrive deux grandes classes d'instructions "ACCES" et "AFFECT" paramétrées par des informations venant de la mèmoire associative de PAC (indicateurs TROUVE et RESOLU).

PAC envoie donc à FILE un mot de 3 bits appelé CODEPAC et contenant:

- 1 bit de BIPAC permettant de différencier les deux classes d'instructions,
- l'indicateur TROUVE (venant de la CAM)
- le bit RESOLU (venant de la CAM).

Les entrées D des séquenceurs Am 2909 sont câblées de la façon suivante:

- D0, D1 et D2 sont reliées à la masse,

- D3, D4, D5 et D6 reçoivent la sortie d'un quadruple multiplexeur 4 vers 1 permettant l'aiguillage entre les 4 bits poids fort de BIPOP ou le mot constituté par les 3 bits de CODEPAC pour le poids faible et un "1" pour le poids fort.

L'entrée "sélection" de ce multiplexeur vient de la partie contrôle du microprogramme et est aussi câblée sur l'entrée D7 des séquenceurs. On obtient donc l'implantation suivante pour les points d'entrée des algorithmes, spécifique à chacune des instructions passées par POP ou PAC.

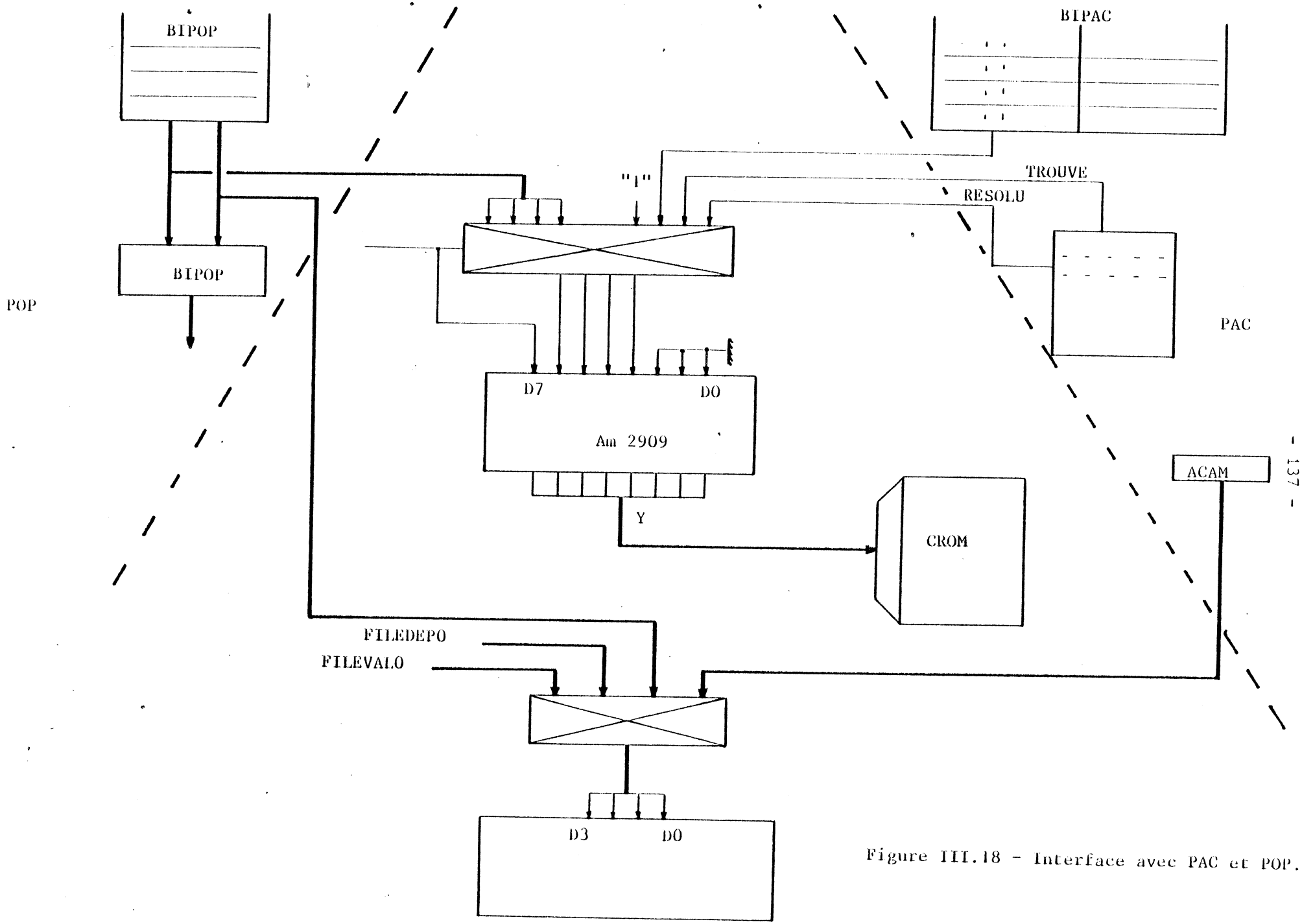


Figure III.18 - Interface avec PAC et POP.

adresse en binaire			(décimal)	instruction	
0	1001	000	(72)	AFFECT, TR	pour les instructions venant de PAC
0	1010	000	(80)	AFFECT, TR, RES	
0	1011	000	(88)	AFFECT, TR, RES	
0	1101	000	(104)	ACCES, TR	
0	1110	000	(112)	ACCES, TR, RES	
0	1111	000	(120)	ACCES, TR, RES	
1	0001	000	(128)	CTRL1	pour les instructions venant de POP
1	0011	000	(152)	UNAIRE	
1	1011	000	(168)	CTRL2	
1	0111	000	(184)	BINAIRE	
1	1000	000	(192)	RET/ENT	
1	1001	000	(200)	AFFECTPOP	
1	1010	000	(208)	INITSAUVE(N)	
1	1011	000	(216)	OF(N)	
1	1100	000	(224)	INIT(N)	
1	1101	000	(232)	AVANCE(N)	
1	1110	000	(240)	SAUVE(N)	
1	1111	000	(248)	RECUL(N)	

III.3.3.4.Utilisation des entrées Ri

Les entrées ARi du séquenceur reçoivent les adresses des sous-microprogrammes de travail utilisés par le processeur FILE.

Ces adresses sont issues du champ ADBR de la microinstruction. Ce champ n'apparait pas dans la microinstruction, il vient directement de la mémoire de microprogramme sur les entrées Ri. La barrière temporelle sur ce champ est à l'intérieur du séquenceur.
Le registre AR interne aux Am 2909 tient lieu de registre

microinstruction pour ce champ ADBR.

Le microprogramme de FILE utilise 7 sous-programmes nommés :

- TOUR1
- P2+1
- TOURn
- P2+n
- ALLOC
- TESTNIL
- DEP

Il faudra donc trois bits pour les adresser. Le câblage retenu est donné par la figure III.19.

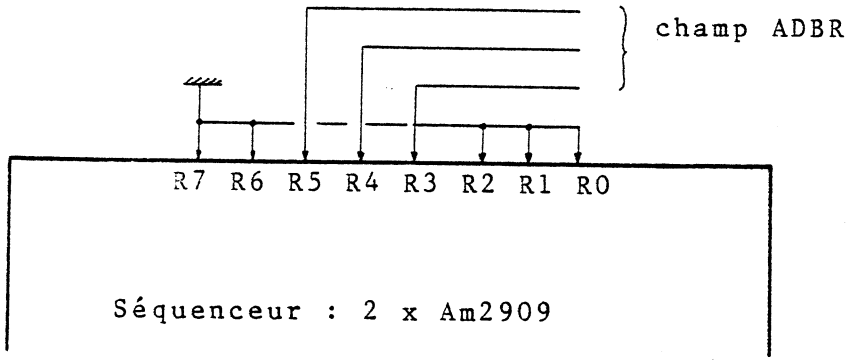


Figure III.19 champ ADBR

Ce qui donne l'implantation suivante :

étiquette	adresse du début du sous-microprogramme
TOUR1	00 001 000
P2+1	00 010 000
TOURn	00 011 000
ALLOC	00 100 000
P2+n	00 101 000
TESTNIL	00 110 000
DEP	00 111 000

III.3.3.5. Séquencement des microinstructions

Les ordres de séquencement utilisés par le microprogramme du processeur FILE sont donnés dans le tableau suivant :

Ordre de séquencement	adresse de la microinstruction cours d'exécution	adresse de la microinstruction suivante	observations
Continue	I	I+1	
Si CONDITION, continue	I	I+1 ou I+3	séquencement conditionne
Attente ($\overline{VIDE}$)	I	I ou I+1	bouclage sur I tant que $\overline{VIDE}$ = "0"
Attente (DPOP)	I	I ou I+1	bouclage sur I tant que DPOP = "0"
JMPR, D(ALLOC)	I	ALLOC ALLOC + 1 ALLOC + 64 ALLOC + 128	voir § Allocation
JMPR(ADBR)	I	ADBR	branchement inconditionnel le registre AR contient la nouvelle adresse
Si CONDITION JMPR(ADBR)	I	ADBR ou ADBR+2	branchement conditionnel
End Loop, $\overline{DEM}$	I	ALLOC ALLOC+1 ALLOC+64 ALLOC+128	décrémentation du pointeu de pile
SMPD(BIPOP)	I	BIPOP	allocation à POP, recherch de l'instruction à exécute l'adresse BIPOP se trouve sur l'entrée D
JMPD(CODEPAC)	I	CODEPAC	allocation à PAC CODEPAC se trouve sur D
JSR (ADBR)	I	ADBR	branchement inconditionnel avec empilage de I+1 au sommet de pile

...../..

ordre de séquencement	adresse de la microinstruction en cours d'exécution	adresse de la microinstruction suivante	observations
Si CONDITION JSR(ADBR)	I	ADBR ou ADBR+2	appel de sous-microprogramme conditionnel avec stockage de l'adresse de retour (I+1) en sommet de pile
Push stack	I	I+1	continue avec stockage de I+1 en sommet de pile (début de boucle)
Stack Ref.	I	STK_0	référence au sommet de pile (boucle)
RTS	I	STK_0	idem avec décrémentation du sommet de pile

Cet ensemble de 15 ordres de séquencement nécessite :

- 5 bits de code pour les séquenceurs (S0, S1, FE, Pup, Cin),
- 1 bit pour valider les demandes (DEM utilisé pendant le cycle d'allocation,
- 1 bit pour valider le multiplexeur de condition,
- 1 bit pour valider le multiplexeur d'attente,
- 1 bit pour sélectionner l'attente,
- 1 bit de sélection du multiplexeur sur l'entrée D (peut être tout le temps validé).

Soit 10 bits auxquel viennent s'ajouter les trois bits de sélection d'une condition parmi 8.

Remarque
Les bits Cin, de validation du multiplexeur de condition, de validation du multiplexeur d'attente et de sélection de l'attente peuvent être codés sur 2 bits seulement (CA0 et CA1). leur décodage est réalisé par un multiplexeur 4 vers 1 comme le montre la figure III.20.

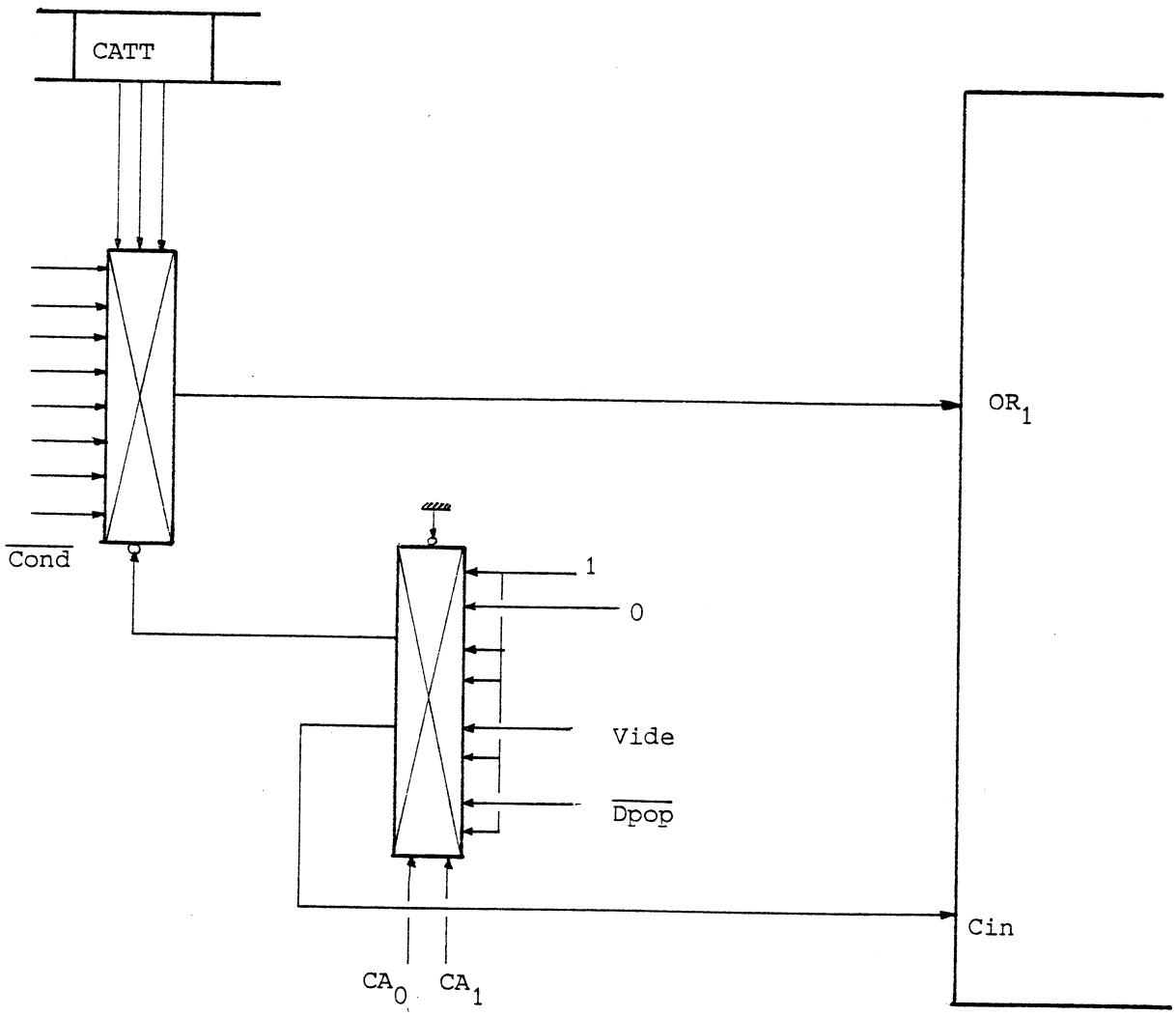


Figure 3.20 Conditions et attentes

codage de la sélection

CA1	CA0	Cin	COND	
0	0	1	0	instruction conditionnelle
0	1	1	1	ni attente, ni condition
1	0	VIDE	1	attente (VIDE)
1	1	DPOP	1	attente (DPOP)

Les 15 opérations de séquencement nécessitent 11 bits de microinstruction (8 bits de commande et 3 bits de sélection d'une condition).

Dans un soucis de minimisation de la largeur des microinstructions, les 15 opérations de séquencement ont été codés sur 4 bits. Là encore l'utilisation d'une matrice de décodage a permis de diminuer de 4 bits la largeur de la microinstruction.

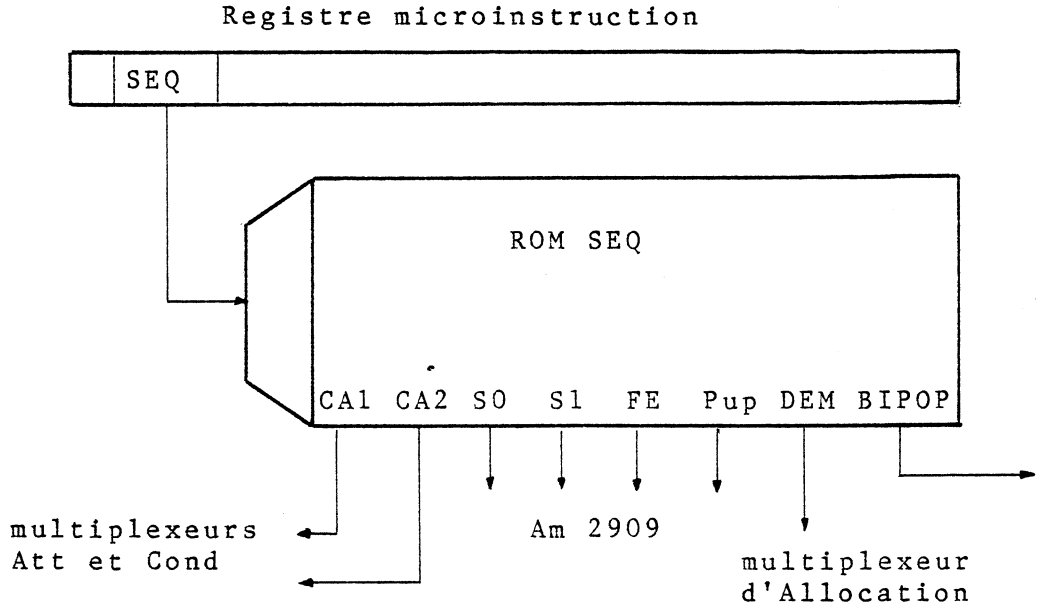
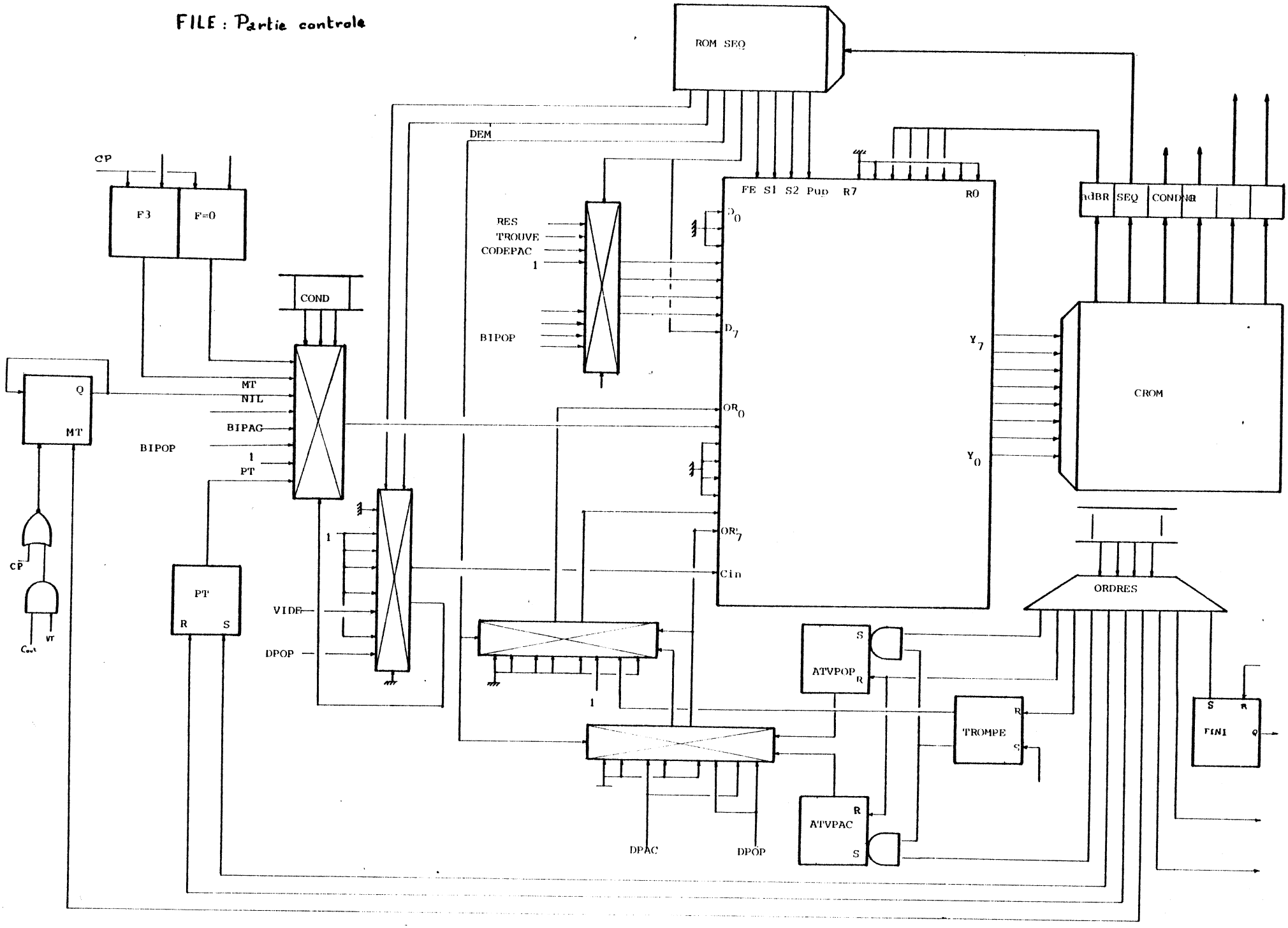


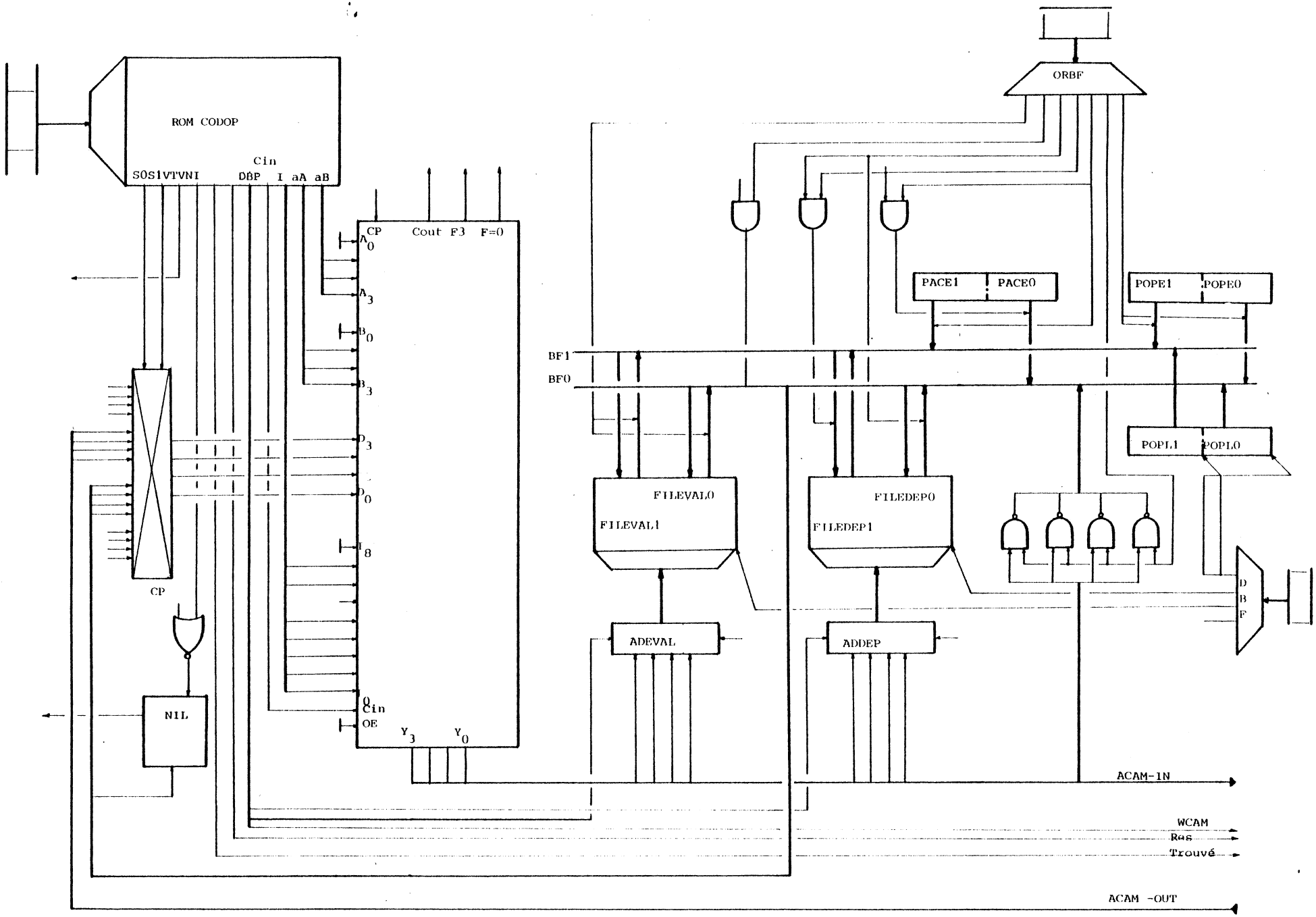
Figure III.21 Décodage des opérations de séquencement.

Schémas du processeur FILE

FILE : Partie controle



FILE : chemin de données



Microprogramme symbolique du processeur FILE

" 107 microinstructions "

adresse	étiquette	commande partie opérative	ORBF	DBF	séquencement	attente	condition	@branch.	ORDRE
0	IPL	P2←P2-P2	-	-	continue	-	-	-	RTROMPE
1		ECR←P2+1	-	-	continue	-	-	-	RATV
2		FIN←FIN-FIN	-	-	continue	-	-	-	DPOP←0
3		NIN←NIN-NIN	-	-	continue	-	-	-	DPAC←0
4		-	-	-	continue	-	-	-	PT←0
5		-	-	-	IMPR,D	-	-	ALLOC	MT←0
6									
7									
8	TOUR1	ECR-Q	-	-	continue	-	-	-	-
9		-	-	-	IMPR	-	C_{n+4}	P2+1	-
A									
B		Q-ECR	-	-	continue	-	-	-	-
C		-	-	-	IMPR	-	C_{n+4}	P2+1	-
D									
E									
F									
10	P2+1	P2+1→Q ; VTOUR	-	-	JMPR, D	-	-	ALLOC	ATVPOP
11									
12		Q→P2, Y→ADEVAL	-	-	continue	-	-	-	DPOP←0
13		P1←P2-1	FILEVAL	POPL	RTS	-	-	-	-
14									
15									
16									
17									
18	TOURN	ECR-Q	-	-	continue	-	-	-	-
19		-	-	-	JMPR	-	C_{n+4}	P2+N	-
1A		Q-ECR	-	-	continue	-	-	-	-
1B		-	-	-	IMPR	-	C_{n+4}	P2+N	-
1C									
1D									
1E									
1F									
20	ALLOC	-	-	-	IMPR, D	-	-	ALLOC	-
21	TROMPE	P2+1→ECR	-	-	Push stack	-	-	-	MT←0
22		NIN-NOUT-1	-	-	continue	-	-	-	RATV
23		-	-	-	continue	-	F=0	-	RTROMPE
24		-	-	-	JMPR,D ;POP stack	-	-	ALLOC	-
25									
26		NIN←NIN-1;ACAM-IN(VIDE)←Y	-	-	Stack Ref	-	-	-	-
27									

...../..

adresse	étiquette	commande partie opérative	ORBF	DBF	séquencement	attente	condition	@ branch	Ordre
28	P2+N	Q←P2+BIPOP ; VTOUR	-	-	IMPR, D	-	-	ALLOC	ATVPOP
29									
2A		P2←Q, Y→ADEVAL	-	-	continue	-	-	-	DPOP←0
2B		P1←P2+1	FILEVAL	POPL	RTS	-	-	-	-
2C									
2D									
2E									
2F									
30	TESTNIL	-	-	-	continue, AH	VIDE	-	-	-
31		-	-	-	IMPD(BIPOP)	-	-	-	-
32		BF_0→RT ; VNIL	$FILEVAL_0$	-	continue	-	-	-	-
33		-	POPE	FILEVAL	continue	-	-	-	-
34		-			IMPR	-	NIL	TESTNIL	-
35									
36									
37									
38	DEP	-	-	-	JMPR, D	-	-	ALLOC	ATVPAC
39									
3A		NIN←NIN+1 ; ADDEP←NIN	-	-	continue	-	-	-	-
3B		ECR←ECR+1 ; Y=ECR, VTOUR	NIL	FILEDEP	continue	-	-	-	DPAC←0
3C		Y=NIN-1, Y→ACAM-IN(VIDE, RESOL)	-	-	JMPR	-	-	ALLOC	-
3D									
3E									
3F									
40									
41									
42									
43									
44									
45									
46									
47									
48	AF,$\overline{TR}$	NIN-NOUT-1	PACE	FILEVAL	continue	-	-	-	-
49		-	-	-	JMPR	-	F=0	DEP	-
4A									
4B									
4C									
4D									
4E									
4F									

..../..

adresse	étiquette	commande partie opérative	ORBF	DBF	séquencement	attente	condition	@branch.	ordre
50	AF,TR,$\overline{RES}$	-	-	-	JMPR, D	-	-	ALLOC	ATVPAC
51									
52									
53									
54									
55									
56									
57									
58	AFF,TR,RES	ADDEP←ACAM-OUT & → ACAM-IN(VIDE)	-	-	continue	-	BIO	-	-
59		NIN-NOUT-1	FILEDEP$_1$ PACE$_0$	FILEVAL	continue	-	-	-	-
5A		-	-	-	JMPR	-	F=0	DEP	-
5B		NIN-NOUT-1	PACE	FILEVAL	continue	-	-	-	-
5C			-	-	JMPR	-	-	DEP	-
5D									
5E									
5F									
60	ALPAC	ECR-FIN-1	-	-	continue	-	(=1)	-	- *
61									
62									
63		-	-	-	continue	-	(F=0)	-	-
64		-	-	-	JMPR(D)	-	-	ALLOC	ATVPAC
65									
66		ADEVAL←ECR	-	-	JMPRD(BIPAC)	-	-	-	RATV
67									
68	ACC,$\overline{TR}$	ECR←ECR+1 ; VTOUR	PACE	FILEVAL	JMPR, D	-	-	ALLOC	DPAC←0
69									
6A									
6B									
6C									
6D									
6E									
6F									
70	ACC,TR,$\overline{RES}$	ADDEP←ACAM-OUT	-	-	continue	-	-	-	-
71		ECR←ECR+1 ; VTOUR	FILEDEP	FILEVAL	continue	-	-	-	DPAC←0
72		Y=ECR-1	Y	FILEDEP$_0$	JMPR,D	-	-	ALLOC	
73									
74									
75									
76									
77		..../..							
		* forçage de OR.							

adresse	étiquette	commande partie opérative	ORBF	DBF	séquencement	attente	condition	@branch.	ordre
78	ACC,TR,RES	ADDEP←ACAM-OUT	-	-	continue	-	-	-	-
79		ECR←ECR+1 ; VTOUR	FILEDEP	FILEVAL	JMPR,D	-	-	ALLOC	DPAC←0
7A									
7B									
7C									
7D									
7E									
7F									
80									
81									
82									
83									
84									
85									
86									
87									
88	CTRL1	-	-	-	JMPR, D	-	-	ALLOC	DPOP←0
89									
8A									
8B									
8C									
8D									
8E									
8F									
90									
91									
92									
93									
94									
95									
96									
97									
98	UNAIRE	-	-	-	JMPR, D	-	-	ALLOC	DPOP←0
99									
9A									
9B									
9C									
9D									
9E									
9F									

...../..

adresse	étiquette	commande partie opérative	ORBF	DBF	séquencement	attente	condition	@branch.	ordre
A0	ALPOP	-	-	-	continue	-	PT=1	-	-
A1		-	-	-	RTS	-	-	-	-
A2									
A3		-	-	-	continue	VIDE	-	-	-
A4		-	-	-	JMP(BIPOP)	-	-	-	-
A5									
A6									
A7									
A8	$CTRL_2$	P1-FIN-1 , ADEVAL←P1	-	-	continue	-	(=1)	-	- *
A9									
AA									
AB		P1←P1-1	-	-	continue	-	F=0	-	DPOP←0
AC		FIN←P2-1	FILEVAL	POPL	JMPR, D	-	⊢	ALLOC	-
AD									
AE		-	FILEVAL	POPL	JMPR, D	-	-	ALLOC	-
AF									
B0									
B1									
B2									
B3									
B4									
B5									
B6									
B7									
B8	BINAIRE	P1-FIN-1	-	-	continue	-	(=1)	-	- *
B9									
BA									
BB		P1←P1-1 ; ADEVAL←P1	-	-	continue	-	F=0	-	DPOP←0
BC		FIN←P2-1	FILEVAL	POPL	JMPR, D	-	-	ALLOC	-
BD									
BE		-	FILEVAL	POPL	JMPR, D	-	-	ALLOC	-
BF									
C0	RET/ENT	-	-	-	JMPR, D	-	-	ALLOC	DPOP←0
C1									
C2									
C3									
C4									
C5									
C6									
C7									

...../..

*forcage de OR_-

adresse	étiquette	commande partie opérative	ORBF	DBF	séquencement	attente	condition	@branch.	ordre
C8									
C9	AFFECTPOP	P2+1→Q, VTOUR	-	-	continue	-	-	-	-
CA		-	-	-	JSR	-	MT	TOUR1	-
CB		-	-	-	JSR	-	-	ALLOC	PT←1
CC		NOUT←NOUT+1;Y→ADDEP & ACAM-IN(RES)	-	-	POP Stack	-	-	-	PT←0
		BF_0→ADEVAL ; VNIL	$FILEDEP_0$	-	continue	-	-	-	-
CD		-							
CE			POPE	FILEDEP	JMPR	-	NIL	TESTNIL	-
CF									
D0	INITSAUV(M)	P2+1→Q , VTOUR							
D1		-	-	-	continue	-	-	-	-
D2		BIPOP→RT	-	-	JSR	-	MT	TOUR1	-
D3		RT←RT-1	-	-	Push stack	-	-	-	DPOP←0
D4		-	-	-	continue	-	-	-	-
D5		Y=P1 ; P1←ADEVAL	-	-	continue	-	F=0	-	-
D6		P1←P1-1	-	-	continue	DPOP=1	-	-	-
D7		-	FILEVAL	POPL	Stack ref	-	-	-	DPOP←0
D8	OFN	P2+BIPOP→Q, VTOUR	-	-	JMPR,D+POP stack	-	-	ALLOC	-
D9		BIPOP→RT	-	-	continue	-	-	-	-
DA		-	-	-	JSR	-	MT	TOURN	-
DB		RT←RT-1	-	-	JSR	-	-	ALLOC	PT←1
DC		-	-	-	continue	-	-	-	-
DD		Y=P1 ; P1→ADEVAL	-	-	continue	-	F=0	-	-
DE		P1←P1-1	-	-	continue	-	-	-	-
DF		-	FILEVAL	POPL	Stack Ref	-	-	-	DPOP←0
E0	INITN	Q←P2+BIPOP, VTOUR	-	-	JMPR,D+POP Stack	-	-	ALLOC	FINI←1
E1		-	-	-	Continue	-	-	-	-
E2		-	-	-	JSR	-	MT	TOURN	-
E3			-	-	JMPR, D	-	-	ALLOC	-
E4									
E5									
E6									
E7									
E8	AVANCE(N)	Q←P2+BIPOP, VTOUR							
E9		RT←P2	-	-	continue	-	-	-	-
EA		-	-	-	JSR	-	-	TOURN	-
EB		Y=RT, ADEVAL	-	-	JSR	-	-	ALLOC	PT←1
EC		-	-	-	continue	VIDE	-	-	-
ED			POPE	FILEVAL	JMPD(BIPOP)	-	-	-	-
EE									
EF									

...../..

adresse	étiquette	commande partie opérative	ORBF	DBF	séquencement	attente	condition	@branch.	ordre
F0	SAUVE(N)	RT←BIPOP	-	-	Push stack	-	-	-	
F1		Y=P1 ; P1←ADEVAL	-	-	continue	DPOP=1	-	-	
F2		RT←RT-1	FILEVAL	POPL	continue	-	-	-	DPOP←0
F3		P1←P1-1	-	-	continue	-	F=0	-	-
F4		-			Stack ref	-	-	-	-
F5									
F6		-	-	-	JMPRD+POP Stack	-	-	ALLOC	-
F7									
F8	RECUL(N)	P1←P1-BIPOP	-	-	JMPR,D	-	-	ALLOC	DPOP←0
F9									
FA									
FB									
FC									
FD									
FE									
FF									

DEUXIEME PARTIE

Chapitre IV

DESCRIPTION TECHNIQUE DU PROCESEUR PINS

IV.1. Mécanisme d'horlogerie

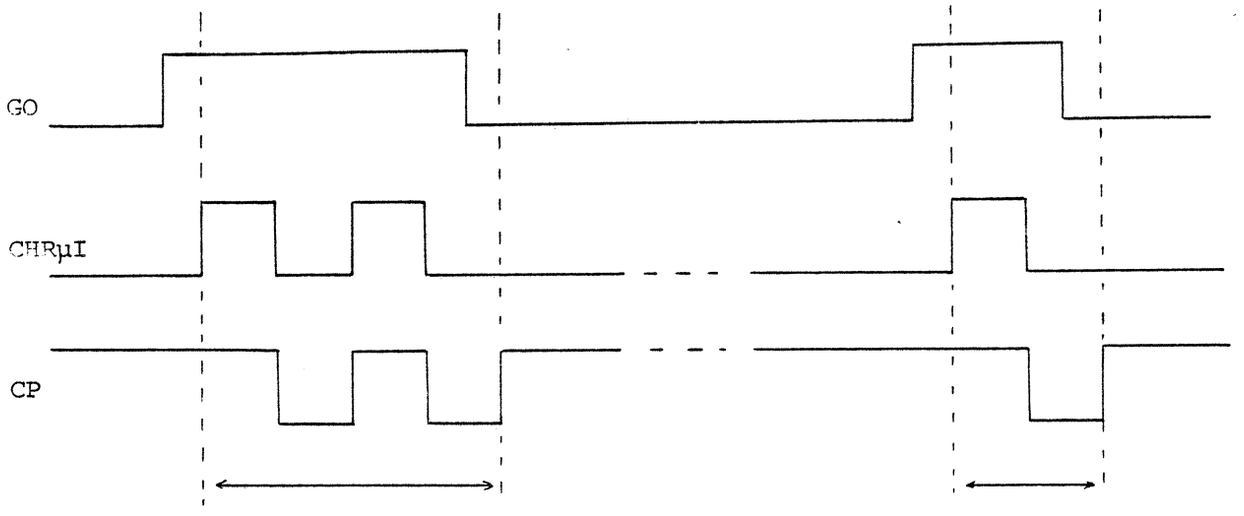
La mise au point des processeurs microprogrammés constituant PASCHLL est faite à l'aide d'un Matériel d'Aide au Développement d'Applications Microprogrammées, appelé système MADAM, développé dans le cadre du projet.
Ce système permet de charger des microprogrammes de test dans une mémoire RAM rapide (50 ns) et de contrôler son exécution en agissant sur l'horloge du processeur, par un signal appelé GO (marche arrêt).

Au niveau du processeur PINS, arrivent d'une part ce signal GO et d'autre part une horloge de base HB. A partir de ces deux signaux, le montage de la figure 1 permet de construire deux horloges:

- CHRMUI, qui est à ZERO à l'arrêt, charge la microinstruction suivante sur sa transition 0 -> 1,

- CP, qui est à UN à l'arrêt, définit chaque cycle de microinstruction.

Exemple: 2 pas, arrêt, 1 pas, arrêt...:



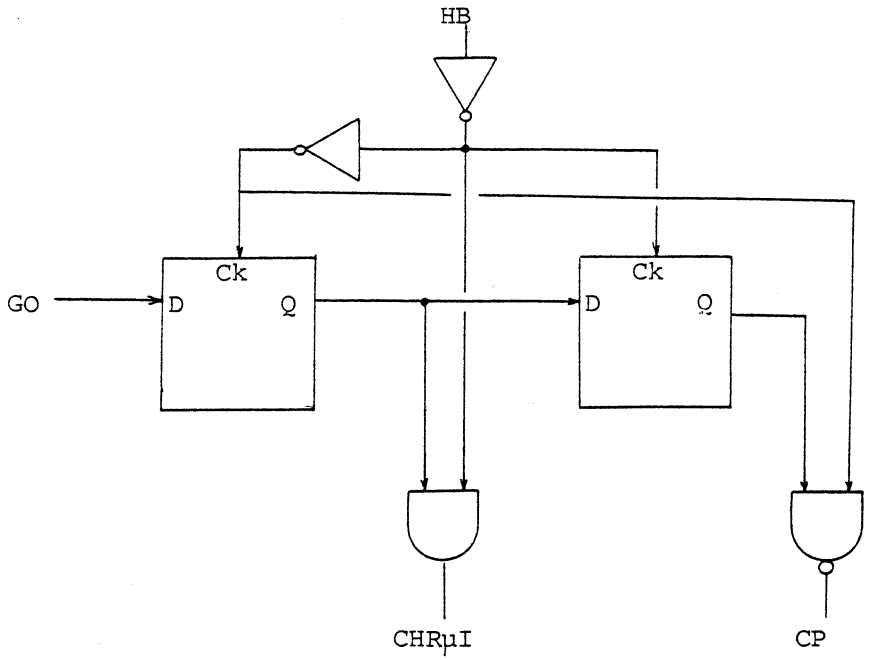


Figure IV.1

En prévision du fonctionnement simultané et synchrone de plusieurs processeurs de PASCHLL, une synchronisation a été prévue en utilisant les signaux GO:

GO de PINS, de PAC, de POP, de FILE et de PME.

Ces cinq signaux sont combinés de telle sorte que l'arrêt d'un quelconque des processeurs entraîne l'arrêt de tous.

On appelle ET-DES-GO le ET logique de tous les signaux GO associés à chaque processeur.

Le schéma de la figure IV.2. montre la synchronisation réalisée.

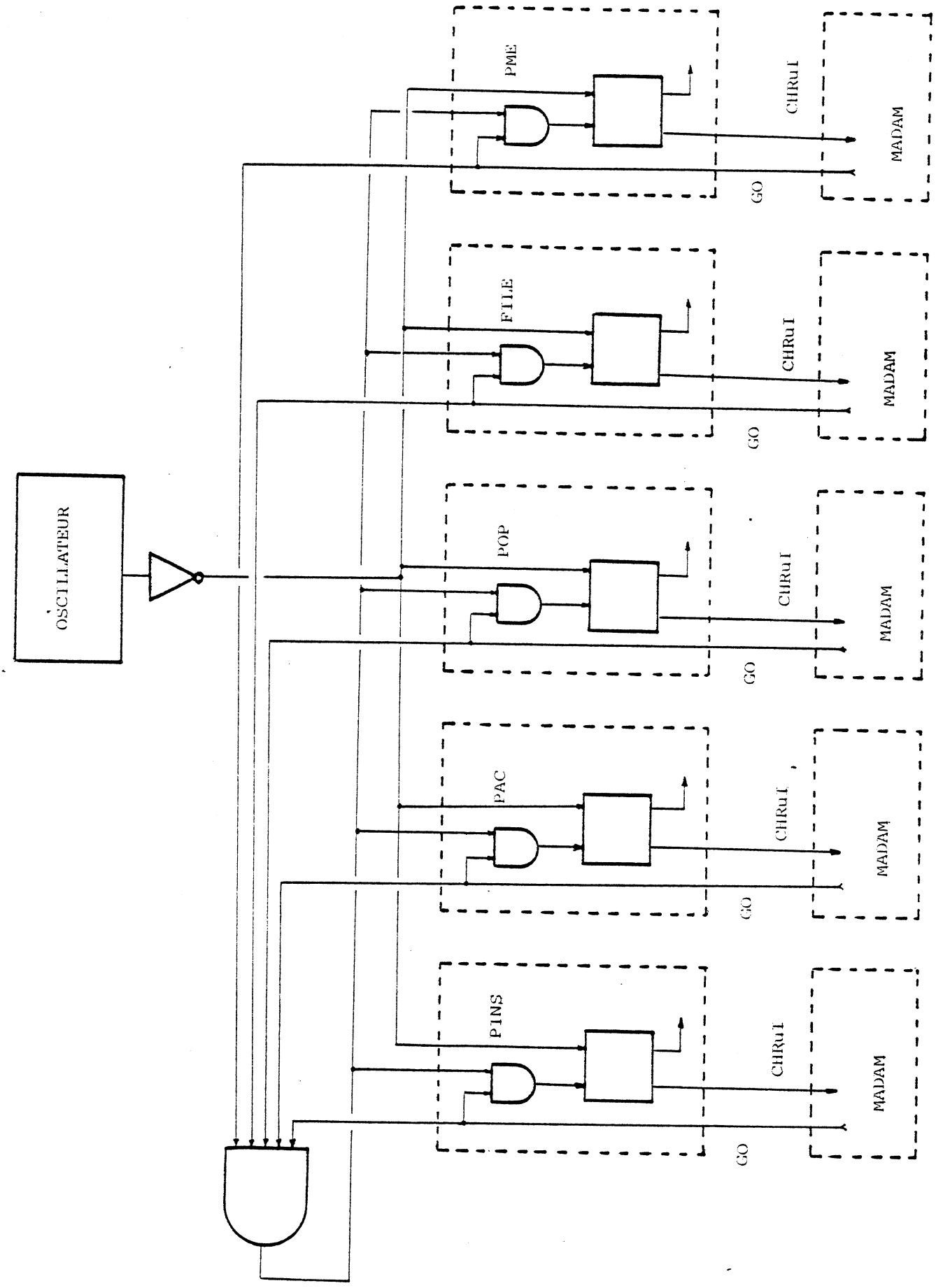


Figure IV.2 -Synchronisation des processeurs de PASC-HLL

IV.2. Méthode de séquencement

L'utilisation des macrocomposants AM 2900 suggère une structure dite "pipeline" pour la partie contrôle:

> Pendant l'exécution d'une microinstruction au cycle i, on calcule l'adresse de la microinstruction suivante (celle qui sera exécutée au cycle i+1).

La réalisation d'un tel mécanisme implique l'introduction d'une barrière temporelle, constituée par un registre dans lequel on charge la microinstruction extraite de la mémoire de contrôle, au début de chaque cycle.
Ce registre est appelé registre microinstruction, ou RmicroI.
Il est chargé sur le front montant de l'horloge appelée CHRMUI.

Les autres composants du système reçoivent une horloge appelée CP, active sur sa partie basse (elle est à UN au repos).

Une impulsion (négative) sur CP provoque la progression du séquenceur, le chargement du résultat d'un calcul dans l'opérateur.
On choisira donc le blocage de cette horloge CP pour réaliser les mécanismes d'attente et les actions conditionnelles, en générant un signal appelé FAUX, qui, lorsqu'il vaut UN, bloque l'horloge CP envoyée aux composants provisoirement bloqués.

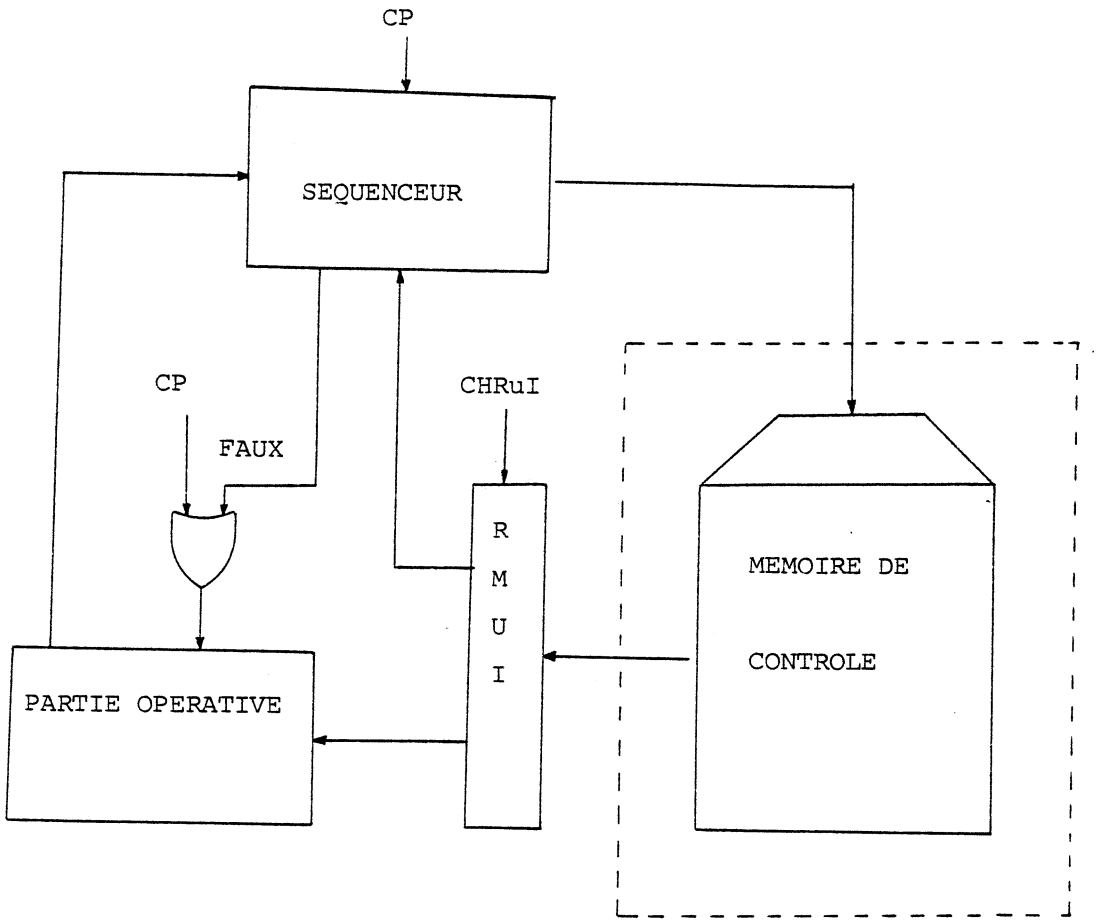


Figure IV.3

IV.2.1. Les branchements immédiats

L'étude des besoins en branchements à des séquences étiquetées a montré qu'il existait des relations assez fortes entre certaines séquences de traitement et des séquences étiquetées, en particulier des sous-programmes spécialisés.

Par ailleurs, le nombre minimum de microinstructions étant supérieur à 128, il est requis de passer à la puissance de 2 supérieure et donc d'étaler le microprogramme sur 256 adresses.

Il est alors intéressant d'envisager une éventuelle duplication de séquences étiquetées ou de sous-programmes, la duplication ne coûtant rien, au profit d'une réduction de la taille de la valeur immédiate nécessaire à l'adressage des étiquettes.

Le microprogramme a été découpé en 4 zones de 64 microinstructions.
Chacune de ces zones est elle-même découpée en 2 zones de 32 microinstructions, appelées A et B.
L'entrée dans une zone de type A est faite directement (voir

branchements directs), alors que la zone B va contenir les étiquettes adressables depuis la zone A associée. Comme de plus on désire combiner un branchement immédiat et un test par forçage d'une condition sur le bit de poids faible, les étiquettes seront implantées en adresses paires et seront donc au nombre de 16 par zone de type B.
Il y a donc théoriquement 4 groupes de 16 étiquettes, accessibles à partir des 4 zones de type A correspondantes.
Il faut en fait tenir compte d'une perte possible en nombre d'étiquettes si des séquences étiquetées comportent plus de deux microinstructions.

Le branchement immédiat est donc en fait relatif à une base définie par l'adresse de la microinstruction qui demande le branchement.

Ce qui est important est la réduction de taille de la valeur immédiate dans la microinstruction:

4 bits (V3, V2, V1, V0) pour adresser 64 étiquettes.

Si (Y7, Y6) sont les poids forts de l'adresse courante, le branchement a lieu à l'adresse:

(Y7, Y6, 1, V3, V2, V1, V0, 0)

zone B étiquette adresse paire

Le vecteur précédent est câblé sur l'entrée R du séquenceur, il est chargé dans le registre AR à chaque début de cycle, il est donc utilisable au cycle suivant comme adresse de branchement.

Le tableau 1 donne l'implantation du microprogramme de PINS (avec les étiquettes).

Tableau n° 1

MSB / LSB	zone 0		zone 1		zone 2		zone 3	
	0	1	2	3	4	5	6	7
00	DEPART	SREP	OPOP	OPE	THEN	S1	CARACC	SCPAC
01								
02				ETIQ			POPBIT	SF
03			CTZO		ELSE			
04		EMPILE		ACCES		S2	PACBIT	S11
05			OPZO		UNTIL			
06	CTCF	RCALL					POPAC	
07					NEHT			
08		OCALL		PFETCH			ATTBIT	S10
09	IO		OPAC		GOTO			
0A				FETCH		S5	ATTPOP	TESTK
0B								
0C		SAVEG	ACAC	S3	CASE	PUSH	ATTPAC	
0D	CODINV							
0E		SAVE	ACFE				→ERR	FETCH
0F	ERREUR		OPFE					
10		SSAVE		S4	OF	ACCES	→ERR	ACCES
11			ACZO					
12		FSAVE	LACC4	CTRL			→ERR	
13								
14						PFETCH	→ERR	PFETCH
15			UN2		FO			
16	ACOP	PUSH				POP	→ERR	TRP
17			BIN		ESAC			
18	ACCF		ENTER			FETCH	CONDATT	
19	CTAC				RETURN			ACCF
1A		FETCH					CATPOP	
1B								OPCF
1C	BIN2	SIO	EXIT				→ERR	
1D					PARAM			
1E	FETCH						→ERR	
1F								TROMPE

IV.2.2. Les branchements directs

La complexité de l'algorithme d'interprétation des instructions par le processeur PINS a conduit à la définition de tableaux d'adresses de branchements mémorisés dans des mémoires PROM (32x8 bits).

Les 6 bits de poids fort de chaque instruction définissent la classe à laquelle appartient cette instruction. Cette classe est caractérisée par:

- des caractéristiques enregistrées dans un premier tableau (tableau 2) donnant les informations suivantes:
  - PACBIT : instruction à envoyer à PAC
  - POPBIT : instruction à envoyer à POP
  - ATTBIT : attendre qu'on soit sorti de l'état conditionnel
  - CONDBIT : entrer dans l'état conditionnel
  - 2 bits de classe: accés , opérateur, fin d'expression ou contrôle
  - 1 bit spécial distinguant des cas particuliers.
- l'adresse de la séquence d'interprétation de l'instruction enregistrée dans un deuxième tableau (tableau 2).

Les caractéristiques de l'instruction courante, associées à celles de l'instruction précédente, définissent, par l'intermédiaire du tableau 3 des transitions, l'adresse de la séquence de traitement d'une transition.

Finalement, la réalisation des attentes conditionnelles étant faite par bouclage de l'adresse, il existe quatre sources possibles d'adressage direct:

- le tableau des transitions,
- le tableau des caractéristiques,
- le tableau de la séquence d'interprétation,

		PINS		MOD	classe	SP		→ ETC attente	→ PAC → POP	
00	////////	CODINV	CD		11			11	11	
01	THEN(m)	THEN	80		10			11	01	
02	WHILE/EXITIF	FETCH	1E		10			11	C1	
03	UNTIL	UNTIL	85		10			11	01	
04	CASE(m)	CASE	8C		10			01	01	
05	ROF+/ROF-	UNTIL	85		11			11	01	
06	INCAI/DECAI..	BIN2			10			00	01	
07	INCA/DECA..	FETCH	1E		10			00	01	
08	UNO	FETCH	1E		01			11	11	← cas d'erreur
09	UN1	FETCH	1E		01			11	11	
0A	UN2	FETCH	1E		01			11	11	
0B	UN3	FETCH	1E		01			11	11	
0C	UN4	FETCH	1E		01			11	11	
0D	UN5	FETCH	1E		01			11	11	
0E	UN6	FETCH	1E		01			11	11	
0F	FIELD/INXI..	UN2	55		01			11	11	
10	////////	CODINV	0D		11			11	11	
11	FO	FO	95		11			01	00	
12	ESAC	ESAC	97		11			01	00	
13	////////	CODINV	0D		11			11	11	
14	FORUP(m)	THEN	80		10			11	01	
15	FORDOWN(m)	THEN	80		10			11	01	
16	AFFA/AFFX	FETCH	1E		10			00	01	
17	////////	CODINV	0D		11			11	11	
18	BIN0	BIN	57		01			11	11	
19	BIN1	BIN	57		01			11	11	
1A	BIN2	BIN	57		01			11	11	
1B	BIN3	BIN	57		01			11	11	
1C	BIN4	BIN	57		01			11	11	
1D	BIN5	BIN	57		01			11	11	
1E	BIN6	BIN	57		01			11	11	
1F	BIN7	BIN	57		01			11	11	

CODE

CARACTERISTIQUES

Tableau n° 2 (1ère partie)

				MOD	Classe	SP		→ ETC attente	→ PAC → POP
20	AFFECT	FETCH	1E	1	10			00	11
21	PARAM	PARAM	9D	1	10			00	10
22	INCR	FETCH	1E	1	10			00	11
23	DECR	FETCH	1E	1	10			00	11
24	HALT	HALT	0F		11			01	00
25	EXITFOR	EXIT	5C		11			01	01
26	ENTER	ENTER	58		11			01	00
27	RETURN	RETURN	99		11			01	00
28	OF	OF	90		10	1		11	00
29	WITH	FETCH	1E		11			01	10
2A	CALL	RCALL	26		11			01	10
2B	EXIT	EXIT	5C		11			01	00
2C	CLRV	FETCH	1E	1	10			00	11
2D	WAIT(n)	WAIT			11			01	00
2E	CALLF	EMPILE	24		11	1		01	10
2F	SETV	FETCH	1E	1	10			00	11
30	NAME	FETCH	1E		00			11	11
31	INDD	FETCH	1E		00			11	11
32	ENDLOOP	UNTIL	85		11			00	00
33	TRAP(n)	TRAP			11			01	00
34	ZERO	ERREUR	0E		00	1		11	11
35	NEHT(m)	NEHT	87		11			01	00
36	ELSE	ELSE	83		11			01	00
37	FI				11			01	00
38	LIT8	FETCH	1E		00			11	11
39	LIT16				00			11	11
3A	LITI	LACC4	52		00			11	11
3B	LITS	LACC4	52		00			11	11
3C	ONE	ERREUR	0E		00	1		11	11
3D	GOTO(m)	GOTO	88		11			00	00
3E	LOOP(m)	THEN	8C		11			01	00
3F	////////	CODINV	CD		11			11	11

↑ CODE

↑ CARACTERISTIQUES

Tableau n° 2 (2ème partie)

Codage des classes d'instructions

Au sens des transitions définissant l'algorithme d'évaluation formelle des expressions post-fixées, nous avons défini quatre classes d'instructions:

- les instructions d'accés codées 00
- les opérateurs codés 01
- les fins d'expression codées 10
- les instructions de contrôle codées 11

Un bit supplémentaire a été introduit.. Il est appelé SP et définit des cas particuliers au sens des transitions:

- contrôle spécial 111 = CALLF
- fin d'expression spéciale 101 = OF
- accés spécial 001 = ZERO ou ONE
- opérateur spécial 011 = code invalide.

Un autre type de cas particulier existe: il s'agit des transitions

fin d'expression -> fin d'expression et

contrôle -> fin d'expression,

normalement interdites parce que détectant une expression vide: elles peuvent cependant se produire dans l'exécution d'une boucle FOR.

Transition FOR -> AFFECT puis transition ROF -> AFFECT.

Ces deux transitions définissent une séquence CTRL* dans laquelle on vérifie que le registre instruction contient bien le code de AFFECT (20).

S_{t-1} A_4A_3	S_t A_2A_1	SP A_0	Transition	Etiquette	Adresse
00	00	0	ACC → ACC	ACAC	4C
		1	ZERO/ONE	ACZO	51
	01	0	OP	ACOP	16
		1	OP, SP	!	0E
	10	0	FE	ACFE	4E
		1	OF	ACOF	18
	11	0	CT	!	0E
		1	CALLF	ACCF	F9
01	00	0	OP → ACC	OPAC	49
		1	ZERO/ONE	OPZO	45
	01	0	OP	OPOP	40
		1	OP, SP	!	0E
	10	0	FE	OPFE	4F
		1	OF	!	0E
	11	0	CT	!	0E
		1	CALLF	OPCF	FB
10	00	0	FE → ACC	CTAC	19
		1	ZERO/ONE	CTZO	43
	01	0	OP	!	0E
		1	OP, SP	!	0E
	10	0	FE	CTRL*	72 (FOR → AFFEC
		1	OF	!	0E
	11	0	CT	CTRL	72
		1	CALLF	CTCF	06
11	00	0	CT → ACC	CTAC	19
		1	ZERO/ONE	CTZO	43
	01	0	OP	!	0E
		1	OP, SP	!	0E
	10	0	FE	CTRL*	72 (ROF → AFFEC
		1	OF	!	0E
	11	0	CT	CTRL	72
		1	CALLF	CTCF	06

Tableau n° 3 - Décodage des transitions

1 1 0	COND	ATT	PAC	POP	ETC1	Etiquette	Adresse	Caractéristique
"	0	0	0	0	0	CARACO	C0	nulle = suite
					1		C1	"
"	0	0	0	1	0	POPBIT	C2	envoi à POP
					1		C3	"
"	0	0	1	0	0	PACBIT	C4	envoi à PAC
					1		C5	"
"	0	0	1	1	0	POPAC	C6	envoi à PAC et à POP
					1		C7	"
"	0	1	0	0	0	ATTBIT	C8	attente $\overline{ETC}$ = 1
					1		C9	suite
"	0	1	0	1	0	ATTPOP	CA	attente $\overline{ETC}$ = 1
					1		CB	envoi à POP
"	0	1	1	0	0	ATTPAC	CC	attente $\overline{ETC}$ = 1
					1		CD	envoi à PAC
"	0	1	1	1	0	APACPOP	CE	attente $\overline{ETC}$ = 1
					1		CF	envoi à PAC et POP
"	1	0	0	0	0		D0	
					1		D1	
"	1	0	0	1	0		D2	
					1		D3	invalide => erreur
"	1	0	1	0	0		D4	
					1		D5	
"	1	0	1	1	0		D6	
					1		D7	
"	1	1	0	0	0	CONDATT	D8	attente $\overline{ETC}$ = 1
					1		D9	entrée dans ETC
"	1	1	0	1	0	CATPOP	DA	attente $\overline{ETC}$ = 1
					1		DB	entrée dans ETC et envoi à POP
"	1	1	1	0	0	CATPAC	DC	attente $\overline{ETC}$ = 1
					1		DD	entrée dans ETC et envoi à PAC
"	1	1	1	1	0	CATPACPOP	DE	attente $\overline{ETC}$ = 1
					1		DF	entrée dans ETC et envoi à PAC et POP

Tableau n° 4 - Décodage des caractéristiques par la PROM - CARAC
(génération d'une adresse sur l'entrée D du séquenceur)

Les tableaux étant mémorisés dans des mémoires PROM à sortie 3 états, il est naturel de définir un bus 3 états pour l'entrée D du séquenceur.

La sélection de l'une des origines du bus parmi quatre, aurait pu être faite par un décodeur 2 -> 4 :
cette solution implique deux bits de sélection d'une part, et d'autre part l'existence d'un registre à sortie 3 états pour l'adresse et une porte 3 états pour les caractéristiques.
Disposant en fait de 3 bits de sélection (voir ROM de commande du séquenceur), il était plus simple de choisir un multiplexeur 2 -> 1 à sortie 3 états, qui assure le multiplexage d'un registre d'adresse normal et des bits caractéristiques.

La sélection des trois origines du bus est assurée par 3 bits dont un seul vaut zéro.
Celle du multiplexeur est assurée par le bit de poids fort de l'adresse de la ROM de commande: ainsi les instructions de séquencement dont le numéro est compris entre 0 et 7 sélectionnent une entrée et celles entre 8 et 15 l'autre entrée.

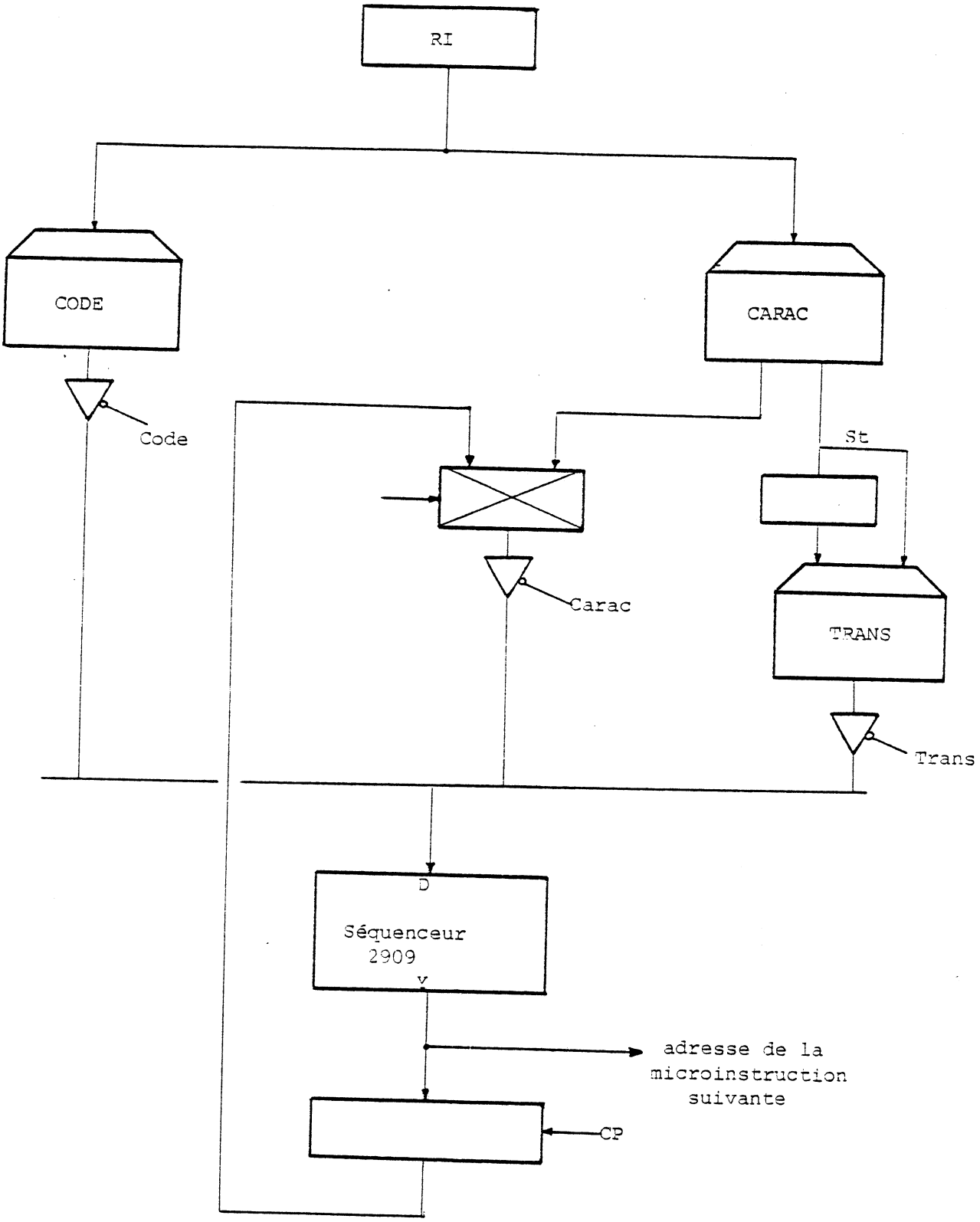


Figure IV.4 - Entrée D du séquenceur de PINS

IV.2.3. Les branchements conditionnels, les attentes

L'analyse de l'algorithme du processeur PINS a montré l'existence de deux types de branchements conditionnels:

- le premier est du type:

  si condition alors branchement sinon continuer

- le second est du type:

  allera ETIQUETTE + condition

Il est clair que le premier type suppose le choix parmi deux codes de séquencement, le second suppose le forçage d'une condition sur un bit d'adresse, en l'occurrence celui de poids faible.
Les attentes sont comparables au premier type:

  si EVENEMENT alors branchement sinon boucler

Les conditions du premier type sont au nombre de deux:

  (Qd = 0) et (k = 0)

Les évènements attendus sont au nombre de six:

- REPAC (réponse de PAC)
- BIPOP-NON-PLEIN (pour écrire dans BIPOP)
- BIPAC-NON-PLEIN (pour écrire dans BIPAC)
- NEXTOC (octet suivant d'instruction disponible)
- AMC (registre adresse mémoire libre)
- VMC (registre donnée mémoire disponible).

Ces huit conditions seront donc multiplexées et pourront définir le bit de poids faible de l'adresse de la ROM de séquencement.

Il y a par ailleurs 7 conditions du deuxième type:

- PLEIN (pile pleine)
- VIDE (pile vide)
- BASC (bascule mémorisant le bit D14)
- RI1 (bit n.1 de RI)
- D14 (bit n.14 du bus D)
- TRP (bascule TROMPE)
- ETC (bascule d'état conditionnel)

Ces 7 conditions sont multiplexées pour être amenées sur l'entrée ORO du séquenceur.

Les conditions du premier type et celles du second type n'étant pas utilisées simultanément, elles pourront être référncées par le même champ de 4 bits de la microinstruction, sachant que lorsque l'on sélectionne une condition du premier type, l'entrée ORO vaut zéro (multiplexeur 2 non sélectionné) et dans l'autre

cas la condition d'attente vaut zéro (multiplexeur 1 non sélectionné).

IV.2.4. Les instructions de séquencement

L'analyse de l'algorithme a montré le besoin de cinq types d'instructions de séquencement qui sont:

- CONTINUE passage en séquence
- JMPR branchement sur entrée R
- JSRR appel de sous-programme sur entrée R
- RTS retour de sous-programme
- JMPD branchement sur entrée D

avec un certain nombre de variantes:

- JMPD-code, valide interruption
- JMPD-carac, valide interruption
- JMPD-étoile, valide interruption
- JMPD-code, attente et valide interruption
- JMPD-transition, valide interruption et attente
- JMPR, conditionnel
- JMPR, valide interruption
- JMPR, attente
- JSRR, conditionnel
- JSRR, attente
- CONTINUE, attente

Ces 16 instructions sont décodées par une mémoire PROM de 32 mots de 8 bits, la condition entrant sur le bit de poids faible de son adresse.

Un bit spécial, appelé FAUX, est donné par cette PROM, qui autorise l'exécution des ordres uniquement lorsqu'une instruction conditionnelle est accompagnée d'une condition VRAIE (FAUX=0).

Les 8 bits de la PROM de séquencement sont les suivants:

- CODE actif à 0, valide CODE sur entrée D
- CARAC actif à 0, valide CARAC sur entrée D
- TRANS actif à 0, valide TRANS sur entrée D
- FAUX vaut 1 si condition FAUSSE
- TRP valide prise en compte interruption
- S1
- S0 codes envoyés au séquenceur 2909
- FE

Remarque:

le bit POP envoyé au séquenceur est toujours égal à S0.

Code	CATT	Adresse		CODE CARAC	TRANS	FAUX	TRP	S1	SO	FE	POP	Valeur
00	0	00	CONT	1	1	1	0	0	0	0	1	0	E1
	1	01	CONT	1	1	1	0	0	0	0	1	0	E1
01	0	02	JMPR	1	1	1	0	0	0	1	1	1	E3
	1	03	JMPR	1	1	1	0	0	0	1	1	1	E3
02	0	04	JSRR	1	1	1	0	0	0	1	0	1	E2
	1	05	JSRR	1	1	1	0	0	0	1	0	1	E2
03	0	06	JMPR, TRP	1	1	1	0	1	0	1	1	1	EB
	1	07	JMPR, TRP	1	1	1	0	1	0	1	1	1	EB
04	0	08	CONT, FAUX	1	1	1	1	0	0	0	1	0	F1
	1	09	JMPD-CODE	0	1	1	0	0	1	1	1	1	67
05	0	0A	CONT, FAUX	1	1	1	1	0	0	0	1	0	F1
	1	0B	JMPR	1	1	1	0	0	0	1	1	1	E3
06	0	0C	JMPD-CARAC, TRP	1	0	1	0	1	1	1	1	1	AF
	1	0D	JMPD-CARAC, TRP	1	0	1	0	1	1	1	1	1	AF
07	0	0E	RTS	1	1	1	0	0	1	0	0	0	E4
	1	0F	RTS	1	1	1	0	0	1	0	0	0	E4
08	0	10	CONT, FAUX	1	1	1	1	0	0	0	1	0	F1
	1	11	JSRR	1	1	1	0	0	0	1	0	1	E2
09	0	12	JMPD-*, TRP	1	0	1	0	1	1	1	1	1	AF
	1	13	JMPD-*, TRP	1	0	1	0	1	1	1	1	1	AF
0A	0	14	JMPD-CODE, TRP	0	1	1	0	1	1	1	1	1	8F
	1	15	JMPD-CODE, TRP	0	1	1	0	1	1	1	1	1	8F
0B	0	16	JMPD-*. FAUX. TRP	1	0	1	1	1	1	1	1	1	BF
	1	17	JMPD-CODE, TRP	0	1	1	0	1	1	1	1	1	8F
0C	0	18	JMPD-*, FAUX	1	0	1	1	0	1	1	1	1	B7
	1	19	CONT	1	1	1	0	0	0	0	1	0	E1
0D	0	1A	JMPD-*, TRP, FAUX	1	1	0	1	1	1	1	1	1	BF
	1	1B	JMPD-TRANS, TRP	1	1	0	0	1	1	1	1	1	CF
0E	0	1C	JMPD-*, FAUX	1	0	1	1	0	1	1	1	1	B7
	1	1D	JSRR	1	1	1	0	0	0	1	0	1	E2
0F	0	1E	JMPD-*, FAUX	1	0	1	1	0	1	1	1	1	B7
	1	1F	JMPR	1	1	1	0	0	0	1	1	1	E3

Tableau n° 8 - PROM de décodage des instructions de séquencement

IV.2.5. Le traitement des interruptions

La prise en compte des interruptions est autorisée par certaines instructions de séquencement: le bit TRP sortant de la ROM vaut alors UN.

Il y a deux types d'interruptions:

1/ lorsque le processeur PINS a fait un choix (anticipation) et que ce choix était mauvais, le processeur POP l'avertit en mettant à UN la bascule TROMPE: on a choisi de forcer l'adresse 'FF' en mettant à UN toutes les entrées OR, de OR0 à OR7 ;

2/ lorsqu'une erreur a été détectée par l'un des processeurs PAC, POP ou PME, on force l'adresse 'OF' en mettant à UN toutes les entrées OR et en mettant à ZERO l'entrée ZERO du séquenceur 2909 de plus fort poids.

Ces deux fonctions sont générées selon le tableau n.7, qui prend en compte une interruption provenant de PME sur erreur.

INT-PME	valide TRP	TROMPE	ETC	ERREUR	Adresse
0	x	x	x	x	0F
1	0	x	x	x	suivante
1	1	x	x	1	0F
1	1	1	1	0	FF

Tableau 7 - Interruptions, forçage d'une adresse.

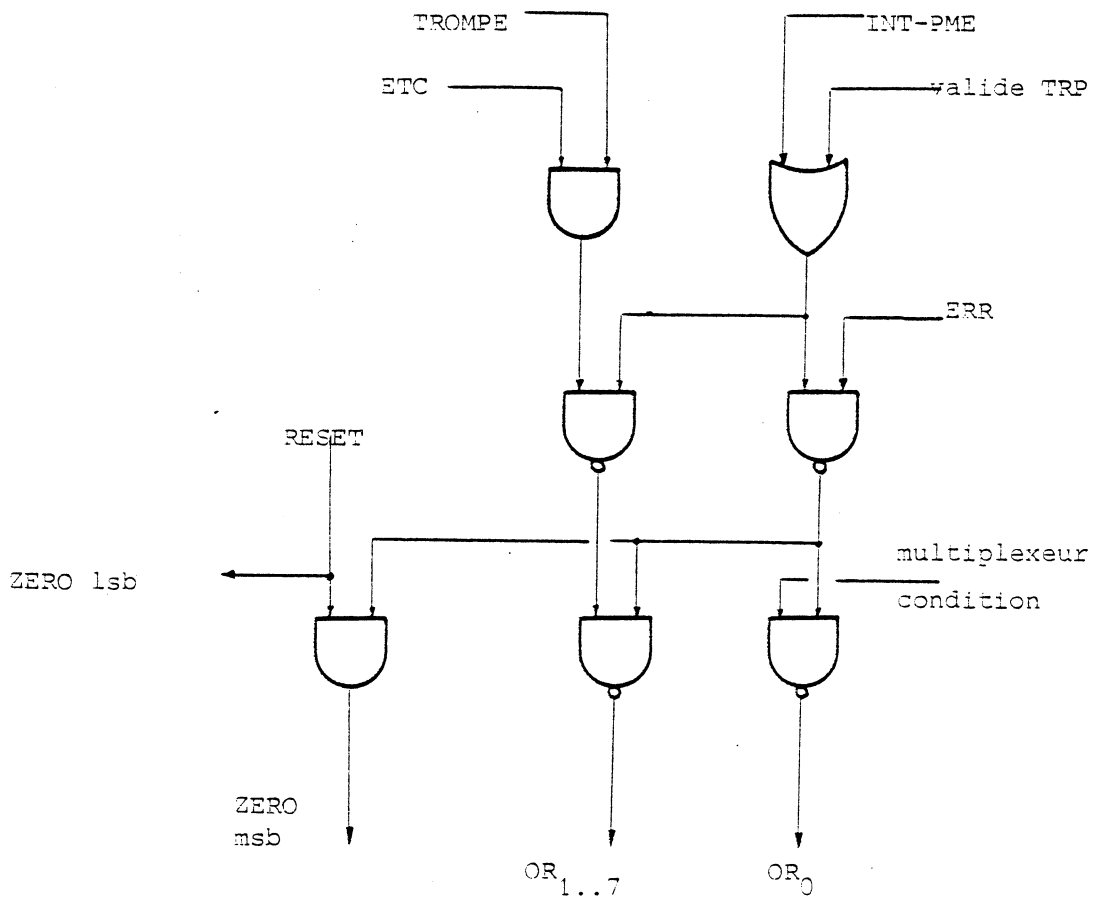


Figure IV.5 -Forçage des adresses d'interruption

IV.3. - Partie opérative du processeur PINS

IV.3.1. Implantation des variables

Cette implantation est faite, une fois de plus, autour d'un opérateur en tranches AM 2901

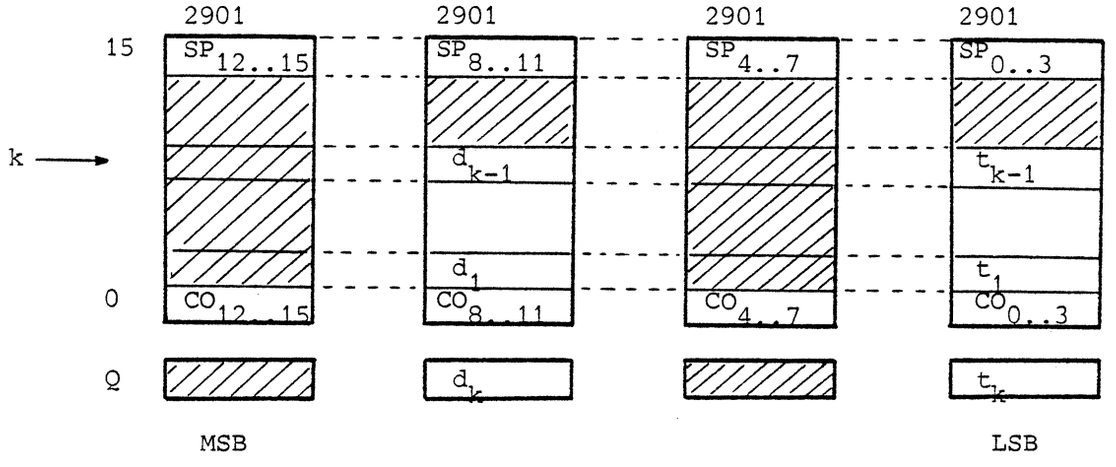
- soit dans ses registres internes (16 registres + 1 accumulateur),
- soit sur ses deux bus d'entrée (D) et de sortie (Y).

Il existe deux variables de 16 bits qui sont le compteur ordinal CO et le sommet de pile SP, sur lesquelles on doit effectuer des opérations

- d'addition (pour un branchement),
- d'incrémentation (sur CO et SP) ou
- de décrémentation (sur SP).

De plus, la gestion de l'image théorique de la file d'évaluation suppose l'existence de deux ensembles de variables de 4 bits, appelées suites (dk) et (tk) qui doivent être gérées en pile.

La solution retenue consiste à combiner dans un ensemble de 4 tranches à la fois les deux variables de 16 bits CO et SP et les deux piles (dk) et (tk), en plaçant les deux sommets dk et tk dans les accumulateurs et les autres éléments d1,...,dk-1 et t1,...,tk-1 dans des registres de numéros 1 à k-1.

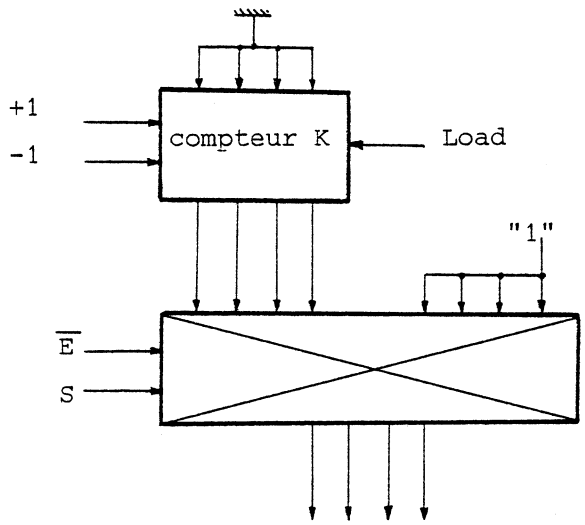


L'adressage de ces éléments nécessite la génération du numéro de registre qui peut être égal à 0, 15 ou k.

Il est réalisé par un multiplexeur "quadruple deux vers un", en utilisant deux commandes, select et enable, selon le tableau:

enable	select	n. registre
1	x	0 compteur ordinal
0	0	k dk ou tk
0	1	15 sommet de pile

Le montage est le suivant:



numéro de registre (A ou B)

IV.3.2. Problème de la propagation de la retenue

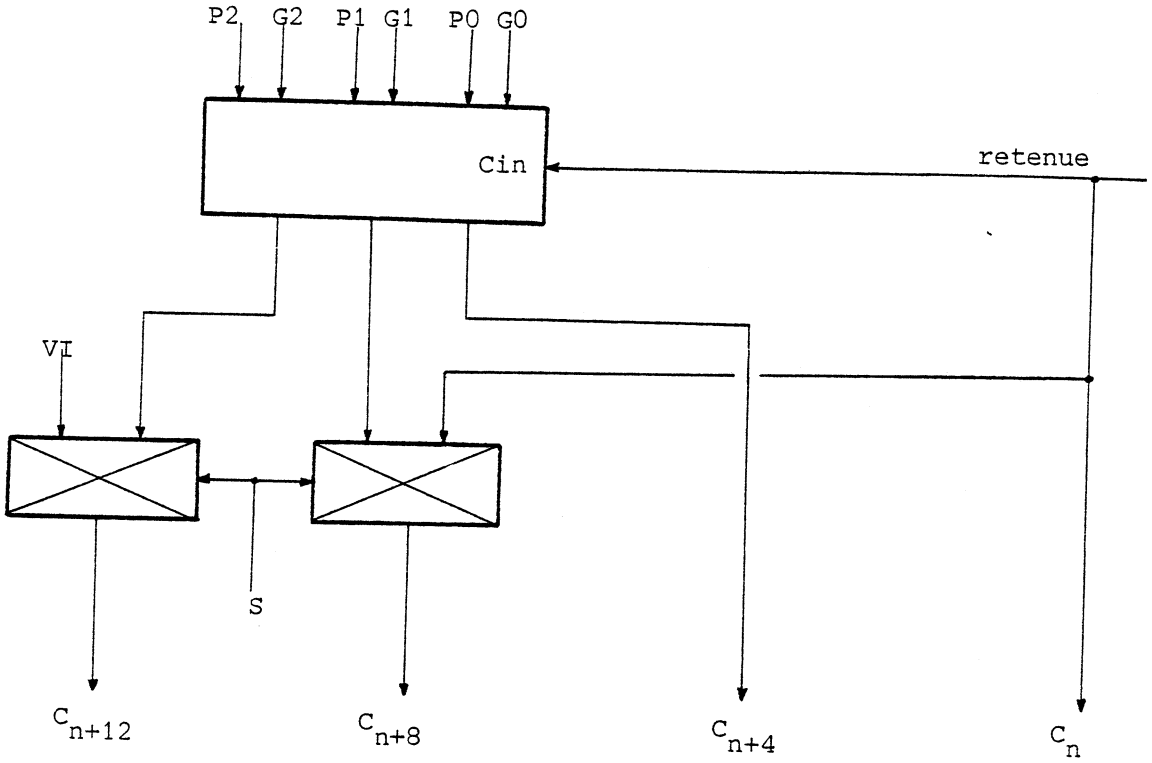
L'implantation précédente implique l'existence de deux modes de fonctionnement pour l'opérateur:

- fonctionnement en 16 bits ou
- fonctionnement en 2 fois 8 bits (si possible en parallèle)

Il faudra donc faire une distinction au niveau du codage des instructions et du mode de propagation de la retenue.

Il s'agit en fait de multiplexer la retenue entrant sur les poids forts entre une valeur immédiate et la retenue propagée par un circuit "carry look-ahead" AM 2902.

Il sera nécessaire de définir deux bits de contrôle donnant la retenue entrante pour la tranche de poids faible, et la commande de sélection de la retenue entrante pour les deux tranches de poids fort.



IV.3.3. Etude des instructions opératives

L'analyse du microprogramme symbolique fait apparaitre deux classes d'instructions opératives correspondant soit au traitement des suites (dk) et (tk), soit au traitement de variables de 16 bits (CO et SP).

La première classe comprend les opérations :

Qd + dk -> dk	et	Qt + tk -> tk
Qd + 1 -> Qd	et	Qt + 0 -> Qt
Qd - 1 -> Qd	et	Qt + 1 -> Qt
Yd = Qd -> Qd	et	Yt = Qt -> Qt
Yd = dk	et	Yt = tk
Qd + dk -> Qd	et	Qt + tk -> Qt
0 -> Qd	et	0 -> Qt
Qd -> dk	et	Qt -> tl
PILE 0 -> Qd	et	0 -> Q

La deuxième classe comprend les opérations :

Y = 0

Y = CO
Y = SP
Y = SP et SP - 1 -> SP
Y = SP + 1 -> SP
Y = CO + 1
Y = CO + 1 -> CO
Y = CO - 1 -> CO
Y = CO avec Y15 = 1
Y = $\overline{\text{PILE 0}}$ (PILE de contrôle LSB)
Y = $\overline{\text{PILE 1}}$ (PILE de contrôle MSB)
Y = RML 0 (tampon lecture mémoire LSB)
Y = RML 1 (tampon lecture mémoire MSB)
Y = BI0 (bus instruction 10 bits LSB étendus par ZEROS)
Y = BI1 (bus instruction 16 bits)
Y = ETAT (ensemble des bits d'état)
Y = $\overline{\text{RML0}}$ -> SP
Y = $\overline{\text{PILE 0}}$ -> CO
Y = $\overline{\text{PILE 1}}$ -> CO
Y = RML 1 -> CO
Y = CO + BI0 -> CO (branchements relatifs courts)
Y = CO + BI1 -> CO (branchements relatifs longs)
Y = CO + BI0

Le nombre total d'opérations est égal à 32. On introduit donc une mémoire PROM de décodage de 32 mots dont la sortie va permettre de commander :

- Les opérateurs 2901
- La retenue entrante
- Le numéro de registre
- L'origine de l'entrée D des opérateurs 2901.

On regroupe toutes ces commandes "hardware" en une seule commande "firmware" dans un champ de 5 bits de la microinstruction.

IV.3.4. Codage des instructions opératives

L'opérateur AM 2901 impose un certain codage, avec parfois plusieurs codes possibles pour commander une opération donnée. On choisira donc le code le mieux adapté au sens de l'ensemble des codes nécessaires, en essayant de faire apparaître soit des bits constants, soit des bits égaux deux à deux, ce qui permettra de réduire d'autant le nombre total de bits.

Soit M la matrice de commande, $M = (C_i^j)$

Où i est le numéro d'un bit de commande $(0 \leq i \leq n)$

j est le numéro de la commande $(0 \leq j \leq p)$

1) Recherche des bits constants :

$\exists i$ tel que $C_i^j = Cst \quad \forall j \in [0,p]$

Conséquence : Le bit i peut disparaître de la matrice.

2) Recherche des bits égaux deux à deux :

$\exists i1$ et i2 tels que $C_{i_1}^j = C_{i_2}^j \quad \forall j \in [0,p]$

Conséquence : l'un des deux bits i1 ou i2 peut disparaître.

3) Recherche des bits "presque égaux" deux à deux :

$\exists i1$ et i2 tels que $\forall j \in [0,p]$ on ait:

- soit $C_{i_1}^j = C_{i_2}^j$ (egalité)
- soit $C_{i_1}^j = \varnothing$ (indifférent)
- soit $C_{i_2}^j = \varnothing$ (indifférent)

Conséquence : l'un des deux bits i1 ou i2 peut disparaître, à condition que :

- si $C_{i_1}^j = \varnothing$ alors $C_{i_1}^j = C_{i_2}^j$
- si $C_{i_2}^j = \varnothing$ alors $C_{i_2}^j = C_{i_1}^j$

4) Recherche de combinaisons :

Lorsque le nombre de bits de commande a été réduit en utilisant les trois règles précédentes, on peut rechercher des combinaisons.

Exemple :

$\exists i1$ et 12 tels que $\forall j \in [0,p]$

on ait : $C_{i_3}^j = C_{i_1}^j \cdot C_{i_2}^j$,

avec éventuellement $C_{i_1}^j$ ou $C_{i_2}^j$ ou $C_{i_3}^j = \varnothing$

Conséquence :
Le bit i3 peut disparaître de la matrice et sa génération suppose le calcul d'un fonction de bits sortant de la matrice, à l'extérieur de la matrice.

Cette solution sera adoptée lorsque le nombre de bits de commande sera légérement supérieur à la taille physique de la mémoire PROM contenant cette matrice.

Exemple : Si l'on a 18 bits de commande après utilisation des trois premières règles de réduction, on essaiera de "faire sortir" deux bits, plutôt que d'étendre le nombre de bits à 24.

L'application des règles précédentes fait apparaître dans ce cas particulier que :

- Le bit I8 de contrôle des opérateurs 2901 est égal à ZERO,

- Les bits I0, I1, I2, I6 et I7 de contrôle des opérateurs 2901 sont les mêmes pour les opérateurs de poids faible et pour ceux de poids fort, et donc qu'il ne reste que I3 et I5 à distinguer ;

- Si le vecteur (D2, D1, D0) donne le code de l'origine de l'entrée D, et si l'on affecte les numéros

(1, 1, 0) et (1, 1, 1)

pour PILE0 et PILE1, alors on génère les deux bits I4 pour les opérateurs 2901 par :

I4 (MSB) = D2 . D1

I4 (LSB) = I4 (MSB) . I6

- Si on appelle E et S les deux bits de sélection du numéro de registre (cf. paragraphe 1), on remarque que

lorsque E = 1, la valeur de S est indifférente.

On peut donc réserver la combinaison

(E, S) = (1, 1) et Cin = 1 (retenue entrante)

pour valider le chargement d'une bascule avec la sortie F = 0 ce qui permettra de tester Qd ≠ 0

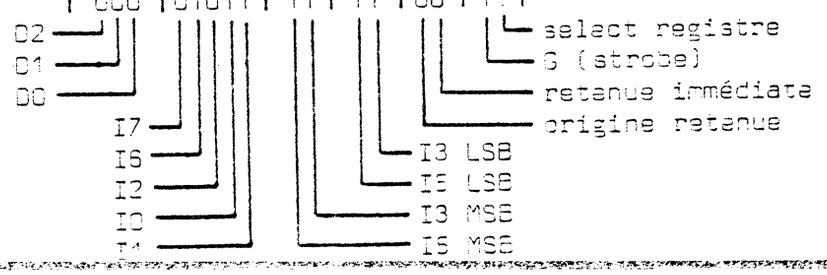
valide (Qd ≠ 0) = E.S.R

force (Y15 = 1) = E.S.

Le tableau complet de la matrice de commande de la partie opérative est donné par le tableau suivant :

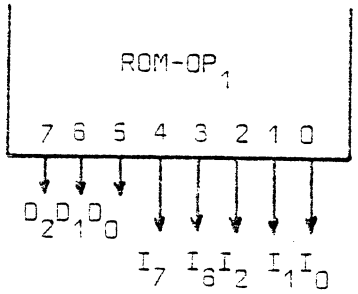
Numéro opération	Codage opérateur					
00	000	01011	10	10	00	00
01	000	01011	11	11	00	11
02	000	01011	11	11	00	01
03	000	10100	01	01	00	01
04	000	11100	00	00	01	01
05	000	01100	00	00	01	10
06	000	11100	00	00	01	10
07	000	11100	01	01	00	10
08	000	11000	00	00	00	00
09	000	00010	00	01	11	00
0A	000	00010	01	00	01	11
0B	000	00010	00	00	00	00
0C	000	01011	00	00	00	00
0D	000	00000	00	00	00	00
0E	000	00011	10	10	01	11
0F	000	11010	00	00	00	00
10	110	00111	11	11	01	00
11	110	01111	11	11	00	00
12	111	01111	11	11	00	00
13	100	01111	00	00	00	00
14	101	01111	00	00	00	00
15	010	01111	00	00	00	00
16	011	01111	00	00	00	00
17	001	01111	00	00	00	00
18	100	11111	00	00	00	01
19	110	11111	11	11	00	10
1A	111	11111	11	11	00	10
1B	101	11111	00	00	00	10
1C	010	11101	00	00	00	10
1D	011	11101	00	00	00	10
1E	010	01101	00	00	00	10
1F	000	01011	11	11	00	11

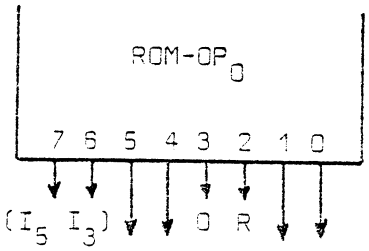
N° entrée D	Origine
00	rien
01	ETAT
02	M_0
03	M_1
04	RML_0
05	RML_1
06	$PILE_0$
07	$PILE_1$



PROM partie opérative

N°	Opération		OP_1	OP_0
00	$Y = 0$		08	A0
01	$Y = CO$		08	F2
02	$Y = SP$		08	F1
03	$Y = SP,\ SP - 1 \rightarrow SP$		14	51
04	$Y = SP + 1 \rightarrow SP$		1C	05
05	$Y = CO + 1$		0C	06
06	$Y = CO + 1 \rightarrow CO$		1C	06
07	$Y = CO - 1 \rightarrow CO$		1C	52
08	$Q_d + d_k \rightarrow d_k$	$Q_t + t_k \rightarrow t_k$	18	00
09	$Q_d + 1 \rightarrow Q_d$	$Q_t + 0 \rightarrow Q_t$	02	1C
0A	$Q_d - 1 \rightarrow Q_d$	$Q_t + 1 \rightarrow Q_t$	02	47
0B	$Y_d = Q_d \rightarrow Q_d$	$Y_t = Q_t \rightarrow Q_t$	02	00
0C	$Y_d = d_k$	$Y_t = t_k$	08	00
0D	$Q_d + d_k \rightarrow Q_d$	$Q_t + t_k \rightarrow Q_t$	00	00
0E	$0 \rightarrow Q_d$	$0 \rightarrow Q_t$	03	A7
0F	$Q_d \rightarrow d_k$	$Q_t \rightarrow t_k$	1A	00
10	$\overline{PILE}_0 \rightarrow Q_d$	$0 \rightarrow Q_t$	C7	F4
11	$Y = \overline{PILE}_0$		CF	F0
12	$Y = \overline{PILE}_1$		EF	F0
13	$Y = RML_0$		8F	00
14	$Y = RML_1$		AF	00
15	$Y = M_0$		4F	00
16	$Y = M_1$		6F	00
17	$Y = ETAT$		2F	00
18	$Y = RML_0 \rightarrow SP$		9F	01
19	$Y = \overline{PILE}_0 \rightarrow CO$		0F	F2
1A	$Y = \overline{PILE}_1 \rightarrow CO$		FF	F2
1B	$Y = RML_1 \rightarrow CO$		BF	02
1C	$Y = CO + M_0 \rightarrow CO$		50	02
1D	$Y = CO + M_1 \rightarrow CO$		70	02
1E	$Y = CO + M_0$		40	02
1F	$Y = CO,\ Y_{15} = 1$		08	F3



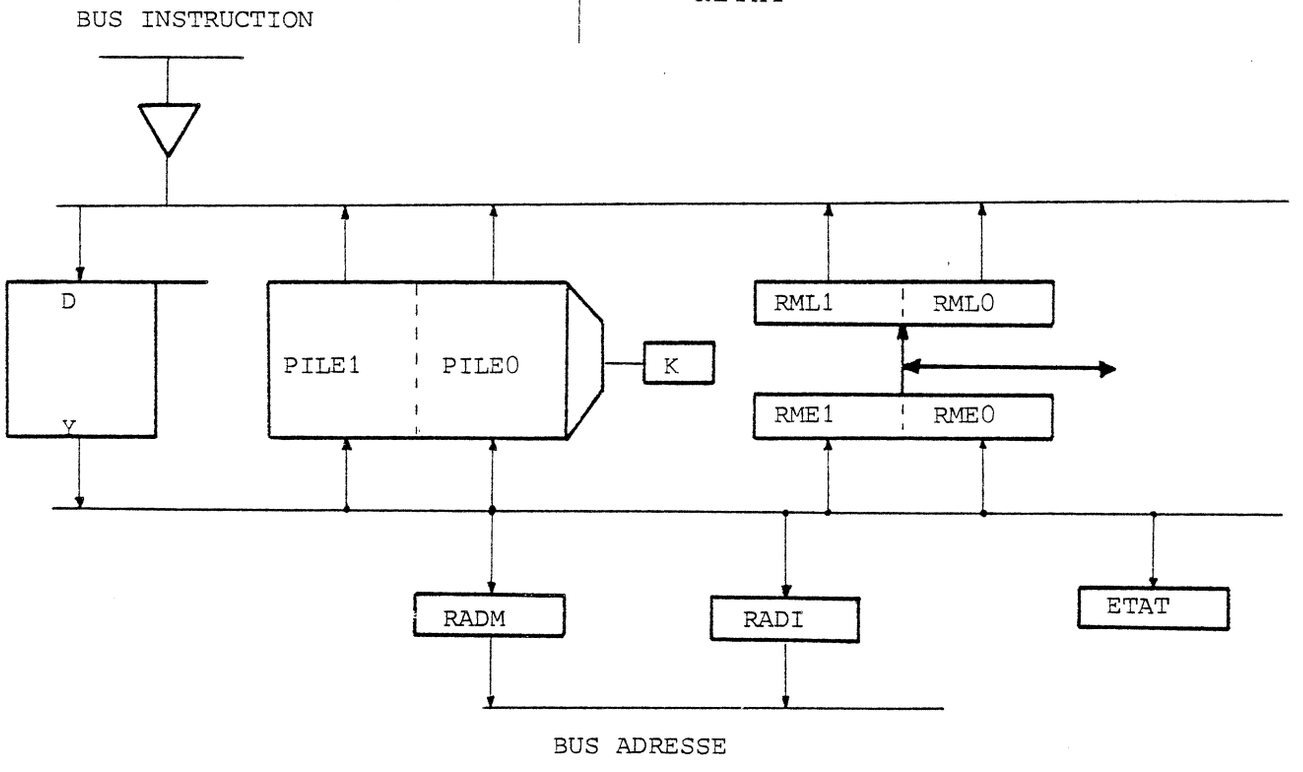


G	S	A=B
1	x	O=CO
0	0	k
0	1	F=SP

Schéma du chemin de données autour de l'opérateur

Codage de la destination de la sortie Y

00	rien
01	PILE0
02	PILE1
03	RME0
04	RME1
05	RADM (mémoire de contexte)
06	RADI (lecture d'instructions)
07	RETAT



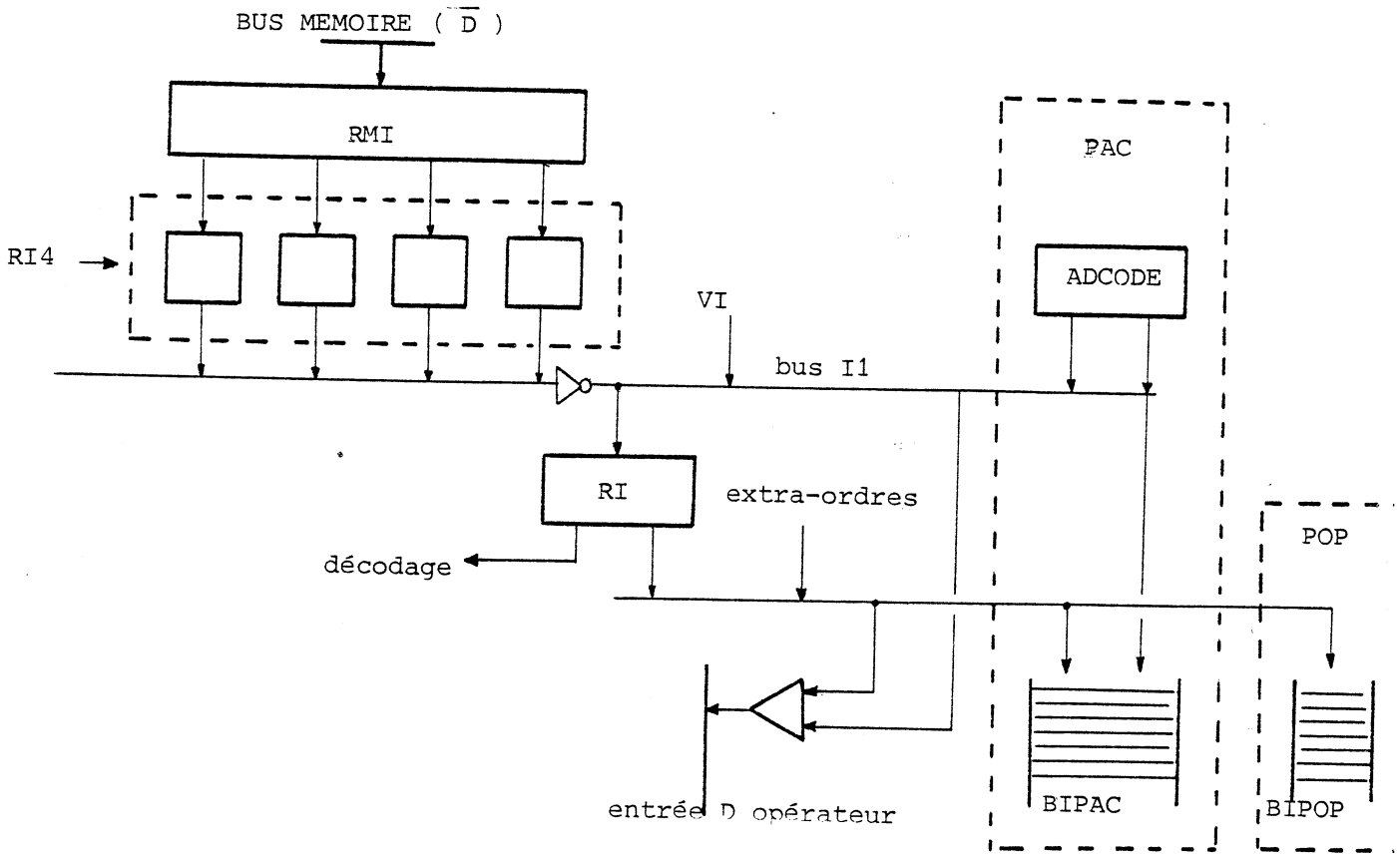
III.4. Commandes du bus instruction

Le bus instruction est géré par le processeur PINS qui l'utilise pour

- recevoir les instructions de la mémoire centrale,
- les envoyer aux deux processeurs PAC et POP à travers les files d'attente BIPAC et BIPOP,
- générer les 'extra-ordres' vers les processeurs POP et FILE pour la gestion des pointeurs sur la file d'évaluation, et finalement
- pour recevoir l'adresse du début d'un segment de code fourni par le processeur PAC à travers le registre ADCODE.

Le bus instruction est décomposé en deux parties appelées BI0 et BI1.

Le chemin de données autour de ces deux bus est le suivant :



Les commandes à générer sont les suivantes :

1) envoi d'une instruction dans BIPAC

- instruction de 16 bits :

(RI, RI4) -> BIPAC

- instruction ZERO/ONE :

(RI, VI) -> BIPAC

- instruction de SAUVEGARDE :

SAVEG -> BIPAC

- instruction ENTER/ENTEREXT
  RETURN/RETURNEXT :

(RI, BASC) -> BIPAC

(la bascule BASC contient le mode interne ou externe).

2) envoi d'une instruction dans BIPOP

- transmission d'un opérateur :

RI -> BIPOP

- instruction ENTEREXT ou RETURNEXT:

(RI,BASC) -> BIPOP

- extra-ordres:

(RECULE, Yt)
(INIT,Yd)
(AVANCE/INIT,Yd)
(OF,Yd) -> BIPOP
(SAUVE,Yd)
(RESTAURE,Yd)
SAVEG

3/ chargement de RI:

RI4 -> RI

4/ lecture de ADCODE:

ADCODE -> BI0 et BI1

Génération des extra-ordres

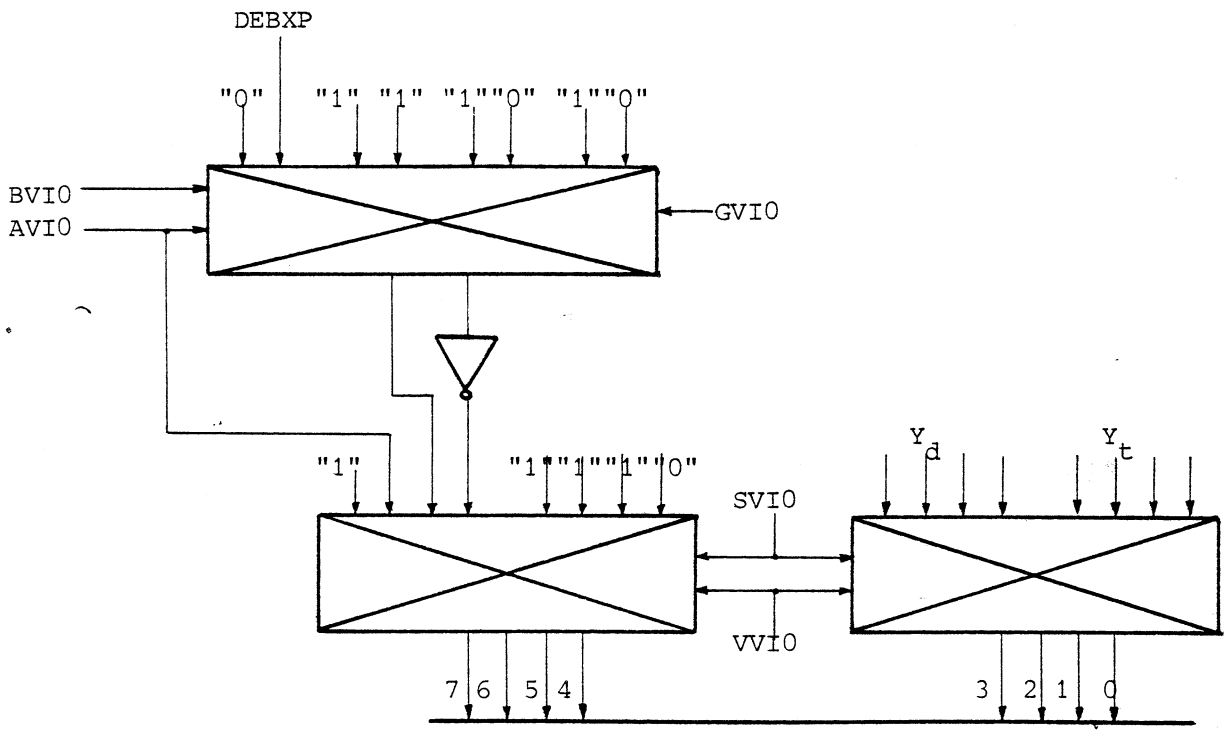
Les codes à générer sont les suivants:

RECULE, Yt	1 1 1 0	
INIT, Yd	1 1 0 1	
SAVEG	1 0 0 1	
RESTAURE, Yd	1 0 1 1	
SAUVE, Yd	1 1 1 1	
OF, Yd	1 0 1 0	
AVANCE/INIT	1 1 0 x	x = DEBXP

Ils peuvent être générés par le circuit combinatoire suivant, qui nécessite quatre commandes:

- SVI0 = sélection entre (RECULE,Yt) et (ORDREi,Yd)

- GVI0, AVI0, BVI0 = validation et sélection entre les différents extra-ordres à générer (INIT, SAVEG, RESTAURE, SAUVE, OF).



Les autres commandes à générer sont les suivantes:

- V4 : validation de RI4 sur le bus BI1
- VVI1 : validation des constantes ZERO et UN sur BI1
- VVIO : validation des extra-ordres sur BIO
- VRIO : validation du registre RI sur BIO
- SRI : sélection entre BASC et RI1 sur BIO (1)
- VAD : validation de ADCODE sur BIO et BI1

On obtient le tableau de commandes suivant:

ORDRE	V4 GVIO	BVIO	AVIO	VVI1 SVIO	VVIO	VRIO	SRI	VAD
ADCODE -> BI	1	0	0	1	1	1	0	0
(RI,RI4) -> BIPAC	0	1	0	1	1	0	0	1
(RI,VI) -> BIPAC	1	1	0	0	1	0	0	1
RI4 -> RI	0	1	1	1	1	0	0	1
(RI,BASC) -> BIPOP	1	0	1	0	1	0	1	1
RI -> BIPOP	1	0	1	0	1	0	0	1
(RECUL,Yt) -> BIPOP	1	0	0	0	0	1	1	1
(INIT,Yt) -> BIPOP	1	0	1	1	0	1	1	1
SAVEG -> BIPOP	1	0	0	1	0	1	1	1
(AV/INIT) -> BIPOP	0	1	1	1	0	1	1	1
(OF,Yd) -> BIPOP	0	1	0	1	0	1	1	1
(SAUVE,Yd) -> BIPOP	0	0	0	1	0	1	1	1
(RI,BASC) -> BIPOP	1	1	0	0	1	0	1	1

Ce tableau fait apparaître deux égalités: V4 = GVIO et VVI1 = SVIO et une répartition par quatre du couple (GVIO,VVIO) qui permet d'utiliser directement le numéro de l'opération pour ces deux commandes.

On obtient le nouveau tableau:

N°	ordre	commandes	hexa
00	RECULE	1 0 0 0 1 1 1 1	8F
01	SAUVE	0 0 1 1 1 1 1 1	3F
02	RESTAURE	0 0 0 1 1 1 1 1	1F
03	SAVEG → PAC	1 1 0 1 0 1 0 1	D5
04	ADCODE → BUS	1 0 0 1 0 1 0 0	94
05	(RI,BASC) → POP	1 0 1 0 1 0 1 1	AB
06	RI → POP	1 0 1 0 1 0 0 1	A9
07	$(RI,RI_4) \rightarrow BUS$	0 0 0 1 0 0 0 1	11
08	INIT	1 0 1 1 1 1 1 1	BF
09	SAVEG → POP	1 0 0 1 1 1 1 1	9F
0A	AV/INIT	0 0 1 1 1 1 1 1	3F
0B	OF	0 0 0 1 1 1 1 1	1F
0C	$(RI,RI_4) \rightarrow PAC$	0 1 0 1 0 0 0 1	51
0D	(RI,VI) → PAC	1 1 0 0 0 0 0 1	C1
0E	$RI_4 \rightarrow RI$	0 1 1 1 1 0 0 1	79
0F	(RI,BASC) → PAC	1 1 0 0 0 0 1 1	C3

A_3A_2 — VVI_0

BVI_0

VAD

SRI

VRI_0

$DEST_0$

SVI_0, VVI_1

AVI_0

$DEST_1$

V_4, GVI_0

$DEST_1$	$DEST_0$	destination
0	0	rien
0	1	BIPOP
1	0	BIPAC
1	1	RI

III.5. - Définition des ordres

L'étude de l'algorithme du processeur PINS fait apparaître 25 ordres différents, correspondant à:

- la gestion de la pile de contrôle interne:
  - K+1 -> K,
  - K-1 -> K,
  - N+1 -> N,
  - N-1 -> N,
  - 15 -> N(initialisation)
- les demandes de lecture/écriture en mémoire centrale:
  - LMC,
  - EMC,
  - "1" -> MODE
- le contrôle de l'état des transitions:
  - load ETR,
  - "00" -> ETR,
  - "11" -> ETR,
  - "0" -> DBXP,
  - "1" -> DBXP,
  - "1" -> ETC
- l'accés aux instructions et le contrôle des anticipations:
  - "1" -> ETC,
  - "0" -> TRP,
  - "0" -> REPAC,
  - SEQ,
  - clear BIPAC et BIPOP
- la gestion des suites (dk) et (tk):
  - 0 -> k,
  - k+1 -> k,
  - k-1 -> k
- le fonctionnement des bascules d'erreurs et de sauvegarde:
  - "1" -> ERR1,
  - "1" -> ERR2,
  - "1" -> FINSAUVE

L'existence des décodeurs 4 -> 16 ou 3 -> 8 impose une répartition des ordres qui est en fait guidée par un besoin de parallélisme ou d'indépendance entre plusieurs classes d'odres.

Méthode utilisée

On remplit deux tableaux en classant en tête les ordres qui doivent être commandés simultanément , après avoir regroupé en un seul ordre ceux qui sont toujours commandés simultanément.

Dans le cas du processeur PINS, on obtient le classement suivant:

Deux groupes d'ordres indépendants:

O_1	O_2
décodeur 4→16 validé par CP et FAUX	décodeur 3→8 validé par CP et FAUX
00 Noop	00 Noop
01 k+1 → k	01 LMC
02 k-1 → k	02 EMC
03 K+1 → K	03 N+1 → N
04 K-1 → K	04 N-1 → N
05 0 → k et 1 → DBXP et 1 → BZ	05 0 → TRP et 0 → REPAC
06 0 → DBXP	06 "11" → ETR
07 1 → ETC	07 15 → N
08 1 → MODE	
09 SEQ	
0A clear BIPOP et BIPAC	
0B 1 → FINSAUV	
0C 1 → ERR_1	
0D 1 → ERR_2	
0E "00" → ETR	
0F load ETR (rendu systématique par la sélection de la ROM-TRANS)	

Remarque:

On a pris soin de répartir les ordres en deux groupes de 15 et 7 pour assurer leur indépendance. Le nombre de bits de commande (4 pour 01 et 3 pour 02) peut sembler élevé, mais toute autre solution accroîtrait la complexité de la logique.

III.6. - Problèmes liés au codage des instructions

Pour des raisons d'économie en PROM de décodage, nous avons cherché à rendre les instructions significatives sur 6 bits seulement pour PINS, ce qui est naturel, en ce sens que toutes les instructions d'accés ou de contrôle qui ont un paramétre ont un code de 6 bits. Le processeur PINS ne décode donc que 64 instructions, bien qu'il en existe beaucoup plus (116 au total),

mais certaines d'entre elles représenteront 4 instructions pour le processeur POP. On doit donc regrouper sous le même code de 6 bits jusqu'à 4 instructions qui ont exactement les mêmes caractéristiques pour le traitement de PINS.

Une autre contrainte, également draconienne, provient du fait que les extra- ordres (AVANCE,RECULE,...) ont un paramètre de 4 bits. Si l'on veut les envoyer à POP (et FILE) en une seule fois (1 octet), il faut leur attribuer un code opération de 4 bits, ce qui va "recouvrir" 4 instructions initialement décodées par PINS. Comme il y a 6 extra-codes, on va ainsi "perdre" 24 codes pour les instructions opératives.

Une solution a tout de même été trouvée qui est rèsumée dans les tableaux suivants:

Code	processeur PINS	processeur FILE	processeur POP	processeur PAC
00	INXI(P)	CONTROLE 1	UNAIRE	ERR
01	THEN(m)	"	test VRAI	"
02	WHILE/EXITIF	"	test VRAI/FAUX	"
03	UNTIL	"	test FAUX	"
04	CASE(m)	"	CASE	"
05	ROF+/ROF-	"	ROF+/ROF-	"
06	INCAI/DECAI ..	"	INCAI/DECAI ..	"
07	INCA/DECA ..	"	INCA/DECA ..	"
08	$UNAIRE_0$	UNAIRE	4 UNAIRES	"
09	$UNAIRE_1$	"	"	"
0A	$UNAIRE_2$	"	"	"
0B	$UNAIRE_3$	"	"	"
0C	$UNAIRE_4$	"	4 UNAIRES	"
0D	$UNAIRE_5$	"	"	"
0E	$UNAIRE_6$	"	"	"
0F	FIELD	"	"	"
10	INDX	CONTROLE 2	code-inv.	"
11	FO	"	"	"
12	ESAC	"	"	"
13	INTER	"	"	"
14	FORUP(m)	"	FORUP	"
15	FORDOWN(m)	"	FORDOWN	"
16	AFFA/AFFX	"	AFFA/AFFX	"
17	code-inv.	"	code-inv.	"
18	$BINAIRE_0$	BINAIRE	4 BINAIRES	"
19	$BINAIRE_1$	"	"	"
1A	$BINAIRE_2$	"	"	"
1B	$BINAIRE_3$	"	"	"
1C	$BINAIRE_4$	"	4 BINAIRES	"
1D	$BINAIRE_5$	"	"	"
1E	$BINAIRE_6$	"	"	"
1F	$BINAIRE_7$	"	"	"

Code	processeur PINS	processeur FILE	processeur POP	processeur PAC
20	AFFECT(s,d)	AFFECTPOP	AFFECT/CLRV	AFFECT(s,d)
21	PARAM(s,d)	"	PARAM	PARAM(s,d)
22	CLRV (s,d)	"	INCR	AFFECT(s,d)
23	INCV (s,d)	"	DECR/SETV	AFFECT(s,d)
24	HALT	NOOP	SAVEG	SAVEG
25	EXITFOR	"	EXITFOR	code-inv.
26	ENTER	"	ENTER-EXT	ENTER
27	RETURN	"	RETURN-EXT	RETURN
28	OF(m)	OF(n)	OF	code-inv.
29	WITH(n)	"	"	WITH(n)
2A	CALL(s,d)	"	"	CALL(s,d)
2B	EXIT	"	"	code-inv.
2C	CALLH(S,D)	REST(n)	REST	CALL(S,D)
2D	WAIT	"	"	code-inv.
2E	SETV (s,d)	"	"	AFFECT(S,D)
2F	DECV (s,d)	"	"	AFFECT(s,d)
30	NAME(s,d)	AVANCE(n)	AVANCE	NAME(s,d)
31	INDD(s,d)	"	"	INDD(s,d)
32	ENDLOOP	"	"	code-inv.
33	TRAP	"	"	"
34	ZERO	INIT(n)	INIT	LIT8
35	NEHT(m)	"	"	code-inv.
36	SS(L,ad)	"	"	LITI
37	ELSE	"	"	"
38	LIT8	RECULE(n)	RECULE	LIT8
39	LIT16	"	"	LIT16
3A	LITI	"	"	LITI
3B	LITS	"	"	LITS
3C	ONE	SAUVE(n)	SAUVE	LIT8
3D	GOTO(m)	"	"	code-inv.
3E	LOOP(m)	"	"	code-inv.
3F	SETV	"	"	LITI

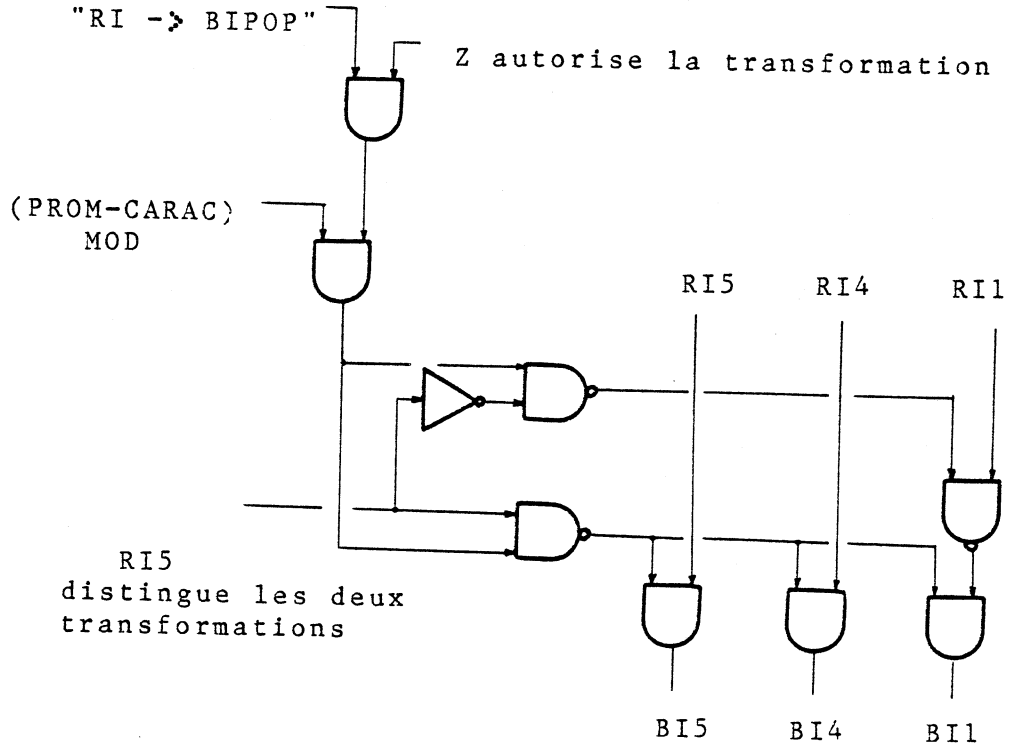
Le deuxième tableau fait apparaître une exception à la relative simplicité de transmission des codes initiaux vers POP et PAC.

En effet, les codes '2C' (CLRV) et '2F' (SETV) ne peuvent être envoyés à POP qui les reconnaîtrait comme un extra-ordre REST (restauration de FILEVAL après CALLF).
Il faut donc les transformer, pour les "projeter" sur la classe AFFECTPOP entre les codes 20 et 23. De plus, cette transformation de code doit être automatique, sans intervention d'un microprogramme particulier: on introduit donc une caractéristique supplémentaire dans la PROM des caractéristiques, appelée MOD, qui, lorsqu'elle vaut UN, force la transformation du code au moment où l'instruction est transférée de RI vers BIPOP.

Tableau de la transformation:

code	Instruction	code POP
20xx	AFFECT(S,D)	200x
21xx	PARAM(S,D)	210x
22xx	INCR(S,D)	220x
23xx	DECR(S,D)	230x
2Cxx	CLRV(S,D)	201x
2Fxx	SETV(S,D)	231x

Réalisation de la transformation:



Z	RI->POP	MOD	RI5	BI5	BI4	BI1
0	x	x	x	RI5	RI4	RI1
1	0	x	x	"	"	"
1	1	0	x	"	"	"
1	1	1	0	"	"	0
1	1	1	1	0	0	1

IV.7. -Interface entre PINS et PME

Le processeur PINS utilise la mémoire (zone CTX) pour initialiser son état et stocker sa pile de contrôle d'imbrication des segments (de contrôle). Il se présente avec la priorité 3, son automate d'accès aux instructions ayant la priorité 4 (la plus faible).

Il dispose d'un registre d'adresse et de deux couples de registres de données pour la lecture (RML1,RML0) et pour l'écriture (RME1,RME0). Il envoie deux ordres LMC et EMC pour déposer soit une demande de lecture, soit une demande d'écriture. Il attend la réponse de la mémoire sur deux signaux:

- AMC indique que RADM est libre (la demande précédente a été prise en compte).
- VMC indique que la donnée lue en mémoire est disponible dans (RML1,RML0).

Les deux ordres LMC et EMC, les deux signaux AMC et VMC, constituent l'interface de contrôle entre le microprogramme de PINS et un automate de dialogue avec PME que nous décrivons ci-après.

L'envoi d'un ordre LMC ou EMC a pour effet de générer une demande DEM3 vers le processeur PME et de mettre soit à "0" soit à "1" une bascule Read/Write qui indiquera le type d'opération mémoire à effectuer. Lorsque PME décide en fonction des priorités d'allocation, d'allouer la mémoire à PINS, il envoie un signal BMA3 qui a pour effet:

- de mettre RADM sur le bus adresse relative pour effectuer le calcul de l'adresse absolue,
- de mettre à "1" une bascule qui validera la prise en compte du signal suivant.

Le signal suivant, appelé DEBOP, est envoyé par PME et reconnu par l'automate qui soit validera le chargement de (RML1,RML0), soit mettra (RME1,RME0) sur le bus mémoire.

Le signal AMC passe à "1" en fin de cycle de BMA,
le signal VMC passe à "1" en fin de cycle DEBOP.

Finalement, l'automate peut être initialisé par le signal de remise à zéro générale (RAZG) ou remis dans son état initial en cas d'erreur (débordement de zone) par le signal INT-PME.

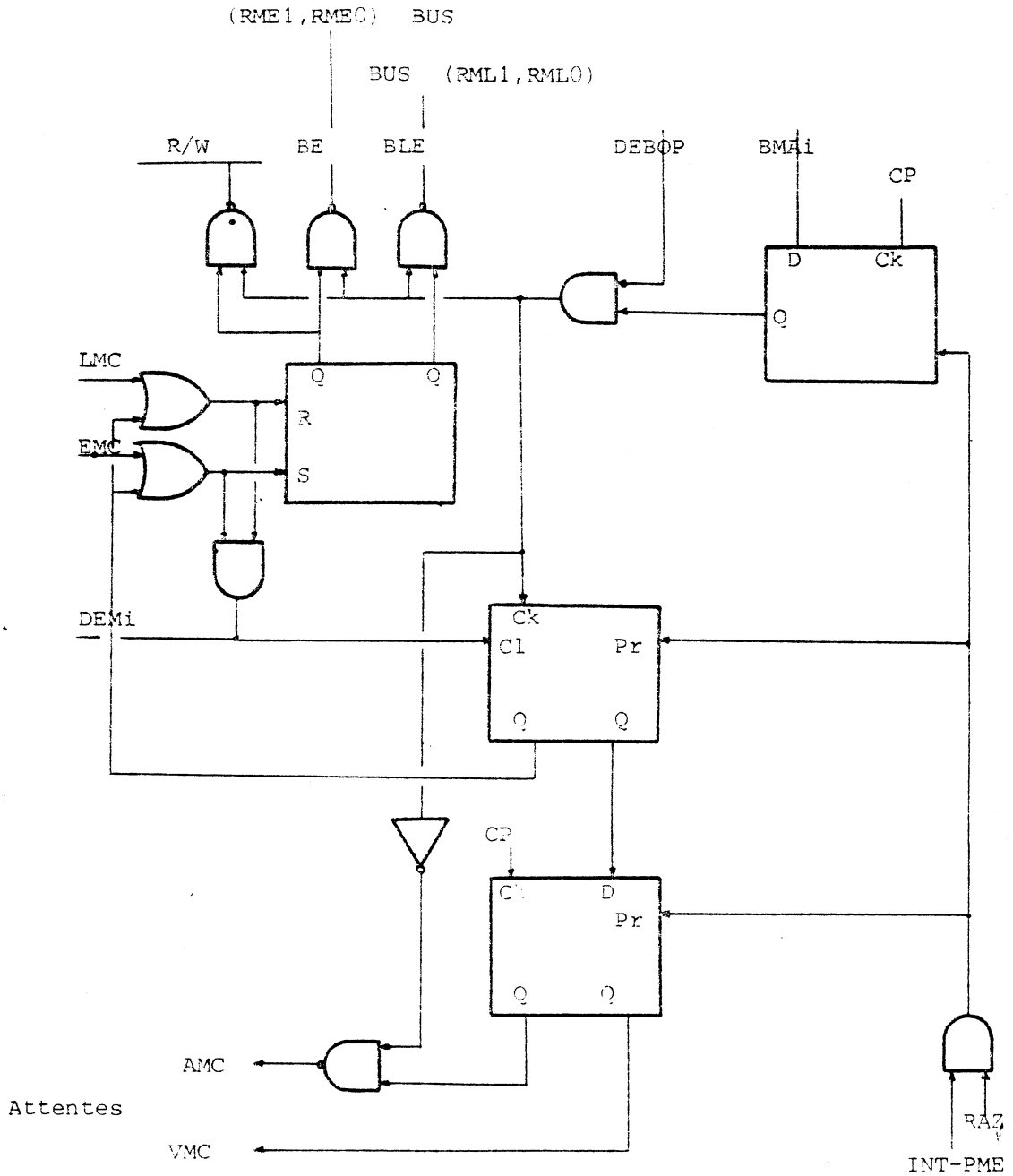


Figure -Interface avec le processeur mémoire

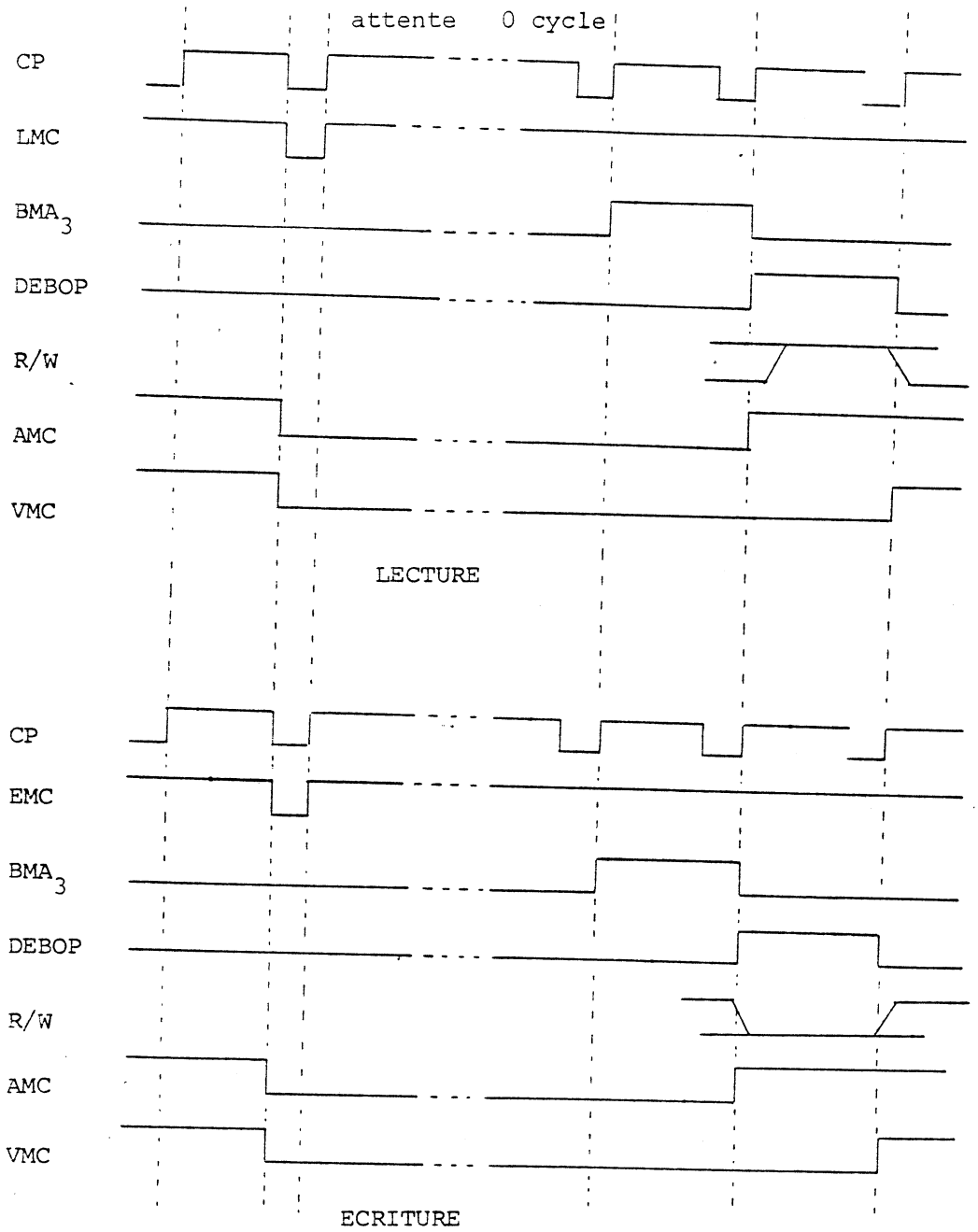


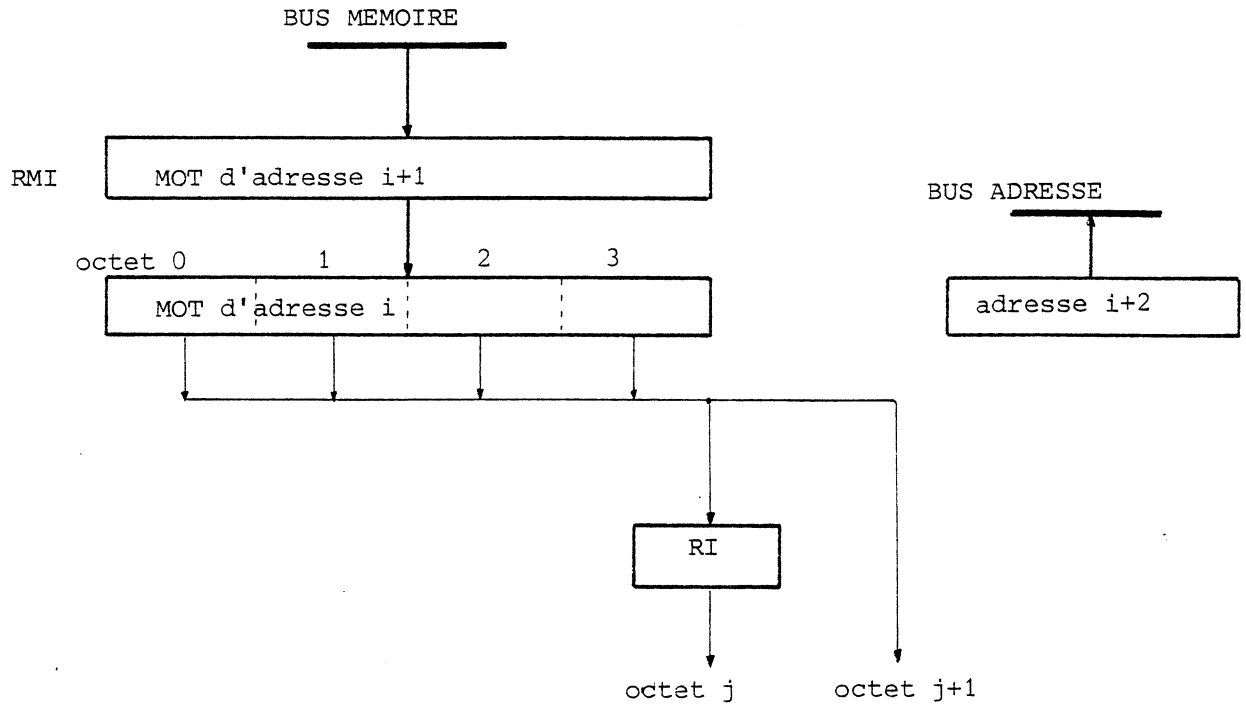
Figure -Interface PME : diagramme des temps

IV.8. - Automate d'accès aux instructions

L'accés aux instructions présente un caractère séquentiel à double titre:

- on accède séquentiellement aux octets d'un même mot mémoire,
- on accède séquentiellement à des mots mémoire.

Un premier mécanisme, dit de désérialisation des octets dans un mot, doit être défini, ainsi qu'un automate qui accède, d'une maniére anticipée, aux mots suivants en mémoire.
Cet ensemble constitue une structure "pipe-line" , matérialisée par la présence d'un registre RMI, contenant le mot suivant (i+1) et d'un registre RADI contenant l'adresse du mot qui suit le mot suivant (i+2), le mot courant (i) étant dans RI4, et l'octet décodé dans RI.

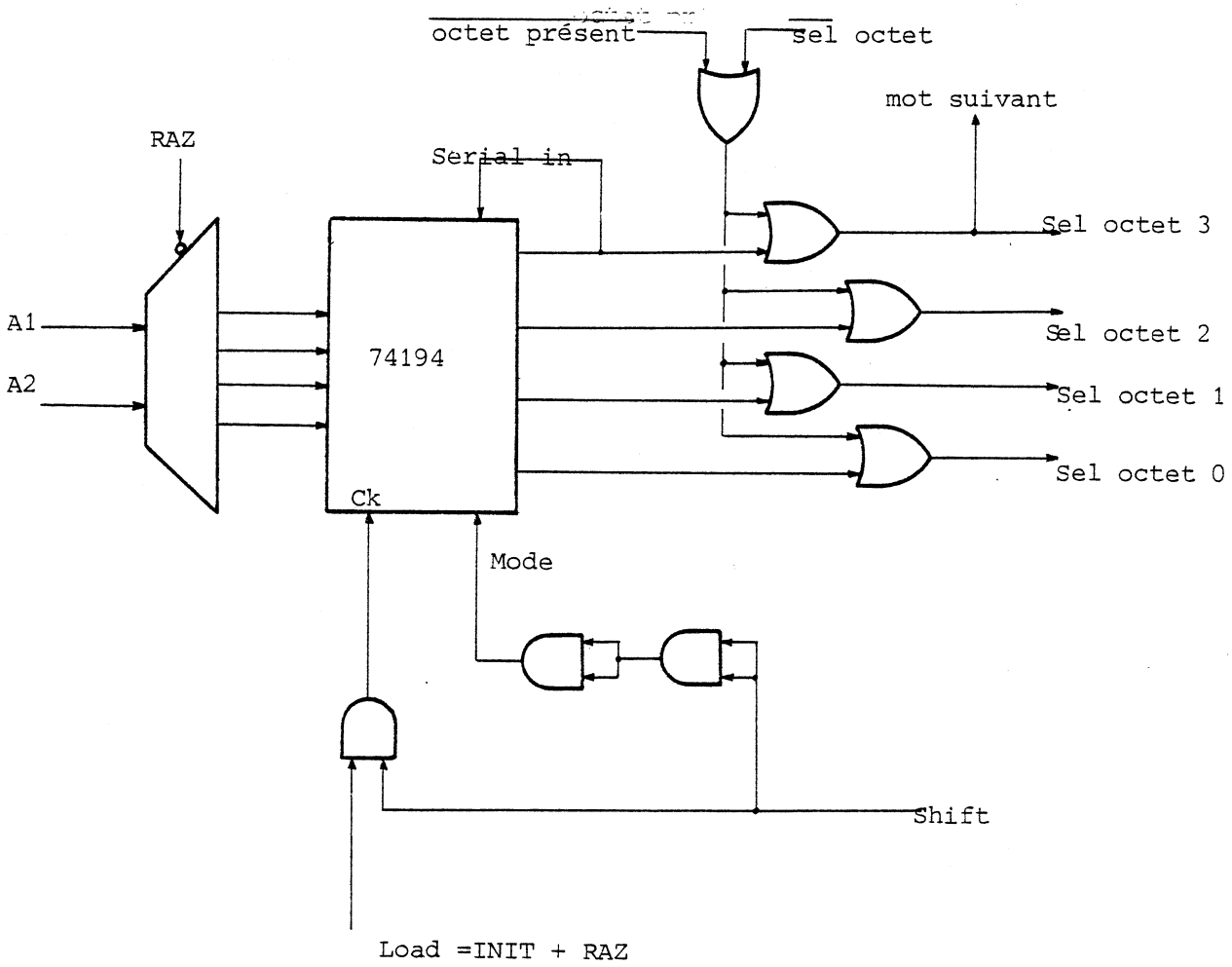


La performance de ce chemin de données suppose que RMI puisse être "passant" pour l'initialisation et que RADI soit un compteur pour automatiser son incrémentation. La sélection des différents octets contenus dans RI4 doit par ailleurs être rendue simple et le fait de référencer le dernier octet (n.3) de RI4 doit déclencher le transfert de RMI dans RI4 et la demande de lecture du mot suivant.

IV.8.1. Mécanisme de désérialisation des octets

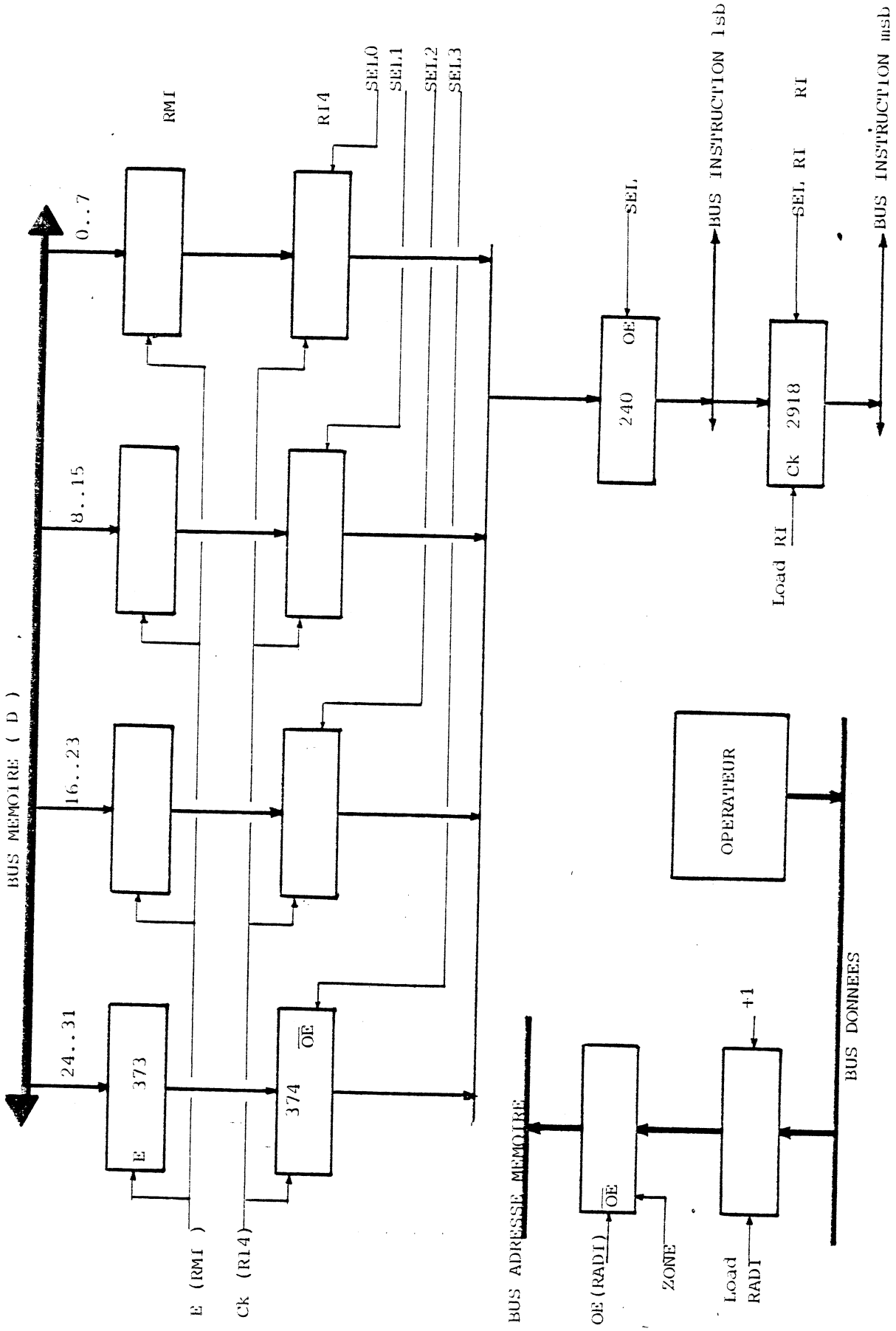
Le microprogramme de PINS tient à jour le compteur ordinal qui contient l'adresse de l'octet qui suit l'instruction en cours. Lors d'une initialisation ou d'une rupture de séquence, une nouvelle adresse d'octet est calculée par le microprogramme. Les deux bits de poids faible donnent le numéro du premier octet à décoder, les 14 bits de poids fort l'adresse du mot qui contient cet octet.
Ces derniers sont chargés dans RADI et une demande de lecture est lancée.
Quant aux 2 bits de poids faible, ils sont chargés, après avoir été décodés, dans un registre à décalage dont les quatre sorties sélectionnent un octet parmi les quatre qui composent RI4. En décalant ce registre, de 0 vers 3, on accède séquentiellement aux 4 octets.
Lorsque le quatrième et dernier octet est référencé par le microprogramme, on doit déclencher le transfert du mot suivant dans RI4.

La réalisation de ces fonctions est faite par le montage suivant:



Remarque:

Le décodeur (74139) est invalidé lors d'une remise à zéro (RAZ) qui provoque ainsi un chargement du vecteur (1,1,1,1) dans le registre à décalage, de manière à éviter d'avoir une sélection parasite de l'octet n.3.


BUS MEMOIRE (D)
0..7
8..15
16..23
24..31
RMI
R14
SEL0
SEL1
SEL2
SEL3
E
373
374
OE
E (RMI)
Ck (R14)
SEL
240
OE
BUS INSTRUCTION lsb
SEL RI
RI
Ck 2918
Load RI
BUS INSTRUCTION msb
OPERATEUR
+1
BUS DONNEES
BUS ADRESSE MEMOIRE
OE (RADI)
ZONE
Load
RADI

IV.8.2. - Automate de lecture en mémoire: ordre INIT

Cet automate est initialisé par un ordre INIT, envoyé par le microprogramme qui charge en même temps une nouvelle adresse dans RADI. Cet ordre peut arriver alors qu'une précédente demande est en cours.
Il faut donc distinguer quatre cas à l'occurrence d'un ordre INIT:

a/ aucune demande en cours,

b/ demande en cours non encore prise en compte,

c/ demande en cours et adresse prise en compte (BMA=0),

d/ lecture en cours (DEBOP=1).

Cas a/:

L'ordre INIT produit une demande de lecture.

Lorsque BMA passe à zéro, la bascule BMA est mise à zéro, de telle sorte que le DEBOP arrivant au cycle suivant soit pris en compte.

Pendant le cycle de DEBOP, le registre RMI est passant (c'est un LATCH), et un ordre de chargement de RI4 est envoyé: NEXTOC peut passer à "1" au début du cycle suivant, indiquant au microprogramme que l'octet demandé est présent.

Dès que RI4 a été chargé, RMI peut être considéré comme vide: une nouvelle demande de lecture est générée et l'adresse est incrémentée (ordre +1 (RADI)).

Cas b/:

Réception d'un ordre INIT alors qu'une demande a été faite.

Si la demande n'a pas été prise en compte (BMA=1 et DEBOP=0), il est inutile d'en refaire une autre, puisqu'il en existe une en attente dans l'allocateur de PME, il suffit de modifier l'adresse et d'attendre l'arrivée du BMA.

Cas c/:

L'ordre INIT est envoyé pendant un cycle de BMA: il est alors trop tard pour changer d'adresse ; il faut par contre masquer le BMA pour éviter que le DEBOP qui arrive forcément au cycle suivant ne soit considéré comme la bonne lecture. Il suffit alors de générer une nouvelle demande de lecture.

Cas d/:

L'ordre INIT est envoyé pendant un cycle de lecture (DEBOP=1). Il

faut invalider cette dernière lecture et refaire une demande.

IV.8.3. Automate de lecture: ordre SEQ

Entre deux ordres INIT, le microprogramme accède séquentiellement aux octets présents dans RI4.

L'automate doit détecter le fait que le microprogramme utilise le dernier octet de RI4, c'est-à-dire lorsqu'il valide la lecture d'un octet et que cet octet porte le numéro 3 (signal L03).

Dans ce cas, si RMI a été chargé avec le mot suivant, son contenu est transféré dans RI4 et une nouvelle demande de lecture est faite.

Dans le cas contraire, il faut attendre que la demande en cours soit satisfaite, comme dans le cas a/ de l'INIT.

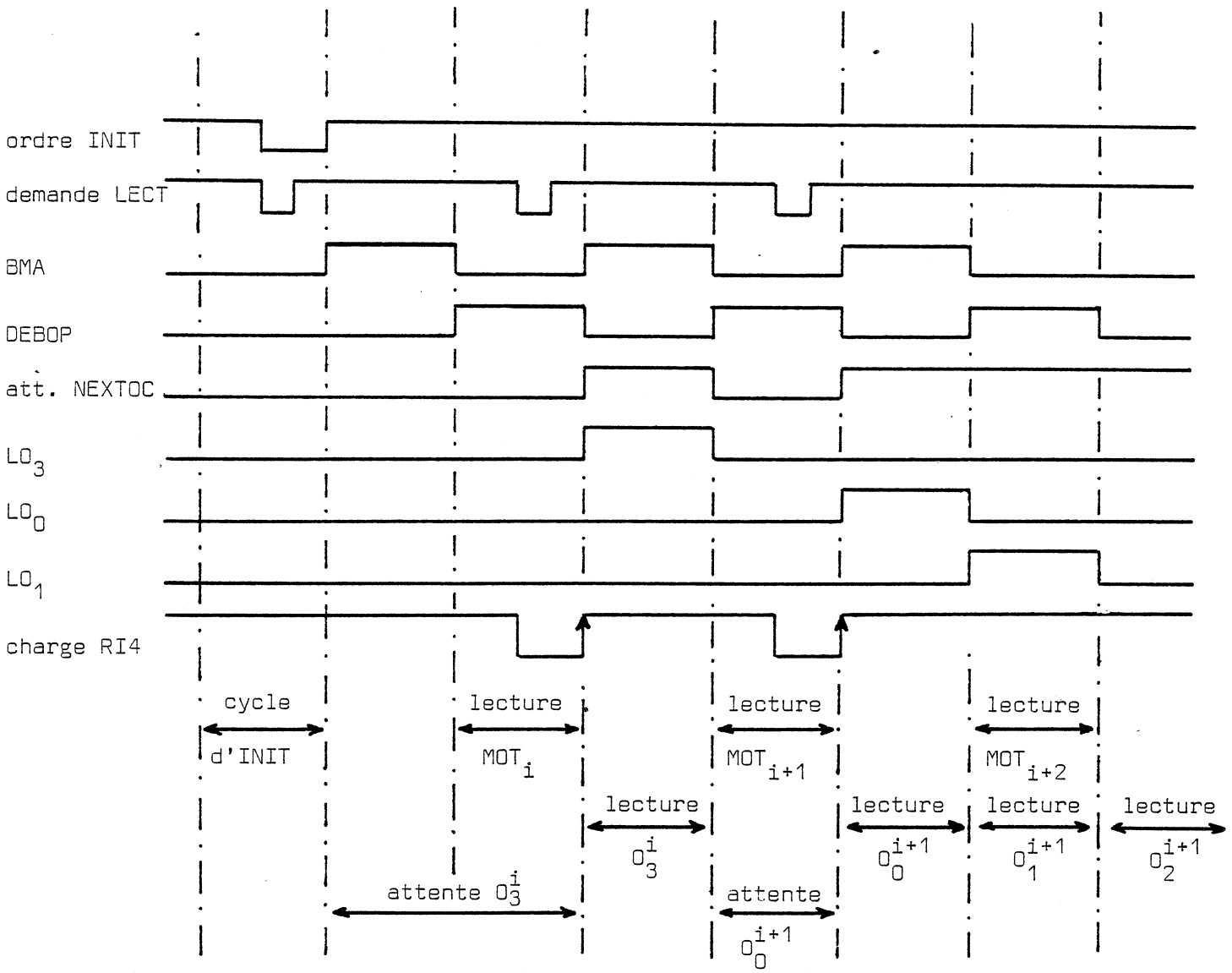
Cas particulier:
Lorsqu'un ordre INIT spécifie une adresse multiple de 3, l'automate génère trois demandes de lecture:

- la première pour lire le premier mot, dont seul le troisième octet sera utilisé

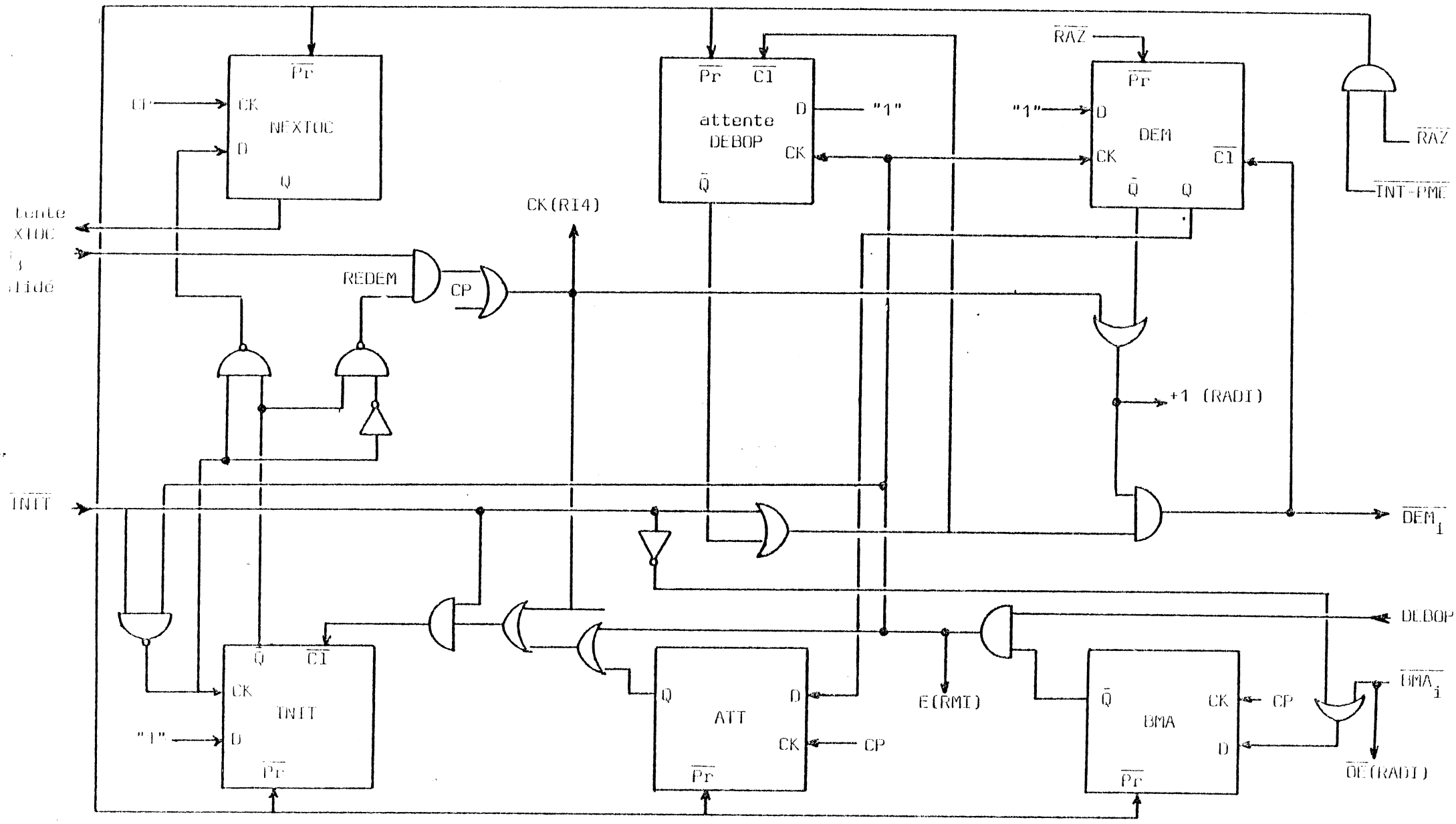
- dès que ce mot est chargé dans RI4, une deuxième demande est faite et,

- dès que le troisième octet du premier mot est utilisé par le microprogramme, RMI est transféré dans RI4 et une troisième demande de lecture est envoyée.

Dans le diagramme qui suit, on suppose qu'il n'y a pas d'attente sur la mémoire.


ordre INIT
demande LECT
BMA
DEBOP
att. NEXTOC
LO3
LO0
LO1
charge RI4
cycle
d'INIT
lecture
MOTi
lecture
MOTi+1
lecture
MOTi+2
lecture
O3i
attente O3i
attente
O0i+1
lecture
O0i+1
lecture
O1i+1
lecture
O2i+1

Automate d'accès aux instructions



Chapître V

DESCRIPTION TECHNIQUE DU PROCESSEUR D'ACCES AUX OPERANDES

V.1. - Calcul de l'adresse d'un descripteur

Ce calcul est provoqué par l'exécution d'une instruction de type NOM(S,D) envoyé par PINS à travers BIPAC.

Il doit se faire à partir du doublet (niveau lexicographique S, déplacement D) codé sur les 10 bits poids faible de l'instruction (voir le rappel du mode d'adressage dans le chapître I).

L'élément D du doublet indique un déplacement par rapport à une base dont le numéro est donné par l'élément S.

L'adresse relative de 16 bits du descripteur à fournir au processeur mémoire PME sera donc:

base de n. S + déplacement D

Les macrocomposants Am 2901 sont bien adaptés à ce type de calcul:

- ils sont cascadables et il est facile d'obtenir un opérateur travaillant sur 16 bits,

- une partie de leurs 16 registres internes à double accés peut contenir la valeur des bases,

- l'opération RAM(A) + D -> Y ne demande qu'un microcyle.

RAM(A) est le contenu de la mémoire interne d'adresse A, D est le bus d'entrée de l'opérateur, Y est sa sortie.

On peut donc espérer avoir un calcul d'adresse performant si l'on est capable de fournir :

- 4 bits sur l'entrée A de l'opérateur représentant un numéro de base (soit S*) et

- 16 bits sur son bus d'entrée (soit D*).

On a vu que S pouvait occuper 2 ou 3 bits tandis que D en occupe 8 ou 7. Il faut que les transcodages:

S -> S* et D -> D*

se fassent automatiquement pour ne pas pénaliser les temps de calcul, c'est-à-dire le temps d'accés aux opérandes, et finalement les performances globales de PASCHLL.

On doit donc interposer de la logique purement combinatoire entre les sorties 0 à 9 de BIPAC et l'opérateur 16 bits.

La solution retenue consiste une fois de plus à utiliser une mémoire morte (PROM) contenant la matrice de décodage. Elle est adressée par les bits 6, 7, 8 et 9 de l'instruction et elle fournit en sortie:

- un numéro de registre de base S*

- un bit de signe permettant l'extension du déplacement sur 16 bits.

Le codage de cette PROM est donné dans le tableau 1.

adresse				sorties						observations
RI9	RI8	RI7	RI6	S*3	S*2	S*1	S*0	D*8 à D*15	D*7	
0	0	0	0	1	0	0	0	0	0	niveau global (déplacement toujours positif)
0	0	0	1	1	0	0	0	0	0	
0	0	1	0	1	0	0	0	0	1	
0	0	1	1	1	0	0	0	0	1	
0	1	0	0	1	0	0	1	0	0	positif niveau 2
0	1	0	1	1	0	0	1	1	1	négatif
0	1	1	0	1	0	1	0	0	0	positif niveau 3
0	1	1	1	1	0	1	0	1	1	négatif
1	0	0	0	1	1	0	0	0	0	positif niveau 4
1	0	0	1	1	1	0	0	1	1	négatif
1	0	1	0	1	1	0	1	0	0	positif niveau 5
1	0	1	1	1	1	0	1	1	1	négatif
1	1	0	0	1	1	1	0	0	0	positif niveau 6
1	1	0	1	1	1	1	0	1	1	négatif
1	1	1	0	0	0	0	1	0	0	relatif à w positif
1	1	1	1	0	0	0	0	1	1	relatif à SP négatif

Tableau 1

Remarque:
Le bit S*3 distingue un adressage relatif par rapport aux registres SP et W d'un adressage relatif par rapport aux bases reflétant la structure de bloc des programmes.

La figure 1 montre l'insertion de la PROM de décodage dans le chemin de données du processeur PAC :

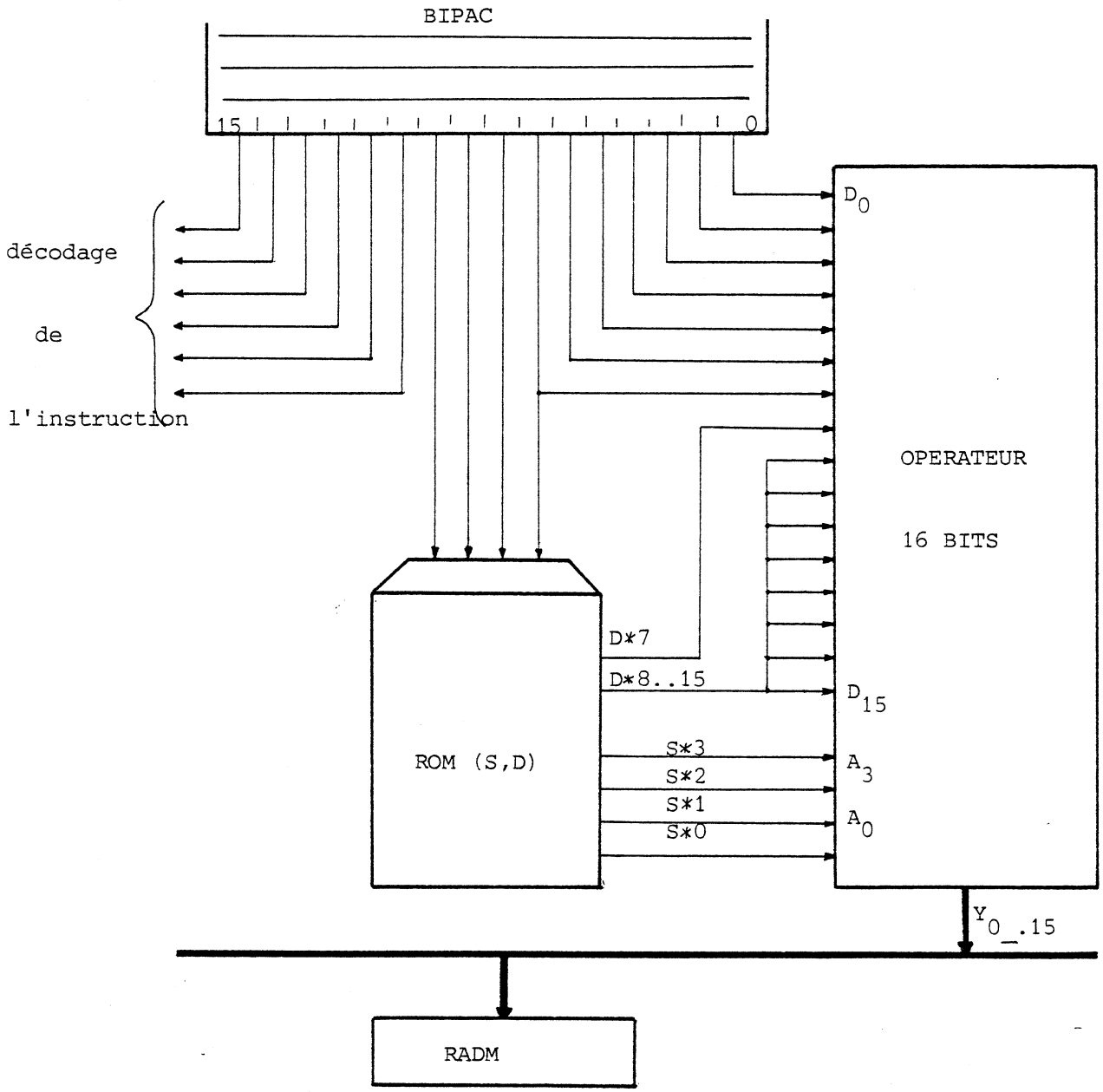


Figure V.1. Décodage du nom d'une variable.

V.2. Mémoire associative

La mémoire associative située sur la carte du processeur PAC est en fait la partie associative de la file des dépendances du processeur FILE.

Elle est utilisée par les deux processeurs,

- en mode associatif par PAC et
- en mode lecture/écriture par FILE.

Il faudra donc résoudre un problème d'allocation et de synchronisation.

V.2.1. Description

L'occurence d'une instruction d'accès de type NOM (SD) entraîne une recherche sur le nom du descripteur (SD).

Si ce nom existe dans la mémoire associative PAC n'aura pas besoin de lancer une lecture en mémoire centrale parce que soit:

- le descripteur de la variable est présent dans la file de dépendances (variable cachée),
- soit FILE trouvera dans la file des dépendances un pointeur dans FILEVAL sur un descripteur de dépendances.

PAC doit avertir FILE du résultat de sa recherche par CODEPAC en positionnant sa demande (DPAC <-1).

CODEPAC se compose de trois bits :

- Un bit issu de BIPAC permettant de distinguer une instruction du type NOM SD, d'une instruction de type AFFECT SD.
- L'indicateur TROUVE venant de la logique de la mémoire associative et indiquant le résultat de la recherche par le SD.
- Un bit RESOLU de la mémoire associative permettant de distinguer une variable "cachée", d'une variable "sous dépendance".

PAC doit en outre fournir à FILE l'adresse à laquelle le nom du descripteur de la variable a été trouvé .
Cette adresse correspond à la position dans la file des dépendances soit

- du descripteur (TROUVE = 1 et RESOLU = 1), soit

- d'un pointeur sur un descripteur de dépendances dans la file d'évaluation (TROUVE = 1, RESOLU = 0).

Cette adresse ACAM-OUT de 4 bits est envoyée au bus pointeur.

D'autre part, une instruction AFFECT SD entraîne l'initialisation d'une dépendance, c'est-à-dire l'écriture dans la mémoire associative de ce (S,D) et du bit RESOLU à une adresse transmise par FILE par ACAM-IN.

On rappelle qu'une dépendance est résolue quand POP exécute l'affection correspondante.
Pour cela POP écrit la nouvelle valeur de la variable en mémoire centrale et il la transmet à FILE qui l'écrit en FILEVAL(P2) puis remonte éventuellement la chaîne de reprise.
Ensuite FILE doit marquer cette dépendance comme étant résolue (écriture du bit RESOLU = 1 dans la mémoire associative) après avoir recopié le résultat dans FILEDEP à l'adresse correspondante.

L'emploi d'une mémoire associative de 16 mots de 12 bits

10 pour le SD
\+ le bit RESOLU
\+ l'indicateur de présence VIDE)

permet donc de résoudre de façon originale le problème des dépendances lié à toutes machines pipe-line en empêchant l'accès à des descripteurs non à jour.

La solution choisie permet d'utiliser le problème des dépendances comme un avantage.
Elle permet une diminution assez sensible du nombre d'accès mémoire et donc une amélioration du fonctionnement global de PASC-HLL, d'autant plus que la consultation de la mémoire associative se fait en parallèle avec le calcul de l'adresse du descripteur et elle dure comme celui-ci un microcyle si bien que si le SD n'existe pas dans la mémoire associative PAC lance immédiatement une demande de lecture mémoire.

Cet accès mémoire n'est absolument pas pénalisé par la recherche associative au point de vue temps d'exécution.

En conclusion, au plus le programme à exécuter comporte d'affectation de variables simples réutilisées par le programme (ce qui est fréquent : si on modifie la valeur d'une variable c'est pour utiliser cette nouvelle valeur un peu plus tard), au plus les performances de la machine seront intéressantes.

C'est le paradoxe de PASC-HLL.

V.2.2. Réalisation de la mémoire associative

Un mot de la mémoire associative est constitué par :

- 10 bits codant le nom symbolique de la variable (S,D)

- le bit RESOLU distinguant une variable sous dépendance d'une variable "cachée"

- le bit VIDE (indicateur de présence) permettant l'initialisation de la mèmoire associative (marquage de tous les mots à vide) et servant à lever les ambiguités comme par exemple 2 affectations successives à une même variable simple.

En fait la recherche associative se fait sur 11 bits (S,D + VIDE).
Le bit RESOLU est seulement lu pour être envoyé à FILE par CODEPAC.

description matérielle

Le composant utilisé pour réaliser la mémoire associative est le boîtier I3104 (Content Adressable Memory) qui est constitué par une matrice de 4 mots de 4 bits.
On utilise donc un un tableau de 3x4 pour avoir 16 mots de 12 bits.

Cette mémoire peut aussi être utilisé de façon classique, avec cependant la particularité d'avoir un adressage linéaire.

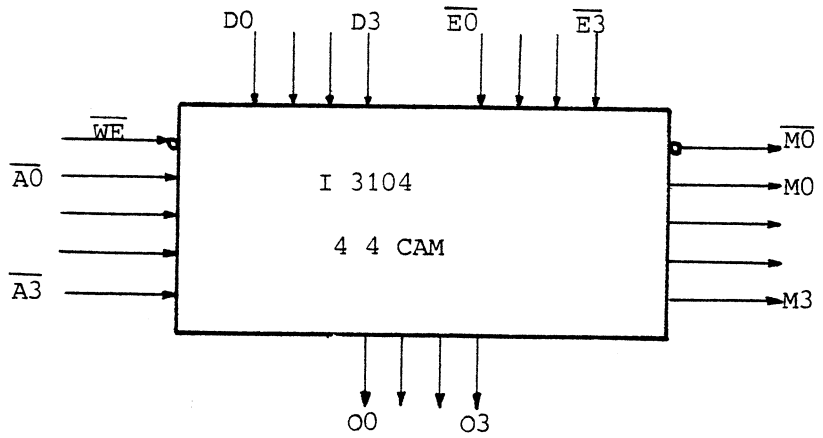


Figure V.2. boîtier Content Adressable Memory.

V.2.3. Nature des opérations sur la mémoire associative

Initialisation

Au début de l'éxécution d'un programme, la CAM doit être initialisée en forçant l'indicateur VIDE à la valeur "1" pour tous les mots.
Sachant que l'on peut écrire en parallèle dans plusieurs mots, il suffit de générer une adresse égale à 00....0 en entrée de la CAM.

Cette adresse peut être obtenue après une recherche associative en ne validant aucun des bits. On obtient ainsi une correspondance pour tous les mots, donc un vecteur de sortie M égal à 11....1, qu'il suffit da complémenter.

Algorithme d'initialisation

1. - Recherche assiciative avec $\overline{Ei}$ = 1 pour i = 0...11
   donne Mj = 1 pour j = 0...15

2. - Ecriture avec Aj = Mj pour j = 0...15, $\overline{E11}$ = 0 et D11 = 1

Ecriture d'un nouveau NOM dans la CAM

Lorsque le processeur PAC reçoit une instruction AFFECT (SD), il demande au processeur FILE de créer un descripteur de dépendance. Ce dernier dispose d'un pointeur appelé NIN, qui indique la position du premier mot libre dans la file des dépendances et la CAM.

Le processeur FILE doit donc écrire (SD, RESOLU, VIDE) à l'adresse indiquée par le pointeur NIN. Cette adresse codée sur 4 bits est décodée par un circuit décodeur 4 vers 16 (74 154), dont la sortie est telle que :

$$O_{NIN} = 0 \text{ et } O_j = 1 \text{ pour } j = 0..15 \text{ et } j = NIN$$

Les valeurs des champs S et D proviennent de la sortie du FI-FO d'instructions BIPAC.

On force:

D10 et D11 à 0

et on valide tous les bits:

$\overline{Ei}$ = 0 pour i = 0..11.

Résolution d'une dépendance

Le processeur FILE, à la demande du processeur POP, peut supprimer l'ancienne occurence d'une variable

par D11 = 1 et $\overline{E11}$ = 0

ou bien marquer une dépendance comme étant résolue

par D10 = 1 et $\overline{E10}$ = 0

en écrivant dans la CAM.

L'adresse utilisée pour l'écriture provient d'une recherche associative antérieure, ou est obtenue par décodage de la valeur d'un pointeur.

Recherche associative

Lorsqu'une instruction d'accès est décodée, le processeur PAC recherche si le nom (SD) est présent dans la CAM. Il valide donc les bits 0 à 9 et le bit 11

avec $\begin{cases} \overline{E0} = \ldots\ldots = \overline{E9} \text{ et } \overline{E11} = 0 \\ D0 \ldots D9 = (SD) \text{ et } D11 = 0. \end{cases}$

Le résultat de la recherche est un vecteur de 16 bits. La présence de l'information cherchée est indiquée par l'existence d'un bit égal à 0 dans le vecteur.
Un encodeur de priorité indique cette présence, et permet d'encoder l'adresse sur 4 bits.

Le changement de contexte qui se produit lors d'un appel de procédure (ou de fonction) nécessite une recherche associative sur le champ qui contient le niveau lexicographique S, pour éviter les conflits possibles en cas de récursivité.

Cette recherche est faite en validant les bits 7 à 9 et le bit 11

avec $\begin{cases} \overline{E7} = \overline{E8} = \overline{E9} = \overline{E11} = 0 \\ D7 \ldots D9 = (S) \text{ et } D11 = 0. \end{cases}$

On obtient ainsi un vecteur de correspondance pouvant avoir plusieurs zéros correspondant aux positions des mots qui contiennent des noms de niveau S.

Tous ces mots doivent être marqués "vides"

$\overline{E11}$ = 0 et D11 = 1

en utilisant le vecteur de correspondance comme adresse d'écriture.

Lors d'un passage de paramètres, avant l'entrée dans une procédure, les noms des paramètres sont rangés dans la CAM pour chaque instruction PARAM (SD).

Le niveau lexicographique spécifié fait référence au sommet de pile SP comme registre de base, et le nom SD se présente sous la forme :

9 8 7 6 5 4 3 2 1 0

1 1 1 1 x x x x x x

Sachant qu'à l'intérieur d'une procédure tout paramètre est référencé comme une variable locale de déplacement négatif, son NOM sera différent de celui qui se trouve dans la CAM.
Afin de bénéficier du mécanisme de dépendances qui doit aussi être efficace pour les paramètres, nous allons changer les NOMs de paramètres de 111 en S au moment de l'entrée dans la procédure (instruction ENTER).

On fait une recherche associative en validant les bits 6 à 9 et en affichant

D6 = .. = D9 = 1.

Puis on modifie les bits 7 à 9, dans les mots où il y a correspondance, par

$\overline{E7}$ =...= $\overline{E9}$ = 0

et

D7...D9 = NIVEAU.

Nature des opérations	Validation des bits					Entrées données				
	$\overline{E0..5}$	$\overline{E6}$	$\overline{E7..9}$	$\overline{E10}$	$\overline{E11}$	D6	D7..9	D10	D11	
Recherche (S,D,VIDE)	0	0	0	1	0	BI6	BI1.9	0	0	
Recherche (S, VIDE)	1	1	0	1	0	0	NIVEA	0	0	
Recherche (IIII,VIDE)	1	0	0	1	0	1	111	0	0	demandées
Recherche partout	1	1	1	1	1	0	0	0	0	par
Ecriture (VIDE)	1	1	1	1	0	0	0	0	1	PAC
Ecriture (NIVEAU)	1	1	0	1	1	0	NIVEA	0	0	
Ecriture (S,D,VIDE, RESOLU)	0	0	0	0	0	BI6	BI7.9	0	0	
Ecriture VIDE	1	1	1	1	0	0	0	0	1	demandées par
Ecriture (RESOLU)	1	1	1	0	1	0	0	1	0	FILE

- Récapitulation des opérations sur la CAM

V.2.4. Réalisation matérielle des opérations sur la CAM

- Le chemin de données des adresses.

Le problème est de rendre l'adressage linéaire (16 lignes non décodées) de la CAM compatible avec le bus pointeur du processeur FILE véhiculant des adresses codées de 4 bits.

Une adresse émise par FILE doit être décodée pour pouvoir adresser la CAM par son vecteur d'entrée d'adresse $\overline{A0}...\overline{A15}$.
On a vu qu'il était intéressant que cette CAM puisse être adressée par son vecteur de correspondances (M0...M15) lors des opérations d'initialisation notament.

Les logiques utilisées pour le vecteur $\overline{Ai}$ et le vecteur Mi sont complémentaires, une couche d'inverseur est donc nécessaire entre les Mi et les $\overline{Ai}$.

Le multiplexage de l'adressage de la CAM se fait de la manière suivante :

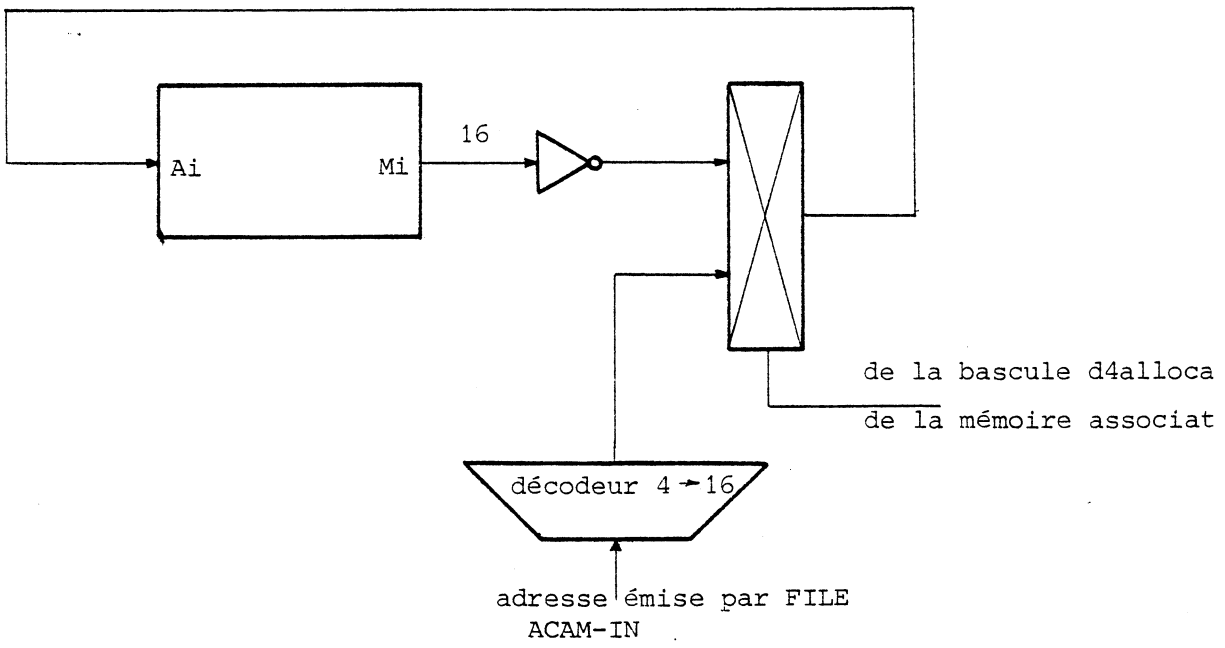


Figure V.3. multiplexage d'adressage CAM.

Dans l'autre sens, l'adresse émise par la CAM (vecteur Mi) doit être encodé avant d'être envoyée à FILE via le bus pointeur. Pour ce faire on va utiliser deux boîtiers encodeur de priorité qui génèrent en outre l'indicateur TROUVE qui constituera un des bits

de CODEPAC envoyé à FILE lors d'une demande de PAC.

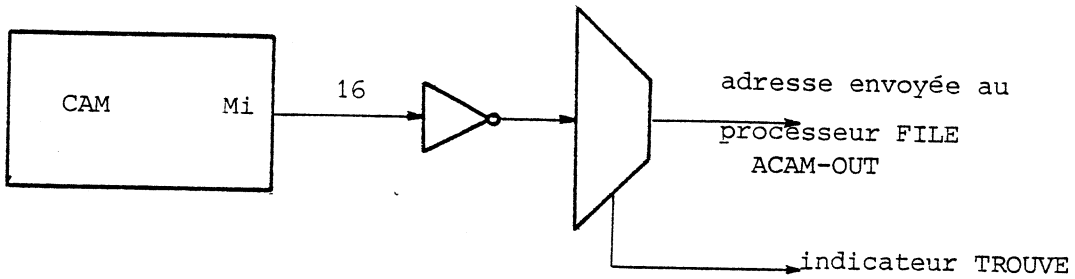


Figure IV.4. Résultat d'une recherche.

Remarque :

Un tampon est nécessaire car la valeur au temps i du vecteur de correspondance M0...15 peut être utilisée au temps i + 1 comme adresse A0...15 par exemple pour initialiser la mémoire ou pour marquer à VIDE tous les mots contenant un certain niveau lexicographique.

Un seul circuit (SN74LS298) regroupe la fonction de multiplexage deux vers un et la fonction de mémorisation pour 4 bits.

Le décodage 4 -> 16 est réalisé à l'aide d'un boîtier SN 74 154.

L'encodage est légèrement plus complexe : il n'existe que des encodeurs 8 sur 3 dans les catalogues de fabricants de circuits intégrés.

Pour réaliser l'encodage 16 vers 4 il faudra deux de ces circuits (SN 74 148) plus un boîtier de quatre portes NAND à deux entrées (SN 74 00).

Ces circuits sont associés de la manière suivante :

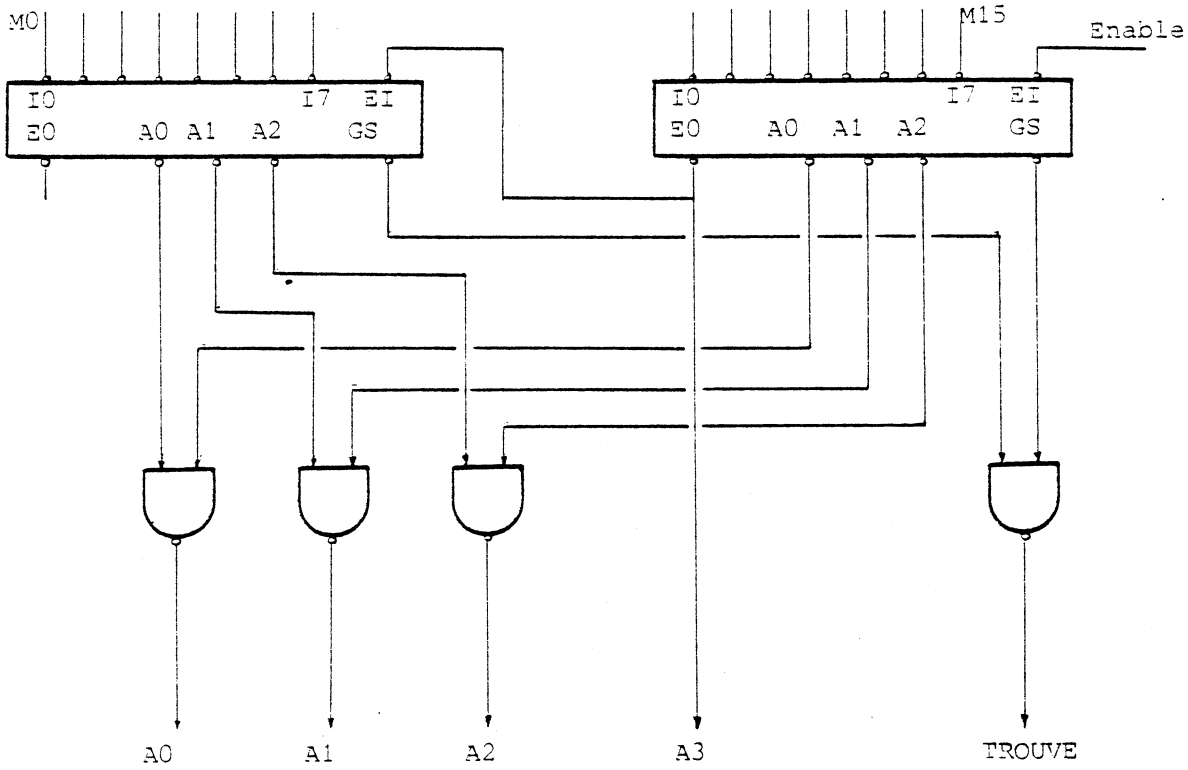


Figure V·5 - Encodeur de priorité 16 vers 4

V.2.5. Commandes et contrôle de la CAM

- Définition du problème :

On a vu que la CAM était partagée par les processeurs PAC et FILE. Il faut donc résoudre tout d'abord un problème d'allocation.

PAC doit avoir la possibilité de commander six opérations différentes sur la CAM, trois bits de microinstruction lui seront nécessaire pour coder l'une d'entre elle.

FILE a trois types d'opérations à demander à la CAM, deux bits lui suffisent pour les coder (PCAM1 et PCAM2).

Pour commander la CAM il faut d'une part générer le signal $\overline{WE}$ et d'autre part générer le masque de 12 bits (vecteur $\overline{Ei}$) permettant de modifier (ou d'associer) la totalité ou une partie des bits de la CAM

($\overline{Ei}$ = 1 -> aucune opération de recherche ou d'écriture dans la colonne n. i,

$\overline{Ej}$ = 0 -> recherche ou modification dans la colone n. j).

En outre, en fonction du vecteur $\overline{Ei}$ il faudra sélectionner sur les entrées Di le profil à associer ou à écrire dans la CAM.

Remarque :

Le tableau récapitulatif des opérations sur la CAM montrent les différentes possiblités pour les entrées Di et le masque correspondant $\overline{Ei}$.

On voit que les entrées D0 et D5 sont

- soit les bits 0 à 5 de BIPAC,
- soit n'importe quoi

Les sorties de BIPAC pourront donc être câblées directement sur D0...5.

L'entrée D6 reçoit

- soit BI6,
- soit 1,
- soit n'importe quoi,

tandis que D7 à D9 reçoivent

- soit BI7 à BI9,
- soit 111,
- soit la sortie du registre NIVEAU .

Notons que quand D7...D9 reçoit NIVEAU, D6 peut accepter

n'importe quoi .

Quant à D10 et D11, on peut remarquer qu'ils sont équivalents entre eux et à $\overline{E7}...\overline{E9}$ (si l'on ne tient pas compte des 0).

La commande $\overline{E7}...\overline{E9}$ sera donc câblée sur D10 et D11 .

Quatre multiplexeurs (trois suffiraient) sont suffisants pour les entrées Di de da CAM . Ils seront commandés par deux signaux (S1 et S2) générés par la logique de décodage .

Le probdème de la commande de la CAM consiste à gènèrer les fonctions suivantes :

- $\overline{WE}$
- $\overline{E0}..5$
- $\overline{E2}$
- $\overline{E7}..9$
- $\overline{E10}$
- $\overline{E11}$
- S1
- S2

et ceci à partir de:

- 3 bits de microinstruction de PAC
- 2 bits venant de FILE
- la sortie de la bascule ALLOCAM

Plusieurs solutions s'offrent à nous pour la réalisation de ces fonctions :

- Logique discrète
- circuits MSI
- FLPA
- matrice de décodage (PROM)

Les deux premières solutions ont été écartées car elles nécessitent un trop grand nombre de boîtier pour la première et la génération de 8 fonctions de 6 variables n'est pas très adaptée à l'emploi de circuits MSI (multiplexeurs, décodeurs, ...).

On va examininer les deux autres :

solution FPLA

Ce type de circuit permet de faire de la logique discrète en programmant une matrice ayant des portes ET en lignes et des portes OU en colonnes.
La solution proposée utilise un circuit Am 27971 possédant 16 entrées et contenant 48 produits (ET) intermédiaires.

Les fonctions de logique alléatoire sont implémentées comme 8 grandes portes ET-OU-NON.

Jusqu'à 48 fonctions ET peuvent être générées dans le boîtier dans lequel chaque fonction ET est le produit d'un nombre quelconque d'entrées entre 0 et 16 (I0...I15) ou de leur complément.
Chacune de ces fonctions ET peut être "OU câblée" pour forcer une fonction de sortie sur une des sorties F.
Ces sorties F peuvent aussi être programmées pour être inversées (possibilité de générer les fonctions ET-OU-NON aussi bien que ET-OU).

Les PLAs sont particularisés en claquant électriquement des fusibles pour fabriquer les fonctions nécessaires.

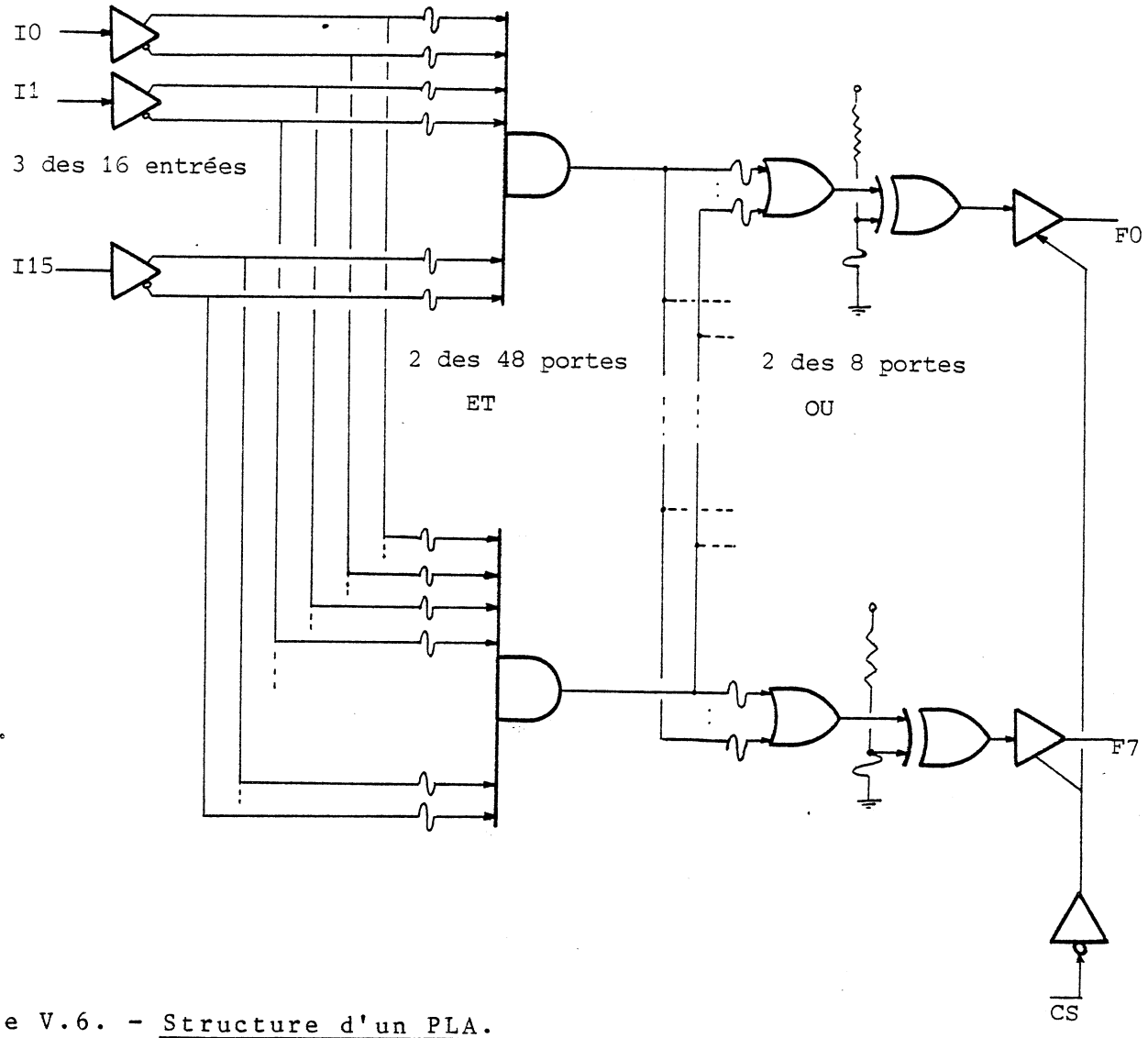


Figure V.6. - Structure d'un PLA.

Si l'on adopte la convention suivante:

- I0 = PCAM0
- I1 = PCAM1 — PCAM0 et PCAM1 codent la requête de FILE

- I2 = PAC0
- I3 = PAC1 — Ces trois bits codent la requête PAC
- I4 = PAC2

- I5 = ALLOCAM

Alors :

- F0 = $\overline{E0}=\overline{E1}=\overline{E2}=\overline{E3}=\overline{E4}=\overline{E5}$

- F1 = $\overline{E6}$

- F2 = $\overline{E6}=\overline{E8}=\overline{E9}$=D11=D12

- F3 = $\overline{E10}$

- F4 = $\overline{E11}$

- F5 = S1

- F6 = S2

avec:

- $\overline{Ei}$ entrée d'inibhition d'opération sur la CAM pour la colonne numéro i
- Di entrée donnée de la mémoire associative
- S1 et S2 commande de la sélection des entrées 6 à 9 de la CAM parmi :

  - les bits correspondants de BIPAC,
  - la sortie du registre NIVEAU,
  - le profil binaire "1111".

Les fonctions à programmer dans le FPLA peuvent s'écrire:

- F0 = $\overline{I5}.\overline{I4}.I3+\overline{I5}.\overline{I4}.I2+\overline{I5}.\overline{I3}.I2+\overline{I5}.\overline{I4}.I3+I5.\overline{I1}.I0+I5.I1.\overline{I0}$
- F1 = $\overline{I5}.\overline{I4}.I3+\overline{I5}.\overline{I4}.I2+\overline{I5}.\overline{I3}.I2+I5.\overline{I1}.I0+I5.I1.\overline{I0}$
- F2 = $\overline{I5}.\overline{I4}.I3.I2+\overline{I5}.I4.\overline{I3}.\overline{I2}+I5.\overline{I1}.I0+I5.I1.\overline{I0}$
- F3 = $\overline{I5}.\overline{I4}+\overline{I4}.\overline{I3}+I5.\overline{I1}.I0$
- F4 = $I5.\overline{I4}.I3.I2+\overline{I5}.I4.\overline{I3}.I2+I5.I1.\overline{I0}$
- F5 = $\overline{I5}.\overline{I4}.I3$
- F6 = $\overline{I5}.\overline{I3}.I2$

On obtient la réalisation matérielle suivante:

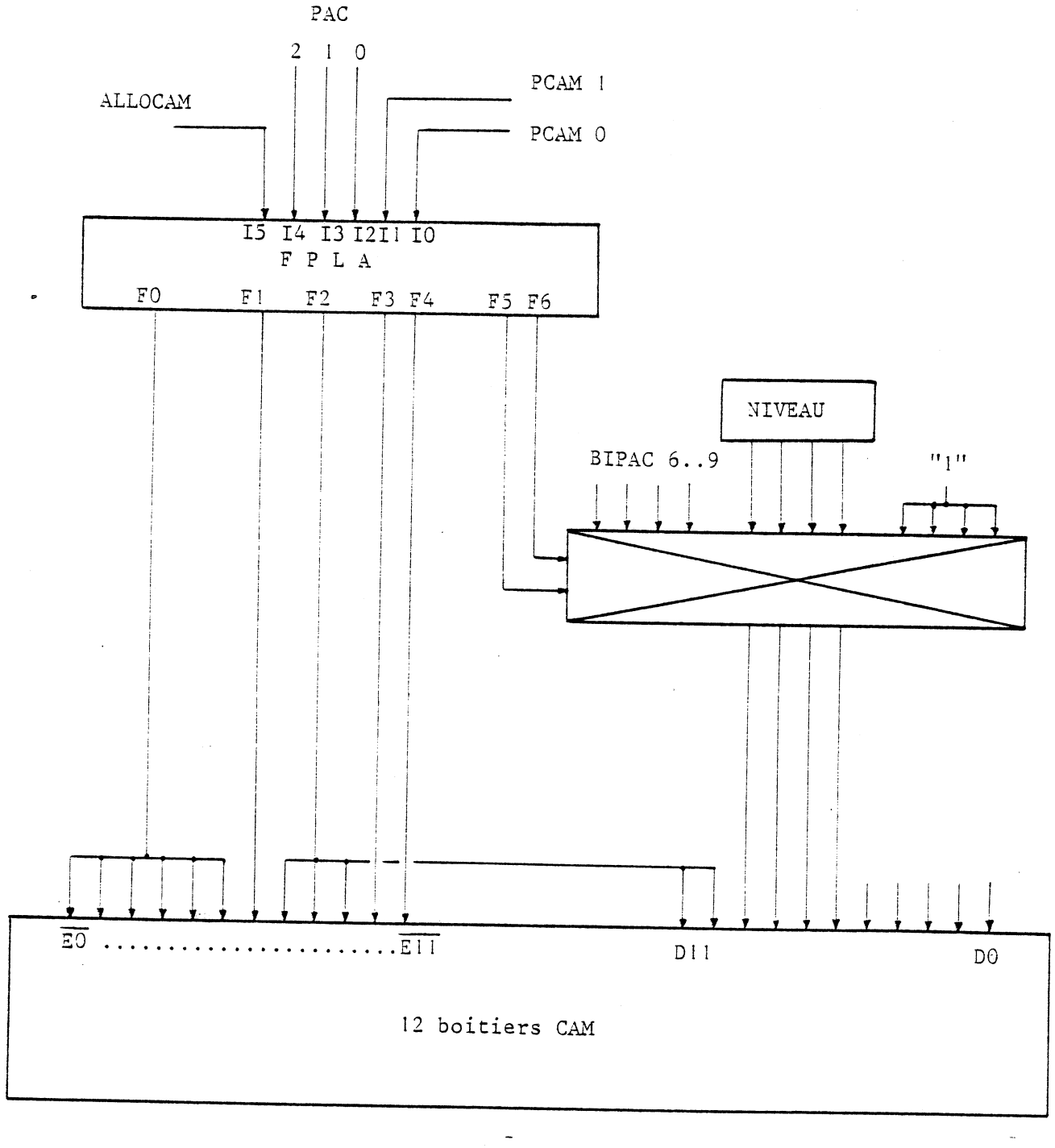


Figure V.7. - Commande de la mémoire associative.
- solution FPLA

Solution matrice de décodage

Le problème consiste à contrôler 9 opérations sur la mémoire associative en générant les huit signaux nécessaires à la commande à partir d'ordres émanant soit du microprogramme de PAC soit de celui de FILE.

La distinction PAC/FILE se fait par la bascule ALOCAM (voir interface PAC/FILE).

Les 3 commandes de FILE sont codées sur 2 bits (PCAM0 et PCAM1).
Les 6 commandes de PAC sont codées sur 3 bits (PAC0, PAC1 et PAC2).

Ces 6 signaux sont nécessaires pour déterminer 1 opération parmi les 9 possibles.
Ce nombre 6 est incompressible du fait que la CAM soit une ressource partagée entre PAC et FILE.

Le problème peut être résolu par une mémoire morte de 64 mots de 8 bits dont seulement 9 mots seront significatifs . Cette réalisation demande 2 boîtiers PROM 74S 288 de 16 broches.

On peut aussi n'utiliser qu'un boîtier PROM et un multiplexeur quadruple 2 vers 1 recevant les paramètres de FILE sur une voie et les paramètres de PAC sur l'autre voie, la sélection se faisant par ALOCAM.

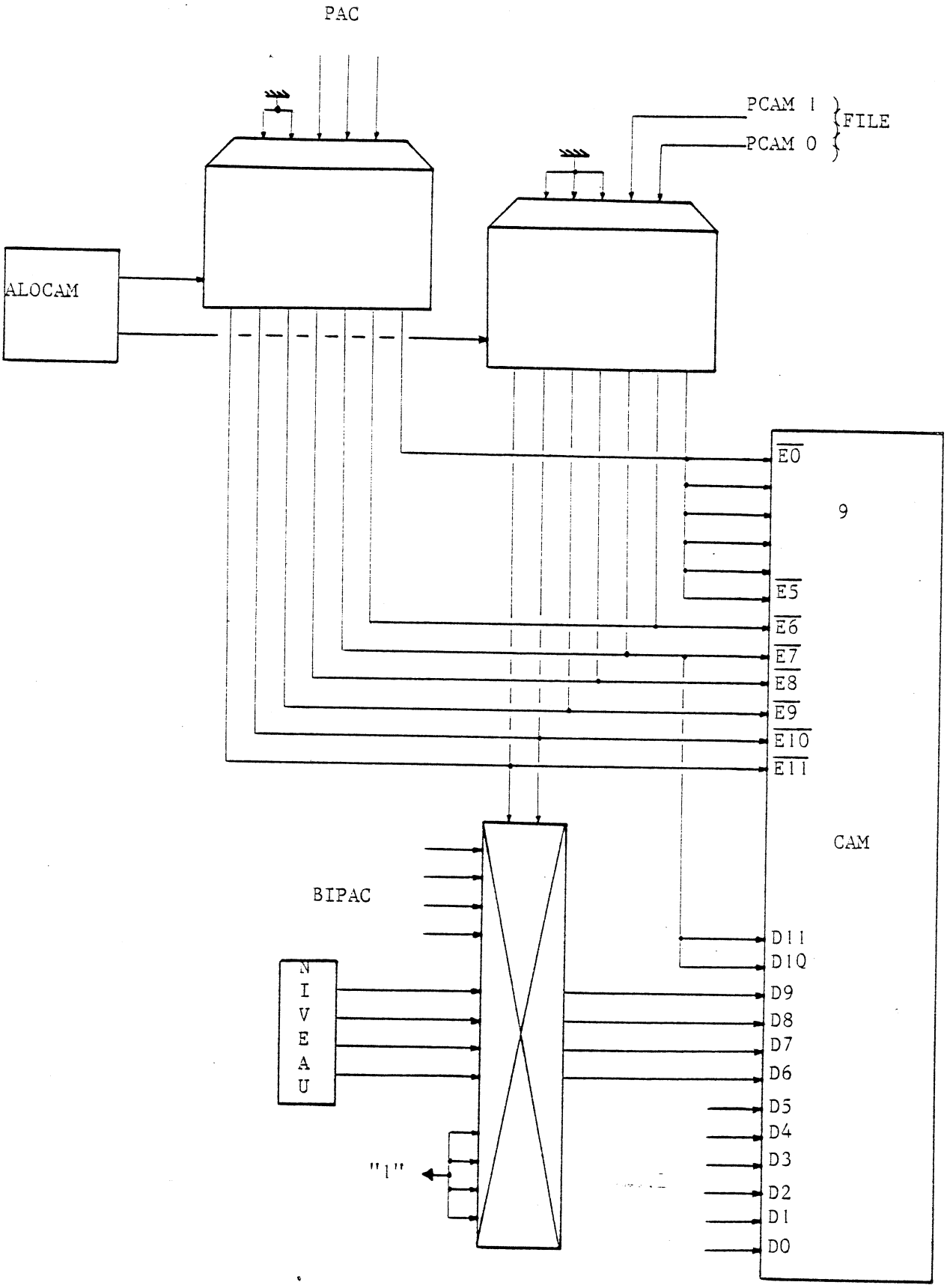


Figure V.8. - Commandes de la mémoire associative.
- solution matrice de décodage avec 2 PROM.

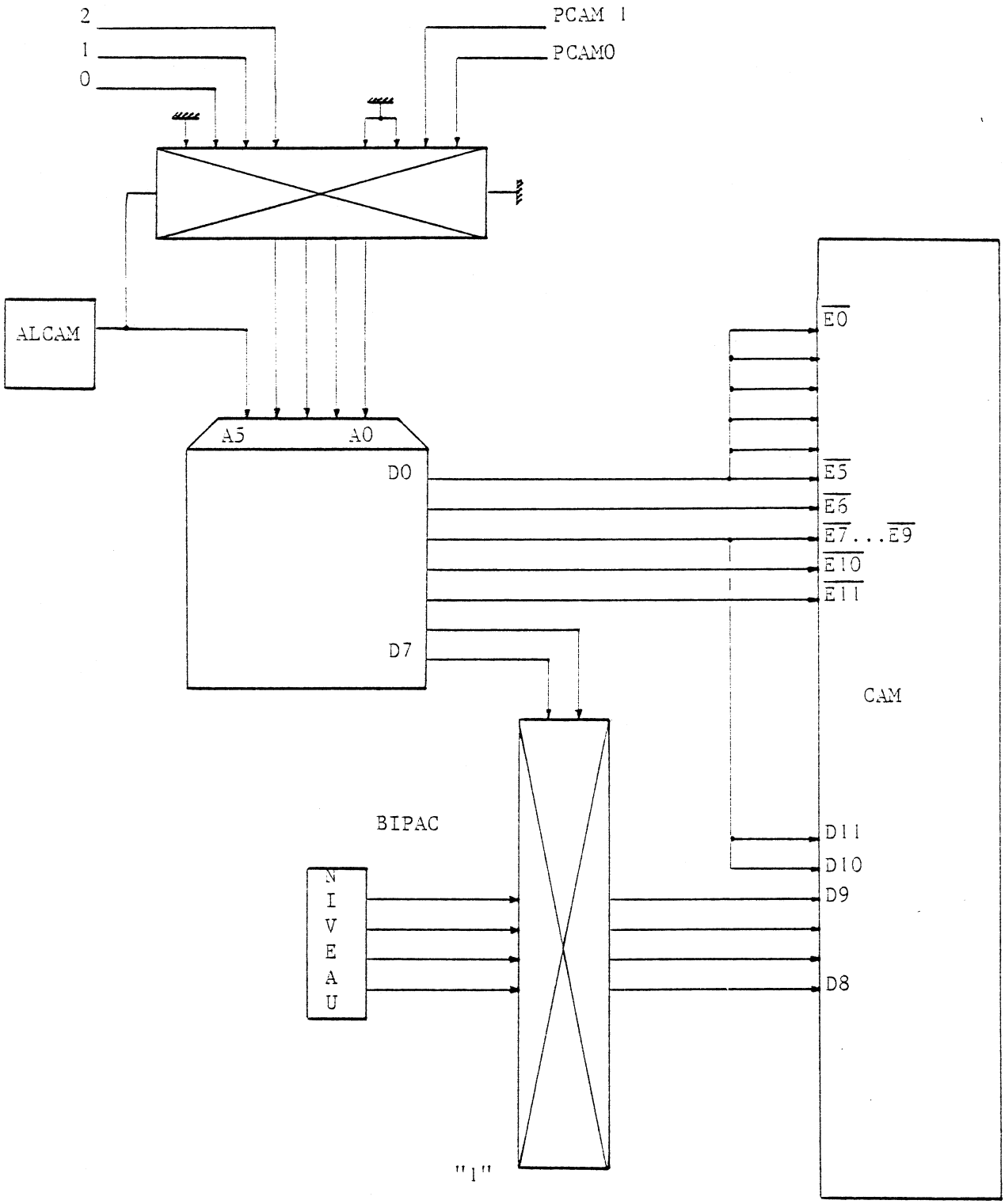


Figure V.9. Commandes de la mémoire associative.
- solution matrice de décodage avec 1 PROM et 1 multiplexeur.

Le codage de la PROM est donné dans le tableau ci-après.

Le câblage global de la CAM, de ses commandes et de son chemin de données d'adresses est indiqué dans la figure suivante.

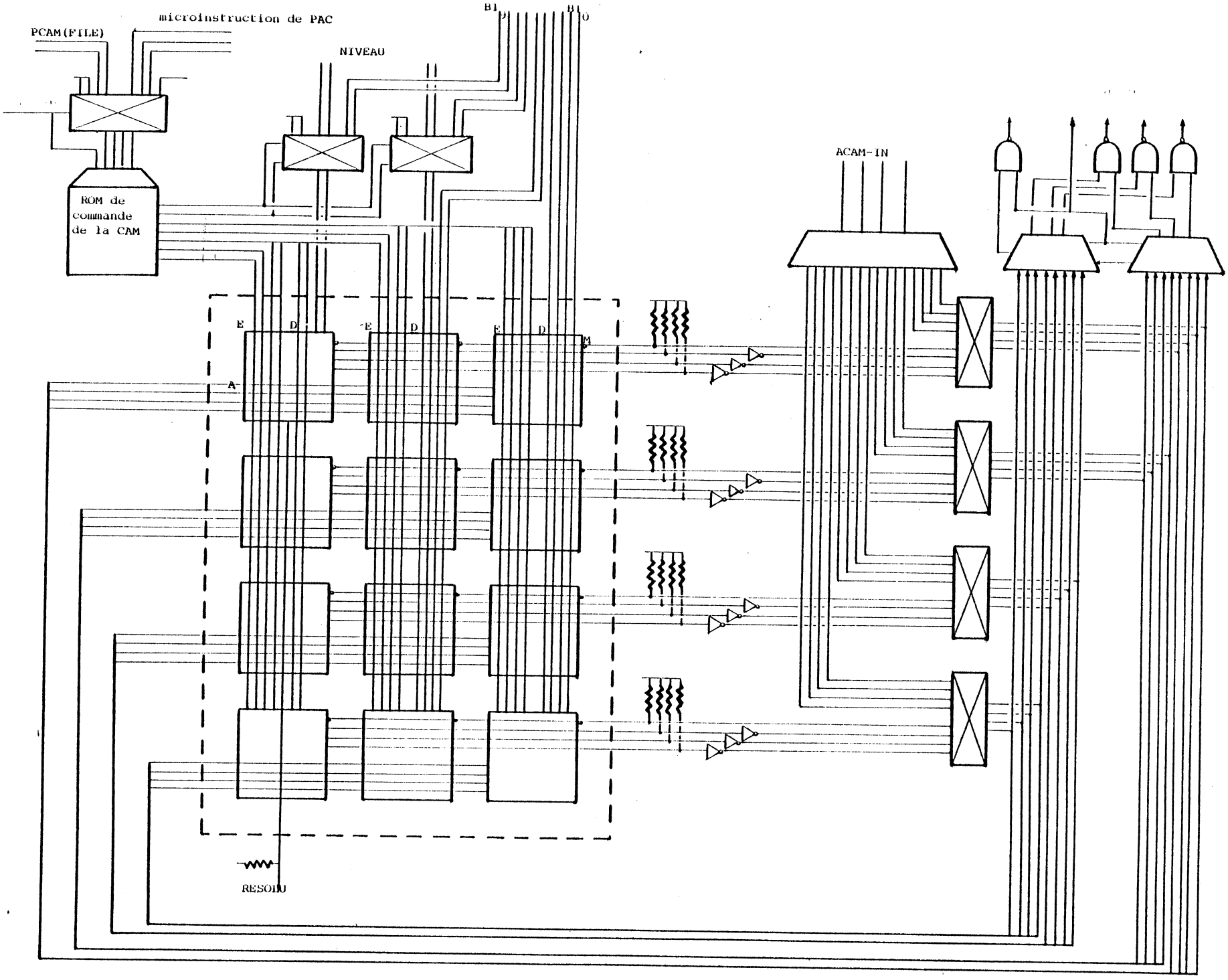
Remarque:
Cinq bits de la microinstruction de PAC ont été utilisés: pour minimiser la longueur de la microinstruction, on a fait un recouvrement de champ:

Si Rmicr.I27.Rmicr.I26 = 1

Alors Rmicr.I1 24 à Rmicr.1 25 représente une commande CAM,

sinon ils servent à coder un ordre.

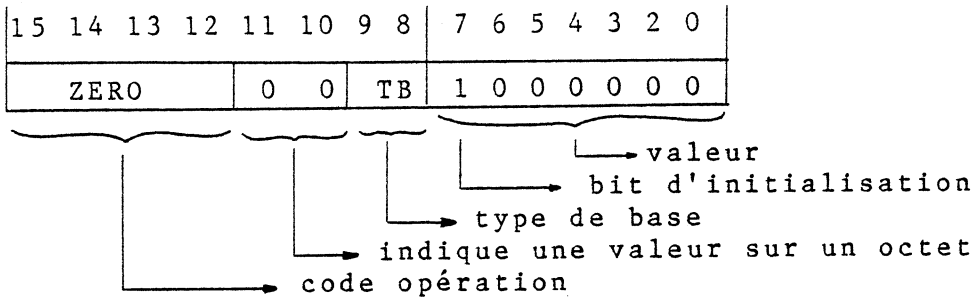
ADRESSES						SORTIES								FONCTIONS
!Alocam	PAC2	PAC1	PAC0	PCAM1	PCAM0	$\overline{E0..5}$	$\overline{E6}$	$\overline{E7..9}$	$\overline{E10}$	$\overline{E11}$	S1	S0	$\overline{WE}$	
0	Ø	Ø	Ø	0	0	0	0	0	0	0	0	0	⊔	ECRITURE (S, D, $\overline{VIDE}$, RES)
0	Ø	Ø	Ø	0	1	1	1	1	1	0	Ø	Ø	⊔	ECRITURE (VIDE)
0	Ø	Ø	Ø	1	0	1	1	1	0	1	Ø	Ø	⊔	ECRITURE (RES)
0	Ø	Ø	Ø	1	1									NOP
1	0	0	0	Ø	Ø	0	0	0	1	0	0	0	1	RECH (S, D, $\overline{VIDE}$)
1	0	0	1	Ø	Ø	1	1	0	1	0	0	1	1	RECH (S, $\overline{VIDE}$)
1	0	1	0	Ø	Ø	1	0	0	1	0	1	0	1	RECH (III, $\overline{VIDE}$)
1	0	1	1	Ø	Ø	1	1	1	1	1	0	0	1	RECH ($\forall$)
1	1	0	0	Ø	Ø	1	1	1	1	1	0	0	⊔	ECR (VIDE)
1	1	0	1	Ø	Ø	1	1	0	1	1	0	1	⊔	ERC (S)
1	1	1	0	Ø	Ø									NOP
1	1	1	1	Ø	Ø									NOP


microinstruction de PAC
PCAM(FILE)
NIVEAU
B1
B0
ROM de commande de la CAM
ACAM-IN
E
D
A
M
RESOLU

V.3. - Construction de valeurs immédiates

Le processeur reçoit dans sa file d'attente des instructions (BIPAC), des instructions d'accés immédiat qui contiennent des valeurs littérales (ou l'adresse de la valeur). Ce sont

- l'instruction ZERO qui apparait dans BIPAC sous la forme:



- l'instruction ONE

15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
ONE 0 0 TB	1 0 0 0 0 0 0 1

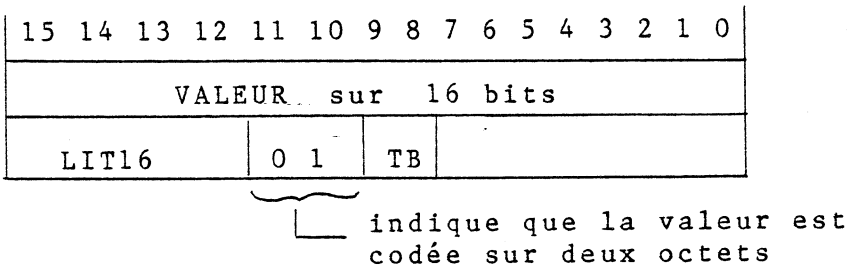
Remarque:
Les instructions ZERO et ONE n'occupent qu'un octet dans la zone CODE en mémoire centrale. Le processeur PINS l'étend sur 16 bits en construisant la valeur à partir du code instruction avant de l'envoyer à BIPAC.

- l'instruction LIT8

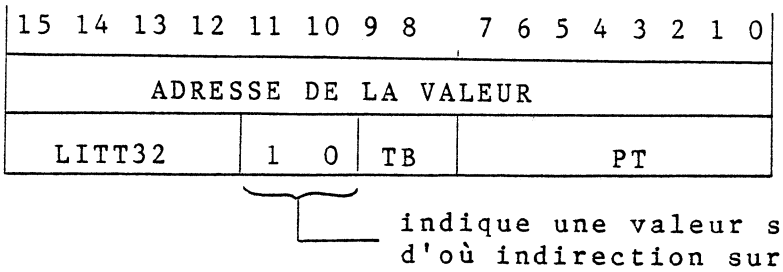
15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
LIT8 0 0 TB	VALEUR

Remarque:
Ces trois instructions demandent une interprétation identique par le microprogramme de PAC.

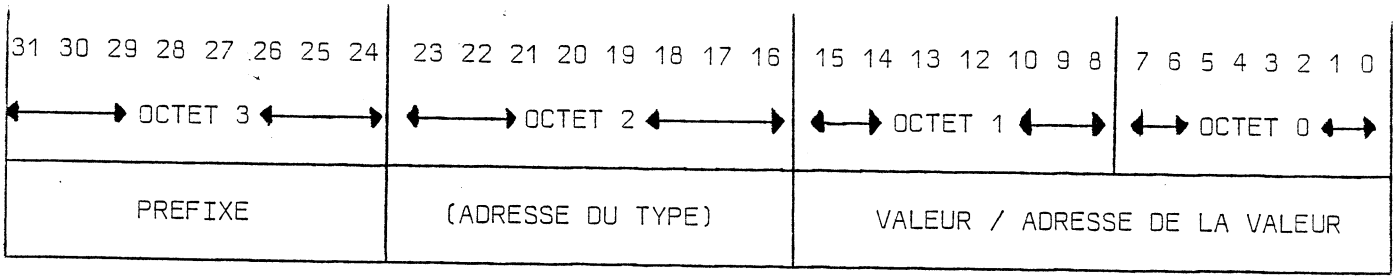
- l'instruction LIT16 qui occupe deux mots de BIPAC.



- l'instruction LIT32 occupant aussi deux mots.

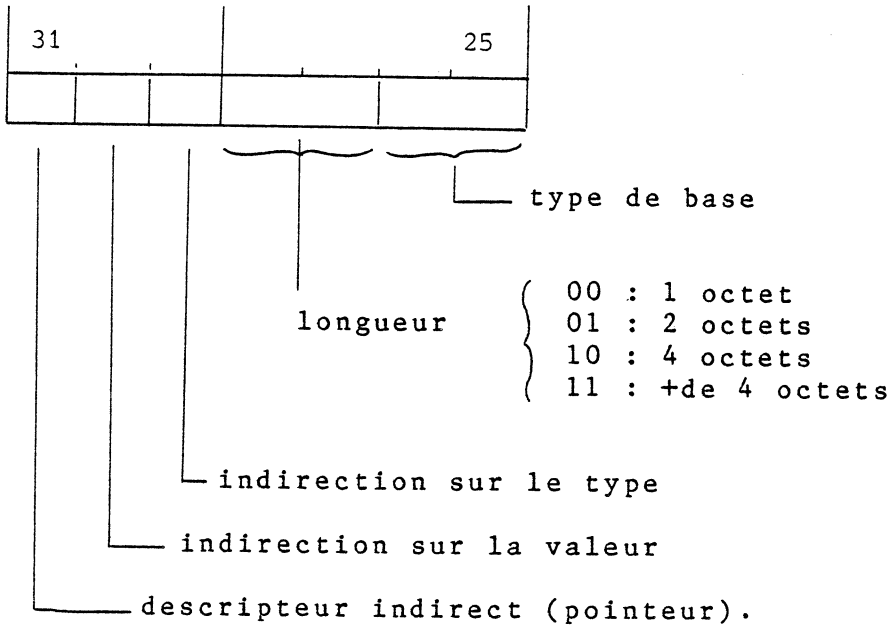


Le travail du processeur d'accés consiste, à partir de telles instructions, à construire un descripteur de variables à envoyer à FILE et dont on rappelle le format:



La valeur (ou l'adresse de la valeur) se trouvant déjà dans BIPAC, on aura seulement à la cadrer convenablement dans le descripteur à construire.

La construction du préfixe pose plus de problèmes. Son format est le suivant:



On peut remarquer que les champs longueur et type de base du préfixe existent déjà dans l'instruction BIPAC. Il s'agit donc de

construire les 4 bits poids fort du préfixe et plus particulièrement les indicateurs d'indirection sur le type et sur la valeur.

En définitive, la construction d'une valeur immédiate se fera de la manière suivante:

- pour une valeur littérale sur 8 bits(instruction ZERO, ONE et LIT8):

à partir de:

BIPAC															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CODOP				0	0	TB		1	VALEUR						

on doit générer:

0	0	0	0	0	0	T	B	0	0	0	0	0	0	0	0

PACE 1

0	0	0	0	0	0	0	0	VALEUR	

PACE 0

Les liaisons BIPAC -> PAC E1 et BIPAC -> PACE0 se font à travers l'opérateur 16 bits de PAC.

Les 0 indiquent des valeurs indéterminées (n'importe quoi), ils peuvent être remplacés par les bits correspondants de BIPAC.

PACE1 reçoit les bits 0 à 11 de BIPAC (les champs significatifs TB et L sont correctement cadrés) et 0000 en poids fort.

PACE0 reçoit la même chose (les 8 bits poids faible de BIPAC forment le champ valeur du descripteur de variable qui se trouvera dans l'octet 0 du descripteur).

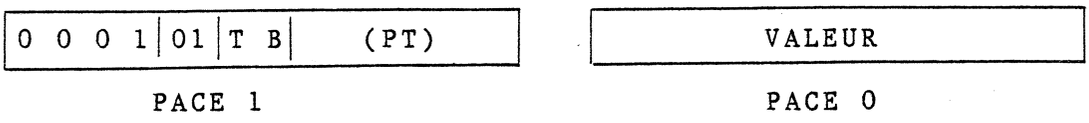
Ce chargement des buffers de communication de descripteur entre PAC et FILE peut se faire en une ou deux fois suivant le type de décodages adopté pour les commandes de destination du bus de sortie de l'opérateur.

- pour une valeur littérale sur 16 bits:

A partir de:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VALEUR SUR 16 BITS															
LIT 16				0	1	TB									

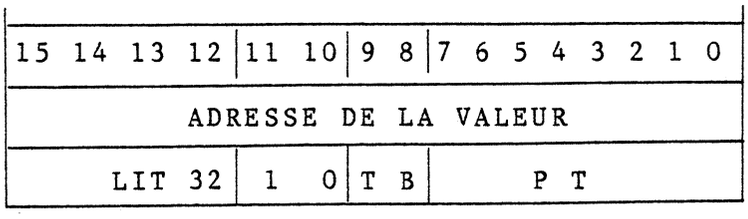
on doit générer:



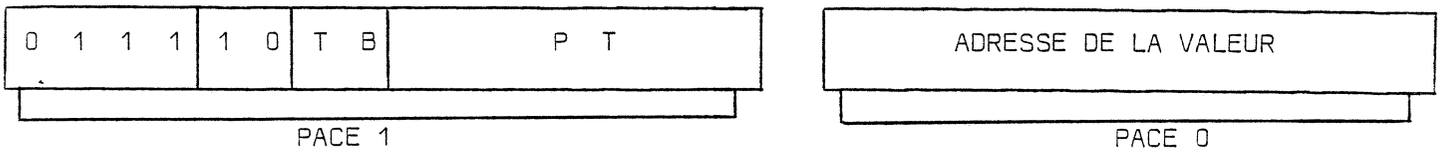
PACE 1 reçoit donc en poids faible les 12 bits poids faible de BIPAC et le profil 0001 sur les poids forts, tandis que PACE 0 reçoit le mot suivant issu de BIPAC. Il faudra bien évidemment faire un contrôle de présence à ce niveau là (BIPAC/VIDE).

- Pour une valeur littérale de 32 bits

A partir de :



Il faut générer :



C'est-à-dire que PACE 1 reçoit BIPAC sur ses 12 bits poids faible étendus avec le profil 0111 en poids fort tandis que PACE 0 recevra le prochain mot issu de BIPAC (la remarque précédente reste vraie).

Il s'agit de multiplexer au niveau des 4 bits poids forts de l'entrée D de l'opérateur de 16 bits les profils à construire pour les préfixes à envoyer à PACE 1 avec les 4 bits poids fort de BIPAC pour le changement de PACE 0 et de commander à l'opérateur l'instruction "PASS" en utilisant D comme origine d'opérande.

V.4. Gestion de la pile de contexte

Pour gérer la pile de contexte PASCHLL dispose de trois instructions envoyées au processeur PAC qui sont :

- l'instruction CALL qui prépare le passage des paramètres sans changer le contexte d'adressage,

- l'instruction ENTER qui change le contexte et met à jour les chaînages statiques et dynamiques dans la marque de bloc,

- l'instruction RETURN terminant l'exécution de toute

procédure.

- L'instruction CALL

Elle est paramétrée par le n. de segment (= S) associé à la procédure dont le descripteur se trouve dans la table des segments au déplacement = S par rapport au début de la zone CODE ou EXTERNE selon le mode.

Le descripteur de segment contient l'adresse du bloc de description des paramétres de la procédure dont le premier mot contient :

- le nombre des paramètres (NP) et

- la taille de la zone valeur de paramètres (TZVP).

Au moment du CALL le pointeur SP sur la pile de contexte contient l'adresse de la base de la future zone valeur des paramètres.
On va donc le mémoriser dans un registe de travail interne à l'opérateur de PAC appelé BV.

L'interprétation de l'instruction CALL se déroulera de la manière suivante :

- Lecture dans la table des segments de l'adresse du bloc de description de la procédure dont le premier mot contient le nombre de paramètres et la taille de la zone valeur des paramètres.

- Sauvegarde de cette adresse qui servira pour la recopie des paramètres et sauvegarde de SP dans BV.

- Lecture de NP et TZVP et stockage de NP dans un autre registre de travail interne (RT).

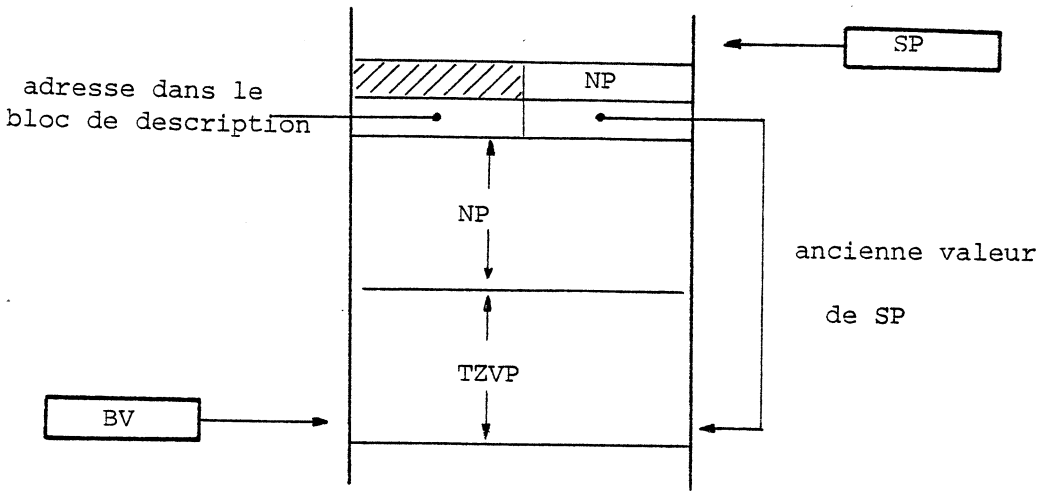
- Incrémentation de SP avec TZVP et mise à zéro de la Zone mémoire comprise entre la nouvelle valeur de SP et BV et qui deviendra la zone des paramètres.

- Incrémentation de SP avec NP de façon à pointer sur le mot de CTX dans lequel doit être construit la MARQUE de bloc.

- Recopie à cette adresse de l'adresse courante dans le bloc de description de la procédure (utile pour les instructions PARAM si une procédure est passée en paramètre), et de l'ancienne valeur de SP (BV), puis incrémentation de SP.

- Recopie du nombre de paramètres et nouvelle incrémentation de SP.

Tout ceci pour avoir l'image suivante de CTX:



- l'instruction ENTER

Entre l'instruction CALL et l'instruction ENTER, un certain nombre (égal à NP en principe) d'instructions PARAM ont été exécutées. Chacune d'elles doit mettre à jour dans CTX l'adresse courante dans le bloc de description permettant la recopie des paramètres et le nombre de paramètres doit être décrémenté.
Il doit être nul au moment de l'instruction ENTER. Si tel n'est pas le cas, il y a une erreur et donc un déroutement vers le microprogramme d'erreur correspondant.

Après cette vérification, PAC doit mettre à jour le chaînage dynamique en remplaçant dans CTX le champ donnant l'adresse courante dans le bloc de description par la base de la procédure appelante

DISPLAY(NVC)

c'est-à-dire un registre interne de l'opérateur de PAC dont l'adresse est donnée par un registre externe appelé NVC donnant le numéro lexicographique de la procédure appelante.

Ensuite PAC doit mettre à jour le chaînage statique : pour celà il doit changer le niveau courant (de la procédure appelante) NVC avec celui de la procédure appelée (changement de contexte) défini par le champ S du nom SD.
Le chaînage statique, égal à la valeur de l'ancienne base du même niveau contenue dans DISPLAY(NVC) est empilé dans CTX et la nouvelle base est positionnée.

De plus, on doit faire une mise à jour de la mémoire associative (récursivité) en faisant une recherche associative sur le champ S=NVC et en marquant à vide toutes les cases pour lesquelles il y a correspondance. Après quoi il reste à recopier les variables locales contenues dans le bloc de description de la procédure.

- l'instruction RETURN

Cette instruction termine l'exécution de toute procédure. Elle a pour effet la restauration de tous les registres de bases:

- le registre W dont l'ancienne valeur a été empilée à l'adresse W-1,
- le chaînage statique,
- le niveau d'appel (NVC),
- le chaînage dynamique.

Elle doit en outre remettre à jour la mémoire associative (récursivité).

V.5. - Chemin de données

Il est organisé autour de l'opérateur 16 bits du processeur PAC. Celui-ci doit permettre, comme nous l'avons vu dans les chapîtres précédents:

- de construire une adresse de 16 bits à partir du SD des instructions de type NOM SD pour la transmettre au registre mémoire.

- de faire le transit entre les registres de lecture en mémoire centrale RML0 et RML1, et les buffers d'écriture dans FILEVAL (PACE0 et PACE1).

- de transmettre à ces mêmes registres des valeurs immédiates construites à partir des instructions de type LITTERAL.

- de mettre à jour la pile de contexte CTX par l'intermédiaire des registres d'écriture dans la mémoire centrale RME0 et RME1.

Toutes ces opérations déterminent le chemin de données de ce processeur.

V.5.1. Bus d'entrée D de l'opérateur 16 bits

Pour réaliser toutes les opérations le bus d'entrée des données D doit être multiplexé entre :

- Les 6 bits poids faible de BIPAC étendu avec le bit 7 et le bit 8 (8 fois) de la ROM interne (voir calcul d'adresse).

- Les sorties des registres mémoires RML 0 et RML 1 (accès à un descripteur de variables par exemple).

- Les 12 bits poids faibles de BIPAC étendus avec les quatre bits de préfixe de valeur immédiate

(VI 15, VI 14, VI 13, VI 12).

- BIPAC en son entier (demi mot donnant la valeur (LIT 16) ou l'adresse de la valeur (LIT 32) pour un accès immédiat).

- La sortie du registre mémorisant le niveau courant sur les 4 bits poids faible étendue avec n'importe quoi (appel et retour de procédure).

Toutes les combinaisons possibles sur le bus d'entrée de l'opérateur 16 bits sont consignées dans le tableau ci-après :

On a donc 8 sources possibles pour le bus d'entrée D de l'opérateur. La première solution venant à l'esprit pour faire ce multiplexage est d'utiliser des boîtier "multiplexeur 8 vers 1" commandés par trois signaux. Le coût en matériel est de 16 boîtiers pour cette solution.

Si l'on découpe ce bus d'entrée en tranches de 4 bits, on s'aperçoit que seule la tranche poids fort nécessite un multiplexage 8 vers 1, pour les autres tranches des mutiplexeurs 4 vers 1 sont suffisants. Le nombre de boîtiers nécessaires passe de 16 à 10, mais il faut 7 signaux de contrôle.

On peut remarquer que certaines origines sur le bus D sont des regitres ou des mémoires (ROM ou FIFO) à sorties à "trois états". Ce type de sortie permet de réaliser directement le multiplexage des origines d'un bus. Cette possibilité est utilisable pour certaines sources connectées uniquement sur le bus d'entrée de l'opérateur (registres de lecture mémoire RML0 et RLM1 par exemple).

La solution retenue est un compromis entre sorties à "trois états" et multiplexeurs à sorties à "trois états".

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
D^*8	D^*8	D^*8	D^*8	D^*8	D^*8	D^*8	D^*8	D^*7	BIPAC 6	BIPAC 5	BIPAC 4	BIPAC 3	BIPAC 2	BIPAC 1	BIPAC 0
RMLO15	RMLO14	RMLO13	RMLO12	RMLO11	RMLO10	RMLO9	RMLO8	RMLO7	RMLO6	RMLO5	RMLO4	RMLO3	RMLO2	RMLO1	RMLO 0
RML1 15	RML1 14	RML1 13	RML1 12	RML1 11	RML1 10	RML1 9	RML1 8	RML1 7	RML1 6	RML1 5	RML1 4	RML1 3	RML1 2	RML1 1	RML1 0
0: VI15	0: VI14	0: VI13	0: VI12	BIPAC11	BIPAC10	BIPAC 9	BIPAC 8	BIPAC 7	BIPAC 6	BIPAC 5	BIPAC 4	BIPAC 3	BIPAC 2	BIPAC 1	BIPAC 0
1: BIPAC15	1: BIPAC14	1: BIPAC13	1: BIPAC12	BIPAC11	BIPAC10	BIPAC 9	BIPAC 8	BIPAC 7	BIPAC 6	BIPAC 5	BIPAC 4	BIPAC 3	BIPAC 2	BIPAC 1	BIPAC 0
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	NVC 3	NVC 2	NVC 1	NVC 1

Légende D

D^*7, D^*8 = sortie 7 et sortie 8 de la ROM de décodage du nom interne d'une variable.

Tableau : Entrées D de l'opérateur 16 bits

IV.5.2. Commande de la partie opérative

La même démarche que pour les commandes de la partie opérative du processeur FILE à été suivie, à savoir :

- Enumération de toutes les opérations exécutées sur le chemin de données.
- Choix des primitives et des paramètres de commandes.
- Codage.

Finalement la solution retenue demande cinq bits pour choisir une opération sur les Am2901 en débanalisant l'entrée D.
(le multiplexage du bus d'entrée de l'opérateur fait partie de ce champ appelé CODOP).
Ces cinq bits de microinstruction sont décodés à l'aide d'une ROM constituée de deux boîtiers de 32 mots de 8 bits permettant de générer 16 signaux de commande qui sont:

- l'entrée Cin de l'opérateur,
- les commandes I0...I4, I6 et I8 de l'opérateur (I5 est câblé à la masse et I7 est identique à S1(VI)),
- l'entrée $\overline{OE}$ du registre RML 0
- l'entrée $\overline{OE}$ du registre RML 1
- l'entrée interrogation du multiplexeur générateur de valeur immédiate (GVI)
- les deux entrées de sélection de ce multiplexeur S1(VI) et S2(VI).
- l'entrée interrogation du multiplexeur M3 (GM3)
- l'entrée de selection des multiplexeurs M3, M2 et M1
- l'entrée de selection du multiplexeur M0.

Remarque

Les entrées interrogation de M0, M1 et M2 sont obtenues en faisant le ET logique entre GVI et GMS :

$$GM0 - 2 = GVI \wedge GM3$$

Le codage de la ROM est donné dans le tableau suivant :

OPERATION	C_{in}	I_8	$\overline{I_6}$	I_4	I_3	$\overline{I_2}$	$\overline{I_1}$	I_0	$\overline{OE}_{AHL0}$	$\overline{OE}_{AHL1}$	GVI	$\overline{I_7}$ $S_3(VI)$	$S_2(VI)$	GMI	SM	SM_0
Y=RML(0)-1	0	0	1	1	0	1	1	1	0	1	1	0	Ø	1	Ø	Ø
Y=RML(1)->RB	1	0	1	1	0	1	1	1	1	0	1	1	Ø	1	Ø	Ø
Y=RB-RA->RB	1	0	1	0	1	1	0	1	1	1	1	1	Ø	1	Ø	Ø
Y=RA+1->RB	1	0	1	0	0	1	0	0	1	1	1	1	Ø	1	Ø	Ø
Y=ETAT->RB	1	0	1	1	0	1	1	1	1	1	0	1	1	1	0	0
Y=RA->RB	0	0	1	0	0	1	0	0	1	1	1	1	Ø	1	Ø	Ø
Y=RA,RA+1->RB	1	0	0	0	0	1	0	0	1	1	1	1	Ø	1	Ø	Ø
Y=NVC	1	0	1	1	0	1	1	1	1	1	1	0	Ø	0	Ø	1
Y=RML(0)->RB	1	0	1	1	0	1	1	1	0	1	1	1	Ø	1	Ø	Ø
2*Q->RB	0	1	0	0	0	0	1	0	1	1	1	1	Ø	1	Ø	Ø
Y=RA+RB->RB	0	0	1	0	0	0	0	1	1	1	1	1	Ø	1	Ø	Ø
Y=Q RML->RB	0	0	1	1	1	1	1	0	1	0	1	1	Ø	1	Ø	Ø
Y=RA-1->RB	0	0	1	0	1	1	0	0	1	1	1	1	Ø	1	Ø	Ø
Y=ROM+RA->RB	0	0	1	0	0	1	0	1	1	1	1	1	Ø	0	1	0
Y=RML(1)->Q	1	0	0	1	0	1	1	1	1	0	1	0	Ø	1	Ø	Ø
Y=Q->RB	0	0	1	0	0	0	1	0	1	1	1	0	Ø	1	Ø	Ø
Y=Q+1->Q	1	0	0	0	0	0	1	0	1	1	1	0	Ø	1	Ø	Ø
Y=RML(0)+RA->RB	0	0	1	0	0	1	0	1	0	1	1	0	Ø	1	Ø	Ø
Y=RML(0)->Q	0	0	0	0	0	0	1	0	0	1	1	0	Ø	1	Ø	Ø
Y=Q+1	1	0	1	0	0	0	1	0	1	1	1	0	Ø	1	Ø	Ø
Y=VI(1)	1	0	1	1	0	1	1	1	1	1	0	0	0	1	0	0
Y=BIPAC	1	0	1	1	0	1	1	1	1	1	1	0	Ø	0	0	0
Y=VI(2)	1	0	1	1	0	1	1	1	1	1	0	0	1	1	0	0
Y=VI(3)	1	0	1	1	0	1	1	1	1	1	0	1	0	1	0	0

IV.5.3. Destination Y

On aura besoin de 3 bits (champ DY) pour spécifier la destination du bus Y de l'opérateur. Ces trois bits sont décodés à l'aide d'un circuit MSI classique (74S 138) pour générer un ordre de chargement vers l'un des registres suivants :

- PACE 0
- PACE 1

} en liaison avec le bus file

- RME 0
- RME 1

} en liaison avec le bus mémoire

- RADM — en liaison avec le bus adresse mémoire
- ADCODE — en liaison avec le bus instruction du processeur PINS
- NVC — mémorisant le niveau lexicographique courant.

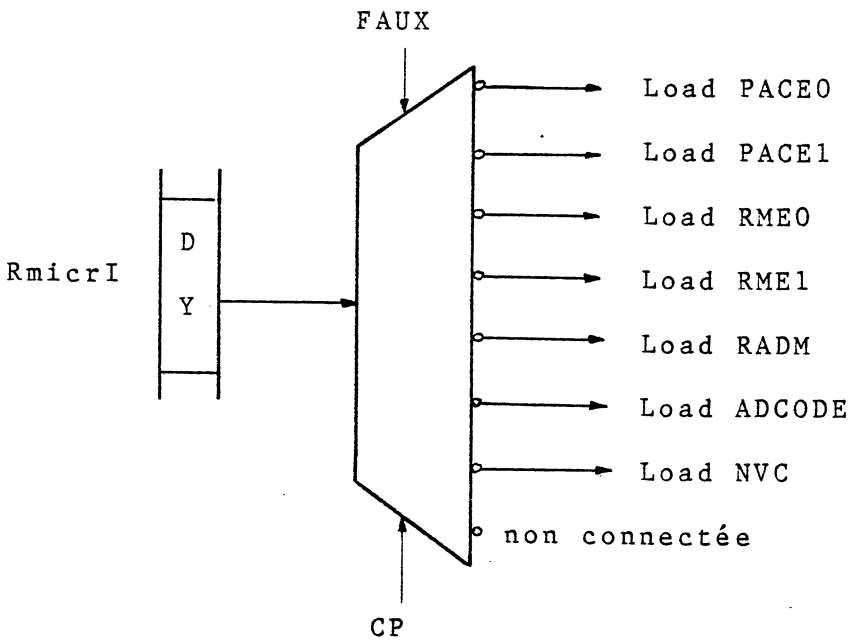


Figure IV.11. Décodage de la destination Y.

IV.5.4. Génération de l'adresse des registres internes

9 bits sont nécessaires pour commander les entrées adresses des registres internes de l'opérateur.

Ces adresses peuvent venir :

- soit de la microinstruction (registres de travail + registre SP et W)

- soit du registre NVC

- soit de la sortie S* de la ROM de décodage du nom interne.

Les deux dernières sources permettent d'adresser les registres de base (display).
Sur l'adresse B, on n'a pas besoin d'avoir S*, en effet on n'effectue jamais de chargement dans le display quand il est adressé par le nom interne (display (S*) n'est utilisé que pour exécuter un calcul d'adresse de la forme

Y = RA + ROM, RA = RAM (S*), Y -> RADM).

Seulement 6 registres de travail sont utilisés par le microprogramme. Si l'on ajoute les registres SP et W adressables par microprogramme on obtient 8 registres, donc un champ adresse de 3 bits est suffisant.

Les commandes des entrées adresses demandent 9 bits décomposables en :

- 3 bits ADA

- 3 bits ADB

- 1 bit de sélection pour le multiplexeur sur l'adresse B.

- 2 bits de sélection pour le multiplexeur d'adresse A.

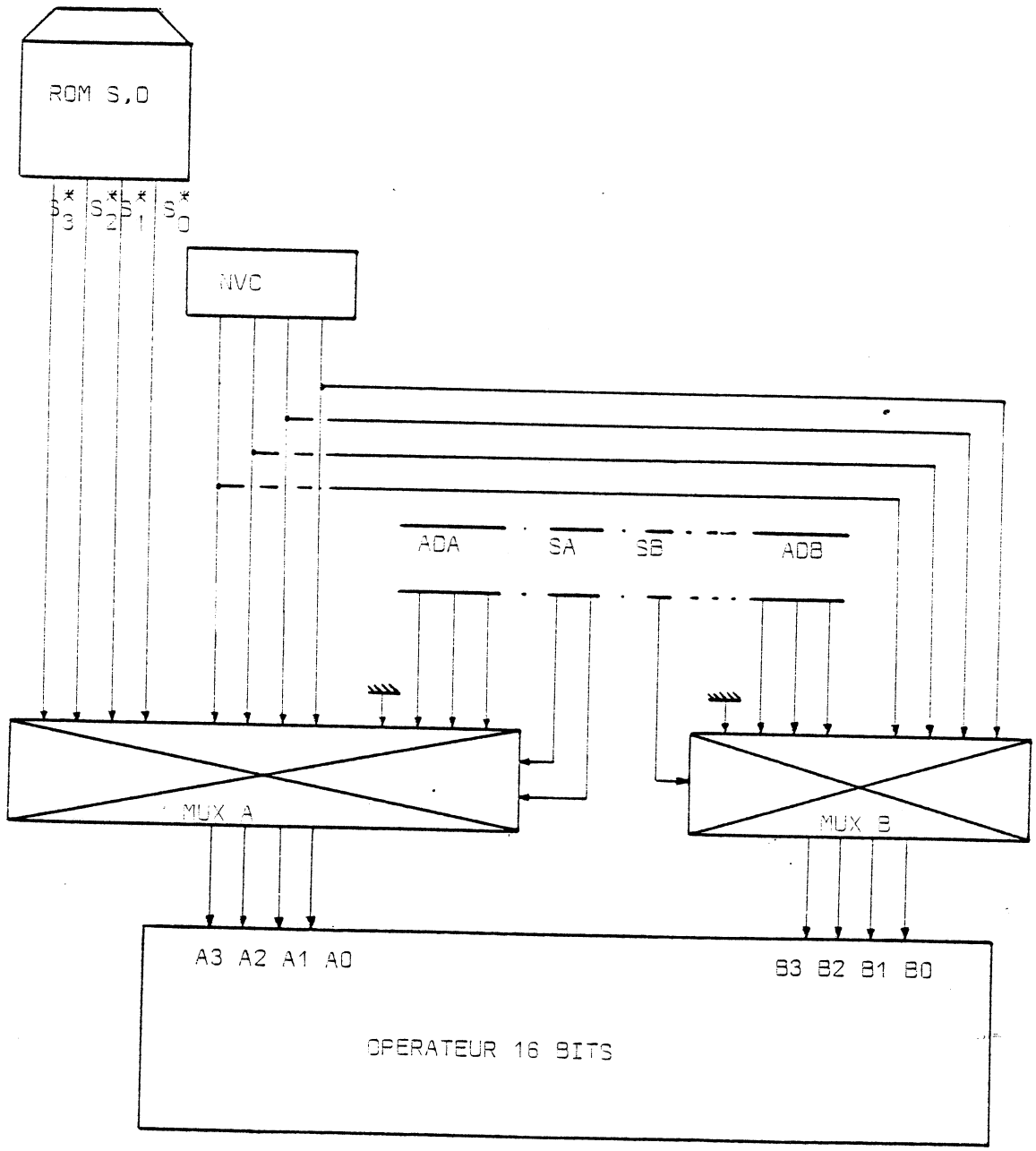


Figure V.12. Adressage des registres internes de l'opérateur.

V.6. Partie contrôle du processeur PAC

Elle est bâtie autour du macrocomposant Am 2909 dont le rôle est d'assurer le séquencement du microprogramme en calculant pendant le cycle numéro i l'adresse de la microinstruction qu'il faudra exécuter au cycle i + 1 en fonction des champs SEQ, CATT et BRA de la microinstruction.

Le fonctionnement de ce circuit est décrit dans le chapitre consacré au processeur FILE.

V.6.1. Besoins en branchement

On distingue plusieurs types de branchements possibles :

a) Branchement direct à un point d'entrée directement lié à la macroinstruction :

L'adresse est fournie par les bits 10 à 14 de BIPAC, le 15ème bit est toujours à 1 pour les instructions envoyées à PAC (voir le codage desinstructions). Les sorties de BIPAC sont connectées aux entrées D2 à D6 du séquenceur, l'entrée D7 est câblée à la masse et D1 et D0 permettent de contrôler l'oportunité du branchement :

- D1 reçoit $\overline{DCAM}$ pour s'assurer que la mémoire associative est disponible pour PAC.
- D0 reçoit le signal $\overline{TROMPE}$.

La condition $\overline{D0.D1}$ est une condition d'erreur.

Les adresses de point d'entrée sont de la forme

0 x x x x x 1 1

La commande utilisée pour de tel branchements est : JMPD.

b) Branchement à une adresse étiquetée :

On utilise :
- soit la commande JMPR (branchement à l'adresse contenue dans le registre R),
- soit la commande JSR (appel à un sous-micro- programme).

Le registre R a été chargé en début de cycle par le champ BRA de la microinstruction.
En fait, ce registre interne au séquenceur doit être considéré comme le registre microinstruction pour ce champ BRA.

Les besoins en branchement à des séquences étiquetées montrent que 6 bits suffisent pour BRA. Ils seront câblés sur les entrées R2 à R7 du séquenceur, R0 et R1 sont reliés à la masse.

Un couple de conditions câblé sur OR0 et OR1 permettent un branchement à quatre directions si elles sont validées.

c) Bouclage sur la microinstruction courante :

Ce bouclage permet d'attendre un évènement extérieur qui permettra soit le passage en séquence, soit un branchement.

d) Branchement à une adresse d'interruption :

Certains événements extérieurs doivent provoquer l'arrêt de l'exécution du processeur (ou des processeurs). Ce sont :

- Une erreur détectée par PME qui a généré le signal INT PME dont l'occurence provoque l'arrêt de l'exécution en cours et le branchement à l'adresse DC du microprogramme.

- Le signal TROMPE signalant une mauvaise anticipation de PINS, provoque le branchement en 5D et un bouclage sur cette adresse jusqu'à ce que BIPAC redevienne non vide.

- Le bit 15 de BIPAC qui lorsqu'il vaut 1 indique un code invalide et déroute le microprogramme à l'adresse 5C.

- Le signal DCAM provoque un branchement en 5E et une attente d'allocation de la mémoire associative.

V.6.2. Conditions et attente

Comme pour le processeur PINS on distingue les opérations de séquencement conditionnelles des branchements multidirectionnels.

Les opérations de séquencement conditionnelles se traduisent par deux commandes différentes du séquenceur suivant que la condition est ou n'est pas remplie, tandis que pour les branchements multidirectionnels la (ou les) condition(s) ne modifie(ent) pas le code du 2909 mais agit (agissent) sur les entrées OR 0 et OR 1 pour forcer à "1" les bits poids faibles d'adresse.

Les attentes peuvent être classées parmi les opérations de séquencement conditionnelles.

Les évènements susceptibles d'être attendus par PAC sont :

- BIPAC NON VIDE traduisant l'état du "buffer" d'instruction

- $\overline{\text{DPAC}}$ indiquant que FILE a satisfait la requête de PAC

- AMC (registre adresse mémoire libre

- VMC (registre donnée mémoire disponible)

Les conditions pouvant entraîner une opération différente de séquencement sont:

- F=0

- F=0

- ALCAM

Les couples de conditions multiplexèes sur les entrèes OR0 et OR1 du sèquenceur sont:

- TROUVE,"0" (indicateur issu de la logique de CAM)

- D14, "0" (le bit 14 de la partie préfixe d'un descripteur indique une indirection sur la valeur).

- D15, D14.D15 (D15 indique si la variable est un pointeur).

- "0", DCAM

Comme pour PINS, la mise en parallèle des deux listes et l'examen des incompatibilités entre telle condition et tel couple ont permis de minimiser la largeur du champ CATT de la microinstruction et de la réduire à trois bits.

V.6.3. Commandes de séquencement

Elles sont consignées dans une PROM de 32 mots de 8 bits contenant la table des commandes. Elle est adressée par les quatre bits du champ SEQ de la microinstruction et par la sortie du multiplexeur de condition/attente.

Elle fournit en sortie :

- Les quatres bits de commande des 2909 (S0, S1, FE, Pup).

- La retenue entrante du même circuit (Cin).

- Le signal VOR 0-1, validant les entrées OR0 et OR1.

- Le signal OR2-6 permettant la prise en compte des interruptions autre que INT-PME.

- Le signal FAUX pour bloquer l'horloge CP.

La liste des ordres de séquencement et le codage de ROM-SEQ sont donnés dans le tableau suivant :

ADRESSE	OPERATION	VOR01	FAUX	OR2-6	OE	S0	S1	FE	Pub
00	JMPR(VOR0-1)	0	0	1	1	1	0	1	x
01	Deb Loop	1	0	1	1	0	0	0	1
02	Pop Stack	1	0	1	1	0	0	0	0
03	Continue(VOR0-1)	0	0	1	1	0	0	1	x
04	Loop	1	0	1	1	0	1	1	x
05	JSR AR	1	0	1	1	1	0	0	1
06	RTS	1	0	1	1	0	1	0	0
07	Continue(CondLoop)	1	1	1	1	0	0	1	x
08	Continue(AHJMPD)	1	1	1	0	0	0	1	x
09	Continue(AHJMPR)	1	1	1	0	0	0	1	x
0A	Continue(AH)	1	1	1	0	0	0	1	x
0B	Continue (AHLoop)	1	1	1	0	0	0	1	x
0C	Continue(CondJMPR)	1	1	1	0	0	0	1	x
0D	Continue(CondEndLoop)	1	1		0	0	0	1	x
0E	JMPR	1	0	1	1	1	0	1	x
0F	Continue	1	1	1	0	0	0	1	x
//									
10	JMPR (VOR0-1)	0	0	1	1	1	0	1	x
11	Deb Loop	1	0	1	1	0	0	0	1
12	Pop Stack	1	0	1	1	0	0	0	0
13	Continue (VOR0-1)	0	0	1	1	0	0	1	x
14	Loop	1	0	1	1	0	1	1	x
15	JSR AR	1	0	1	1	1	0	0	1
16	RTS	1	0	1	1	0	1	0	0
17	Loop	1	0	1	1	0	1	1	x
18	JMPD	1	0	0	1	1	1	1	x
19	JMPR	1	0	1	1	1	0	1	x
1A	Continue	1	0	1	1	0	0	1	x
1B	Loop	1	0	1	1	0	1	1	x
1C	JMPR	1	0	1	1	1	0	1	x
1D	End Loop	1	0	1	1	1	0	0	0
1E	JMPR	1	0	1	1	1	0	1	x
1F	Continue	1	1	1	0	0	0	1	x

Codage de ROM-SEQ.

Réalisation du mécanisme d'interruption

Il s'agit de forçer le déroutement du microprogramme à des adresses prédéfinies en fonction de quatre évènements :

- INTPME ------> forçage à DC
- BIPAC15 -----> " " à 5C
- TROMPE ------> " " à 5D
- DCAM --------> " " à 5E

L'évènement INTPME correspond à une erreur fatale provoquant une sauvegarde générale de PASC-HLL. Il doit provoquer le déroutement du microprogramme à DC quel que soit l'état de PAC.

Les autres ne seront pris en compte qu'au moment du "fetch" de la microinstruction suivante (instruction de séquencement Att JMPD positionnant à "0" le signal OR2-6 de ROM SEQ).

La réalisation proposé pour ce générateur de fonction d'interruption utilise un multiplexeur 8 vers 1 (74 LS 151) et les 6 des 8 opérateurs "OR" internes au séquenceur câblées ce la façon suivante :

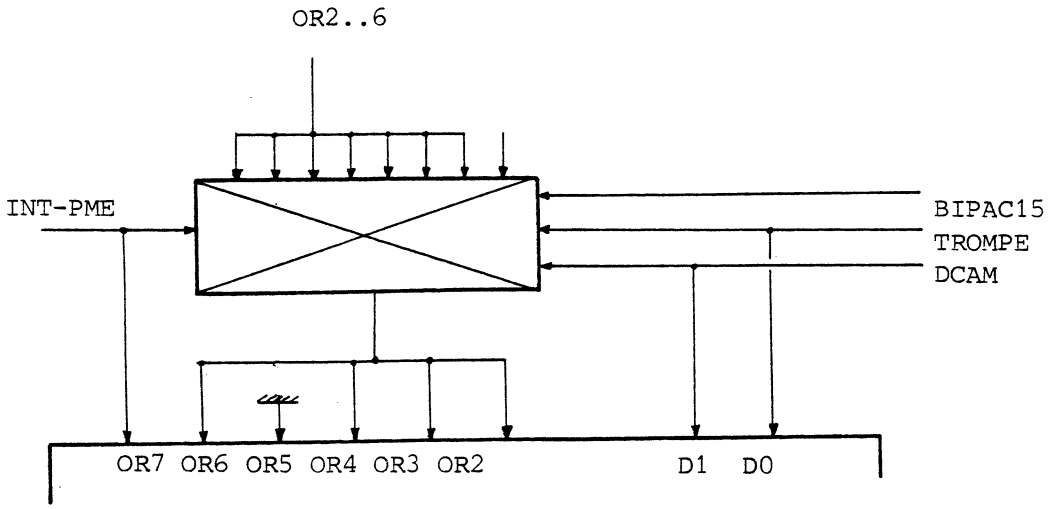


Figure V.13. - Mécanisme d'interruption.

Le signal INT PME invalide le multiplexeur quand il vaut 1 et entraîne:

OR7 = OR6 = OR4 = OR3 = OR2 = "1", OR5 = 0

donc une des adresses DC, DD, DE ou DF est forçée en sortie (la première instruction de la sauvegarde devra être répérée 4 fois).

Quand il est validé et que l'on est pas en séquence de "FETCH" les entrées OR sont toutes à zéro. Pendant la séquence de FETCH elles seront à zéro seulement si les trois entrées de sélections sont à zéro.

Si l'une des entrées vaut 1 on aura :

OR7 = OR5 = 0

OR6 = OR4 = OR3 = OR2 = 1

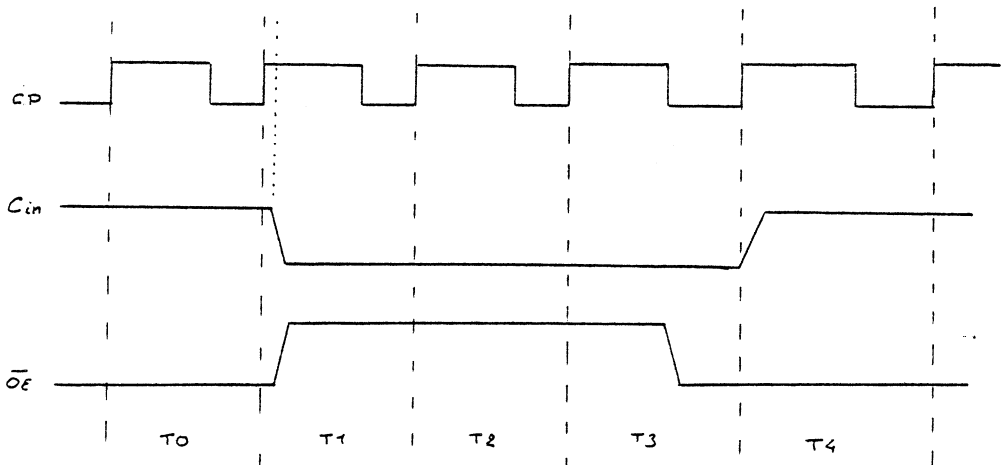
Si BIPAC 15 = 1 alors l'adresse forçée sera DC

Si $\overline{\text{TROMPE}}$ = 1 " " " " DD

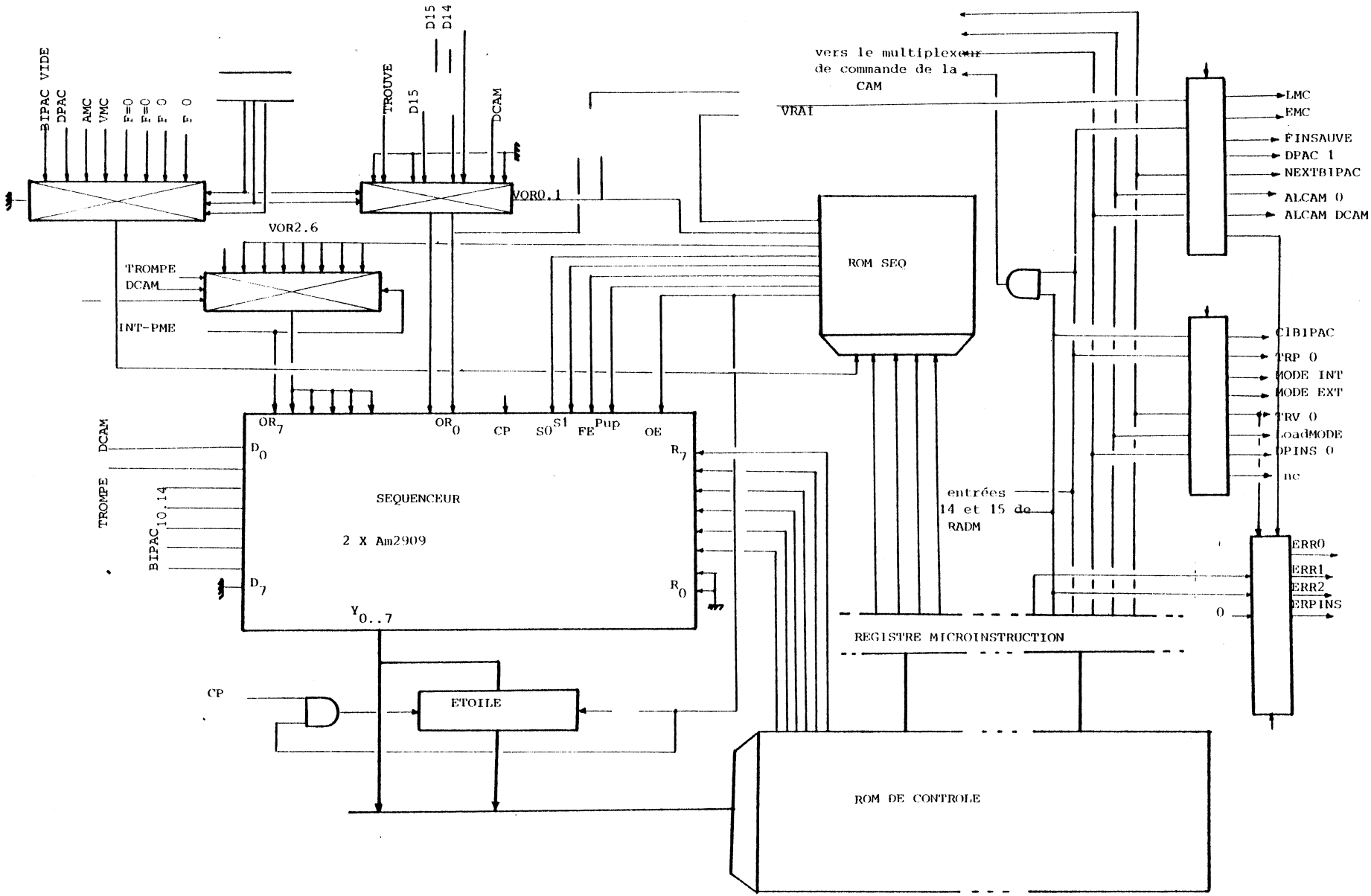
Si $\overline{\text{DCAM}}$ = 1 " " " " DE

Adresse	Sortie	Opération	VOR0-1	FAUX	OR2-6	OE	S0	S1	$\overline{FE}$	Pup
00	3A	JMPR(VOR0-1)	0	0	1	1	1	0	1	x
01	B1	Deb Loop	1	0	1	1	0	0	0	1
02	B0	Pop Stack	1	0	1	1	0	0	0	0
03	B2	Continue(VOR0-1)	0	0	1	1	0	0	1	x
04	B6	Loop	1	0	1	1	0	1	1	x
05	B9	JSR AR	1	0	1	1	1	0	0	1
06	B4	RTS	1	0	1	1	0	1	0	0
07	F2	Continue(CondLoop)	1	1	1	1	0	0	1	x
08	E2	Continue(AHJMPD)	1	1	1	0	0	0	1	x
09	E2	Continue(AHJMPR)	1	1	1	0	0	0	1	x
0A	E2	Continue(AH)	1	1	1	0	0	0	1	x
0B	E2	Continue(AHLoop)	1	1	1	0	0	0	1	x
0C	F2	Continue(CondJMPR)	1	1	1	1	0	0	1	x
0D	F2	Continue (Cond End Loop)	1	1	1	1	0	0	1	x
0E	BA	JMPR	1	0	1	1	1	0	1	x
0F	B2	Continue	1	0	1	1	0	0	1	x
10	3A	JMPR (VOR0-1)	0	0	1	1	1	0	1	x
11	B1	Deb Loop	1	0	1	1	0	0	0	1
12	B0	Pop Stack	1	0	1	1	0	0	0	0
13	B2	Continue(VOR0-1)	0	0	1	1	0	0	1	x
14	B6	Loop	1	0	1	1	0	1	1	x
15	B9	JSR AR	1	0	1	1	1	0	0	1
16	B4	RTS	1	0	1	1	0	1	0	0
17	B6	Loop	1	0	1	1	0	1	1	x
18	9E	JMPD	1	0	0	1	1	1	1	x
19	BA	JMPR	1	0	1	1	1	0	1	x
1A	B2	Continue	1	0	1	1	0	0	1	x
1B	B6	Loop	1	0	1	1	0	1	0	x
1C	BA	JMPR	1	0	1	1	1	0	1	x
1D	B8	End Loop	1	0	1	1	1	0	0	0
1E	BA	JMPR	1	0	1	1	1	0	1	x
1F	B2	Continue	1	0	1	1	0	0	1	x

		J-1	Continue
	T0	J	-
Attente	--> T1	J+1	Si Att cont
	T2	J+1	"
Attente Levée	--> T3	J+1	"
	T4	J+2	JMPR (AR)
	T5	AR	-



PC	J+1	J+2	J+2	J)2	J+S	AR+1
Sorite Am2909	J+1	J+2/TS	TS	J+2	AR	AR+1
Adresse ROM	J+1	J+2/J+1	J+1	J+2	AR	AR+1
Sortie ROM	I(J+1)	I(J+1)	I(J+1)	I(J+2)	I(AR)	I(AR+1)
Ien exécut.	I(J)	I(J+1)	I(J+1)	I(J+1)	I(J+2)	I(AR)
Registre *	I	J+1	J+1	J+1	J+2	AR
Sortie de Reg*	TS	J+1	J+1	TS	TS	TS


BIPAC VIDE
DPAC
AMC
VMC
F=0
F=0
F 0
F 0
TROUVE
D15
D14
D15
DCAM
VOR0.1
VOR2.6
TROMPE
DCAM
INT-PME
vers le multiplexeur
de commande de la
CAM
VRAI
ROM SEQ
LMC
EMC
FINSAUVE
DPAC 1
NEXTBIPAC
ALCAM 0
ALCAM DCAM
CLBIPAC
TRP 0
MODE INT
MODE EXT
TRV 0
LoadMODE
OPINS 0
nc
ERR0
ERR1
ERR2
ERPINS
0
entrées
14 et 15 de
RADM
DCAM
TROMPE
BIPAC 10..14
OR7
OR0
CP
S0
S1
FE
Pup
OE
D0
D7
R7
R0
SEQUENCEUR
2 X Am2909
Y0..7
CP
ETOILE
REGISTRE MICROINSTRUCTION
ROM DE CONTROLE

V.7. Ordres

Sont regroupés dans cette rubrique les signaux permettant le dialogue et la synchronisation avec les processeurs interlocuteurs, les ordres de positionnement et de gestion de la bascule "MODE", la génération du numéro de zone mémoire en conjugaison avec le chargement de RADM (DY = RADM), les commandes d'opérations sur la mémoire associative, le positionnement des bits d'erreur sur le registre d'état.

Certains ordres sont des niveaux devant être maintenus durant toute la durée d'un microcycle (commande CAM par exemple), d'autres sont synchronisées par l'horloge CP et certains doivent être inhibées pour les instructions conditionnelles.

V.7.1. Signaux de dialogue avec PME

Ce sont les ordres de lecture (LMC) et d'écriture (EMC) en mémoire centrale. PME répond par les signaux AMC et VMC testés par le mécanisme d'attente de PAC. L'interface PME est pratiquement identique pour tous les processeurs et il est décrit en détail dans le chapitre consacré au processeur PINS.

V.7.2. Signaux de dialogue avec PINS

Il y a tout d'abord l'ordre permettant l'initialisation de BIPAC (ou sa réinitialisation après l'occurence du signal TROMPE) et qui a pour effet de vider le buffer d'instruction et de mettre la bascule TROMPE à zéro :

INIT = Clear-BIPAC & TROMPE <--- "0"

La séquence de "FETCH" de l'instruction suivante est caractérisée par l'ordre :

NEXT-BIPAC

La fin d'une sauvegarde se traduit par l'ordre :

FINSAUVE

Pour avertir PINS que le registre ADCODE à été chargé et que le point d'entrée dans le code d'une procédure est disponible sur le bus instruction, PAC génére la demande :

DPINS <--- "0"

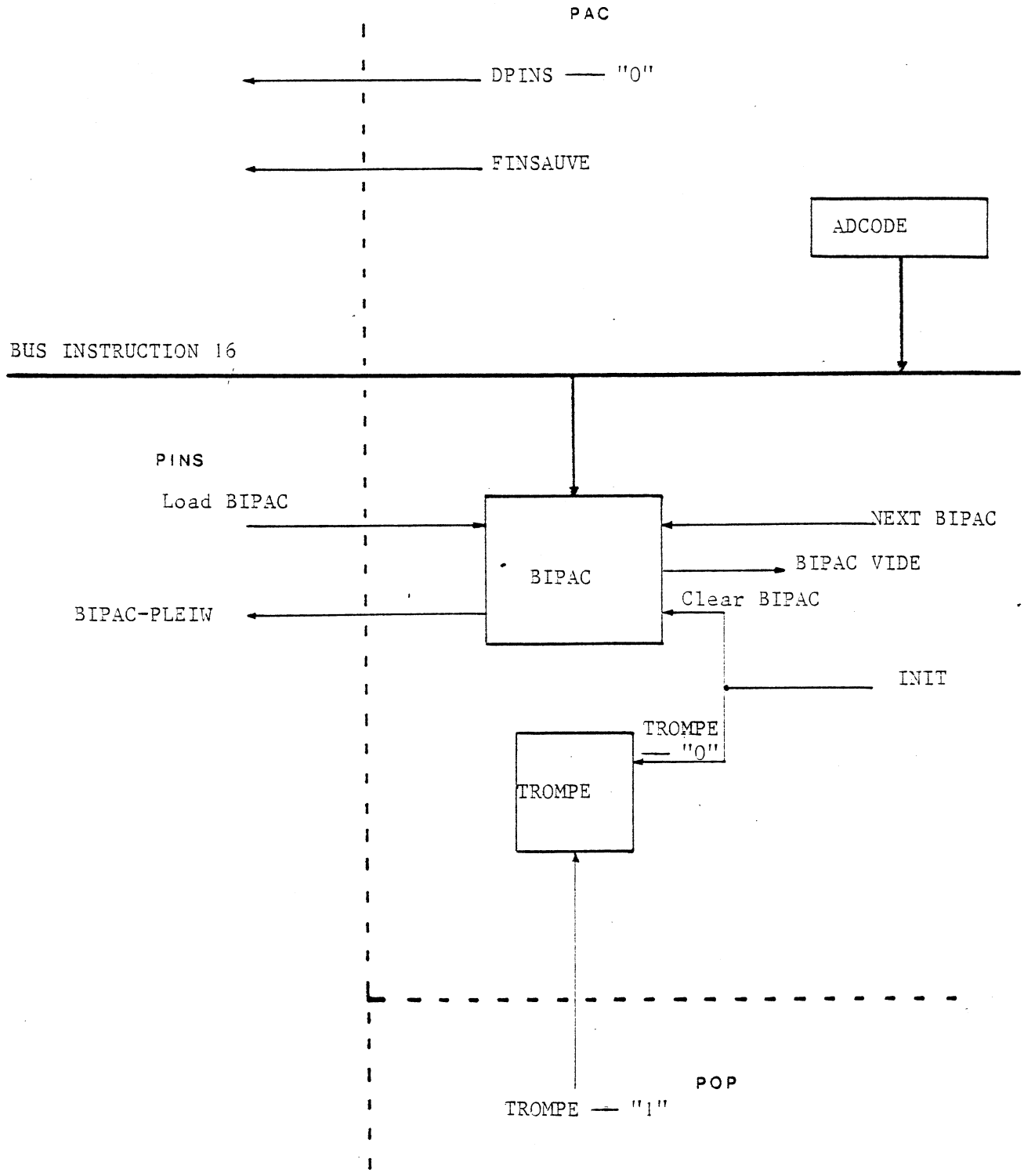


Figure V.15. - Dialogue PAC/PINS (les ordres de PAC sont soulignés).

V.7.3.Signaux de dialogue avec FILE

Le processeur FILE est une ressource partagée entre les processeurs PAC et POP qui disposent chacun d'une bascule de demande gérée par l'allocateur de FILE. La synchronisation PAC/FILE est assurée par la bascule DPAC mise à "1" par l'ordre (DPAC <-"1") et remise à zéro par FILE en fin d'exécution de la requête.

La mémoire associative pose un problème plus complexe d'allocation.
En effet elle est utilisée :

- par le microprogramme de PAC indépendemment de l'allocation de FILE à PAC (consultation de la mémoire associative pour toutes les instructions d'accès pendant le calcul d'adresse, et changement de niveau après l'entrée dans une procédure),

- par le microprogramme de FILE quand il travaille pour PAC (initialisation d'une dépendance et suppression d'un élément de FILEDEP),

- par le microprogramme de FILE quand il travaille pour POP (résolution d'une dépendance).

Bien que la fréquence d'utilisation de la CAM soit la plus élevée pour PAC il faut que la résolution des dépendances soit l'opération la plus prioritaire et par conséquent qu'elle soit allouée en priorité à FILE (ALCAM=0).

On a utilisé deux bascules pour cet automate d'allocation:

- une bascule RS (DCAM) située sur FILE
- une bascule D (ALCAM) sur PAC : elle reçoit $\overline{\text{DCAM}}$ sur son entrée D.

La bascule ALCAM est initialisée à "0" par PAC.
Quand FILE est sur le point de résoudre une dépendance, il positionne une demande (DCAM <- 0) et se met en attente de ALCAM=0. Quand sa requête est satisfaite, il libère la mémoire associative (DCAM <-1).

Si PAC veut faire une recherche, il prend en compte l'état de DCAM (ALCAM <- DCAM) et se met en attente de ALCAM=1. Deux cas peuvent se présenter:

- FILE est en train de résoudre une dépendance et PAC doit attendre qu'il ait terminé.
- il n'y a pas eu de demande de FILE et PAC n'attend pas.

Quand il ne l'utilise pas PAC rend la CAM à FILE

(ordre ALCAM <- "0").

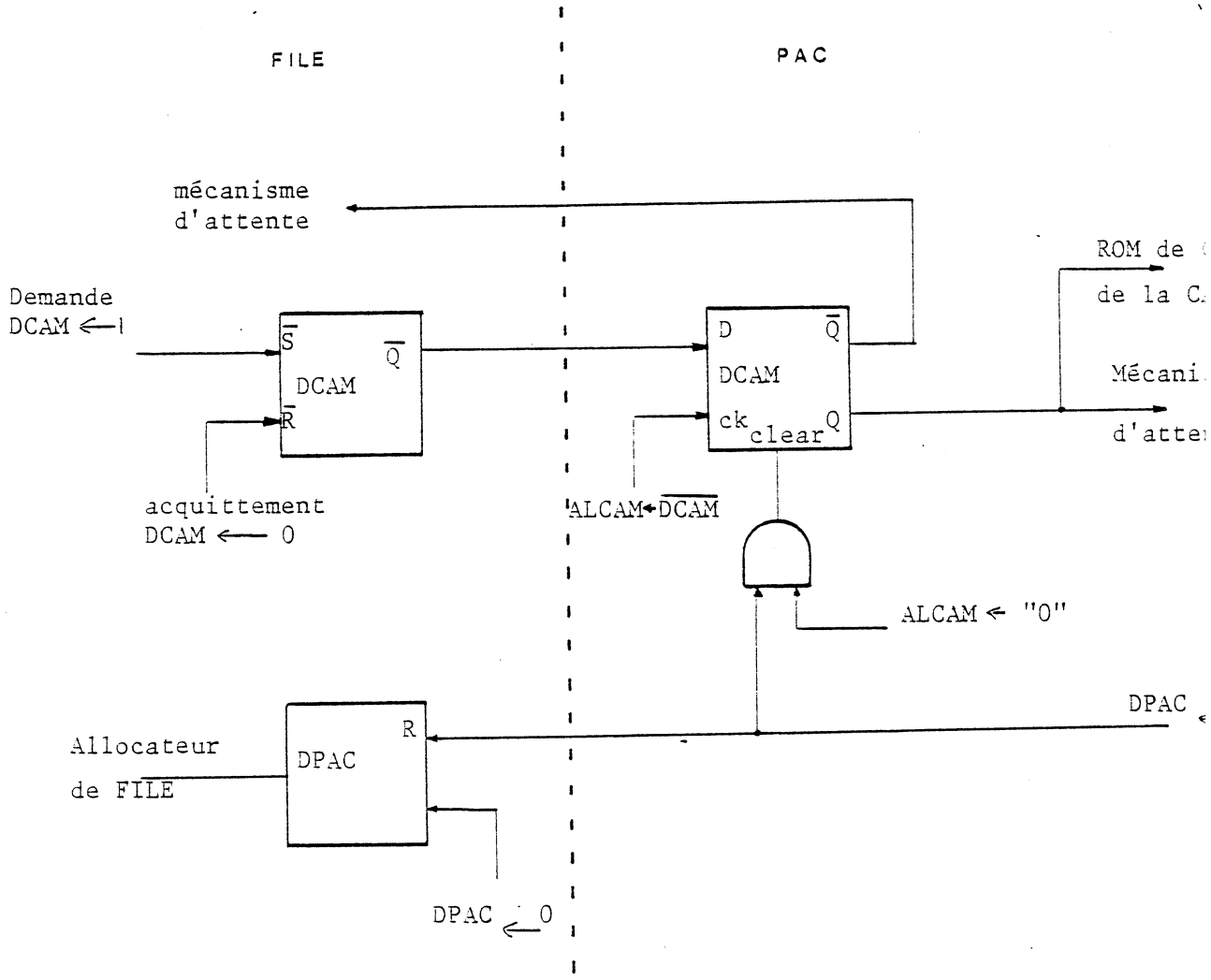


Figure V.16. - Synchronisation PAC/FILE : les ordres de PAC sont soulignés.

V.7.4. Gestion de la bascule de mode

MODE indique si l'on se trouve dans le module principal ou dans le module externe. Trois ordres sont nécessaires à sa gestion:

- MODE <- INT
- MODE <- EXT
- Load MODE

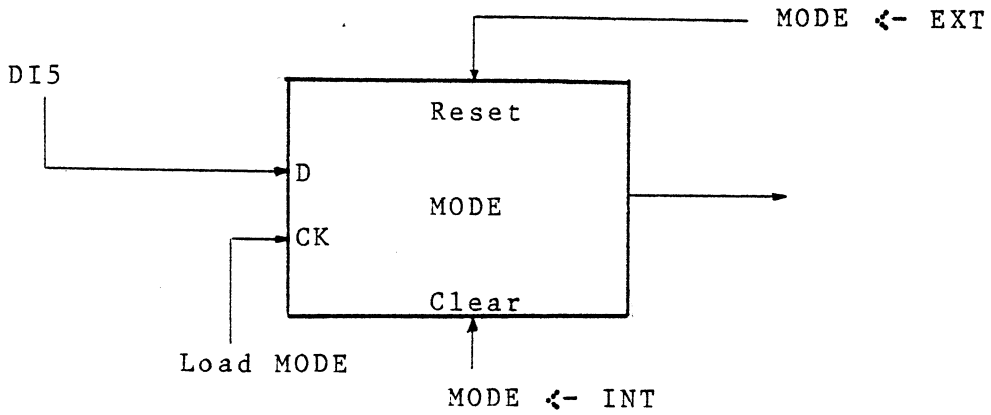


Figure V.17 - Câblage de la bascule de MODE.

V.7.5. Commandes de la CAM

Elles ont été décrites dans le paragraphe consacré à la mémoire associative.

V.7.6. Chargement du numéro de zone mémoire dans RADM

Deux bits du champ ORDRE de la microinstruction de PAC sont utilisés avec la bascule MODE pour charger les deux bits poids forts de RADM en conjugaison avec DY=RADM.

Suivant l'état de ces deux bits, PAC pourra adresser les quatre zones de la façon suivante:

ORD6	ORD5	RADM 15	RADM14	Zone
0	0	0	0	CTX
1	0	1	MODE	CODE/EXT
1	1	1	1	EXT
0	1	0	1	DYN

La réalisation demande une porte ET et une porte OU:

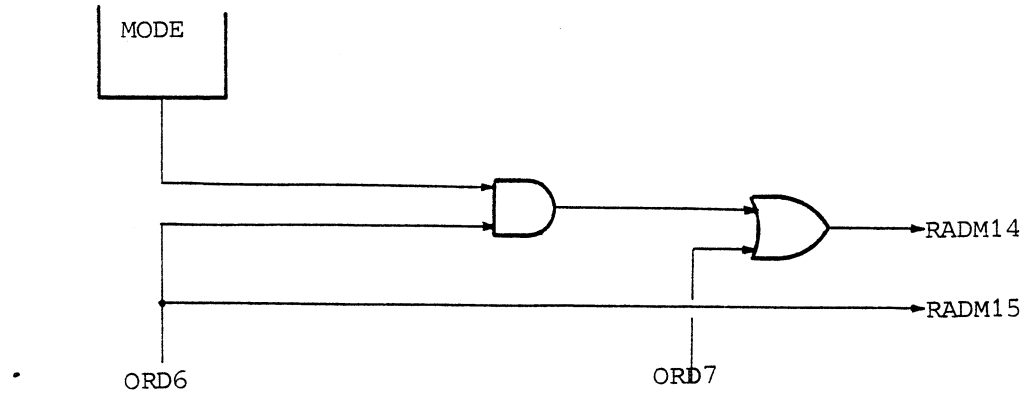


Figure V.18 - Génération du numéro de zone.

Remarque
Le fait d'utiliser deux bits du champ ORDRE pour adresser une zone mémoire implique une contrainte quant à l'écriture du microprogramme: la microinstruction ayant son champ DY=RADM ne pourra pas exécuter d'ordre.

V.7.7. Réalisation et codage des ordres

On utilise un champ de 6 bits de la microinstruction pour coder:

- les 14 ordres impulsionnels,
- les 6 commandes de la CAM,
- le chargement du numéro de zone mémoire,
- le positionnement d'une erreur parmi 4 possibles.

La génération et le codage des ordres de PAC utilisant deux circuits décodeurs 8 vers 3 et un circuit 8 bits adressables latch est montrée dans la figure suivante.

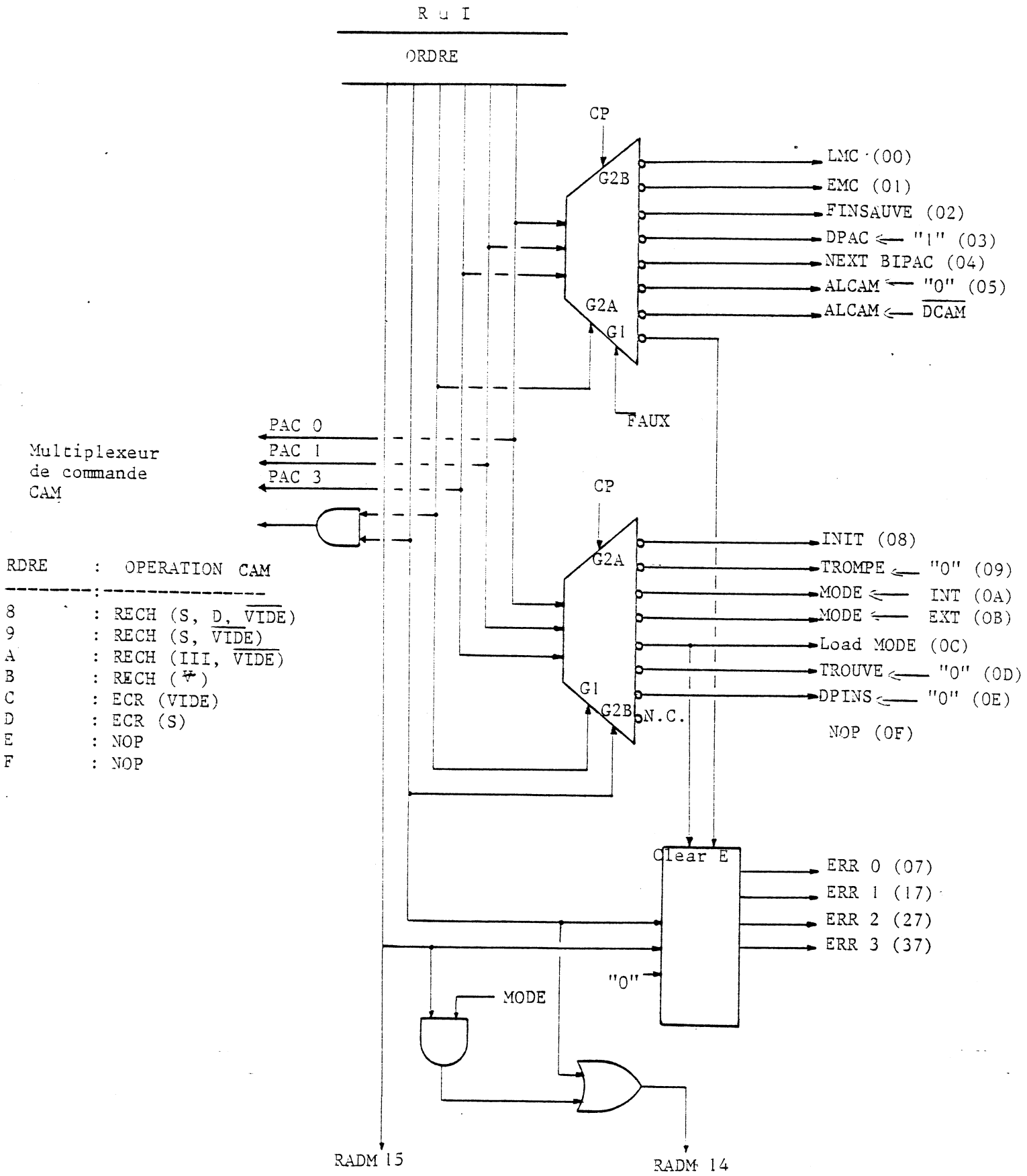
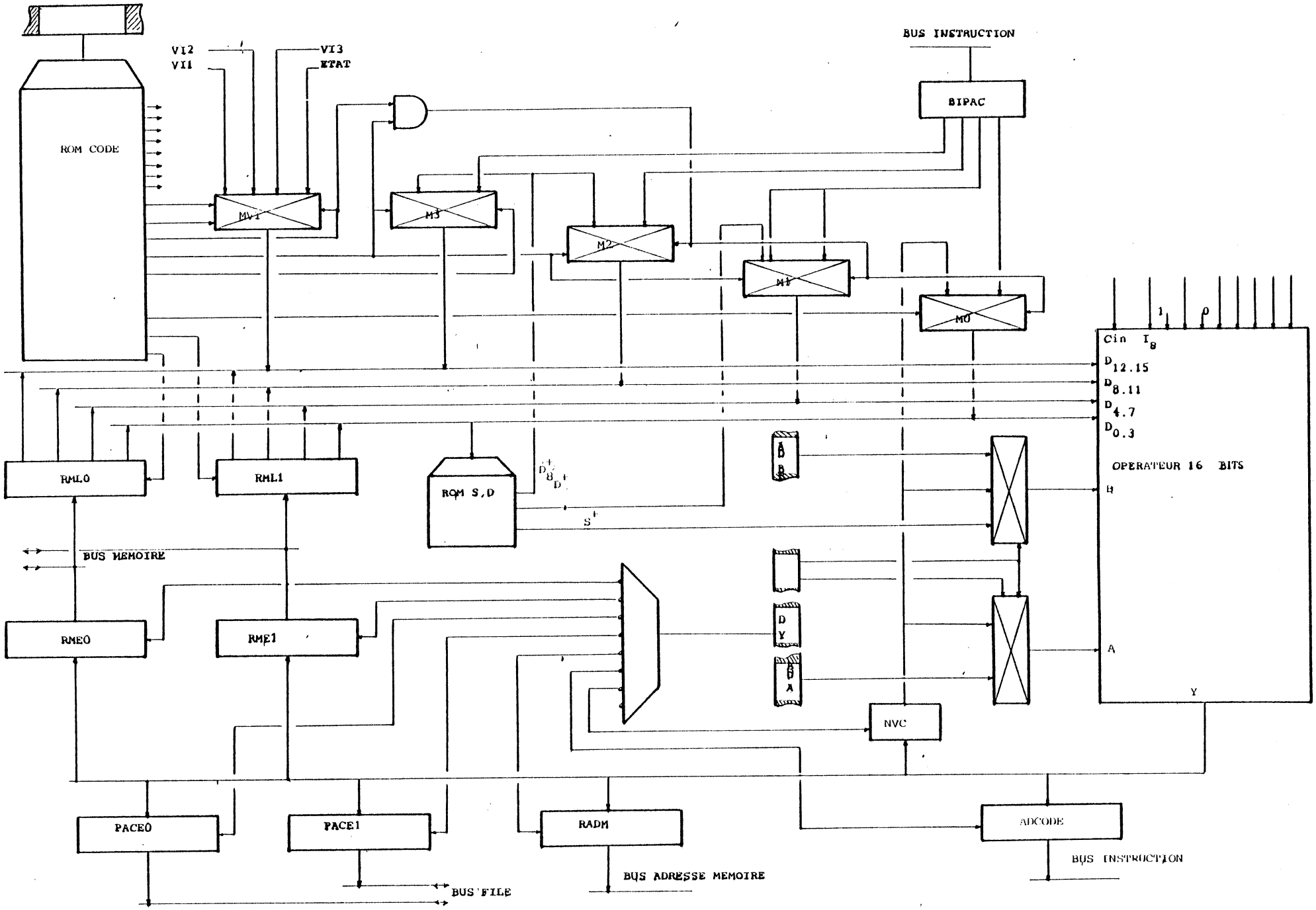


Figure V.19. - Décodage des ordres.

Si DY = RADM la zone mémoire

Adressée sera : CTX si ORDRE = 0F
DYN si " = 1F
CDE ou EXT si " = 2F
EXT si " = 3F


VI2
VI1
VI3
ETAT
BUS INSTRUCTION
BIPAC
ROM CODE
MV1
M3
M2
M1
M0
Cin
I8
D12.15
D8.11
D4.7
D0.3
OPERATEUR 16 BITS
B
A
Y
RML0
RML1
ROM S,D
BUS MEMOIRE
RME0
RME1
NVC
PACE0
PACE1
RADM
ADCODE
BUS ADRESSE MEMOIRE
BUS FILE
BUS INSTRUCTION

Chapitre VI

PRESENTATION TECHNIQUE DU PROCESSEUR POP

VI.1. PARTIE CONTROLE

VI.1.1. Description du séquenceur

La partie contrôle de POP est organisée autour du circuit Am 2910 qui calcule une adresse de microinstruction sur 12 bits remplaçant ainsi trois tranches Am 2909.

Bien que la finalité de ces deux séquenceurs soit la même, ils présentent certaines différences :

- Gestion d'un compteur interne sur le 2910
- Jeu d'instruction plus sophistiqué.
- Nouvelle philosophie de prise en compte des conditions externe.

L'adresse de la microinstruction suivante a quatre sources possibles :

- Le microcompteur ordinal (µPC) chargé avec l'adresse de la microinstruction courante incrémentée ou non de 1 en fonction de l'état de la retenue entrante CI.
- Le sommet d'une pile interne de profondeur 5 mots mémorisant les adresses de retour de sous microprogrammes (possibilité d'imbrication de cinq niveaux de procédures).
- La sortie d'un registre-compteur AR chargeable depuis l'extérieur par les entrées Di du circuit et utilisé pour contrôler les boucles.
- Les entrées Di elles-mêmes.

Le PLA de décodage des commandes (entrées I0...I3) sélectionne une de ces sources, gère la pile et le compteur (décrémentation, test de passage à zéro).

Le traitement des conditions ne se fait plus par forçage d'un bit à 1 (entrées OR : des 2909). L'opération de séquencement du 2910 peut être modifiée suivant l'état de l'entrée CC si elle est validée par CCEN. En fait, la condition validée est décodée en

même temps que les entrées I0...I3 par le PLA (voir le tableau des instructions du 2910).

L'initialisation ne se fait plus par une entrée spéciale (ZERO du 2909) mais par le forçage d'une instruction particulière JZ (Jump to Zéro).

Outre les commandes élémentaires du chemin de données interne au séquenceur et de son contrôle, le PLA génère trois signaux PL, VECT, MAP disponibles en sortie et tel qu'au maximum l'un d'eux soit à l'état bas pendant un microcycle. Ils sont donc disponibles pour contrôler le multiplexage de trois sources "trois états" sur le bus d'entrée D.

La figure VI.1. montre l'organisation interne du séquenceur et la figure VI.2 présente la liste des instructions avec leur codage et fait ressortir les opérations internes de gestion du compteur et les sources autorisées sur D.

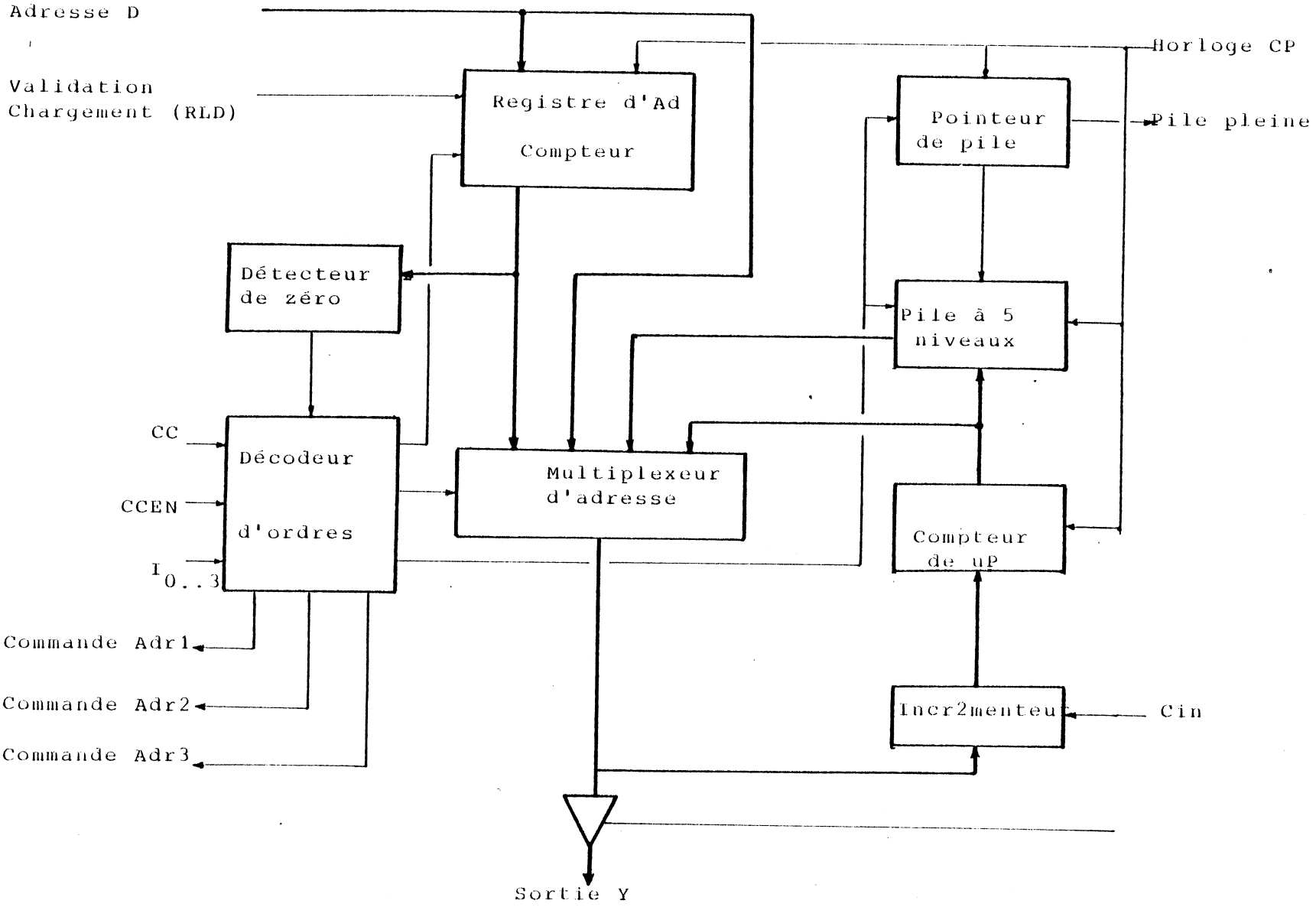


Figure VI.1 -Synoptyque du Am2910

Microcode d'ordres				Signification	[C]	Si condition non réalisée ($\overline{CC}$ = 1 et $\overline{CCEN}$ = 0)			Si condition réalisée ($\overline{CC}$ = 0 ou $\overline{CCEN}$ = 1)		
I_3	I_2	I_1	I_0			Sortie Y	Action Pile	Compteur de boucle	Sortie Y	Action Pile	Compteur de boucle
O	O	O	O	Branchement à l'adresse O	X	O	RAZ	X	O	RAZ	X
O	O	O	I	Appel conditionnel de sous µ P à D2	X	µ PC	X	X	D2	PUSH	X
O	O	I	O	Branchement à l'adresse D1	X	D1	X	X	D1	X	X
O	O	I	I	Branchement cond. à l'adresse D2	X	µPC	X	X	D2	X	X
O	I	O	O	PUSH + charg. conditionnel du C.B.	X	µPC	PUSH	X	µPC	PUSH	LOAD
O	I	O	I	Appel de sous µP à l'adresse R ou D2	X	(R)	PUSH	X	D2	PUSH	X
O	I	I	O	Branchement cond. à l'adresse D3	X	µ PC	X	X	D3	X	X
O	I	I	I	Branchement à l'adresse R ou D2	X	(R)	X	X	D2	X	X
I	O	O	O	Répétition de boucle (avec adresse de début de boucle en pile)	≠ 0	Pile	X	DECR	Pile	X	DECR
					= 0	µPC	POP	X	µPC	POP	X
I	O	O	I	Répétition de boucle (avec adresse de début de boucle = D2)	≠ 0	D2	X	DECR	D2	X	DECR
					= 0	µPC	X	X	µPC	X	X
I	O	I	O	Retour conditionnel de sous µP	X	µPC	X	X	Pile	POP	X
I	O	I	I	Cond. POP + branchement à D2	X	µPC	X	X	D2	POP	X
I	I	O	O	Poursuite en séquence + charg. du C.B.	X	µPC	X	LOAD	µPC	X	LOAD
I	I	O	I	Test de fin de boucle	X	Pile	X	X	µPC	POP	X
I	I	I	O	Poursuite en séquence	X	µPC	X	X	µPC	X	X
I	I	I	I	Branchement a 3 voies	≠ 0	Pile	X	DEC	µPC	POP	DECR
					= 0	D2	POP	X	µPC	POP	X

Figure VI.2 -Instructions du Am2910

VI.1.2. Besoins en branchement

Nous distinguerons quatre types de branchements possibles :

a) Branchement à une adresse issue du mécanisme de décodage de l'instruction se trouvant dans RIPOP et déterminant un point d'entrée dans le microprogramme.

b) Branchement à un sous microprogramme étiqueté.

c) Branchement à l'adresse de la microinstruction courante (résolution des attentes).

d) Branchement à l'adresse 0 pour l'initialisation.

Chaque type de branchement peut être conditionné par l'apparition d'un évènement extérieur. Le quatrième est exécuté au moment de la mise sous tension par forçage de l'instruction JZ.

Le décodage des instructions se fait en deux étapes : un premier décodage des 6 bits poids fort de RIPOP, puis un décodage plus fin prenant en compte RI 0 et RI 1 mais aussi le (ou les) type(s) de base du (ou des) opérande(s).

Le mécanisme de décodage est connecté au bus D du séquenceur par deux ports "trois états" appelés adresse du premier décodage et adresse du second décodage et commandés respectivement par les sorties VECT et MAP du séquenceur.

La sortie PL est disponibles pour les branchements à des séquences étiquetées en commandant l'ouverture du champ ETIQ de la microinstruction sur le bus D.

Comme pour les autres processeurs, les microinstructions d'attente d'un évènement nécessitent l'emploi d'un registre ETOILE extérieur au séquenceur.

Les entrées CC et CCEN sont utilisées pour traiter les microinstructions conditionnelles par modification du séquencement, tandis que les éclatements à plusieurs directions du microprogramme demandent l'adjonction de portes OU sur les entrées D du séquenceur.

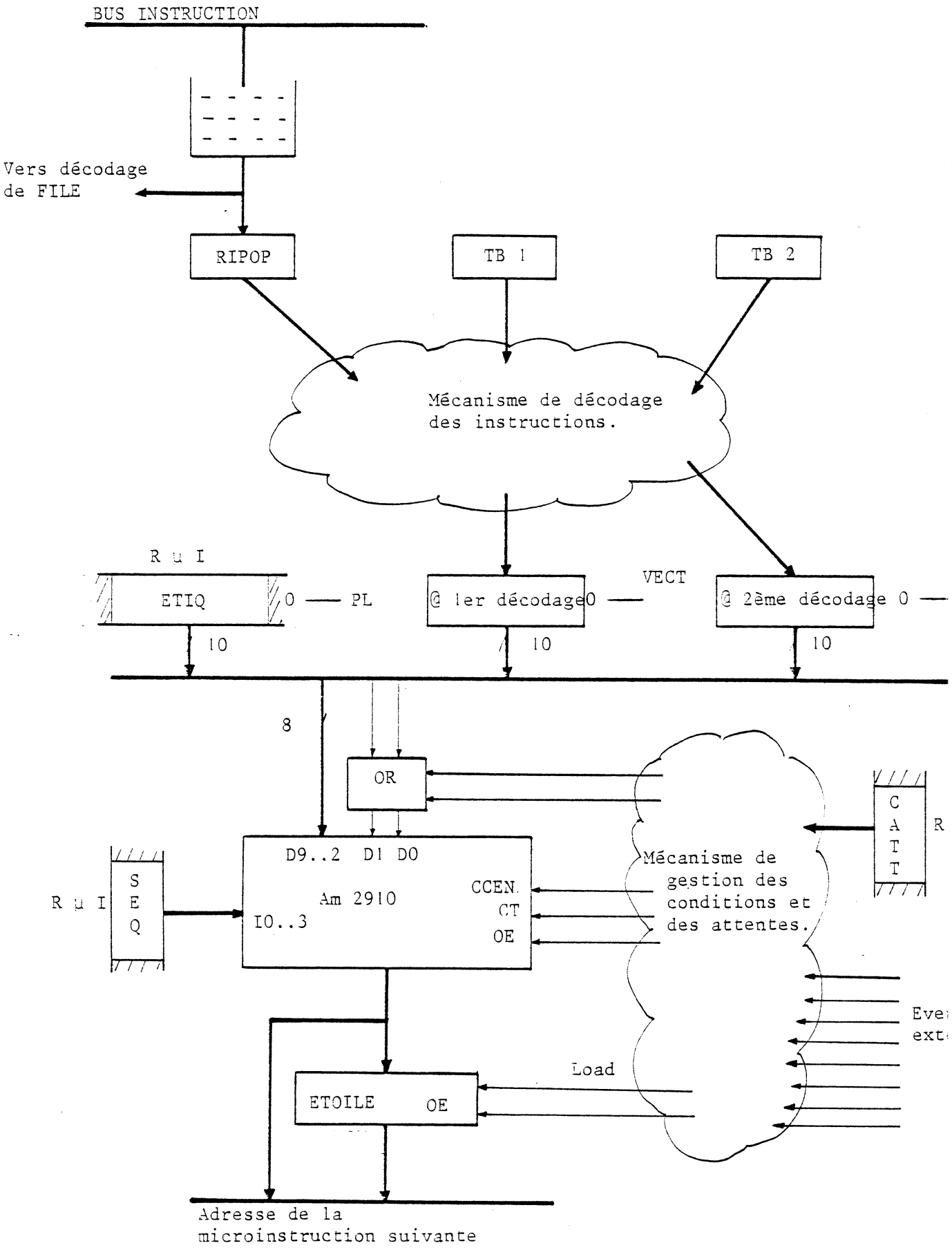


Figure VI.3. - Génération de l'adresse de la microinstruction suivante.

VI.1.3. Décodage des instructions

Les instructions reçues par POP sont réparties en 64 classes et 4 instructions. A chaque classe est associé un ensemble de caractéristiques consignées dans une mémoire morte de 64 mots de 16 bits appelée CARAC et adressée par les 6 bits poids fort de l'instruction (RI 2..RI 7).

Pour certaines instructions de contrôle, la seule information d'appartenance à une classe est suffisante pour son décodage, les bits RI 0 et RI 1 n'ont aucune signification. Pour d'autre il sera nécessaire de les décoder :

par exemple l'opérateur ADD appartient à la classe 18 des opérateurs binaires au même titre que MULT, pour les distinguer on doit connaître leur numéro dans la classe.

Le décodage des instructions se fera en une ou deux étapes selon leurs caractéristiques. On distinguera les caractéristiques pour le premier décodage des caractéristiques pour le second décodage.

VI.1.3.1. Les caractéristiques d'une classe pour le premier décodage

Elles vont permettre de calculer l'adresse du point d'entrée dans le microprogramme pour le premier décodage. Ce sont :

- C0
- C1
- C2

} donnent le numéro de classe pour le premier décodage.

- VIV2 autorise la prise en compte de IV2 (bit de PF2, préfixe du 2ème opérande) pour générer le bit A0 de l'adresse.

Lorsque C0.C1 = 1 la prise en compte de IV1 (bit de PF1 préfixe du premier opérande) est validée pour générer A1, donc toutes les classes dont le numéro se termine par 3 ou 7 prennent IV1 en compte).

- VI0 indiquent que l'on doit tenir compte des valeurs de RI0 et RI1 pour générer l'adresse du deuxième décodage.

Comme on valide soit aucun des bits RI0, RI1, soit RI1 tout seul, soit RI0 et RI1, la combinaison RI0 seul validé est disponible. Elle est utilisée pour prendre en compte les sorties NI0 et NI1 de la ROM CARAC pour construire l'adresse du premier décodage.

En définitive, l'adresse du premier décodage est construite de la façon suivante :

```
A0  = IV2 . VIV2
A1  = IV1 .(C0.C1)
A2  = "1"
A3  = C0
A4  = C1
A5  = C2
A6  = NI0 (VI0.VI1)
A7  = "1"
A8  = NI1 (VI0.VI1)
```

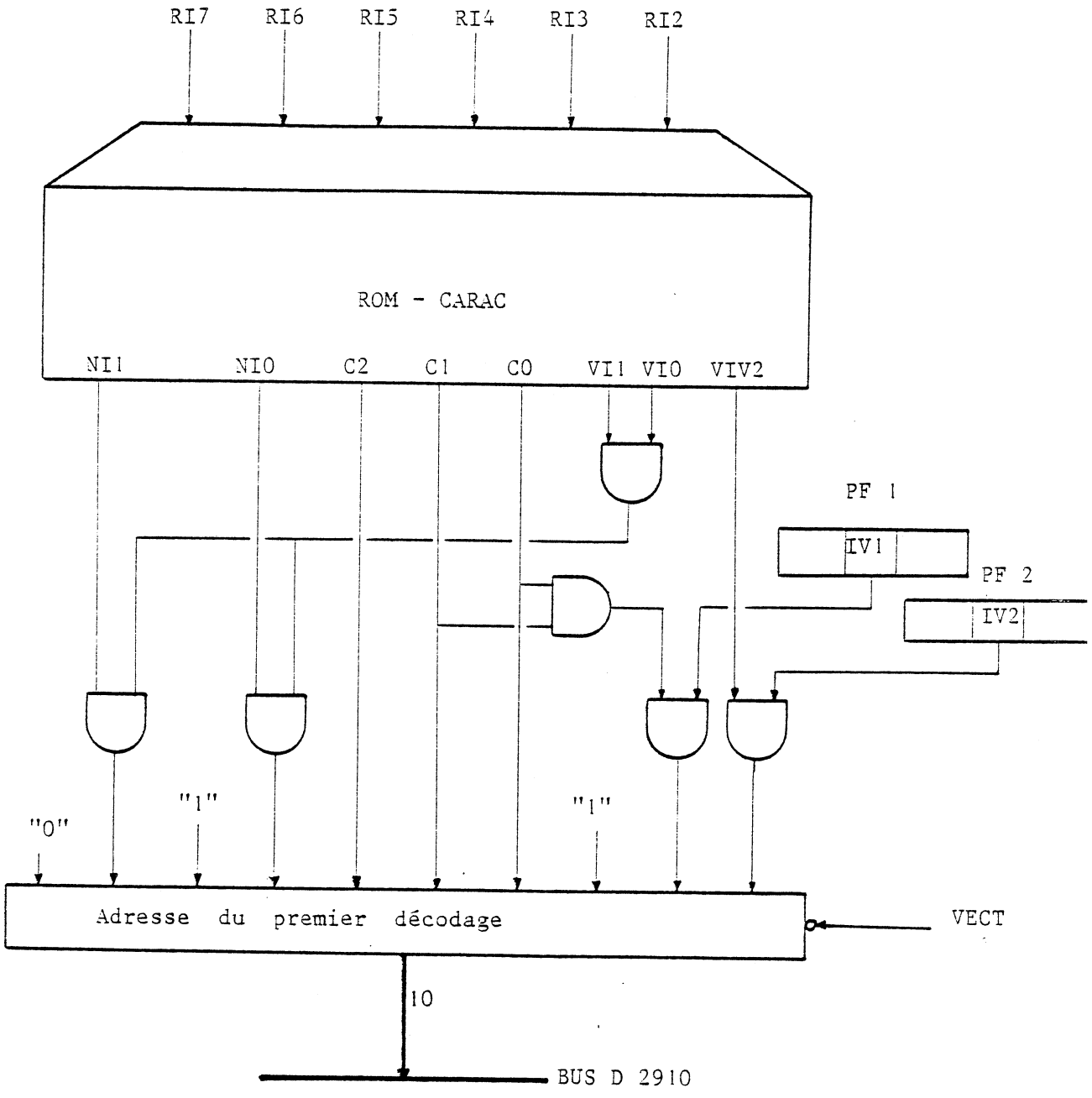


Figure VI.4. - Construction de l'adresse du premier décodage.

Ce qui donne les points d'entrée suivants :

N.CLASSE	(NI1,NI0)	page	ADRESSE					INSTRUCTION
0	-	0	84	85				UNAIRE (IV2)
1	RESERVE		8C	(8D)				INCA
2			94	95				
3	POUR		9C	9D	9E	9F	(IV1)	BINAIRE(IV2)
4			A4	A5				AFFA (IV2)
5	2EME		AC	(AD)				CLRV
6	DECODAGE		B4	B5				INDX (IV2)
7			BC	BD	BE	BF	(IV1)	
0	1	0	C4	C5				TVRAI (IV2)
1	1		CC	CD				TVF (IV2)
2	1		D4	D5				TFAUX (IV2)
3	1		DC	DD	DE	DF	(IV1)	
4	1		E4	(E5)				ROF
5	1		EC	(ED)				INCAI
6	1		F4	(F5)				FIELD
7	1		FC	FD	FE	FF	(IV1)	
0	2	1	84	(85)				SAVEG
1	2		8C	(8D)				EXITFOR
2	2		94	(95)				ENT-EXT
3	2		9C	(9D)	9E	(9F)	(IV1)	CODINV
4	2		A4	(A5)				RET-EXT
5	2		AC	(AD)				
6	2		B4	B5				CASE (IV2)
7	2		BC	BD	BE	BF	(IV1)	
0	3	1	C4	(C5)				OF
1	3		CC	(CD)				RESTAURE
2	3		D4	(D5)				AVANCE
3	3		DC	DD	DE	DF	(IV1)	
4	3		E4	(E5)				INIT
5	3		EC	(ED)				RECULE
6	3		F4	(F4)				SAUVE
7	3		FC	FD	FE	FF	(IV1)	

Les points d'entrée du premier décodage sont soulignés.

Figure VI.5. - Points d'entrées issus du premier décodage.

VI.1.3.2. Les caractéristiques d'une classe pour le 2ème décodage

On distingue, pour le 2ème décodage, les opérations unaires et les opérations binaires.

2ème décodage pour les opérations unaires :

On combine dans ce cas un numéro d'instruction (NI3, NI2, NI1, NI0) issu de la ROM des caractéristiques, et la valeur du type de base qui est soit TB1 soit TB2.
Cette dernière distinction est faite par le numéro de type (NT3, NT2, NT1, NT0) issu de la ROM des caractéristiques en définissant de vrais unaires

(UN = 1 quand NT3.NT2 = 1)

et les faux binaires

(UNV = UN + NT3.NT1)

qui utilisent TB1.

Le numéro d'instruction peut également être modifié en prenant en compte les deux bits du registre instruction RI1 et RI0, selon les valeurs de VI0 et VI1.

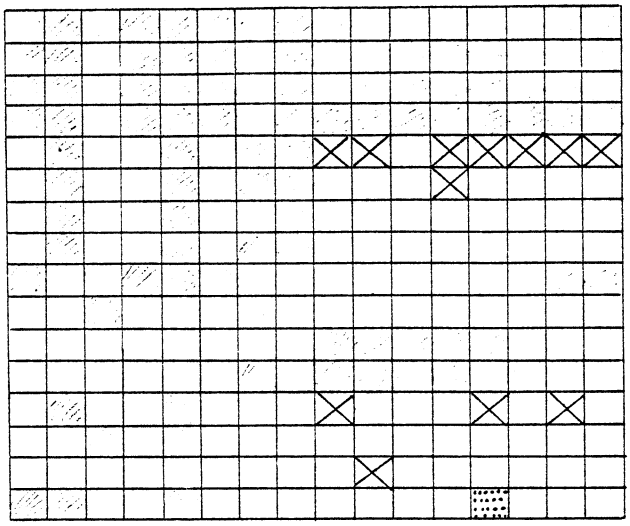
N.INSTRUCTION	VI1 VI0	(1) ADRESSES		INSTRUCTIONS
0	x	1	00	INCA/DECA
(1)		1	10	CLRA/SETA
2		1	20	INXI
3		1	30	INCI/DECI
4		1	40	CLRV/SETV
5		1	50	INCV/DECV
6		1	60	NOT
7		1	70	EQ0/NEQ0/EQ1/NEQ1
8	x	1	80	NEG/ABS
(9)		1	90	TRUNC
(A)		1	A0	INCAI/DECAI
(B)		1	B0	ASSI/ASSXI
C		1	C0	LE0/LT0/GE0/GT0
D		1	D0	LE1/LT1/GE1/GT1
E	x	1	E0	INC/DEC
(F)		1	F0	INT/CHAR

Les adresses générées sont préfixées par UNV : elles appartiennent donc à la page 1. Elles sont par contre post-fixées par TB1 ou TB2.

Tous les points d'entrée du 2ème décodage unaire sont répertoriés dans le tableau suivant, avec le type de Base TB en numéro de ligne, et le numéro d'instruction en numéro de colonne.

TB NI --> 0 1 2 3 4 5 6 7 8 9 A B C D E F

I8 = 0
I16 = 1
I32 = 2
I64 = 3
S8 = 4
S16 = 5
S32 = 6
S64 = 7
C = 8
CS = 9
R32 = A
R64 = B
B = C
BS = D
A = E
RD = F



point d'entrée premier décodage

point d'entrée deuxième décodage

erreur de type unaire

2ème décodage pour les opérations binaires

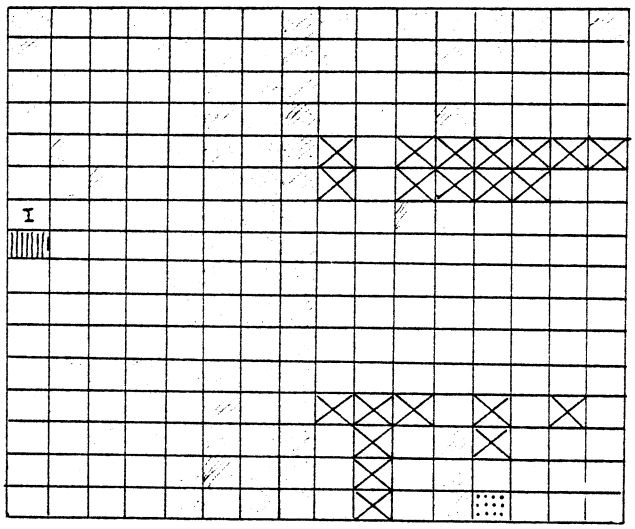
On combine là encore un numéro d'instruction (NI3, NI2, NI1, NI0) issu de la ROM des caractéristiques, et éventuellement modifié par RI1 et RI0 selon les valeurs de VI0 et VI1, avec une valeur de 4 bits issue d'une ROM appelée COMBI, qui code les combinaisons autorisées de TB1 et TB2 sur 4 bits.

N.INSTRUCTION	VI1 VI0	(0) ADRESSES	INSTRUCTIONS
0	x (x)	00	ADD/SUB
(1)		10	(SUB)
(2)		20	MULT/DIV
(3)		30	(DIV)
4		40	AFFECT
5		50	PARAM
6	x	60	ASSA
(7)		70	ASSX
8	x	80	IDIV/MOD
(9)		90	(MOD)
A		A0	AND/OR/XOR/XAND/NAND/...
B		B0	EQ/NEQ/LT/LE/GT/GE
C		C0	ELEM
D		D0	IN
E		E0	INTER
F		F0	FOR+/FOR

COMBI(TB1,TB2)

NI --> 0 1 2 3 4 5 6 7 8 9 A B C D E F <-- MSB

I8	0
I16	1
I32	2
I64	3
S8	4
S16	5
S32	6
S64	7
C	8
CS	9
R32	A
R64	B
B	C
BS	D
A	E
RD	F



LSB

PACE 0

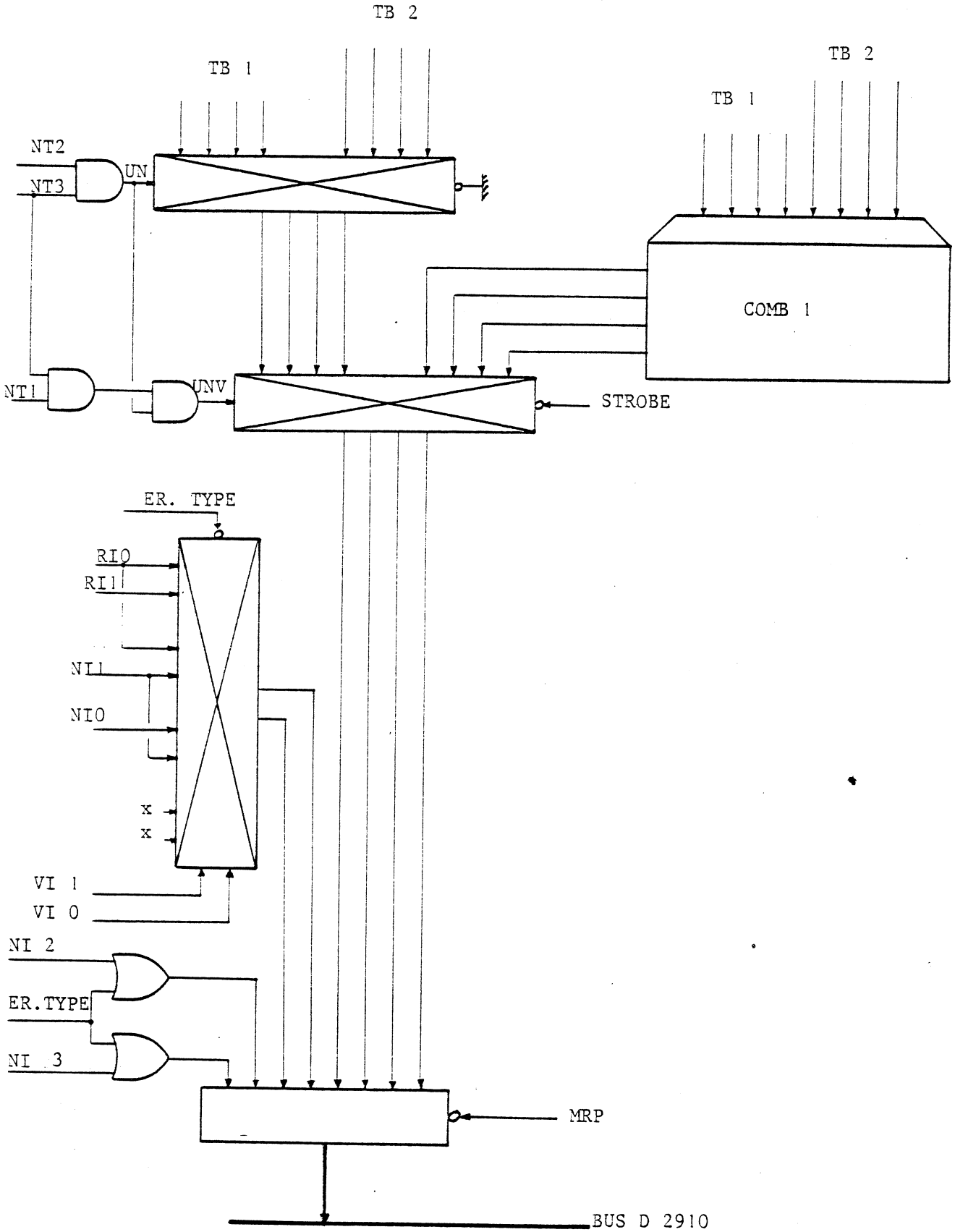
= Pt. d'entrée 1er décodage.
= Pt. d'entrée 2ème décodage
= Erreur Type Binaire (mauvaise combinaison).
= Initialisation
= Erreur externe.

CONTENU DE LA ROM-COMBI (TB1, TB2)

MSB

LSB --- TB2		TB1 0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
I8	0	F	F	D	D	F	F	E	E		8	4	4			F		0
I16	1	F	F	D	D	F	F	E	E		8	4	4			F		1
I32	2	E	E	C	C						8	3	3					2
I64	3	E	E	C	C						8	3	3					3
S8	4	F	F	D	D	9	9	9	9	F								4
S16	5	F	F	D	D	9	9	9	9	F								5
S32	6	E	E	C	C	9	9	9	9	E								6
S64	7	E	E	C	C	9	9	9	9	E								7
C	8					F	F	E	E	F						F		8
CS	9	7	7	7	7						5	7	7					9
R32	A	B	B	A	A						8	2	2					A
R64	B	B	B	A	A						8	2	2					B
B	C													F				C
BS	D														6			D
A	E															1		E
RD	F																0	F

L'adresse du point d'entrée "2ème décodage" dans le microprpgramme est construite de la façon suivante:



Les signaux NT 1, NT2, NT3, NI0, NI1, NI2, NI3 sont issus de CARAC.

Figure VI.6. - Construction de l'adresse du deuxième décodage.

VI.1.3.3. - Les caractéristiques d'une classe pour le contrôle de type

On distingue, pour le contrôle de type, les opérations unaires et les opérations binaires. Dans les deux cas, on combine un numéro de type (NT3, NT2, NT1, NT0), où NT0 est éventuellement modifié par RI1, et le(s) type(s) de base TB1 (TB2) issu(s) du (des) préfixe(s).

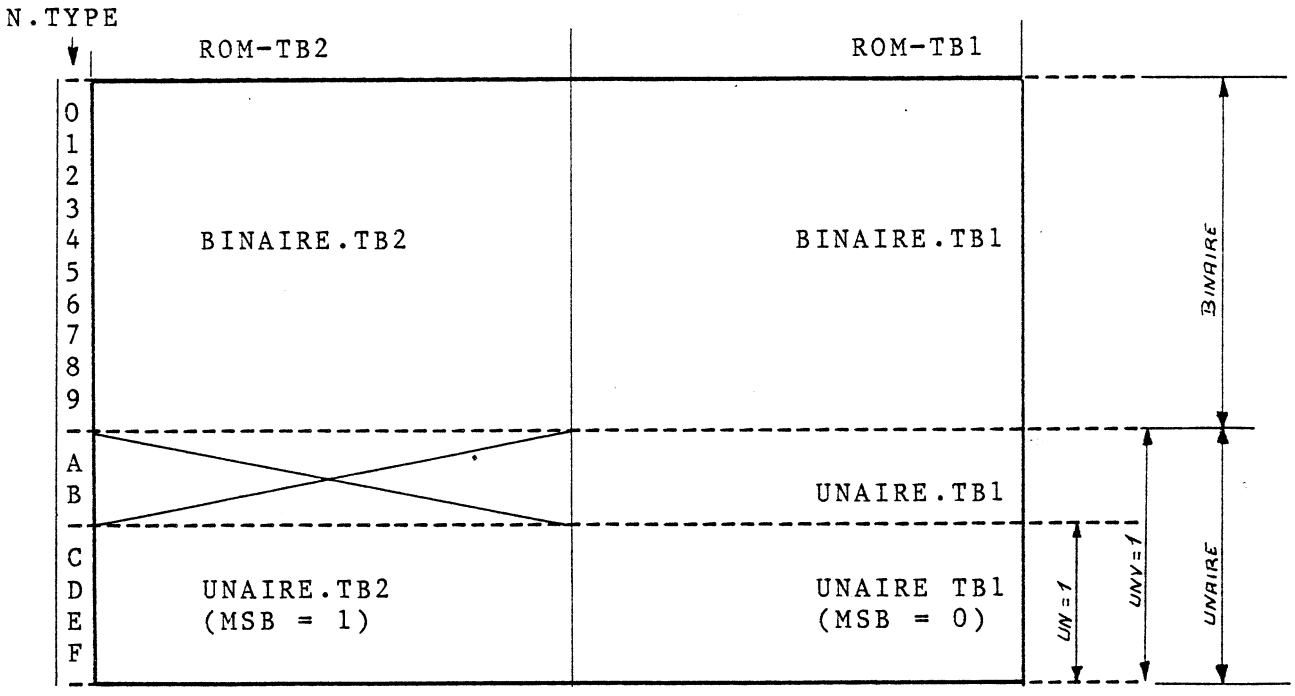
a) - Contrôle de type pour une opération unaire

On dispose de deux ROM de contrôle ROM-TB1 et ROM-TB2.

La ROM-TB1, est adressée par la combinaison
(NT, TB1) si UN = 0
et par la combinaison
(NT, TB2) si UN = 1.

La ROM-TB2 est adressée par la combinaison (NT, TB2).

Les deux ROM sont organisés de la manière suivante :



On distingue ainsi de 10 types d'opérateurs unaires pour le contrôle.

MSB = 1	MSB = 0
	A : TB1
	B : "
C : TB2	C : TB2
D : "	D : "
E : "	E : "
F : "	F : "

Les contrôles de type pour les opérateurs unaires sont les suivants :

ROM-TB1 N. TYPE	INSTRUCTION	TYPES AUTORISES	
A	INCV/DECV	I + C	TB1
B	CLRV/SETV	I + B + S + (PT)	"
C	THEN/WHILE/UNTIL	B	TB2
D	NEG../.../LE0..	I + R	"
E	LE1.../INCA...	I + C + (PT)	"
F	CLRA...	I + B + S + (PT)	"

ROM-TB2 N. TYPE	INSTRUCTION	TYPES AUTORISES	
C	NOT	B + S	TB2
D	EQ0...	I + S + (PT)	"
E	FIELD	RECORD	"
F	INXI	A + CS + BS	

Cas particulier du type PT (Pointeur)

Le bit PT du préfixe indique un Pointeur (sur un type quelconque), il n'apparaît pas dans le type de base TB et ne peut donc pas être contrôlé par les ROM-TB1 ou ROM-TB2. On utilise un bit de caractéristique supplémentaire VPT qui valide sa prise en compte.

L'erreur est égale à ER2 ou ER1 selon la valeur du bit MSB.

Dans le cas où VPT = 1, il y a erreur si PT = 0 ou si ER1, ou ER2 = 1.

INSTR. TB2	INSTR. TB1ouTB2	ROM-TB2	E2	VC2	V12	V02	E1	VC4	VII	V01	ROM-TB2 ROM-TB1	ROM-TB1
	INCVIDECV : I+C	A0-->A3	1	1			1	0			FB	AC-->AF
		A4-->A7	1	1			0	0			F3	A8-->AB
		A8	1	1			1	0			FB	A7
		A9-->AF	1	1			0	0			F3	A0-->A6
	CURV/SETV : I+B+S	B0-->B7	1	1			1	0			FB	B8-->BF
		B8-->BB	1	1			0	0			F3	B4-->B7
		BC	1	1			1	0			FB	B3
		BD-->BF	1	1			0	0			F3	B0-->B2
NOT :B+S	THEN... :B	C0-->C3	0	1			0	0			73	CC-->CF
		C4-->C7	1	1			0	0			F3	C8-->CB
		C8-->CB	0	1			0	0			73	C4-->C7
		CC	1	1			1	0			FB	C3
		CD-->CF	0	1			0	0			73	C0-->C2
EQ0.. :(I+S)	LE0.. :(I+R)	D0-->D3	1	1			1	0			FB	DC-->DF
		D4-->D7	1	1			0	0			F3	D8-->DB
		D8-->D9	0	1			0	0			73	D6-->D7
		DA-->DB	0	1			1	0			7B	D4-->D5
		DC-->DF	0	1			0	0			73	D0-->D3
FIELD :RD	LE1.. :I+C	E0-->E3	0	1			1	0			7B	EC-->EF
		E4-->E7	0	1			0	0			73	E8-->EB
		E8	0	1			1	0			7B	E7
		E9-->EE	0	1			0	0			73	E1-->E6
		EF	1	1			0	0			F3	E0
INXI :A+CS +BS	CURA.. :I+B+S	F0-->F3	0	1			1	0			7B	FC-->FF
		F4-->F8	0	1			1	0			7B	F7-->FB
		F9	1	1			0	0			7B	F6
		FA-->FB	0	1			0	0			73	F4-->F5
		FC	0	1			1	0			7B	F3
		FD	1	1			0	0			F3	F2
		FE	1	1			0	0			F3	F1
		FF	0	1			0	0			73	F0

Contrôle de type UNAIRE par les ROM TB1 et TB2.

Le contrôle d'erreur Unaire est réalisé, en logique négative, par le schéma suivant :

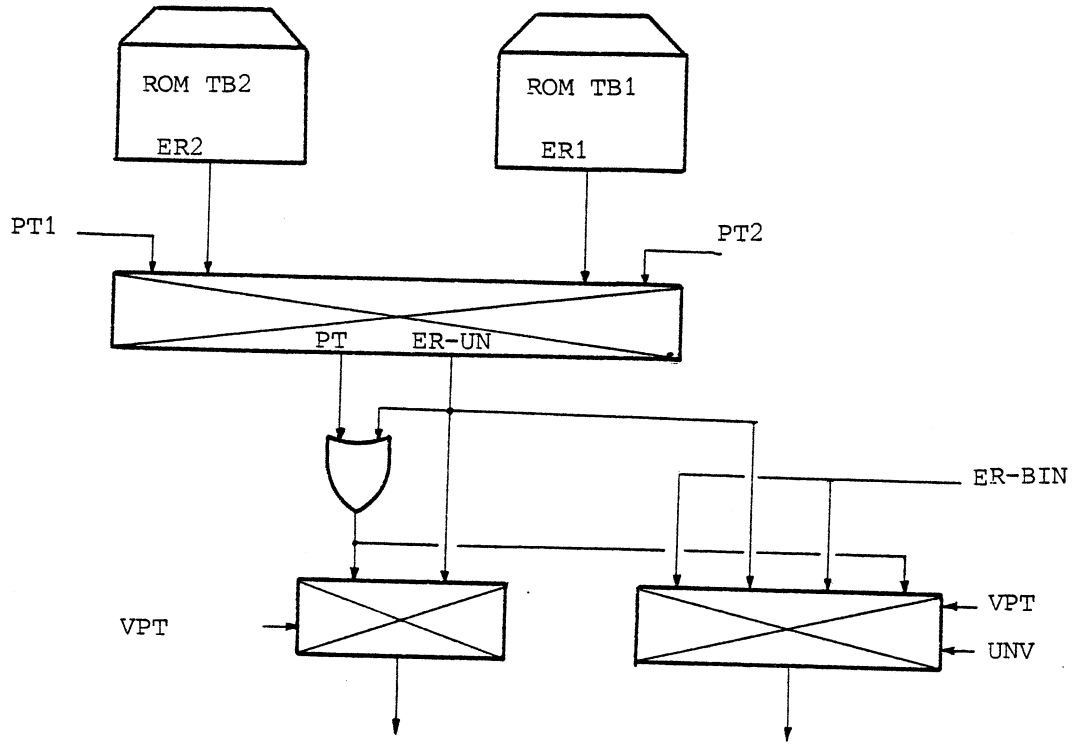


Figure VI.7. - Contrôle de type unaire.

b) Contrôle de type pour une opération binaire :

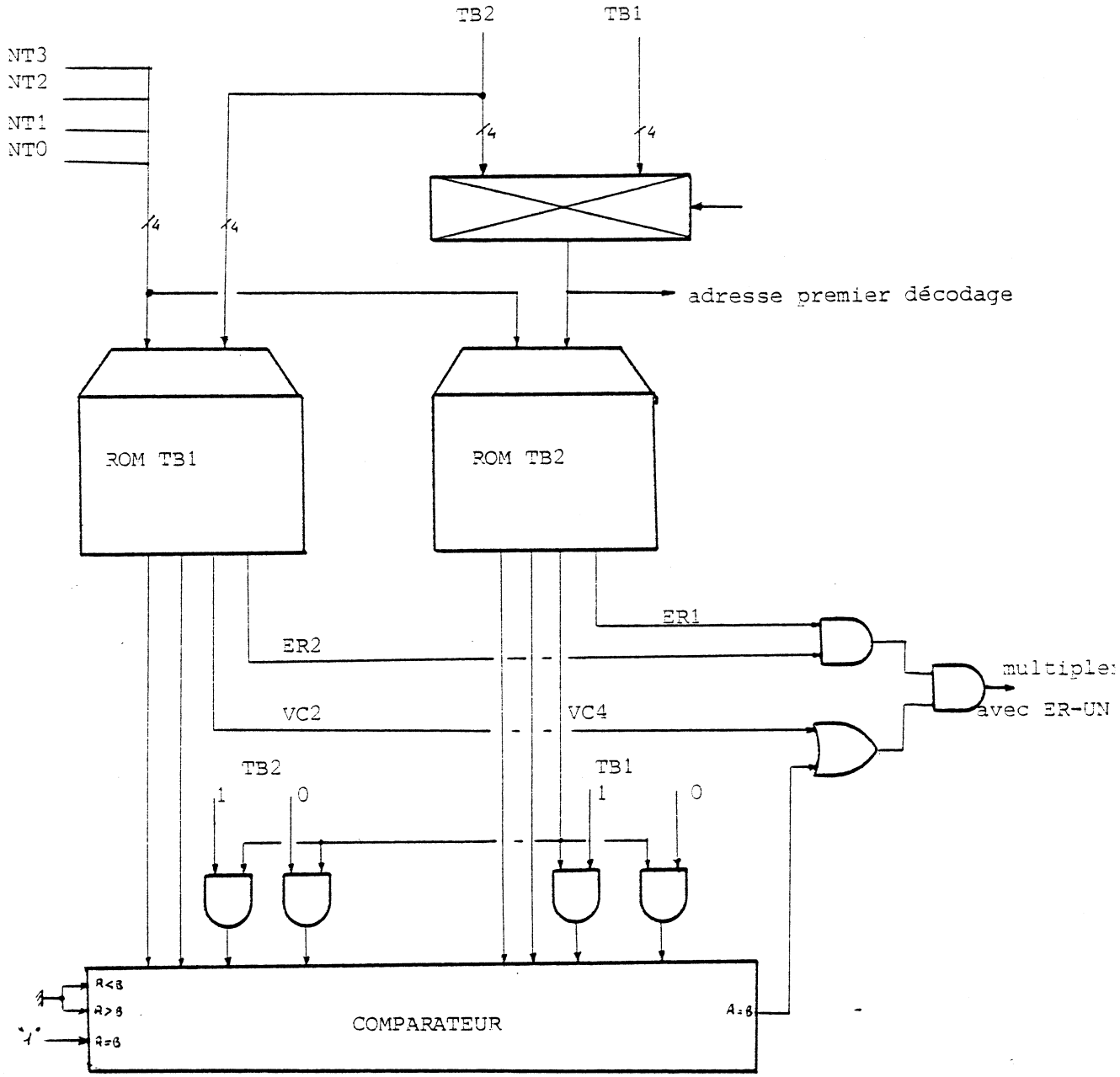
La ROM-TB2 vérifie la combinaison (NT,TB2), et la ROM-TB1 la combinaison (NT,TB1).

Chacune de ces deux ROM sort une "valeur" sur 2 bits, un bit d'erreur et un bit de validation de comparaison :

VC2 : valide la comparaison des 2 valeurs sortantes,

VC4 : valide la comparaison des 2 bits de poids faibles de TB1 et TB2.

La détection d'erreur binaire est faite par le circuit suivant :



Figuer VI.8. - Circuit de détection d'erreur binaire

INSTR COMBINAISONS AUTORISEES	ROM-TB2	E2VC2V12V02 1 1 x x	E1VC1V11V01 1 0 x x	ROM-TB2 ROM-TB1	ROM-TB1
ADD/SUB (I+R).(I+R) MULT/DIV	00-->03 04-->09 0A-->0B 0C-->0F	1 1 0 1 1 1 0 1	1 0 0 0 1 0 0 0	FB 73 FB 73	0C-->0F 06-->0B 04-->05 00-->03
IDIV/MOD I.I	10-->13 14 ->1F	1 1 0 1	1 0 0 0	FB 73	1C-->1F 10 ->1B
LOGIQUE B.B+S.S	20 ->23 24 ->27 28 ->2B 2C 2D ->2F	0 1 1 0 0 0 0 1 1 0 0 1 0 1	0 0 1 0 0 0 0 0 1 0 0 1 0 0	73 88 73 99 73	2C ->2F 28 ->2B 24 ->27 23 20 ->22
EQ/NEQ Type.Type PT.PT	30->33 34->37 38->3B 3C->3F	1 0 0 0 1 0 0 1 1 0 1 0 1 0 1 1	1 0 0 0 1 0 0 1 1 1 1 0 1 1 1 1	88 99 AE BF	3C->3F 38->3B 34->37 30->33
COMPAR (I+R).(I+R) +S.S+C.C	40->43 44->47 48 49 4A->4B 4C->4F	1 0 0 0 1 0 0 1 1 0 1 0 0 1 1 0 0 0 0 1	1 0 0 0 1 0 0 1 1 0 1 0 0 0 1 0 0 0 0 0	88 99 AA 73 88 73	4C->4F 48->4B 47 46 44->45 40->43
ELEM (IO+C).S	50->51 53->53 54->57 58 59->5F	1 1 0 1 0 1 1 1 0 1	0 0 0 0 1 0 0 0 0 0	F3 73 7B F3 73	5E->5F 5C->5D 58->5B 57 50->56
IN S.(IO+C)	60->61 62->63 64->67 68 69->6F	0 1 0 1 1 1 0 1 0 1	1 0 0 0 0 0 1 0 0 0	7B 73 F3 7B 73	6E->6F 6C->6D 68->6B 67 60->66
INDX (A+CS). (IO+C)	70->71 72->77 78 79 7A->7D 7E 7F	1 1 0 1 1 1 0 1 0 1 0 1 0 1	0 0 0 0 0 0 1 0 0 0 1 0 0 0	F3 73 F3 7B 73 7B 73	7E->7F 78->7D 77 76 72->75 71 70
FOR IO.IO+C.C	80->81 82->87 88	1 0 0 0 0 1 1 0 0 1	1 0 0 0 0 0 1 0 0 1	88 73 99	8E->8F 88->8D 87

	89->8F	0	1			0	0			73		80->86				
AFFA Type.Type	90->93	1	0	0	0	1	0	0	0	8	8	9C->9F				
AFFECT +(I+R+CS).	94->97	1	0	0	1	1	1	0	1	9D		98->9B				
(I+R+CS)+	98	1	0	1	0	1	0	1	0	AA		97				
PT.PT	99->9B	1	0	0	0	1	0	0	0	88		94->96				
	9C->9F	1	0	1	1	1	1	1	1	BF		90->93				

VI.1.4. - Conditions et attentes

Cinq bits de la microinstruction (champs CATT) sont utilisés pour commander le mécanisme de gestion des conditions et attentes. On peut donc avoir jusqu'à 32 évènements susceptibles de modifier le déroulement algorithmique du microprogramme de POP.

Nous distinguerons:

- les évènements permettant de débloquer le microprogramme par la levée d'une attente en agissant sur les commandes du registre ETOILE:

  - AMC (registre adresse mémoire disponible)
  - VMC (donnée mémoire disponible)
  - DPOP=0 (état de la bascule de synchronisation avec FILE)
  - BIPOP-non-VIDE

- Les évènements provoquant un branchement bidirectionnel en agissant sur les bits poids faible de l'adresse de branchement:

  - FINI (fin d'échanges avec FILE pour les instructions répétitives)
  - VIDE
    état de la pile interne de POP
  - PLEIN
  - LG4 (la valeur de la variable accédée en mémoire occupe deux mots)
  - NOCL3 (on vient de lire l'octet n.3 dans un mot).

- les couples d'évènements provoquant un éclatement à quatre directions en agissant sur les deux bits poids faibles de l'adresse de branchement:

  - PLEIN, OCC (OCC indique que la valeur de la variable accédée doit être empilée, il faut donc contrôler l'occupation de la pile)
  - RI1, RI0 (2 bits poids faible de RIPOP)
  - NOCE3, NOCL3 (troisième octet d'un mot en écriture, lecture)
  - $TB2^1$, $TB2^0$ (deux bits du champ préfixe (PF) du deuxième opérande)
  - TA1, TA0 (deux bits de la ROM d'accés valeur renseignant sur les tailles de l'opérande).

- les couples d'évènements provoquant d'abord une levée d'attente

et un branchement bidirectionnel:

- Att(VMC);ORO=TX (TX indique une indirection sur la valeur à accéder).

- les évènements susceptibles de modifier les commandes du séquenceur, donc agissant sur l'entrée CC du 2910. Celle-ci est connectée à la sortie CT d'un AM 2904 (status and shift control) prévu pour faciliter l'exécution des opérations arithmétiques complexes en générant les décalages sur les AM 2903 et en contrôlant le microprogramme en fonction des sorties d'état (Z, OVR, COUT) des "super slices".

- test	A > B	non signé
	A > B	signé
	A < B	non signé
	A < B	signé
	A = B	
	A = B	
	< 0	
	< 0	
	OVF	
	OVF	

Les cinq bits du champ CATT codant la sélection d'un évènement (ou d'un couple d'évènements) dans un groupe donné doit être décodé pour commander les multiplexeurs nécessaires.
Là encore, nous avons introduit une mémoire morte de 32 mots de 8 bits pour générer les signaux de validation et de sélection. La figure VI.9. montre la réalisation matérielle du mécanisme de gestion des conditions et des attentes, le codage de la ROM CATT est donné dans la figure VI.10.

Les signaux L64, TX, TA1, TA0 et OCC sont utilisés par le microprogramme d'accés à la valeur d'un opérande en mémoire centrale. Ils sont issus du décodage des bits PT, TB3, TB2, TB1, TB0 du registre de travail sur les préfixes RPF pour la ROM d'Accés Valeur (RAV). Son codage est indiqué par la figure VI.11.

Remarque:
Toutes ces sorties ne sont pas utilisées, on peut donc s'en servir pour gérer des fonctions des entrées adresses et en particulier le signal TB0+TB1 utile au décoage du champ DA.

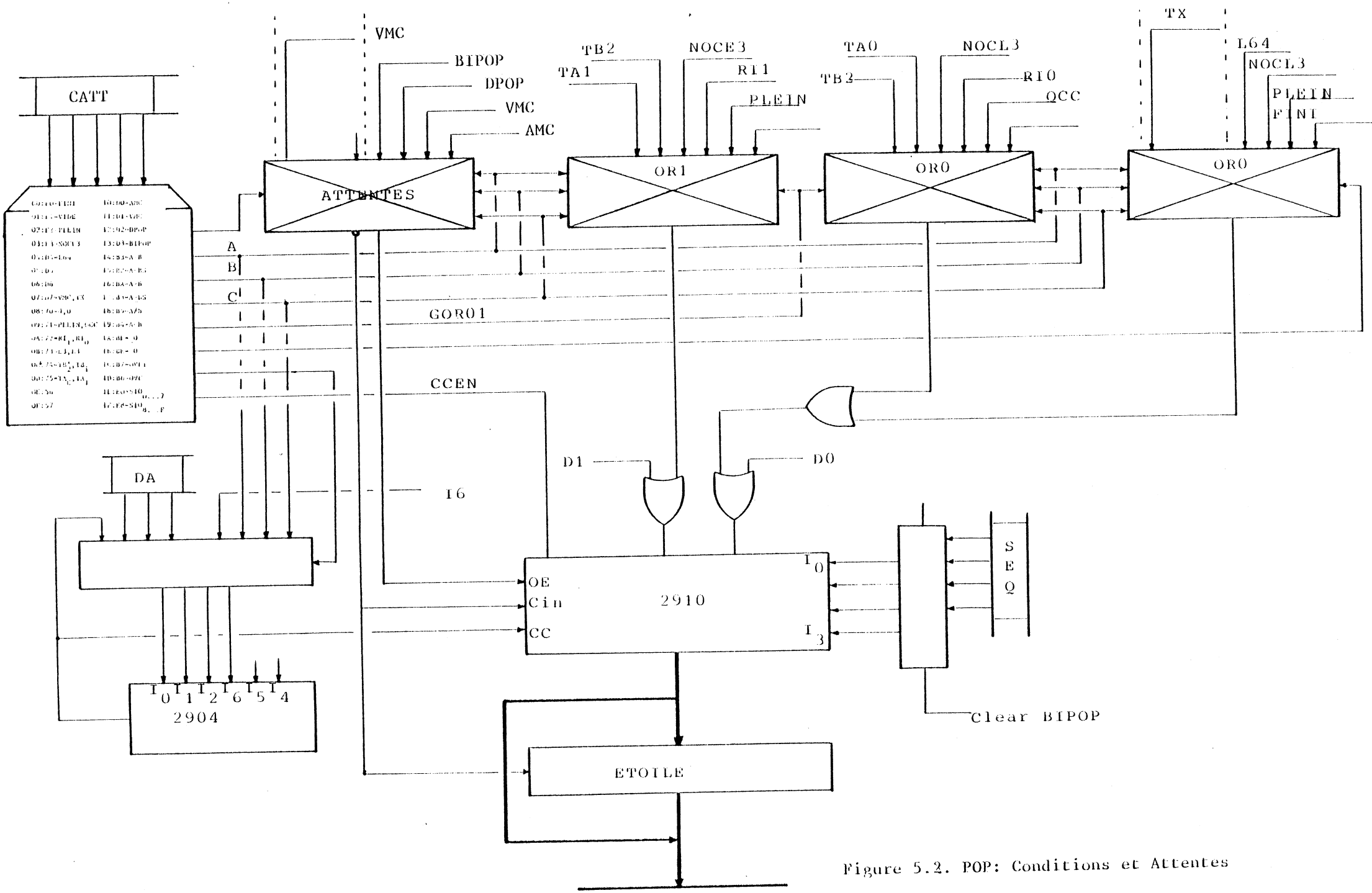


Figure 5.2. POP: Conditions et Attentes

adresses	évènements	valeur
00	OR0 = FINI	F1
01	OR0 = VIDE	F2
02	OR0 = PLEIN	F3
03	OR0 = NOCL3	04
04	OR0 = L64 et Attente (fausse)	D5
05		D6
06		D7
07	OR0 = TX et Attente (VMC)	70
08	OR1 = 0 et OR0 = 0	71
09	OR1= PLEIN et OR0=OCC	72
0A	OR1=RI1 et OR0=RI0	73
0B	OR1=NOCE3 et OR0=NOCL3	74
0C	OR1=TB21 et OR0=TB20	55
0D	OR1=TA1 et OR0=TA0	56
0E		57
0F		
10	ATTENTE(AMC)	D0
11	ATTENTE(VMC)	D1
12	ATTENTE(DPOP=0)	D2
13	ATTENTE(BIPOP-NON-VIDE)	D3
14	TEST A > B NON SIGNE	BB
15	A > B SIGNE	B2
16	A < B NON SIGNE	BA
17	A < B SIGNE	B3
18	A = B	B5
19	A = B	B4
1A	< 0	BF
1B	> 0	BE
1C	OVF	B7
1D	OVF	B6
1E	SI00=0..7 donnée par DA2,DA1,DA0	E0
1F	SI00=8..F "	E8

Figure VI.9.- Codage conditions et attentes

ROM ACCES VALEUR RAV (32 X 8)

PT	TB3	TB2	TB1	TB0	type de base	L64	Occ	TA1	TA0	TX	TB0+TB1	@	valeur
0	0	0	0	0	I8	0	0	0	0	0	0	0 0	0 0
	0	0	0	1	I16	0	0	0	0	0	1	0 1	0 1
	0	0	1	0	I32	0	1	0	0	1	1	0 2	1 3
	0	0	1	1	I64	1	1	1	0	1	1	0 3	3 B
	0	1	0	0	S8	0	0	0	1	0	0	0 4	0 8
	0	1	0	1	S16	0	0	0	1	0	1	0 5	0 9
	0	1	1	0	S32	0	1	0	0	1	1	0 6	1 3
	0	1	1	1	SV	0	0	1	1	X	1	0 7	0 D
	1	0	0	0	C	0	0	0	1	0	0	0 8	0 4
	1	0	0	1	CS	0	0	1	1	X	1	0 9	0 D
	1	0	1	0	R32	1	1	1	0	0	1	0 A	3 9
	1	0	1	1	R64	1	1	1	0	1	1	0 B	3 B
	1	1	0	0	B	0	0	0	0	0	0	0 C	0 0
	1	1	0	1	BS	0	0	0	1	0	1	0 D	0 5
	1	1	1	0	ARRAY	0	0	1	1	X	1	0 E	0 D
	1	1	1	1	RECORD	0	0	1	1	X	1	0 F	0 D
1	X	X	X	X	POINTER	0	0	0	1	0	1	10 -> 1F	0 5

figure VI.10.

VI.2. - PARTIE OPERATIVE

L'organisation générale du chemin des données a été décrite dans le chapître 2. On ne s'intéressera ici qu'aux éléments le constituant et à leurs commandes.

Les variables manipulées par POP sont en général de 16 bits (adresse du champ PV des descripteurs).
L'unité arithmétique et logique sera donc constituée de quatre tranches AM 2903 ("super slice") cascadées pour opérer sur 16 bits.
Ces macrocomposants présentent quelques améliorations par rapport aux 2901 utilisés pour la réalisation des autres processeurs.

Ils sont plus orientés calcul arithmétique: en effet, de nouvelles instructions ont été rajoutées pour permettre les multiplications et divisions sur des variables codées en "complément à deux" ou en "valeur absolue + signe" sans rajouter de matériel extérieur et avec une microprogrammation simplifiée, surtout si on utilise le séquenceur AM 2910 possédant un compteur de boucle et le circuit AM 2904 pour générer les décalages nécessaires.

Les possibilités d'entrée/sortie ont été étendues: le AM 2903 possède deux bus d'entrée DA et DB permettant une extension des registres internes de l'opérateur sans ne rien perdre de l'architecture à deux adresses: on peut réaliser toutes les opérations entre deux registres internes, un registre interne et un port d'entrée, ou entre deux ports d'entrée.

VI.2.1. - Les chemins de données

Le coeur du chemin de données est l'unité arithmétique et logique et on va distinguer le chemin de données entrant (bus D) sur lequel sont connectés les éléments sources, du chemin de données sortant (bus Y) vers les éléments destinations. Les figures VI.11 et VI.12 les schématisent et font apparaître la distinction poids fort (éléments d'indice 1) et poids faible (éléments d'indice 0) pour les registres ou autres éléments de mémorisation de largeur 32 bits.

Remarques:

- Les éléments poids faibles correspondant à (PF1, PT1) et (PF2, PT2) sont les registres RO_1 et RO_2 internes à l'UAL.

- Le bus M issu du multiplexage de la sortie Y avec le bus 1 permet de charger les destinations poids fort soit avec la sortie de l'opérateur, soit avec le bus d'entrée poids fort.

Certains transferts de 32 bits sont ainsi rendus possibles en une seule microinstruction:

- le transfert des poids faibles se fait par le bus 0 et le

bus Y via l'UAL,
- tandis que le transfert des poids forts se fait par le bus 1 et le bus M (exemple transfert mémoire-pile).

Tout ce chemin de données est géré par trois champs de microinstructions:

- un champ entrée DA pour le multiplexage du bus 0, de BIPOP et du champ valeur immédiate et pour toutes les commandes d'extension de signe, de croisement d'octet etc... à l'entrée de l'opérateur.

- un champ destination Y commandant tous les chargements possibles du registre d'adresse mémoire (RADM) et les destinations poids faibles.

- un champ BUS 01 gérant tout le reste: l'origine du bus 0, l'origine du bus 1, la destination du bus 1 et les commandes du multiplexeur M.

Pour être complet, il faut rajouter:

- le champ CODOP commandant l'UAL,

- le champ valeur immédaite (VI),

- le champ ORDRE qui permet le positionnement des bascules d'état, de synchronisation, de dialogue ou d'erreur mais aussi certains chargements particuliers.

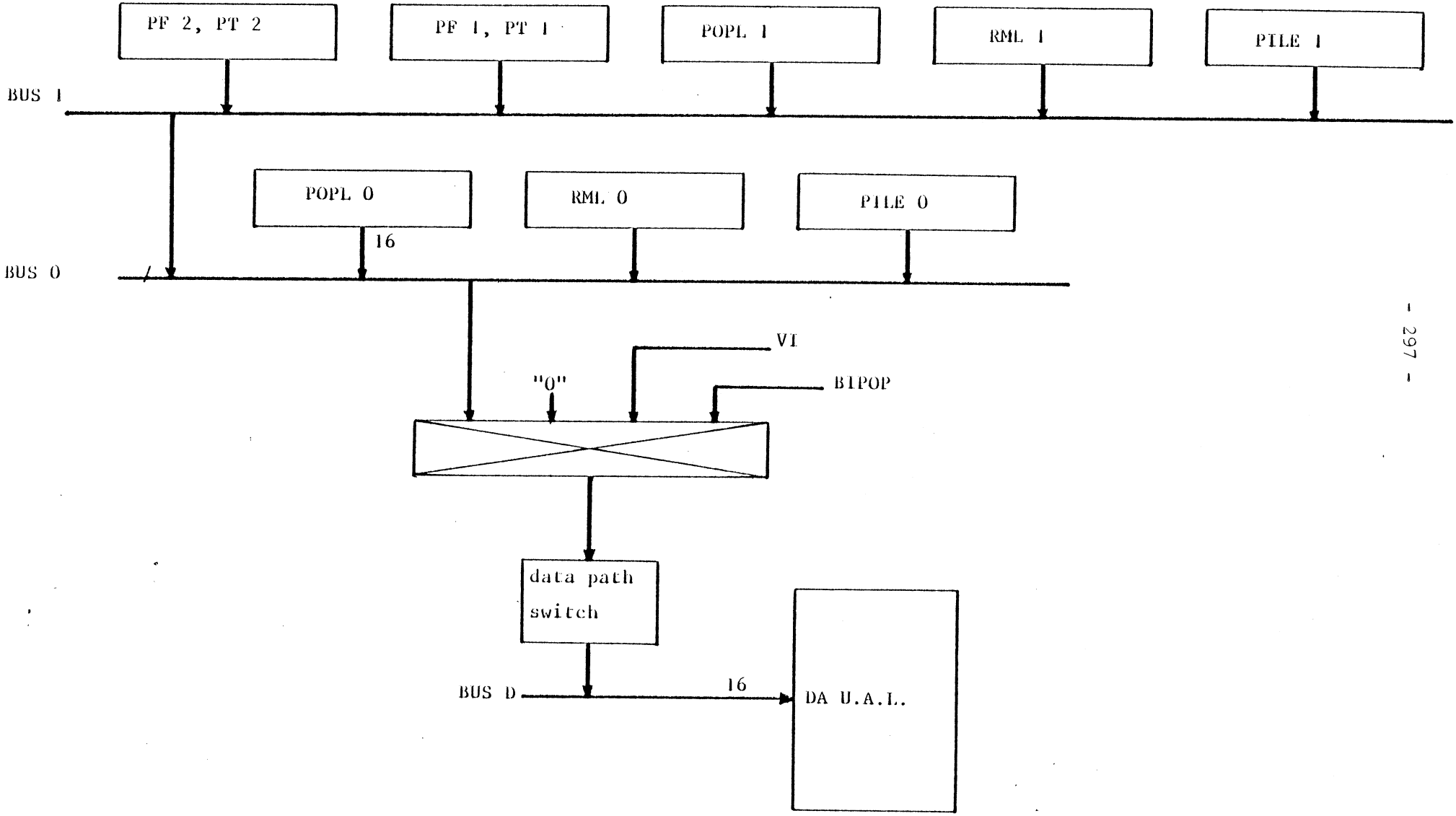


Figure VI.11. - Entrée de l'UAL de POP.

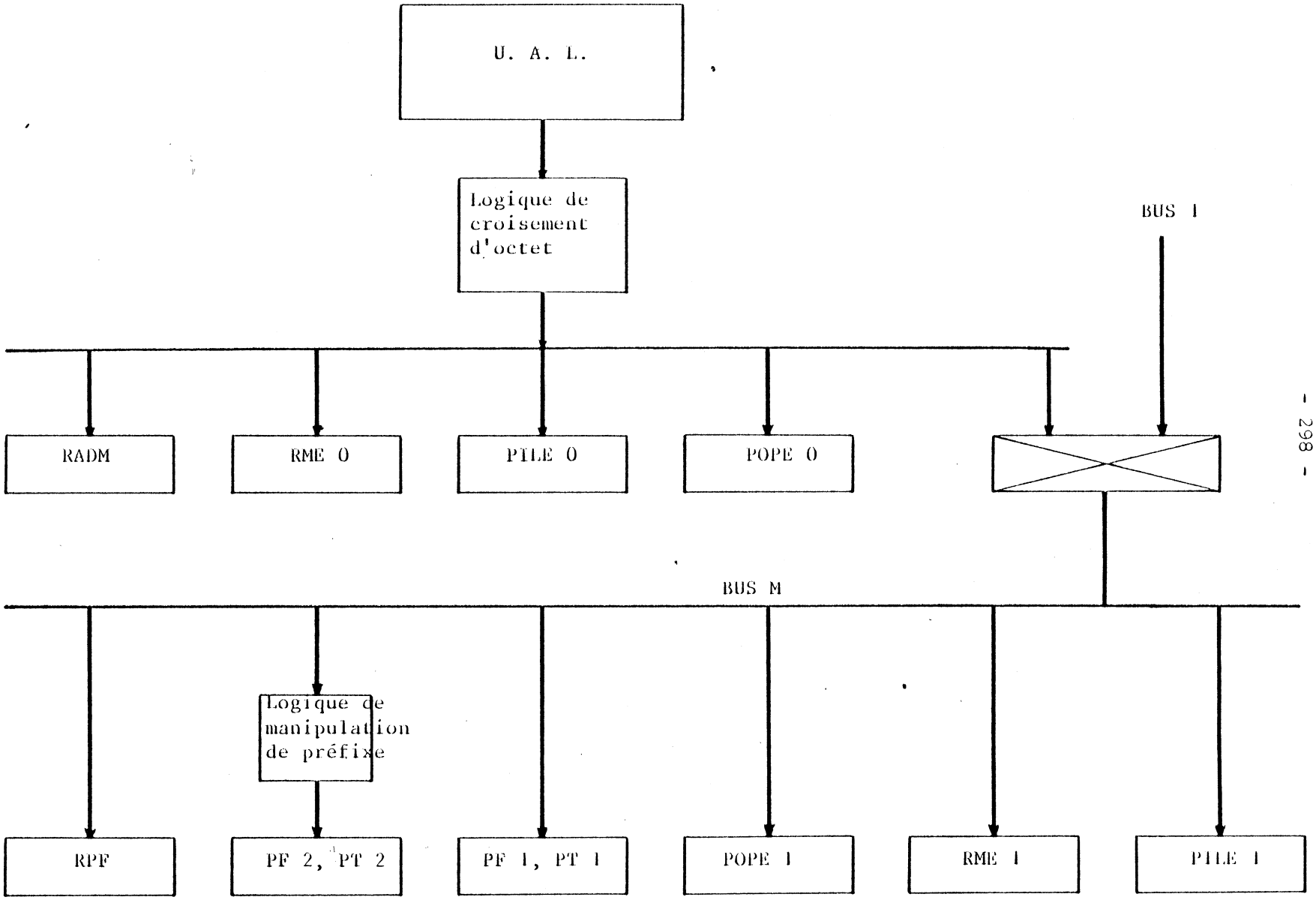


Figure VI.12. - Sortie de l'UAL de POP.

VI.2.1.1. Entrées DA des 2903

Sur l'entrée DA de l'opérateur sont multiplexés:

- le BUS
- la sortie de BIPOP (sur 8 bits)
- une valeur immédiate (sur 8 bits)
- le compteur de pile N (sur 4 bits).

Certaines données manipulées sur BUS 0 sont des octets mais elles ne se trouvent pas obligatoirement en poids faible. Il faut donc pouvoir croiser les deux octets d'un demi-mot avant d'étendre l'octet poids faible, soit avec des zéros, soit avec le bit de signe avant qu'il rentre dans l'opérateur 16 bits.

Les besoins de multiplexage du bus DA sont résumés dans la figure VI.13.

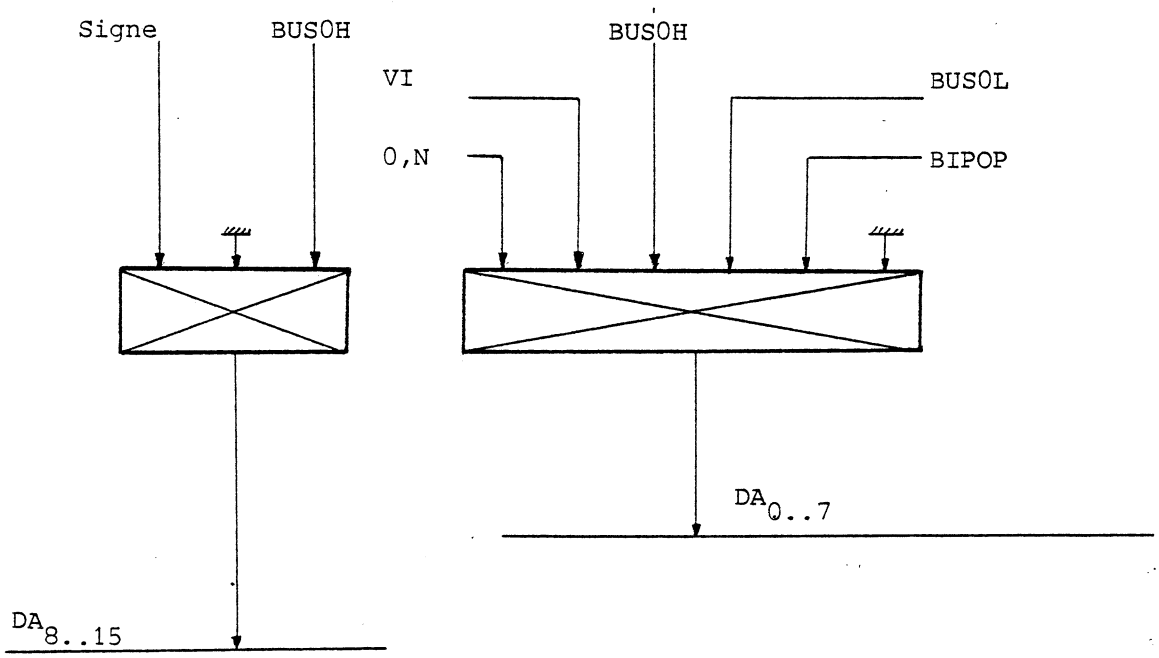


Figure VI.13 - Besoins de multiplexage sur DA.

Toutes les combinaisons entre octet poids fort et octet poids faible ne sont pas autorisées. Celles qui sont autorisées sont énumérées dans le tableau suivant:

octet poids fort	octet poids faible	
BUSOH	BUSOL	
0	BUSOL	
SIGNE	BUSOL	
0	BUSOH	
SIGNE	BUSOH	
0	VI	
SIGNE	VI	
0	BIPOP	
SIGNE	BIPOP	
0	0,N	
BUSOH	0	
SIGNE BUSOH	BUSOL	
SIGNE BUSOH	BUSOL/H BUSOL	la distinction se fait suivant le type de base de l'opérande (combinaison des 2 bits TB0 et TB1 de RPF)
SIGNE BUSOH	BUSOH/L BUSOL	

Nous avons six sources possible pour l'octet poids faible (dont l'une est à "0") et trois pour l'octet poids fort (dont l'une est "0").

La réalisation matérielle demande 8 multiplexeurs , quatre vers un (4 boîtiers 74LS153) pour $DA_{0..7}$.
L'une des entrées est multiplexée entre le compteur de pile N et les 4 bits poids faible de BIPOP.
Le forçage à 0 se fait en agissant sur l'entrée de validation.

Cinq commandes sont nécessaires:

SL0 et SL1 sélectionnent une voie parmi 4
SN choisit entre N et $BIPOP_{0...3}$
GL0 et GL1 valident les multiplexeurs.

Pour $DA_{8...15}$ on utilise 8 multiplexeurs, deux vers un (2 boîtiers 74LS157):

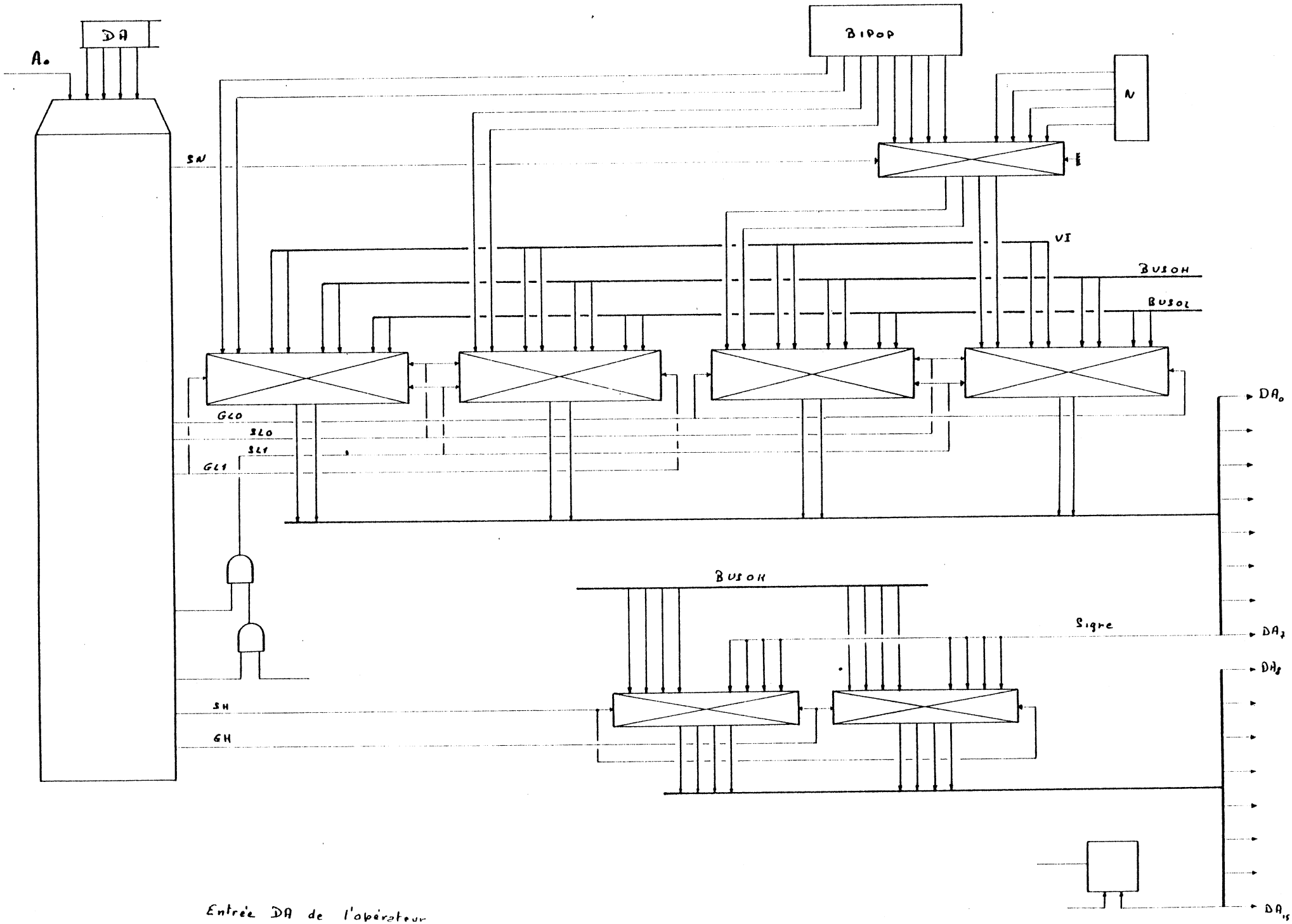
- sur une voie est câblé BUS0H,

- sur l'autre, le signe de l'octet poids faible (DA7).

SH commande ce choix tandis que le forçage à 0 se fait par validation GH des multiplexeurs.

Tous ces signaux de commande sont issus de la matrice de décodage ROM-DA adressé par 4 bits de la microinstruction (champ DA) et par le signal TB0+TB1 généré par la ROM RAV et indiquant si l'opérande est ou n'est pas un octet.

La figure VI.14 donne le schéma de câblage de l'entrée DA de l'opérateur et la figure VI.15 le codage de ROM DA.

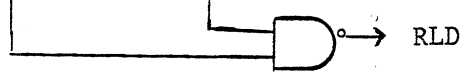


Entrée DA de l'opérateur

Figure V.15. PROM COMMANDE ENTREE DA

ADRESSE	FONCTION	$\overline{VNOCL\ 0}$	$\overline{GH}$	SH	$\overline{GL\ 1}$	$\overline{GL\ 0}$	SN	$\frac{SL\ 1}{SL\ 0}$	VALEUR HEXA
00	BUS H, BUS L	0	0	1	0	0	x	01	21
01									21
02	0, BUS L	0	1	x	0	0	x	01	41
03									41
04	SG, BUS L	0	0	0	0	0	x	01	01
05									01
06	0, BUS H	0	1	x	0	0	x	00	40
07									40
08	SG, BUS H	0	0	0	0	0	x	00	00
09									00
0A	0, VI	0	1	x	0	0	x	11	43
0B									43
0C	SG, VI	0	0	0	0	0	x	11	03
0D									03
0E	0, BIPOP	0	1	x	0	0	0	10	42
0F									42
10	SG, BIPOP	0	0	0	0	0	0	10	02
11									02
12	0, (0,N)	0	1	x	1	0	1	10	56
13									56
14	BUS H, 0	0	0	1	1	1	x	xx	38
15									38
16	SG, BUS L	0	0	0	0	0	x	01	01
17	BUS H, BUS L	0	0	1	0	0	x	01	21
18	SG, BUS H/L	1	0	0	0	0	x	01	81
19	BUS H, BUS L	1	0	1	0	0	x	01	A1
1A	0, BUS H/L	1	1	x	0	0	x	01	C1
1B	BUS H, BUS L	1	0	1	0	0	x	01	A1
1C	0, VI et $\overline{RLD}$	0	1	1	0	0	1	11	67
1D		0	1	1	0	0	1	11	67
1E									
1F									

A0 = TB1+TB0 (2 bits LSB de RPF)



VI.2.1.2. Sorties Y des 2903

Besoins en chargement

Du point de vue destination du résultat, les opérations exécutées par l'UAL peuvent être classées en 5 types:

- calcul d'adresse se traduisant par le chargement de RADM et éventuellement la génération d'un masque d'écriture.

- opération arithmétique ou logique dont le résultat doit être rangé soit dans POPE0, soit dans PILE0, soit dans PILE1.

- opération d'écriture mémoire (fin d'expression) impliquant le chargement de RME0 ou de RME1 avec ou sans croisement des octets.

- modification de préfixe nécessitant le chargement de POPE1.

- sauvegarde en mémoire.

calcul d'adresse

Le processeur POP a la possibilité d'accéder aux quatre zones de la mémoire centrale. Les deux bits poids fort A15, A14 de l'adresse en indique le numéro:

A15	A14	Zone
0	0	CTX
0	1	DYN
1	0	CODE PRINCIPAL
1	1	CODE EXTERNE

POP peut les adresser soit en donnant explicitement le numéro, soit en le prenant dans le préfixe.

La génération des deux bits poids fort de l'adresse mémoire demande deux commandes C0 et C1 et le codage est le suivant:

C1	C0	Z1	Z0	A15	A14
0	0	X	X	0	0
0	1	X	X	0	1
1	0	0	0	0	0
		0	1	0	1
		1	X	1	MODE
1	1	X	X	1	MODE

où la bascule de MODE indique soit le mode principal (=0), soit le mode externe (=1).

Les deux fonctions sont réalisées par un double multiplexeur 4 vers 1 (74LS153) avec pour sélection A = C0+Z1 et B=C1.

Chargement du registre d'adresse

Le processeur POP manipule soit une adresse de mot égale à

(Z1, Z0,Y13...Y0)

soit une adresse d'octet égale à

(Z1, Z0, Y1VI....Y2).

Un registre à double entrée (74LS298) est donc utilisé comme adresse mémoire. Le signal WS sert de sélection. Une barrière 3 états (74LS244) commandée par le signal BMA (Bus Mémoire Alloué) permet de se connecter sur le bus adresse mémoire.

Numéro d'octet

Dans le cas où une adresse d'octet est envoyée, le numéro de l'octet doit être mémorisé. On distingue un numéro d'octet en lecture (NOCL) et un numéro d'octet en écriture (NOCE).

- Le premier (NOCL) est utilisé pour choisir une lecture en RML1 et RML0,
- le second (NOCE) pour choisir une écriture entre RME1 et RME0 et pour générer un masque d'écriture.

Les deux registres de deux bits NOCE et NOCL sont implantés dans un seul registre de 4 bits à deux entrées (74LS298) avec un montage croisé des entrées: quand on charge NOCL on reboucle NOCE et vice-versa.

génération du masque d'écriture

Selon les cas, on doit modifier en mémoire:

- un des quatre octets,
- un des deux demi-mots,
- un mot complet,
- un nombre variable d'octets adjacents.

Il faut générer les profils suivants:

	NOCE/NOCL		masque
1 mot	x	x	0 0 0 0
1/2 mot	0	0	1 1 0 0
	1	0	0 0 1 1
1 octet	0	0	1 1 1 0
	0	1	1 1 0 1
	1	0	1 0 1 1
	1	1	0 1 1 1

On doit aussi avoir la possibilité de calculer dynamiquement un masque sur plusieurs octets adjacents, pour générer par exemple:

INIT	1 1 1 1	
premier octet 0 1	1 1 0 1	
deuxème octet 1 0	1 0 0 1	
troisième octet 1 1	0 0 0 1	=> NOCE=3

Il existe donc deux modes de fonctionnement:

- le mode instantané (une seule écriture)
- le mode bouclé où l'on tient compte de l'état précédent du masque.

D'autre part quand une écriture dans la pile de POP est demandée, on doit controler s'il y a ou non debordement. Pour ce faire on dispose d'un mécanisme simple autour d'un comparateur 4 bits nécessitant une commande VCOMP.

En conclusion, la seule information "chargement de RADM" pour un acces mémoire de POP est insuffisante , il faut la paramétrer.

On distinguera les cas suivants:

- chargement de RADM pour un accés mot dans CTX

  (RADM(0,0);MOT)

- chargement de RADM pour un accés MOT dans DYN

  (RADM(0,1);MOT)

- chargement de RADM pour un acces MOT dans CTX avec contrôle de débordement de la pile

  (RADM(0,0);MOT;VCOMP)

- chargement de RADM pour un accés MOt dans CODE PRINCIPAL/EXTERNE suivant le mode

(RADM(1,M);MOT)

- chargement de RADM (et de POPE0) pour une ECRITURE d'OCTET dans CTX

(RADM(0,0);OCTET;NOCE;POPE0)

- chargement de RADM pour une ECRITURE D'OCTET dans DYN

(RADM(0,1);OCTET; NOCE)

- chargement de RAM pour une LECTURE D'OCTET dans CTX

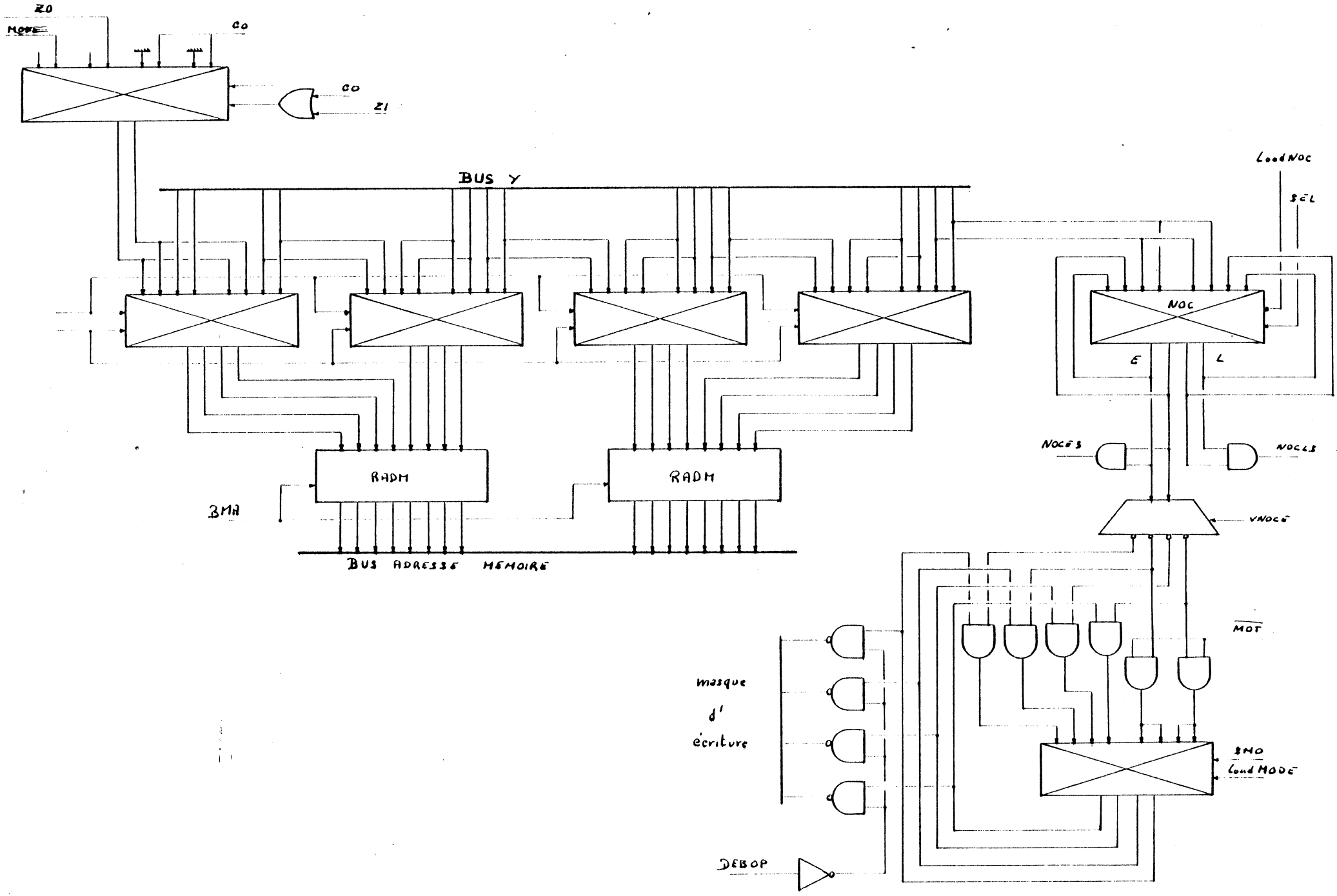
(RADM(0,0);OCTET;NOCL)

- chargement de RADM pour une LECTURE D'OCTET dans la zone spécifiée par PF2

(RADM(zone);OCTET;NOCL).

Outre les 14 bits d'adresse A0...A15 du bus Y il faut générer les signaux suivants pour le chargement de RADM:

- C0 et C1 permettent le calcul du numero de zone (bit A14 et A15)
- WS choisit entre mot (=0) ou octet
- SEL indique une lecture (=1) ou une écriture d'octet si WS=1
- VNOCE0 permet les croisements d'octets
- VCOMP pour controler les débordements de pile
- Load NOC pour charger le numero d'octet
- Load RADM pour charger l'adresse.



Chargement de RADM et génération du masque d'écriture

D'autre part on doit pouvoir orienter la sortie Y vers les autres destinations possibles, c'est-à-dire:

- PILE0
- RME0
- RME0 avec croisement des octets poids fort et poids faible
- RME0 et PILE0
- RMEj (j 0,1) selon NOCE1
- POPE0
- le séquenceur 2910
- le registre d'Etat
- le registre d'adresse de la ROM de génération de profil
- binaire ROM SET
- RME1
- PILE1
- POPE1

On a en tout 12+9=21 destinations possibles pour le bus Y. Une solution immédiate consisterait à utiliser un champ de microinstuctions de 5 bits pour coder toutes les combinaisons possibles. La plupart des commandes de chargement sont des fronts descendants. Il faut donc combiner les signaux générés par le décodage avec une horloge (CPA=CP.ATT par exemple). Ceci peut être fait sans beaucoup de matériel en rentrant CPA en adresse de la ROM pour générer les fronts nécessaires. Ceci implique des ROM de 64 mots de 19 bits.

Cette solution est trop coûteuse pour le décodage et la limitation à 47 bits de microinstruction due à l'utilisation de MADAM nous a conduit à essayer de l'optimiser.

On peut tout d'abord remarquer que RME1, PILE1 et POPE1 sont aussi chargeables par le bus poids fort BUS1, donc que les signaux de commandes peuvent être générés par le décodeur DESTBUS1.

D'autre part, il reste des possibilités de codage pour l'extension du champ ordre (utilisation du champ DA comme ordre quand ORDRE=0F).

Le recouvrement de champs de microinstructions (champ ayant plusieurs significations suivant le contexte) es restrictif quant à la puissance de la microinstruction, mais cette restriction n'est pas très génante dans le cas présent bien qu'il faille l'avoir à l'esprit pendant l'écriture du microprogramme.

Le choix qui a été fait consiste à coder dans le champ DESTY tous les chargements de RADM et de la partie poids faible des registres doubles, la partie poids fort est sous le contrôle de DB1, le registre d'état, d'adressage de ROMSET et le séquencement sont commandés par DA quand ordre = 0F.

Les signaux à générer par le décodage de DEST-Y sont donc:

- Load PILE0
- Load POPE0
- Load RME0
- VCOMp
- VNOCE0 ce sont des états et non des impulsions
- MHL
- Load NOC
- Load RADM
- C0
- C1
- WS
- SEL

Le codage choisi est donné par le tableau suivant:

	A3	A2	A1	A0	DESTINATION Y	WS	SEL	C1	C0	Load RADM	Load NOC	V COMP	Load RME 0	Load POPE 0	Load PILE 0	V NOCE 0
0	0	0	0	0	RADM(0 0);MOT	0	ϕ	0	0		1	0	1	1	1	0
1	0	0	0	1	RADM(0 1);MOT	0	ϕ	0	1		1	0	1	1	1	0
2	0	0	1	0	RADM(zone);MOT	0	ϕ	1	0		1	0	1	1	1	0
3	0	0	1	1	RADM(1,M);MOT	0	ϕ	1	1		1	0	1	1	1	0
4	0	1	0	0	RADM(0 0);MOT,VCOMP	0	ϕ	0	0		1	0	1	1	1	0
5	0	1	0	1	PILE 0	ϕ	ϕ	ϕ	ϕ	1	1	0	1	1		0
6	0	1	1	0	RME 0	ϕ	ϕ	ϕ	ϕ	1	1	0		1	1	0
7	0	1	1	1	RME 0 (H — L)	ϕ	ϕ	ϕ	ϕ	1	1	0		1	1	1
8	1	0	0	0	RADM(00);OCTET,NOCE,&POPE 0	1	0	0	0			0	1		1	0
9	1	0	0	1	RADM(01);OCTET;NOCE	1	0	0	1			0	1	1	1	0
10	1	0	1	0	RME 0 & POPE 0	ϕ	ϕ	ϕ	ϕ	1	1	0			1	0
11	1	0	1	1	RMEj croisé selon NOCE 0	ϕ	ϕ	ϕ	ϕ	1	1	0	1/	1	1	0
12	1	1	0	0	RADM(00;OCTET,NOCL	1	1	0	0			0	1	1	1	0
13	1	1	0	1	RADM(01);OCTET,NOCL	1	1	0	1			0	1	1	1	0
14	1	1	1	0	POPE 0	ϕ	ϕ	ϕ	ϕ	1	1	0	1		1	0
15	1	1	1	1	NOP	ϕ	ϕ			1	1	0	1	1	1	0

- Codage du champ Destination Y.

On remarquera que :

C0 = A0
C1 = A1
SEL = A2
WS = A3

Il reste 7 signaux à générer et le décodage de DEST-Y ne demande qu'une PROM de 32 mots de 8 bits dont le câblage est donné dans la figure VI.16.

On notera que la commande de croisement H<->L nécessite un boîtier 74LS00 et qu'il a été nécessaire de retarder les commandes C0, C1, SEL et WS par rapport aux adresses de la ROM.

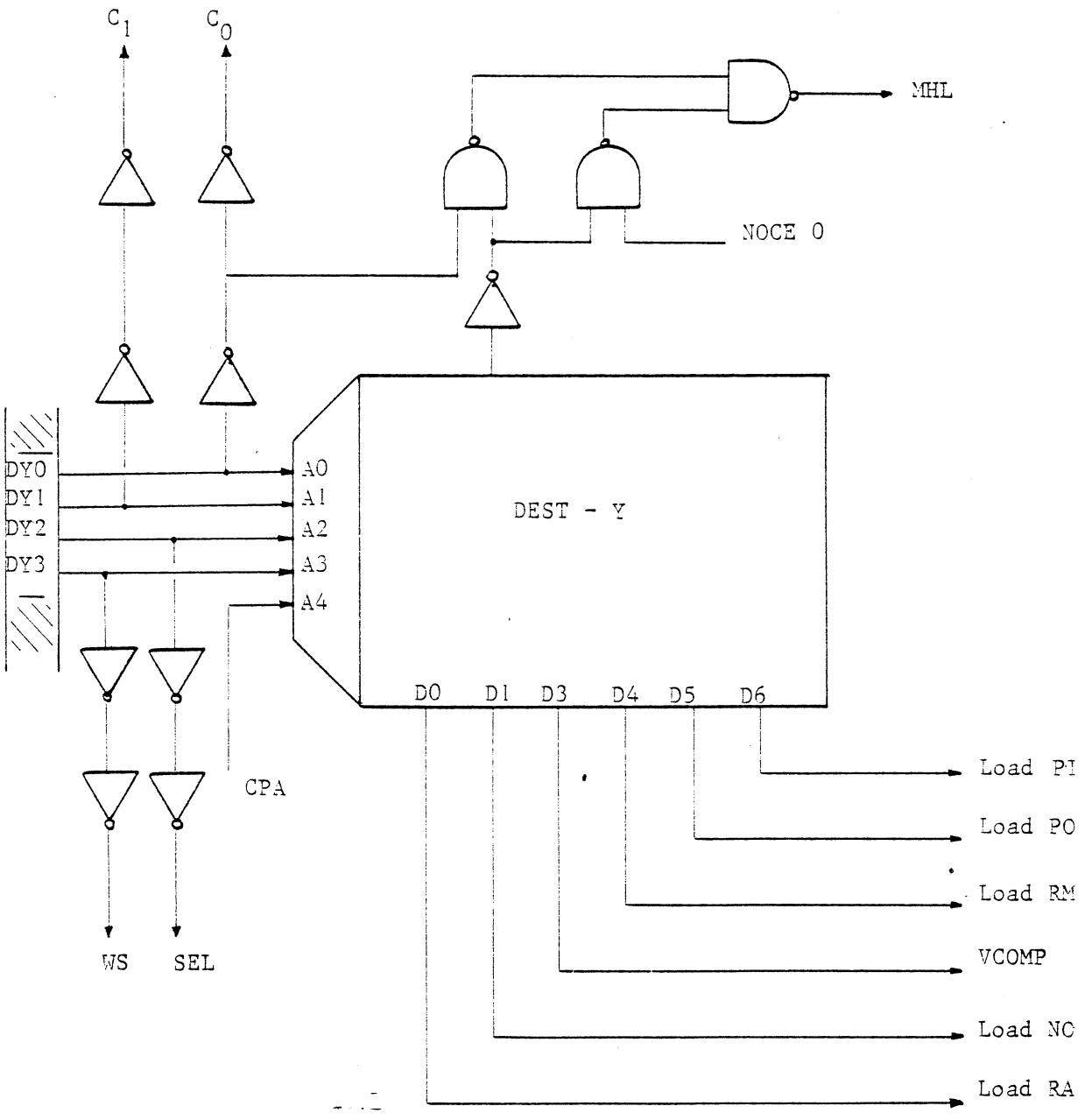

C1
C0
MHL
NOCE 0
DY0
DY1
DY2
DY3
A0
A1
A2
A3
A4
DEST - Y
D0
D1
D3
D4
D5
D6
CPA
WS
SEL
Load PI
Load PO
Load RM
VCOMP
Load NC
Load RA

Figure 6.16 -

VI.2.1.3. BUS0 et BUS1.

- Organisation du BUS0

Il n'a qu'une destination possible : l'entrée de l'opérateur à travers le multiplexeur commandé par le champs DA de la micro-instruction. Il peut recevoir comme sources les poids faibles des registres de communication avec la mémoire et avec le processeur FILE (RML_0 et $POPL_0$); les poids faibles de la pile interne $PILE_0$ et le bus poids fort BUS1.
Le choix de l'origine peut être conditionné par le bit d'indirection sur la valeur (IV1) du préfixe de l'opérande ou par le bit poids fort du numéro d'octet en lecture ($NOCL_1$). Par exemple, la valeur d'un descripteur venant de FILE, donc contenu dans POPL peut se trouver dans :

- $POPL_0$ si le bit IV du préfixe est égal à 0,
- RME_0 si IV = 1 et $NOCL_1$ = 0,
- RME_1 si IV1 = 1 et $NOCL_1$ = 1.

Dans le dernier cas il faudra choisir BUS1 comme origine de BUS0 et RME1 comme origine sur le BUS1.

-Organisation du BUS1

Il réalise le multiplexage entre RML_1, $POPL_1$, $PILE_1$, les registres contenant le préfixe du premier et du second opérande [(PF1,PT1) et (PF2 et PT2)], et les informations d'état, il peut être chargé soit dans (PF1,PT1), soit dans (PF2,PT2), soit dans un registre de travail sur les préfixes RPF (contenant PF1 ou bien PF2 et nécessaire au décodage de l'accès à la valeur d'un opérande).
Il peut enfin être multiplexé avec le bus Y pour charger $POPE_1$, RME_1 ou $PILE_1$ ou être connecté au BUS0.

Dans l'exemple précédent, on a vu que l'origine sur BUS1 pouvait être conditionnée par l'origine BUS0 suivant les valeurs de IV1 et de $NOCL_1$.
La destination peut aussi être différente suivant la valeur du bit poids fort du numéro d'octet en lecture ($NOCE_1$).
D'autre part pour le chargement de (PF2,PT2) on doit distinguer:

- le chargement des 8 bits à partir de BUS1,
- le chargement des 4 bits de poids faibles contenant le type de base,
- le chargement des 5 bits de poids faibles (type de base + PT),
- le chargement des 4 bits de poids forts (IV, PT, Z1, Z2).

La réalisation matérielle de ces fonctions requiert le montage de la figure 6.17.

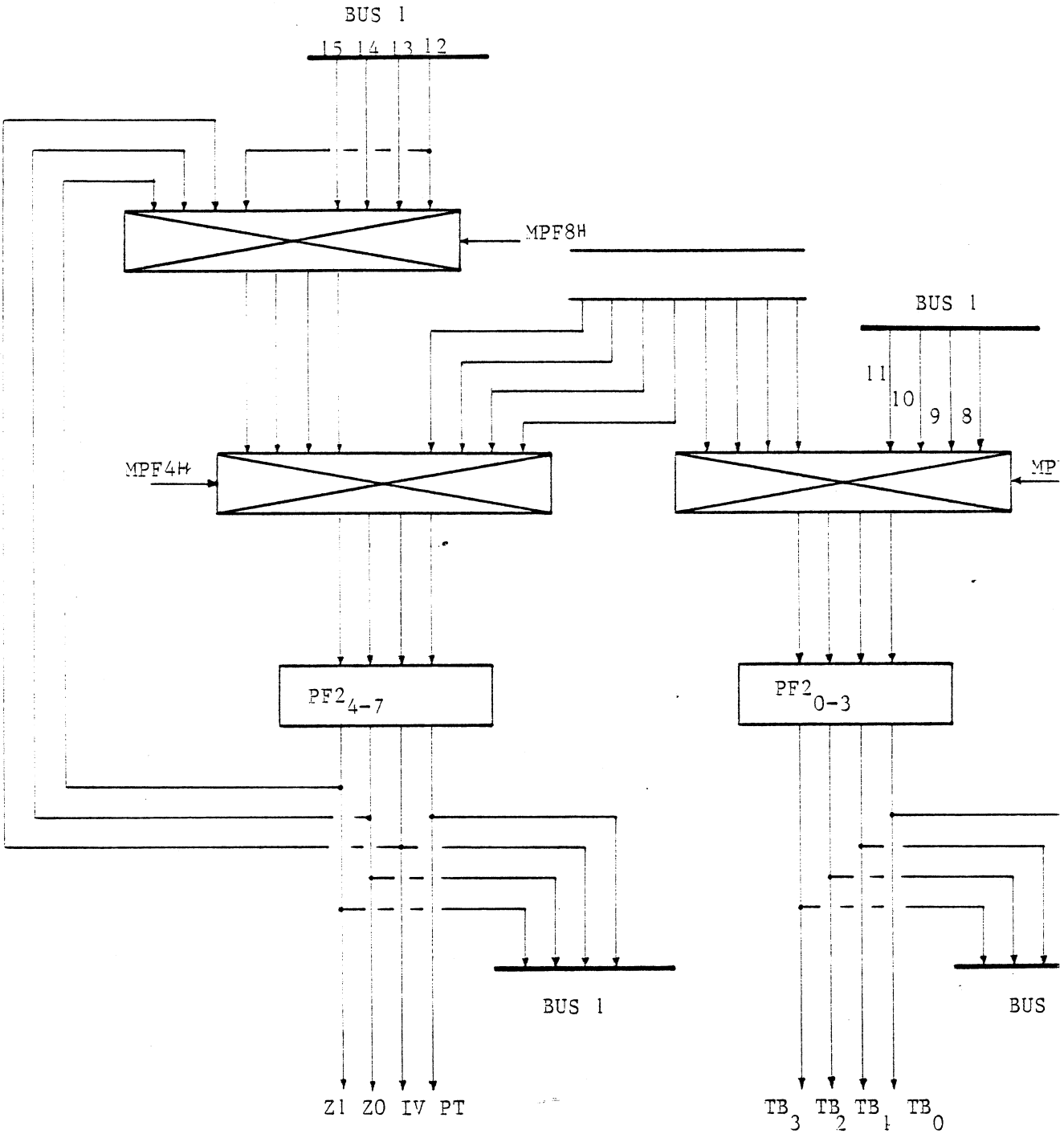


Figure 6.17 - Chargement de PF2

Le tableau suivant présente l'inventaire des opérations de transfert sur les BUS0 et BUS1 et met en évidence les besoins de parallélisme et l'influence des IV1, $NOCL_1$ et $NOCE_1$ pour le choix dynamique des sources et destinations.

	OPERATIONS	ORIGINE BUS 0	DESTINATION BUS 1	ORIGINE BUS 1	REMARQUE
0	POPL 0 →BUS 0	POPL 0	x	ϕ	
1	RML 0 → BUS 0	RML 0	x	ϕ	
2	PILE 0 → BUS 0	PILE 0	x	ϕ	
3	RML 1 → BUS 0	BUS 1	x	RML 1	
4	PILE 1 → BUS 0	BUS 1	x	PILE 1	
5	(PF1,PT1) → BUS 0	BUS 1	x	(PF1,PT1)	
6	(PF2,PT2) → BUS 0	BUS 1	x	(PF2,PT2)	
7	POPL 1 → BUS 0	BUS 1	x	POPL 1	
8	RML0 →BUS 0 ; RML 1→ PILE 1	RML 0	PILE 1	RML 1	"
9	RML0 →BUS 0 ; RML1 → PF2(5)	RML 0	PF2(5)	RML 1	"
A	(PF2,PT2) →BUS 0 ; VI →PF2	BUS 1	(PF2,PT2)	(PF2,PT2)	"(voir chargement de PF2)
B	POPL 0 →BUS 0 ; (PF2,PT2) →POPE 1	POPL 0	POPE 1	(PF2,PT2)	"
C	(PF1,PT1) → POPE 1	ϕ	POPE 1	(PF1,PT1)	
D	(PF2,PT2) → POPE 1	ϕ	POPE 1	(PF2,PT2)	
E	Y → RME 1	ϕ	RME 1	ϕ	Le multiplexeur entre Bus Y et bus 1 sélectionne le bus 1.
F	Y → PILE 1	ϕ	PILE 1	ϕ	
10	PILE 0 →BUS 0 ; PILE 1 →RME 1	PILE 0	RME 1	PILE 1	
11	PILE 0 →BUS 0 ; PILE 1 →POPE 1	PILE 0	POPE 1	PILE 1	
12	POPL 0 →BUS 0 ; POPL 1 →(PF1,PT1)	POPL 0	POPL 1	(PF1,PT1)	

uite du tableau

13	POPL 0 / RML i → BUS 0	POPL 0	x	ϕ	IV1 = 0	
		RML 0	x	ϕ	IV1 = 1	NOCC1 = 0
		BUS 1	x	RML 1		NOCC1 = 1
14	PILE 0 / RML 0 → BUS 0	PILE 0	x		IV1 = 0	
		RML 0	x		IV1 = 1	
15	PILE 1 / RML 1 → BUS 0	BUS 1	x	PILE 1	IV1 = 0	
				RML 1	IV1 = 1	
	RML i → RME j	RML 0	DEST Y = RME 0 et VDRCP RME 0	ϕ	NOCL 1 = 0 ; NOCE 1 = 0	
		RML 0	RME 1	ϕ	NOCL 1 = 0 ; NOCE 1 = 1	
		BUS 1	DEST Y = RME 0 et VDRCP RME 0	RME 1	NOCL 1 = 1 ; NOCE 1 = 0	
		BUS 1	RME 1	RME 1	NOCL 1 = 1 ; NOCE 1 = 1	
17	RML 0 / RML 1 → BUS 0	RML 0	x	ϕ	NOCL 1 = 0	
		BUS 1	x	RML 1	NOCL 1 = 1	
18	(PF1,PT1) → (PF2,PT2)	ϕ	(PF2,PT2)	(PF1,PT1)		
19	ETAT → RME 1	ϕ	RME 1	ETAT		
1A	VI 0...3 → PF2 0...3	ϕ	PF2 0...3	x	Voir chargement de PF2	
1B	VI 4...7 → PF2 4...7	ϕ	PF2 4...7	x		
1C	PF1 → RPF	ϕ	RPF	PF1		
1D	PF2 → RPF	ϕ	RPF	PF2		

Les signaux nécessaires pour commander les flux de données sur les BUS0 et BUS1 sont énumérés ci-après :

OE(PF1,PT1)
OE(PF2,PT2)
OE(POPL1)
OE(RML1)
OE(ETAT)
OE(PILE1)
Select(M1Y)
} Origine BUS1

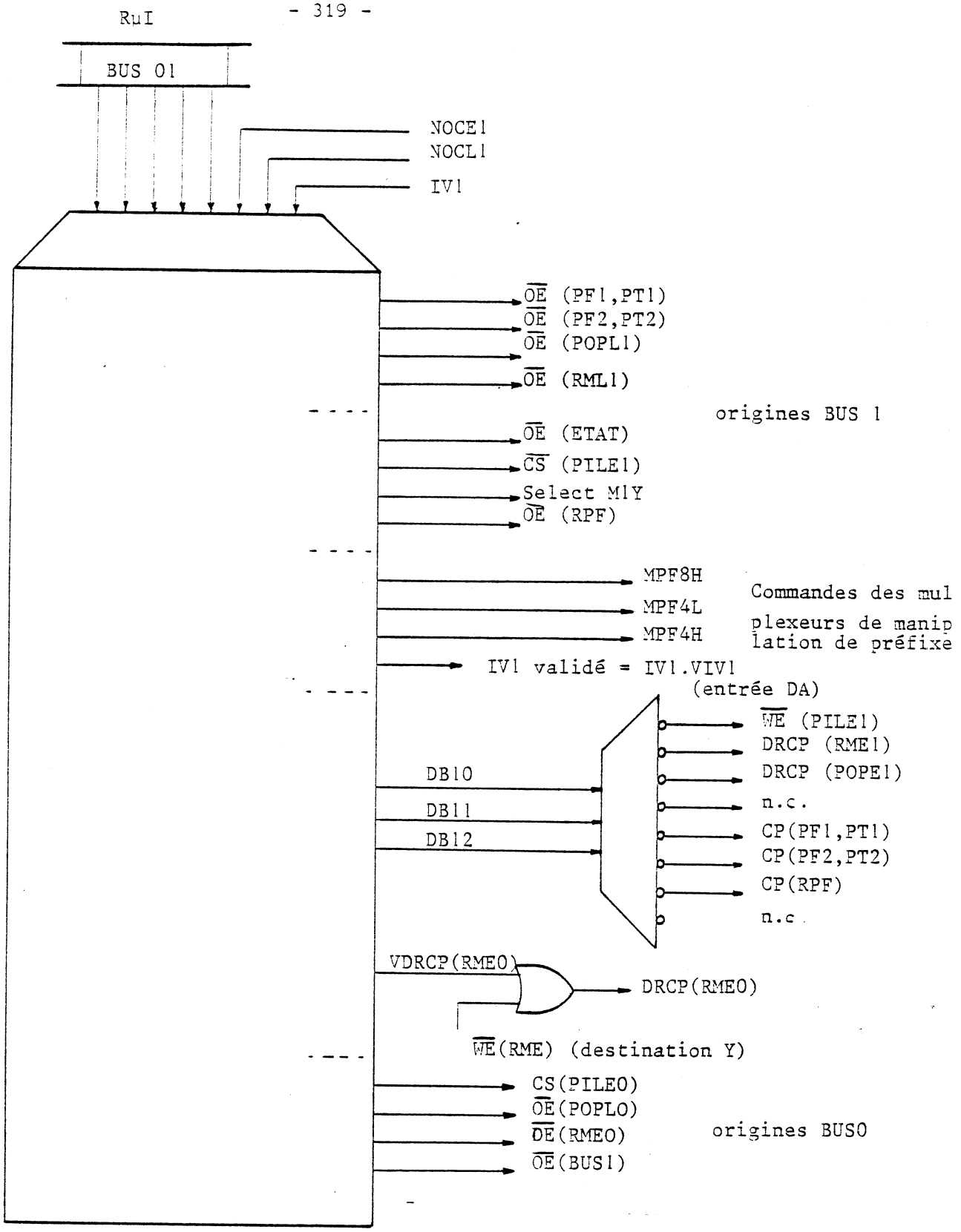
MPF8H
MPF4L
MPF4H
} Commande des multiplexeurs par chargement de PF

$RD(PILE_0)$
$OE(POPL_0)$
$OE(RML_0)$
OE(BUS1)
} Origine BUS0

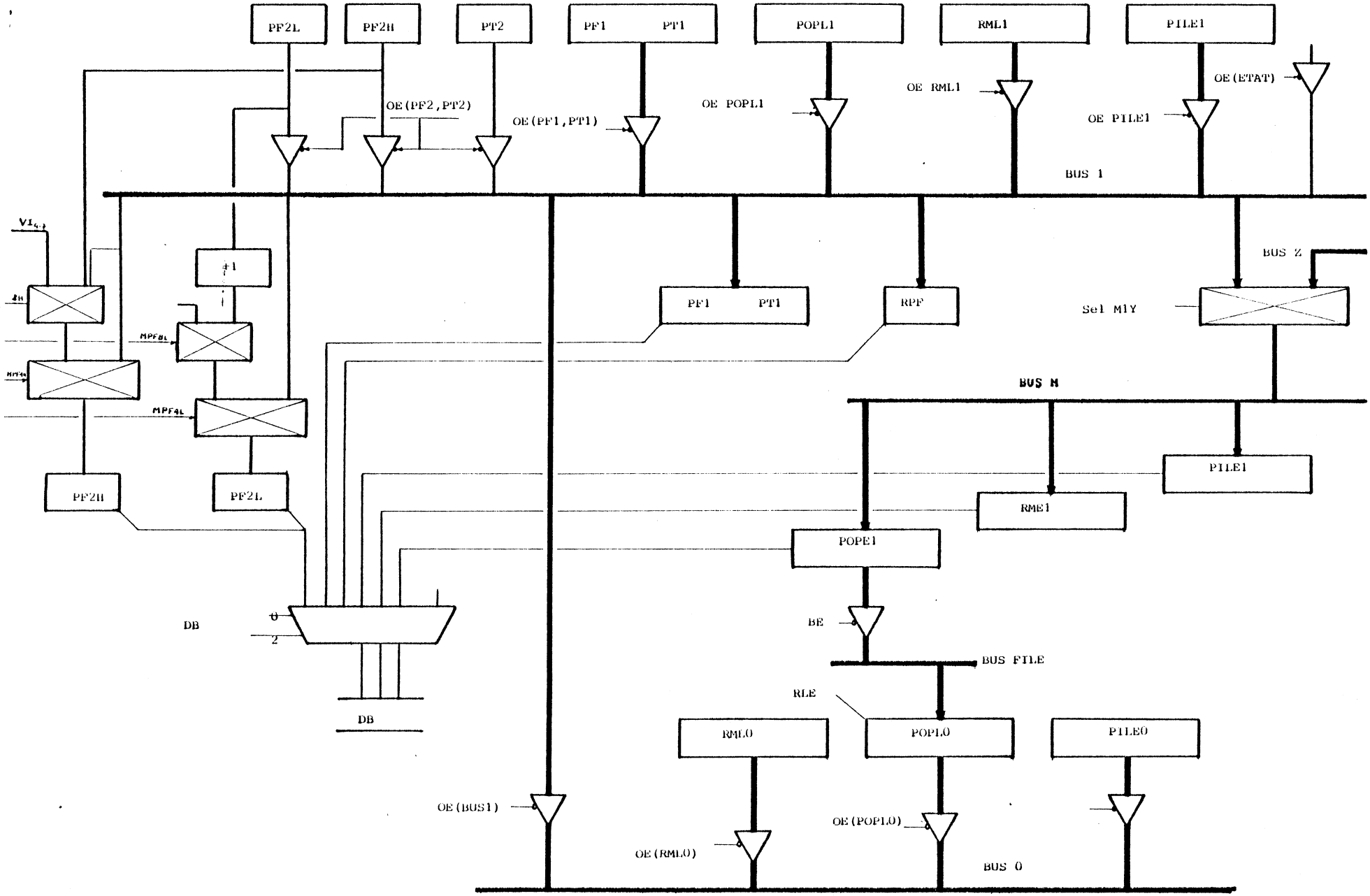
WE(PILE1)
DCRP(RME1)
DCRP(POPE1)
CP(PF1,PT1)
CP(PF2,PT2)
CP(RPF)
} Destination BUS1

$VDCRP(RME_0)$ permettant la prise en compte de WE RME_0 venant du décodage de la destination Y.

Les trente possibilités de transfert peuvent être codé par 5 bits adressant une mémoire de 32 mots de 24 bits (3 boîtiers 74S288) pour générer les commandes.
Une telle solution impose d'ajouter un certain nombre de portes ou de circuits MSI pour prendre en compte IV1, NOCE1, NOCL1 d'une part et pour valider les destinations de bus par une horloge car les chargements de registres se font sur des fronts et non sur des états, de plus ils ne doivent pas avoir lieu si le séquencement est en attente d'un événement extérieur.
La solution qui a été retenue consiste à mettre un décodeur 3 vers 8, validé par l'horloge CP si la condition d'attente est levée pour la destination du BUS1. Il reçoit en entrée les signaux DBl_0, DBl_1 et DBl_2 issus d'une matrice de décodage constiituée par 5 boîtiers ROM 256*4 adressés par 5 bits de la micro-instruction (champ BUSO1) + les informations IV1, NOCL1 et NOCE1 si bien que le choix dynamique des opérandes se fait sans matériel annexe (figure 5.18).



FigureVI.18 - Commandes des bus 0 et 1.


PF2L
PF2H
PT2
PF1 PT1
POPL1
RML1
PILE1
OE (ETAT)
OE (PF2, PT2)
OE (PF1, PT1)
OE POPL1
OE RML1
OE PILE1
BUS 1
BUS Z
+1
PF1 PT1
RPF
Sel M1Y
MPF8L
MPF4L
BUS M
PILE1
RME1
PF2H
PF2L
POPE1
DB
BE
BUS FILE
RLE
RML0
POPL0
PILE0
OE (BUS1)
OE (POPL0)
OE (RML0)
BUS 0
DB

VI.2.2. Commandes de l'opérateur.

Vingt bits sont nécessaires pour :

- choisir une opération de l'Unité Arithmétique et Logique (entrées I_1, I_2, I_3, I_4 des tranches 2903),

- choisir la destination de la sortie de l'ALU ainsi que les sens des décalages (entrées I_5, I_6, I_7, I_8 des tranches 2903),

- contrôler les origines des opérandes (I_0, EA 2903; OE_B 2903; OE BUS0; OE_{ROMSET}),

- contrôler les entrées décalages SIO_0, SIO_{15}, QI_0, QI_{15} (I_6, I_7, I_8, I_9 du 2904),

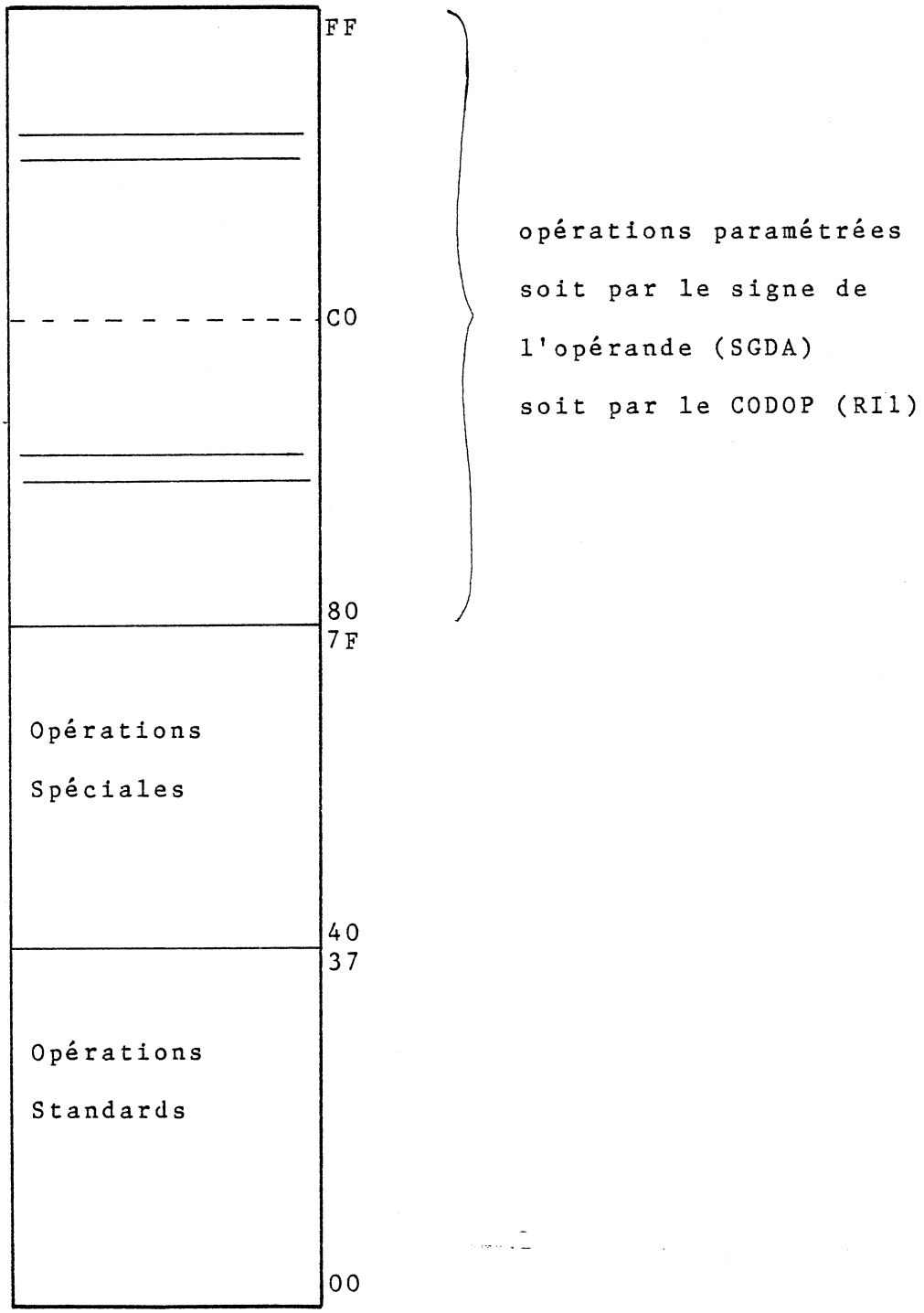
- contrôler la génération de CARRY-IN (I_{10}, I_{11}, I_{12} du 2904).

Ils sont issus d'une ROM de commande adressée par 8 bits de micro-instruction. Cette matrice de décodage est divisée en quatre zônes de 64 mots de 20 bits :

- la première (d'adresses 00 à 3F) contient les commandes des opérations standards (au moins une des entrées I_0, I_1, I_2, I_3, I_4 est à l'état haut),

- la seconde (40 à 7F) contient les commandes pour les opérations spéciales de calcul arithmétiques ($I_0=I_1=I_2=I_3=I_4=0$),

- les deux autres permettent de coder des commandes paramétrables (SGDA), soit par un bit du code-opération (RI_1). Par exemple une soustraction ou une addition dérouleront le même micro-programme, seules les commandes ALU sont différentes, et le bit RI_1 distingue ADD et SUB dans la même classe d'opérateurs binaires. Cette paramétrisation permet d'améliorer les performances du décodage des instructions.


FF
C0
80
7F
Opérations
Spéciales
40
37
Opérations
Standards
00
opérations paramétrées
soit par le signe de
l'opérande (SGDA)
soit par le CODOP (RI1)

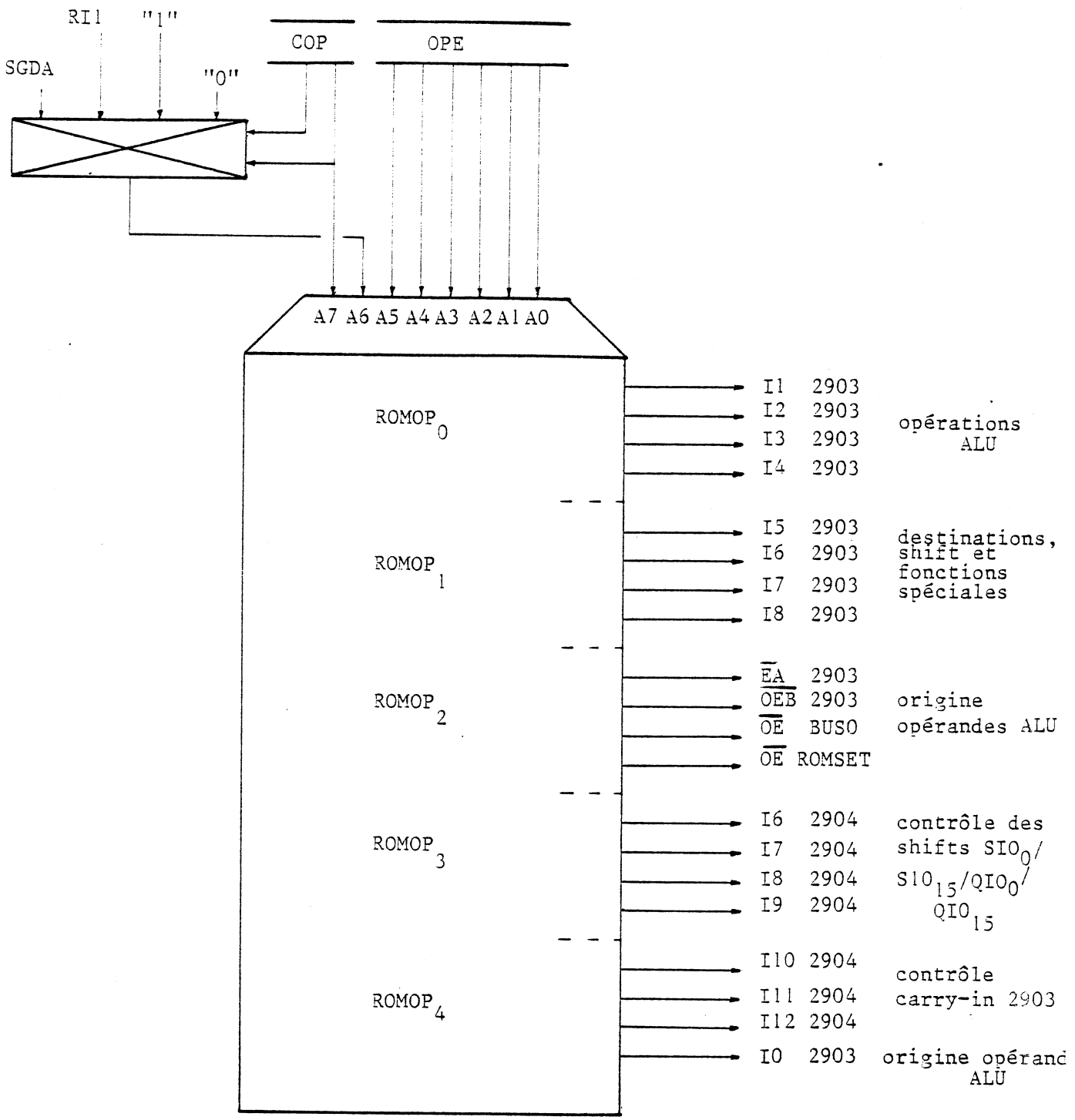


Figure 6.20. - Génération des commandes de l'opérateur

VI.2.3. Gestion de la pile de POP.

POP utilise une pile pour stocker les résultats intermédiaire longs et les descripteurs de boucles FOR.

L'implantation de cette pile doit répondre à des contraintes contradictoires :
- minimisation du nombre d'accès à la mémoire centrale,
- limitation de la taille d'une mémoire locale,
- réduction du temps de sauvegarde générale (multiprogrammation).

Un compromis doit être recherché. Notre choix consiste à disposer sur le processeur d'une mémoire de 16 mots se prolongeant en mémoire centrale dans une zône fixe comprise entre les adresses 0000 et 01FF et divisée en 16 éléments de 32 mots, et un mécanisme de gestion interne de cette pile.
Le dernier élément entré dans la pile est pointé par un compteur de 4 bits, K. L'adresse du sommet de la pile en mémoire centrale est mémorisée dans un registre SP interne à l'opérateur.
Lorsque la pile interne est pleine, l'élément le plus ancien est recopié en mémoire centrale (opération PUSH) et inversement lorsqu'elle est vide, le dernier élément empilé est transféré dans la pile interne (opération POP).
A chaque opération PUSH (ou POP) la valeur de SP doit être comparée à celle de la borne supérieure BORNSUP (ou de la borne inférieure BORNINF) afin de détecter des débordements éventuels.
L'état de la pile de POP est consigné dans deux bascules PLEIN et VIDE. Un compteur d'occupation de 4bits N indique le nombre d'éléments de la mémoire locale, il est automatiquement incrémenté à chaque écriture et décrémenté à chaque lecture. Les sorties "carry" et "borrow" permettent de commander PLEIN et VIDE. En effet, on remarque que :

VIDE 0 si N passe de 15 à 0 et PLEIN = 0,
VIDE 1 si N passe de 0 à 15 et PLEIN = 0,
PLEIN 0 si N passe de 0 à 15 et VIDE = 0,
PLEIN 1 si N passe de 15 à 0 et VIDE = 0.

- Contrôle du débordement de la pile de POP.

Les limites de la zône allouée à la pile de POP sont envoyées par PME dans la phase d'initialisation des processeurs et rangées dans RETAT.
A chaque occurence d'une opération d'empilage/dépilage la valeur de SP doit être comparée soit à la borne supérieure, soit à la borne inférieure. La pile de POP occupe un certain nombre ($\leq$16) d'incréments de 32 mots mémoire. Chaque borne peut être codée par son numéro sur 4 bits qui devra être comparé aux bits 5 à 8 de SP incrémentée de 1 ou décrémentée de 1. Cette détection de débordement se fait automatiquement si elle est validée par le signal VCOMP issu de la ROM de décodage de la destination du bus Y de la façon suivante :

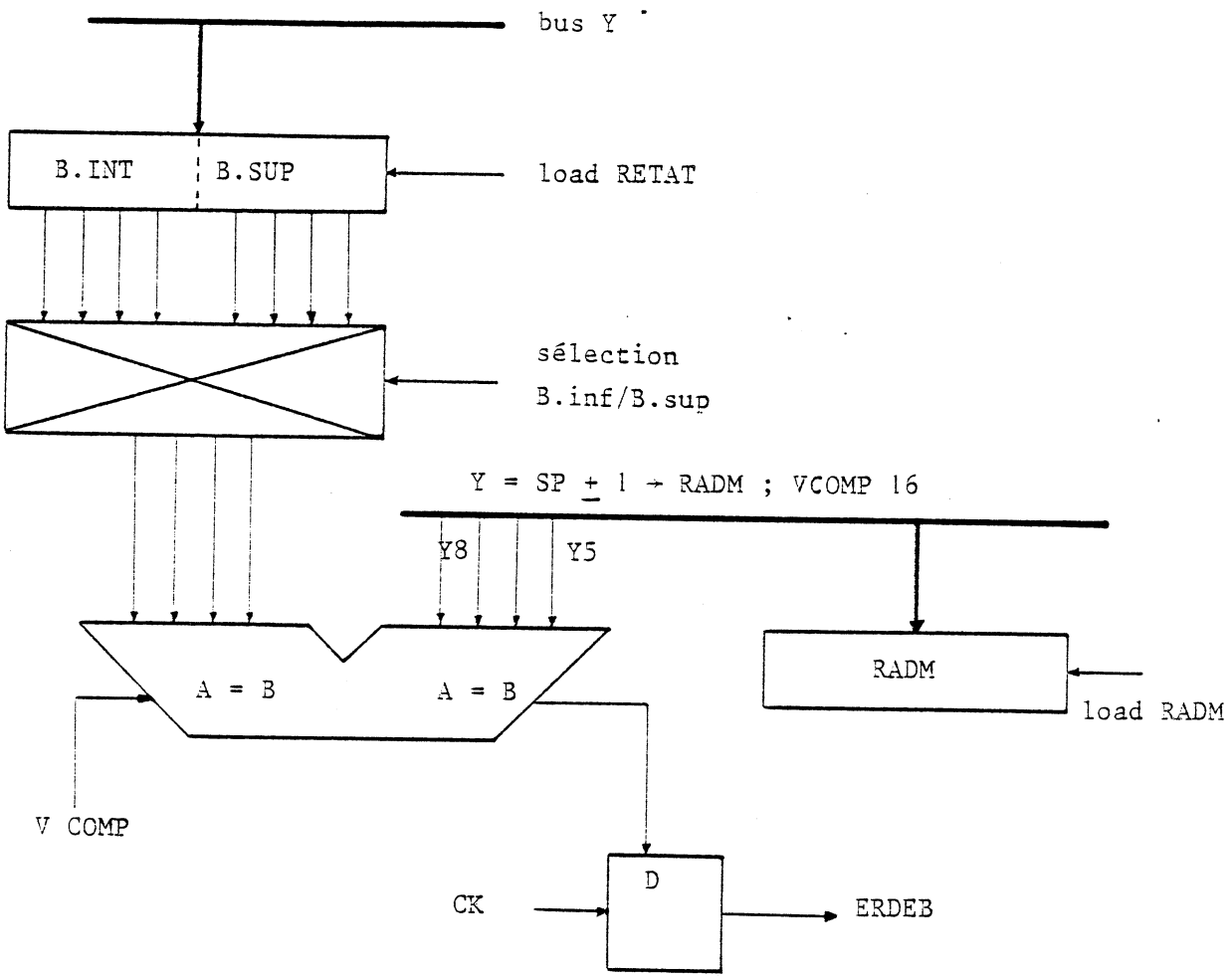


Figure 6.21 - Contrôle de débordement de la pile

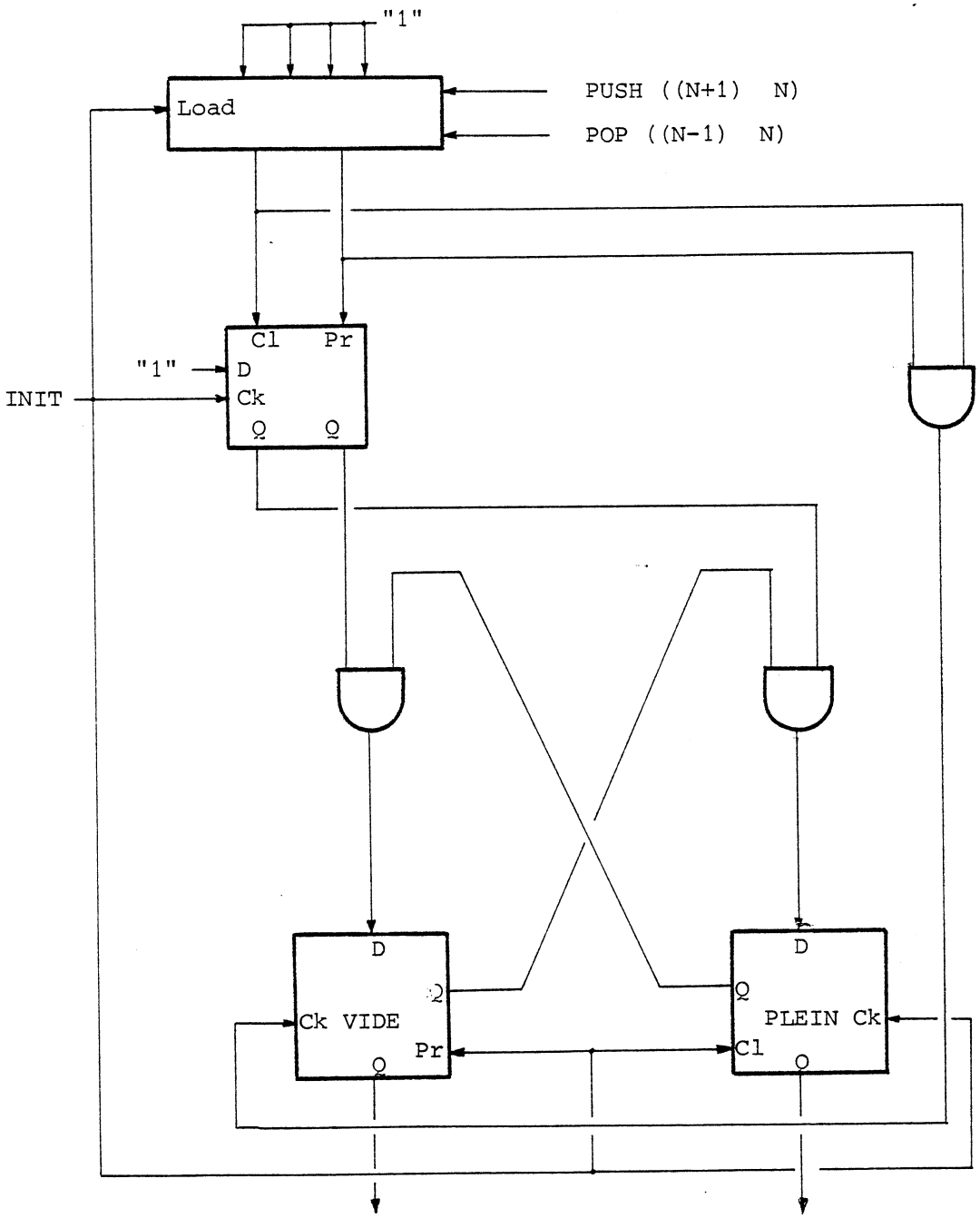


Figure VI.21. - Bascules d'état de la pile interne

VI.2.4. Ordres.

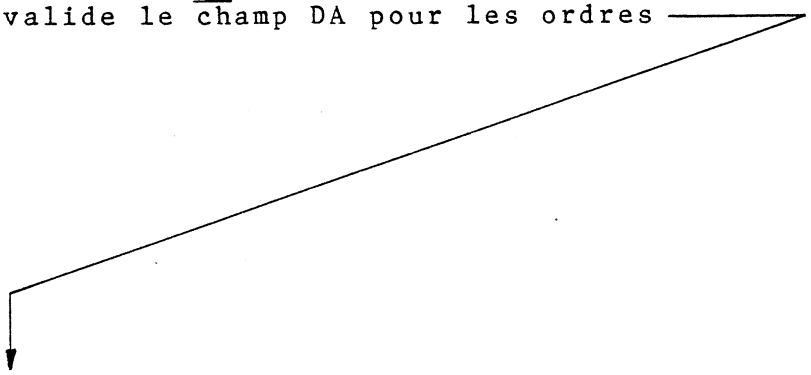
Nous ne disposons plus que de quatre bits de micro-instruction pour coder tous les ordres de POP, or il y en a 22 différents. Cette limitation est imposée par le système MADAM qui ne peut fournir que 47 bits de données. Nous avons donc été amené à utiliser un autre champ (en l'occurence DA) pour coder certains ordres. Cette superposition de champ impose des restrictions au niveau de l'écriture du microprogramme, certains ordres devenant incompatibles avec l'utilisation de l'entrée de l'opérateur.
Nous avons cherché à les classer en deux groupes : ceux qu'il est nécessaire d'exécuter en parallèle avec l'utilisation du chemin de données (comme les accès mémoire ou la gestion de la pile) et les autres.
Cette classification apparait dans le tableau suivant donnant la liste et le codage de tous les ordres.

Si ORDRE = 0F alors DA code un ordre (ou une destination Y)

Les chargements du registre d'état, du registre d'adresse de ROMSET, ou du compteur de boucles du séquenceur sont des "destinations pour le bus Y". L'activation de ces commandes ne peut se faire que si ORDRE = 0F et DA n'est pas utilisé pour sélectionner une entrée de l'opérateur.
Les ordres sont décodés par une matrice de décodage générant les impulsions de chargement ou les états de positionnement des bascules ainsi que les signaux permettant la génération du masque d'écriture (VNOCE, 8M0, M0T). Le champ DA est décodé par un décodeur 4 vers 16 validé par une des sorties de la ROM.

champ ORDRE

0	NOP	
1	LMC	
2	EMC-Byte	
3	EMC-Halfword	Demande d'accès mémoire
4	EMC-Word	
5	RAZ-MSK	Remise à zéro du masque d'écriture
6	DPOP <-- 1	
7	NEXT-BIPOP	
8	NEXT-OP	
9	K+1 --> K	
A	N+1 --> N	
B	K+1 --> K et N+1 --> N	Gestion du compteur
C	K-1 --> K	d'adresse et du compteur
D	N-1 --> N	d'occupation de la pile
E	K-1 --> K et N-1 --> N	de POP
F	valide le champ DA pour les ordres	



champ DA

0	MODE <-- 0	
1	MODE <-- 1	Bascule de mode
2	TROMPE	
3	BON	Levée de l'état conditionnel de PINS
4	ERR-POP	
5	INIT-PILE	
6	A9 <-- A9	Zône MADAM
7		
8		
9	Load ETAT	registre d'état
A	Load ADS	Destination Y registre d'adresse ROM-SET
B	FINSAUVE	
C	BLD2910	Destination Y = compteur du séquenceur
D		
E		
F		

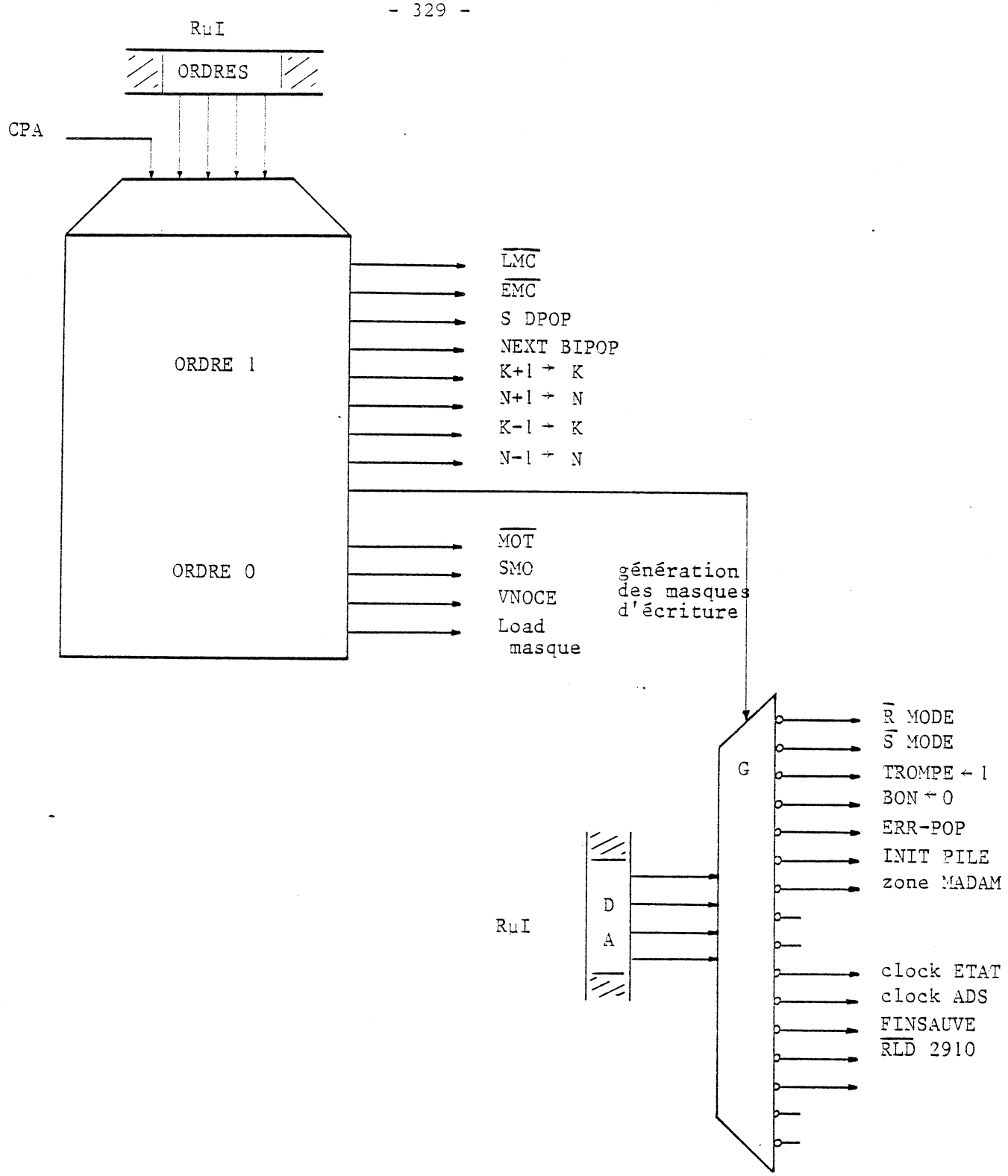


Figure 5.22. - Décodage des ordres

Chapître VII

PRESENTATION TECHNIQUE DU PROCESSEUR PME

Les trois processeurs PAC, POP et PINS sont des clients potentiels pour la mémoire centrale contenant les données nécessaires à l'exécution du programme en exécution et chargées par la machine système.

Le processeur mémoire PME a été introduit pour régler les conflits possibles d'accés et faire les translations d'adresses virtuelles (émises par le demandeur) =. adresse réelle en mémoire centrale (figure VI.1).

VII.1. Interface entre les demandeurs de PME

Tous les processeurs sont munis du même dispositif de dialogue avec PME donné par la figure VII.2) générant une demande (DEMi) à partir d'ordres de lecture ou d'écriture mémoire (LMC ou EMC) prise en compte si aucune autre demande plus prioritaire n'a été formulée.

Ils attendent une réponse de la mémoire sur deux signaux:

- AMCi indique que RADM est libre (la demande précédente a été prise en compte),

- VMCi indique que la donnée lue en mémoire est disponible dans RML1 et RML0.

L'envoi par le processeur numéro i d'un ordre LMC ou EMC a pour effet de générer une demande DEMi vers PME et de mettre soit à "0", soit à "1" une bascule R/W qui indique le type de l'opération mémoire à effectuer.
Lorsque PME décide, en fonction de l'ordre ds priorités, d'allouer la mémoire au processeur numéro i, il envoie le signal BMAi qui a pour effet:

- de mettre RADM sur le bus d'adresse relative pour effectuer le calcul et le contrôle de l'adresse réelle,

- de mettre à "1" une bascule qui validera la prise en compte du signal DEBOP envoyé par PME et reconnu par l'automate qui, selon qu'il s'agit d'une lecture ou d'une écriture, validera le chargement de RML1, RML0 (RLE), ou mettra RME1, RME0 sur le bus mémoire (BE).

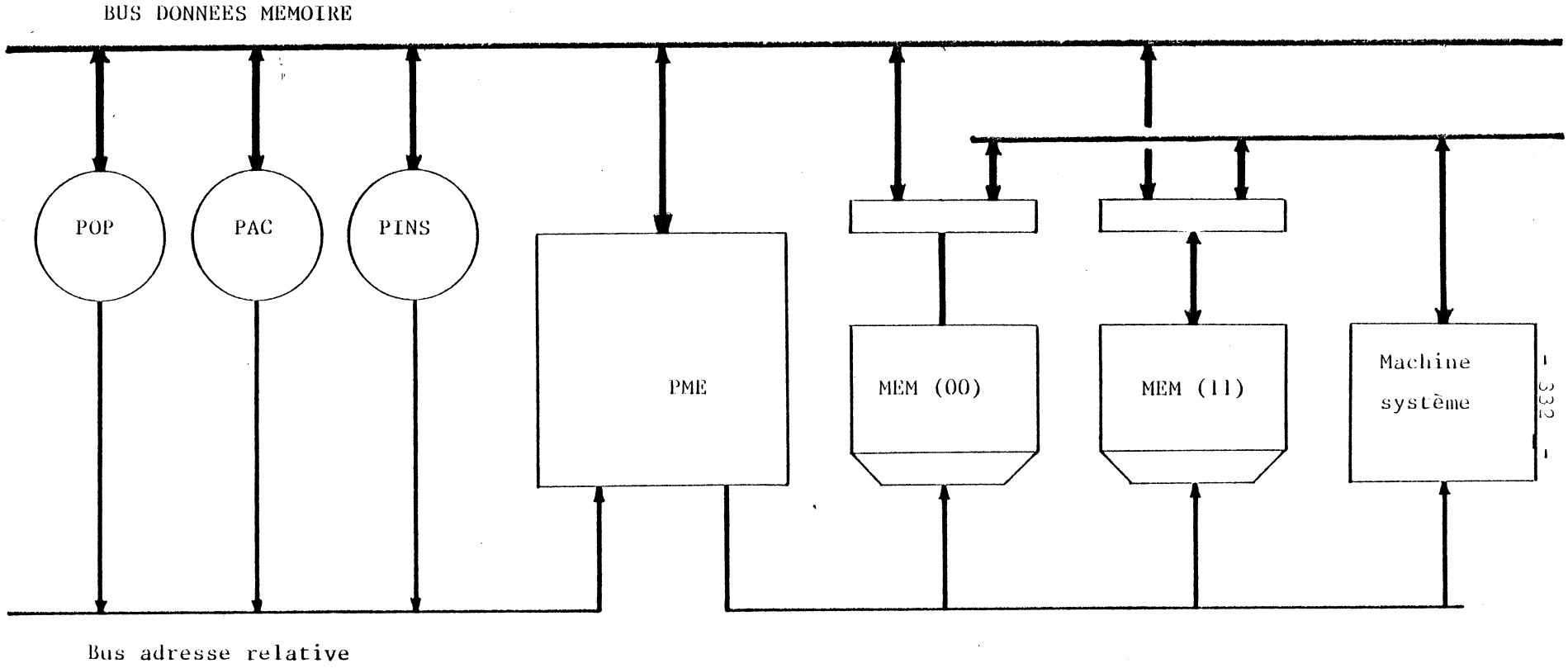


Figure VII.1 - Insertion de PME entre PASC-HLL et la machine système.

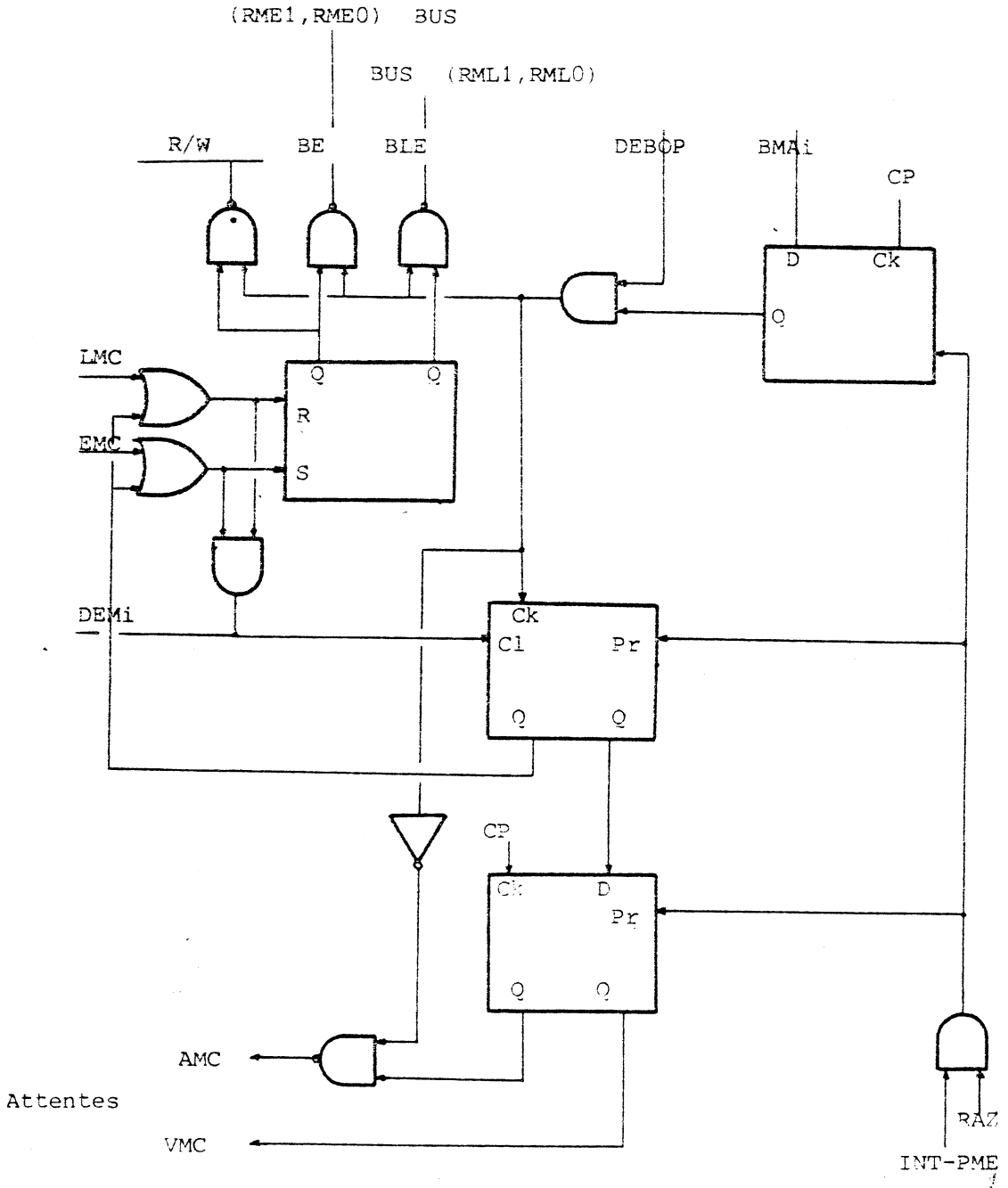
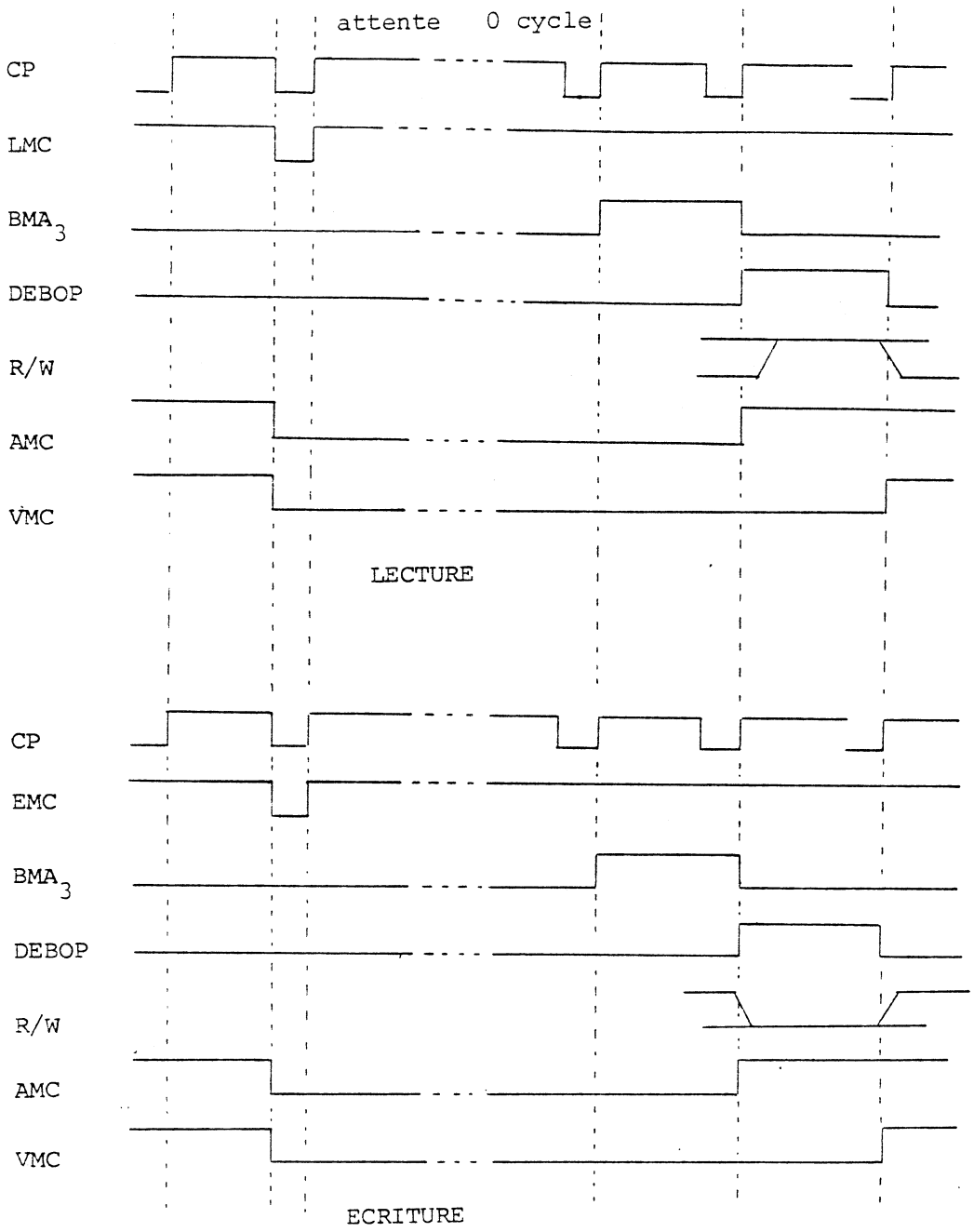


Figure VII.2. - Interface avec les autres processeurs



Le signal AMC remonte en fin de cycle de BMA

Le signal VMC remonte en fin de cycle DEBOP

L' automate peut être initialisé soit par RAZG à la mise sous tension, soit par INT-PME si une erreur a été détectée .

Une simulation de PASCHLL a permis de définir l'ordre de priorité suivant:

0 - POP
1 - PAC
2 - PINS)
) *
3 - PINSI)

* PINS se présente comme deux demandeurs: il peut en effet, soit formuler des requêtes par les ordres microprogrammés EMC ou LMC, soit automatiquement par son automate d'accés aux instructions PINSI.

VII.2. Fonction calcul d'adresse

Le bus adresse virtuelle entre les processeurs demandeurs et PME est de 16 bits:

A15 A14 donnent le numéro de la zone mémoire à accéder

et A13..A0 un déplacement positif par rapport à la base de cette zone.

Pour les calculs de translation d'adresse (MODE CALCUL):

Adresse réelle = BASE(A15 A14) + (00A13..A0)

et la vérification de non dépassement de zone:

[LIMITE(A15 A14) - (00A13..A0)] > 0

PME a besoin d'un opérateur 16 bits constitué de 4 tranches Am2901. Ces registres internes contiennent les BASE et LIMITE de chacune de ces zones.

VII.2.1. Initialisation

Au moment du GO, PME est en mode INIT et il monopolise la mémoire centrale pour aller lire aux adresses 0 à 3 les couples BASE, LIMITE des quatre zones afin de les ranger dans huit reistres internes de son opérateur de la façon suivante:

REG(0) = BASE(CTX
REG(1) = BASE(DYN)
REG(2) = BASE(CODE)
REG(3) = BASE(EXT)
REG(4) = LIMITE(CTX)
REG(5) = LIMITE(DYN)
REG(6) = LIMITE(CODE)
REG(7) = LIMITE(EXT)

Le numéro de la zone mémoire est câblé sur les entrées adresses B1 et B0.
Le bit B2 fait la distinction entre BASE (B2=0) et LIMITE (B2=1).
Le bit B3 est câblé à la masse.

L'algorithme d'initialisation est donné par la figure VII.4.

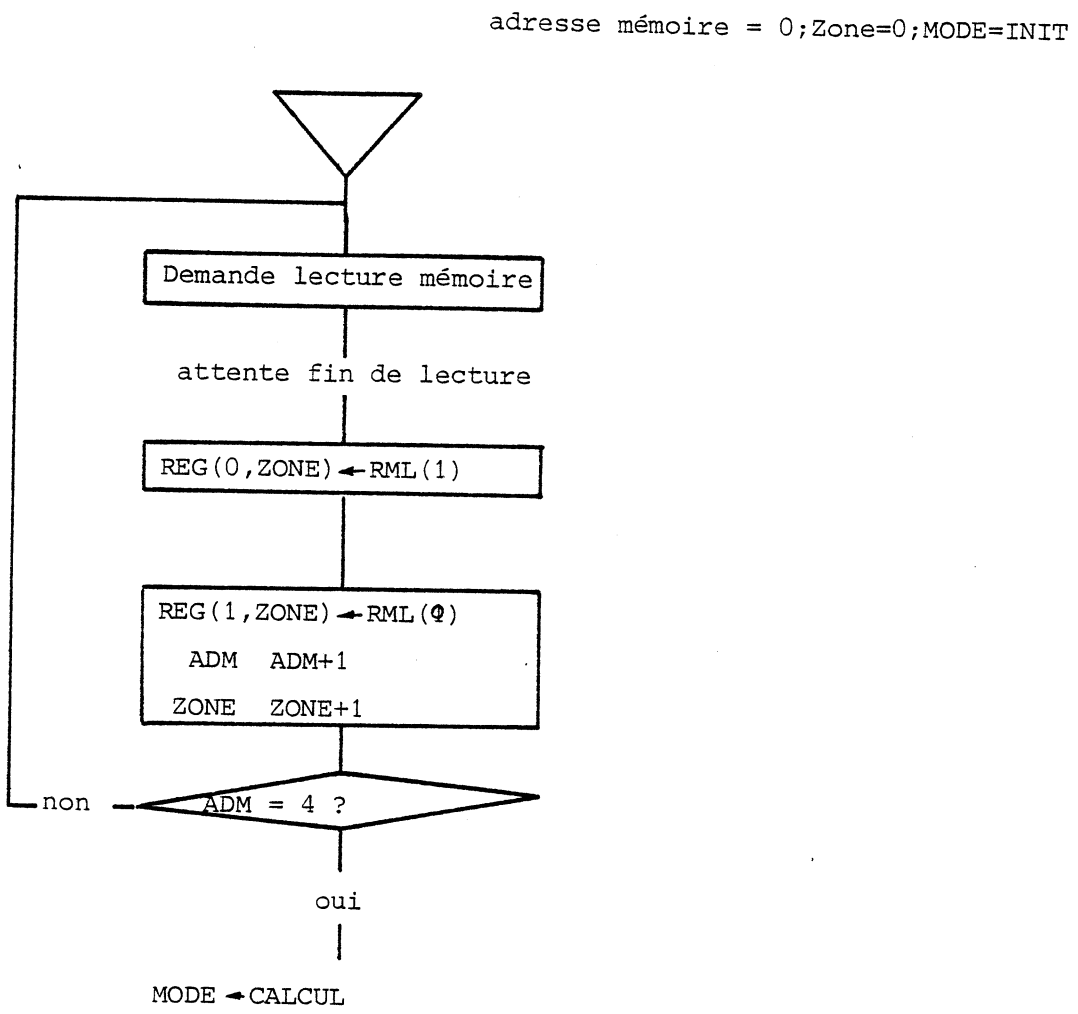


Figure VII.4. Algorithme d'initialisation de l'opérateur de PME

VII.2.2. Calcul

En fin d'initialisation, PME passe en mode CALCUL. Son opérateur bascule sur deux instructions:

- RAM(A)-D avec A2=1 pour le test de débordement de zone

- RAM(A)+D pour la génération de l'adresse réelle.

Les deux bits poids faible de l'adresse du registre A sont donnés par A14 et A15 tandis que l'entrée D reçoit A0..A13 étendus avec deux zéros (entrées D14 et D15 invalidées).
Qu'il y ait ou non une demande d'opérations mémoire ces deux instructions sont executées.
Les résultats ne sont pris en compte, par l'activation du mécanisme d'ERREUR pour la première et le chargement du registre adresse mémoire pour le second, que si un BMAi a eu lieu (le signal AttDA est alors à "1").

VII.2.3. Commande de l'opérateur

En résumé 16 cycles sont nécessaires à l'initialisation de PME (4 boucles de 4 cycles) et 2 cycles rebouclés pour les calculs. Seulement 4 codes différents sont à générer pour l'opérateur:

mode INIT MODE=0	NOP (par exemple PASSD)
	D → RAM(B) (chargement des bases et limites)
mode CALCUL MODE=1	RAM(A) - D
	RAM(A) + D

Si l'on regarde les profils binaires à générer sur I0..I8 et Cin des tranches Am2901 pour qu'elles exécutent ces instructions, on constate que certains bits sont constants et que d'autres sont égaux deux à deux:

I0 = I2 =I6 = "1"
I5 = I3 = "0"
I3 = Cin
I1 = I4 = MODE = validation des entrées D14 et D15

Donc, outre le signal MODE il faut générer seulement deux commandes:

I3/CIN et I7

pour différencier les 4 instructions.

Plutôt que de parcourir quatre fois les quatre instructions de la boucle d'accés aux bases et limites (en mode INIT) nous avons choisi d'éxécuter 16 instructions dont le numéro est fourni par un compteur modulo 16:
Il est initialisé à zéro au moment de la mise sous tension ou après une sauvegarde. Il est incrémenté à chaque top d'horloge. Après le 16ème top, une retenue (carry) est générée. Celle-ci va être utilisée pour commander le passage de PME en mode CALCUL (positionnement à "1" de la bascule MODE).

Sur le tableau de la figure VII.5, on remarque qu'en mode INIT, les signaux de commande de l'opérateur et d'adressage de ses registres internes sont étroitement liés au numéro de l'instruction et on peut écrire:

I3 = Cn = "1"
I7 = Y1
B0 = Y2
B1 = Y3
B2 = Y0

Pour ce qui est du mode CALCUL, l'adresse Y0 change à chaque cycle et permet de différencier les deux instructions en agissant sur les commandes Y3 et Cin de l'opérateur. On peut écrire:

I3 = Cn = Y0
I7 = Q
A0 = A14
A1 = A15
A2 = Y0

Donc, dans tous les cas :

I3 = Cn = MODE.Y0
I7 = MODE.Y1

D'autre part, en ce qui concerne les entrées Di de l'opérateur, on constate qu'en mode INIT:

- l'ouverture du registre RML0 pour le chargement des BASES doit se faire aux cycles 2, 6, A et E, c'est-à-dire:

chargement BASE = OE(RML0) = Y1.Y0

- l'ouverture de RML1 pour charger les LIMITES doit avoir lieu aux cycles 5, 7, B et F, c'est-à-dire:

chargement LIMITE = OE(RML1) = Y1.Y0

- la même condition doit entraîner l'incrémentation de l'adresse mémoire

(+1) = Y1.Y0

En mode CALCUL, les sorties RML0 et RML1 sont maintenues dans l'état "haute impédence" et les entrées D14 et D15 sont forcées à zéro pour pouvoir lire les adresses virtuelles si un BMAi a lieu.

Pour l'adressage de la mémoire centrale, nous avons opté pour un registre compteur chargeable par les sorties de l'opérateur en mode CALCUL, initialisé à zéro par RAZG à la mise sous tension ou après une sauvegarde générale et incrémenté par le signal (+1).

Les demandes lecture mémoire de PME (DEM LEC) ont lieu aux cycles numéro 0, 4, 8 et C, c'est-à-dire:

DEMLEC = Y0 + Y1

La figure VII.6 montre le câblage de l'automate opératif de PME. Tout cet automate opératif est synchronisé par une horloge CP1 ayant une fréquence double de l'horloge générale de PASCHLL. Ainsi l'initialisation de PME dure 4 cycles de base et les vérifications et calcul d'adresse , un seul cycle. Pour ce faire, PME est câblé en technologie TTLS et l'opérateur est réalisé avec la version la plus rapide des tranches AM2901 A (105ns de temps de cycle minimum).

Y3	Y2	Y1	Y0	instruction	B2	B1	B0	D15...D0	I7	I4/ I1	I3/ C0	DEM LEC	+1
0	0	0	0	PASS D	x	x	x	x	0	1	1	0	1
0	0	0	1	"	x	x	x	x	0	1	1	1	1
0	0	1	0	D → RB	0	0	0	RML0	1	1	1	1	1
0	0	1	1	"	1	0	0	RML1	1	1	1	1	0
0	1	0	0	PASS D	x	x	x	x	0	1	1	0	1
0	1	0	1	"	x	x	x	x	0	1	1	1	1
0	1	1	0	D → RB	0	0	1	RML0	1	1	1	1	1
0	1	1	1	"	1	0	1	RML1	1	1	1	1	0
1	0	0	0	PASS D	x	x	x	x	0	1	1	0	1
1	0	0	1	"	x	x	x	x	0	1	1	1	1
1	0	1	0	D → RB	0	1	0	RML0	1	1	1	1	1
1	0	1	1	"	1	1	0	RML1	1	1	1	1	0
1	1	0	0	PASS D	x	x	x	x	0	1	1	0	1
1	1	0	1	"	x	x	x	x	0	1	1	1	1
1	1	1	0	D → RB	0	1	1	RML0	1	1	1	1	1
1	1	1	1	"	1	1	1	RML1	1	1	1	1	0

MODE INIT (=0)

le chargement de RADM et le mécanisme d'ERREUR ne sont pas validés.

Y3	Y2	Y1	Y0	instruction	A2	A1	A0	D15	D14	D13...D0	I7	I4/ I1	I3/ Cn	AEDA	valid load	valid erreur
x	x	x	0	RA - D	0	A15	A14	0	0	A15...A0	0	0	1	0	1	1
			1	RA + D	0	A15	A14	0	0	A13...A0	0	0	0	0	1	1
			0	RA - D	1	A15	A14	0	0	A13...A0	0	0	1	1	1	0
			1	RA + D	0	A15	A14	0	0	A13...A0	0	0	0	1	0	1

MODE CALCUL (=1)

les signaux DEM LEC et +1 sont inhibés

$I1 = I4 = \overline{MODE}$ $\quad B0 = Y2 \quad A0 = A14 \quad DEM\ LEC = (\overline{Y0 + Y1}).\overline{MODE}$

$I3 = Cn = \overline{MODE.Y0} \quad B1 = Y3 \quad A1 = A15 \quad +1 = Y1.Y0.\overline{MODE} = \overline{OF(RML1)}$

$I7 = \overline{MODE}.Y1 \quad B2 = Y0 \quad A2 = \overline{Y0} \quad \overline{OE(RML0)} = Y1.\overline{Y0}.\overline{MODE}$

$\overline{Valid}(load) = MODE.A+DA.Y0$

$\overline{Valid}(erreur) = MODE.A+DA.Y0$

Figure 3.5 - Tableau des commandes opérateur.

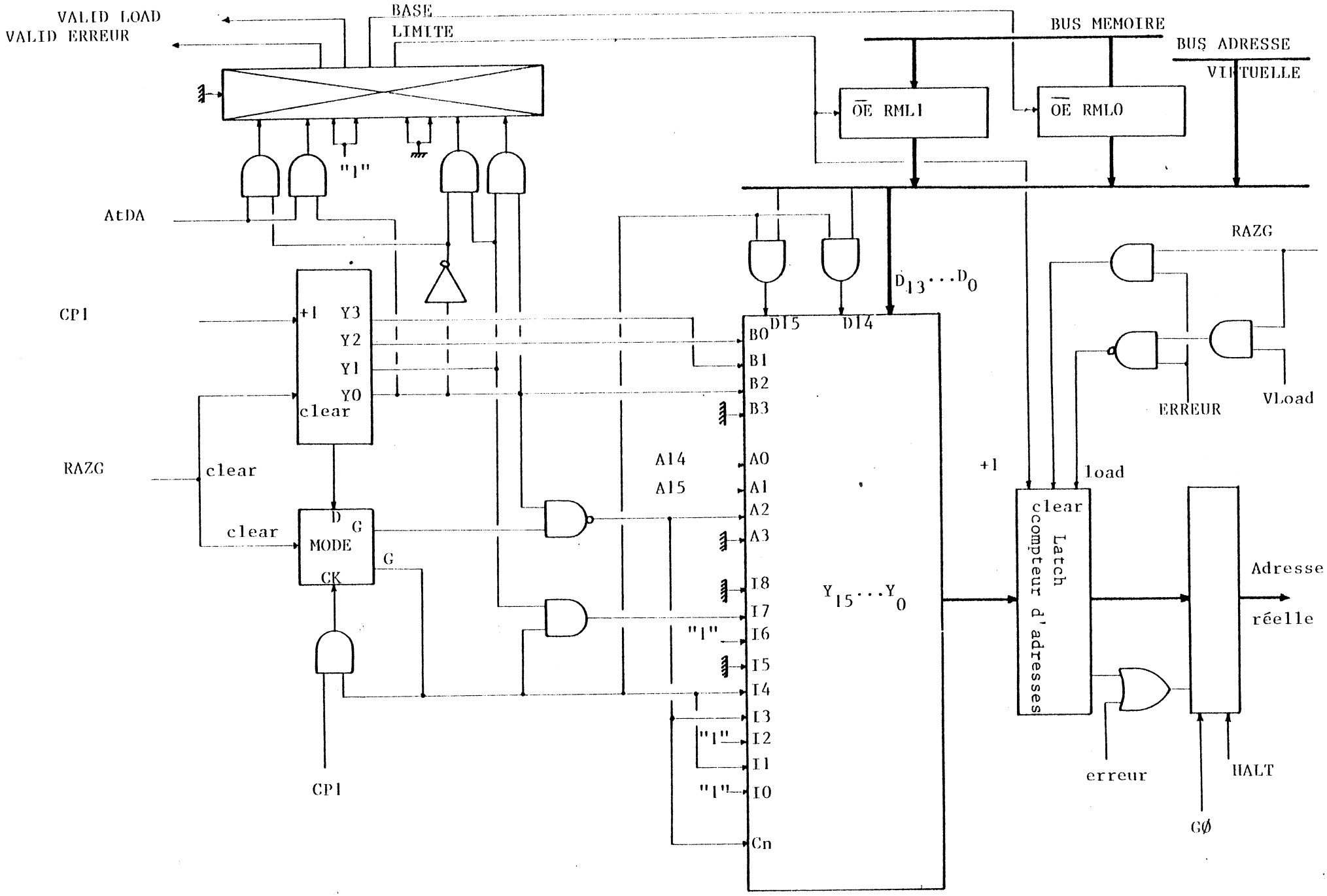


Figure 8.6. - Automate opératif.

VII.3. Automate allocateur

Une demande émanant du processeur numéro i (DEMi) positionne à zéro la bascule de mise en attente correspondante de PME.

Les sorties des quatre bascules sont câblées en entrée d'un encodeur de priorité 74LS148 qui, s'il est validé (entrée E1=0), donne sur ses sorties A0, A1, A2, le numéro du demandeur le plus prioritaire.
Ce numéro est décodé par un circuit 74LS138 dont la sortie Y0 ou Y1 ou Y2 ou Y3 provoque par son passage au niveau bas, la mise à zéro de la bascule maître-esclave correspondante de prise en compte de la demande et est rebouclée de façon à réinitialiser la bascule de mise en attente.

Les signaux BMAi sont issus de ces bascules de prise en compte des demandes et durent un cycle d'horloge.

La reconnaissance provoque la remontée de la sortie Y7 du décodeur (pas de demande ou encodeur invalidé implique que ses sorties sont toutes à "1" et donc que Y7=0).
Ce phénomène est utilisé pour faire remonter E1, donc invalider l'encodeur de façon à éviter le risque de prise en compte d'une demande plus prioritaire pendant l'allocation de PME.

La sortie E0 de l'encodeur est échantillonnée à chaque cycle pour fournir le signal AttDA, puis DEBOP au cycle suivant.
La fin de l'allocation de PME se traduit par l'absence de demandeur pendant la validation de l'encodeur, donc par la retombée de DEBOP.

La figure VII.7 donne le schéma de l'automate allocateur et la figure VII.8 le diagramme des temps correspondant pour deux demandes simultanées.

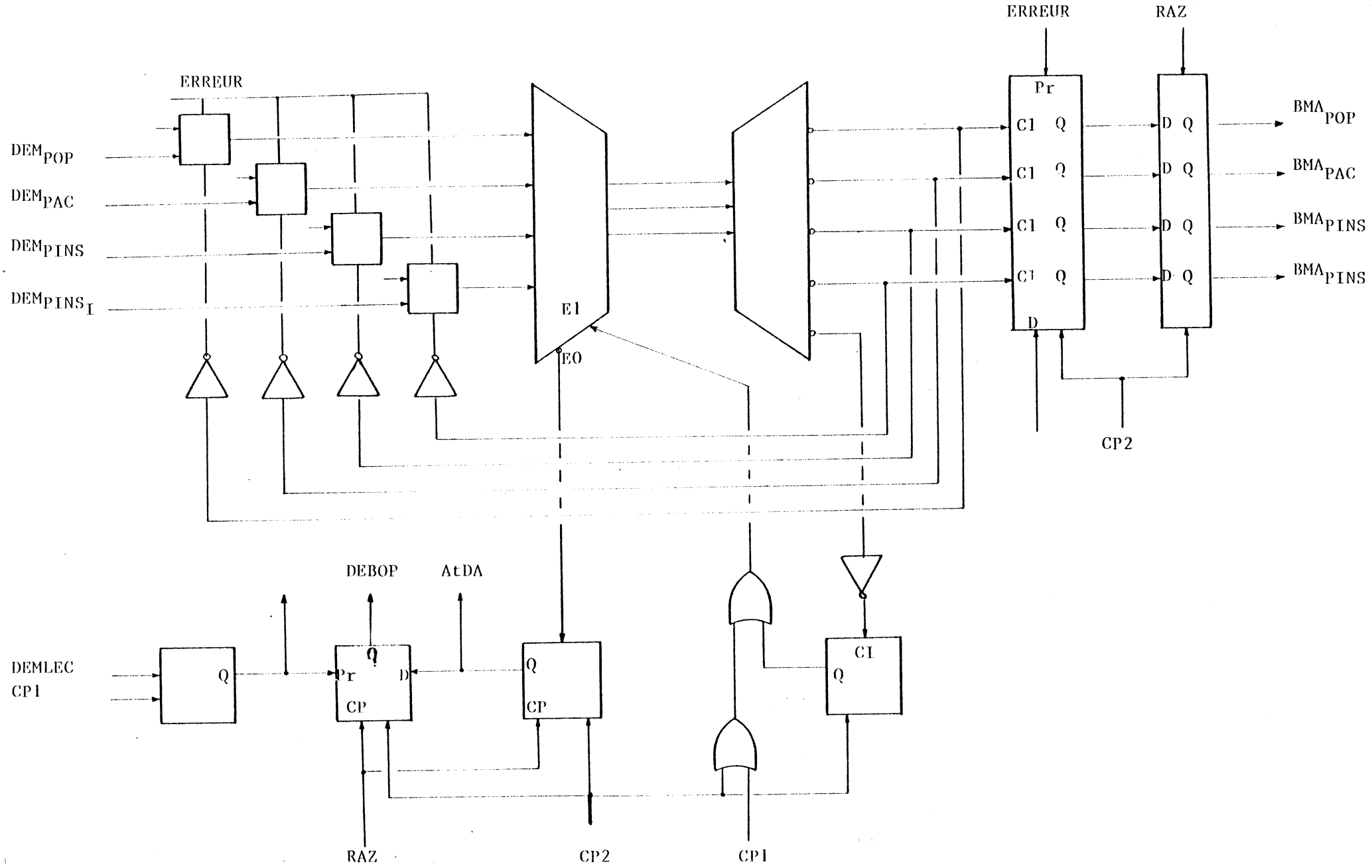


Figure 7.6. - PME : automate allocateur.

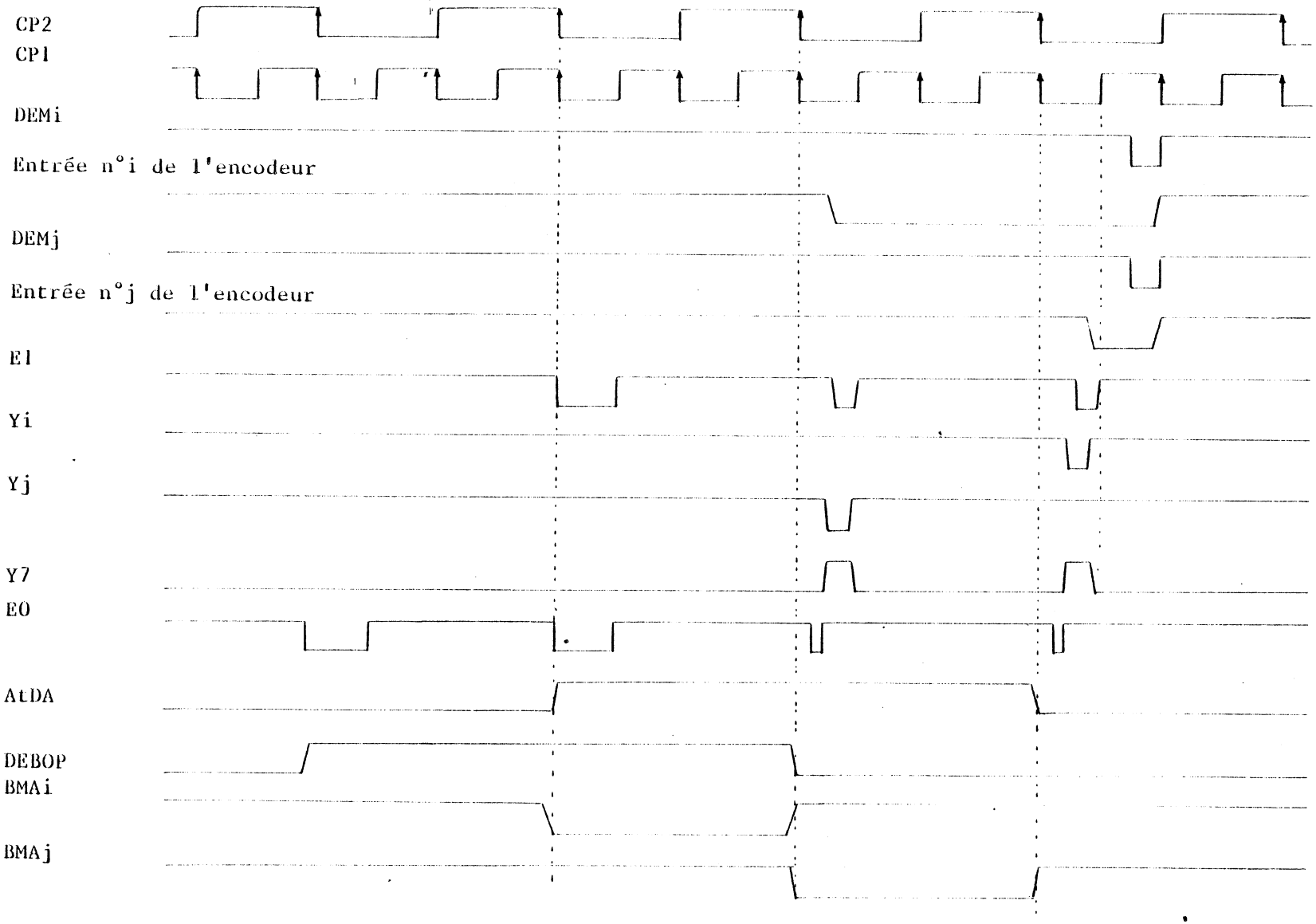


Figure 6.7 - Diagramme des temps de l'automate allocateur pour deux demandes i et j simultanées où i est plus prioritaire que j.

VII.4. Traitement des erreurs

Le processeur mémoire (PME) détecte deux types d'erreurs:

- ERREUR ZONE s'il y a tentative d'écriture dans une zone protégée en écriture.
Le bit A15 de l'adresse joue le rôle de protection. En effet, quand il est à un, on adresse soit la zone CODE, soit la zone EXTERNE dans lesquelles une écriture est interdite. Donc la conjugaison de A15=1 et du signal R/W déroute PME en erreur.

- ERREUR CALCUL s'il y a débordement de zone.
Si l'opération

LIMITE-Adresse virtuelle

est validée par un BMAi le signal VALID ERREUR prmet de prendre en compte la sortie Y15 de l'opérateur qui, s'il est à un, indique un résultat négatif, donc un débordement de zone.

ERREUR ZONE ou ERREUR DE CALCUL donne le signal ERREUR qui initialise une écriture à l'adresse 4 en mèmoire d'un octet contenant le numéro du processeur fautif et le type d'erreur qu'il a commis.

Ce signal est envoyé à tous les processeurs (INTPME) pour qu'ils engagent leurs séquences de sauvegarde.

A la fin de celle-ci (ET logique des FINSAUVE de tous les processeurs) PME est prêt pour une nouvelle initialisation.

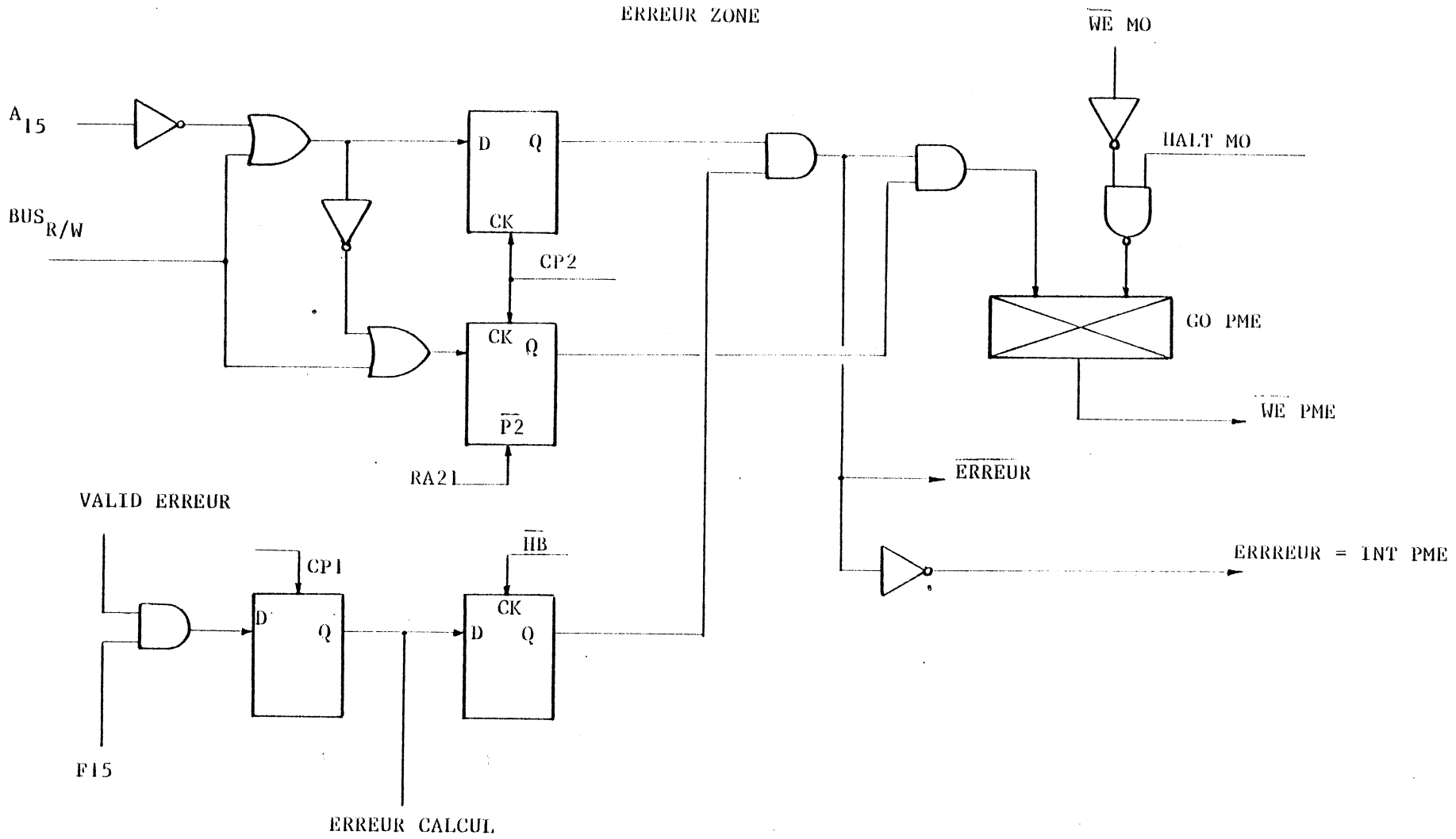


Figure VII.10. - Traitement des erreurs

CONCLUSION

Le prototype qui a fait l'objet de cette ètude occupe 15 grandes cartes (30 x 40) situèes dans deux paniers: un panier pour les processeurs (5 cartes) et un panier pour le systémes de développement (5 fois 2 cartes).

La réalisation du processeur PINS a demandé 134 boîtiers équivalents 16 broches et 256 microinstructions de 36 bits,
celle de PAC 136 boîtiers et 256 microinstructions de 32 bits,
celle de POP 197 boîtiers et 1024 microinstructions de 47 bits,
celle de FILE 69 boîtiers et 256 microinstructions de 24 bits,
soit un total de 625 circuits ce qui représente plusieurs centaines de mètres de connections wrappées.

Nous pensons que cette étude a couvert de nombreux aspects de l'architecture des ordinateurs en apportant des éclairages nouveaux quant à leur conception et à la définition de leur langage intermédiaire, tout en illustrant une méthodologie de conception descendante.

Elle nous a permis d'aborder tous les problèmes rencontrés depuis l'élaboration du cahier des charges jusqu'à la réalisation physique et la mise au point d'un prototype d'une machine spécialisée.
Pour la mise au point du matériel et des microprogrammes de chacun des cinq processeurs nous avons conçu et réalisé le système MADAM (pour matériel d'aide au développement d'applications microprogrammées) dont on trouvera une description en annexe.
Cet outil relativement peu coûteux et d'un intérêt pédagogique certain, permet de charger et de modifier les microinstructions champ par champ dans des mémoires à accès alléatoire se substituant aux mémoires mortes de contrôle des processeurs et d'en contrôler leur exécution (fonctionnement avec l'horloge réelle des processeurs, en pas à pas, jusqu'à un point d'arrêt ...).

Malgré cet outil, la mise au point du matériel et des microprogrammes s'est avérée longue et laborieuse à cause d'une mauvaise conception mécanique du "rack".
En effet, lorsque toutes les cartes étaient en place, il nous était impossible de suivre un signal d'un processeur sur l'autre. Une carte prolongateur permettait de tester facilement tout ce qui se passait sur l'un des processeurs, mais le contrôle des diaïogues nécessitait de changer de cartes avec tous les problèmes de connexions en fond de panier que cela implique.Cette conception mécanique qui à priori ne semblait pas fondamentale peut expliquer en partie le retard pris par ce projet.

Cette réalisation d'une machine langage avec des macrocomposants peut sembler dépassée à l'heure des circuits à très haute intègration, mais les concepts architecturaux développés dans cette étude peuvent trés certainement être transférés sur le silicium et aider la conception de circuits spécialisés.
Elle prouve que dans université française et avec des moyens limités, il est possible de mener à bien une étude de grande envergure débouchant sur un prototype.

ANNEXE 1

M A D A M

MATÉRIEL D'AIDE AU DÉVELOPPEMENT

D'APPLICATIONS MICROPROGRAMMÉES

INTRODUCTION

Le développement d'applications microprogrammées requiert l'utilisation d'un matériel spécialisé. Il existe sur le marché des systèmes généralement coûteux*, qui de plus ne répondent pas forcément aux besoins des utilisateurs. Nous avons cherché à définir un *outil de faible coût* qui offre une *généralité* suffisante pour répondre aux besoins *d'utilisateurs très différents*: mise en oeuvre de microprocesseurs rapides (SIGNETICX 8X300 par exemple), et développement d'applications microprogrammées diverses (bit-slice AM 2900 par exemple).

La généralité du système MADAM est obtenue en définissant des primitives de bas niveau: une instruction est une chaîne de bits, une primitive permet de donner un sens à des champs de cette chaîne.
La souplesse du système et son faible coût sont obtenus par l'utilisation d'un microprocesseur (MC 6800). Son programme interprète un langage de requêtes très simple, qui permet de contrôler jusqu'à 8 applications de natures très différentes.

* Le système 29 de AMD, par exemple.

1ÈRE PARTIE: ASPECTS LOGICIELS DE MADAM

1.1. QU'EST-CE QU'UNE APPLICATION MICROPROGRAMMÉE ?

On entend par "application microprogrammée" tout système électronique a besoin de lire des instructions dans une mémoire de programmes. Cette définition, très large, recouvre donc aussi bien le domaine des processeurs classiques, que celui des ordinateurs à contrôle micropro et elle montre que MADAM ne se restreint pas à une classe particulièr

La liaison entre MADAM et l'application microprogrammée est très nett cette dernière envoie l'adresse de l'instruction suivante, MADAM lui cette instruction. Cette fonction très simple est également appelée " lation de ROM" et elle est réalisée par des appareils appelés "simula de ROM".

La fonction précédente n'est cependant pas suffisante: il faut en out offrir à l'utilisateur:

. des moyens informatiques pour entrer et modifier les instructions c tuant les programmes,

. des moyens électroniques pour contrôler l'exécution des instruction

1.2. MOYENS INFORMATIQUES FOURNIS PAR MADAM

Le contrôle de MADAM par l'utilisateur est fait par l'intermédiaire d'un téléimprimeur (télétype) qui, muni d'un lecteur/perforateur de ruban, constitue un périphérique d'entrée/sortie économique qui sera utilisé soit pour entrer un ruban généré par un cross-assembleur, soit pour sortir un ruban représentant un microprogramme entré au clavier.
(Une commande (E = i) définit que l'on travaille avec MADAM n° i).

1.2.1. Environnement d'édition

Le système MADAM propose à l'utilisateur un jeu de trois commandes pour définir le format symbolique des microinstructions, entrer des microinstructions et imprimer leurs valeurs.

a) Définition d'un format

La commande FOR (format) permet de définir un découpage en champs fonctionnels d'une microinstruction. Chaque champ est défini par son NOM symbolique (jusqu'à 4 caractères) et sa position dans la chaîne de bits: borne inférieure (poids faible) et borne supérieure (poids fort).
La déclaration de format suivante correspondant à l'exemple donné ci-après.

```
.FOR
ETIQ=1C 1F ,SEQ=18 1B ,COND=14 17 ,01=10 13 ,02=0D 0F ,OP=08 0C ,DY=04 07 ,BI=00
```

b) Entrée de valeurs pour les champs d'une microinstruct

L'entrée de mêmes valeurs pour les champs de plusieurs microinstructions est possible en spécifiant une adresse de début et une adresse de fin.
Exemple d'entrée des valeurs par défaut:

```
.I 000 0FF
ETIQ=00,SEQ=00.COND=08,01=00,02=00,OP=00,DY=00,BI=04;
```

L'entrée de valeurs pour un champ particulier spécifie une seule adresse.

Exemple:

```
.I 020
O1=08,O2=01,DY=05;N
021
SEQ=0C,COND=07,OP=1B,DY=06;N
022
SEQ=0C,COND=05,O1=09,BI=0E;N
023
SEQ=0C,COND=05,O1=09,OP=1D,DY=04,BI=07;N
024
OP=01,DY=03;N
025
SEQ=0C,COND=05,O1=09,BI=0E;N
026
SEQ=0C,COND=05,O2=02,OP=16,DY=05,BI=07;N
027
SEQ=0C,COND=06;N
028
SEQ=09;
```

On passe à l'adresse suivante en frappant (N). (On pourrait passer à l'adresse précédente en frappant (U)).

Détecteur d'erreurs:

. Entrée d'un champ non déclaré

```
.I 020
ETUQ=,ETQQ=,ETIQ=00;
```

. Entrée d'un chiffre non hexadécimal

```
.I 020
ETIQ=V ?00;
```

. Entrée d'une valeur incompatible avec la taille déclarée pour le champ

```
.I 020
ETIQ=FF=00
```

c) Impression des microinstructions situées entre deux adresses

. P <adresse début> <adresse fin>

```
.P 020 028
020 ETIQ=00 SEQ=00 COND=08 O1=08 O2=01 OP=00 DY=05 BI=04
021 ETIQ=00 SEQ=0C COND=07 O1=00 O2=00 OP=1B DY=06 BI=04
022 ETIQ=00 SEQ=0C COND=05 O1=09 O2=00 OP=00 DY=00 BI=0E
023 ETIQ=00 SEQ=0C COND=05 O1=09 O2=00 OP=1D DY=04 BI=07
024 ETIQ=00 SEQ=00 COND=08 O1=00 O2=00 OP=01 DY=03 BI=04
025 ETIQ=00 SEQ=0C COND=05 O1=09 O2=00 OP=00 DY=00 BI=0E
026 ETIQ=00 SEQ=0C COND=05 O1=00 O2=02 OP=16 DY=05 BI=07
027 ETIQ=00 SEQ=0C COND=06 O1=00 O2=00 OP=00 DY=00 BI=04
028 ETIQ=00 SEQ=09 COND=08 O1=00 O2=00 OP=00 DY=00 BI=04
```

1.2.2. Génération de rubans perforés

a) Perforation d'un ruban binaire

. D <adresse début> <adresse fin> N = <nombre d'octets>

```
.D 020 028 N=4

S0034FB
S1070020542088003E3
S107002164187000E3
S107002200E00590C6A
S107002347105900C13
S1070024340180000026
S1070025 0E00590C67
S1070026575650 0CD0
S1070027040060 0C68
S1070028040080094A
S9
```

Un premier enregistrement, préfixé par S0, contient le nombre d'octets. Chaque enregistrement suivant, préfixé par S1, contient le nombre d'octets plus 3, l'adresse, la valeur des octets et une somme de contrôle (check-sum).

La fin du ruban est définie par S9.

b) Perforation d'un ruban pour un programmateur de PROM

. Ø <adresse début> <adresse fin>

N = <nombre de PROM>

L = <nombre de bits par PROM (4 ou 8)>

Exemple 1: 4 PROM de 32 mots de 8 bits

```
•Ø Ø2Ø Ø3F N=4 L=8
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
54 64 ØE 47 34 ØE 57 Ø4 Ø4 Ø4 Ø4 Ø4 Ø4 Ø4 Ø4 Ø4 Ø4 Ø4 Ø4 Ø4 Ø4 Ø4 Ø4 Ø4 Ø4 Ø4 Ø4 Ø4 Ø4 Ø4 Ø4
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
2Ø 1B ØØ 1D Ø1 ØØ 56 ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
88 7Ø 59 59 8Ø 59 5Ø 6Ø 9Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
ØØ ØC ØC ØC ØØ ØC ØC ØC Ø9 ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

Exemple 2: 8 PROM de 32 mots de 4 bits

```
 Ø Ø2Ø Ø2
E=2
 A=ØØØ C=Ø3C
.Ø Ø2Ø Ø3F N=8 L=4
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
4 4 E 7 4 E 7 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
5 6 Ø 4 3 Ø 5 Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Ø B Ø D 1 Ø 6 Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
2 1 Ø 1 Ø Ø 5 Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
3 3 9 9 Ø 9 Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø 3 Ø Ø 3 Ø Ø Ø 3 Ø Ø Ø Ø Ø Ø Ø Ø Ø 3
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
8 7 5 5 8 5 5 6 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Ø C C C Ø C C C 9 Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
3 3 Ø Ø Ø Ø Ø Ø Ø Ø Ø 3 Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø 3
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

c) Lecture d'un ruban

Tout ruban perforé par la commande D, ou généré par un cross-assembleur selon le même format, peut être chargé dans la mémoire de microprogramme par la commande L (LOAD) qui vérifie la cohérence des informations (somme de contrôle).

1.3. POSSIBILITÉS DE CONTRÔLE DE L'EXÉCUTION

Une fois qu'un microprogramme a été chargé ou modifié dans la mémoire de contrôle, l'utilisateur peut naturellement chercher à le faire exécuter par son application microprogrammée. Il dispose pour cela d'un certain nombre de commandes de contrôle de l'exécution.

1.3.1. Remise à l'état initial

La commande R (reset) frappée au clavier, génère une impulsion d'une durée de quelques microsecondes, qui peut être utilisée pour la remise à l'état initial de l'application.

1.3.2. Définition de points d'arrêt

Avant de lancer l'exécution, l'utilisateur peut poser un point d'arrêt sur n'importe quelle instruction. Lorsque, en cours d'exécution, une telle micro instruction sera exécutée, alors l'exécution de la microinstruction suivante sera suspendue.

Définition de points d'arrêt:

```
.K 000 00F 028
```

Suppression des points d'arrêt entre deux adresses:

```
.S 000 0FF
```

1.3.3. Introduction de paramètres (clefs)

Des informations logiques, portées par 16 signaux, peuvent être envoyées par MADAM vers l'application, pour positionner des paramètres d'entrée. Ces paramètres sont statiques en ce sens qu'ils ne peuvent être modifiés par l'utilisateur que si l'application est arrêtée.

1.3.4. Lecture de résultats (voyants)

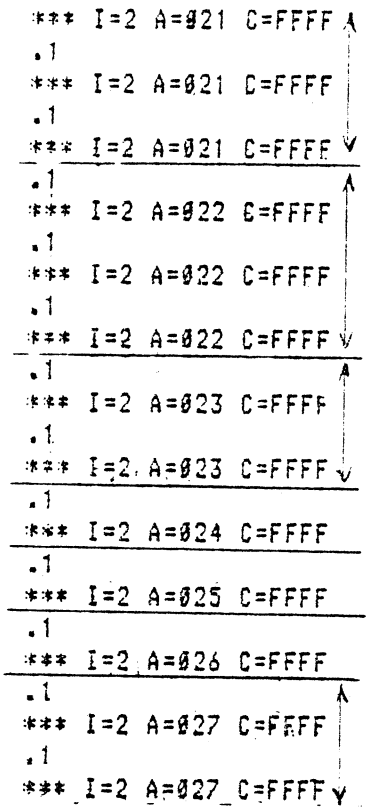
D'autres informations logiques, portées par 16 signaux, peuvent être envoyé par l'application vers MADAM, pour afficher des valeurs en sortie. Ces valeurs sont statiques, en ce sens qu'elles ne peuvent être imprimées que si l'application est arrêtée.

1.3.5. Lancement de l'exécution

L'utilisateur peut choisir parmi plusieurs modes de lancement de l'exécutio

a) La plus simple est l'exécution en pas à pas, demandée par la commande '1

Exemple faisant apparaître des cycles d'attente:



b) Une adresse de départ et un nombre de cycles peuvent être spécifiés:
G A = <adresse> N = <nombre>

Remarque: lorsqu'on veut continuer là où l'exécution a été suspendue, on frappe G A = X ...

c) Une adresse de départ peut être spécifiée, ainsi que l'arrêt après le premier point d'arrêt rencontré (option K): G A = <adresse> K

d) Combinaison des deux options: on spécifie un nombre de cycles et l'arrêt sur point d'arrêt. La première condition satisfaite provoque la suspension de l'exécution

G A = <adresse> N = <nombre> K

```
.G A=000 N=4

*** I=2 A=021 C=FFFF.
.G A=X N=3

*** I=2 A=022 C=FFFF.
.G A=X N=4

*** I=2 A=026 C=FFFF.
```

Remarque: on peut remettre à zéro le compteur de cycles (entre deux commandes G), par la commande Z, ce qui permet de lire le nombre de cycles exécutés entre deux points d'arrêt.

2ÈME PARTIE: ASPECTS MATÉRIELS DE MADAM

Le système MADAM est construit autour d'un microprocesseur MOS 8 bits de MOTOROLA (MC 6800 ou SFF 96800 chez EFCIS).

On distingue trois modules:

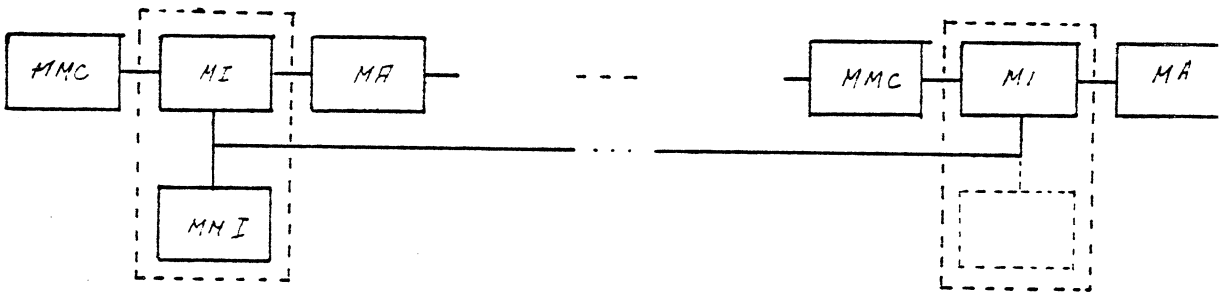
. le module microprocesseur (MMI),

. le module interface (MI),

. le module mémoire de contrôle (MMC).

Le module interface MI assure l'interface entre le module microprocesseur, le module mémoire de contrôle et une application (MA).

Ce découpage permet, avec un seul module microprocesseur, de contrôler plusieurs applications: le système est alors appelé MULTI-MADAM.



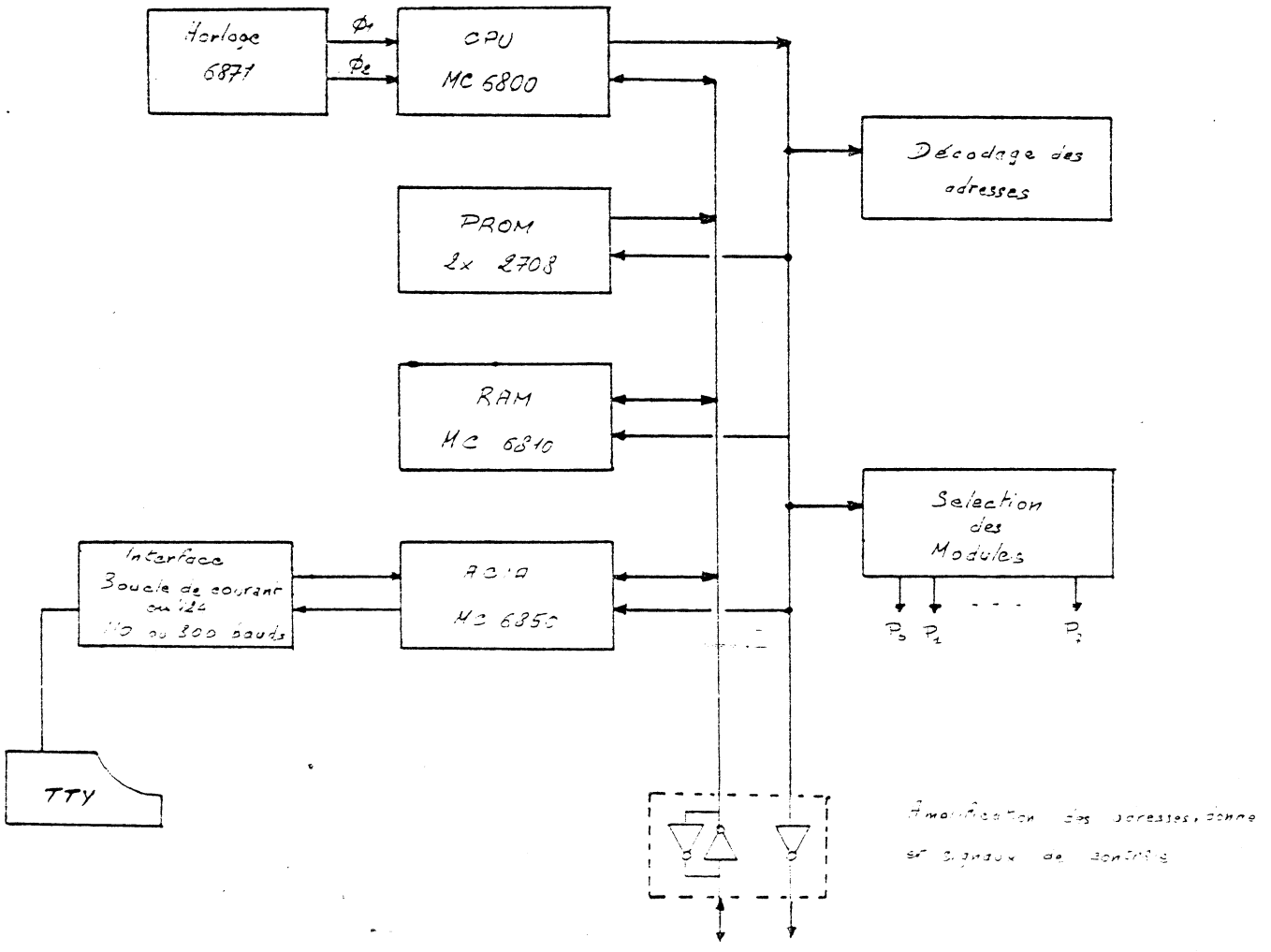
La même carte peut contenir un module MMI et un module MI. Dans le cas d'un système MULTI-MADAM, une seule de ces cartes est montée avec le module MMI, toutes les autres étant montées avec le module MI seul. Chaque module MI est physiquement défini par un numéro. Lorsque MMI veut dialoguer avec un MI, il préfixe toutes les adresses qu'il envoie avec ce numéro (appelé numéro de PAGE).

2.1. LE MODULE MICROPROCESSEUR MMI

Ce module est très simple et relativement classique. Il est organisé autour d'un microprocesseur 8 bits MC 6800.

Il est constitué de:

. 2 K de mémoire de programme (2 circuits 2708 REPROM),
. 128 mots de mémoire vive (1 circuit MC 6810),
. une interface série (ACIA - MC 6850),
. un décodeur de pages,
. un encodeur de priorité pour les interruptions,
. des amplificateurs de bus adresse et données.



Le décodage des adresses permet de sélectionner les composants du module MMI, qui sont implantés aux adresses suivantes:

PROM : entre F800 et FFFF (2 K octets)

RAM : entre 0000 et 007F (128 octets)

ACIA : registres d'état en 6000
registres de donnée en 6800

module MI_0 : à partir de l'adresse 1000

module MI_1 : à partir de l'adresse 1100

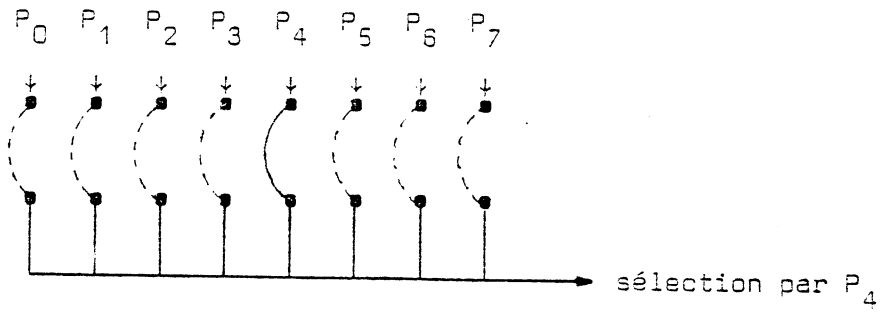
:

module MI_7 : à partir de l'adresse 1700

2.2. LE MODULE INTERFACE MI

Le module n° i reconnaît les adresses qui sont sous la forme 1ixx.
Le module MMI envoie une sélection de la page n° i et la reconnaissance est faite par le positionnement d'un commutateur à 8 positions.

Exemple:



Les signaux de contrôle envoyés par MMI valident le décodage des adresses qui provoque la sélection d'un des composants du module MI.

. Une mémoire RAM de 128 octets pouvant contenir la description d'un format de microinstruction (jusqu'à 21 champs), située entre les adresses relatives 80 et FF.

. Six tampons d'écriture et six portes de lecture pour échanger des données de 48 bits avec la mémoire de contrôle (module MMC), situés entre les adresses 00 et 05.

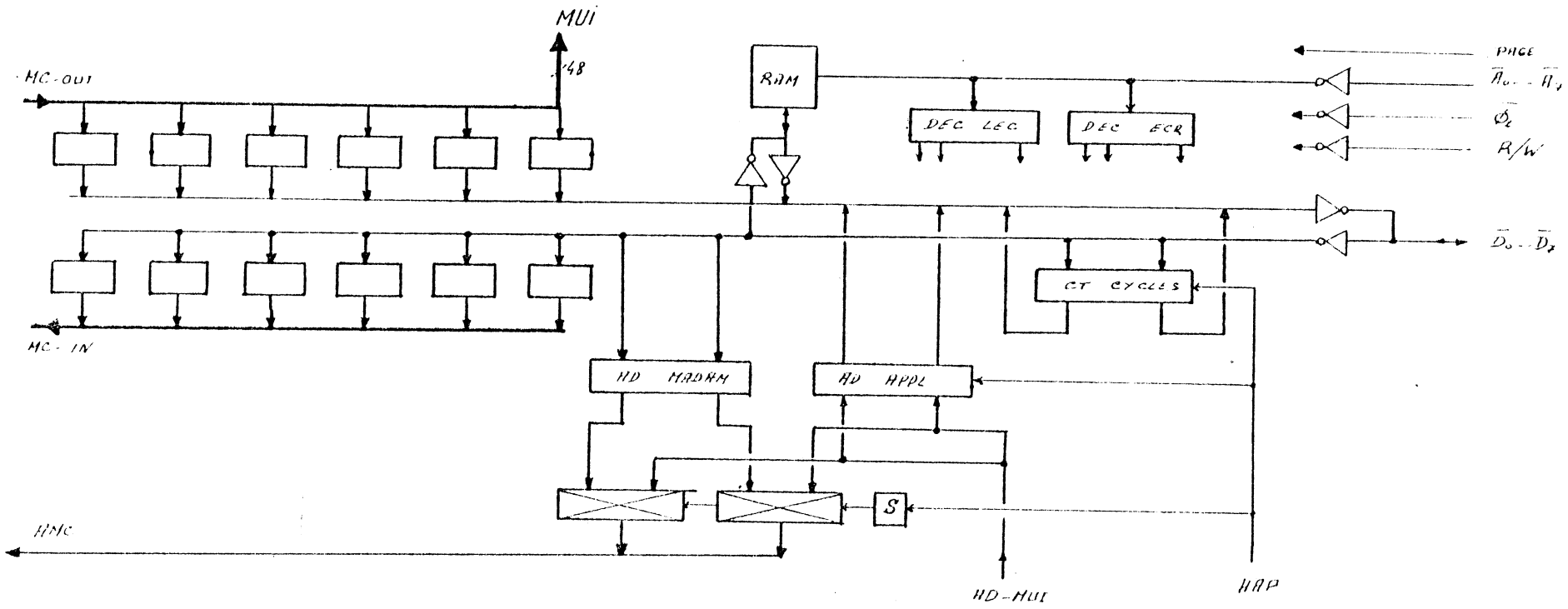
. Deux tampons d'adresse pour lire ou charger les microinstructions (AD.MADAM) deux portes pour lire l'adresse de la microinstruction suivante (AD.APPL.), situés aux adresses 06 et 07.

. Un compteur de cycles (CT.CYCLES) de 16 bits aux adresses 08 et 09.

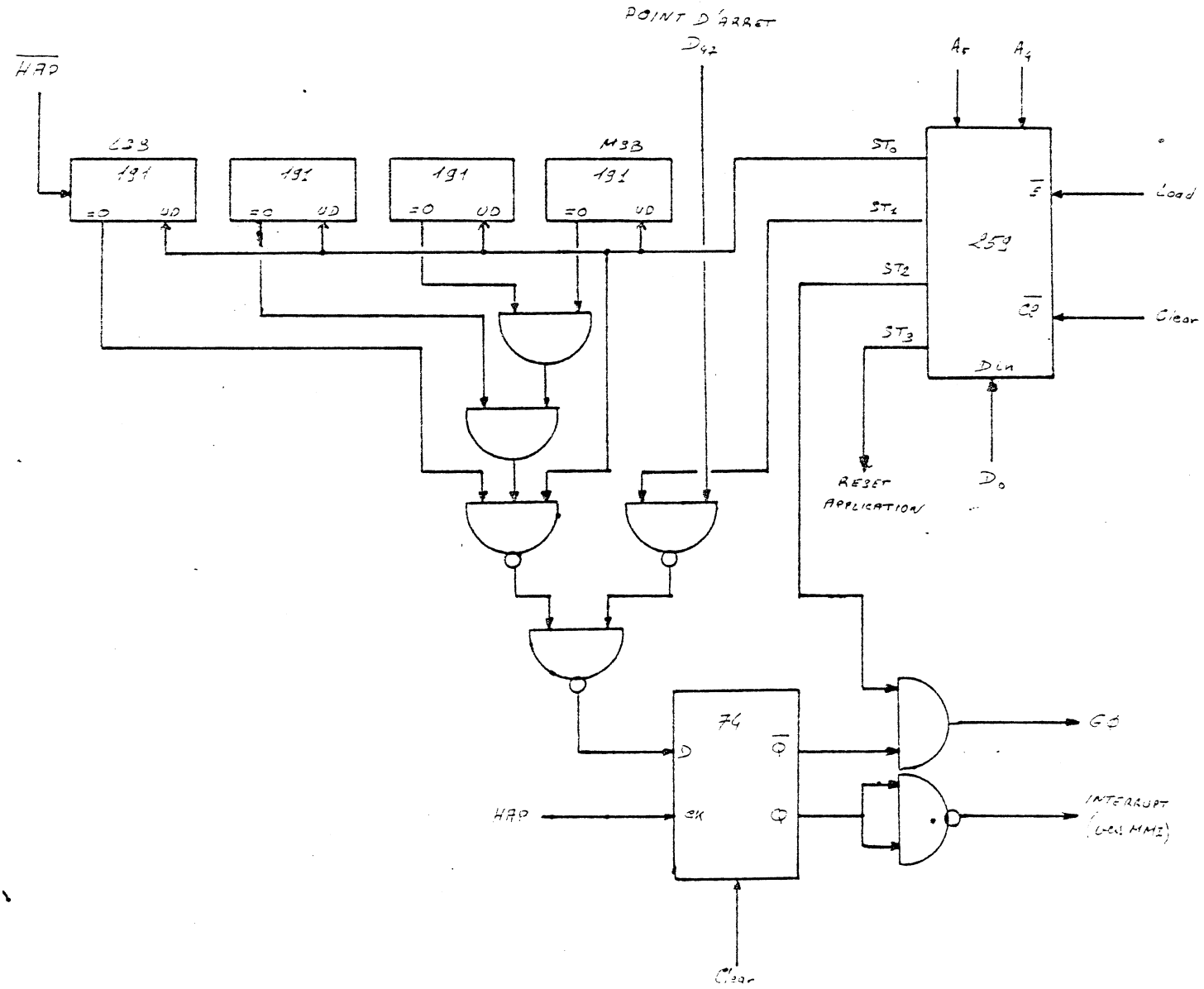
. Des ordres de contrôle:
- remise à zéro du module, adresse 0B,
- écriture en mémoire de contrôle, adresse 0D,
- validation d'arrêt sur compteur, adresse 0A,
- validation d'arrêt sur point d'arrêt, adresse 1A,
- mise à 1 du signal marche/arrêt, adresse 2A,
- contrôle du signal RESET, adresse 3A.

Le contrôle de l'exécution doit être fait en temps réel, au rythme donné par l'horloge de l'application HAP.
Un registre de contrôle (chargé par la commande G) permet de lancer ou d'arrêter l'exécution par ST_2 en agissant sur le signal GO,
de valider l'arrêt sur passage à zéro du compteur par ST_0,
de valider l'arrêt sur une rencontre d'un point d'arrêt ($D_{47} = 1$) par ST_1

Ce contrôle est réalisé par le montage dessiné à la page 16.



Chemin de Donnée du Module Interface de MADRM


POINT D'ARRET
D_{42}
$\overline{HAD}$
A_5
A_4
C33
M33
191
=0
UD
ST_0
ST_1
ST_2
ST_3
259
$\overline{E}$
$\overline{CR}$
Load
Clear
D_{LN}
D_0
RESET APPLICATION
74
D
$\overline{Q}$
CK
Q
HAD
Clear
GO
INTERRUPT

2.3. LE MODULE MÉMOIRE DE CONTRÔLE MMC

Ce module peut se présenter sous des formes différentes, selon le type de mémoire RAM utilisé.

Fonctionnellement, il est défini comme une mémoire RAM de N mots de M bits, avec des entrées (DIN) et des sorties (DOUT) distinctes et de polarité inverse.
Dans la version actuelle:

$N \leq 4\ K = 4096$ mots

$M = 48$ bits

le module se décomposant en "cartes" de 512 mots de 48 bits, que l'on peut restreindre à, par exemple, 256 mots de 32 bits. Les circuits RAM utilisés sont de type INTEL 3106 ou AMD 27LS00, mémoires rapides de 256 fois 1 bit.

3ÈME PARTIE: PROGRAMMATION DU SYSTÈME

Le contrôle du système MADAM est assuré par un microprocesseur 8 bits (MC 6800) qui exécute un programme d'une taille d'environ 2 K octets. Ce programme se présente sous la forme d'un interpréteur de commandes qui peut se trouver dans un environnement spécifié par l'utilisateur.

3.1. NOTION D'ENVIRONNEMENT

La commande E = i ($0 \leq i \leq 7$), frappée par l'utilisateur, conduit à l'initialisation d'une variable appelée CONF avec la valeur hexadécimale '1i00'. L'interprétation de toute autre commande commence par le chargement du registre d'index avec la valeur de cette variable (LDX CONF). Tout élément du module interface n° i est alors adressé relativement à l'index (adressage indexé).

3.2. TABLE DES FORMATS

Chaque champ déclaré par la commande FORMAT est décrit par 6 octets. Les 4 premiers octets mémorisent le nom du champ ($\leq$ 4 caractères), les 2 derniers la borne inférieure et la borne supérieure.

Cette table est mémorisée dans une mémoire RAM de 128 octets, les deux premiers servant à noter l'adresse de la fin de la table.

On peut déclarer jusqu'à 21 champs ($6 \times 21 + 2 = 128$).

3.3. ENTRÉE D'UNE VALEUR DANS UN CHAMP

Un champ est décrit par son nom symbolique. Une recherche dans la table des champs fournit ses bornes inférieure et supérieure, qui définissent sa taille (nombre de bits) et sa position dans un octet ou à cheval sur deux octets.

La modification de la valeur d'un champ suppose donc un traitement assez sophistiqué: des décalages, des masquages et une recopie des octets non modifiés, avant toute modification dans la mémoire de contrôle.

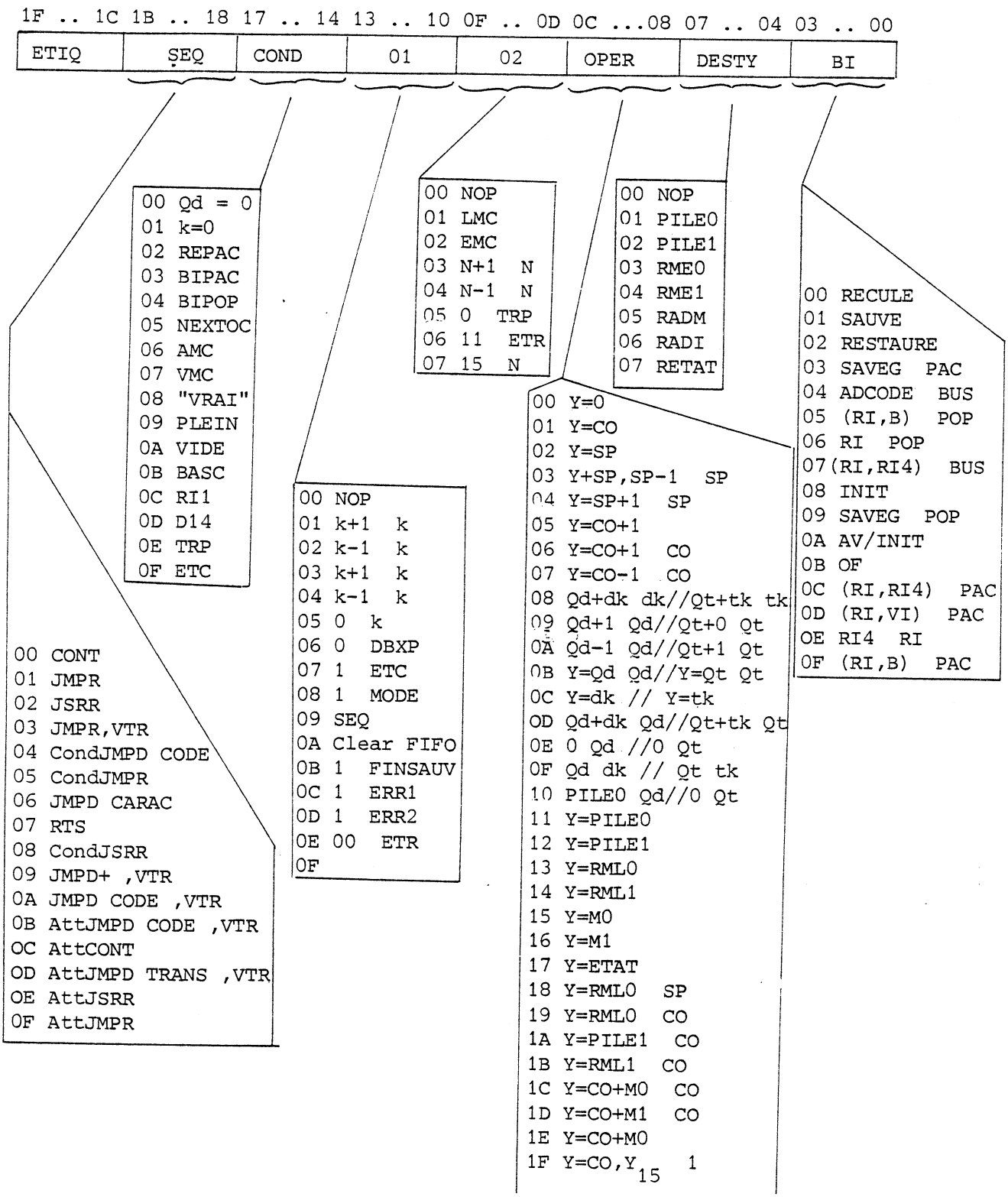
3.4. IMPRESSION DE LA VALEUR DES CHAMPS

Elle suppose un parcours de la table des champs. Chaque champ est extrait de l'octet ou des octets dans lesquels il se trouve et, après des décalages et des masquages, il est imprimé sous forme hexadécimale.

ANNEXE 2

FORMAT ET CODAGE DES MICROINSTRUCTIONS

PINS : Format et codage des microinstructions



@	Étiquette	SEQ	ETIQ	COND ATT	01 ORDRE	02 ORDRE	OPERATION	DEST	BUS INSTRUCTION	Commentaires
00	RESTART	CONT			clear FIFO	LMC	Y=0	RADM		départ
01		att,CONT		VMC			$Y=RML_1$	ETAT		lecture en ZERO
02		CONT				LMC	$Y=RML_0 \rightarrow SP$	RADM		lecture en [SP]
03		CONT				15 → N	SP-1 → SP			
04		att,CONT		VMC			$Y=RML_1 \rightarrow CO$	RADI		lecture code
05		JMPR	SREP	D_{14}			Y=ETAT			
20	SREP(0)	JMPR	FETCH							reprise I/O
21	SREP(1)	att,CONT		NEXTOC	SEQ		COM → CO		$RI_4 \rightarrow RI$	reprise CALL
22		att,CONT		NEXTOC	SEQ	11 → ETR	COM → CO			
23		att,JMPR	RCALL	BIPAC					$RI,RI_4 \rightarrow PAC$	
24	EMPILE	JSRR	PUSH	PLEIN						
25		JSRR	PUSH	PLEIN			$Y_d=Q_d$	$PILE_0$		
26	RCALL	att,CONT		REPAC		0 → REPAC	$Y=M_1$	$PILE_0$	ADCODE → BI	
27		JMPR	DCALL	D_{14}			$Y=M_1$		ADCODE → BI	
28	DCALL(0)	JMPR	FETCH							code présent
29	DCALL(1)	CONT			0 → CIO		CO-1 → CO			code absent
2C	SAVEG	att,CONT		BIPOP					SAVEG → POP	
2D		JMPR	SAVE	VIDE						
2E	SAVE(0)	att,JMPR	SSAVE	AMC			Y=SP+1 → SP	RADM		
2F	SAVE(1)	att,JMPR	FSAVE	AMC			Y=0	RADM		
30	SSAVE	CONT				N-1 → N	$Y=PILE_0$	RME_0		sauvegarde
31		JMPR	SAVE	VIDE	K-1 → K	EMC	$Y=PILE_1$	RME_1		
32	FSAVE	CONT					Y=SP	RME_0		
33		CONT				EMC	Y=ETAT	RME_1		
34		att,CONT		AMC	FINSAUV					
35		JMPD,*								

@	Étiquette	SEQ	ETIQ	COND ATT	O1 ORDRE	O2 ORDRE	OPERATION	DEST	BUS INSTRUCTION	Commentaires
06	CTCF	JMPD,*		$\overline{ETC}$	$0 \rightarrow k$		$0 \rightarrow Q_d$			CT → CALLF
07		att,CONT		NEXTOC	SEQ		$CO+1 \rightarrow CO$			
08		att,JMPR	EMPILE	BIPAC					$RI,RI_4 \rightarrow PAC$	
16	ACOP	att,CONT		BIPOP	$0 \rightarrow DBXP$		$Y_d=Q_d$		AV/INIT	ACC → OP
17		att,JMPD-R		BIPOP					RI → POP	
18	ACOF	att,JMPD-R		BIPOP			$Y_d=Q_d$		OF → POP	ACC → OF
19	CTAC	CONT			$0 \rightarrow k$		$0 \rightarrow Q_d$			CT → ACC
1A		att,CONT		NEXTOC	SEQ		$CO+1 \rightarrow CO$			
1B		att,JMPD-R		BIPAC			$Q_d+1 \rightarrow Q_d$		$RI,RI_4 \rightarrow PAC$	
40	OPOP	cond,JMPR	OPE	$Q_d \neq 0$						OP → OP
41		att,CONT		BIPOP			$Y_t=t_k$		RECULE	
42		JMPR	OPE		$k-1 \rightarrow k$		$Q_t+t_k \rightarrow t_k$			
43	CTZO	CONT			$0 \rightarrow k$		$0 \rightarrow Q_d$			CT → ZO
44		att,JMPR	FETCH	BIPAC			$Q_d+1 \rightarrow Q_d$		RI,VI → PAC	
45	OPZO	cond,JSRR	ETIQ	$Q_d \neq 0$	$k+1 \rightarrow k$					OP → ZO
46		CONT					$Q_t+t_k \rightarrow t_k$			
47		CONT					$0 \rightarrow Q_t$			
48		att,JMPR	FETCH	BIPAC			$Q_d+1 \rightarrow Q_d$		RI,VI → PAC	
49	OPAC	cond,JSRR	ETIQ	$Q_d \neq 0$	$k+1 \rightarrow k$					OP → ACC
4A		CONT					$Q_t+t_k \rightarrow t_k$			
4B		CONT					$0 \rightarrow Q_t$			
4C	ACAC	att,CONT		NEXTOC	SEQ		$CO+1 \rightarrow CO$			ACC → ACC
4D		att,JMPD-R		BIPAC			$Q_d+1 \rightarrow Q_d$		$RI,RI_4 \rightarrow PAC$	
4E	ACFE	att,JMPR	CTRL	BIPOP			$Y_d=Q_d$		AV/INIT	ACC → FE
4F	OPFE	cond,JMPR	CTRL	$Q_d \neq 0$						OP → FE
50		att,JMPR	CTRL	BIPOP			$Y_t=Q_t$		RECULE	

Microprogrammes des transitions

@	Étiquette	SEQ	ETIQ	COND ATT	O1 ORDRE	O2 ORDRE	OPERATION	DEST	BUS INSTRUCTION	Commentaires
51	ACZO	att,JMPR	FETCH	BIPAC			$Q_d+1 \rightarrow Q_d$		RI,VI → PAC	ACC → ZO
F9	ACCCF	JMPD,*		$\overline{ETC}$						ACC → CALLF
FA		att,JMPR	SF	BIPOP			$Y_d=Q_d$		AV/INIT	
FB	OPCF	JMPD,*	/	$\overline{ETC}$						OP → CALLF
FC		att,CONT		BIPOP			Y=O		AV/INIT	
FD		cond,JMPR	S10	$Q_d \neq O$	k+1 → k					
FE		JMPR	S11				$Q_t+t_k \rightarrow t_k$			
E2	SF	cond,JMPR	S10	$Q_d \neq O$	k+1 → k					
E3		JMPR	S11				$Q_t+t_k \rightarrow t_k$			
E4	S11	att,CONT		BIPOP			$Q_d+1 \rightarrow Q_d$		INIT	
E5		att,CONT		NEXTOC	SEQ		CO+1 → CO			
E6		att,CONT		BIPAC					$RI,RI_4 \rightarrow PAC$	
E7		att,JMPR	TESTK	BIPOP			$Y_d=Q_d$		SAUVE	
E8	S10	CONT					$Q_d \rightarrow d_k$			
E9		JMPR	S11				$O \rightarrow Q_d$			
EA	TESTK	cond,JMPD-R		k=O						
EB		att,CONT		BIPOP			$Y_t=t_k$		RECULE	
EC		att,CONT		BIPOP			$Y_d=d_k$		SAUVE	
ED		JMPR	TESTK		k-1 → k		$d_k+Q_d \rightarrow Q_d$			
EE	FETCH	att,JMPD-T		NEXTOC	SEQ		COM → CO		$RI_4 \rightarrow RI$	FETCH
60	OPE	att,JMPD-R		BIPOP					RI → POP	
62	ETIQ	CONT					$Q_t \rightarrow t_k$			
1C	BIN2	att,CONT		NEXTOC	SEQ		COM → CO		$RI_4 \rightarrow RI$	binaire L=2
		att,JMPR	FETCH	BIPOP				Z=1	RI → BIPOP	Unaire L=2

Microprogrammes des transitions

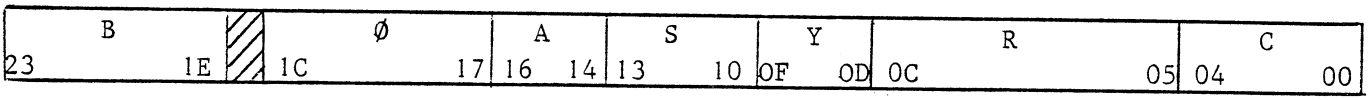
!	Étiquette	SEQ	ETIQ	COND ATT	01 ORDRE	02 ORDRE	OPERATION	DEST	BUS INSTRUCTION	Commentaires
30	THEN	JSRR	PUSH	PLEIN						
		CONT					Y=CO+1 → CO	$PILE_0$		
		att,JMPR	FETCH	NEXTOC	SEQ		Y=CO+M_0	$PILE_1$	RI,RI_4 → BI	
33	ELSE	JSRR	PUSH	PLEIN						
		JMPR	FETCH				Y=CO	$PILE_0$		
35	UNTIL	JSRR	ACCES	VIDE						
		JMPR	FETCH				Y=$PILE_0$ → CO	RADI		
37	NEHT	JSRR	POP	VIDE			CO+1 → CO			
		att,JMPR	FETCH	NEXTOC			Y=CO+M_0 → CO	RADI	RI,RI_4 → BI	
39	GOTO	JSRR	ACCES	VIDE						
		att,CONT	1	NEXTOC	SEQ		Y=$PILE_0$ → CO		RI_4 → RI	
		att,JMPR	FETCH	NEXTOC			Y=CO+M_1 → CO	RADI	RI,RI_4 → BI	
3C	CASE	JSRR	PUSH	PLEIN			CO+1 → CO			
		att,CONT		NEXTOC	SEQ		Y=CO+M_0	$PILE_1$	RI,RI_4 → BI	
		JSRR	PUSH	PLEIN						
		JMPR	FETCH							
90	OF	JSRR	POP	VIDE						
		JSRR	ACCES	VIDE						
		JSRR	PUSH	PLEIN			Y=CO+1 → CO	$PILE_0$		
		CONT					Y=CO	$PILE_1$		
		att,JMPR	FETCH	NEXTOC			Y=CO+M_0 → CO	RADI	RI,RI_4 → BI	
95	FO	JSRR	ACCES	VIDE						
		JMPR	PFETCH	VIDE			Y=$PILE_1$ → CO	RADI		
97	ESAC	JSRR	POP	VIDE						
		JMPR	PFETCH	VIDE						

Microprogramme de traitement des instructions de contrôle

@	Étiquette	SEQ	ETIQ	COND ATT	01 ORDRE	02 ORDRE	OPERATION	DEST	BUS INSTRUCTION	Commentaires
C0	$CARAC_0$	JMPD-R								
C1		JMPD-R								
C2	POPBIT	att,JMPD-R		BIPOP					RI → POP	
C3		att,JMPD-R		BIPOP					RI → POP	
C4	PACBIT	JMPR	SOPAC							
C5		JMPR	SOPAC							
C6	POPAC	att,JMPR	SOPAC	BIPOP					RI → POP	
C7		att,JMPR	SOPAC	BIPOP					RI → POP	
C8	ATTBIT	JMPD,*		$\overline{ETC}$						
C9		JMPD-R								
CA	ATTPOP	JMPD,*		$\overline{ETC}$						
CB		att,JMPD-R		BIPOP					RI → POP	
CC	ATTPAC	JMPD,*		$\overline{ETC}$						
CD		JMPR	SOPAC							
D8	CONDATT	JMPD,*		$\overline{ETC}$						
D9		JMPD-R			1 → ETC					
DA	CATTPOP	JMPD,*		$\overline{ETC}$						
DB		att,JMPD-R		BIPOP	1 → ETC				RI → POP	
E0	SOPAC	att,CONT		NEXTOC			CO+1 → CO			
		att,JMPD-R		BIPAC					$RI,RI_4 \rightarrow PAC$	

Micorprogrammes de traitement des caractéristiques

PAC : FORMAT DES MICROINSTRUCTIONS



09 : SPARAM
0A : ERR
0B : ERRSUI
0C : ATFILE
0D : ATLIT
16 : LIT
20 : DEBUT
25 : SSAUVE
28 : SENTER
2D : BOUC 1
2E : ET 2
31 : FINENT
32 : SCALL
35 : NP
37 : ETI 1
38 : MARK
39 : TSEG
3D : SRET

00 : BIPAC VIDE
01 : DPAC
02 : AMC
03 : VMC
04 : F = 0
05 : F ≠ 0
06 : ALCAM
07 : BIPAC 8

00 : LMC
01 : EMC
02 : FINSAUVE
03 : DPAC — "1" & ALLOCAM — "0"
04 : NEXTBIPAC
05 : ALLOCAM — "0"
06 : ALLOCAM — DCAM

08 : INIT
09 :
0A : MODE — "INT"
0B : MODE — "EXT"
0C : Load MODE

0D : TROUVE — "0"
0E : DPINS — "0"
0F : /Nop/

07 : ERR 0
17 : ERR 1
27 : ERR 2
37 : ERR 3 — PINS

18 RECH (S,D,VIDE)
19 RECH (S,VIDE)
1A RECH (III,VIDE)
1B RECH ( )
1C ECR (VIDE)
1D ECR (S)
1E Nop
1F Nop

Adressage de la mémoire centrale
en conjugaison avec
DY = RADM :

0F : CTX
1F : CDE
2F : EXT
3F : DYN

00 : JMPR(VORO-1)
01 : DEBLOOP
02 : POPSTACK
03 : Continue(VORO-1)
04 : LOOP
05 : JSR AR
06 : RTS
08 : Continue(CondLoop)
09 : Continue(AH.JMPR)
0A : Continue(AH.Continue)
0B : Continue(AH.Loop)
0C : Continue(Cond.JMPR)
0D : Continue (CondEnd Loop)
0E : JMPR
0F : /Continue/

00	Nop/
01	Y="0"
02	Y="0" — RE
03	Y=Q
04	Y=Q+1
05	Y=Q+1 — Q
06	Q/2 — Q
07	Y=2*Q — RB
08	Y=Q RML(1)
09	Y=RML(0) —
0A	Y=RML(1) —
0B	Y=ROM+RA —
0C	Y=BIPAC
0D	Y=NVC
0E	Y=VI(1)
0F	Y=VI(2)
10	Y=ETAT
11	Y=RML(0)
12	Y=RML(1)
13	Y=RML(0)-1
14	Y=RML(0)—
15	Y=RML(1)—
16	Y=RML(0)+RA
17	Y=ROM+RA— R
18	Y=RA
19	Y=RA-1
1A	Y=RA+1 — R
1B	Y=RA;RA+1—
1C	Y=RA+RB —
1D	RA — RB
1E	Y=RA-1 — R
1F	RA+RB

ETIQUETTE	@	BRANCH	ORDRE	ATTENTE CONDITION	SEQUENCEMENT	DESTINATION Y	REGISTRES RA	REGISTRES RB	CODOP
DEPART	00		INIT		Continue			R0	Y="0" — RB
	01		DCAM — "0"		Continue		R0	R1	Y=RA+1 — RB
	02	DEBUT	LMC (CTX)		JMPR		R1	R1	Y=RA+1 — RB
AFFECT	03		LMC (CTX)	$\overline{TR}$	Continue OR 0=1			R0	2*Q — RB
	04	ATFILE	DPAC — 1 ; ALCAM — 1		JMPR	PACE (0)	R0	R0	Y=RA+RB
	05				Continue	PACE (0)	R0	R0	Y=RA+RB
	06	ATFILE	DPAC — 1 ; ALCAM — 1	VMC	JMPR, AH	PACE (1)			Y=RML(1)
PARAM	07		LMC (CTX)		Continue			R0	2*Q — RB
	08				Continue	PACE (0)	R0	R0	Y=RA+RB
	09		DPAC — 1 ; ALCAM — 1	VMC	Continue, AH	PACE (1)			Y=RML(1)
	0A	SPARAM	LMC (CTX)		JMPR	RADM	SP		Y=RA-1
INCREMENT	0B		LMC (CTX)	$\overline{TR}$	Continue, OR 0-1			R0	2*Q — RB
	0C	ATFILE	DPAC — 1 ; ALCAM — 1		JMPR	PACE (0)	R0	R0	Y=RA+RB — RB
	0D				Continue	PACE (0)	R0	R0	Y=RA+RB — RB
	0E	ATFILE	DPAC — 1 ; ALCAM — 1	VMC	JMPR, AH	PACE (1)			Y=RML (1)
DECREMENT	0F		LMC (CTX)	$\overline{TR}$	Continue OR 0-1			R0	2*Q — RB
	10	ATFILE	DPAC — 1 ; ALCAM — 1		JMPR	PACE (0)	R0	R0	Y=RA+RB — RB
	11				Continue	PACE (0)	R0	R0	Y=RA+RB — RB
	12	ATFILE	DPAC — 1 ; ALCAM — 1	VMC	JMPR AH	PACE (1)			Y=RML(1)
SAUVE	13				Continue				Y="0" — RB
	14				Continue		R0	R0	Y=RA+1 — RB
	15				Continue	RADM	R0	R0	Y=RA+1 — RB
	16	SSAUVE			JMPR	RME (1)			Y=ETAT
//////////	17								
	18								
	19								
	1A								
ENTER	1B		LMC (CTX)		Continue	RADM	SP	SP	Y=RA-1 — RB

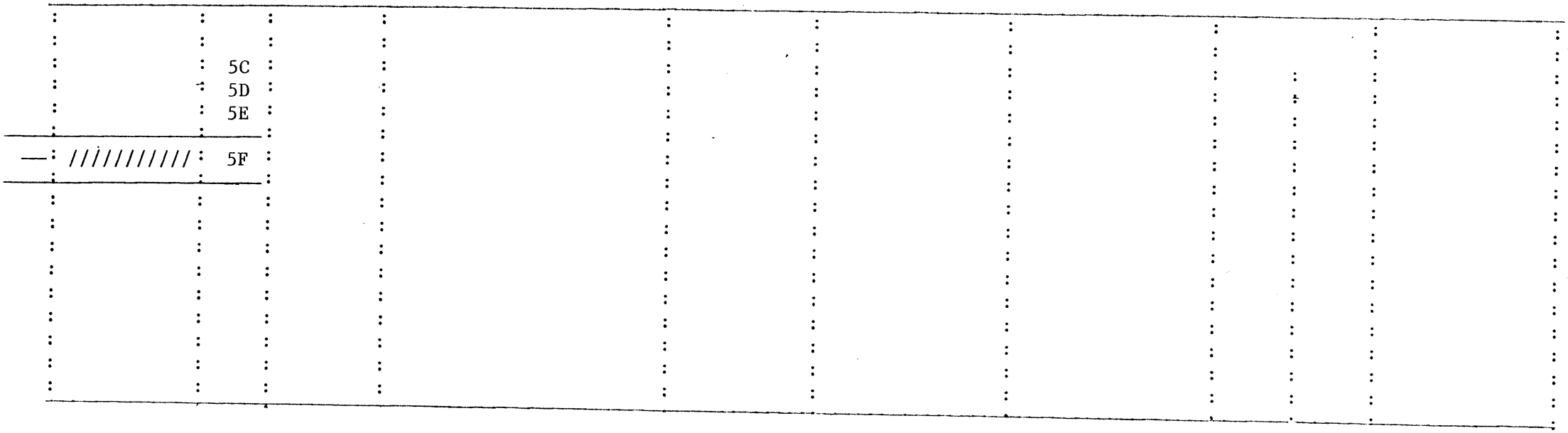
Suite du Tableau.

	1D	ERR	ERR 1 — 1	F = 0	JMPR Cond			Nop
	1E	SENTER	MODE — EXT	BIPAC 8	JMPR		RO	Y=RML(1) — RB
— RET	1F		LMC (CTX)		Continue	RADM		Y=RA-1

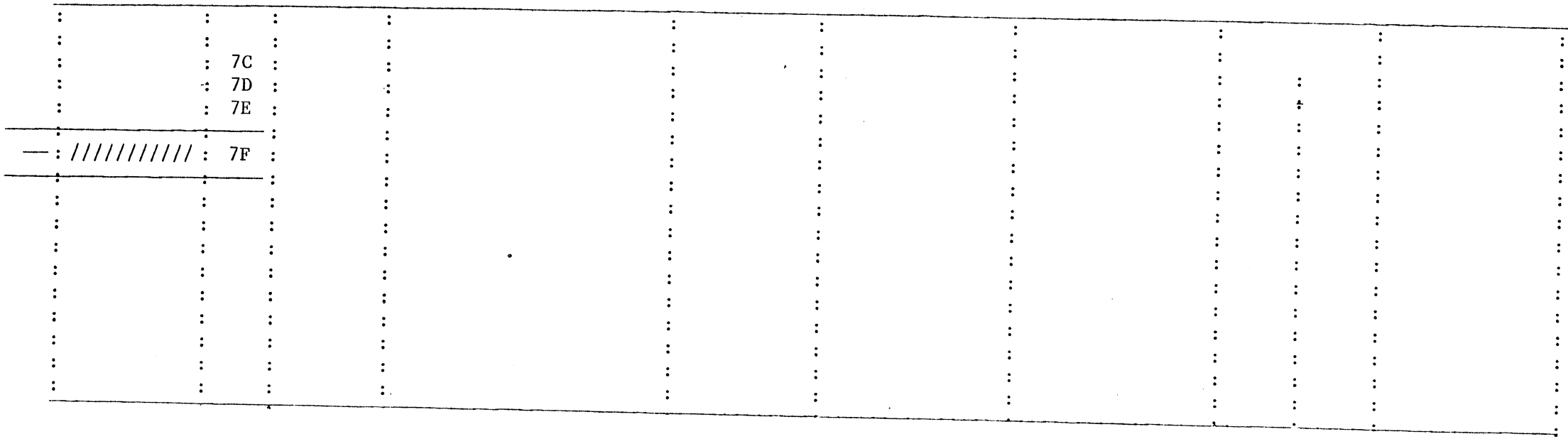
ETIQUETTE	@	BRANCH	ORDRE	ATTENTE' CONDITION	SEQUENCEMENT	DESTINATION Y	REGISTRES RA	REGISTRES RB	CODOP
	20			VMC	Continue AH			W	Y=RML(0) — RB
	21		LMC (CTX)		Continue	RADM	NVC	SP	Y=RA-1 — RB
	22	SRET	Rech (S)	VMC	JMPR AH			NVC	Y=RML(0) — RB
///////////	23								
SPARAM	24			VMC	Continue AH	RME(0)			Y=RML(0)-1
	25	ATFILE	EMC (CTX)		JMPR	RME (1)			Y=RML(1)
	26								
W +/-	27		NEXT BIPAC		Loop		SP	SP	Y=ROM(1)+RA—RB
	28								
	29								
	2A								
CALL	2B		LMC (CTX)		Continue				Nop
	2C		LMC (CDE)	VMC	Continue AH	RADM		R0	Y=RML(1)— RB
	2D		ALLOCAM — "0"		Continue			R1	Y=RML(0) — RB
	2E	TCALL		VMC	JMPR AH				Nop
///////////	2F								
ATFILE	30		ALCAM — "0"	DPAC = 0	Continue AH				Nop
	31		NEXT BIPAC		Loop				Nop
	32								
///////////	33								
TCALL	34	TSEG		D14, D15	JMPR OR 01				Y=RML(1)
	35								
	36								
///////////	37								
	38								
	39								
	3A								
CALLF	3B		LMC (CTX)		Continue				Nop

	3C	TSEG	LMC (CDE) ALLOCAM —— "0"	VMC	Continue AH	RADM		R0	Y=RML(1) —— RB
	3D				Continue				
	3E			VMC	JMPR AH			R1	Y=RML(0) —— RB
//////////	3F								Nop

	ETIQUETTE	@	BRANCH	ORDRE	ATTENTE, CONDITION	SEQUENCEMENT	DESTINATION Y	REGISTRES RA	REGISTRES RB	CODOP
		40								
		41								
		42								
—	NOM	43		LMC (CTX)	$\overline{TR}$	Continue OR 01				Nop
		44	ATFILE	ALCAM — 1, DPAC — 1		JMPR				Nop
		45			VMC	Continue AH	PACE (0)			Y=RML(0)
		46	ATFILE	ALCAM — 1, DPAC — 1		JMPR	PACE (1)			Y=RML(1)
—	INDD	47		LMC (CTX)		Continue			R0	2*Q — RB
		48				Continue	PACE (0)	R0	R0	Y=RA+RB — RB
		49		TROUVE — "0"		Continue				Q/2 — Q
		4A	ATFILE		VMC	JMPR (AH)	PACE (1)			Y=Q RML(1)
—	////////////	4B								
		4C								
		4D								
		4E								
—	////////////	4F								
		50								
		51								
		52								
—	LIT 8	53		TROUVE — "0"		Continue	PACE (1)			Y=VI(1)
		54	ATFILE	DPAC — 1, ALCAM — 1		JMPR	PACE (0)			Y=VI(1)
		55								
		56								
—	////////////	57								
		58								
		59								
		5A								
—	////////////	5B								


5C
5D
5E
5F

ETIQUETTE	@	BRANCH	ORDRE	ATTENTE, CONDITION	SEQUENCEMENT	DESTINATION Y	REGISTRES RA	REGISTRES RB	CODOP
	60								
	61								
	62								
— LIT 8	63		TROUVE — "0"		Continue	PACE (1)			Y=VI(1)
	64	ATFILE	DPAC — 1, ALLOCAM— 1		JMPR	PACE (0)			Y=VI(2)
	65								
	66								
— LIT 16	67		TROUVE — "0"		Continue	PACE (1)			Y=VI(1)
	68		NEXT BIPAC		Continue				Nop
	69	ATLIT	DPAC — 1, ALLOCAM — 1	BIPAC NON VIDE	JMPR AH	PACE (0)			Y=BIPAC
	6A								
— LIT I	6B		TROUVE — "0"		Continue	PACE (1)			Y=VI(2)
	6C		NEXT BIPAC		Continue				Nop
	6D	ATLIT	DPAC — 1, ALLOCAM — 1	BIPAC NON VIDE	JMPR AH	PACE (0)			Y=BIPAC
	6E								
— LIT S	6F		TROUVE — "0"		Continue	PACE (1)			Y=VI(2)
	70		NEXT BIPAC		Continue				Nop
	71	ATLIT	DPAC — 1, ALLOCAM — 1	BIPAC NON VIDE	JMPR AH	PACE (0)			Y=BIPAC
	72								
— LIT S	73		TROUVE — "0"		Continue				Y=VI(1)
	74	ATFILE	DPAC — 1, ALLOCAM — 1		JMPR				Y=VI(1)
	75								
	76								
— //////////	77								
	78								
	79								
	7A								
— //////////	7B								


7C
7D
7E
7F

ETIQUETTE	@	BRANCH	ORDRE	ATTENTE CONDITION	SEQUENCEMENT	DESTINATION Y	REGISTRES RA	REGISTRES RB	CODOP
— DEBUT	80		ALCAM — $\overline{\text{DCAM}}$		Continue		R1	R0	RA — RB
	81		Load Mode	VMC	Continue AH				Y=RML(1)
	82		LMC (CTX)		Continue	RADM		SP	Y=RML(0)— RB
	83		RECH ( )		Continue	RADM	SP	R1	RA — RB
	84			VMC	Continue AH			W	Y=RML(1) — RB
	85		ECR (VIDE)		Continue			R2	Y=RML(0) — RB
	86		LMC (CTX)		Continue	RADM	R1	R1	Y=RA+1 — RB
	87				Deb Loop	NVC		R3	Y=0 — RB
	88				Continue			NVC	Y=RML(1) — RB
	89				Continue	NVC	R3	R3	Y=RA+1 — RB
	8A				Continue			NVC	Y=RML(0) — RB
	8B		LMC (CTX)		Continue	RADM	R1	R1	Y=RA+1 — RB
	8C				Continue	NVC	R3	R3	Y=RA+1 — RB
	8D				Continue		R1	R1	RA-1 — RB
	8E			F ≠ 0	Cond Loop				Nop
	8F				Pop stack	NVC	R2		Y=RA
	90				Deb Loop				Nop
— FETCH	91		RECH (S,D)	BIPAC NON VIDE	JMPD AH	RADM	S*		Y=ROM(0)+RA— 0
	92								
	93								
— SSAUVE	94		EMC (CTX)		Continue	RME (0)	SP		Y=RA
	95			AMC	Continue AH	RADM	SP	SP	Y=RA,RA+1 — RB
	96				Continue	RME (1)	W		Y=RA
	97		EMC (CTX)		Continue	RME (0)			Y=NVC
	98				Continue		R1	R1	Y=RA+1 — RB
	99				Deb Loop	NVC		R2	Y="0" — RB
	9A			AMC	Continue AH	RADM	SP	SP	Y=RA,RA+1 — RB

	9C 9D 9E		EMC (CTX)	F ≠ 0	Continue Continue Cond Loop	NVC RME (0) NVC	R2 NVC R2	R2	R2	Y=RA+1 — RB Y=RA Y=RA+1 — RB
	9F		FINSAUVE	AMC	Pop Stack AH					Nop

	ETIQUETTE	@	BRANCH	ORDRE	ATTENTE, CONDITION	SEQUENCEMENT	DESTINATION Y	REGISTRES RA	REGISTRES RB	CODOP
—	ETOILE	A0	ETOILE			JMPR				Nop
		A1								
		A2								
		A3								
	SENTER	A4		LMC (CTX)		Continue	RADM	SP		Y=RA-1
—		A5				Continue	RME (1)	NVC		Y=RA
		A6		EMC (CTX)	VMC	Continue AH	RME (0)			Y=RML(0)
		A7				Continue				Y=RML(1) — Q
		A8			AMC	Continue AH	RME (1)			Y=NVC
		A9				Continue	NVC	R0		Y=RA
		AA		RECH (S, 2 VIDE)		Continue	RME (0)	NVC		Y=RA
		AB		EMC (CTX)		Continue	RADM	SP	SP	Y=RA;RA+1 — RB
		AC		ECR (VIDE)		Continue		SP	NVC	RA — RB
		AD		LMC (CDE)	AMC	Continue AH	RADM			Y=Q
		AE		ECR (S)	VMC	Continue			R1	Y=RML(1) — RB
		AF	FINENT		F = 0	JMPR Cond			R2	Y=RML(0) — RB
		B0		ALLOCAM — "0"		Continue		SP	R3	RA — RB
		B1	BOUC 1			JMPR		R1	SP	RA+RB — RB
		B2								
		B3								
—	BOUC 1	B4		LMC (CDE)		Continue	RADM			Y=Q+1 — Q
		B5			VMC	Continue AH	RME (1)			Y=RML(1)
		B6	ETI 2		D14, D15	JMPR OR 01	RME (0)			Y=RML(0)
		B7								
—	ETI 2	B8				Continue	RME (0)	SP		Y=RML(0)+RA
		B9		EMC (CTX)		Continue	RADM	R3	R3	Y=RA;RA+1 — RB
		BA				Continue		R1	R1	RA-1 — RB
		BB	BOUC 1		F ≠ 0	JMPR Cond				Nop

	BC BD BE	FINENT		F = 0 VMC	Continue JMPR Cond Continue AH	RME (0)	R2 R2	SP	Y=RA RA+RB —— RB Y= "0"	
	BF				Deb Loop	RME (1)			Y="0"	

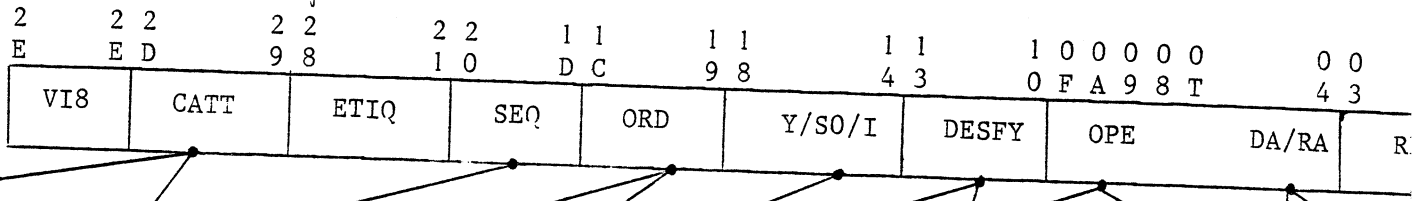
ETIQUETTE	@	BRANCH	ORDRE	ATTENTE / CONDITION	SEQUENCEMENT	DESTINATION Y	REGISTRES RA	REGISTRES RB	CODOP
	C0		EMC (CTX)	AMC	Continue AH	RADM	R3	R3	Y=RA;RA+1 —— RB
	C1				Continue		R2	R2	Y+RA-1 —— RB
	C2	FINENT		F = 0	End Loop Cond				Nop
	C3								
FINENT	C4		ALLOCAM —— $\overline{\text{DCAM}}$	VMC	Continue AH	RME (0)	W		Y=RA
	C5		EMC (CTX)		Continue	RADM	SP	SP	Y=RA,RA+1 —— RB
	C6		NEXT BIPAC		Loop		SP	W	RA —— RB
	C7								
SCALL	C8		LMC (CDE)		Continue	RADM			Y=RML(0) —— Q
	C9				Continue		SP	R2	RA —— RB
	CA				Continue AH			R3	Y=RML(0) —— RB
	CB	NP		F = 0	JMPR Cond		R3	SP	RA+RB —— RB
	CC				Continue		R2	R4	Y=RA —— RB
	CD				Continue	RME (0)			Y="0"
	CE		EMC (CTX)		Deb Loop	RME (1)			Y="0"
	CF				Continue		R3	R3	RA-1 —— RB
	D0	NP		F = 0	End Loop Cond				Nop
	D1			AMC	Loop AH	RADM	R4	R4	RA+1 —— RB
	D2								
	D3								
NP	D4				Continue			R3	Y=RML(1) —— RB
	D5	MARK		F = 0	JMPR Cond		R3	SP	RA+RB —— RB
	D6				Continue		R3	R4	RA —— RB
	D7				Deb Loop		SP	R5	RA —— RB
	D8		LMC (CDE)		Continue	RADM			Y=Q+1 —— Q
	D9			VMC	Continue AH	RME (0)			Y=RML(0)
	DA	ETI 1		D14, D15	JMPR	RME (1)			Y=RML(1)
	DB								

ETI 1	DC		EMC (CTX)		Continue	RME (0)	R2		Y=RML(0)+RA
	DD				Continue	RADM	R5	R5	RA-1 — RB
	DE				Continue		R4	R4	RA-1 — RB
	DF			F = 0	Loop Cond				

ETIQUETTE	@	BRANCH	ORDRE	ATTENTE, CONDITION	SEQUENCEMENT	DESTINATION Y	REGISTRES RA	REGISTRES RB	CODOP
	E0			AMC	Continue AH	RADM	SP	SP	Y=RA;RA+1 — RB
	E1				Pop Stack	RME (1)			Y=Q+1
	E2		EMC (CTX		Continue	RME (0)	R3		Y=RA
	E3			AMC	Continue AH	RADM	SP	SP	Y=RA;RA+1 — RB
	E4				Continue	RME (1)	R1		Y=RA
	E5		EMC (CTX)		Continue	RME (0)	R4		Y=RA
	E6		ALLOCAM — DCAM		RTS				Nop
	E7								
	E8								
	E9								
	EA								
	EB								
TSEG	EC	ERR	DPINS — "0"		JMPR	ADCODE	R0		Y=RA
	ED	SCALL	DPINS — "0"		JSR	ADCODE			Y=RML(1)
	EE		NEXT BIPAC		Loop				Nop
	EF		LMC (EXT		Continue	RADM			Y=RML(0)
	F0		DPINS — "0"	VMC	Continue AH	ADCODE			Y=RML(1)
	F1	SCALL	MODE — EXT		JSR				Nop
	F2		MODE — INT		Continue				Nop
	F3		NEXT BIPAC		Loop				Nop
	F4		ECR (VIDE)		Continue				Nop
	F5		LMC (CTX)		Continue	NVC			Y=RML(1)
	F6		MODE — INIT	BIPAC 8	Continue	RADM	SP		Y=RA-1
	F7			VMC	Continue AH		NVC		Y=RML(1) — RB
	F8		NEXT BIPAC		Loop			SP	Y=RML(0) — RB
	F9								
	FA								
	FB								

	FC FD FE										
	FF										

POP : Format et codage des microinstructions



00 FINI
01 VIDE
02 PLEIN
03 NOCL3
04 L64
05
06
07 VMC,TX
08 0,0
09 PLEIN,OCC
0A RI1,RI0
0B E3,L3
0C TB2,TB2
0D TA1,TA0
0E
0F
10 AMC
11 VMC
12 DPOP
13 BIPOP
14 A≥B NS
15 A≥B
16 A<B NS
17 A<B
18 = 0
19 ≠ 0
1A < 0
1B ≥ 0
1C OVF
1D OVF
1E SI0 0..7
1F SI0 8..F

00 JZ
01 CJS
02 JMAP
03 CJP
04 PUSH
05 JSRP
06 CJV
07 JSR
08 RFCT
09 RPCT
0A CRTN
0B CJPP
0C LDCT
0D T LOUP
0E CONT
0F TWB

00 Noop
01 LMC
02 EMC-B
03 EMC-HW
04 EMC-W
05 RAZ-MSK
06 DPOP
07 NEXT-BI
08 NEXTOP
09 K+1
0A N+1
0B K+1 et N+1
0C K-1
0D N-1
0E K-1 et N-1

0F →

DA = ↓

00 MODE ← 0
01 MODE ← 1
02 TROMPE
03 BON
04 ERR-POP
05 INIT-PILE
06 A9 ← $\overline{A9}$
07
08
09 Load ETAT
0A Load ADS
0B FINSQUV
0C RLD(2910)
0D
0E
0F

30 RA±Q → RB
31 RA±RB → RB
32 RA±RB+C → RB
33 RA±1 → RB
34 RB±RA → RB
35 RA±1+C → RB
36 RA-RB/RB-RA
37 DB±1 → RB
38 DB-RA/RA-DB
39 RA/RA+RB
3A DA/DA+RB
3B ASR(RA/RB)
3C LSR(RA/RB)
3D
3E
3F

00 RADM 00 W
01 RADM 01 W
02 POPE 0
03 RADM 1X W
04 RADM 00 VC
05 PILE 0
06 RME 0
07 RME 0 L→H
08 RADM00 B,P.,E
09 RADM01 B,E
0A RME 0 etPOPE0
0B RMEj
0C RADM00,B,L
0D RADM 00,B,L
0E RADM(Z),B,L
0F Noop

00 POPL2
01 RML 0
02 PILE 0
03 RML 1
04 PILE 1
05 PF1, PT1
06 PF2, PT2
07 POPL 1
08 RML0 et RML1 → PILE1
09 RML0 et RML1 → PF2
0A PF2,PT2, et VI→PF2
0B POPL0 et POPL1→PF2
0C PF1 → POPE 1
0D PF2 → POPE 1
0E Z → RME 1
0F Y → PILE 1
10 PILE0 et PILE1→RME1
11 PILE0 et PILE1→POPE1
12 POPL0 et POPL4→PF1
13 POPL0 / RML 0
14 PILE0 / RML 0
15 PILE1 / RML 1
16 RMLi → RMLj
17 RML 0 / RML 1
18 PF1,PT1 →PF2,PT2
19 ETRT → RME 1
1A VI(0.3) → PF2-L
1B VI(4..7) → PF2-H
1C PF1 → RPF et POPL 0
1D PF2 → RPF
1E
1F

00 RA → RB
01 RA+1 → RB
02 RA+C → RB
03 -RA → RB
04 RA-1 → RB
05 $\overline{RA}$ → RB
06 RA+C → RB
07 DA → RB
08 0 → RB
09 S10 0 → RB
0A DA+1 → RB
0B DA+RB → RB
0C DA-RB → RB
0D DA-RB+C → RB
0E DA+RB+C → RB
0F DA+RB+I → RB
10 -DA+RB+I → RB
11 RQ+RB → RB
12 RA+RB+C → RB
13 -RA+RB → RB
14 RA-RB → RB
15 RA-RB+C → RB
16 Q → RB
17 RA+Q → RB
18 RA → Q
19 0 → Q
1A DA → Q
1B -DA+RB → Q
1C DA+RB → Q
1C DA+RB → Q
1D Q-1
1E Q+1
1F DA-RB
20 -DA+RB
21 -DA+DB
22 DA-DB
23 DA-1
24 DA
25 DB-RA
26 RA
27 RA
28 RA-1
29 -RA-1
2A 0
2B RA-RB
2C DB
2D DB → RB
2E RB
2F RB+1 → RB

00 H,L
01 0,L
02 S,L
03 0,H
04 S,H
05 0,VI
06 S, VI
07 0,BI
08 S,BI
09 0,N
0A H,O
0B 8/16 I
0C 8M6
0D 8 6 NS
0E
0F

00 SP
01 BT
02 BC
03 R2 0
04 R2 1
05 R2 2
06 R2 3
07 AL
08 AE
09 R1 1
0A R1 1
0B RA1
0C RA2
0D
0E
0F

BIBLIOGRAPHIE

[1] J.P.SCHOELLKOPF
"A top-down approach of design and functional description of hardware"
ACM German Chapter, IEEE German section, Darmstadt, 1..8.74.

[2] J.P.SCHOELLKOPF
"Microprogramming: a step of a top-down design methodology"
7th Annual workshop on microprogramming, MICRO 7, SIGMICRO, Palo-Alto, USA, 1.10.74.

[3] F.ANCEAU
"Contribution à l'étude des systèmes hiérarchisés de ressources dans l'architecture des machines informatiques"
Thèse de doctorat d'Etat, Grenoble, 5.12.74.

[4] GREBERT
"Space programming language machine architecture study"
SAMSO TR 72, May 1972.

[5] R.FORTIER
"Conception descendante de machine informatique - Etude et définition d'un langage intermédiaire et d'une machine formelle multiprocesseur orientée vers les exécutions du langage PASCAL"
Thèse de 3eme cycle, Grenoble, octobre 1975.

[6] Basic principles of the B6500
(Burroughs corporation)

[7] CRE ECH
"Architecture of the B6500"
COINS 69 - 3rd Int. Symp. on computer and information science, Miami, USA, 1969.

[8] J.P.SCHOELLKOPF
"Machine PASCHLL: Définition d'une architecture pipe-line pour une unité centrale adaptée au langage PASCAL"
Thèse de 3eme cycle, Grenoble, juin 1977.

[9] F.ANCEAU
"Some design problems relating to system oriented computer architecture"
Symp. on logical organization of computers, Varsovie, Septembre 1975.

[10] G.BERGER-SABBATEL
"Etude et évolution sur maquette de l'implantation hardware d'algorithmes de gestion de fichiers, adaptés aux petits systèmes"
Contrat SESORI, n.74 1366, juillet 1975.

[11] G.H.POUJOULAT
"The CORAIL building block system"
EUROMICRO, Newsletter, October 76.

[12] G.BAILLE, J.P.SCHOELLKOPF
"Evaluation d'une expression post-fixée sur une file d'attente et réalisation d'une machine pipe-line de haut niveau"
Séminaire de programmation, ENSIMAG, Grenoble, juin 1975.

[13] G.BAILLE, J.P.SCHOELLKOPF
"Evaluation of polish form expression on a FI FO queue: a new approach towards the realization of a high level pipe line computer"
SAGAMORE Computer conf., Syracuse University.

[14] F.ANCEAU, G.BAILLE, J.P.SCHOELLKOPF
"Machine informatique électronique destinée à

l'exécution parallèle d'un langage post-fixé"
Brevet ANVAR, n.75 29 473, déposé le 26.9.75.

[15] G.BAILLE, J.P.SCHOELLKOPF
"A pipe-line polish string computer"
AFIPS 1976, National Comp.Conf., NCC, New York,
juin 1976.

AUTORISATION DE SOUTENANCE

VU les dispositions de l'article 6 du règlement de l'Etablissement, approuvé par le Conseil Scientifique du 23 février 1976,

VU les rapports de présentation de Messieurs

. F. ANCEAU, Professeur

. V. CORDONNIER, Professeur

Monsieur Gérard BAILLE

est autorisé à présenter une thèse en soutenance pour l'obtention du titre de DOCTEUR DE L'I.N.P.-G, spécialité "Informatique".

Fait à Grenoble, le 12 octobre 1983

Le Président de l'I.N.P.-G

D. BLOCH
Président
de l'Institut National Polytechnique
de Grenoble

P.O. le Vice-Président,

RESUME :

PASC-HLL est une Unité Centrale d'ordinateur adaptée à l'exécution du langage PASCAL. Cette réalisation illustre une méthodologie de conception descendante qui, à partir d'un cahier des charges (but) et en fonction des matériels existants (moyens) permet de concevoir la machine étape par étape depuis le langage PASCAL jusqu'à la réalisation matérielle. Cette méthodologie conduit à des processeurs très spécialisés et donc optimaux.

PASC-HLL est constituée de cinq processeurs indépendants travaillant en parallèle. Ils sont tous microprogrammés et réalisés avec des microprocesseurs en tranches.

MOTS CLEFS

Architecture des ordinateurs - Processeurs fonctionnels - Conception descendante - Architecture pipe-line - Machine langage - Langage PASCAL - Microprogrammation - Microprocesseurs en tranches