Etude d'une structure de multiprocesseur virtuel : application à un réseau d'opérateurs adapté au traitement du signal

Enrique Mitrani Abenchuchan

THESE

Présentée à

L'UNIVERSITE SCIENTIFIQUE ET MEDICALE

DE GRENOBLE

pour obtenir le grade de

DOCTEUR INGENIEUR

PAR

Enrique MITRANI ABENCHUCHAN

INGENIEUR IAG

ETUDE D'UNE STRUCTURE DE MULTIPROCESSEUR VIRTUEL. APPLICATION A UN RESEAU D'OPERATEURS ADAPTE AU TRAITEMENT DU SIGNAL.

Soutenue le 27 Janvier 1975, devant la Commission d'Examen

Monsieur	L. BOLLIET	Président
Messieurs	F. ANCEAU S. KRAKOWIAK	Examinateurs
Messieurs	G. NOGUEZ R. GARIOD	Invités

UNIVERSITE SCIENTIFIQUE
ET MEDICALE DE GRENOBLE

INSTITUT NATIONAL POLY-
TECHNIQUE DE GRENOBLE

M. Michel SOUTIF
M. Gabriel CAU

Présidents M. Louis NEEL
Vice-Présidents MM. Lucien BONNETAIN
Jean BENOIT

MEMBRES DU CORPS ENSEIGNANT DE L'U.S.M.G.

PROFESSEURS TITULAIRES

MM.	ANGLES D'AURIAC Paul	Mécanique des fluides
	ARNAUD Georges	Clinique des maladies infectieuses
	ARNAUD Paul	Chimie
	AUBERT Guy	Physique
	AYANT Yves	Physique approfondie
Mme	BARBIER Marie-Jeanne	Electrochimie
MM.	BARBIER Jean-Claude	Physique expérimentale
	BARBIER Reynold	Géologie appliquée
	BARJON Robert	Physique nucléaire
	BARNOUD Fernand	Biosynthèse de la cellulose
	BARRA Jean-René	Statistiques
	BARRIE Joseph	Clinique chirurgicale
	BEAUDOING André	Pédiatrie
	BERNARD Alain	Mathématiques Pures
Mme	BERTRANDIAS Françoise	Mathématiques Pures
MM.	BEZES Henri	Chirurgie générale
	BLAMBERT Maurice	Mathématiques Pures
	BOLLIET Louis	Informatique (IUT B)
	BONNET Georges	Electrotechnique
	BONNET Jean-Louis	Clinique ophtalmologique
	BONNET-EYMARD Joseph	Pathologie médicale
	BOUCHERLE André	Chimie et Toxicologie
	BOUCHEZ Robert	Physique nucléaire
	BOUSSARD Jean-Claude	Mathématiques Appliquées
	BRAVARD Yves	Géographie
	CABANEL Guy	Clinique rhumatologique et hydrologie
	CALAS François	Anatomie
	CARRAZ Gilbert	Biologie animale et pharmacodynamie
	CAU Gabriel	Médecine légale et Toxicologie
	CAUQUIS Georges	Chimie organique
	CHABAUTY Claude	Mathématiques Pures
	CHARACHON Robert	Oto-Rhino-Laryngologie
	CHATEAU Robert	Thérapeutique
	CHIBON Pierre	Biologie animale
	COEUR André	Pharmacie chimique et chimie analytique
	CONTAMIN Robert	Clinique gynécologique
	COUDERC Pierre	Anatomie Pathologique
	CRAYA Antoine	Mécanique
Mme	DEBELMAS Anne-Marie	Matière médicale
MM.	DEBELMAS Jacques	Géologie générale
	DEGRANGE Charles	Zoologie
	DEPORTES Charles	Chimie minérale
	DESRE Pierre	Métallurgie
	DESSAUX Georges	Physiologie animale
	DODU Jacques	Mécanique appliquée
	DOLIQUE Jean-Michel	Physique des plasmas
	DREYFUS Bernard	Thermodynamique
	DUCROS Pierre	Cristallographie
	DUGOIS Pierre	Clinique de Dermatologie et Syphiligraphie
	FAU René	Clinique neuro-psychiatrique

MM.	GAGNAIRE Didier	Chimie physique
	GALLISSOT François	Mathématiques Pures
	GALVANI Octave	Mathématiques Pures
	GASTINEL Noël	Analyse numérique
	GAVEND Michel	Pharmacologie
	GEINDRE Michel	Electroradiologie
	GERBER Robert	Mathématiques Pures
	GERMAIN Jean-Pierre	Mécanique
	GIRAUD Pierre	Géologie
	KAHANE André	Physique générale
	KLEIN Joseph	Mathématiques Pures
	KOSZUL Jean-Louis	Mathématiques Pures
	KRAVTCHENKO Julien	Mécanique
	KUNTZMANN Jean	Mathématiques Appliquées
	LACAZE Albert	Thermodynamique
	LACHARME Jean	Biologie végétale
	LAJZEROWICZ Joseph	Physique
	LATREILLE René	Chirurgie générale
	LATURAZE Jean	Biochimie pharmaceutique
	LAURENT Pierre	Mathématiques Appliquées
	LEDRU Jean	Clinique médicale B
	LLIBOUTRY Louis	Géophysique
	LONGEQUEUE Jean-Pierre	Physique nucléaire
	LOUP Jean	Géographie
Mle	LUTZ Elisabeth	Mathématiques Pures
	MALGRANGE Bernard	Mathématiques Pures
	MALINAS Yves	Clinique obstétricale
	MARTIN-NOEL Pierre	Seméiologie médicale
	MAZARE Yves	Clinique médicale A
	MICHEL Robert	Minéralogie et Pétrographie
	MOURIQUAND Claude	Histologie
	MOUSSA André	Chimie nucléaire
	NEEL Louis	Physique du Solide
	OZENDA Paul	Botanique
	PAYAN Jean-Jacques	Mathématiques Pures
	PEBAY-PEYROULA Jean-Claude	Physique
	RASSAT André	Chimie systématique
	RENARD Michel	Thermodynamique
	REULOS René	Physique Industrielle
	RINALDI Renaud	Physique
	ROGET Jean	Clinique de pédiatrie et de puériculture
	DE ROUGEMONT Jacques	Neuro-chirurgie
	SEIGNEURIN Raymond	Microbiologie et Hygiène
	SENGEL Philippe	Zoologie
	SOUTIF Michel	Physique générale
	TANCHE Maurice	Physiologie
	TRAYNARD Philippe	Chimie générale
	VAILLANT François	Zoologie
	VALENTIN Jacques	Physique Nucléaire
	VAUQUOIS Bernard	Calcul électronique
Mme	VERAIN Alice	Pharmacie galénique
M.	VERAIN André	Physique
MM.	VEYRET Paul	Géographie
	VIGNAIS Pierre	Biochimie médicale
	YOCCOZ Jean	Physique nucléaire théorique

PROFESSEURS ASSOCIES

MM.	ASCARELLI Gianni	Physique
	CHEEKE John	Thermodynamique
	GILLESPIE John	I.S.N.
	ROCKAFELLAR Ralph	Mathématiques appliquées
	WOHLFARTH Erich	Physique du solide

PROFESSEURS SANS CHAIRE

Mle	AGNIUS-DELORD Claudine	Physique pharmaceutique
	ALARY Josette	Chimie analytique
MM.	BELORIZKY Elie	Physique
	BENZAKEN Claude	Mathématiques appliquées
	BERTRANDIAS Jean-Paul	Mathématiques appliquées
	BIAREZ Jean-Pierre	Mécanique
Mme	BONNIER Jane	Chimie générale
MM.	BRUGEL Lucien	Energétique
	CARLIER Georges	Biologie végétale
	CONTE René	Physique
	DEPASSEL Roger	Mécanique des Fluides
	GAUTHIER Yves	Sciences biologiques
	GAUTRON René	Chimie
	GIDON Paul	Géologie et Minéralogie
	GLENAT René	Chimie organique
	HACQUES Gérard	Calcul numérique
	HOLLARD Daniel	Hématologie
	HUGONOT Robert	Hygiène et Méd.Préventive
	IDELMAN Simon	Physiologie animale
	JANIN Bernard	Géographie
	JOLY Jean-René	Mathématiques pures
	JULLIEN Pierre	Mathématiques appliquées
Mme	KAHANE Josette	Physique
MM.	KUHN Gérard	Physique
	LUU-DUC-Cuong	Chimie Organique
	MAYNARD Roger	Physique du solide
	MULLER Jean-Michel	Thérapeutique
	PERRIAUX Jean-Jacques	Géologie et minéralogie
	PFISTER Jean-Claude	Physique du solide
Mle	PIERY Yvette	Physiologie animale
MM.	REBECQ Jacques	Biologie (CUS)
	REVOL Michel	Urologie
	REYMOND Jean-Charles	Chirurgie générale
	ROBERT André	Chimie papetière
	SARRAZIN Roger	Anatomie et chirurgie
	SARROT-REYNAULD Jean	Géologie
	SIBILLE Robert	Construction Mécanique
	SIROT Louis	Chirurgie générale
Mme	SOUTIF Jeanne	Physique générale
MM.	VIALON Pierre	Géologie
	VAN CUTSEM Bernard	Mathématiques appliquées

MAITRES DE CONFERENCES ET MAITRES DE CONFERENCES AGREGES

MM.	AMBLARD Pierre	Dermatologie
	AMBROISE-THOMAS Pierre	Parasitologie
	ARMAND Yves	Chimie
	BEGUIN Claude	Chimie organique
Mme	BERIEL Héléne	Pharmacodynamique
M.	BILLET Jean	Géographie
	BOUCHARLAT Jacques	Psychiatrie adultes
Mme	BOUCHE Liane	Mathématiques (CUS)
MM.	BOUCHET Yves	Anatomie
	BRODEAU François	Mathématiques (IUT B)
	BUISSON Roger	Physique
	BUTEL Jean	Orthopédie
	CHAMBAZ Edmond	Biochimie médicale
	CHAMPETIER Jean	Anatomie et organogénèse
	CHERADAME Hervé	Chimie papetière
	CHIAVERINA Jean	Biologie appliquée (EFP)

MM.	COHEN-ADDAD Jean-Pierre	Spectrométrie physique
	COLOMB Maurice	Biochimie médicale
	COULOMB Max	Radiologie
	CROUZET Guy	Radiologie
	CYROT Michel	Physique du solide
	DELOBEL Claude	M.I.A.G.
	DUSSAUD René	Mathématiques (CUS)
Mme	ETERRADOSSI Jacqueline	Physiologie
MM.	FAURE Jacques	Médecine légale
	FONTAINE Jean-Marc	Mathématiques Pures
	GENSAC Pierre	Botanique
	GIDON Maurice	Géologie
	GRIFFITHS Michaël	Mathématiques Appliquées
	GROS Yves	Physique (stag.)
	GROULADE Joseph	Biochimie médicale
	GUITTON Jacques	Chimie
	IVANES Marcel	Electricité
	JALBERT Pierre	Histologie
	KRAKOWIAK Sacha	Mathématiques appliquées
Mme	LAJZEROWICZ Jeannine	Physique
MM.	LEROY Philippe	Mathématiques
	LOISEAUX Jean-Marie	Physique Nucléaire
	MACHE Régis	Physiologie végétale
	MAGNIN Robert	Hygiène et Médecine préventive
	MARECHAL Jean	Mécanique
	MARTIN-BOUYER Michel	Chimie (CUS)
	MICHOULIER Jean	Physique (I.U.T. "A")
Mme	MINIER Colette	Physique
MM.	MICOUD Max	Maladies infectieuses
	NEGRE Robert	Mécanique
	PARAMELLE Bernard	Pneumologie
	PECCOUD François	Analyse (IUT B)
	PEFFEN René	Métallurgie
	PELMONT Jean	Physiologie animale
	PERRET Jean	Neurologie
	PHELIP Xavier	Rhumatologie
	RACHAIL Michel	Médecine interne
	RACINET Claude	Gynécologie et obstétrique
	RAYNAUD Hervé	M.I.A.G.
	RENAUD Maurice	Chimie
Mme	RENAUDET Jacqueline	Bactériologie
M.	RICHARD Lucien	Botanique
Mme	RINAUDO Marguerite	Chimie macromoléculaire
MM.	ROMIER Guy	Mathématiques (IUT B)
	SHOM Jean Claude	Chimie Générale
	STIEGLITZ Paul	Anesthésiologie
	STOEBNER Pierre	Anatomie pathologique
	VROUSOS Constantin	Radiologie

MAITRES DE CONFERENCES ASSOCIES

MM.	CRABBEE Pierre	C.E.R.M.O.
	CABOT	Mathématiques appliquées
	CURRIE Jan	Mathématiques appliquées

CHARGES DE FONCTIONS DE MAITRES DE CONFERENCES

MM.	BARGE Michel	Neuro-chirurgie
	CONTAMIN Charles	Chirurgie thoracique et cardio-vasculaire
	CORDONNIER Daniel	Néphrologie
	DENIS Bernard	Cardiologie
	KOLODIE Lucien	Hématologie
	RAMBAUD Pierre	Pédiatrie
	ROCHAT Jacques	Hygiène et hydrologie

MEMBRES DU CORPS ENSEIGNANT DE L'I.N.P.G.

PROFESSEURS TITULAIRES

MM.	BENOIT Jean	Radioélectricité
	BESSON Jean	Electrochimie
	BONNETAIN Lucien	Chimie Minérale
	BONNIER Etienne	Electrochimie, Electrométallurgie
	BRISSONNEAU Pierre	Physique du solide
	BUYLE-BODIN Maurice	Electronique
	COUMES André	Radioélectricité
	FELICI Noël	Electrostatique
	PAUTHENET René	Physique du solide
	PERRET René	Servomécanismes
	SANTON Lucien	Mécanique
	SILBER Robert	Mécanique des Fluides

PROFESSEUR ASSOCIE

M.	BOUDOURIS Georges	Radioélectricité

PROFESSEURS SANS CHAIRE

MM.	BLIMAN Samuel	Electronique
	BLOCH Daniel	Physique du solide et Cristallographie
	COHEN Joseph	Electrotechnique
	DURAND Francis	Métallurgie
	MOREAU René	Mécanique
	POLOUJADOFF Michel	Electrotechnique
	VEILLON Gérard	Informatique fondamentale et appliquée
	ZADWORNY François	Electronique

MAITRES DE CONFERENCES

MM.	BOUVARD Maurice	Génie mécanique
	CHARTIER Germain	Electronique
	FOULARD Claude	Automatique
	GUYOT Pierre	Chimie minérale
	JOUBERT Jean Claude	Physique du solide
	LACOUME Jean Louis	Géophysique
	LANCIA Roland	Physique atomique
	LESPINARD Georges	Mécanique
	MORET Roger	Electrotechnique nucléaire
	ROBERT François	Analyse numérique
	SABONNADIERE Jean Claude	Informatique fondamentale et appliquée
Mme	SAUCIER Gabrièle	Informatique fondamentale et appliquée

MAITRE DE CONFERENCES ASSOCIE

M.	LANDAU Ioan Doré	Automatique

CHARGE DE FONCTIONS DE MAITRES DE CONFERENCES

M.	ANCEAU François	Mathématiques appliquées

Fait à St Martin d'Hères JANVIER 1974

Monsieur F. ANCEAU, Chargé d'Enseignement à l'Ecole Nationale Supérieure d'Informatique et de Mathématiques Appliquées de Grenoble.

Monsieur L. BOLLIET, Professeur à l'Université Scientifique et Médicale de Grenoble

Monsieur R. GARIOD, Chef du Laboratoire Mesure, Contrôle et Traitement Electronique du Laboratoire d'Electronique et de Technologie de l'Informatique du CEN/Grenoble

Monsieur S. KRAKOWIAK, Maître de Conférences à l'Université Scientifique et Médicale de Grenoble

Monsieur J. MONGE, Chef du Groupe Mesures du Laboratoire LETI/MCTE

Monsieur G. NOGUEZ, Chargé d'Enseignement à l'Institut de Programmation de Paris

Monsieur J. THUEL

trouveront ici l'expression de mes remerciements.

- TABLE DE MATIERES -

Pages

DEUXIEME PARTIE

- P R E S E N T A T I O N -

Ce travail, développé au sein du Laboratoire d'Electronique et Technologie de l'Informatique (LETI) du C.E.N.G. a bénéficié des conseils de l'Institut de Mathématiques Appliquées de Grenoble (I.M.A.G.). Notre point d départ fut la définition d'un réseau d'opérateurs adapté au traitement du signal et, plus généralement, aux algorithmes impliquant des traitements trè répétitifs portant sur des grands blocs de données. Progressivement, nous sommes arrivés à la conclusion qu'un système multiprocesseur serait une bonn solution à notre problème, à condition d'aboutir à une réalisation assez simple et économique. C'est ainsi que nous avons été conduits à la notion de multiprocesseurs "pipe-line" exposée dans la première partie de ce mémoire.

Le réseau adapté au traitement du signal, décrit dans la deuxième partie, est présenté comme un exemple précis d'application de ce type de structures. En effet, nous pensons que l'utilisation de multiprocesseurs "pipe-line" ne se limite pas à notre cas d'application mais peut s'étendre avantageusement à un grand nombre de problèmes où un multiprocesseur est envisagé.

Une mise en oeuvre adéquate de cette structure exige une coopératic harmonieuse entre les processeurs qui la composent. Nous avons étudié des mécanismes de synchronisation permettant d'obtenir cette coopération. Ces systèmes sont inspirés des sémaphores et primitives de changement d'état P et V [12], mais présentent des différences, plus ou moins grandes, pour les adapter à des problèmes précis. C'est ainsi que nous avons développé deux primitives de changement d'état pour la synchronisation du réseau d'opérateur adapté au traitement du signal (c.f. III.6 et IV.4).

Enfin, le fonctionnement logique d'une partie de ce réseau d'opérateurs, a été vérifié grâce à une simulation en langage CASSANDRE [16].

PREMIERE PARTIE

- C H A P I T R E I -

P R E L I M I N A I R E S

1. SYSTEMES MULTINIVEAUX

Une introduction - même très sommaire - aux systèmes multiniveaux [peut clarifier certains concepts utilisés dans la suite de ce mémoire.

Pour fournir une image d'un tel système, considérons le cas d'un calculateur contrôlé par microprogramme. A chaque travail décrit en langage machine, le calculateur fait appel au processeur pour l'exécution d'une séquence précise d'instructions. A chaque instruction le processeur fait appe au microprocesseur pour l'exécution d'un microprogramme bien déterminé.

Cet exemple nous montre l'existence de plusieurs niveaux d'exécutic Le diagramme - très simplifié - d'un calculateur microprogrammé représenté pa la figure I.1. met en évidence ces niveaux. D'une manière générale on peut di pour ce type de systèmes, que chacun des niveaux doit fournir tous les outils nécessaires au travail du niveau immédiatement supérieur.

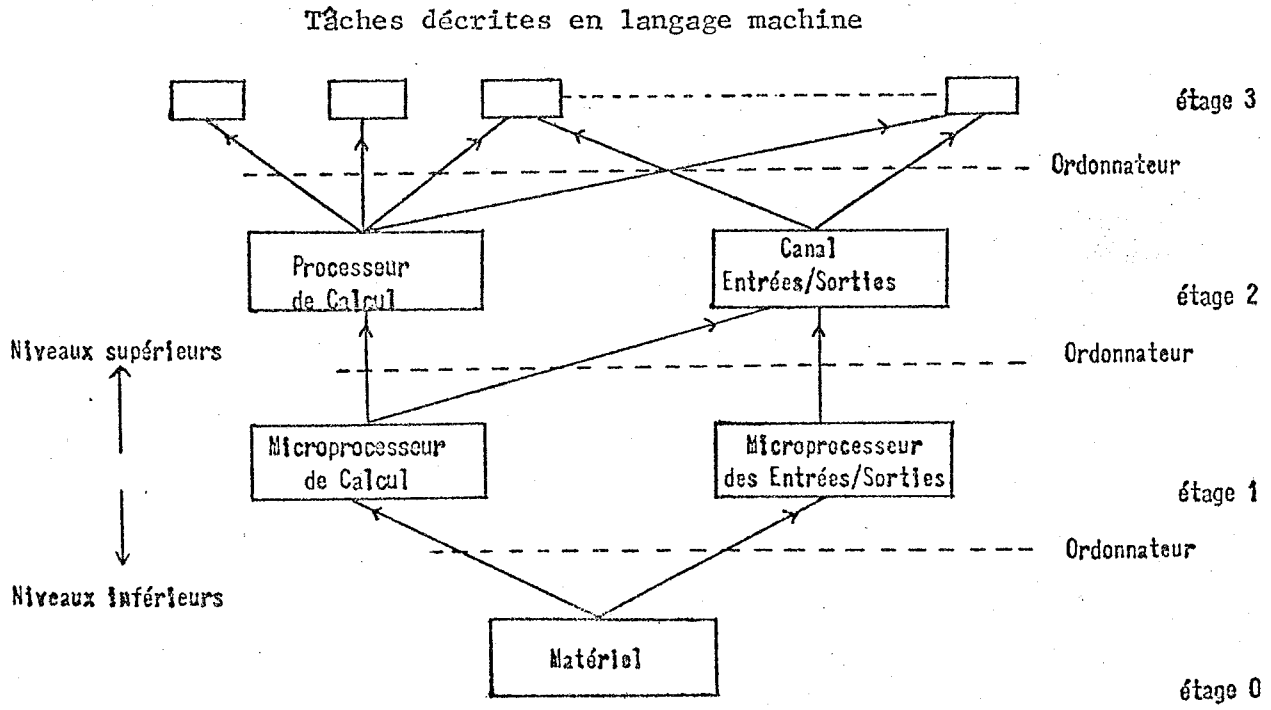


Figure I.1. : DIAGRAMME SIMPLIFIE D'UN CALCULATEUR MICROPROGRAMME

Le travail à effectuer à chacun de ces niveaux est décomposé en une ou plusieurs tâches - processus séquentiels dont les actions sont décrites par un ou plusieurs programmes (sur microprogrammes ou macroprogrammes, etc) -. Toutes les actions sollicitées par une tâche sont nécessairement effectuées par des processeurs. D'une manière générale on peut penser que chacun des niveaux de ces systèmes constitue un ensemble de tâches à exécuter par le niveau immédiatement inférieur, en même temps qu'il est considéré comme un processeur par le niveau immédiatement supérieur. Dans notre exemple, le processeur de calcul est le processeur pour les différentes tâches décrites en langage machine, en même temps qu'il est considéré comme une tâche par le microprocesseur.

A chaque niveau du système on rencontre un ordonnateur, c'est-à-dire un organe chargé de l'enchaînement correct des différentes tâches dans les processeurs, en fonction de critères préétablis [9].

Il est utile de remarquer qu'un processeur peut être composé de sous-processeurs, soit équivalents - chacun d'entre eux pouvant réaliser la fonction processeur -, soit complémentaires - ils concourent tous à l'exécution d'une même tâche - [2].

2. STRUCTURES EN "PIPE-LINE"

Il est normal et presque indispensable de découper l'exécution d'une instruction* en plusieurs phases. Pour mener à terme chacune de ces phases, on dispose d'un ensemble de moyens de calcul, ensemble que nous appelerons étage. Il est possible de trouver un découpage tel que pour effectuer chacune des phases d'une instruction, on n'ait besoin que des résultats fournis par celle qui la précède. Dans cette hypothèse, on peut rendre indépendants les étages associés à ces phases à l'aide de registres temporaires pour la sauvegarde des résultats intermédiaires. A chaque instant l'un de ces étages est affecté à l'exécution d'une phase d'une instruction. En même temps, les autres étages peuvent être utilisés pour l'exécution des phases différentes d'autres instructions du même programme.

---

* Pour éviter d'alourdir inutilement le texte, dans le reste de ce chapitre, le mot instruction désigne indifféremment microinstruction, instruction, etc. De même le mot processeur désigne microprocesseur, processeur, etc.

Ce mode de fonctionnement, connu sous le nom de "pipe-line", peut être illustré par une chaîne de montage dont le premier étage détermine les instructions à exécuter, les étages successifs se chargeant de l'exécution des différentes phases de chacune d'entre elles au fur et à mesure de son passage dans la chaîne. Un avantage évident de cette technique est la possibilité d'exécution imbriquée et quasi-parallèle de plusieurs instructions. On peut donc espérer avec cette structure, une augmentation, au moins théorique, de la puissance de calcul.

D'une manière très générale, la suite d'instructions exécutée par une structure en "pipe-line" est commandée par une seule et même tâche. Cette utilisation présente un grand défaut : le réseau ne sera pas toujours en mesu d'exécuter deux ou plusieurs instructions simultanément si celles-ci ne sont pas indépendantes. Deux instructions sont dites indépendantes si, pour toutes les valeurs possibles de l'ensemble de données qui les intéressent, les résultats obtenus ne dépendent pas de l'ordre suivant lequel elles sont exécutées [3].

Ce problème de dépendance peut réduire, quelquefois de manière importante, la vitesse maximale théorique de traitement d'une structure en "pipe-line". Diverses techniques ont été proposées pour réduire l'importance ce problème, mais elles impliquent très souvent des moyens matériels importants [4].

Un autre problème typique de cette utilisation est le traitement de alarmes internes ou erreurs : quand au cours de l'exécution d'une tâche on détecte une erreur, il est très souhaitable de connaître le contexte qui précède (ou qui suit) l'exécution de l'instruction responsable de cette alarme. Dans les structures en "pipe-line" évoquées dans ce paragraphe, il es très difficile de satisfaire à cette condition : durant l'exécution d'une instruction I, le contexte d'une tâche subit des modifications. Ces modificat ne sont pas seulement la conséquence du travail décrit par I, mais aussi de celui d'autres instructions exécutées simultanément.

3. MULTIPROCESSEURS. RESEAUX EN BARILLET

Pour réaliser un réseau qui vis-à-vis de l'utilisateur ait l'apparence d'un "n" processeur, on peut envisager plusieurs techniques. Une solution extrême consiste à construire n processeurs physiques. Le réseau ainsi obtenu est un multiprocesseur au sens strict : ses n processeurs peuvent travailler en parfait parallélisme, donc, il est très performant mais aussi très coûteux. Par ailleurs, on ne doit pas oublier les problèmes qui peuvent se poser au niveau de la synchronisation et des échanges d'information entre les processeurs du réseau*.

A l'autre extrême, on peut aborder le problème en considérant qu'un seul processeur physique sera partagé par un nombre n de processeurs virtuels. Dans le réseau ainsi obtenu, deux processeurs ne peuvent jamais travailler simultanément. La puissance de calcul est limitée, mais les échanges d'information peuvent s'effectuer très simplement grâce à des bus de données et registres communs. En outre, les opérations de synchronisation sont simplifiées par suite de l'impossibilité d'un parallélisme vrai au sein de ce réseau.

Entre ces deux cas extrêmes, il existe un grand éventail de solutions intermédiaires : deux ou plusieurs processeurs physiques sont partagés par un nombre différent de processeurs virtuels. Dans tous les cas, la présence d'un ordonnateur s'avère nécessaire pour régler les conflits et déterminer une stratégie au niveau des enchaînements processeurs virtuels-processeurs physiques.

Dans le cas d'un multiprocesseur en barillet, le critère de travail de l'ordonnateur est très simple : les processeurs physiques sont attribués cycliquement aux processeurs virtuels du réseau.

---

* Un des principaux problèmes liés à la synchronisation est dû au besoin impératif de rendre indivisibles certaines opérations élémentaires telles que les primitives de changement d'état P et V (c.f. III.1). En outre, pour ne pas compromettre les performances d'un réseau aussi important, on devra compter sur un grand nombre de voies pour les échanges d'information.

- C H A P I T R E II -

DEFINITION D'UN MULTIPROCESSEUR "PIPE-LINE"

Plusieurs techniques ont été développées pour augmenter les performances des calculateurs. Parmi ces techniques on trouve notamment les structures "pipe-line" et les réseaux multiprocesseurs [4] [5]. Dans ce chapitre nous essaierons de développer d'une manière simple, un réseau performant à n processeurs virtuels à partir d'un seul processeur physique. Pour obtenir ce résultat nous profiterons de l'exécution imbriquée et quasi-parallèle de plusieurs instructions* particulière à une structure en "pipe-line", tout en éliminant son plus grand défaut : la dépendance.

1. MULTIPROCESSEURS "PIPE-LINE"

Les problèmes de dépendance et de traitement des alarmes internes des structures en "pipe-line" exposés au paragraphe I.2., sont étroitement liés à l'application presque exclusive de cette technique aux systèmes monoprocesseurs [3] [4] [6] [7]. Ces problèmes sont dus à la rupture de l'ord strictement séquentiel dans l'exécution d'une tâche.

Cette dernière constatation nous a suggéré qu'une manière de contou ces deux problèmes, serait d'utiliser une structure en "pipe-line" pour le traitement simultané de plusieurs tâches. D'une manière générale dans la structure que nous proposons, l'ensemble des instructions exécutées quasi-parallèlement par le système ne doit jamais contenir plus d'une instruction dont l'exécution aurait été commandée par une même tâche : l'exécution de ch instruction de cet ensemble est donc commandée par une tâche différente [8].

---

* De même que pour les deux derniers paragraphes du chapitre I, dans le prem paragraphe de ce chapitre le mot instruction désigne indifféremment microinstruction, instruction, etc, et le mot processeur est employé pour désigner microprocesseur, processeur, etc.

Cette façon de procéder implique que l'ordre d'exécution de chacune des tâches est strictement séquentiel. Les deux problèmes cités plus haut sont simplement éliminés.

Le système ainsi défini est donc capable de traiter plusieurs tâches simultanément. C'est-à-dire qu'il s'agit d'un système multiprocesseur qui bien que ne l'étant pas au sens strict (c.f. I.3) en constitue une bonne approximation : il a la possibilité d'exécuter plusieurs instructions quasi-parallèlement.

Si tous les processeurs ont accès aux mêmes ressources, on dira qu'ils sont équivalents, autrement on dira qu'ils sont complémentaires (c.f. I.1). Dans le premier cas ils auront la possibilité d'exécuter indifféremment toute tâche soumise au système, dans le deuxième cas chacun d'entre eux ne pourra exécuter que certaines tâches spécifiques et normalement ils devront être tous concernés par un même travail global.

Théoriquement, la puissance globale de calcul (nombre d'instructions exécutées par unité de temps) d'un multiprocesseur et celle d'un monoprocesseur construits à partir d'une structure "pipe-line" équivalente, doivent être du même ordre. Néanmoins, l'absence du problème de dépendance dans le multiprocesseur fera que, dans un grand nombre d'applications, cette solution présente une puissance apparente plus grande que le monoprocesseur. En particulier, ce type de réseau est très avantageux quand on peut employer simultanément les processeurs virtuels qui le constituent *.

---

* A titre d'exemple, on peut dire que la performance d'un monoprocesseur "pipe-line" à deux étages est à peu près 1,5 fois plus grande que celle d'un monoprocesseur séquentiel [7]. Pour un biprocesseur "pipe-line" à deux étages, si l'on considère une perte de temps de l'ordre de 10 % due aux synchronisations entre tâches [9], la performance sera 1,8 fois plus grande que celle d'un monoprocesseur séquentiel.

Les systèmes à processeurs complémentaires sont bien adaptés au traitement des algorithmes qui se prêtent à un découpage en tâches spécialisées soit très peu interdépendantes soit d'une interdépendance assez régulière (c.f. chapitre IV). Les systèmes à processeurs équivalents ont probablement un champ d'application plus vaste, mais il est nécessaire de régler certains problèmes, liés surtout aux conflits entre tâches au niveau des entrées-sorties du système.

2. EXEMPLES DE MULTIPROCESSEURS "PIPE-LINE"

Dans ce paragraphe sont exposés deux exemples de multiprocesseurs définis à partir d'une structure en "pipe-line". Le premier exemple est un multimicroprocesseur (processeurs de micro-instructions), le deuxième un multiprocesseur (processeurs d'instructions). Plus loin, au chapitre IV, le premier exemple sera repris comme structure de base d'un réseau d'opérateurs adapté au traitement du signal.

a) Multi-microprocesseur en barillet "pipe-line"

Pour la définition de ce multi-microprocesseur, nous avons retenu les hypothèses suivantes :

- La durée d'exécution de toutes les microinstructions est constante.
- Il est possible de découper l'exécution de toutes les microinstructions en un même nombre de phases (ph. 0, ph. 1, ... ph. n).
- La durée d'exécution de chacune de ces phases est constante.

En outre, nous nous sommes imposés les trois contraintes suivantes :

- Toute phase ph. i doit être exécutée par un étage Ei unique.
- Le mode de travail du micro-ordonnateur (ordonnateur au niveau de l'enchaînement étages-microprocesseurs) doit être aussi simple que possible - à ce niveau, l'ordonnateur est nécessairement câblé ! -.

- La recherche et l'exécution des microinstructions peuvent être simultanées *.

Dans ce cadre, le nombre de processeurs du réseau doit être égal au nombre d'étages du "pipe-line". Un nombre inférieur entraînerait une perte des possibilités du réseau - à tout moment il y aurait au moins un étage inactif -, et un nombre supérieur impliquerait une complication de l'ordonnateur sans pour autant augmenter la puissance de calcul. De plus, étant donné qu'un microprocesseur ne peut traiter qu'une tâche à la fois et que le réseau ne doit pas effectuer simultanément deux microinstructions commandées par une même tâche, le meilleur mode de travail qu'on peut envisager pour le micro-ordonnateur est aussi le plus simple : celui du barillet (c.f. I.3). Chaque étage du "pipe-line" est attribué cycliquement à chacun des n processeurs du réseau (c.f. fig. II.1).

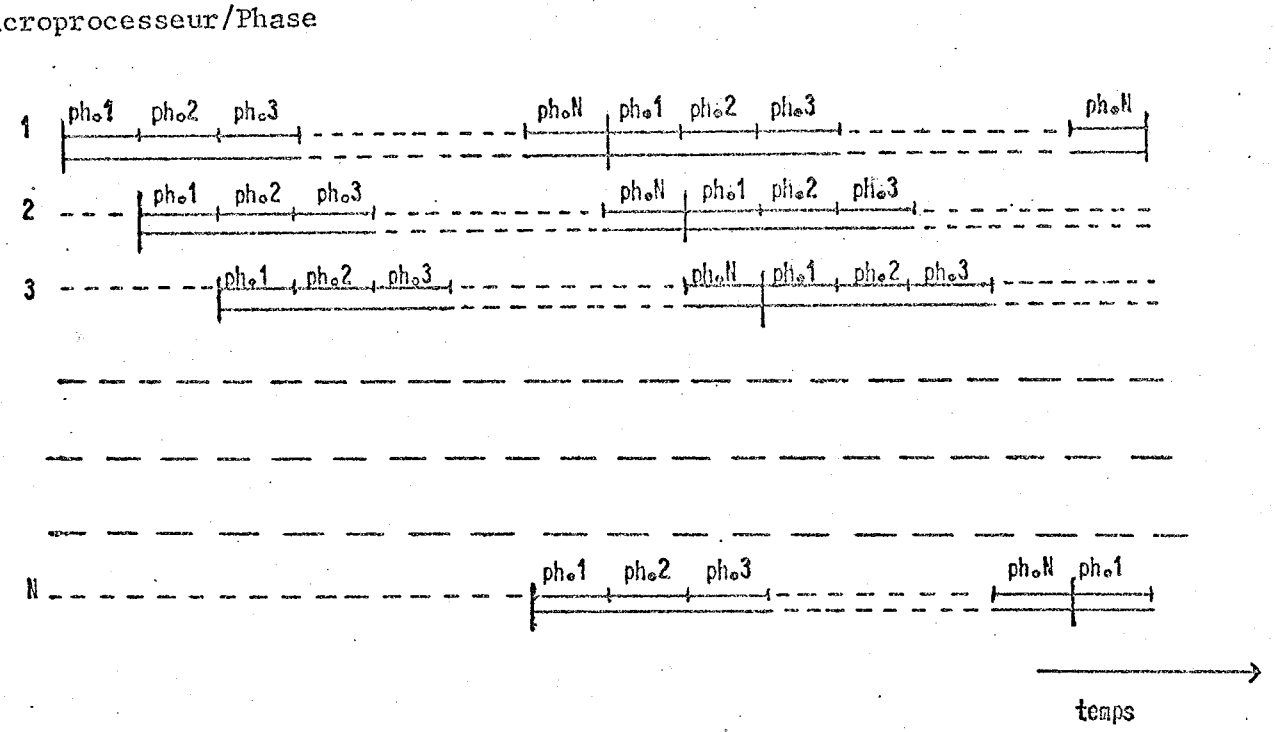


Figure II.1. : MODE DE TRAVAIL DU MICRO-ORDONNATEUR

* Les deux premières contraintes assurent une économie du matériel, la troisième un gain en puissance de calcul.

Ce type de fonctionnement implique que les étages du réseau soient utilisés cycliquement et sans interruption par tous les processeurs, même si, à un instant donné, l'un (ou plusieurs) d'entre eux ne doit pas exécuter d'instructions (soit parce qu'il n'a pas de travail, soit parce qu'il attend un évènement avant de pouvoir continuer son travail). Pour éviter un fonctionnemen logique incorrect, un processeur qui se trouve dans ces conditions doit interdire, à chacun des étages qu'il utilise successivement, l'exécution de toutes les actions susceptibles de changer le contexte du réseau. Ainsi, dès qu ce processeur sera en mesure de continuer son travail (ou d'en commencer un), il pourra le faire sans prendre aucune autre précaution supplémentaire.

La figure II.2 représente le bloc-diagramme d'un microprocesseur barillet "pipe-line". Il est composé de deux parties principales pouvant travailler simultanément : l'unité d'adressage de la mémoire de contrôle (micro-mémoire) et l'unité d'exécution des microinstructions. Pour que le fonctionnement de cette structure soit possible, il est nécessaire que les deux unités comportent le même nombre d'étages. Ceci est toujours possible si on ajoute - le cas échéant - des étages fantômes dans l'une de deux unités.

Il est important de faire remarquer que les registres temporaires (files d'entrée) nécessaires entre les différents étages ne devront jamais contenir plus d'un élément. Sa fonction se réduit presque exclusivement à rendre indépendantes les phases d'exécution de chaque microinstruction : étant données les hypothèses relatives à ces phases et le mode de travail du micro-ordonnateur, il est impossible d'accumuler des travaux à l'entrée d'un étage.

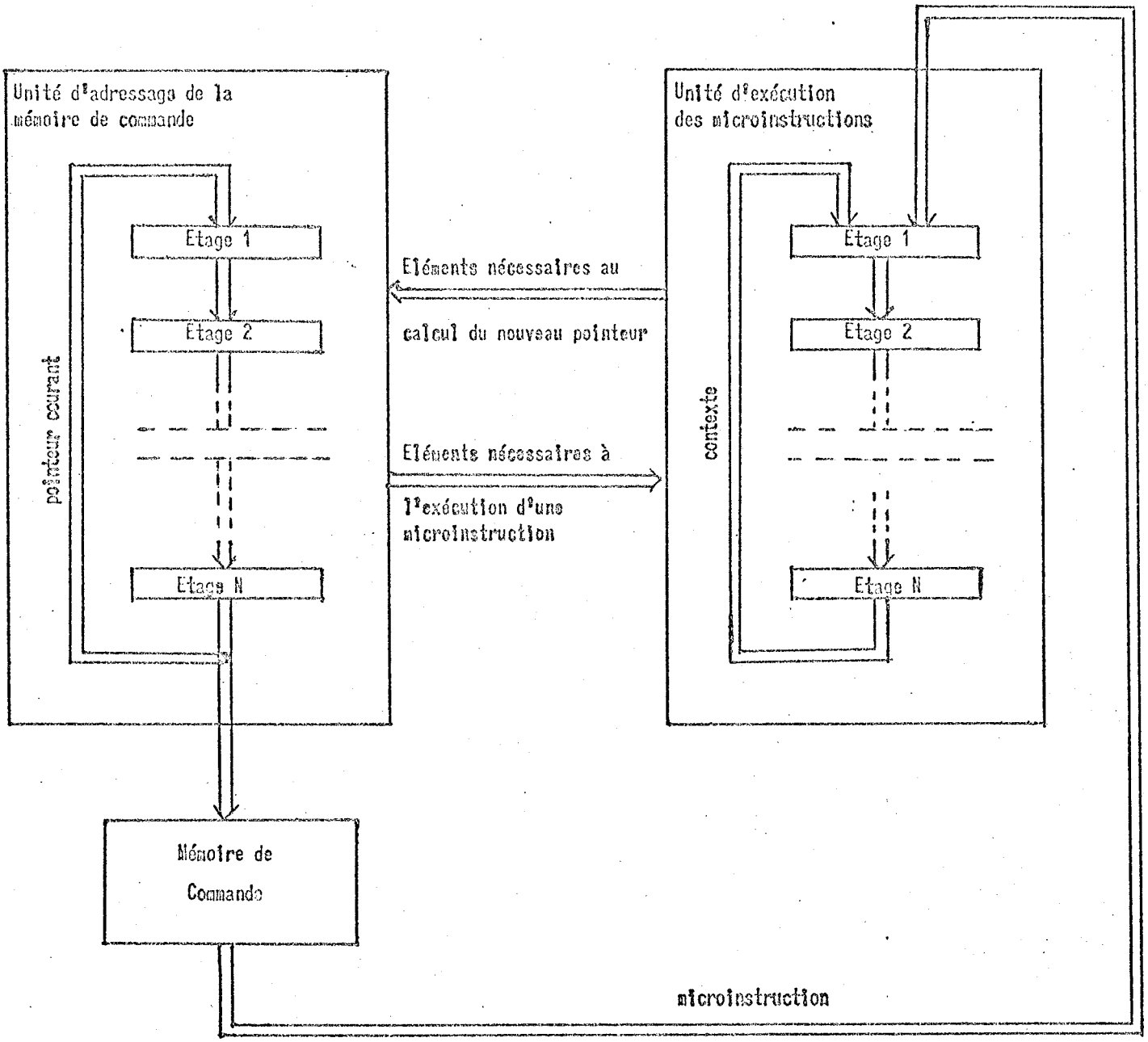


Figure II.2. : BLOC-DIAGRAMME D'UN MULTI-MICROPROCESSEUR BARILLET "PIPE-LINE"

b) Multiprocesseur "Pipe-line" (au niveau des instructions)

Le multiprocesseur défini dans l'exemple précédent ne peut pas convenir à l'exécution de tâches au niveau programmé. En effet, l'hypothèse concernant le découpage en un même nombre de phases pour toutes les instructio ainsi que celle relative à une même durée d'exécution de toutes ces phases, sont difficilement satisfaites à ce niveau. En plus, les contraintes de simplicité imposées à l'ordonnateur peuvent être assouplies du fait que, dans cas, ce ne sera pas forcément un automate câblé : il peut s'agir par exemple, d'un automate micro-programmé (cette dernière remarque peut s'appliquer à tous les étages du système).

Dans ce paragraphe, nous nous proposons de donner quelques éléments définition d'un multiprocesseur "pipe-line" répondant aux conditions suivantes

1. La durée d'exécution des instructions peut être très variable.
2. Le nombre de phases d'exécution des différentes instructions n'est pas le même.
3. Le temps nécessaire pour effectuer chacune de ces phases peut être variable.

La figure II.3 représente un schéma simplifié d'un système de traitement en "pipe-line" pouvant satisfaire ces conditions. Nous faisons une différence entre les étages intermédiaires et les étages finaux : les étages finaux sont ceux qui exécutent la dernière phase d'une instruction, tous les autres sont des étages intermédiaires. A tout moment une liste contient les mots d'état des tâches exécutées par les processeurs du système (l'information contenue dans le mot d'état d'une tâche doit permettre de restituer son contexte). A chaque processeur du système on fait correspondre un bit indicate d'exécution. Ce bit est égal à 1 tant que le processeur associé exécute une instruction dans le système, sinon il est égal à 0. L'ordonnateur choisit un processeur Pi parmi ceux dont l'indicateur d'exécution est égal à 0*, et détermine (à partir du mot d'état de la tâche exécutée par Pi) l'adresse de l'instruction à exécuter. Si la file d'attente du premier étage n'est pas plei

---

* Le critère de choix de l'ordonnateur peut être quelconque, par exemple une recherche cyclique (en donnant la même priorité à tous les processeurs) ou b une recherche tenant compte de certains principes de priorité.

l'ordonnateur y placera cette adresse ainsi que le nom du processeur Pi concerné. En même temps il forcera à la valeur 1 l'indicateur d'exécution correspondant. Si par contre cette file est pleine, l'ordonnateur devra attendre qu'une place se libère avant de continuer son travail.

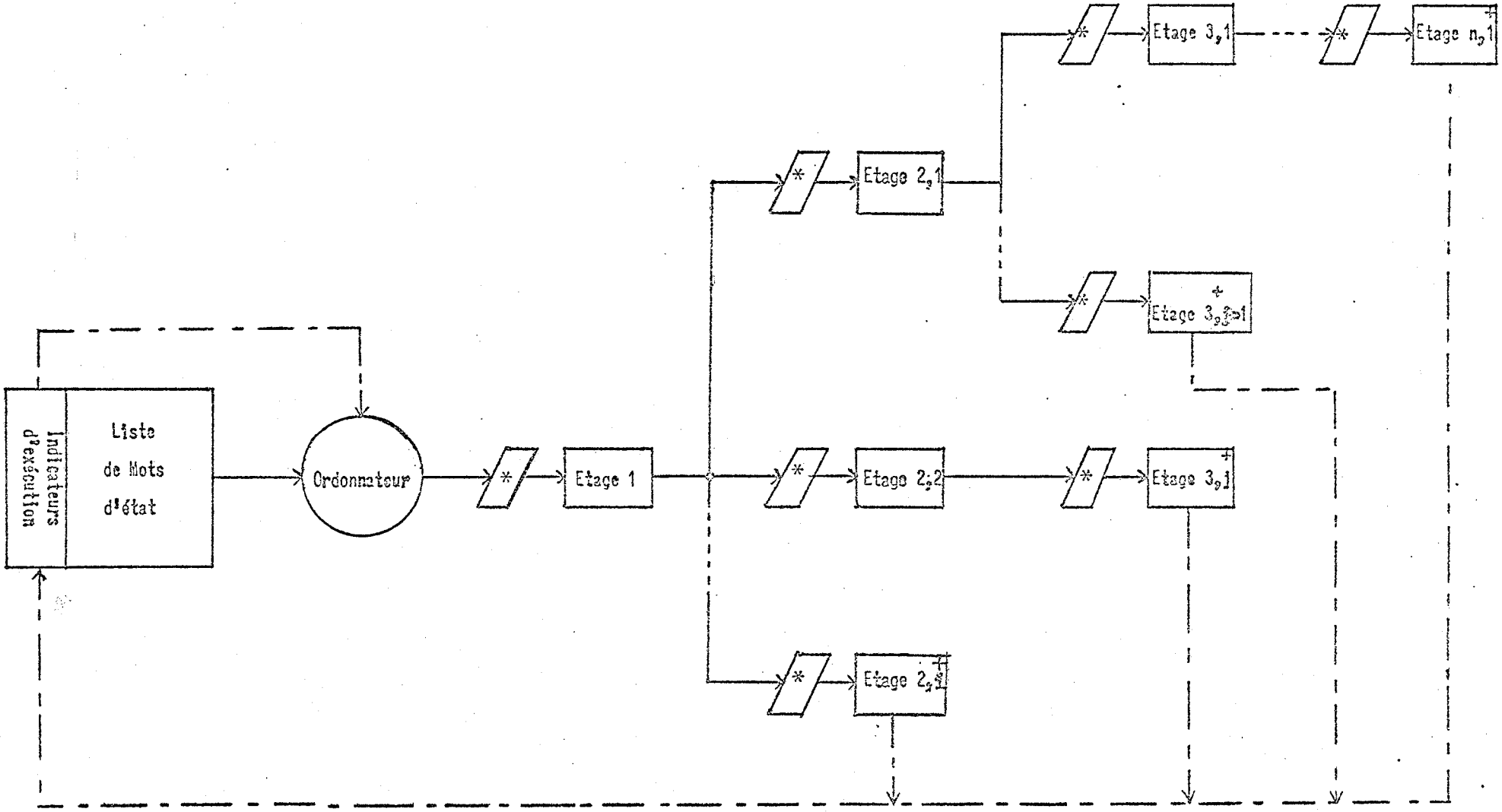
Dans les files d'entrée correspondantes, tous les étages disposent des indications concernant les travaux à effectuer ainsi que les noms des processeurs qui les commandent (aussi longtemps qu'une de ces files est vide, l'étage associé n'aura pas de travail à effectuer et sera en attente). Chaque fois qu'un étage intermédiaire achève une phase d'une instruction, il le signale à l'étage chargé de continuer le traitement de l'instruction. Pour ce faire, il place dans la file d'attente de ce dernier toutes les indications nécessaires à la poursuite de l'exécution de l'instruction. Chaque fois qu'un étage final termine le traitement d'une instruction, il le signale à l'ordonnateur en forçant la valeur 0 dans l'indicateur d'exécution correspondant.

Un processeur ne pouvant traiter qu'une seule tâche à la fois, l'indicateur d'exécution n'autorisera jamais l'exécution simultanée de deux instructions commandées par une même tâche - condition indispensable au bon fonctionnement du multiprocesseur "pipe-line" (c.f. II.1) -.

Le nombre de processeurs du système dépend de la capacité de la liste de mots d'état. Ce nombre doit être défini en fonction de critères qui tiennent compte de :

- La quantité de matériel supplémentaire nécessaire pour réaliser chaque nouveau processeur.
- La puissance individuelle de calcul demandée aux processeurs
- Le nombre de phases d'exécution des instructions

Intuitivement, nous pensons que pour la meilleure utilisation du système, le nombre de processeurs doit être voisin du nombre maximum de phases nécessaires à l'exécution de l'instruction la plus compliquée. Un plus grand nombre de processeurs entraînerait :



* files d'attente

\+ Etages finaux

Figure II.3. : SCHEMA SIMPLIFIE D'UN MULTIPROCESSEUR "PIPE-LINE" POUR L'EXECUTION DE TACHES AU NIVEAU PROGRAMME.

- Une augmentation du matériel supplémentaire (p. ex. les registres particuliers nécessaires à chaque processeur).
- Une diminution de la puissance individuelle de calcul de chaque processeur du système (La puissance globale de calcul restant, au moins théoriquement, constante).

- C H A P I T R E III -

O U T I L S D E S Y N C H R O N I S A T I O N

Les systèmes multiprocesseurs exposés au deuxième chapitre, ont la possibilité de traiter plusieurs tâches simultanément. Il est courant que ces tâches participent ensemble à l'exécution d'un travail plus vaste et, dans ces cas, les tâches doivent être synchronisées. Les opérations de synchronisation ne doivent pas perturber le déroulement des tâches, et dans le cas idéal elles doivent se limiter à différer leur exécution. D'une manière générale les solutions apportées à ce problème se classent en deux groupes :

- Conception d'un organe privilégié - moniteur ou superviseur - chargé de réaliser toutes les synchronisations du système. Chaque fois qu'une tâche désire exécuter une de ces opérations, elle fait appel au moniteur.
- Conception d'outils adaptés permettant aux tâches d'effectuer directement les opérations de synchronisation. Cette solution ne demande pas (au moins conceptuellement) l'existence d'organes privilégiés.

Par rapport à la première méthode, la deuxième offre l'avantage d'une gestion décentralisée de l'exécution des tâches. Ce type de gestion est plus souple et plus compréhensible pour l'utilisateur.

En 1965 DIJKSTRA [10] a introduit des outils adaptés à ce type de traitement : les sémaphores et deux primitives de changement d'état. Mais avant de décrire leur fonctionnement, il est nécessaire de donner quelques définitions

1. ETATS D'UNE TACHE. PRIMITIVES DE CHANGEMENT D'ETAT

La figure III.1. représente un diagramme en arbre de tous les états possibles d'une tâche* [11].

---

* Les définitions qu'on donne par la suite pour le couple processeur-tâche, peuvent s'étendre à tous les niveaux de traitement (c.f. I.1.).

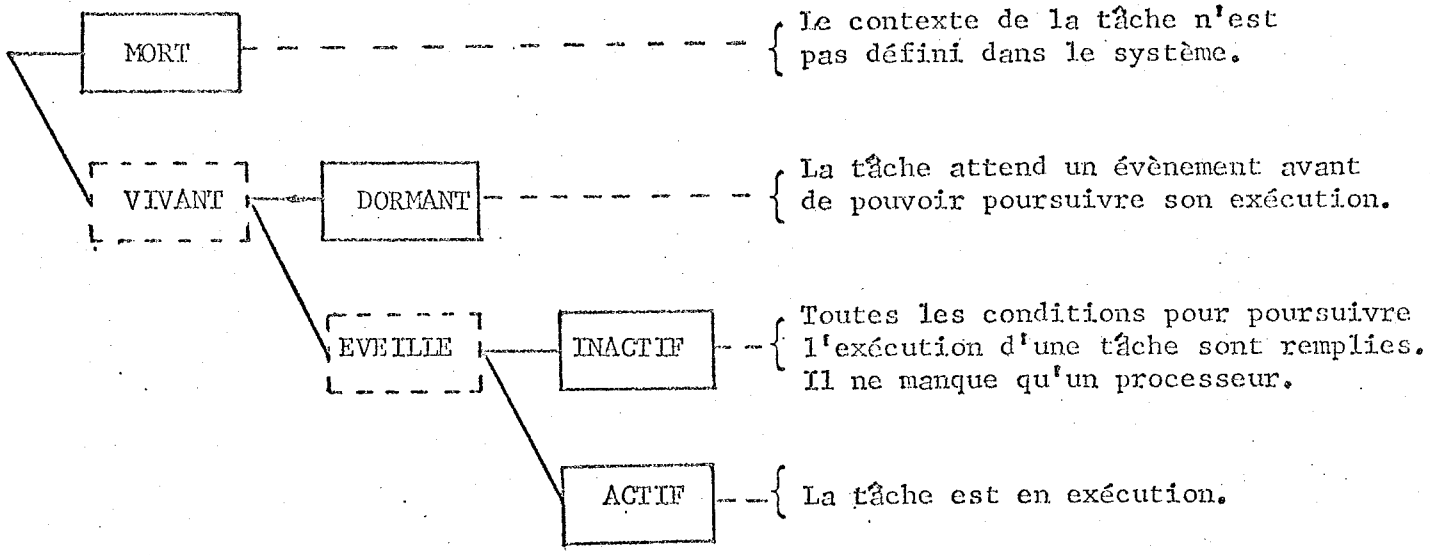


Figure III.1. : ETATS D'UNE TACHE

D'un point de vue formel, les seuls états significatifs d'une tâche sont les états : mort, vivant, dormant et éveillé. La distinction actif ou inactif de l'état éveillé est due exclusivement à une limitation de la quantité de matériel de calcul. Les transitions entre les états actif et inactif sont décidées par l'ordonnateur (c.f. I.1) d'après les disponibilités du matériel et d'autres critères (p. ex. priorités).

Les passages de l'état vivant à l'état mort et vice-versa concernent la gestion dynamique du nombre des tâches dans le système. Finalement, les transitions entre les états éveillé et dormant sont liées à la synchronisation entre tâches.

Le passage d'une tâche Ti de l'état éveillé à l'état dormant ou mort ne peut être décidé que par elle même (pour que ce type de passage soit possible, il faut que Ti soit à l'état actif). Par contre, le passage d'une tâche Tj de l'état dormant ou mort à l'état éveillé ne peut être décidé que par une tâche Tk différente, (Tk doit être nécessairement à l'état actif).

2. SEMAPHORES ET PRIMITIVES DE CHANGEMENT D'ETAT P ET V [12]

a) Définition d'un sémaphore

Essentiellement, un sémaphore est l'association d'une variable entiè S et d'une file d'attente "F". Pour des raisons de simplicité, le nom du sémaphore et celui de la variable associée sont les mêmes. La valeur absolue d S représente :

- Si $S \geqslant 0$, le nombre de ressources* disponibles dans le sémaphore.
- Si $S \leqslant 0$, le nombre de tâches dormant sur le sémaphore (elles attendent une ressource).

Si $S < 0$, la file F contient les mots d'état (c.f. I.4.b) des tâches qui dorment sur le sémaphore. Si $S \geqslant 0$, le contenu de F n'a pas de significati (cette dernière remarque, on le verra plus loin, n'est pas applicable à d'autr types de sémaphores).

b) Les primitives de changement d'état P et V

Les seules opérations que l'on autorise sur les sémaphores, sont cel définies par les primitives de changement d'état P et V. La primitive P permet à une tâche Ti de demander les ressources nécessaires à son exécution. Elle constitue la seule possibilité pour que Ti passe de l'état éveillé à l'état dormant. Le fait que Ti commande la primitive P(S) se traduit par l'exécution la séquence suivante :

```
P(S) ⟹   DEBUT  S : S - 1;
                SI  S < 0 ALORS "endormir Ti sur le sémaphore S";
                                (Son mot d'état est retiré de la liste
                                des tâches éveillées et rangé dans la
                                file F. Ti passe à l'état dormant).
         FIN;
```

---

* On considère qu'une ressource est tout élément (donnée, évènement, etc...) nécessaire pour qu'une tâche puisse, logiquement, continuer à être exécutée un processeur.

La primitive V permet à une tâche Tj de mettre une ressource à la disposition de l'ensemble de tâches. Elle constitue la seule possibilité pour faire passer une tâche Tk (k # j) de l'état dormant à l'état éveillé. Le fait que Tj commande la primitive V(S) se traduit par l'exécution de la séquence suivante :

V(S) ⟹ DEBUT S : S + 1;

SI S ≤ 0 ALORS "Réveiller une tâche Tk parmi celles qui dorment sur le sémaphore S"; (Le mot d'état de Tk est pris dans F pour être inséré dans la liste de tâches éveillées).

FIN;

Les primitives P(S) et V(S) sont indivisibles c'est-à-dire : pendant qu'elles s'exécutent, aucune autre modification de la variable S ou de la file F n'est autorisée.

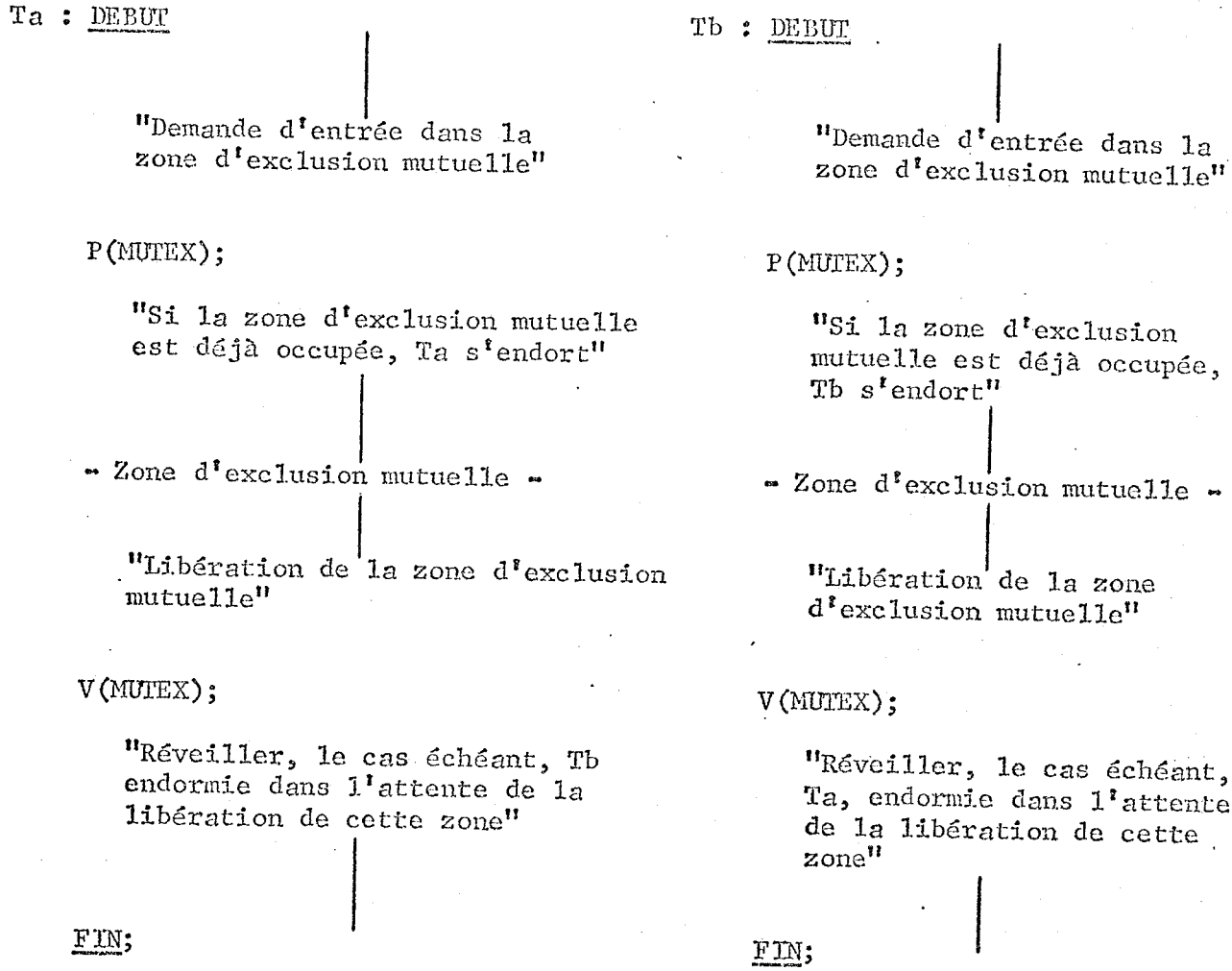
c) Les sémaphores privés

Il est courant qu'il n'existe qu'une seule tâche T autorisée à commander l'exécution de la primitive P sur un sémaphore SP donné. Dans ce cas on dit que SP est un sémaphore privé. La capacité de la file d'attente FP d'un tel sémaphore est unitaire car elle ne pourra contenir que le mot d'état de T.

d) Exemples d'application

Problème 1 : Soit à réaliser une zone d'exclusion mutuelle entre deux tâches Ta et Tb, c'est-à-dire une section de programme qui, pour un fonctionnement logique correct, ne doit jamais être utilisée simultanément par les deux tâches.

Pour ce faire, il suffit de définir un sémaphore "MUTEX" et de l'initialiser à la valeur 1. Le fonctionnement de Ta et Tb est décrit par les programmes suivants :



Problème 2 : Soit à réaliser n + 1 tâches cycliques, dont n du type CREER(j) (j = 1, n), chargées de générer des ressources, et une du type SAISIR chargée de les utiliser. On dispose d'une file D de capacité limitée à x ressources. On peut y introduire et retirer une ressource simultanément, mais il est interdit d'y introduire deux ressources en même temps. Cette introductio devra se faire donc dans une zone d'exclusion mutuelle.

Pour résoudre ce problème, il suffit de définir trois sémaphores :

- LIBRE (initialisé à la valeur x) qui indique le nombre de places libres dans la file D.
- RESSOURCE (sémaphore privé initialisé à la valeur 0) qui indique le nombre de ressources disponibles dans D.
- MUTEX (initialisé à la valeur 1) pour l'obtention d'une zone d'exclusion mutuelle pour l'introduction de données dans D.

Le fonctionnement des tâches CREER(j) et SAISIR est décrit par :

CREER(j) :

DEBUT

BOUCLE :

- Génération d'une ressource -

"Demande d'une place libre dans la file D"

P(LIBRE);

"S'il n'y a pas de places libres, la tâche CREER(j) s'endort"

"Demande d'entrée dans la zone d'exclusion mutuelle"

P(MUTEX);

"Si la zone d'exclusion mutuelle est déjà occupée, CREER(j) s'endort"

- Introduction de la ressource dans D -

"Libération de la zone d'exclusion mutuelle"

V(MUTEX);

"Réveiller, le cas échéant, l'une des tâches endormies dans l'attente de la libération de la zone d'exclusion mutuelle"

V(RESSOURCE);

"Réveiller, le cas échéant, la tâche SAISIR endormie dans l'attente d'une ressource"

ALLERA BOUCLE;

FIN;

SAISIR :

DEBUT

HAUT : "Demande d'une ressource contenue dans la file D"

P(RESSOURCE);

"Si la file D est vide, la tâche SAISIR s'endort"

- Retirer la ressource de D -

"Libération d'une place dans la file D"

V(LIBRE);

"Réveiller, le cas échéant, l'une des tâches endormies dans l'attente d'une place libre dans la file D"

- Utilisation de la ressource -

ALLERA HAUT;

FIN;

3. MISE EN OEUVRE DES SEMAPHORES ET PRIMITIVES P ET V DANS UN MULTIPROCESSEUR "PIPE-LINE"

Dans ce paragraphe nous nous proposons de donner quelques éléments pour la mise en oeuvre des sémaphores et primitives P et V pour la synchronisation entre tâches, au niveau des instructions, dans un système multiprocesseur "pipe-line" comme celui défini en II.2.b. Les hypothèses de travail retenues sont les suivantes :

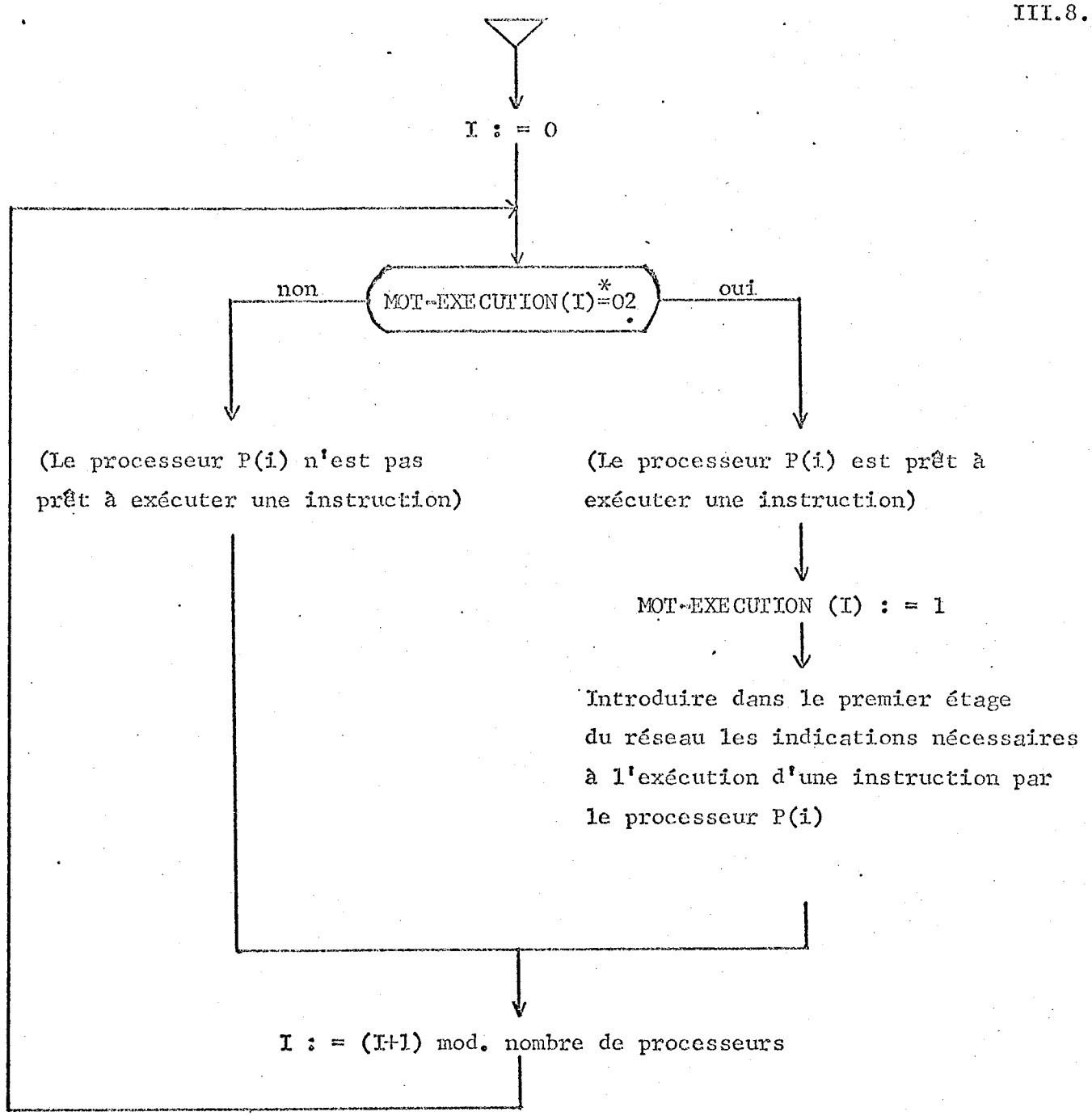
- Les tâches disposent de deux instructions spéciales pour assurer leur synchronisation : Ip et Iv qui impliquent respectivement l'exécution d'une primitive P et d'une primitive V.
- Chaque tâche traitée par le système est munie d'une priorité représentée pa un nombre positif. Une tâche en cours d'exécution peut changer sa priorité l'aide d'une instruction spéciale Ich, mais en aucun cas ne peut modifier c des autres.
- Les processeurs du système sont requérables : ils doivent être toujours affectés à l'exécution des tâches recueillies les plus prioritaires.

Par ailleurs, nous avons considéré que tous les processeurs en état d'exécuter une instruction, ont la même priorité d'accès aux différents étage du réseau.

La réalisation de ce système nécessite deux ordonnateurs :

- O e.p pour l'allocation des étages aux différents processeurs du réseau.
- O p.t pour l'allocation des processeurs aux différentes tâches exécutées pa le réseau.

La figure II.3 montre que ce multiprocesseur est composé de plusieu étages. L'exécution de chaque instruction est décomposée en un nombre de phas précises (p. ex. lecture de l'instruction, recherche des opérandes, etc...). Pour mener à terme chacune de ces phases, on fait appel à un étage bien déterminé.



* Le MOT-EXECUTION correspond plus ou moins à l'indicateur d'exécution ex en II.2.b mais, on verra plus loin, mot pourra avoir des valeurs autres que 0 ou 1.

Figure III.2. : ORGANIGRAMME DU FONCTIONNEMENT DE 0 e.p.

Dans le cas des instructions Ip et Iv, il est nécessaire d'exécuter les primitives de changement d'état P et V respectivement. Une solution pour obtenir un traitement indivisible de ces deux primitives, consiste à grouper toutes les actions qu'elles impliquent dans une seule et même phase d'exécutio ph.S. Comme dans le système il n'y a qu'un étage Es capable de mener à terme cette phase, les primitives de changement d'état seront exécutées séquentiellement. Par ailleurs, dans le cas des instructions du type Ich, la phase de changement de priorité proprement dite, sera exécutée dans la même phase que les primitives P et V, et par là, menée à terme par le même étage Es

Lors de changements d'état d'une tâche, il peut se produire des conflits pour l'allocation de processeurs. Ces conflits, on l'a vu, doivent être réglés par l'ordonnateur O p.t suivant le critère de la plus grande priorité. Pour faciliter son travail, on considère qu'un processeur libre - c'est-à-dire qui n'a pas de tâches à exécuter - est affecté à l'exécution d'une tâche fictive de priorité - 1 (la priorité de cette tâche est ainsi plus petite que celle de n'importe quelle tâche réelle).

Une solution pour affecter la même priorité à tous les processeurs d réseau, est de faire travailler l'ordonnateur O e.p suivant l'organigramme de la figure III.2, c'est-à-dire celui d'une allocation cyclique du réseau aux processeurs prêts à exécuter une instruction.

Les actions de Es, O e.p. et O p.t. sont étroitement liées. L'utilisation de certaines données par ces trois organes (p. ex. files d'attente des tâches actives et inactives, registres de priorité, etc.) doit s faire à l'intérieur de zones d'exclusion mutuelle, ce qui limite les possibilités d'un déroulement parallèle de leurs travaux. Ainsi, la définition d'un seul organe de calcul pour exécuter toutes les actions impliquées par Es, O e.p. et O p.t. est une solution facile pour régler leurs conflits sans pour autant compromettre les performances.

Pour la mise en oeuvre de cet organe de calcul, nous allons considérer que :

- Es est un étage final.
- La notion d'indicateur d'exécution exposée en II.2.b est remplacée par celle de mot d'exécution. Ce mot prend les valeurs 0, 1, 2, 3, 4 ou 5 selon que son processeur associé se trouve dans l'un des cas suivants :

  0 → Le processeur est prêt à exécuter une instruction

  1 → Le processeur est en train d'exécuter une instruction, mais il ne se trouve pas dans la phase ph. S.

  2 à 4 → Le processeur se trouve dans la phase d'exécution ph. s d'une instruction Ip, Iv et Ich. respectivement.

  5 → Le processeur n'est attaché à l'exécution d'aucune tâche réelle.

- Avant d'allouer le réseau à un processeur, il faut forcer la valeur 1 dans son mot d'exécution.
- A la fin de la dernière phase d'exécutin d'une instruction autre que Ip, Iv ou Ich, on force la valeur 0 dans le mot d'exécution du processeur concerné.
- A la fin de l'avant dernière phase d'exécution d'une instruction Ip, Iv ou Ich, on force le mot d'état du processeur concerné à la valeur 2, 3 ou 4 respectivement.
- Chaque processeur du réseau dispose d'un registre RPS propre. Lors de l'exécution des instructions Ip et Iv, avant d'entamer la phase ph. s, les processeurs doivent y forcer le nom du sémaphore concerné. De même, avant l'exécution de la phase ph. s d'une instruction Ich, RPS doit contenir la valeur de la nouvelle priorité de la tâche respective.

L'organigramme de la figure III.3 décrit le fonctionnement combiné de l'étage Es et des ordonnateurs O e.p. et O p.t.

Remarque : Dans le système ainsi défini, il est très facile de traiter les appels externes (ou interruptions prioritaires). Il suffit de traduire chacun de ces appels en une opération V sur un sémaphore privé particulier à la tâche chargée de sa gestion.

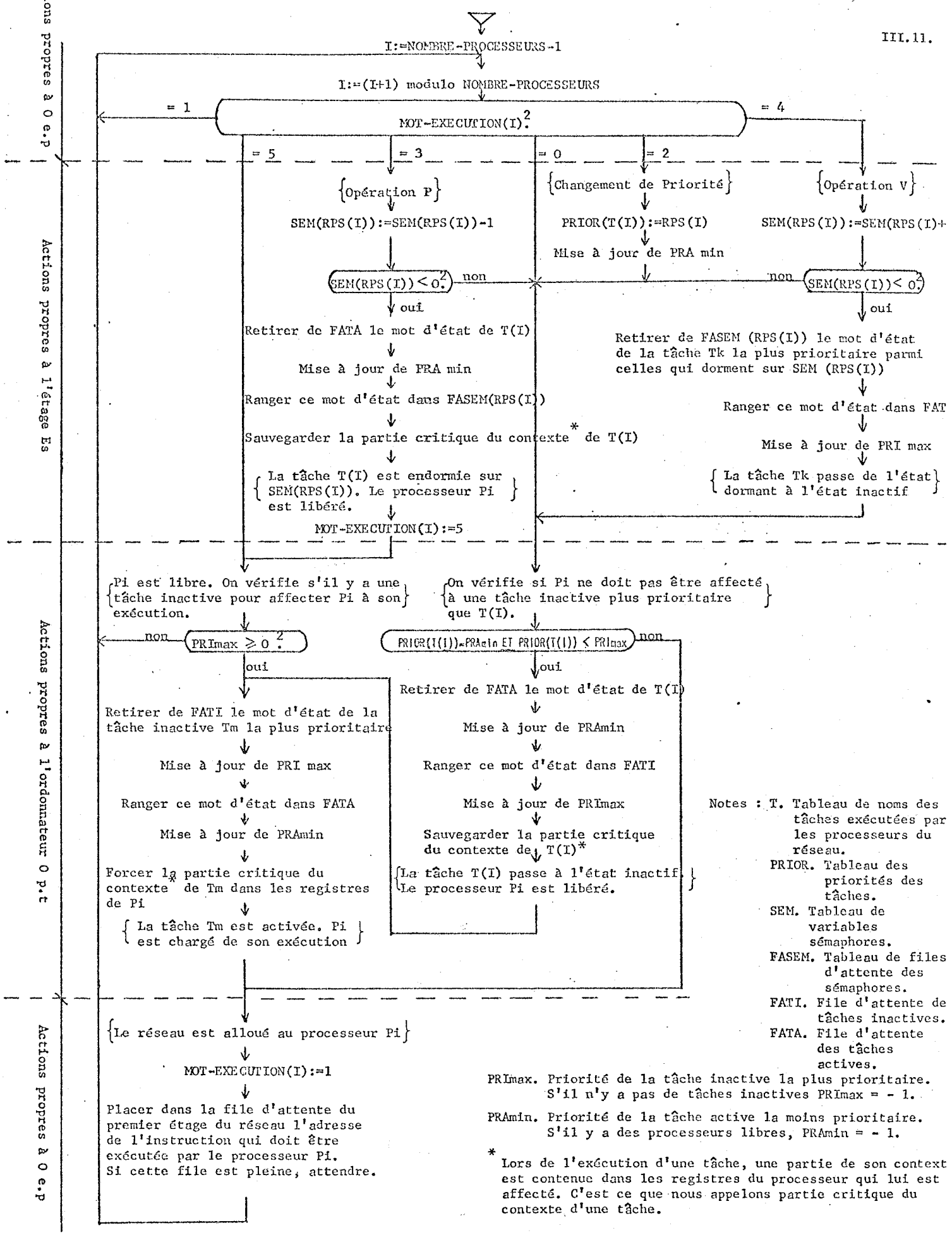


Figure III.3. : ORGANIGRAMME DU SYSTEME DE SYNCHRONISATION

4. SEMAPHORES COMMUNICANTS

Le problème 2 présenté au paragraphe III.1.d pourrait être formalisé plus facilement si les sémaphores avaient la possibilité de communiquer certains messages. Or SAAL et RIDDLE [13] ont remarqué que dans un sémaphore S tel que DIJKSTRA l'avait défini, on n'employait la file F que lorsque S < 0 pour y ranger les mots d'état des tâches endormies, ils ont proposé de l'utiliser dans le cas contraire (S > 0) pour garder des messages éventuels. Ils ont appelé ces outils "sémaphores communicants", et ont proposé les primitives ENVOYER et RECEVOIR pour leur gestion. La commande par une tâche Ti de la primitive mssg(i) : = RECEVOIR(S), et la commande par une tâche Tj de la primitive ENVOYER(S) : = mssg(j) se traduisent par l'exécution des séquences suivantes :

```
mssg(i) : = RECEVOIR(S) ⟹ DEBUT S : = S - 1;
                            SI S < 0 ALORS Endormir Ti sur le sémaphore S;
                                     SINON DEBUT Prendre un message mssg(k)
                                                 parmi ceux gardés en F;
                                                 mssg(i) : = mssg(k);
                                           FIN;
                            FIN;

ENVOYER(S) : = mssg(j) ⟹ DEBUT S : = S + 1;
                            SI S ≤ 0 ALORS DEBUT Réveiller une tâche Tk
                                                 parmi celles qui dorment
                                                 sur le sémaphore S;
                                                 mssg(k) : = mssg(j);
                                           FIN;
                                     SINON Garder mssg(j) dans la file F;
                            FIN;
```

Il faut faire trois remarques concernant ces primitives :

- De même que P et V, RECEVOIR et ENVOYER sont des opérations indivisibles.
- Dans le cas des sémaphores non employés pour communiquer des messages, les primitives simplifiées RECEVOIR(S) et ENVOYER(S) correspondent exactement aux primitives P(S) et V(S).
- Tout problème réalisable avec les sémaphores communicants peut l'être aussi avec ceux de DIJKSTRA. Néanmoins, les solutions apportées par les premiers peuvent être plus simples, nécessiter moins de sémaphores et faire appel à l'exécution d'un nombre plus limité de primitives de changement d'état.

Pour illustrer l'utilisation de ces sémaphores, reprenons le problème des tâches CREER(j) et SAISIR. On n'aura plus besoin de la pile D, puisque l'accumulation de ressources est faite dans les files d'attente des sémaphores communicants. En plus seulement deux sémaphores seront utilisés :

- LIBRE : (Sémaphore simple initialisé à la valeur x) indique le nombre de plac libres dans la file d'attente du sémaphore RESSOURCE.
- RESSOURCE : (Sémaphore communicant initialisé à la valeur 0) indique le nombr de ressources contenues dans sa file d'attente.

Le fonctionnement des tâches CREER(j) et SAISIR est maintenant décrit par les programmes :

CREER(j) : DEBUT

BOUCLE :

Génération d'une
Ressource Ressou(j)

RECEVOIR(LIBRE);

ENVOYER(RESSOURCE) : = Ressou(j);

ALLERA BOUCLE;

FIN;

SAISIR : DEBUT

HAUT:RES:=RECEVOIR(RESSOURC

ENVOYER (LIBRE);

Utilisation de la
ressource RES

ALLERA HAUT;

FIN;

La solution apportée à ce problème par les sémaphores de DIJKSTRA exige l'exécutin de six primitives pour chaque ressource transmise, tandis que cette solution n'en exige que quatre.

5. PRIMITIVES PARALLELES DE CHANGEMENT D'ETAT

PATIL [14] a démontré formellement l'existence de problèmes de synchronisation qui nécessitent d'adjoindre aux sémaphores et primitives P et V des instructions conditionnelles au niveau des programmes*. Il a aussi remarqué que ces programmes peuvent être très compliqués et, surtout, demander l'exécution d'un grand nombre de primitives P et V. Or en fait, ces problèmes pourraient être traités très facilement si l'on disposait d'une primitive Pp pouvant agir d'une façon parallèle et indivisible sur plusieurs sémaphores : une tâche qui commande l'exécution de la primitive parallèle Pp (S_1, ... Si, ... Sn), devra attendre que toutes les ressources demandées soient disponibles avant d'agir parallèlement sur tous les sémaphores Si impliqués. Par extension, on peut aussi définir la primitive parallèle Vp (S_1, ... Si, ... Sn), bien que celle-ci ne présente pas un intérêt aussi grand que la première : si l'exécution de la séquence P(S_1), ... P(Si), ... P(Sn) ne peut pas toujours se substituer avec succès à celle de Pp (S_1, ... Si, ... Sn), l'exécution de V(S_1), ... V(Si), ... V(Sn) peut toujours se substituer à celle de Vp (S_1, ... Si, ... Sn).

Il est important de faire quelques remarques à propos de ces primitives :

- Le mot d'état d'une tâche endormie ne peut pas être toujours rangé dans une file d'attente associée à un sémaphore particulier, à moins de prendre certaines précautions particulières. En effet, une tâche peut être endormie sur plusieurs sémaphores.
- Contrairement aux primitives de DIJKSTRA, les primitives parallèles autorisent qu'une ou plusieurs tâches dorment sur un sémaphore même si celui-ci dispose des ressources.

---

* La démonstration de l'existence de ces problèmes ainsi que l'analyse des programmes de synchronisation proposés, sont développés en [14]. Ces raisonnements étant assez longs et laborieux, nous ne les avons pas rapportés ici.

- Lors de l'exécution d'une primitive Vp (S_1, ... Si, ... Sn), il est beaucoup plus difficile de déterminer s'il y a des tâches à réveiller, car il ne suffit pas de tester seulement les sémaphores Si impliqués par Vp : pour pouvoir réveiller une tâche il faut tenir compte de tous les sémaphores sur lesquels elle dort.

On peut imaginer plusieurs méthodes pour mettre en oeuvre ces primitives et sémaphores parallèles, selon les applications envisagées et les moyens matériels disponibles. Au paragraphe suivant, nous décrirons un système de synchronisation à base de ces outils.

6. SYNCHRONISATION A L'AIDE DES SEMAPHORES ET PRIMITIVES DE CHANGEMENT D'ETAT PARALLELES DANS UN MULTI-MICROPROCESSEUR EN BARILLET "PIPE-LINE"

Le but de ce paragraphe est d'illustrer la mise en oeuvre des sémaphores et primitives de changement d'état parallèles pour la synchronisatic entre tâches dans un multi-microprocesseur en barillet "pipe-line" semblable à celui défini en II.2.a. Ces primitives devant être employées au niveau des microinstructions, on doit définir un automate câblé pour les exécuter.

Nous avons adopté les hypothèses suivantes :

- L'ensemble des tâches exécutées par le système est décomposé en groupes. Le nombre de tâches qui composent chacun de ces groupes est inférieur ou égal au nombre de microprocesseurs du système.
- Chacun des groupes dispose de N sémaphores. Les tâches ne peuvent se servir que des sémaphores propres à leur groupe.
- A tout instant, les microprocesseurs du système ne peuvent être attachés qu'à l'exécution des tâches d'un seul et même groupe. Tant que le système reste alloué à un groupe, chacune de ses tâches dispose d'un microprocesseur propre

Autrement dit, le système ne peut être alloué que globalement à l'exécution d'un groupe particulier de tâches. Les problèmes d'allocation du système à chacun des groupes, ainsi que ceux dus à la synchronisation entre ces différents groupes de tâches sont traités à un niveau d'exécution différent et nous ne les examinons pas ici*.

Nous allons définir un sémaphore S comme une variable entière dont la valeur, toujours supérieure ou égale à zéro, représente le nombre de ressources disponibles dans le sémaphore. Comme cette variable est symbolisée par le contenu d'un registre Rs (ou tout autre élément physique), elle est bornée supérieurement par un certain nombre Ns. Contrairement aux primitives parallèles Pp (LSEM) (LSEM représente la liste des sémaphores impliqués dans l'opération), les primitives parallèles Vp (LSEM) ne peuvent pas différer l'exécution de la tâche qui les commande. Si au moins l'un des sémaphores S de la liste LSEM a la valeur maximale Ns, l'exécution de Vp (LSEM) entraîne forcément une erreur.

Une façon d'éviter ce problème consiste à associer à tout sémaphore S un sémaphore symétrique S' tel que l'égalité S' = Ns - S soit toujours respectée (les valeurs des sémaphores S et S' sont caractérisées par le contenu du même registre Rs). Ainsi, l'exécution des primitives Pp(S) et Pp(S') entraînent respectivement l'exécution de Vp(S') et Vp(S). Dans ce cadre, nous pouvons définir une primitive parallèle généralisée SYNC qui comporte les deux autres Pp et Vp :

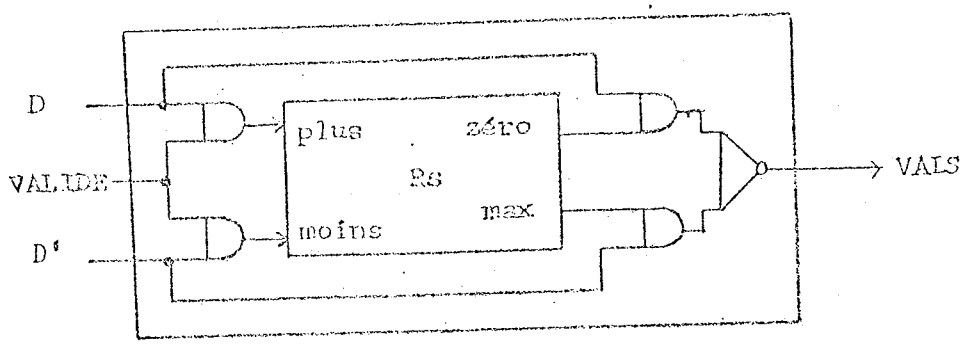
SYNC (LSEM) ⟹ Pp (LSEM); Vp (LSEM')

LSEM' représente la liste des sémaphores symétriques de LSEM.

Pour la mise en oeuvre de cette primitive, nous avons groupé toutes ses actions dans la première phase des microinstructions. Il y a deux raisons principales pour cette décision :

---

* C'est ce mode de synchronisation que nous avons employé dans le réseau d'opérateurs adapté au traitement du signal exposé dans la deuxième partie de ce mémoire. On y considère la mise en oeuvre d'un automate particulier pour l'allocation du réseau aux différents groupes de tâches qu'il exécute.



Zéro = { 1 si Rs = 0 ; 0 autrement

Max = { 1 si Rs = Ns ; 0 autrement

Figure III.4. : CELLULE DES SEMAPHORES SYMETRIQUES S ET S'

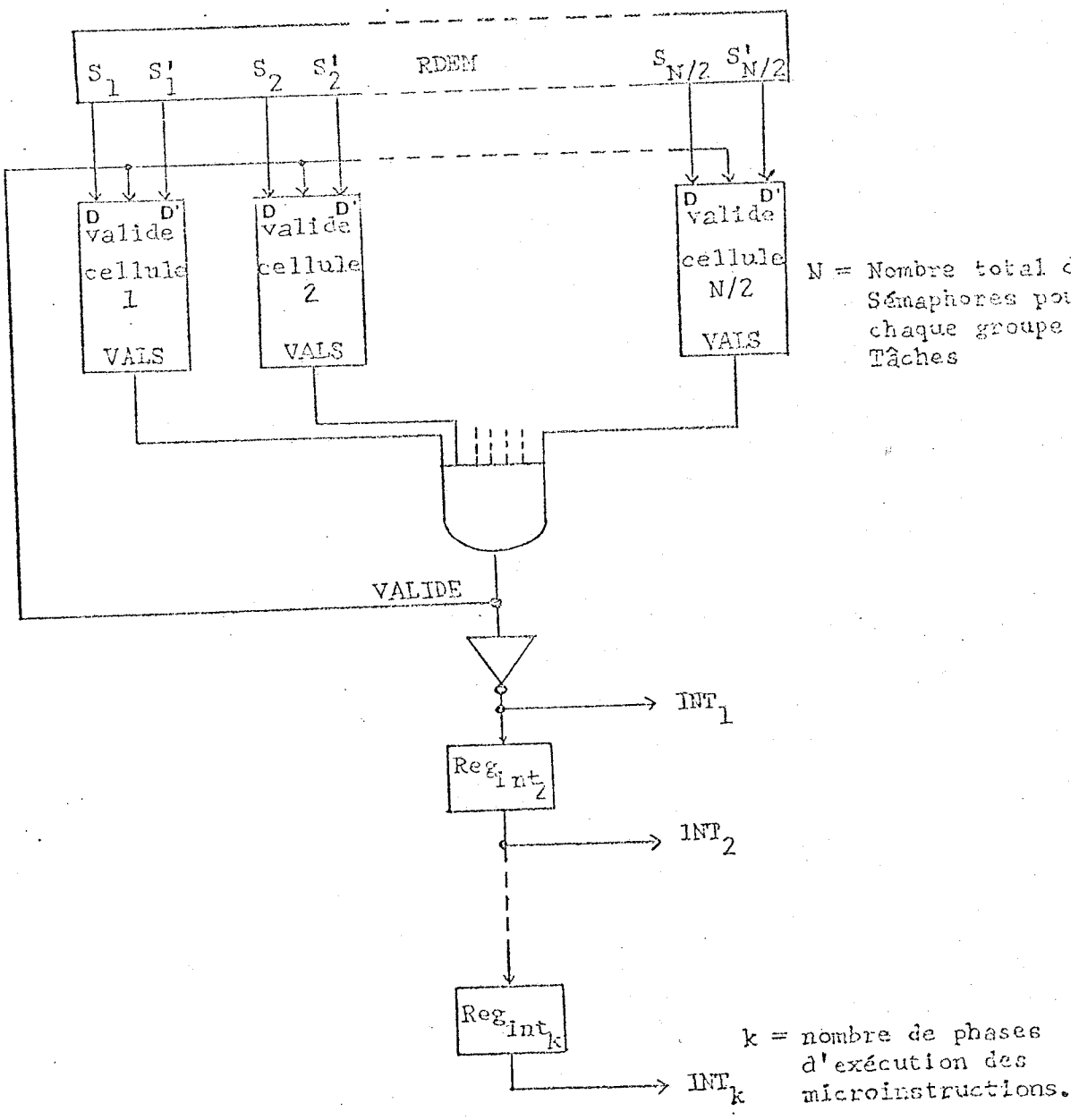


Figure III.5. : BLOC-DIAGRAMME DE L'AUTOMATE DE SYNCHRONISATION

- L'indivisibilité de l'exécution de SYNC est assurée, puisque dans le système il n'y a qu'un seul étage capable d'exécuter chacune des phases des microinstructions.
- Si l'ensemble de ressources demandées par SYNC n'est pas entièrement disponible, on ne doit pas exécuter les actions demandées par la micro-instruction en cours qui risquent de changer le contexte du système (c.f. II.2.a). Le fait d'exécuter les primitives SYNC dans la première phase, facilite ce travail : on produit le cas échéant, un signal INT qui accompagnera la microinstruction dans toutes les phases de son exécution pour interdire toutes les actions de ce type.

Etant donné qu'on dispose d'un automate spécifique pour le traitement de ces primitives, on peut considérer que toutes les microinstructions commandent l'exécution de SYNC (LSEM). Il suffit que la liste LSEM soit vide pour que la microinstruction n'ait pas à considérer les sémaphores.

La figure III.4. représente une cellule de deux sémaphores symétriques S et S'. Pour demander à la cellule s'il y a des ressources associées aux sémaphores S ou S' on force la valeur 1 dans les entrées D ou D' respectivement. En réponse à cette question, la sortie VALS prend la valeur 1 dans le cas affirmatif et 0 dans le cas contraire. En l'absence d'une question (D = D' = 0), la valeur de VALS est 1.

Si toutes les conditions pour exécuter une microinstruction ne sont pas satisfaites, le signal VALIDE prend la valeur 0, et le registre Rs reste inchangé, autrement il prend la valeur 1 et Rs varie conformément au tableau suivant :

D	D'	Nouvelle valeur de Rs	Opérations sur les sémaphores S et S'
0	0	Rs	Aucune
0	1	Rs + 1	Pp(S') ; Vp(S)
1	0	Rs - 1	Pp(S) ; Vp(S')
1	1	Rs	Pp(S,S');Vp(S,S')

La figure III.5. représente un automate à N/2 cellules pour exécuter l'ensemble de la synchronisation. Le registre RDEM comporte N bits (un bit par sémaphore). Pour l'exécution de la primitive SYNC (LSEM) on force, dans un premier temps, la valeur 1 dans les bits associés au sémaphores contenus dans cette liste (bits qui correspondent aux signaux D et D'), et on calcule la valeur du signal VALIDE (intersection logique de toutes les réponses VALS des cellules). Si ce signal vaut 1, on agit sur chacun de ces sémaphores suivant le tableau ci-dessus et on continue normalement l'exécution de la micro-instruction. Autrement, on n'autorise aucun changement des valeurs des sémaphores et on produit le signal INT (INT = VALIDE) qui accompagnera l'instruction dans toutes les phases de son exécution. Dans ce dernier cas, l'exécution de la microinstruction est différée tant que toutes les ressource qu'elle demande ne sont pas disponibles.

7. LES PRIMITIVES DE CHANGEMENT D'ETAT MOURIR ET CREER

Les passages d'une tâche de l'état suivant à l'état mort et vice-ve sont liés à la gestion dynamique du nombre de tâches dans le système. Or de même que pour la synchronisation, cette gestion peut et doit être effectuée p les tâches sans avoir besoin d'un moniteur ou d'un autre organe spécialisé quelconque.

Pour effectuer cette gestion on peut faire usage de deux primitives de changement d'état : MOURIR et CREER [11]. MOURIR sert à faire passer de l'état éveillé à l'état mort la tâche qui commande son exécution. La séquence suivante décrit les actions réalisées quand une tâche Ti commande l'exécution de la primitive MOURIR :

```
MOURIR ====> DEBUT Effacer le mot d'état de Ti
                   de la file de tâches réveillées;
                   Ne faire plus rien;
             FIN;
```

La primitive CREER permet de faire passer une tâche de l'état mort à l'état éveillé. Les actions réalisées lorsque une tâche Tj commande l'exécution de CREER(Tk) sont décrites par la séquence suivante :

CREER(Tk) ⟹ DEBUT Inscrire le mot d'état de
Tk dans la file des tâches éveillées;

FIN;

Avant de pouvoir commander l'exécution de cette primitive, il faut s'assurer que le contexte de Tk est défini (autrement, son mot d'état n'aurait pas de sens).

DEUXIEME PARTIE

- C H A P I T R E IV -

RESEAU D'OPERATEURS ADAPTE AU TRAITEMENT DU SIGNAL

Ce chapitre est consacré à la définition d'un réseau d'opérateurs - très performant - adapté au traitement du signal*, et d'une manière plus large, aux algorithmes dont les traitements sont très répétitifs et portent sur de grands blocs de données. Dans un premier temps on montre que l'utilisation d'un multiprocesseur permet un traitement efficace de ces algorithmes, puis on définit le réseau à partir du multi-microprocesseur à barillet "pipe-line" exposé au paragraphe II.2.b.

1. DECOMPOSITION DES ALGORITHMES EN TACHES

Il est courant dans les algorithmes utilisés pour le traitement de grands blocs de données, que la partie opérative soit souvent freinée par les lectures-écritures de données en mémoire principale et les entrées-sorties du réseau. Souvent, ces opérations peuvent être exécutées en parallèle, par suite nous avons pensé décomposer chacun de ces algorithmes en trois parties principales [15] :

- Lecture de données en mémoire principale
- Opérations sur ces données
- Rangement de données en mémoire principale et gestion des entrées-sorties

Bien que l'interaction entre ces trois parties soit très grande, pour les problèmes qui nous intéressent elle est de nature très régulière. Ainsi, on peut associer une tâche différente à l'exécution de chacune d'entre elles.

---

* Pour préciser les performances exigées, signalons par exemple que le réseau doit être capable d'exécuter de 20 à 25 fois par seconde l'algorithme de Transformée de FOURIER Rapide portant sur 1024 données complexes.

La figure IV.1. montre que, par rapport à la mémoire principale, ces trois tâches coopérent à l'exécution d'un algorithme suivant un schéma "pipe-line"*.

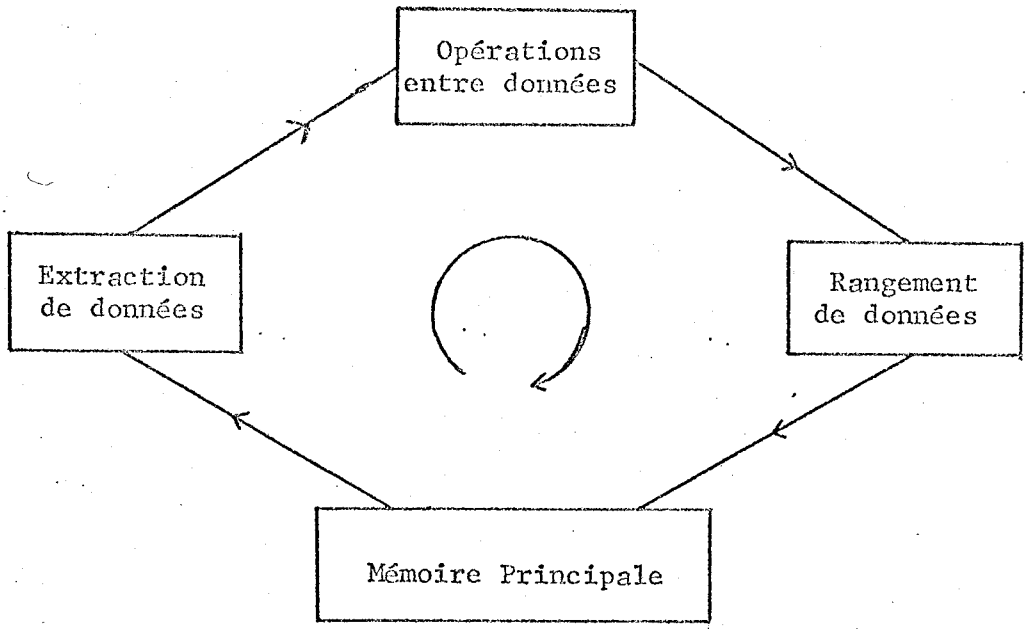


Figure IV.1. : FONCTIONNEMENT "PIPE-LINE" DU RESEAU

L'orientation de cette structure est nette : une exécution performante des trois tâches est obtenue si le "pipe-line" est parcouru par un flot continu de données. Ce fonctionnement n'est possible que pour les algorithmes qui impliquent des traitements réguliers, répétitifs et portant sur de grands blocs de données. Dans la mesure où l'on s'éloigne de ces hypothèses, le flot de données devient de plus en plus irrégulier et les performances du réseau se dégradent.

---

* Il ne faut pas confondre ce fonctionnement "pipe-line" où plusieurs tâches coopérent à l'exécution de chaque algorithme, avec celui exposé au paragraphe I.2. où plusieurs étages coopérent à l'exécution de chaque instruction.

2. LES MICROPROCESSEURS DU RESEAU. STRUCTURE DE BASE

Pour parvenir à une grande puissance de calcul, les trois tâches définies au paragraphe précédent doivent être exécutées simultanément; le réseau dispose de trois microprocesseurs - EXT (extraction), OP (opérateur) et RAN (rangement) - affectés à chaque tâche. Etant donnée la nature de ces tâches, les microprocesseurs sont complémentaires (c.f. I.1.) : il existe des ressources propres à chacun d'eux. Le tableau suivant résume leurs principales attributions :

EXT : - Gestion de la lecture des données en mémoire principale
- Gestion des appels externes (interruptions)
- Gestion des alarmes internes (p. ex. les erreurs)

OP : - Opérations entre données
- Gestion de certains opérateurs spécialisés (p. ex. Multiplieur)

RAN : - Gestion de l'écriture des données en mémoire principale
- Ecriture de micro-instructions en mémoire de commande
- Gestion des entrées-sorties

La figure IV.2. représente un bloc-diagramme du réseau avec ses trois microprocesseurs : EXT et RAN commandent toutes les lectures/écritur de données en mémoire principale et les entrées sorties du réseau, mais se le microprocesseur OP a directement accès à ces données.

La structure de base pour réaliser ces trois microprocesseurs es un multi-microprocesseur en barillet "pipe-line" à trois étages. Il est composé d'une unité d'adressage de la mémoire de commande : UAC, et d'une unité pour l'exécution des micro-instructions : UEM (c.f. II.2.b). Cette organisation nous permet de définir un microprocesseur performant :

- Chacun des microprocesseurs peut exécuter simultanément une micro-instruction.

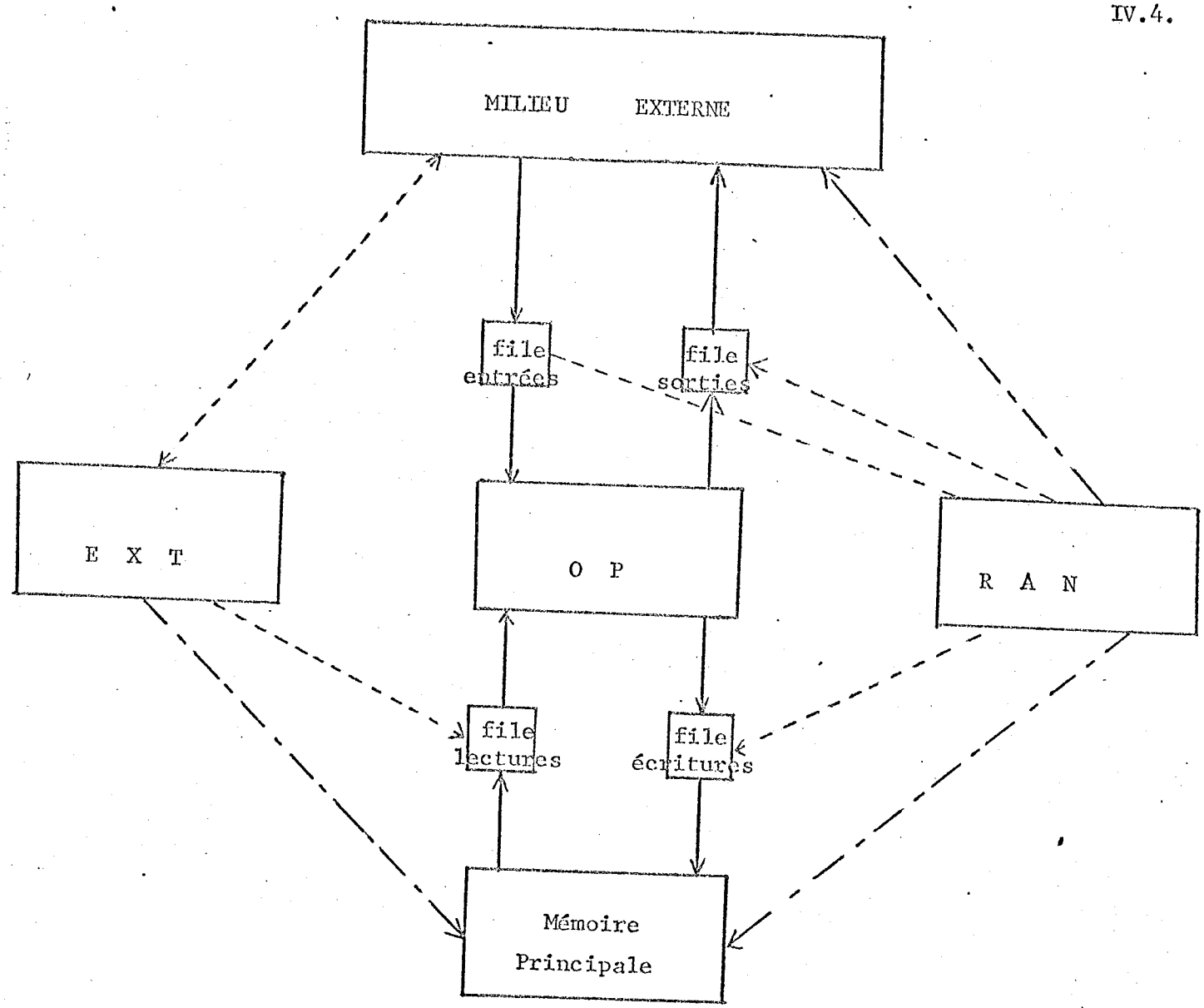


Figure IV.2. : BLOC DIAGRAMME DU RESEAU

- Les temps de calcul des pointeurs et de lecture des micro-instructions sor masqués par ceux nécessaires à l'exécution de ces micro-instructions.

D'autre part, cette solution nous semble économique :

- Le chemin de données, l'unité arithmétique et logique et la mémoire de commande avec son mécanisme d'adressage, sont communs aux trois microprocesseurs.
- L'ordonnateur au niveau étages - microprocesseurs est très simple : l'attribution des étages aux microprocesseurs est cyclique.

Etant donné le caractère complémentaire de EXT, OP et RAN, il y a peu d'algorithmes susceptibles d'être exécutés par un seul de ces microprocesseurs. D'autre part, les mécanismes nécessaires à une gestion indépendante des microprocesseurs seraient complexes. C'est pour ces deux raisons que nous avons limité le fonctionnement de EXT, OP et RAN à l'exécution d'un seul algorithme à la fois. Lors d'appels externes ou d'alarmes internes, les trois microprocesseurs différent simultanément l'exécution de l'algorithme Ai auquel ils étaient affectés pour se consacrer ensemble à l'exécution de l'algorithme Aj impliqué par l'appel externe ou l'alarme interne en question. De même, une fois fini le traitement de Aj, ils reprennent en même temps l'exécution de Ai.

La figure IV.3. illustre le diagramme multiniveau qui correspond au réseau d'opérateurs. Au premier niveau de cet arbre, on trouve le matéri employé pour les trois étages E1, E2 et E3. Au deuxième niveau on a les tro phases exécutées à tour de rôle par ce matériel. Au troisième niveau sont représentés les trois microprocesseurs du réseau. Chaque microprocesseur demande le concours séquentiel des trois phases pour exécuter ses micro-instructions.

Enfin, au sommet de cet arbre on trouve les algorithmes à traiter par le réseau. Pour ces algorithmes, les trois microprocesseurs EXT, OP et RAN forment un tout indissociable.

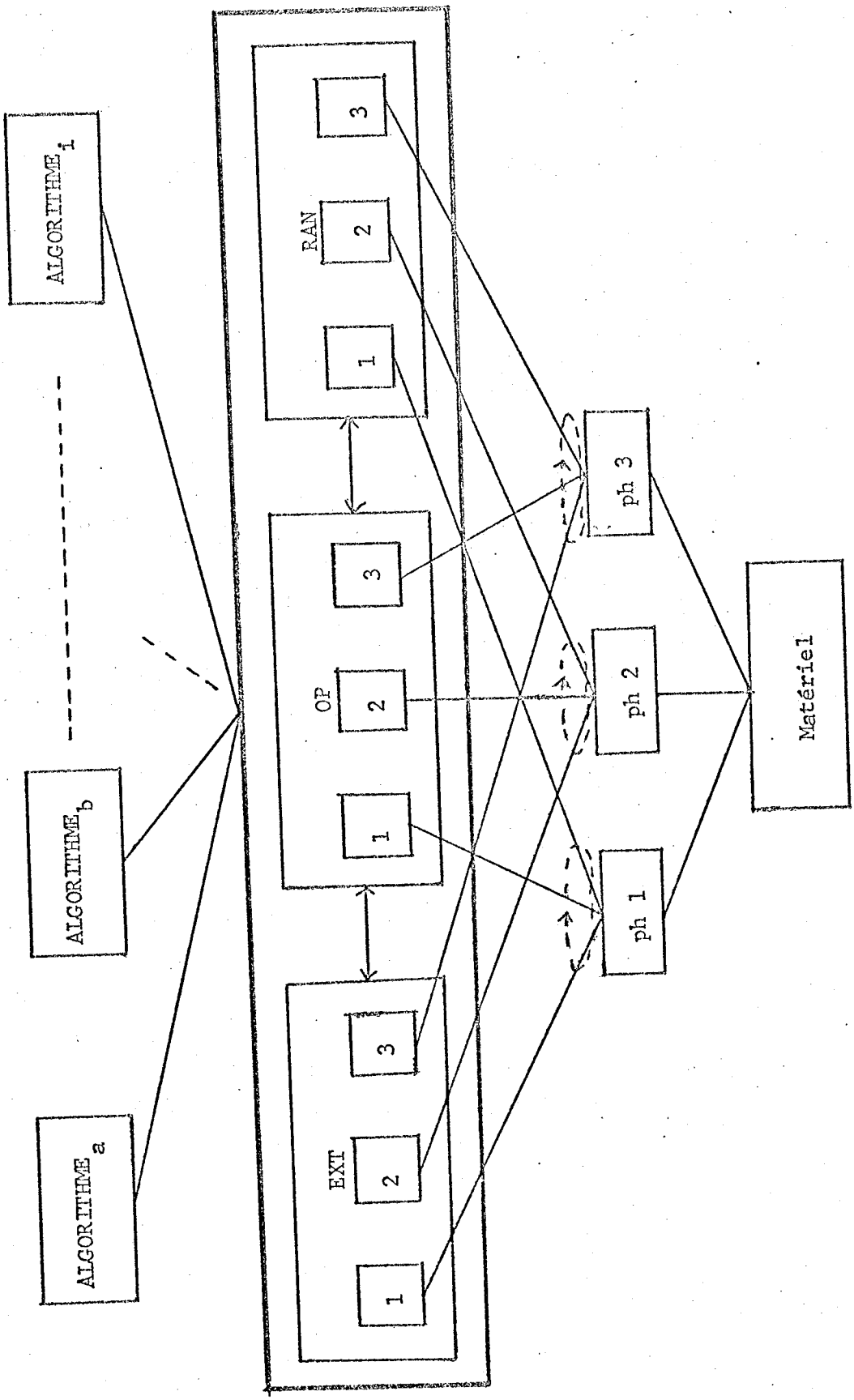


Figure IV.3. : DIAGRAMME MULTI-NIVEAU DU RESEAU

REMARQUE : De même que pour chacun de ses microprocesseurs, on peut considérer que le réseau travaille en trois phases différentes ph-R1, ph-R2 et ph-R3*. Le diagramme de la figure IV.4. donne le relations entre chacune des phases du réseau et les phases d'exécution des microprocesseurs EXT, OP et RAN.

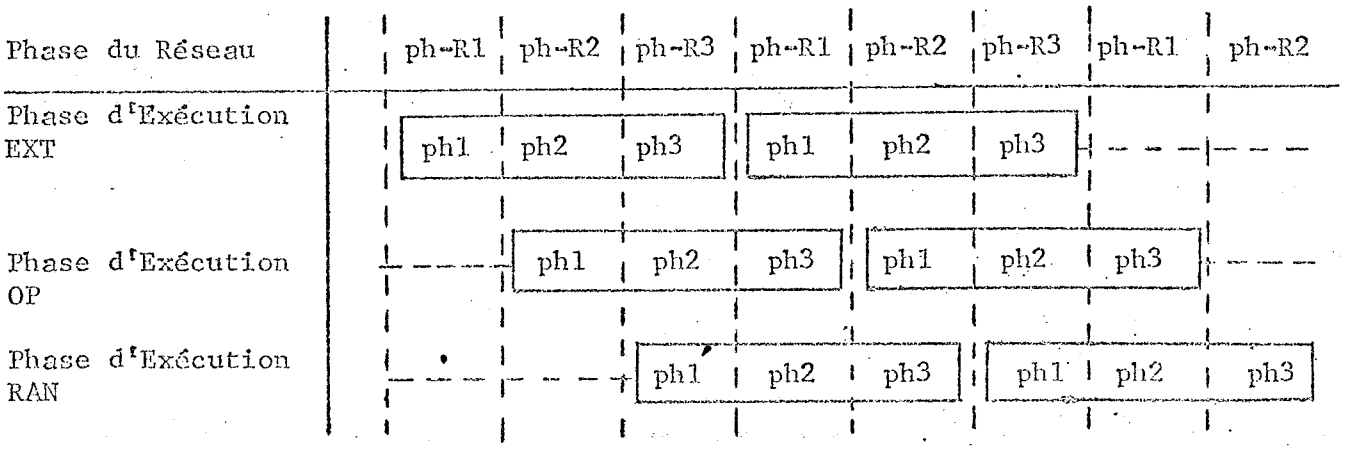


Figure IV.4. : RELATIONS ENTRE LES PHASES DU RESEAU ET LES PHASES D'EXECUTION DES MICROPROCESSEURS.

3. LES UNITES D'ADRESSAGE DE LA MEMOIRE DE COMMANDE - UAC - ET D'EXECUTION DES MICROINSTRUCTIONS - UEM -

Dans ce paragraphe nous allons décrire le fonctionnement des unit UAC et UEM et leur découpage en trois étages, condition nécessaire au bon fonctionnement de notre multi-microprocesseur en barillet "pipe-line" (c.f. II.2.b).

---

* Par la suite, cette définition sera utile pour clarifier la description d fonctionnement du réseau, en particulier pour les appels externes et les alarmes internes.

a) Unité UAC : mémoire de commande et enchaînement des micro-instructions

Le contrôle des trois microprocesseurs du réseau est assuré par une seule mémoire de commande (micro-mémoire). Elle est composée de 512 mots de 30 bits : 256 mots de mémoire permanente (dont les adresses s'étalent entre 0 et 255) et 256 mots de mémoire inscriptible (256 à 511). Chacun des microprocesseurs dispose d'un pointeur de 9 bits ($2^9 = 512$) pour l'adressage de cette mémoire. Dans la première phase d'exécution d'une micro-instruction, le pointeur contient l'adresse de celle-ci, dans la dernière, il contient l'adresse de la micro-instruction à exécuter dans le cycle suivant.

Les seuls cas où on ne respecte pas un enchaînement séquentiel des micro-instructions traitées par chaque microprocesseur, sont la conséquence de :

- L'exécution des micro-instructions comportant un branchement relatif. L'adresse de la micro-instruction suivante est le résultat de l'addition de l'adresse courante et d'une constante binaire-positive ou négative représentée en complément à deux-contenus, le cas échéant, dans les bits 19 à 25 de la micro-instruction en cours.
- L'exécution de la micro-instruction BRANCHEMENT (branchement absolu inconditionnel); l'adresse courante est substituée par les 9 bits poids faibles du premier opérande relatif à cette micro-instruction. Si cette micro-instruction comporte un branchement relatif, l'adresse de la micro-instruction suivante sera calculée comme indiqué ci-dessus, autrement elle sera égale à la nouvelle adresse courante.
- L'exécution de certaines micro-instructions spéciales (RETOUR, RETOUR-RU) pour la gestion de l'allocation du réseau (c.f. IV.5. et IV.7.).
- La prise en compte des appels externes et de l'alarme interne (c.f. IV.5.)

b) Unité UEM : Chemin de données et Unité arithmétique et logique

Le réseau opère avec des mots de 16 bits numérotés 0 à 15. Quand ils contiennent des nombres dans la représentation normale : complément à deux, le signe est le bit 0, les bits 1 à 15 sont, dans cet ordre, les poids binaires décroissants.

Le chemin de données dispose de trois bus de 16 bits pour le transferts de données; deux bus sources : premier opérande - S1(0 : 15 et deuxième opérande - S2(0 : 15) -, et un bus destination : troisième opérande - D(0 : 15) * -. Les bus source sont reliés au bus destination l'intermédiaire d'une unité arthmétique et logique : UAL, composée d'u opérateur arithmétique et logique : OPAL, et un modificateur (figure I

---

* Presque tous les éléments du réseau sont branchés entre les bus sour et le bus destination (les registres généraux et spécifiques, la mém principale, etc), et de ce fait peuvent être directement adressables les microinstructions. C'est ainsi que des opérateurs spécialisés pe être incorporés au réseau d'une manière très naturelle : leur exploi est faite par échanges d'information au moyen des adresses source et destination attribuées à chacun d'eux.

Cette caractéristique est très souhaitable dans un système traitement du signal où un certain nombre d'algorithmes en temps rée peuvent être traités correctement qu'à l'aide des opérateurs spécia

L'unité UAL est commandée directement par certaines microinstructions ou encore par l'intermédiaire d'un registre spécial : RCODE (0 : 15), accessible aux trois microprocesseurs aussi bien en lecture qu'en écriture. L'unité UAL opère sur deux données A et B de 17 bits chacune, matérialisées respectivement par S1 (0) & S1 (0 : 15) et S2 (0) & S2 (0 : 15) (le bit du signe est répété dans les deux cas). L'opérateur OPAL peut exécuter 8 opérations sur ces données : 4 arithmétiques (A + B, A - B - opérations arithmétiques simples - A + B + RET et A - B - $\overline{\text{RET}}$ - opérations arithmétiques enchaînées **) et 4 logiques (A et B, A ou B, A conjonction B et B). Le résultat de ces opérations est un nombre de 17 bits - SOPAL (0 : 16) - dont les bits 0 et 1 ont la même valeur sauf dans les cas des opérations arithmétiques provoquant un débordement.

Le modificateur est constitué par un sélecteur et un registre à décalage - EXTENSION (0 : 15) - accessible aux trois microprocesseurs aussi bien en lecture qu'en écriture. Le sélecteur choisi une certaine configuration de 16 bits prélevés parmi les bits 0 à 16 du signal SOPAL et les bits 0 et 15 du registre EXTENSION, par exemple SOPAL (1 : 16) - configuration standard -, SOPAL (0 : 15) - décalage arithmétique droit -, SOPAL (2 : 16) & EXTENSION (0) - décalage logique gauche -, etc. Parallélement à certains de ces choix, le registre EXTENSION est décalé d'une position à gauche ou à droite.

---

* S1 (0) représente le bit 0 de l'être binaire S1 (0 : 15); S1 (0) & S1 (0 : 15) représente la caractérisation du bit S1 (0) et des 16 bits S1(0 : 15), c'est-à-dire, un être de 17 bits dont les deux de poids le plus fort ont toujours la même valeur.

** RET - retenue d'une opération précédente - est l'un des paramètres contenu dans le registre code condition propre à chaque processeur du réseau (voir plus loin), ainsi, les deux opérations A + B + RET et A - B - $\overline{\text{RET}}$ permettent les enchaînements nécessaires pour les additions ou soustractions de nombres dont la longueur dépasse 16 bits.

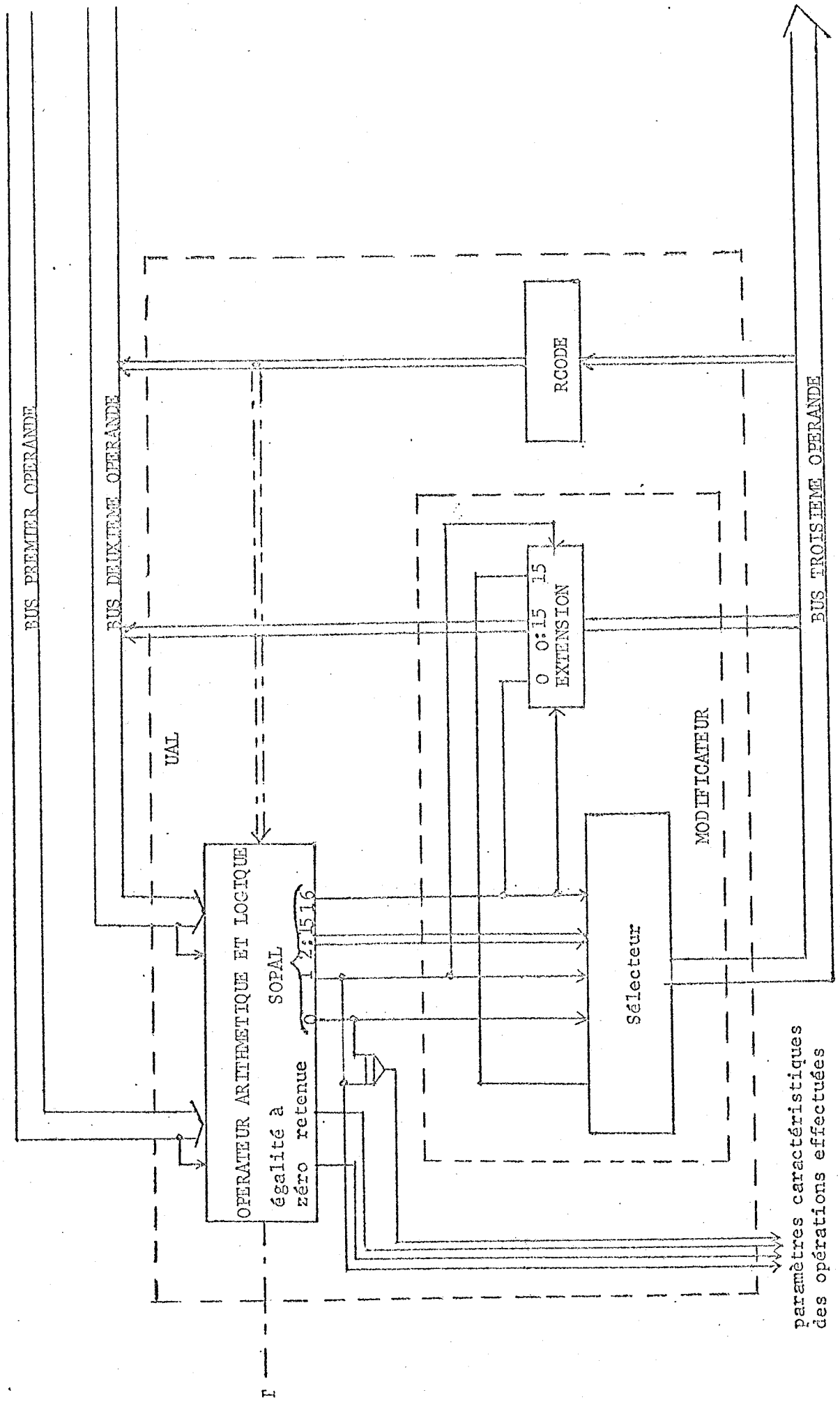


Figure IV.5. : LE CHEMIN DES DONNEES

Les résultats des opérations arithmétiques et logiques sont caractérisés par 4 paramètres : signe du résultat : SOPAL (1); égalité à zéro : les bits 1 à 16 de SOPAL sont tous égaux à zéro; débordement : SOPAL (0) disjonction SOPAL (1); et retenue. Ces deux derniers paramètres ont une valeur zéro pour toutes les opérations logiques. Ces 4 paramètres peuvent être utilisés, grâce à une commande de la micro-instruction, pour effectuer la mise à jour d'un registre de code de condition : RCC (0 : 3), particulier à chacun des microprocesseurs. C'est sur le contenu de ces registres, qui portent la plupart des tests effectués lors des branchements relatifs conditionnels. En particulier, le bit 3 de ces registres (bit qui correspond à la retenue) est employé pour les opérations arithmétiques enchaînées.

Le travail de l'UAL est déterminé par deux codes :

- Le code opération, spécifiant le travail à effectuer par l'OPAL
- Le code d'options, spécifiant :
  - Le choix du modificateur
  - La mise au jour du registre RCC
  - La non activation de l'alarme interne en cas de débordement

Seules les micro-instructions arithmétiques et logiques en ligne, utilisent l'UAL. A une exception près : la micro-instruction EXECUTE qu'on étudiera plus loin, le code d'opération de l'UAL est spécifique à chacune de ces micro-instructions, le code d'options pouvant être donné explicitement; si ce n'est pas ainsi, l'UAL assume le code d'options par défaut, c'est-à-dire :

- Activation de l'alarme interne en cas de débordement
- Le modificateur transfère vers le bus destination les bits 1 à 16 de la sortie de l'opérateur OPAL. Le registre EXTENSION n'est pas modifié.
- Le registre RCC n'est pas modifié

Durant l'exécution des micro-instructions, le chemin de données est utilisé de la manière suivante * :

- Le bus source premier opérande ne peut recevoir qu'une valeur constante contenue, le cas échéant, dans les bits 15 à 29 des micro-instructions (le bit 15 - le poids le plus fort -, est répété à gauche pour obtenir 16 bits nécessaires), ou bien le contenu de l'un de 16 registres généraux dont dispose le réseau; dans ce cas, l'adresse du registre es contenue dans les bits 26 à 29 des micro-instructions.
- Le bus deuxième opérande peut recevoir le contenu de l'un des 16 regist généraux ou bien celui de 16 autres éléments accessibles à chaque microprocesseur, et qui peuvent être : soit des registres communs aux t microprocesseurs (p. ex. les registres EXTENSION et RCODE), soit des registres qui sont propres à chacun d'entre eux, soit finalement des opérateurs spécialisés (p. ex. la file des données lues en mémoire principale - accessible en lecture au microprocesseur OP -).
- Le contenu du bus destination peut être rangé dans l'un des 16 registre généraux ou bien dans l'un des 16 autres éléments : registres spécifiqu opérateurs spécialisés, etc.

Le cas échéant, les adresses du deuxième et troisième opérande sont respectivement contenues dans les bits 9 à 13 et 4 à 8 des micro-instructions.

c) Découpage en étages des unités UAC et UEM

La définition du triprocesseur choisi exige, on l'a vu, un découpage en trois étages des unités UAC d'une part et UEM d'autre part (c.f. II.2.a). Les trois étages de la première de ces unités, exécutent l travaux suivants :

Etage 1 - Calcul des tests pour les branchements relatifs conditionnels (Les branchements relatifs inconditionnels sont traités de la m manière que les conditionnels, mais on considère la condition toujours vérifiée).

---

* Le chemin de données est aussi employé, pendant les opérations d'alloca du réseau, pour les transferts des mots d'état (c.f. IV.5.).

Etage 2 - Calcul de l'adresse de la nouvelle micro-instruction

Etage 3 - Lecture des micro-instructions

Les figures IV.6. et IV.7. représentent respectivement le bloc-diagramme et l'organigramme simplifiés de l'unité UAC. Les registres temporaires RPOINT (0 : 8, 1), RPOINT (0 : 8, 2) et RPOINT (0 : 8, 3) contiennent à tour de rôle les pointeurs microprogramme des microprocesseurs EXT, OP et RAN. A titre d'exemple, considérons le déroulement d'une micro-instruction relative à l'un de ces microprocesseurs.

Pendant la première phase d'exécution, on réalise parallèlement cinq opérations :

- Chargement du registre temporaire RPOINT (0 : 8, 2) avec la valeur du pointeur contenu dans RPOINT (0 : 8, 1).
- Chargement du registre temporaire RPO (7 : 15) avec les bits 7 à 15 du bus source premier opérande - valeur éventuelle du branchement absolu -.
- S'il s'agit d'une micro-instruction BRANCHEMENT, le registre temporaire RCSAUT prend la valeur 1, autrement il prend la valeur 0.
- Repérage des branchements relatifs. Dans l'affirmative, le registre temporaire RTEST prend la valeur 1, autrement il prend la valeur 0.
- Chargement du registre temporaire RDEP (2 : 8) avec la valeur éventuelle du branchement relatif (bits 19 à 25 du registre RMI).

Le pointeur de la micro-instruction suivante est déterminé dans la deuxième phase d'exécution, compte tenu des valeurs des registres RTEST et RCSAUT ainsi que du signal INT(1) et suivant l'organigramme représenté dans la figure IV.7. Le signal INT(1) vaut 0 si l'exécution de la micro-insrtruction peut continuer normalement, autrement il vaut 1.

Pendant la troisième phase, on exécute parallèlement deux opérations :

- La micro-instruction de la mémoire de commande dont l'adresse est le contenu de RPOINT (0 : 8, 3), est forcée dans le registre temporaire RMI (0 : 29).
- Chargement de cette adresse dans le registre temporaire RPOINT (0 : 8, 1)

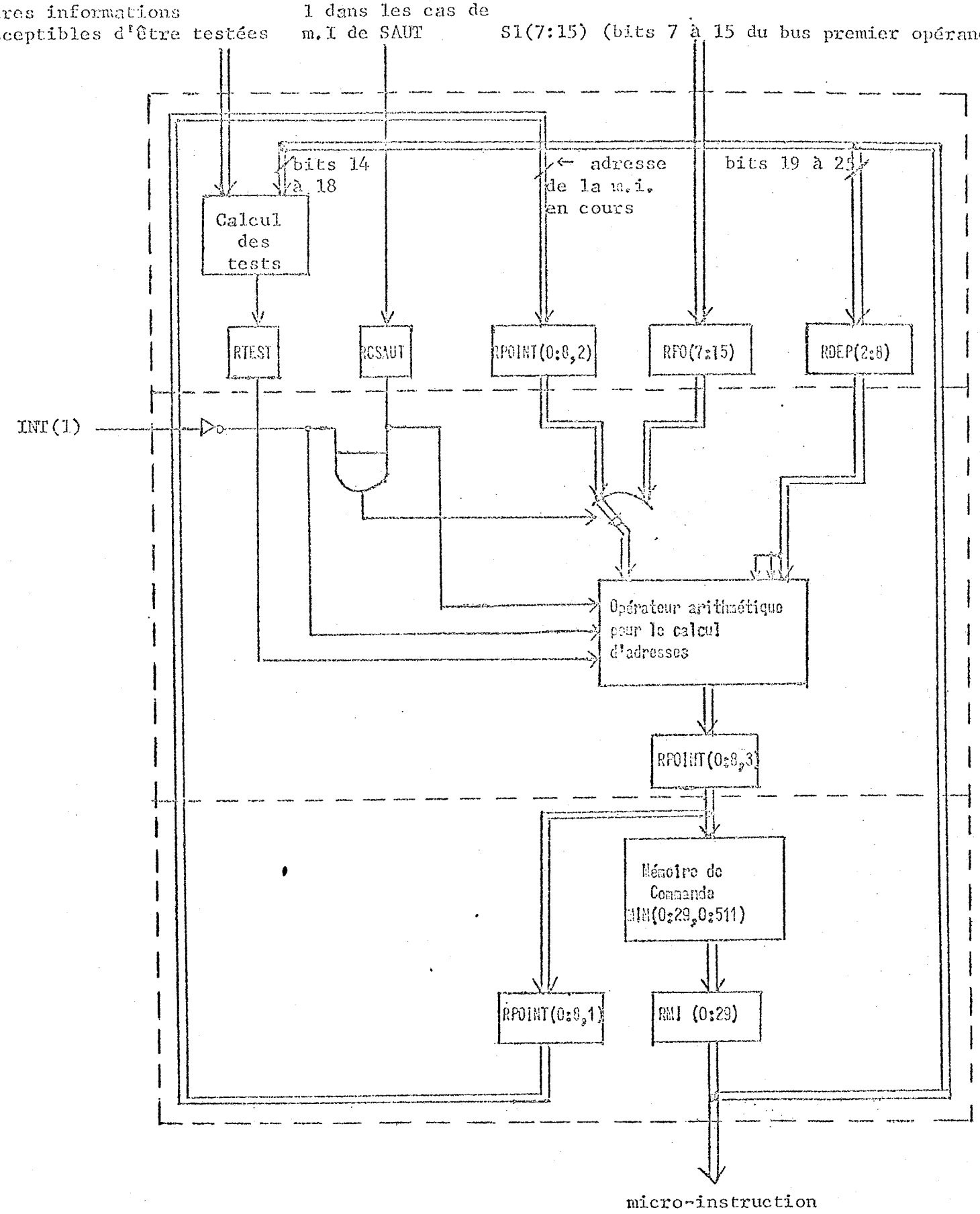
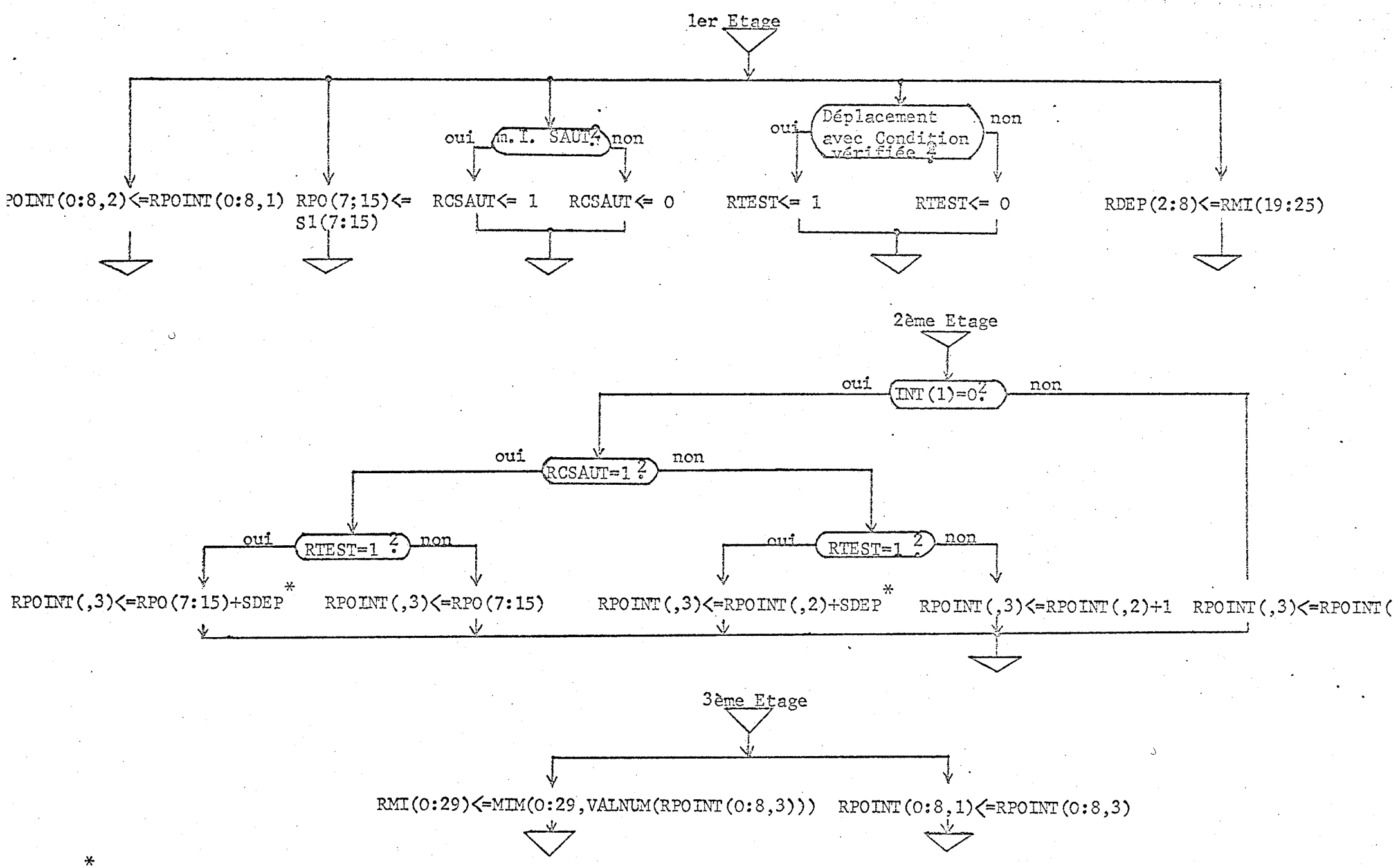


Figure IV.6. : BLOC-DIAGRAMME SIMPLIFIE DE L'UNITE UAC



* SDEP=RDEP(2)&RDEP(2)&RDEP(2:8)

Figure IV.7. : ORGANIGRAMME SIMPLIFIE DU FONCTIONNEMENT DE L'UNITE UAC

Les trois étages de l'unité UEM se chargent des actions suivantes

Etage 1 - Décodage de la micro-instruction

- Détermination du type de la micro-instruction
- Détermination, le cas échéant des codes d'opération et d'options de l'UAL

  - Exécution des primitives de changement d'état SYNC et ISYNC
  - Recherche des opérandes source pour leur transfert vers l'UAL ou bien vers un autre organe* à travers les bus premier et deuxième opérandes.

Etage 2 - Exécution des opérations éventuellement commandées à l'opérateur OPAL.

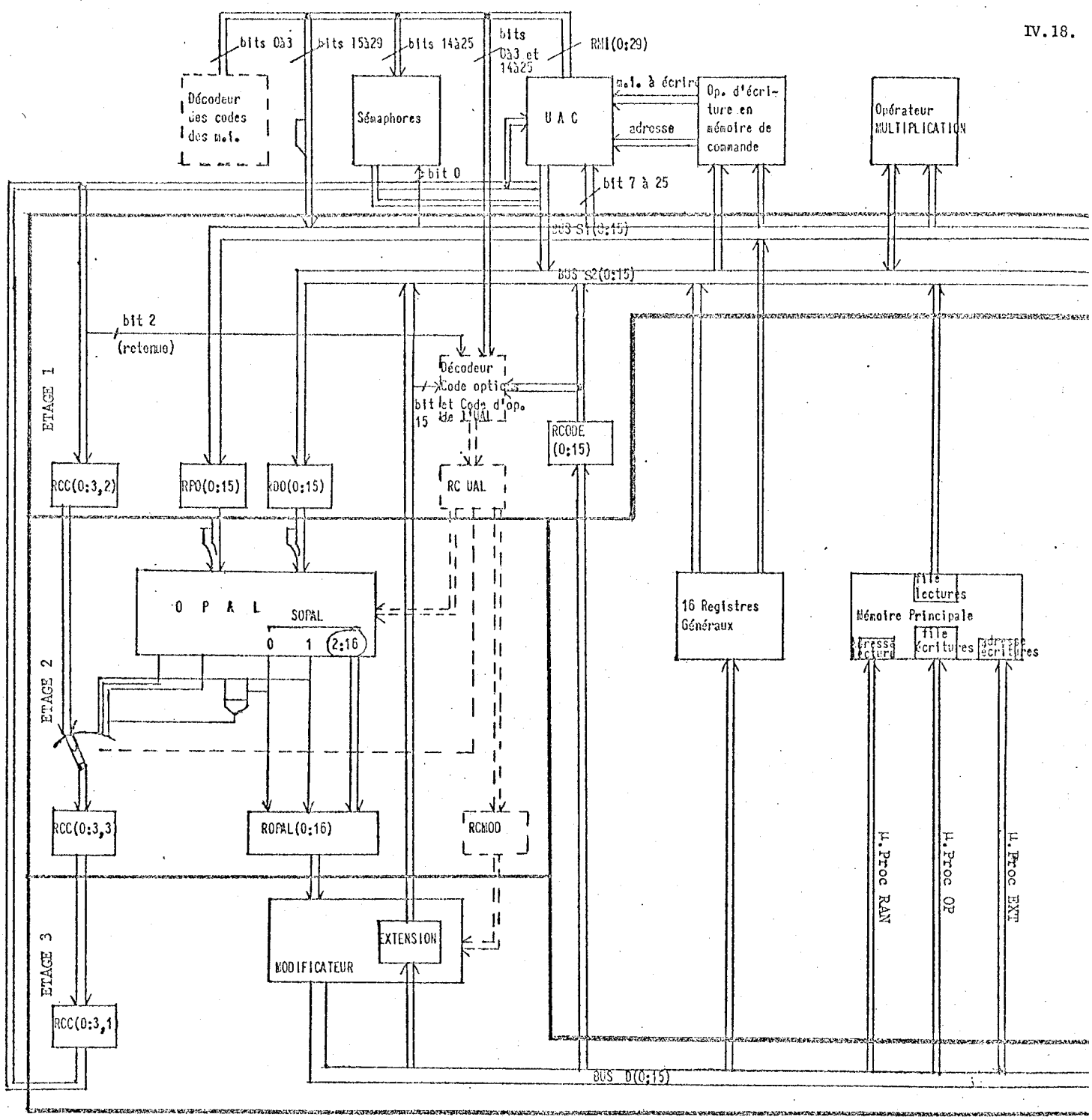
Etage 3 - Modification des résultats de l'OPAL pour leur transfert, à travers le bus destination, vers l'un des 16 registres généraux, ou l'un des opérateurs spécialisés ou bien, tout autre élément idoine.

  - Mise à jour, le cas échéant, du registre de code de condition propose au microprocesseur concerné.

La figure IV.8. représente le bloc diagramme simplifié de l'UEM. Les registres temporaires RCC(0 : 3,1), RCC(0 : 3,2) et RCC(0 : 3,3) ainsi que RPO(0 : 15), RDO(0 : 15) et RSOPAL(0 : 15) contiennent, à tour de rôle respectivement les codes de condition, les premier et deuxième opérande et les résultats des éventuelles opérations arithmétiques ou logiques. Le tableau suivant résume l'attribution de ces registres dans chacune des ph du réseau :

---

* Hormi l'unité UAL, seule l'unité UAC - dans les cas des micro-instructi BRANCHEMENT ou bien d'allocation du réseau - et les opérateurs spéciali pour effectuer les multiplications et l'écriture dans la mémoire de commande, peuvent recevoir des données à travers les bus source. Tous l autres le font à travers le bus résultat.



-------- Eléments de contrôle

Figure IV.8. : BLOC DIAGRAMME SIMPLIFIE DE L'UEM

	RCC (0:3,1)	RCC (0:3,2)	RCC (0:3,3)	RPO (0:15) et RDO (0:15)	ROPAL (0:15)
Ph-R1	EXT	RAN	OP	EXT	RAN
Ph-R2	OP	EXT	RAN	OP	EXT
Ph-R3	RAN	OP	EXT	RAN	OP

4. SYNCHRONISATION

Pour la synchronisation entre les tâches exécutées par le système, nous avons adopté un mécanisme de sémaphores et primitives de changement d'état parallèles similaire à celui exposé au paragraphe III.6. Il met en oeuvre trois paires de sémaphores symétriques A-A', B-B' et C-C'. Le registr associé à chacune des paires ne peut prendre que les valeurs 0 et 1.

On dispose de deux primitives de changement d'état :

- SYNC(LSEM) dont le fonctionnement est décrit au paragraphe III.5.a, mais dans le cas présent, il est impossible que la liste LSEM contienne deux sémaphores appartenant à une même paire. (Une tâche qui implique l'exécuti de SYNC (... S, S',...) sera bloquée définitivement : les sémaphores symétriques S et S' ne pourront jamais être simultanément supérieurs à zér En effet, si S a la valeur 1, S' a forcément la valeur zéro et vice-versa)
- ISYNC(LSEM) qui se traduit par l'exécution indivisible de la suite des primitives SYNC(LSEM); SYNC(LSEM') (LSEM' est la liste de sémaphores symétrique de LSEM). L'exécution d'une tâche qui commande la primitive ISYNC(LSEM) est différée tant que tous les sémaphores impliqués ne sont pa supérieurs à zéro, mais une fois cette condition satisfaite, la tâche peut continuer son travail sans que la valeur des sémaphores soit modifiée.

Le traitement de ces primitives, on l'a vu, doit être fait lors de la première phase d'exécution des micro-instructions. Les programmes d'application proposés plus loin, illustrent l'utilisation de SYNC et ISYNC.

La figure IV.9. représente le bloc diagramme de l'automate de synchronisation utilisé. Les registres RSEM(1), RSEM(2) et RSEM(3) contiennent à tour de rôle la valeur caractérisant les paires de sémaphores A-A', B-B' et C-C' * :

	RSEM(1)	RSEM(2)	RSEM(3)
Ph-R1	A - A'	B - B'	C - C'
Ph-R2	B - B'	C - C'	A - A'
Ph-R3	C - C'	A - A'	B - B'

Les signaux DS et TS employés dans cette figure sont liés aux signaux D et D' de la figure III.4. par les relations suivantes :

$$D = DS \cdot TS$$

$$D' = DS \cdot \overline{TS}$$

---

* Cette circulation des sémaphores a pour but de faciliter leur sauvegarde lors de la prise en compte des appels externes ou de l'alarme interne (c.f. IV.5.).

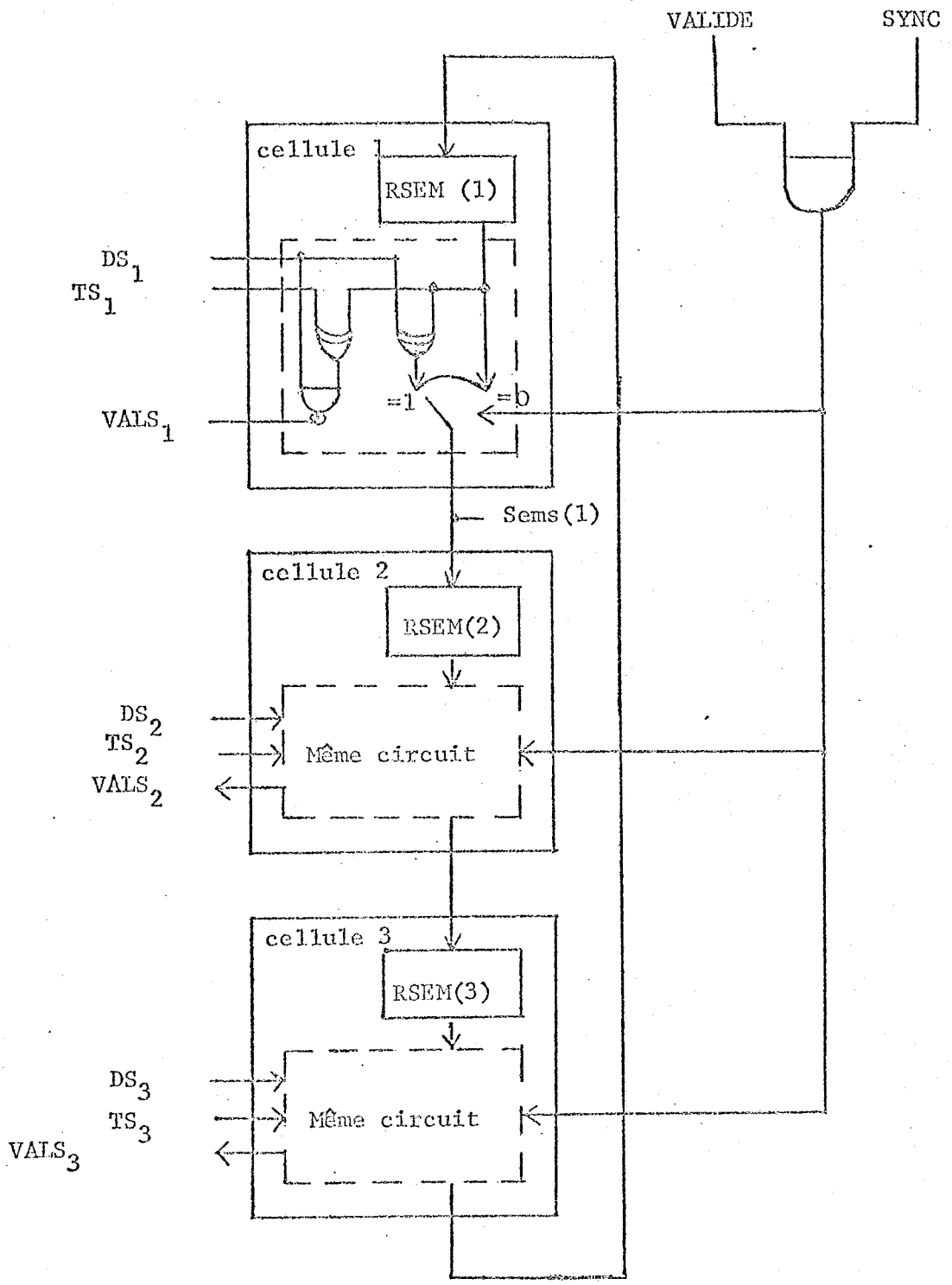
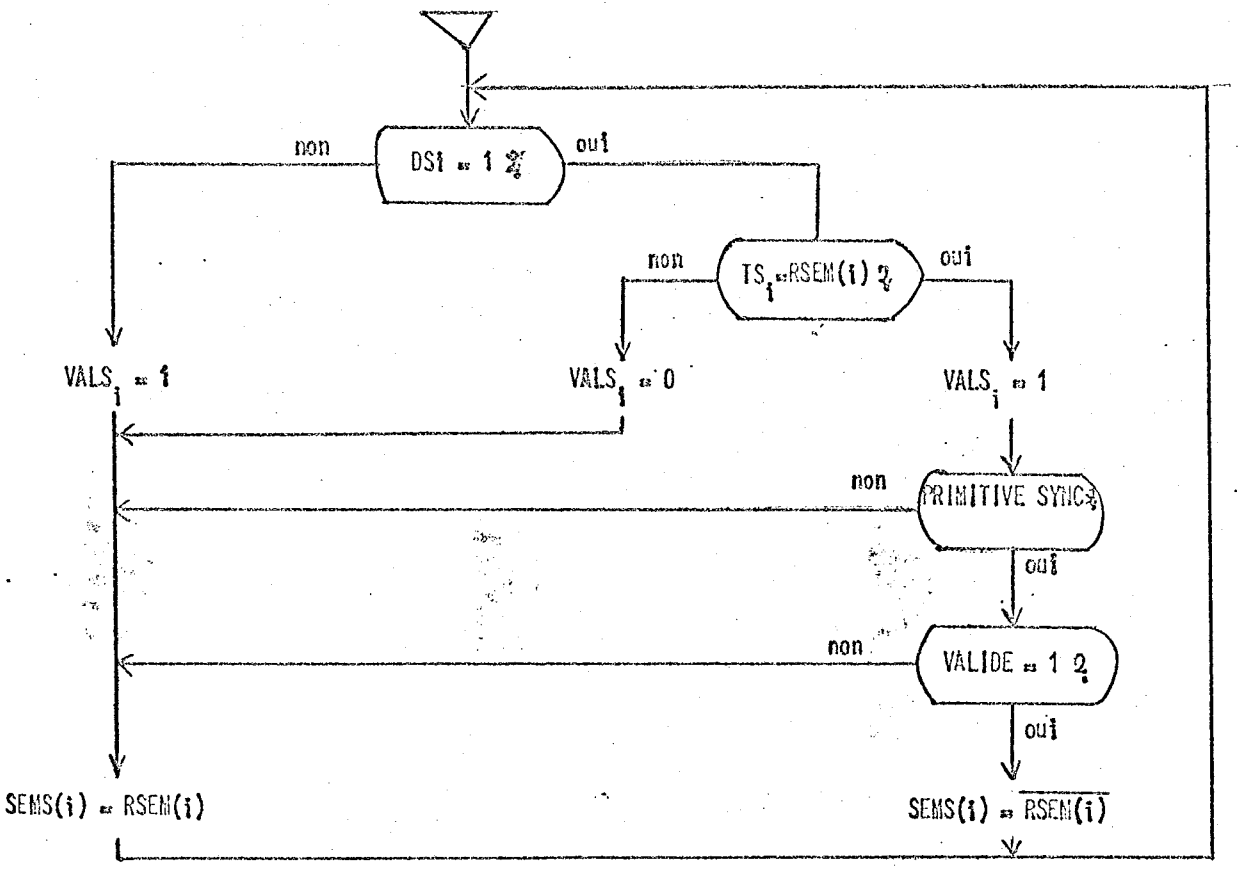


Figure IV.9. : BLOC DIAGRAMME DE L'AUTOMATE DE SYNCHRONISATION

Le signal DSi prend la valeur 1 si à un instant donné on essaye d'exécuter une primitive SYNC ou ISYNC portant sur la paire de sémaphores S-S' contenue dans le registre RSEM(i). Si TSi = 1, la primitive porte sur S, autrement sur S'. Le fonctionnement de chaque cellule est décrit par l'organigramme suivant :



Un autre problème de synchronisation est lié à certaines micro-instructions dont l'exécution demande le concours d'une ou plusieurs ressources physiques. Nous entendons par ressource physique :

- soit un opérateur spécialisé pour l'exécution d'un travail spécifique

- soit le résultat (ou les résultats) de l'un de ces travaux *.

L'exécution de ces microinstructions n'est pas seulement conditionnée par une éventuelle synchronisation entre tâches, mais aussi p la disponibilité des ressources physiques qu'elle implique. De manière formelle, la gestion de ces ressources est effectuée à un niveau inférieur celui des primitives SYNC et ISYNC, néanmoins pour assurer cette gestion nous avons profité d'une partie du mécanisme utilisé pour les sémaphores parallèles.

La figure IV.10. illustre le circuit employé pour la gestion de chacune des ressources physiques. Lors de la première phase d'exécution d' micro-instruction qui demande le concours de la ressource physique Rx, le signal Dx est forcé à la valeur 1. Si Rx est disponible, le signal VALRx conserve la valeur 1, autrement il prend la valeur 0. Pour indiquer - le c échéant - à l'opérateur concerné la prise effective d'une ressource, les signaux Dx et VALIDE doivent prendre simultanément la valeur 1.

La figure IV.11. illustre l'automate assurant la gestion de l'ensemble des sémaphores et ressources physiques. Son fonctionnement est semblable à celui de l'automate représenté figure III.5. mais signalons qu le signal VALIDE est ici le résultat de l'interaction des signaux VALS (réponses des sémaphores) et VALR (réponses des opérateurs spécialisés). S VALIDE = 1 l'exécution de la micro-instruction est possible, autrement cet exécution doit être différée (c.f. III.6.).

---

* Par exemple, pour que le microprocesseur EXT puisse exécuter une micro-instruction de lecture en mémoire principale, il faut que l'opérateur c lecture soit disponible, c'est-à-dire qu'il ne doit pas être en train d'exécuter une autre lecture, et que la file de lectures (figure IV.2.) ne soit pas pleine. Si l'opérateur est libre, EXT pourra exécuter la micro-instruction, sinon son exécution sera différée.

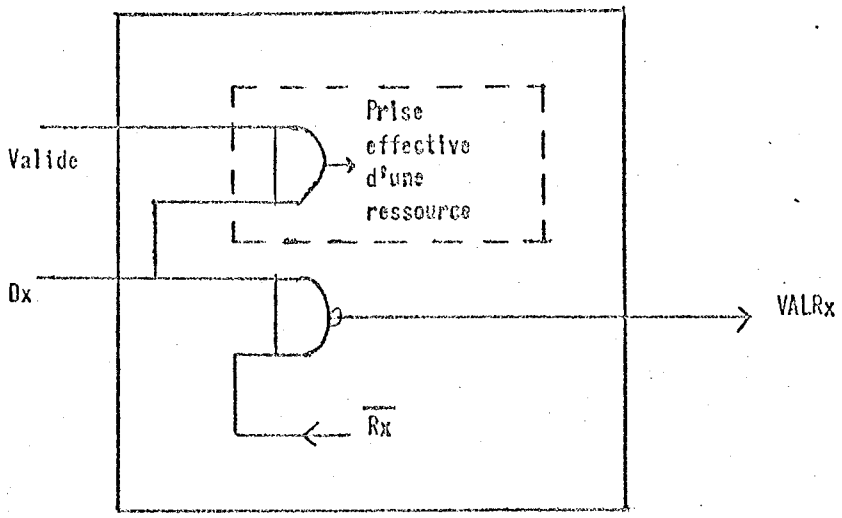


Figure IV.10. : CIRCUIT ASSOCIE A LA GESTION D'UNE RESSOURCE

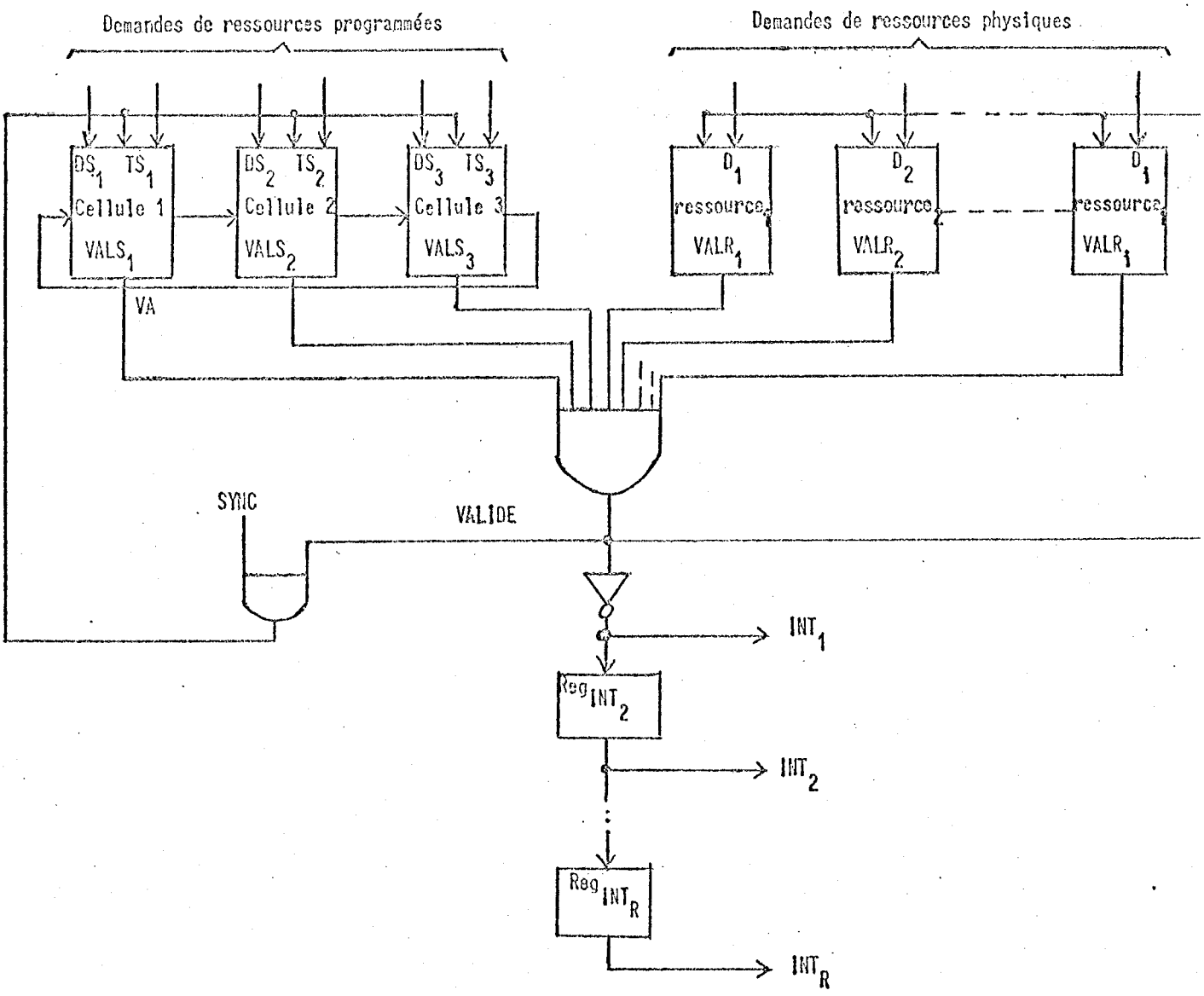


Figure IV.11. : AUTOMATE DE GESTION DES SEMAPHORES ET RESSOURCES PHYSIQUES

5. MECANISME D'AFFECTATION DU RESEAU

On a vu au paragraphe IV.2.c que les trois microprocesseurs du réseau ne peuvent traiter qu'un algorithme à la fois, chacun de ces algorithmes étant décomposé en trois tâches, que nous appellerons TEXT, T et TRAN, exécutées respectivement par EXT, OP et RAN.

L'état d'exécution d'un algorithme Ai est caractérisé par son v d'état VEi, constitué de trois couples : CEXTi, COPi et CRANi. Ces couple se composent respectivement : d'une part des mots d'état des tâches* TEXT TOPi et TRANi, et d'autre part des valeurs de paires de sémaphores A-A', B-B', C-C' employés pour leur synchronisation.

a) Alarme interne et appels externes

L'alarme interne est un signal qui est rendu actif chaque fois qu'on détecte une condition anormale au cours du traitement d'un algorith par exemple, à la suite d'erreurs dues à l'exécution d'une micro-instruct Les appels externes sont des signaux activés par des mécanismes extérieur réseau.

Lors de la prise en compte d'une alarme interne ou d'un appel externe, les trois microprocesseurs du réseau doivent différer l'exécutio l'algorithme Ai auquel ils étaient affectés, pour se consacrer à l'exécut de l'algorithme Aj spécifique de l'alarme interne ou appel externe en question. Pour obtenir ce fonctionnement il faut :

- sauvegarder le vecteur d'état de l'algorithme Ai

---

* Le mot d'état d'une tâche est constitué de son pointeur de microprogram de son code de condition.

- Générer le vecteur d'état propre à l'algorithme Aj*, et le forcer dans le réseau.

On verra un peu plus loin que les vecteurs d'état sont sauvegardés dans une file spéciale FVE.

L'alarme interne et les appels externes ne peuvent être pris en compte que dans la troisième phase du réseau (Ph-R3). Les couples CEXT, COP et CRAN sont respectivement sauvegardés et/ou forcés dans le réseau durant les trois phases Ph-R1, Ph-R2 et Ph-R3 qui suivent la prise en compte de l'un de ces signaux. Pendant ces trois phases, les microprocesseurs du réseau ne sont pas autorisés à commencer l'exécution d'une nouvelle micro-instruction**.

A la fin du traitement de l'algorithme Aj, les microprocesseurs doivent reprendre le traitement de Ai. Pour ce faire, il faut forcer dans le réseau le vecteur d'état de Ai. Le traitement d'Aj étant fini, on ne doit pas sauvegarder son vecteur d'état.

La fin du traitement d'un algorithme ne peut être signalée que par les tâches exécutées par le microprocesseur EXT disposant à cet effet de deux micro-instructions :

---

* Dans tous les cas, ce vecteur d'état est tel que les trois sémaphores A, B et C prendront une valeur 0 - et par là $A' = B' = C' = 1$ - (c.f. IV.4.) et les codes de condition des trois tâches prendront une valeur 0000 (c.f. IV.3.b). D'autre part, les pointeurs de microprogramme seront forcés à une valeur caractéristique de l'alarme interne et de chaque appel externe.

** Pour empêcher l'exécution des micro-instructions, on utilise une partie du circuit de synchronisation illustré par la figure IV.11. Tant que l'on ne désire pas commencer l'exécution d'une micro-instruction, le signal VALIDE est forcé à zéro.

- RETOUR, pouvant être employée pour indiquer la fin du traitement d'un algorithme lié à une alarme interne. Dans les trois phases Ph-R1, Ph-R2 Ph-R3 qui suivent la fin de l'exécution de cette micro-instruction, les couples CEXT, COP et RAN du dernier vecteur d'état sauvegardé dans la fi FVE, sont forcés dans le réseau.
- RETOUR-RU, employée normalement pour indiquer la fin du traitement d'un algorithme lié à un appel externe. Les actions provoquées par l'exécutio de cette micro-instruction sont les mêmes que celles de RETOUR, mais en plus, le bit 0 du registre de priorités PRIOR (exposé par la suite), est forcé à la valeur 1.

L'algorithme traité effectivement par le réseau est affecté d'un priorité de valeur 1, mais celle-ci peut être portée temporairement à la valeur 9 à la suite de l'exécution par le microprocesseur EXT de certaines micro-instructions. Il en est ainsi pendant les trois phases Ph-R1, Ph-R2 Ph-R3 employées pour exécuter : soit une micro-instruction qui affecte la valeur du registre de priorités PRIOR (voir plus loin), soit d'une micro-instruction RETOUR, RETOUR-RU, ENTRER ou SORTIR (voir plus loin). Dans le de ces quatre dernières micro-instructions, la priorité temporaire 9 est encore maintenue pendant les trois phases Ph-R1, Ph-R2 et Ph-R3 qui suiven leur exécution.

Nous avons défini un seul niveau d'alarme interne affecté d'une priorité invariable de valeur 9. Chaque condition anormale est associée à l'une des trois tâches qui composent un algorithme. Ainsi les conditions anormales associées respectivement à TEXT, TOP et TRAN sont détectées dan les phases du réseau PH-R1, PH-R2 et PH-R3, c'est-à-dire, dans la dernièr phase d'exécution des trois microprocesseurs liés à ces trois tâches.

La détection d'une condition anormale entraîne les actions suivantes :

- Augmentation d'une unité de la valeur du compteur NBCA. La valeur de ce compteur correspond au nombre total des conditions anormales détectées.
- Si avant cette augmentation, la valeur de NBCA était zéro, on aurait :

  - Activation de l'alarme interne

- Consignation dans le registre TCA des informations nécessaires à l'identification de la ou des causes de cette condition anormale (p. ex. le type d'erreurs provoquées par l'exécution d'une micro-instruction).

NBCA et TCA font partie du registre RERR. Ce registre, accessible au microprocesseur EXT, nous permet de connaître le nombre de conditions anormales détectées, ainsi que la nature de la première d'entre elles.

Aussi longtemps que l'alarme interne est active, les microprocesseurs ne sont autorisés à commencer l'exécution d'aucune micro-instruction. La prise en compte de cette alarme interne (dans la première phase Ph-R3 où la priorité de l'algorithme en cours d'exécution a une priorité différente de 9) entraîne, nous l'avons vu, l'exécution d'un algorithme spécifique, mais encore elle désactive l'alarme interne et entraîne un fonctionnement dégénéré du réseau.

Pendant ce fonctionnement :

- Les appels externes ne sont plus pris en compte
- L'activation de l'alarme interne provoque l'arrêt immédiat du réseau.

Pour mettre fin au fonctionnement dégénéré, le microprocesseur EXT dispose d'une micro-instruction spéciale : NORM, qui force la valeur zéro dans le compteur NBCA.

D'autre part, nous avons défini sept niveaux d'appels externes APEXT(I) (I = 2 à 8). A chacun de ces niveaux ci, on peut affecter deux rangs de priorité : 0 et I, déterminées par un registre spécial : PRIOR, accessible au microprocesseur EXT. Deux bits de ce registre : PRIOR(0 : 1) déterminent globalement la priorité des appels externes. Si PRIOR(0) = 0 ou PRIOR(1) = 0, toutes les priorités sont égales à zéro. Les bits PRIOR(2 : 8) déterminent individuellement la priorité de chacun des appels externes. Si PRIOR(0) = 1 et PRIOR(1) = 1 et PRIOR(I) = 1, la priorité de APEXT(I) est égale à 10 - I, sinon elle est égale à zéro.

Etant donné que tout algorithme exécuté par le réseau est généralement affecté d'une priorité 1, le registre PRIOR permet la définit de zones d'exclusion mutuelle (masque) entre celui-ci et les algorithmes impliqués par les appels externes.

La prise en compte des appels externes est effectuée dans les phases Ph-R3. L'appel externe actif de plus haute priorité est pris en compte si :

- La priorité de cet appel est supérieure à celle de l'algorithme exécuté le réseau.
- Le fonctionnement du réseau n'est pas dégénéré
- La file FVE contient moins de trois vecteurs d'état
- L'alarme interne n'est pas active
- L'alarme interne n'est pas activée pendant les trois phases Ph-R1, Ph-R2 et Ph-R3 suivantes.

Si les quatre premières conditions sont satisfaites, on dit que l'appel externe est efficace, et on le prend en compte provisoirement. A c fin et durant les phases Ph-R1, Ph-R2 et Ph-R3 suivantes, on sauvegarde le vecteur d'état de l'algorithme Ai en cours d'exécution, et on force dans l réseau celui associé à APEXT(J). Mais, dans la dernière de ces trois phase (Ph-R3), on vérifie si la cinquième condition a été satisfaite. S'il en es ainsi, la prise en compte de l'appel externe devient définitive : on force valeur 0 dans la bascule PRIOR(0), et on en avertit le milieu externe à l' d'un signal : REPON(J), particulier à chaque niveau APEXT(J) *. Sinon, et p pouvoir prendre en compte l'alarme interne dans le contexte où elle a été activée, on récupère le vecteur d'état de l'algorithme Ai et on le force d le réseau; c'est-à-dire, on exécute les mêmes actions que celles impliquée par une micro-instruction RETOUR.

---

* Il est indispensable de prévenir le milieu externe de la prise en compte chacun des appels externes, car, ce sont eux les seuls à pouvoir les désactiver.

La figure IV.12. représente l'organigramme décrivant le fonctionnement - à chacune des phases du réseau - de l'organe chargé de la détection des conditions anormales et de l'activation de l'alarme interne.

La figure IV.13. représente les organigrammes décrivant le fonctionnement du mécanisme chargé de l'affectation du réseau. Cet organe n'est autre chose qu'un ordonnateur. Le travail décrit par chacun des organigrammes est exécuté dans un cycle. Un cycle est décomposé en trois parties correspondant aux trois phases Ph-R1, Ph-R2 et Ph-R3 du réseau.

b) File des vecteurs d'état FVE

Le réseau dispose d'une file FVE, employée normalement pour la sauvegarde des vecteurs d'état. Elle peut en contenir jusqu'à quatre et fonctionne suivant le principe dernier entré - premier sorti. L'introduction ou extraction de chaque vecteur d'état s'effectue en trois temps, à partir de la phase Ph-R1, comme suit :

Ph-R1	→	CEXT
Ph-R2	→	COP
Ph-R3	→	CRAN

Un vecteur d'état ne peut être introduit dans cette file que lors de la prise en compte des appels externes ou de l'alarme interne, ou encore, après l'exécution de la micro-instruction ENTRER. Un vecteur d'état ne peut être extrait de cette file qu'après l'exécution de l'une des micro-instructions suivantes : RETOUR, RETOUR-RU et SORTIR.

Les micro-instructions ENTRER et SORTIR (propres au microprocesseur EXT) sont destinées à une gestion plus souple de la file FVE :

- ENTRER implique la sauvegarde dans la file FVE du vecteur d'état de l'algorithme en cours de traitement, mais l'exécution de cet algorithme continue normalement, sans être différée.

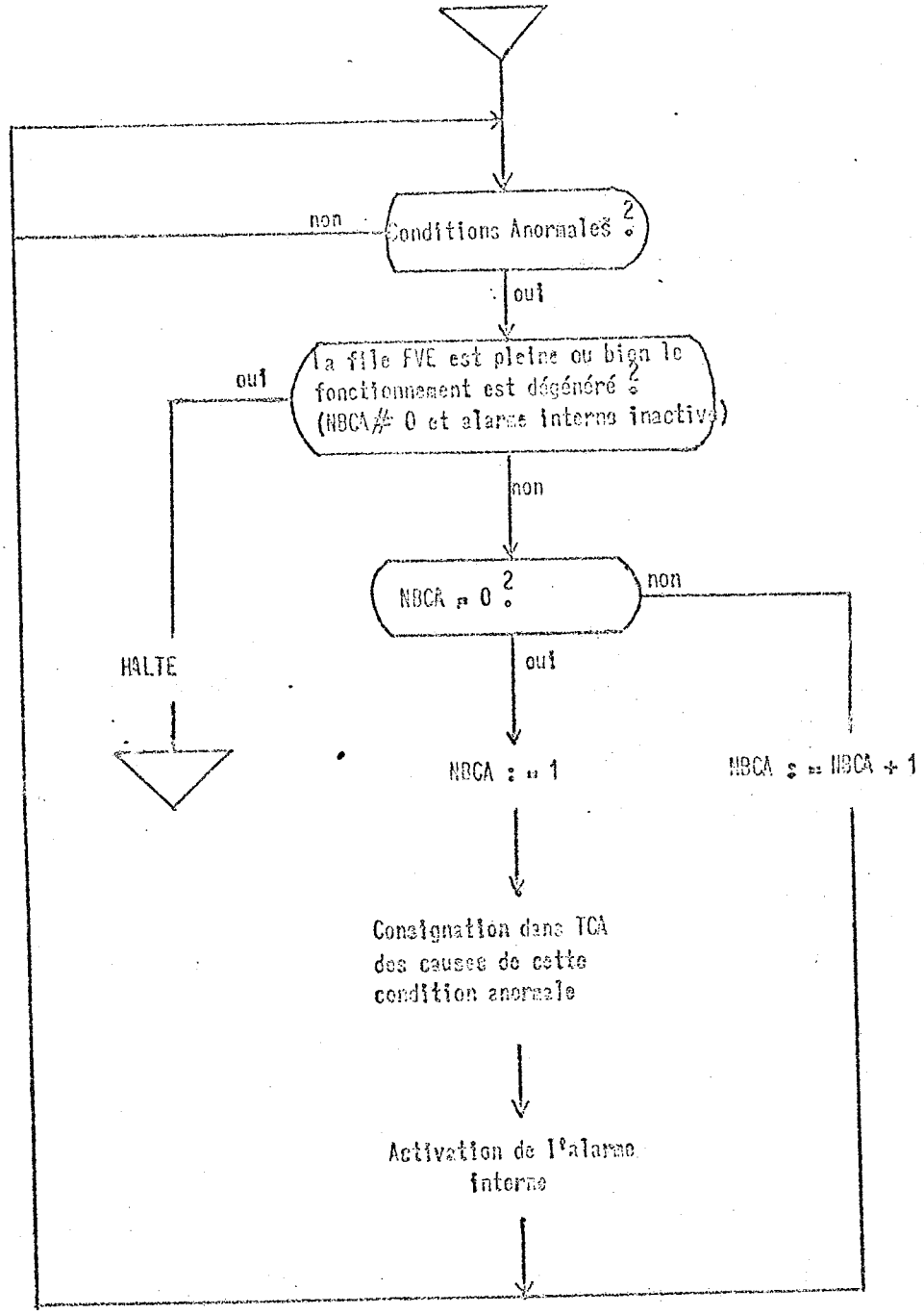
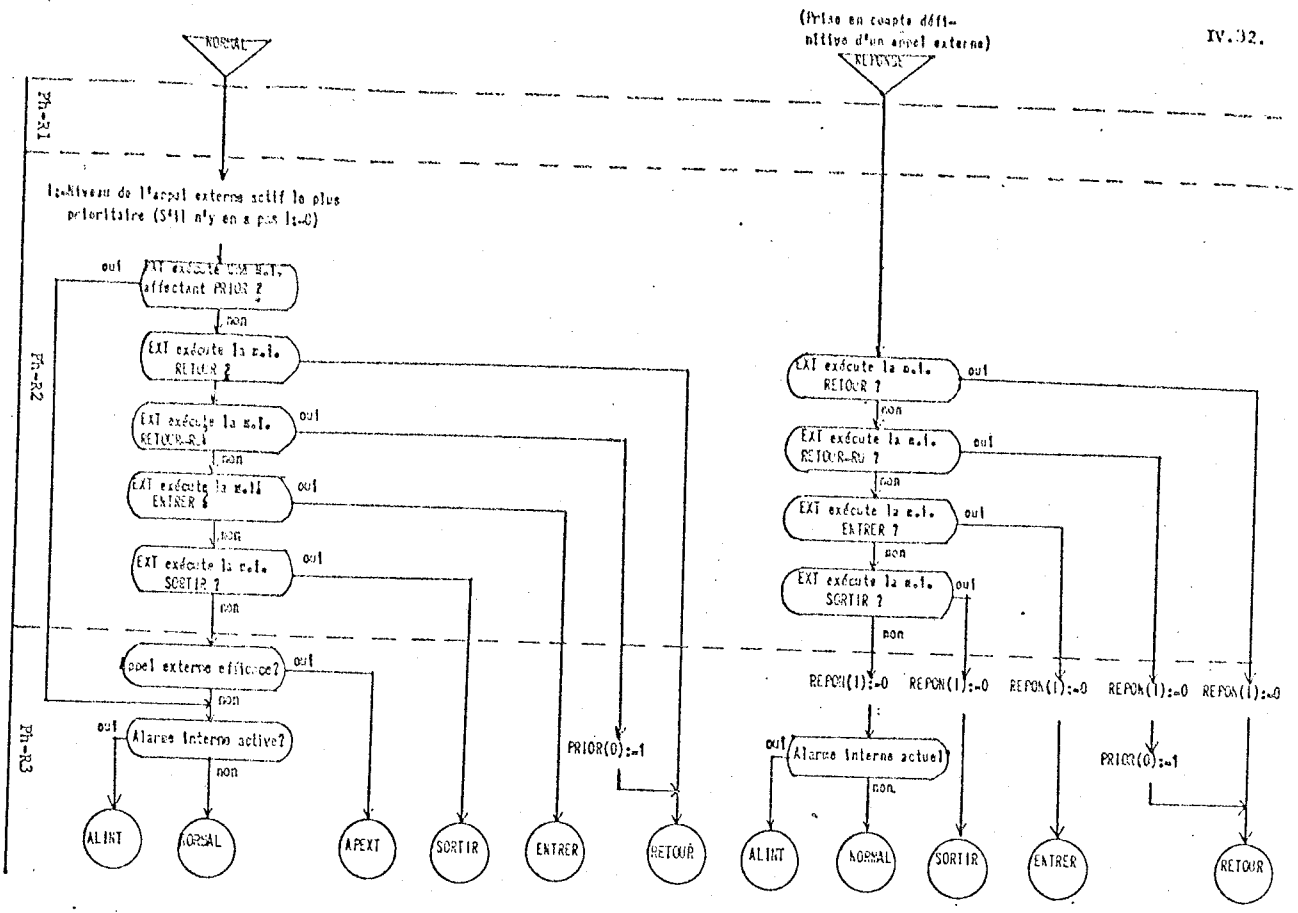


Figure IV.12. : ORGANIGRAMME DU FONCTIONNEMENT DE L'AUTOMATE DE DETECTION DES CONDITIONS ANORMALES.



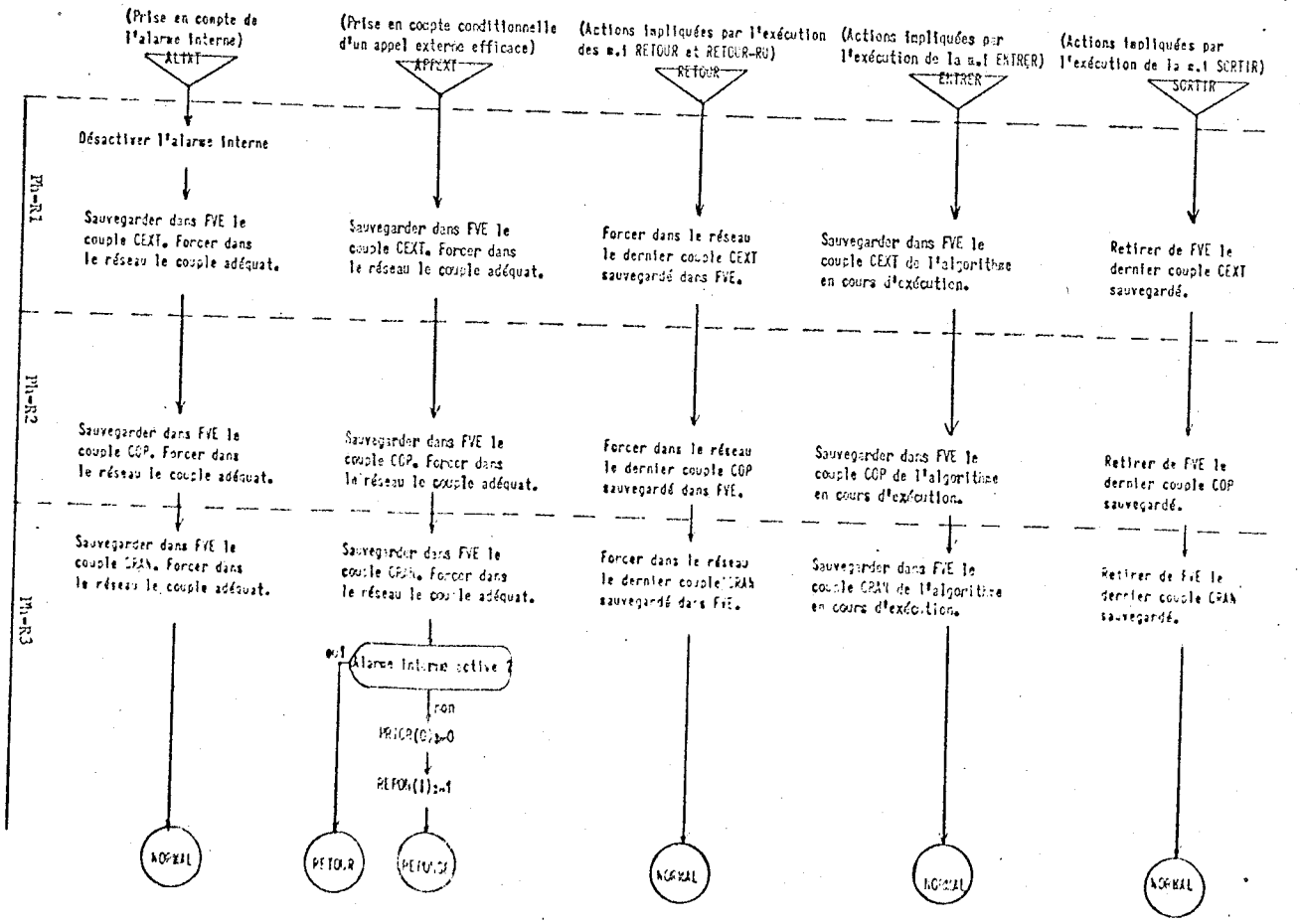


Figure IV.13 : ORGANIGRAMMES DE FONCTIONNEMENT DU MECANISME D'AFFECTATION DU RESEAU

- SORTIR implique l'extraction du dernier vecteur d'état sauvegardé dans FVE, mais sans le forcer dans le réseau.

Il faut remarquer que tant que la file FVE est pleine (4 vecteu d'état), on ne peut sauvegarder aucun autre vecteur d'état. L'activation l'alarme interne pendant cette période provoque l'arrêt immédiat du réseau (voir la figure IV.12.).

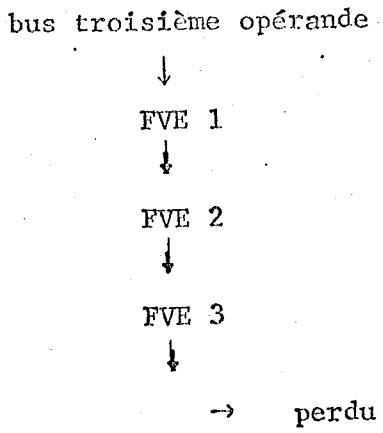
Les couples d'état à sauvegarder dans la file FVE sont transféré par un bus spécifique BCE, tandis que ceux composant les vecteurs d'état générés lors de la prise en compte des appels externes ou de l'alarme inte ainsi que ceux en provenance de la file FCV (après l'exécution des micro-instructions RETOUR ou RETOUR-RU) sont transférés vers les registres adéquats par le bus premier opérande. C'est ainsi que les couples portent 16 bits :

- bit 0 : Valeur d'une paire de sémaphores S - S'
- bits 1 : 2 : Non attribués
- bits 3 : 6 : Code de condition
- bits 7 : 15 : Pointeur microprogramme

Le bus BCE est branché en permanence à la sortie des registres RSEM(1) (c.f. IV.4.), RCC(0 : 3, 1) (c.f. IV.3.c) et RPOINT(0 : 8, 1) (c.f. IV.3.c), ainsi, étant donnée la nature circulante des informations contenues dans ces registres, ce bus est toujours en mesure de transférer FVE les couples d'état adéquats.

. Les couples d'état transférés par le bus premier opérande, sont forcés dans le réseau à l'aide de sélecteurs à deux entrées, placés à l'entrée des registres RSEM(2), RCC(0 : 3, 2) et RPOINT(0 : 8, 2). Normale ces sélecteurs aiguillent vers ces registres les informations SEMS(1), RCC(0 : 3, 1) et RPOINT(0 : 8, 2). En cas de changement de vecteur de contexte ils aiguillent vers ces mêmes registres les informations du bus premier opérande.

D'autre part, pour permettre une meilleure gestion des vecteurs d'état, les couples composant le dernier vecteur d'état sauvegardé dans FVE sont accessibles au microprocesseur EXT. Le dernier vecteur d'état sauvegardé est toujours contenu dans les 3 éléments supérieurs de FVE : FVE 1, FVE 2 et FVE 3. Les micro-instructions exécutées par EXT peuvent employer comme deuxième opérande, le contenu de l'élément FVE 3, et/ou adresser cet ensemble d'éléments comme troisième opérande. Dans ce cas, les informations du bus troisième opérande et celles contenues dans les éléments FVE 1 et FVE 2 sont forcées respectivement dans FVE 1, FVE 2 et FVE 3, le contenu de ce dernier étant perdu *.



---

* Cette procédure est le seul moyen de modifier un vecteur d'état, mais il faut signaler que quand on veut forcer dans le réseau le dernier vecteur d'état sauvegardé dans FVE, on considère que FVE 1, FVE 2 et FVE 3 contiennent respectivement les couples CEXT, COP et CRAN. Pour respecter cet ordre, toute modification d'un vecteur d'état, doit s'accompagner de trois écritures dans la file FVE.

6. LES OPERATEURS SPECIALISES

Dans ce paragraphe, on décrit le fonctionnement des opérateurs spécialisés de base : mémoire principale, unité d'écriture en mémoire de commande, canaux d'entrée-sortie, multiplieur rapide et mémoire locale.

a) Mémoire principale

Nous avons vu (c.f. IV.2) que le microprocesseur OP est le seul à avoir accès, par l'intermédiaire des files FLEC et FECR, à la mémoire principale. Les microprocesseurs EXT et RAN assurent respectivement les calculs d'adresses de lecture et d'écriture. Autour de la mémoire proprement dite, cet opérateur dispose des éléments suivants (figure IV.14) :

- Un registre : RADLEC, pour recevoir les adresses de lecture en provenance du microprocesseur EXT.
- Un registre : RADECR, pour recevoir les adresses d'écriture en provenance du microprocesseur RAN.
- Une file d'attente : FLEC, où sont rangées les données lues en mémoire principale, et destinées au microprocesseur OP.
- Une file d'attente : FECR, pour recevoir les données en provenance du microprocesseur OP et destinées à être rangées en mémoire principale.

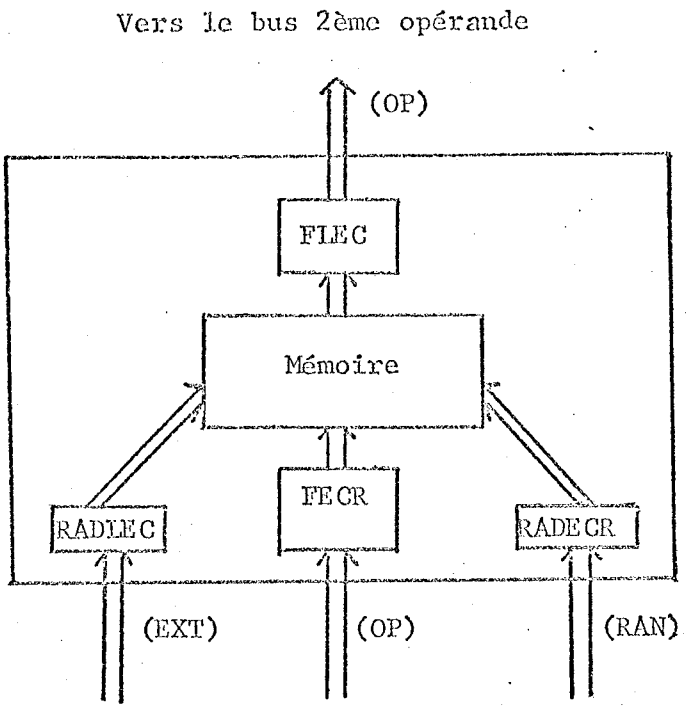


Figure IV.14 : ORGANISATION DE LA MEMOIRE PRINCIPALE

Chacune des files d'attente : FLEC et FECR, peut recevoir 4 données et fonctionne d'après le principe "premier entré, premier sorti".

Pour lire des données en mémoire principale, il faut le concours des microprocesseurs EXT et OP. L'exécution de chacune des microinstructions propres à EXT, adressant RADLEC comme opérande destination, force dans ce registre une adresse de la mémoire principale. La donnée contenue à cette adresse est lue et rangée dans la file FLEC. Les données contenues dans FLEC sont accessibles par les microinstructions propres au microprocesseur OP adressant cette même file comme deuxième opérande.

Pour écrire des données en mémoire principale, il faut le concours des microprocesseurs OP et RAN. OP, adressant FECR comme opérande destination, y introduit les données à ranger dans la mémoire principale. RAN, adressant le registre RADECR comme opérande destination, y force les adresses d'écriture.

Cette forme de fonctionnement peut entraîner l'existence de plusieurs données en attente dans les files FLEC et FECR. Lors des affect du réseau à l'exécution de différents algorithmes (affectations dues à la prise en compte des appels externes ou de l'alarme interne) il est utile libérer rapidement de leur contenu ces deux files. A cet effet, l'opérate mémoire principale dispose d'une file de sauvegarde : FSAUV. L'organisat de cette file est telle qu'on peut y sauvegarder jusqu'à concurrence de quatre ensembles de données en provenance de FLEC et FECR.

La gestion de FSAUV, qui fonctionne suivant le principe "dernie ensemble de données entré, premier ensemble de données sorti", est assuré par deux microinstructions spéciales propres au microprocesseur EXT :

- SAUV - FILES, dont l'exécution provoque le transfert dans FSAUV du cont des files FLEC et FECR.
- REST - FILES, dont l'exécution provoque le transfert dans FLEC et FECR dernier ensemble de données sauvegardées dans FSAUV.

La gestion de l'ensemble des éléments de l'opérateur mémoire principale repose sur les mécanismes décrits au paragraphe IV.4. Ainsi, quand le microprocesseur EXT entraîne l'exécution d'une microinstruction impliquant l'écriture dans le registre RADLEC, la validation sera obtenue si ce registre est libre, et si la file FLEC dispose de places libres pou recevoir une nouvelle donnée lue en mémoire. De même, une microinstructic propre au microprocesseur RAN impliquant l'écriture dans le registre RADE ne pourra être exécutée que si celui-ci est libre, et si dans la file FEC il y a des données à ranger en mémoire.

D'autre part, aucune microinstruction faisant appel à la mémoir principale, ne pourra être exécutée pendant les périodes de sauvegarde ou restauration des files FLEC et FECR.

Enfin, la mémoire principale dispose d'un mécanisme d'accès dir adapté aux périphériques rapides éventuels du réseau.

La figure IV.5 représente un bloc-diagramme simplifié de cet opérateur. Lors des conflits d'accès à la mémoire, un système de synchronisation et contrôle, sélectionne l'opération exécutée en premier.

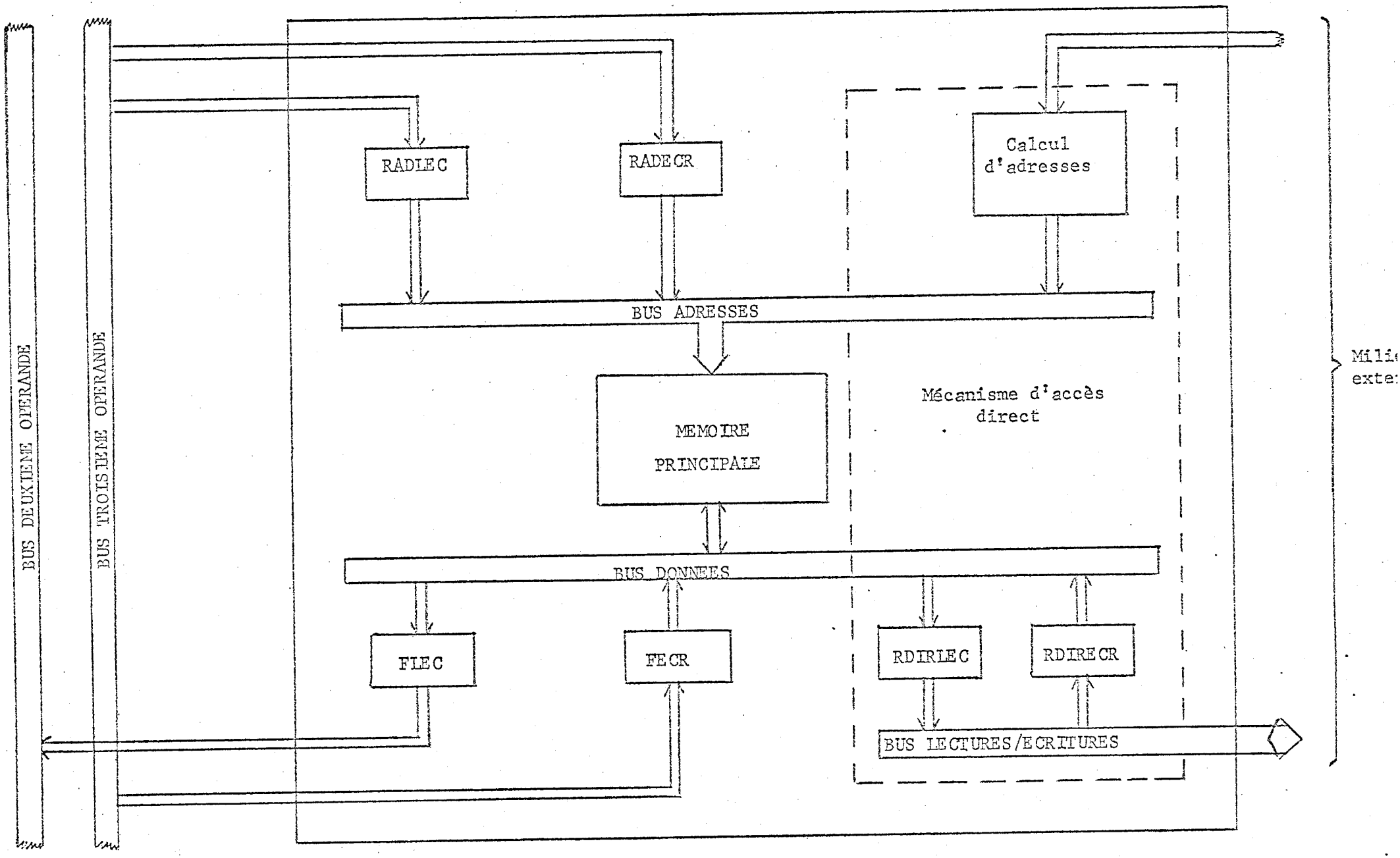


Figure IV.15 : BLOC-DIAGRAMME DE L'OPERATEUR MEMOIRE PRINCIPALE

b) Unité d'écriture en mémoire de commande

L'unité d'écriture en mémoire de commande est superposée à unité d'adressage UAC (c.f. IV.3.a et IV.3.c). La figure IV.16, repré les éléments qui permettent l'écriture des microinstructions dans cet mémoire :

- Un registre de 30 bits : RMICRO (Ø : 29) contenant la microinstruct écrire.
- Un registre de 8 bits : RADCOM (1 : 8) contenant les 8 bits poids f de l'adresse de la mémoire de commande où l'on veut écrire la microinstruction*.
- Un sélecteur pour aiguiller vers la mémoire de commande, soit l'adr contenue dans le registre RPOINT (Ø : 8,2) (fonctionnement normal), l'adresse contenue dans le registre RADCOM.

---

* Le neuvième bit de cette adresse est toujours considéré comme égal à En effet, seuls les 256 mots de la mémoire de commande numérotés de 512, sont inscriptibles, le reste étant constitué d'une mémoire per

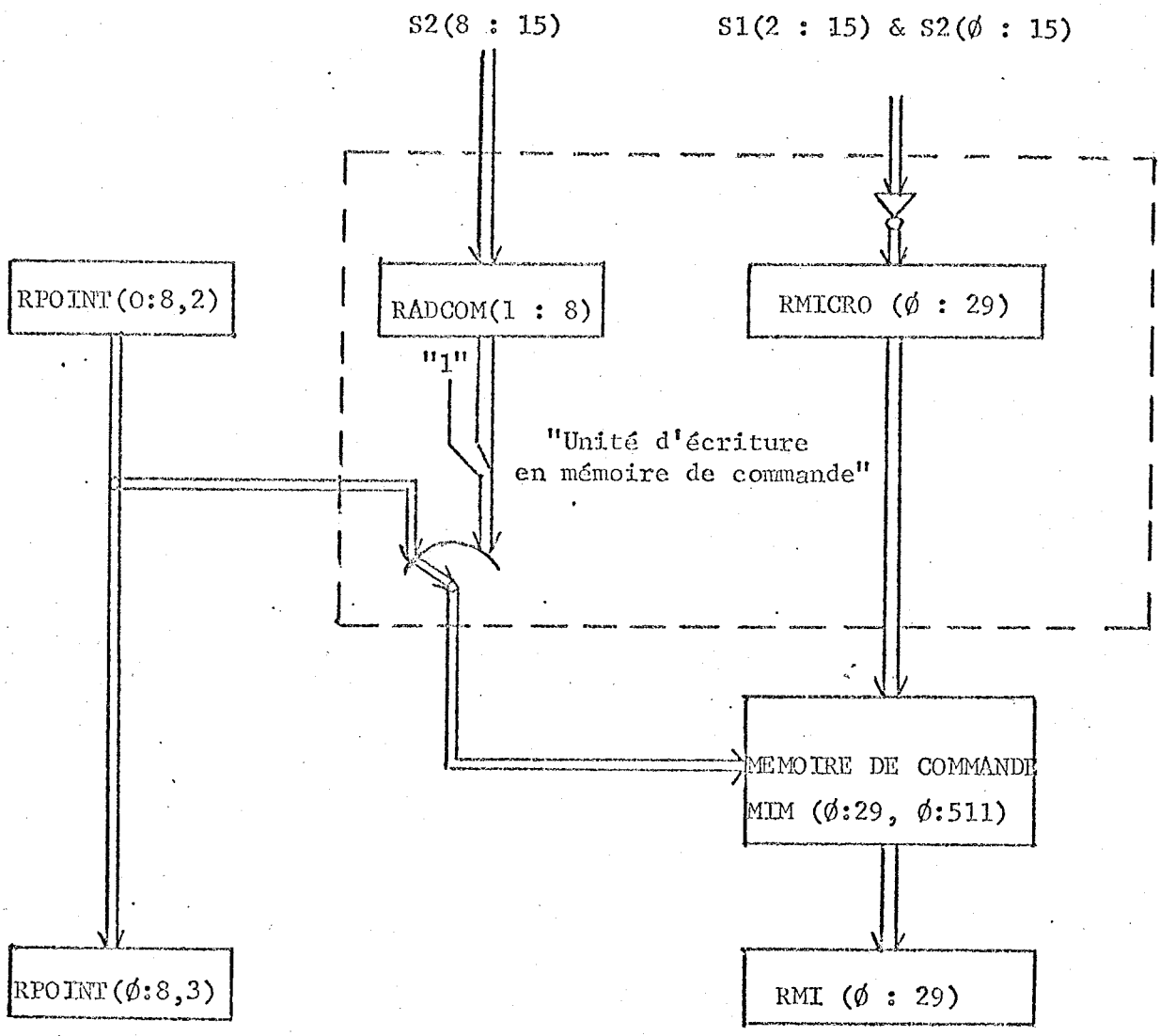


Figure IV.16 : BLOC DIAGRAMME DE L'UNITE D'ECRITURE EN MEMOIRE DE COMMANDE

L'écriture d'une microinstruction est exécutée en deux temps :

- Chargement du registre RMICRO avec le complément des 14 bits poids faibles transférés par le bus premier opérande et des 16 bits transférés par le bus deuxième opérande. $\overline{(S1(2:15)\ \&\ S2(0:15))}$. Cette opération est commandée par une microinstruction spéciale : CH-MICRO, propre au microprocesseur RAN.
- Chargement du registre RADCOM avec les 8 bits poids faibles transférés par le bus deuxième opérande. Transfert à l'adresse 1 & RADCOM (1:8) de la mémoire de commande du contenu du registre RMICRO. Ces opérations sont commandées par une microinstruction spéciale : CH-COMM, propre au microprocesseur RAN.

REMARQUE : La lecture des microinstructions est impossible pendant la période où l'on écrit dans la mémoire de commande, période comprenant les phases du réseau Ph-R2 et Ph-R3 (c.f. IV.2) qui suivent le début de l'exécution de la microinstruction CH-COMM. C'est pourquoi, pendant ces deux phases, les microprocesseurs RAN et EXT respectivement, ne pourront pas accéder à cette mémoire, et n'auront pas de microinstructions à exécuter dans les trois phase suivantes.

c) Opérateur d'entrées/sorties

Le réseau est prévu pour recevoir jusqu'à 4 sous-canaux. Chaque sous-canal doit contenir les circuits logiques nécessaires à l'exécution de commandes d'E/S, ainsi qu'à la détection, et éventuellement localisation, d erreurs apparues en cours d'exécution des opérations d'E/S.

Associés à chaque sous-canal, le réseau dispose de 4 registres principaux :

- Registre d'état (RETAT (0 : 15)), accessible en lecture à partir du microprocesseur RAN.
- Registre de commande (RCOM (0 : 15)), accessible en écriture à partir du microprocesseur RAN.
- Registre de transfert en entrée (RTE (0 : 15)), accessible en lecture à partir du microprocesseur OP.
- Registre de transfert en sortie (RTS (0 : 15)), accessible en écriture à partir du microprocesseur OP.

C'est à l'aide des registres RCOM et RETAT respectivement, que le microprocesseur RAN commande et contrôle les opérations d'E/S propres à cha sous-canal, mais c'est seulement le microprocesseur OP qui, à travers les registres RTE et RTS, peut échanger des données avec le monde extérieur.

Les fonctions "commande-contrôle" et "échanges d'information" de ce système, sont donc séparées et exécutées par deux microprocesseurs différents (RAN et OP respectivement). Théoriquement, les E/S peuvent se faire au rythme d'un échange par microinstruction exécutée par OP. Pratiquement, ce rythme est déterminé par la durée du cycle de la mémoire principale.

Le nombre de sous-canaux, ainsi que leur configuration, dépend de l'environnement physique du réseau. On peut envisager deux cas extrêmes :

- Réseau autonome : le réseau doit disposer de tous les outils nécessaires au contrôle et exécution des E/S.
- Réseau périphérique : le réseau fait partie d'un système plus vaste, et n'a pas à exécuter les opérations d'E/S d'une façon directe, celles-ci étant exécutées par ailleurs. Tous les échanges d'information entre le réseau et le reste du système peuvent se faire à l'aide de l'accès direct de la mémoire principale du réseau. L'initialisation et contrôle de ces échanges est assuré par un seul sous-canal.

Prenons par exemple, un réseau autonome comportant un grand nombre d'éléments d'E/S. On dispose de quatre sous-canaux pouvant être attribués de la façon suivante :

- Sous-canal 1 : E/S rapides par blocs de données (par exemple : disques, bandes magnétiques, etc).
- Sous-canal 2 : E/S rapides mot à mot (par exemple : processus physiques en temps réel).
- Sous-canal 3 : E/S lentes par blocs de données (par exemple : lecteur/perforateur de cartes) et mot à mot (par exemple : lecteur/perforateur de rubans, télétype, etc).
- Sous-canal 4 : Destiné à doubler les sous-canaux 1 et 2

d) Multiplieur rapide

Cet opérateur, affecté exclusivement au microprocesseur OP, perme d'effectuer le produit de deux nombres de 16 bits en représentation complément à deux, le résultat de cette opération étant un nombre de 32 bit

On peut considérer que le multiplieur dispose de quatre registres principaux : A(0:15), B(0:15), PFORT(0:15) et PFAIB(0:15). Les registres PFORT et PFAIB sont accessibles exclusivement en lecture, et peuvent être utilisés par toute microinstruction propre à OP, comportant un deuxième opérande. Les registres A et B, accessibles exclusivement en écriture ne peuvent recevoir que les données transférées par les bus premier et deuxièm opérande respectivement (voir figure IV.8). Le chargement de ces deux registres est commandé par deux microinstructions spéciales propres à OP : CH-MULT et MULT. Dans le cas de la première de ces microinstructions, les données ainsi chargées en A et B sont transférées, sans modification, vers les registres PFORT et PFAIB respectivement, tandis que dans le cas de la deuxième, c'est le produit de ces deux données qui est transféré vers PFORT (16 bits poids forts) et PFAIB (16 bits poids faibles).

Le temps employé par le multiplieur pour exécuter chacune de ces opérations est tel qu'il sera accessible dès la microinstruction qui suit immédiatement l'exécution de CH-MULT, mais il faudra attendre deux microinstructions s'il s'agit de l'exécution de MULT. Ainsi, l'exécution d microinstructions de type MULT ou CH-MULT, et de toute autre faisant intervenir comme deuxième opérande le contenu des registres PFORT ou PFAIB sont différées "d'un temps" si elles suivent la microinstruction MULT.

e) Opérateur mémoire locale

Le microprocesseur OP dispose d'une mémoire propre : MEMLOC, d'u capacité de 256 mots de 16 bits. Cette mémoire est adressée à l'aide des 8 bits poids faibles d'un registre spécial : RCOMPT (0 : 15), accessible à partir de OP aussi bien en lecture qu'en écriture.

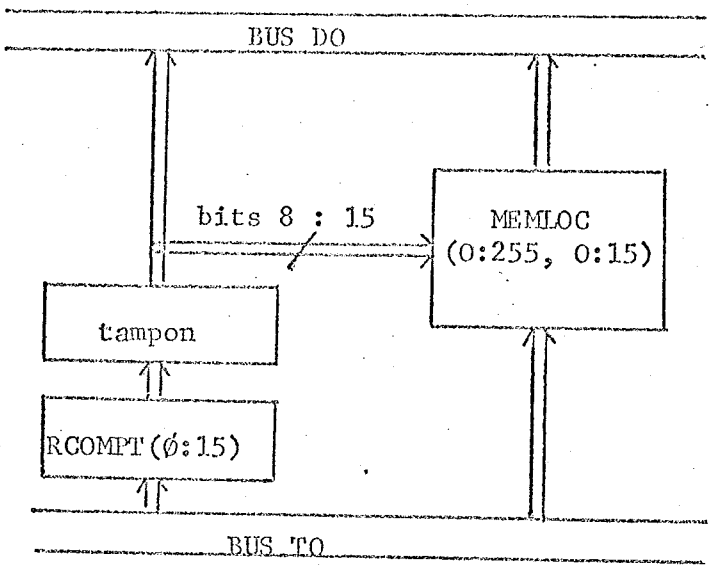


Figure IV.17 : OPERATEUR MEMOIRE LOCALE

Une microinstruction propre à OP faisant appel à MEMLOC comme opérande source (bus DO) ou opérande destination utilisera l'emplacement désigné par les 8 bits RCOMPT (8 : 15).

Pour rendre plus souple l'utilisation de cette mémoire, les microinstructions propres à OP peuvent commander une opération spéciale : Incrément d'une unité et test du registre RCOMPT (test positif si RCOMPT # 0 après l'incrément).

Pour assurer un fonctionnement correct, au début de chaque microinstruction propre à OP, et pendant toute la durée de celle-ci, le contenu du registre RCOMPT est mémorisé de façon que, pendant tout ce temps, on adresse un seul emplacement de la mémoire MEMLOC, ceci en dépit des éventuelles opérations incrément et test effectuées sur le registre RCOMPT.

Par ailleurs, lors de l'exécution d'une microinstruction où le registre RCOMPT est employé comme troisième opérande et l'opération incrément et test est commandée, cette dernière est effectuée normalement, mais en fin d'exécution, le registre RCOMPT aura la valeur imposée par le bus troisième opérande.

7. MICROINSTRUCTIONS. ALARMES INTERNES

a) Champs d'une microinstruction

Une microinstruction est constituée de 30 bits et découpée, selo son type, en 5 ou 7 champs (groupements de bits). Le tableau suivant résum le rôle de chacun de ces champs.

Bits	Conditions	Fonction
0 : 3	-	Code opération (16 codes différents par microprocesseur)
4 : 8	m.i. sans 3ème opérande	Complément du code d'opération, ou bien sans signification
	m.i. avec 3ème opérande	Adresse du 3ème opérande (32 adresses possibles par microprocesseur)
9 :13	m.i. sans 2ème opérande	Sans signification
	m.i. avec 3ème opérande	Adresse du 2ème opérande (32 adresses possibles par microprocesseur)
14	-	Indique si le 1er opérande de la microinstruction est une valeur constante.
15:29	premier opérande constant (bit 14 = 1)	Valeur des bits 1 à 15 du premier opérande. Le bit poids le plus fort est égal au bit 1 (c.f. IV.3.b)
15:18	bit 14 = 0	Détermine s'il s'agit d'une microinstruction soit spécifiant le code d'options de l'UAL (cf. IV.3.b), soit demandant une opération de synchronisation (c.f. IV.4), soit encore avec branchement relatif - si test positif - (cf. IV.3.c).
19:25	m.i. spécifiant le code d'options de l'UAL (bits 14:18 = 00000)	Les bits 19 : 23 spécifient le code d'options de l'UAL (voir plus loin). Les bits 24 : 25 n'ont pas de signification.
	m.i. demandant une opération de synchronisation (bits 14 : 18 = 00001)	Déterminent la primitive SYNC ou ISYNC à exécuter (voir plus loin).
	m.i. avec branchement relatif si test positif (bit 14=0, bits 15 : 17 # 000)	Valeur du branchement relatif exprimé en complément à deux : - 64 à + 63 (voir plus loin).
26:29	bit 14 = 0	Adresse du 1er opérande (16 adresses possibles par microprocesseur)

Figure IV.18 : LES CHAMPS D'UNE MICROINSTRUCTION

b) Codes opération

Le champ du code d'opération constitué de 4 bits, permet 16 combinaisons différentes. Chacun des codes n'a pas forcément la même signification pour les trois microprocesseurs du réseau. D'autre part, les microinstructions qui ne font pas intervenir le troisième opérande, disposent du champ normalement réservé à l'adresse de celui-ci. Dans ces cas là, ce champ représente un complément au code d'opération*. Les tableaux suivants décrivent sommairement les codes d'opération utilisés pour chaque microprocesseur**.

---

* Pour des raisons de simplicité, nous avons imposé la valeur 0 aux 2 bits poids forts de l'adresse deuxième ou troisième opérande lorsque la microinstruction ne fait pas intervenir l'un ou l'autre de ces opérandes.

** Un certain nombre de codes d'opération ne sont pas employés, et sont interprétés comme "non-opération". Ils sont disponibles pour commander les éventuels opérateurs spécialisés pouvant s'adjoindre au réseau.

Code opération	Complément du code OP	m. processeurs EXT	OP	RAN	Opérandes PO	DO	TO	Fonction	Mnémonique	Notes
0000	-	/	/	/	/	/	/	PO "plus" DO "plus" retenue	ADD-RET	La retenue est le bit 3 du code de condition (c.f. IV.3.b)
0001	-	/	/	/	/	/	/	PO "plus" DO	ADD	
0010	-	/	/	/	/	/	/	PO "ou" DO	OU	
0011	-	/	/	/	/	/	/	DO	DOP	A la sortie de l'opérateur arithmétique et logique (OPAL) on retrouve la valeur de DO indépendamment de celle de PO.
0100	-	/	/	/	/	/	/	PO "moins" DO "moins" retenue	ST-RET	La retenue est le bit 3 du code de condition (c.f. IV.3.b)
0101	-	/	/	/	/	/	/	PO "moins" DO	ST	
0110	-	/	/	/	/	/	/	PO "conjonction" DO	CONJ	PO . DO + $\overline{PO}$. $\overline{DO}$
0111	-	/	/	/	/	/	/	PO "et" DO	ET	
1000	sans signification	/	/	/	/	/	/	Branchement absolu	BRANCH	L'adresse de branchement est donnée par les 9 bits poids faibles du premier opérande.
1001	-	/	/	/	?	/	/	Exécution du code d'opération déterminé par les registres RCODE et EXTENSION	EXEC	L'opération arithmétique ou logique exécutée par ce type des m.i. ne peut être que l'une de celles associées aux codes 0000 à 0111. Cette m.i. est exposée plus loin dans ce paragraphe.
1010 à 1100	?	/	/	/	?	?	?			NON ATTRIBUES
1101	000XX	/						Sauvegarde des files de lecture et écriture	SAUV-FILES	c.f. IV.6.a *
1101	001XX	/						Restauration des files de lecture et écriture	REST-FILES	c.f. IV.6.a *
1101	?		/	/	?	?	?			NON ATTRIBUES
1110	sans signification	/						Passage au fonctionnement normal	NORM	c.f. IV.5.a
1110	?		/	/	?	?	?			NON ATTRIBUES
1111	0000X	/						Restauration du dernier vecteur d'état rangé dans la file FVE	RETOUR	c.f. IV.5.a **
1111	0001X	/						Même fonction plus remise à 1 de la bascule PRIOR (0)	RETOUR-RU	c.f. IV.5.a **
1111	0011X	/						Sauvegarde du vecteur d'état dans la file FVE	ENTRER	c.f. IV.5.b **
1111	0010X	/						Elimination du dernier vecteur d'état sauvegardé dans FVE	SORTIR	c.f. IV.5.b **
1111	0012X		/			/	/	Transfert vers la sortie du multiplieur du produit POxDO	MULT	c.f. IV.6.d *
1111	0003X		/			/	/	Transfert vers la sortie du multiplicur de PO et DO	CH-MULT	c.f. IV.6.d *
1111	0004X			/	/	/	/	Chargement du registre RMICRO avec $\overline{PO\ (2:15)}$ & DO	CH-MICRO	c.f. IV.6.b
1111	0014X				/		/	Transfert à l'adresse 1 & DO (8:15) de la mémoire de commande du contenu du registre RMICRO	CH-COMM	c.f. IV.6.b

FIGURE IV.19 : CODES OPERATION

* Codes faisant intervenir une ressource physique pouvant ne pas être disponible. [illegible]

c) Premier, deuxième et troisième opérande

Les adresses 0 à 15 des trois opérandes possibles d'une microinstruction, qu'elle soit propre au microprocesseur EXT, OP ou RAN, désignent les 16 registres généraux du réseau. Le premier opérande n'a accès qu'à ces 16 registres, par contre le deuxième et troisième disposent chacun de 16 adresses supplémentaires pour désigner les registres spécialisés du réseau. Les tableaux suivants (figures IV.20 et IV.21) donnent l'affectation de ces adresses. On remarquera qu'une même adresse peut désigner des êtres différents, selon qu'elle est employée par l'un ou l'autre des microprocesseurs, ou encore, pour un même microprocesseur, selon qu'elle est employée comme deuxième ou troisième opérande (accessibles respectivement en lecture ou en écriture). On remarquera encore que beaucoup d'adresses ne sont pas attribuées. Elles restent disponibles pour établir des échanges entre le réseau et d'éventuels opérateurs spécialisés.

Adresse DO	EXT	OP	RAN
0ϕϕϕϕ	Un des 16 registres généraux : RO à R15	RO à R15	RO à R15
10000	Couple d'état m.P.EXT : CE ext (Sem A-A' & Sem B-B' & Sem C-C' & RCC ext & RPOINT ext)	Couple d'état m.p. OP : CE op (Sem B-B' & Sem C-C' & Sem A-A' & RCC op & RPOINT op)	Couple d'état m.p. RAN : CE ran (Sem C-C' & Sem A-A' & Sem B-B' & RCC ran & RPOINT ran)
10001	Registre erreur : RERR (0 : 15)	File de lectures de la mémoire principale : FLEC	NON ATTRIBUEE
10010	File des vecteurs d'état : RCONTX (0 : 15,3)	Registre compteur : RCOMPT(0:15)	NON ATTRIBUEE
10011	Registre de priorités : PRIOR (0:15)	Mémoire locale : ML. Emplacement pointé par les bits RPOINT (8 : 15)	NON ATTRIBUEE
10100	Registre Code d'opération : RCODE(0:15)	RCODE (0 : 15)	RCODE (0 : 15)
10101	Registre extension : EXTENSION (0:15)	EXTENSION (0 : 15)	EXTENSION (0 : 15)
10110	NON ATTRIBUEES	Registre de transfert en entrée. Sous-canal 1*. RTE1 (0 : 15)	Registre d'état. Sous-canal 1*. RETAT1 (0 : 15)
10111	NON ATTRIBUEES	RTE2 (0 : 15)*	RETAT2 (0 : 15)*
11000	NON ATTRIBUEES	RTE3 (0 : 15)*	RETAT3 (0 : 15)*
11001	NON ATTRIBUEES	RTE4 (0 : 15)*	RETAT4 (0 : 15)*
11010	NON ATTRIBUEE	Poids faibles multiplieur : PFAIB(0:15)	NON ATTRIBUEE
11011	NON ATTRIBUEE	Poids forts multiplieur : PFORT(0:15)	NON ATTRIBUEE
11100 à 11111	NON ATTRIBUEES	NON ATTRIBUEES	NON ATTRIBUEES

* Ressources physiques pouvant ne pas être disponibles. Le cas échéant, l'exécution d'une microinstruction les utilisant peut être différée.

Adresses TO	EXT	OP	RAN
0	Un des 16 registres généraux : RO à R16	RO à R16	RO à R16
10000	NON ATTRIBUEE	NON ATTRIBUÉE	NON ATTRIBUEE
10001	Registre adresse lecture en mémoire principale : RADLEC*.	File d'écritures en mémoire principale : FECR*.	Registre adresse écriture en mémoire principale : RADECR*.
10010	File des vecteurs d'état : RCONTX (c.f. IV.5.b)	Registre compteur : RCOMPT (0:15)	NON ATTRIBUEE
10011	Registre de priorités : PRIOR(0:15)	Mémoire locale. Emplacement pointé par RCOMPT (8 : 15)	NON ATTRIBUEE
10100	Registre code d'opération : RCODE (0 : 15)	RCODE (0 : 15)	RCODE (0 : 15)
10101	Registre extension : EXTENSION (0:15)	EXTENSION (0 : 15)	EXTENSION (0 : 15)
10110	NON ATTRIBUEE	Registre de transfert en sortie. Sous-canal 1*. RTS1 (0 : 15)	Registre de commandes. Sous-canal 1*. RCOM1 (0 : 15)
10111	NON ATTRIBUEE	RTS2 (0 : 15)*	RCOM2 (0 : 15)*
11000	NON ATTRIBUEE	RTS3 (0 : 15)*	RCOM3 (0 : 15)*
11001	NON ATTRIBUEE	RTS4 (0 : 15)*	RCOM4 (0 : 15)*
11010 à 11111	NON ATTRIBUEES	NON ATTRIBUEES	NON ATTRIBUEES

* Ressources physiques pouvant ne pas être disponibles. Le cas échéant, l'exécution d'une microinstruction les utilisant peut être différée.

Figure IV.21 : ADRESSES TROISIEME OPERANDE

d) Microinstructions spécifiant le code d'options de l'UAL

A l'exception de la microinstruction EXEC, toutes les autres f
intervenir l'UAL doivent lui fournir le code d'options, sans quoi l'UAL
d'un code par défaut (c.f. IV.3.b).

Dans ce type de microinstructions, les bits 19 : 23 déterminen
code d'options de l'UAL selon le tableau exposé plus bas (figure IV.22),
bits 14 : 18 ayant la valeur 00000.

bits	Valeur	Fonction			
19	0	Alarme interne si débordement dans l'UAL *			
	1	Autrement (Mnémonique : MSQDEB)			
20	1	Si le code de condition est mis à jour (Mnémonique : POSRCC)			
	0	Autrement *			
21:23		Fonction	Mnémonique	BUS TO	Registre EXTENS
	000	Sortie Normale *		RSOPAL (1 : 16)	Sans modificatio
	001	Inversion binaire	IBS	RSOPAL (16 : 1)	Sans modificatio
	011	Déc. Droit simple avec entrée série	DSS	EXTENSION(15) & RSOPAL (1 : 15)	Sans modificatio
	010	Déc. Gauche arithmétique simple	GAS	RSOPAL(1) & RSOPAL (3:16) & 0	Sans modificatio
	110	Déc. Gauche circulaire étendu	GCE	RSOPAL(2:16) & EXTENSION (0)	EXTENSION(1:15) & ROSPAL (1)
	111	Déc. Droit arithmétique étendu	DAE	RSOPAL (0:15)	RSOPAL(16) & EXTENSION(0:14)
	101	Déc. Droit logique étendu	DLE	0 & RSOPAL(1:15)	RSOPAL(16) & EXTENSION (0:14)
	100	Déc. Gauche logique étendu	GLE	RSOPAL(2:16) & EXTENSION (0)	EXTENSION(1:15) &

* Options par défaut

Figure IV.22 : LES CODES D'OPTIONS DE L'UAL

Remarque : A la fin de l'exécution d'une microinstruction où le registre EXTENSION est employé comme troisième opérande (destination), la valeur rangée dans celui-ci est celle imposée par le bus correspondant, même si le code d'options de l'UAL utilise ce registre. L'opération chargement est prioritaire vis-à-vis de l'opération décalage.

e) Microinstructions avec opération de synchronisation

Le réseau dispose de trois couples de sémaphores : A-A', B-B', C-C', et de deux primitives parallèles de synchronisation : SYNC et ISYNC (c.f. IV.4). On a besoin de deux bits pour les signaux de commande T et D propres à chaque couple de sémaphores, et un bit pour déterminer s'il s'agit d'une primitive de type SYNC ou ISYNC.

Les primitives de synchronisation sont commandées par les microinstructions dont les bits 14 : 18 ont la valeur 00001, les bits 19 : 25 déterminant la primitive à exécuter. Pour des raisons technologiques, nous avons adopté une codification particulière à chaque microprocesseur :

bits / m. proc	19	20	21	22	23	24	25
EXT	DA	TA	DB	TB	DC	TC	Dans tous les cas bit 25=0:primitive ISYNC bit 25=1:primitive SYNC
OP	DB	TB	DC	TC	DA	TA	
RAN	DC	TC	DA	TA	DB	TB	

Par exemple, considérons une microinstruction de ce type : les bits 19 à 25 étant 10 00 11 1, ils seront traduits comme SYNC (a', c), SYNC (b', a) ou SYNC (c', b), selon qu'il s'agisse des microprocesseurs EXT, OP ou RAN.

f) Microinstructions avec branchement relatif

Le branchement relatif est l'un des moyens dont disposent les microinstructions pour agir sur le déroulement des microprogrammes. Ces branchements sont commandés par les microinstructions dont le bit 14 est éga à zéro, et les bits 15 : 18 ont une valeur comprise entre 0010 et 1111, ce q donne 14 possibilités de branchement pour chaque microprocesseur :

bits 15:18	microprocesseur EXT	OP	RAN	Conditions du branchement relatif	Mnémoniqu
0010	/	/	/	Inconditionnel	INC
0011	/	/	/	Si le code de condition (CC) indique égalité à zéro. (= 0)	EZ
0100	/	/	/	Si CC indique inégalité à zéro. ($\neq$ 0)	DZ
0101	/	/	/	Si CC indique supériorité à zéro. ($>$ 0)	GZ
0110	/	/	/	Si CC indique supériorité ou égalité à zéro. ($\geq$ 0)	GEZ
0111	/	/	/	Si CC indique infériorité ou égalité à zéro. ($\leq$ 0)	PEZ
1000	/	/	/	Si CC indique infériorité à zéro. ($<$ 0)	PZ
1001	/	/	/	Si CC indique un débordement	DEB
1010	/	/	/	Si CC indique "retenue égale à 1"	RET
1011 à 1110	/	/	/	NON ATTRIBUEES*	
111		/		Inégalité à zéro du registre RCOMPT après incrémentation d'une unité.	DZRCOMPT
111	/		/	NON ATTRIBUEES*	

* Les codes non encore attribués, sont assimilés à des branchements inconditionnels, mais ils restent disponibles pour le test des opérateurs spécialisés.

Figure IV.23 : CODES DU BRANCHEMENT RELATIF

Les bits 19 à 25 de ces microinstructions contiennent la valeur de l'éventuel branchement relatif, exprimée en complément à deux : rang des valeurs comprises entre - 64 et + 63.

Remarque : Une microinstruction dont le code opération est le branchement absolu (BRANCH), peut aussi commander un branchement relatif, conditionnel ou inconditionnel. Dans ces cas, l'adresse de la microinstruction suivante est obtenue par l'addition algébrique de l'adresse du branchement absolu et de la valeur du branchement relatif.

g) Microinstructions du type EXEC

Les microinstructions du type EXEC utilisent l'UAL sans lui fournir directement ni le code opération ni dans certains cas, celui des options*.

Deux registres spéciaux : RCODE (0 : 15) et EXTENSION (0 : 15), sont utilisés pour déterminer le travail de l'UAL lors de l'exécution de ce type de microinstructions. Le premier de ces registres est divisé en deux sous-registres de 8 bits chacun : RCODE (0 : 7) et RCODE (8 : 15). Le contenu de l'un ou l'autre de ces sous-registres peut être utilisé comme code d'opération de l'UAL (3 bits poids forts) et comme code d'options de la même unité (5 bits poids faibles). Le 15ème bit du registre EXTENSION est employé, dans certains cas, pour déterminer lequel des deux sous-registres est employé comme source pour les deux codes de l'UAL. Le tableau suivant résume les codes retenus par l'UAL pour les microinstructions du type EXEC.

---

* Toutes les microinstructions faisant intervenir l'UAL, peuvent ne pas expliciter le code d'options, mais dans ce cas l'UAL adopte le code d'options par défaut, ce qui, comme il est exposé par la suite, n'est pas vrai pour les microinstructions du type EXEC.

La microinstruction du type EXEC fournit directement le code d'options de l'UAL ? (bits 14 : 18 = 00000)	EXTENSION(15)	Code d'opération retenu par l'UAL	Code reten l'UAL
oui	indifférent	RCODE(13:15)	bits la mi instr
non	0	RCODE (5:7)	RCODE
non	1	RCODE (13:15)	RCODE

L'utilisation de ce type de microinstruction est particulière intéressante pour effectuer des opérations itératives (divisions, décal cadrages, etc...).

h) Alarmes internes

Les alarmes internes détectées en cours d'exécution d'une microinstruction, sont consignées dans le registre spécial RERR (0 : 15 le fonctionnement est exposé au paragraphe IV.5.a.

Les tableaux suivants résument les fonctions associées aux différents champs de ce registre.

RERR (0:2)	Compteur du nombre de conditions anormales détectées : 000 S'il n'y en a pas, 100 s'il y en a une, 110 pour deux et 111 pour trois ou plus.
RERR (3:4)	Déterminent le microprocesseur associé à la première condition anormale détectée : 10 pour EXT, 01 pour OP et 00 pour RAN.
RERR (5:15)	Déterminent la ou les alarmes internes ayant provoqué la première condition anormale détectée.

bit	Microprocesseur EXT	OP	RAN	Alarme interne associée
RERR(5)	/	/	/	Dépassement des bornes dans la mémoire de commande.
RERR(6)	/			La file des vecteurs d'état est pleine lors de l'exécution d'une m.i de type ENTRER, ou bien elle est vide lors de celles des types RETOUR ou SORTIR
RERR(6)		/	/	NON ATTRIBUE
RERR(7)	/	/	/	Débordement dans l'unité UAL
RERR(8)		/		NON ATTRIBUE
RERR(9)	/			La file de sauvegarde de la mémoire principale est pleine ou vide respectivement, lors de l'exécution d'une m.i du type SAUV-FILES ou REST-FILES.
RERR(9)		/	/	NON ATTRIBUE
RERR(10)	/		/	Dépassement des bornes de la mémoire principale
RERR(10)		/		NON ATTRIBUE
RERR (11:15)	/	/	/	NON ATTRIBUES

- C H A P I T R E V. -

A S P E C T S T E C H N O L O G I Q U E S

Nous abordons ici quelques aspects d'une possible réalisation du réseau exposé au chapitre précédent : technologie des circuits, découpage et système d'horloges.

D'autre part, puisque l'issue d'un certain nombre de conflits entr microprocesseurs, au niveau de l'accès des registres communs dépend des choi technologiques, nous exposons ici comment ces conflits sont résolus dans le cadre de ce projet de réalisation.

1. TECHNOLOGIE DES CIRCUITS

Etant données les performances souhaitées pour le réseau, et à l'état actuel de développement des circuits, il est impossible d'envisager une réalisation en technologie MOS. Nous avons donc le choix entre TTL et EC

Nous retiendrons les circuits TTL - dans leurs différentes séries étant donné que :

- Il est possible d'atteindre les performances minimales souhaitées, et ceci à un prix inférieur à celui d'une réalisation en technologie ECL.
- La gamme des produits du catalogue TTL présente des circuits à haut degré d'intégration (LSI) bien adaptés à cette réalisation.

2. DECOUPAGE DU RESEAU, CONNEXIONS

Nous avons déjà dit que le réseau doit être bien adapté pour recev des opérateurs spécialisés, opérateurs reliés au réseau à travers les bus deuxième et troisième opérande, et éventuellement, par un bus pour le code d'opération, la validation, les alarmes internes et le système d'horloges.

Nous avons cherché à minimiser le nombre de fils nécessaires pour relier les différentes cartes du réseau, ce qui nous a amenés à retenir un découpage en six parties. Chacune de ces parties est implantée sur une carte* dont les fonctions principales sont :

Carte # 1 (≈ 90 boîtiers) :
- Mémoire de contrôle avec tous les circuits pour sa gestion
- Sémaphores parallèles

Carte # 2 (≈ 80 boîtiers) :
- 16 registres généraux
- Mémoire locale

Carte # 3 (≈ 80 boîtiers) :
- Unité arithmétique et logique avec ses registres associés
- Circuit de calcul des tests

Carte # 4 (≈ 90 boîtiers) :
- Gestion des appels externes (interruptions) et des alarmes internes
- File de sauvegarde automatique des vecteurs d'état

Carte # 5 (≈ 70 boîtiers) :
- Multiplieur rapide

Carte (S)** # 6 :
- Mémoire principale avec circuits de gestion
- Files de lectures et écritures de la mémoire principale
- File de sauvegarde
- Logique permettant un accès direct de la mémoire principale

---

* La mémoire principale et ses circuits de gestion peuvent demander plusieurs cartes selon la capacité de cette mémoire.

** Cette partie du réseau comporte ≈ 90 boîtiers plus le nombre de boîtiers propre à la mémoire principale proprement dite.

La figure V.1 illustre les connexions principales entre ces car

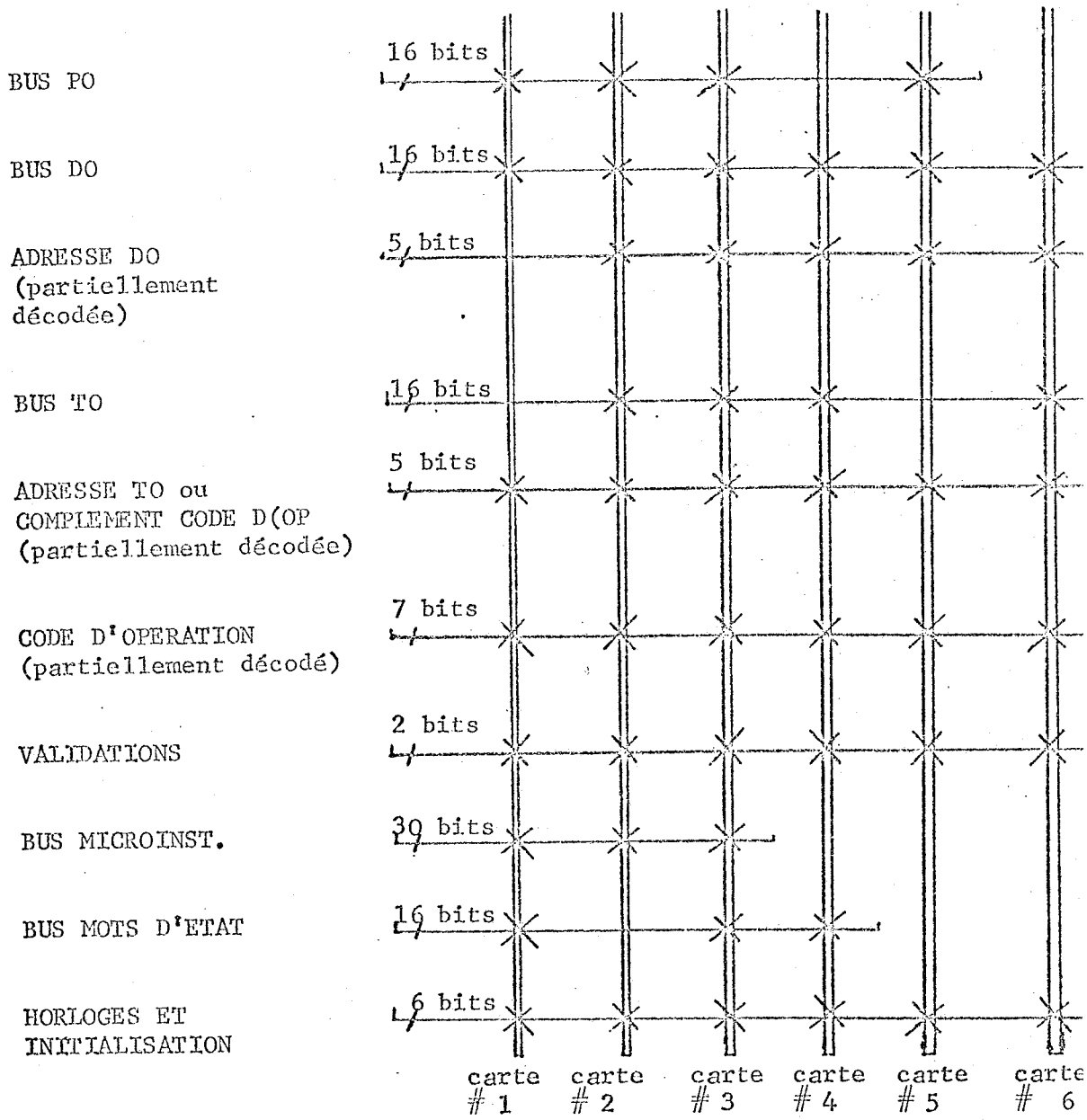


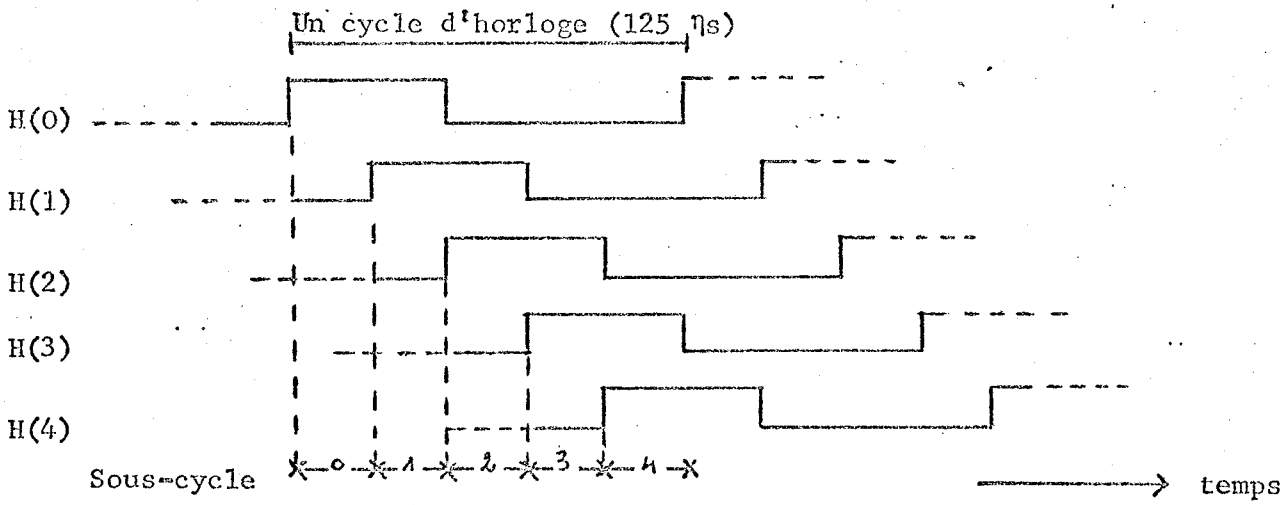
Figure V.1. : PRINCIPALES CONNEXIONS ENTRE LES CARTES DU RESEAU

Compte tenu d'autres connexions mineures, un connecteur à 130 placé à l'arrière de chaque carte, est suffisant pour relier les cartes elles. Les éventuelles connexions entre chaque carte et le milieu extern doivent être assurées par des connecteurs placés en partie avant.

3. SYSTEME D'HORLOGES

La synchronisation du réseau est obtenue à partir d'une horloge dont le cycle est de 125 ηs. Un cycle d'horloge correspond au temps nécessaire pour exécuter chacune des trois phases du réseau. Ainsi, le temps global d'exécution de chaque microinstruction est égal à 375 ηs *.

Pour pouvoir respecter les temps de préparation et permanence** propres aux différents registres et bascules employés, chaque cycle d'horloge est décomposé en 5 sous-cycles, caractérisés par les sous-horloges H(0 : 4) :



---

* Etant donné que le réseau exécute trois microinstructions quasi-parallèlement (c.f. IV.2), le temps apparent consacré à l'exécution de chaque microinstruction est de 375/3 = 125 ηs.

** En anglais "set-up and hold times"

L'adoption de ce système de synchronisation ne va pas sans conséquences. La figure IV.2 illustre le déroulement dans le temps des actic concernant exclusivement le chemin de données, déclenchées par l'exécution d'une microinstruction faisant intervenir l'UAL :

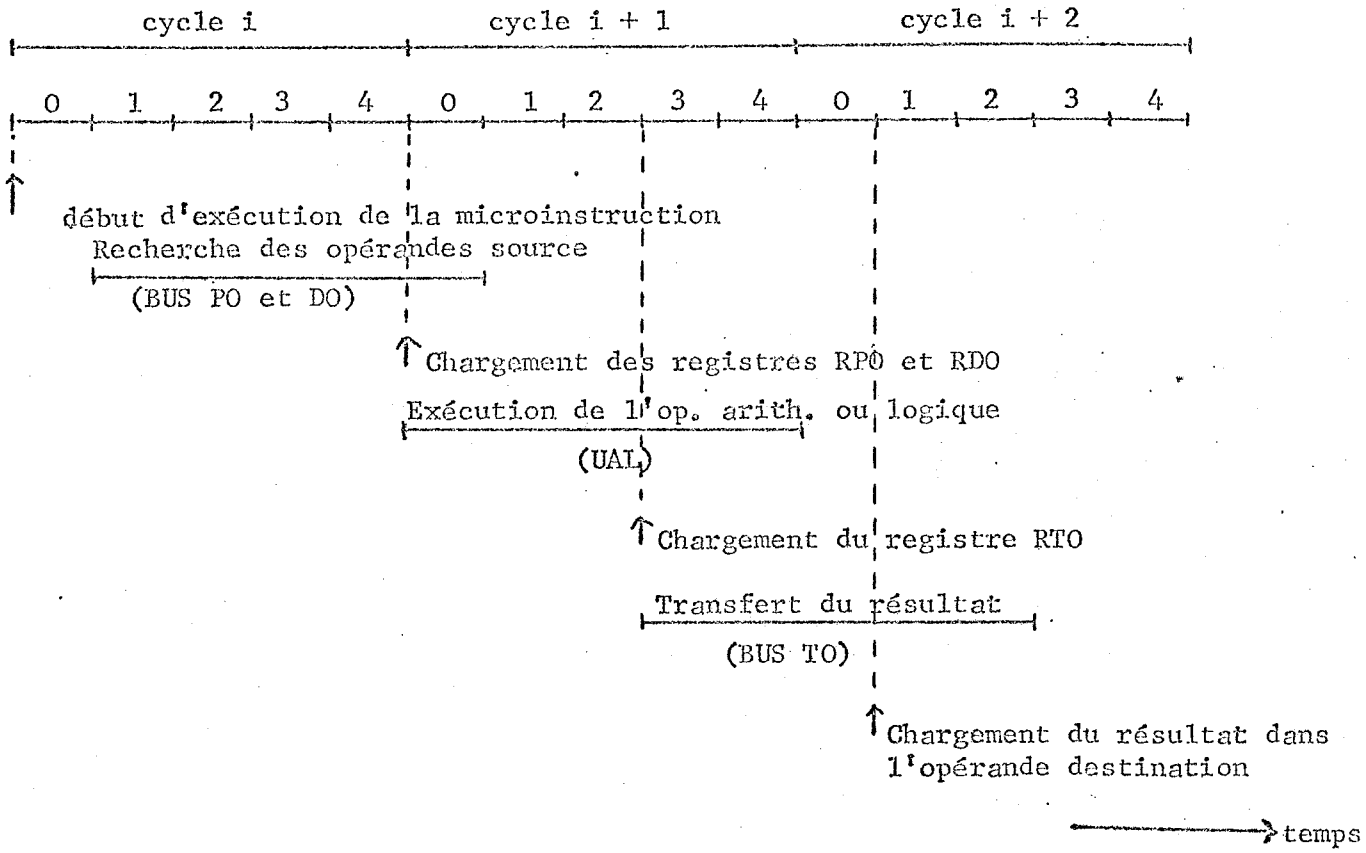


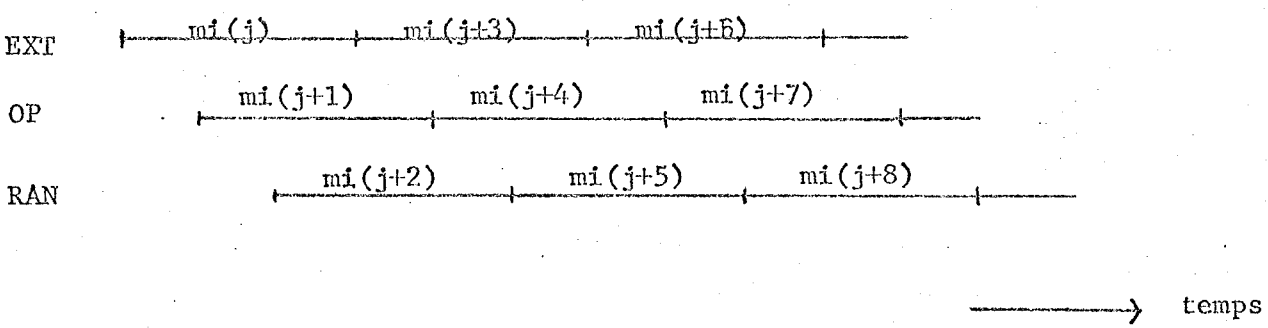
Figure V.2. : DEROULEMENT DES ACTIONS DANS LE CHEMIN DE DONNEES

La notion de phases d'exécution des microinstructions bien déterminées dans le temps, donnée au chapitre IV, devient très floue dans la réalité. Ici, les frontières entre deux phases ne sont pas bien définies, et certaines actions, tel le chargement du résultat de l'UAL dans l'opérande destination, sont effectuées beaucoup plus tôt que prévu. Cette caractéristi est commune à tous les éléments du réseau, mais elle n'a pas d'incidence sur son comportement logique, sauf cas de conflits entre microprocesseurs vis-à-de l'accès aux registres communs.

4. CONFLITS ENTRE MICROPROCESSEURS VIS-A-VIS DE L'ACCES AUX REGISTRES COMMUNS

Au cours de l'exécution de leurs microprogrammes respectifs, il se peut que deux ou même les trois microprocesseurs du réseau utilisent un certain nombre de registres communs quasi simultanément, donnant lieu à un conflit : que se passe t-il lorsque un même registre est sélectionné en lecture et écriture par deux microprocesseurs différents ?

Considérons la suite de microinstructions mi(j), mi(j+1), mi(j+2),... traitées par le réseau à partir d'un instant quelconque (mi(j), mi(j+3),..., etc. propres à EXT, mi(j+1), mi(j+4),..., etc. propres à OP, mi(j+2), mi(j+5),..., etc. propres à RAN) :



Appelons R(j) l'ensemble des valeurs de tous les registres du réseau accessibles aussi bien par le bus deuxième opérande (en lecture) que par le bus troisième opérande.

D'après ce que nous avons vu au paragraphe précédent, le chargement de l'opérande destination intervient 11 sous-cycles après le début d'exécution d'une microinstruction idoine (c.f. figure V.2). Le résultat d'une opération arithmétique ou logique commandée par la microinstruction mi(j) sera rangé dans l'opérande destination après la lecture des opérandes source de la microinstruction mi(j+1), mais avant la lecture de ceux de la microinstruction mi(j+2). D'une manière générale, les opérandes source d'une microinstruction mi(k) sont obtenues de l'ensemble de valeurs R(k-2).

Remarque : Ce décalage entre R(j) et mi(j+2) ne s'applique pas aux sémaphor parallèles du réseau. Les résultats des primitives de changement d'état interviennent immédiatement; c'est-à-dire, pour l'exécuti d'une microinstruction mi(j+1) on considère les valeurs des couples de sémaphores résultant de l'exécution de mi(j) (c.f. IV

5. MEMOIRE PRINCIPALE

Etant données les tendances technologiques actuelles, nous avons prévu les circuits de l'opérateur mémoire principale pouvant gérer des mémoires dynamiques MOS. En particulier, cet opérateur comporte des outils pour faciliter les rafraîchissements (en anglais "refresh") inhérents à ce type des mémoires. La capacité de la mémoire peut aller de 4 à 64 K mots de 16 bits par blocs de 4 K mots.

D'autre part, l'analyse des performances recherchées dans l'exécu d'un certain nombre d'algorithmes clés (transformée de Fourier rapide, filt numérique), nous a montré que le temps de cycle de cette mémoire doit être l'ordre de 600 ηs.

6. SIMULATION DU RESEAU EN LANGAGE CASSANDRE [16]

Actuellement, les circuits logiques composant chacune des cartes exposées au paragraphe V.2, sont, en très grande partie, définis. A partir ces circuits nous avons établi un programme de simulation en langage CASSANDRE. Cette simulation a été effectuée à un niveau assez fin : sans a au niveau de la porte, on respecte assez fidélement toutes les fonctions logiques et tous les éléments de mémorisation. Ainsi, les programmes de simulation constituent, non seulement un moyen pour vérifier le fonctionnem logique du réseau, mais aussi un outil permettant une compréhension détail du fonctionnement de chacune des parties du réseau.

En contrepartie, le temps machine employé pour exécuter une étape de simulation, est assez important*. Ainsi, la vérification d'un algorithme programmé dans le réseau devient rapidement impossible. C'est pourquoi, les exemples de microprogrammation donnés au chapitre suivant n'ont été que partiellement testés.

Nous avons établi la totalité des schémas des circuits du réseau, mais étant donné leur volume, nous avons décidé de ne pas les inclure dans ce mémoire. Nous en avons retenu seulement un ou deux pour illustrer la correspondance entre ceux-ci et le programme de simulation (c.f. APPENDICE).

---

* Une phase du réseau (temps réel = 125 ηs) est simulée en une seconde environ de temps UPC dans un IBM 360/67.

- C H A P I T R E VI -

E X E M P L E S D ' A P P L I C A T I O N

Deux types d'algorithmes sont décrits ici. Ceux du premier type, très simples, illustrent la capacité de calcul individuel du microprocesseur. Ceux du deuxième type illustrent comment, et avec quelles performances, les trois microprocesseurs du réseau coopérent à l'exécution d'un travail.

1. ALGORITHMES EXECUTES PAR LE MICROPROCESSEUR OP

On considère ici la microprogrammation de deux algorithmes : multiplication* et division en arithmétique fixe. Ces deux algorithmes sont traités par le microprocesseur OP sans l'aide de deux autres.

a) Multiplication en arithmétique fixe

Soit à multiplier les nombres contenus dans les registres R1 et EXTENSION. Le résultat de cette multiplication, un nombre de longueur double doit être rangé dans ces mêmes registres : R1 pour les poids forts et EXTENSION pour les poids faibles. Les multiplicandes, ainsi que le produit, sont représentés en complément à deux.

L'algorithme choisi se traduit par une succession d'additions et de décalages pour les 15 bits poids faibles du multiplieur, et d'une soustraction et décalage pour le signe (bit poids fort). L'exécution de cet algorithme fait intervenir 6 registres :

---

* Le microprocesseur OP dispose d'un multiplieur rapide (c.f. IV.6.d). Ce n'est donc qu'à titre d'illustration que nous traitons ici l'algorithme de multiplication.

Registre	Valeur contenue avant exécution	Valeur contenue après exécution
R0	'0000'*	'0000'
R1	Multiplicande	16 bits poids forts du produit
R2	"	Multiplicande
EXTENSION	Multiplieur	16 bits poids faibles du produit
RCOMPT	"	'0000'
RCODE	"	'3BB9'

Les figures VI.1 et VI.2 représentent respectivement l'organigramme et le microprogramme décrivant cet algorithme. Le microprogramme est composé de 9 microinstructions. Une multiplication est obtenue en 20 cycles d'exécution, c'est-à-dire 7,5 microsecondes (un cycle d'exécution "CE" correspond à 375 ηs. c'est-à-dire, le temps d'exécution d'une microinstruction. c.f. V.3).

---

* Une constante binaire de 16 bits est représentée par quatre digits hexadécimaux entre apostrophes. Les valeurs 0 à 15 d'un digit hexadécimaux sont représentées par : 0, 1,...9, A, B, C, D, E et F.

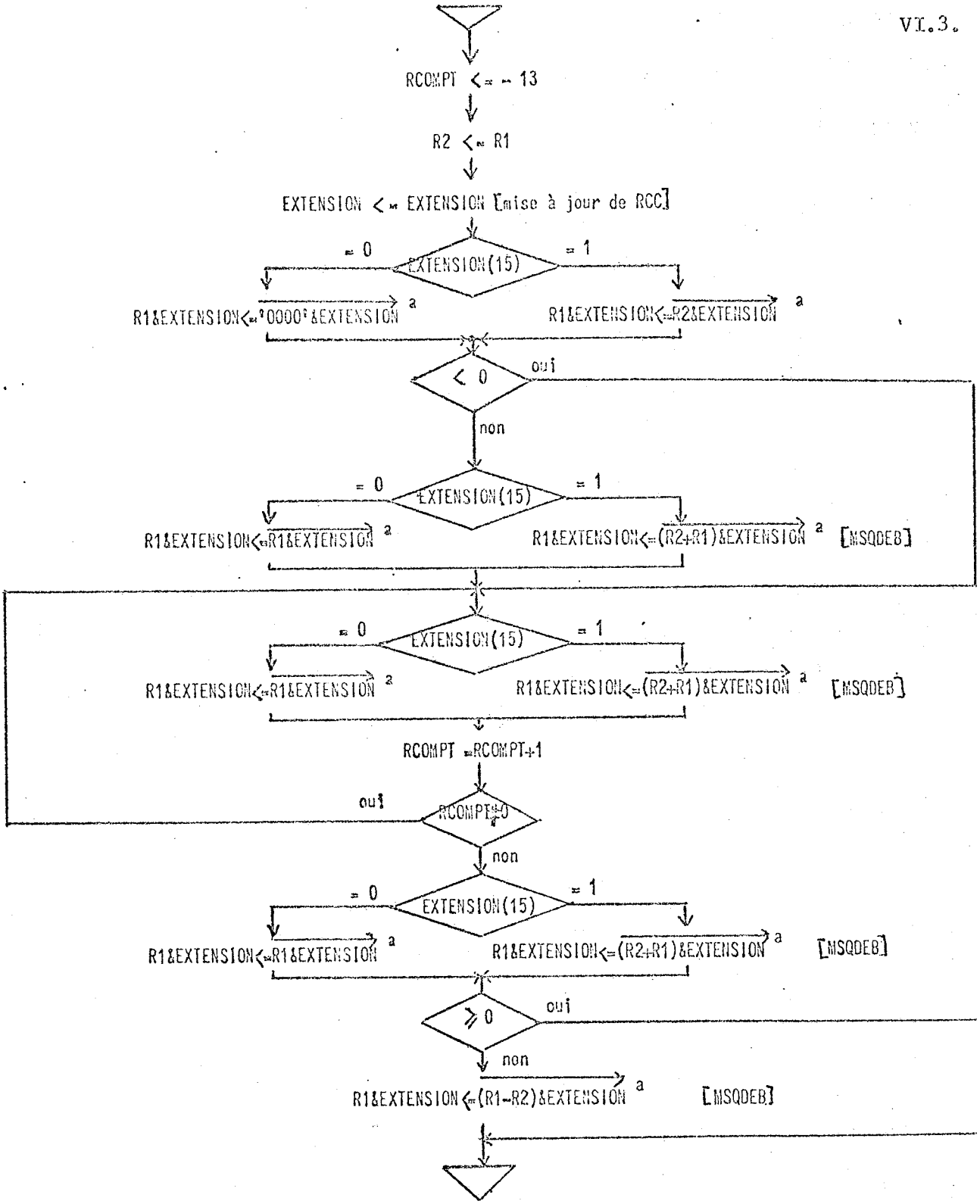


Figure VI.1. : ORGANIGRAMME DECRIVANT L'ALGORITHME DE MULTIPLICATION

N° de m.i.	Microprogramme		
1	ADD	('FFF3', RO, RCOMPT)	*C Point d'entrée du microprogramme
2	ADD	('0000', R1, R2)	
3	ADD	('FBF9', RO, RCODE)	*C Chargement du registre code d'opération
4	ADD	(RO, EXTENSION, EXTENSION)	POSRCC
5	EXEC	(R2, RO, R1)	PZ + 2
6	EXEC	(R2, R1, R1)	INC + 1
7	EXEC	(R2, R1, R1)	DZRCOMPT + 0
8	EXEC	(R2, R1, R1)	GEZ + 2
9	ST	(R1, R2, R1)	MSQDEB, DAE *C Dernière microinstruction du microprogramme

Première m.i. après l'exécution du microprogramme

a) Codification symbolique

N° de m.i.							
1	0001	10010	00000	1	1111	1111111	0011
2	0001	00010	00001	1	0000	0000000	0000
3	0001	10100	00000	1	1111	0111111	1001
4	0001	10101	10101	0	0000	0100000	0000
5	1001	00001	00000	0	1000	0000010	0010
6	1001	00001	00001	0	0010	0000001	0010
7	1001	00001	00001	0	1111	0000000	0010
8	1001	00001	00001	0	0110	0000010	0010
9	0101	00001	00010	0	0000	1011100	0001

b) Codification binaire

Figure VI.2. : MICROPROGRAMME DE MULTIPLICATION EN ARITHMETIQUE FIXE

b) Division en arithmétique fixe

Soit à diviser un nombre de longueur double (32 bits), contenu d
les registres R1 - poids forts - et EXTENSION - poids faibles -, par un no
de longueur simple (16 bits) contenu dans le registre R2. Le quotient et l
reste de cette division, deux nombres de longueur simple, sont rangés respectivement dans les registres R1 et EXTENSION. Tous ces nombres sont représentés en complément à deux.

L'algorithme choisi détermine la valeur absolue du diviseur et d
dividende et procède à une succession de seize décalages et additions ou soustractions. A la fin de ces opérations, on restitue le signe au quotien
au reste.

L'exécution de cet algorithme met en oeuvre 9 registres :

Registre	Valeur contenue avant exécution	Valeur contenue après exécution
R0	'0000'	'0000'
R1	Poids forts dividende	Quotient
R2	Diviseur	Diviseur
R3 R4 et R5	-	-
RCOMPT	-	'0000'
EXTENSION	Poids faibles dividende	Reste
RCODE	-	-

Les figures VI.3 et VI.4 représentent respectivement l'organigramme et le microprogramme décrivant cet algorithme. Le microprogramme est composé de 22 microinstructions, et son exécution demande un nombre de CE dépendant des signes des opérandes :

SIGNE DIVIDENDE	SIGNE DIVISEUR	NB. DE 'CE'	TEMPS D'EXECUTION (microsecondes)
+	+	29.	10,875
+	-	30	11,250
-	-	31	11,625
-	+	32	12,0

Remarque : Pour que la division soit possible, cette méthode exige que le quotient soit inférieur à 2^{15}. Si ce n'est pas ainsi, le microprogramme prévoit un branchement vers une routine pour le traitement d'erreurs.

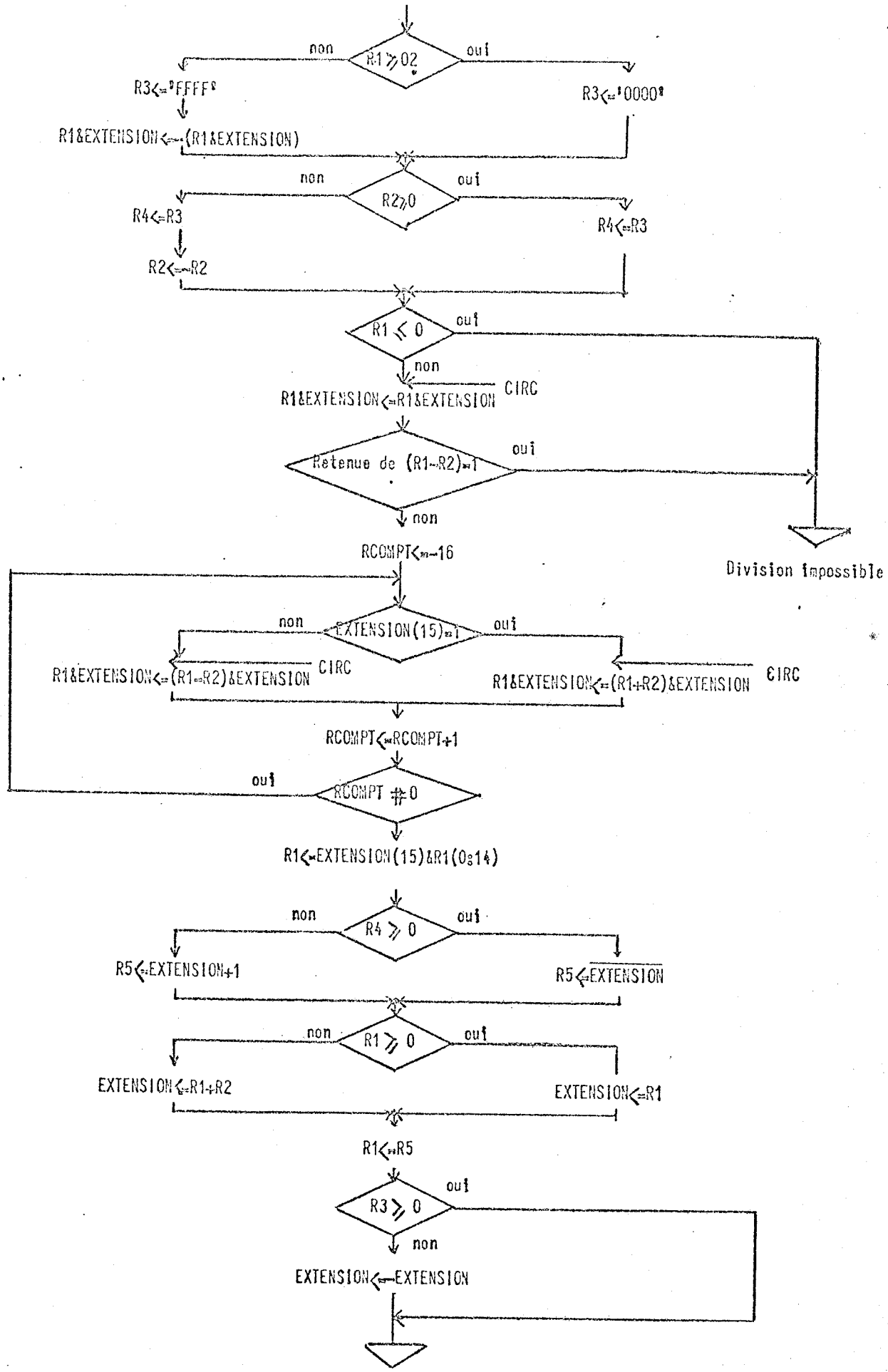


Figure IV.3. : ORGANIGRAMME DECRIVANT L'ALGORITHME DE DIVISION

```
ADD      (RO, R1, R1); POSRCC          *C Point d'entrée du microprogramme
CONJ     (R3, R3, R3); GEZ + 9
ST       (RO, EXTENSION, EXTENSION);
         MSQDEB, POSRCC
ST-RET   (RO, R1, R1); MSQDEB
ADD      (RO, R2, R2); POSRCC
ADD      (RO, R3, R4); PZ + 3
ADD      ('F5F1', RO, RCODE)
EXEC     (R1, RO, R1); INC + 5
ADD      (F1F5', RO, RCODE)
CONJ     (RO, R3, R4); INC - 2
ADD      (RO, R3, R3); INC - 6
ADD      (RO, R1, EXTENSION); INC+9
EXEC     (R2, R1, R1); PZ + x          *C Branchement relatif de x unités si
                                          division impossible
EXEC     (R2, R1, R1); RET+(x-1)       *C Branchement relatif de x-1 unités
                                          si division impossible
ADD      ('FFF2', RO, RCOMPT)
EXEC     (R2, R1, R1); DZRCOMPT+0
ADD      (RO, R1, R1); DSS
CONJ     (R4, EXTENSION, R5)
EXEC     (R3, RO, R3); GEZ - 7
EXEC     (R1, R2, EXTENSION)
ST       (R5, R4, R1); GEZ+2
ST       (RO, EXTENSION, EXTENSION)    *C Dernière m.i. du microprogramme
```

Première m.i. après l'exécution du microprogramme

Figure IV.4. : MICROPROGRAMME DE DIVISION EN ARITHMETIQUE FIXE

2. ALGORITHMES FAISANT INTERVENIR L'ENSEMBLE DU RESEAU

Nous considérons ici deux algorithmes de traitement du signal : filtrage numérique linéaire et transformée de Fourier rapide (TFR), permettant d'illustrer comment, et avec quelles performances, les trois microprocesseurs du réseau coopérent à l'exécution d'un travail.

a) Filtrage numérique linéaire [17]

Considérons le filtre numérique linéaire complet de deuxième de représenté par la fonction de transfert suivante :

$$HE(Z) = \frac{S(Z)}{E(Z)} = \frac{a_o + a_1 \cdot Z^{-1} + a_2 \cdot Z^{-2}}{1 + b_1 \cdot Z^{-1} + b_2 \cdot Z^{-2}}$$

Z^{-1} = retard unitaire = $\exp(-j\,\omega\,T)$

$j = \sqrt{-1}$; T = période d'échantillonnage

La fonction HC(Z) est assurée par une cellule, dont la réalisat: est simplifiée (du point de vue du nombre d'éléments de mémorisation) par l'introduction d'une variable intermédiaire $\omega(Z)$:

$$\omega(Z) = \frac{E(Z)}{1 + b_1 \cdot Z^{-1} + b_2 \cdot Z^{-2}} = E(Z) - \omega(Z)\left[b_1 \cdot Z^{-1} + b_2 \cdot Z^{-2}\right.$$

d'où

$$S(Z) = HC(Z).E(Z) = \omega(Z).\left[a_o + a_1 \cdot Z^{-1} + a_2 \cdot Z^{-2}\right]$$

On peut démontrer que la connexion en cascade de "C" cellules adéquates permet d'obtenir toute fonction de transfert H(Z) de degré "2.C"

$$H(Z) = HC_1(Z) \cdot HC_2(Z) \cdot \ldots\ldots\ldots \cdot HC_C(Z)$$

Nous avons développé un algorithme permettant de créer jusqu'à 51 cellules. On peut ainsi réaliser "F" filtres $H_f(Z)$ différents ($f = 1, 2, \ldots, F$), chacun composé de "C" cellules ($2 \leqslant F.C \leqslant 51$) :

$$H_f(Z) = \prod_{i=1+(f-1).C}^{f.c} HC_i(Z)$$

Les variables intermédiaires "$\omega(Z)\ Z^{-1}$" et "$\omega(Z)\ Z^{-2}$" de chaque cellule sont rangées en mémoire principale. Elles forment un tableau de "Z.F.C" variables dont le dernier élément occupe la place "Bω-1".

Les paramètres : $-b_1$, $-b_2$, a_o, a_1 et a_2 de chaque cellule sont rangés en mémoire locale. Ils forment un tableau de "5.F.C" paramètres dont le premier élément occupe la place "256-5.F.C".

Les entrées/sorties de données nécessaires sont effectuées par le sous-canal 1. On suppose qu'il est capable d'exécuter ces deux opérations simultanément et qu'il est initialement programmé pour échanger "F.NBE" données (NBE = nombre d'échantillons par filtre) à partir du moment où le microprocesseur OP commande la première entrée.

L'algorithme de filtrage est décomposé en 3 tâches exécutables par EXT, OP et RAN :

- Tâche EXT : - Gestion des lectures dans le tableau de variables intermédiaires
- Tâche OP : - Gestion des lectures des paramètres de chaque cellule
  - Calcul des fonctions propres à chaque cellule
- Tâche RAN : - Gestion des écritures dans le tableau des variables intermédiaires
  - Gestion du séquencement des trois tâches

L'exécution de cet algorithme met en oeuvre 17 registres, "5.F.C" mots de la mémoire locale et "Z.F.C" mots de la mémoire principale :

	Notes	Valeur initiale	Valeur finale
R0	Constante	0	0
R1	Constante	1	1
R2	Long. tableau variables int.	2.F.C	2.F.C
R3	Long. tableau paramètres	5.F.C	5.F.C
R4	Nombre de cellules/filtre	C	C
R5	Base du tableau variables int.	BW	BW
R6	Pointeur en lecture du tableau de variables int. (VW)	-	0
R7	Pointeur en écriture du tableau de variables int (VW')	-	0
R8	Compteur du nombre d'échantillons/ filtre (VD)	NBD	0
R9	Gestion exécution des tâches (GES)	-	264
R10-R12	Reg. de travail de OP	-	-
R13	Compteur du Nb. de cellules/ filtre (VC)	c	0
R14	Registre de travail RAN	-	-
RCOMPT	Pointeur mémoire locale	-	0
RCODE	Code opération : ST Code d'options : POSRCC	'4545'	'4545'
ML	Tableau de paramètres ML(256-5.F.C) à ML(255)	Paramètres de chaque cellule	Sans modificatio
Mémoire centrale	Tableau de variables int. BW-5.F.C à BW-1	toutes les variables int. égales à 0.	-

Les couples de sémaphores A-A' et B-B' doivent avoir été initiali
à la valeur 0.

La figure VI.5 représente les organigrammes des trois tâches : TEXT, TOP et TRAN, qui composent l'algorithme de filtrage numérique linéaire. Les figures VI.6 et VI.7 représentent les microprogrammes associés à chacune de ces tâches. Ils sont constitués respectivement de 8, 22 et 14 microinstructions, soit 44 microinstructions pour programmer l'ensemble de l'algorithme. Pour chacun des filtres ainsi réalisé, la période T minimale devant séparer deux échantillons est donnée par :

$$T = 0{,}375\ (1 + 3 \cdot F + 15 \cdot F \cdot C) \quad [\text{microsecondes}]$$

Le tableau suivant donne les valeurs de fréquence d'échantillonage maximale pour un certain nombre de configurations.

Nombre de filtres F	Nombre de cellules/filtre C	Degré de chaque filtre	fréquence minimale d'échantillonage/filtre fmin [Khz]	F.fmin [KhZ]
2	1	2	72,1	144,1
4	1	2	36,5	146,1
8	1	2	18,4	147,1
1	2	4	78,4	78,4
2	2	4	39,8	79,6
4	2	4	20,1	40,1
1	4	8	41,7	41,7
2	4	8	21,0	42,0
1	8	16	21,5	21,5

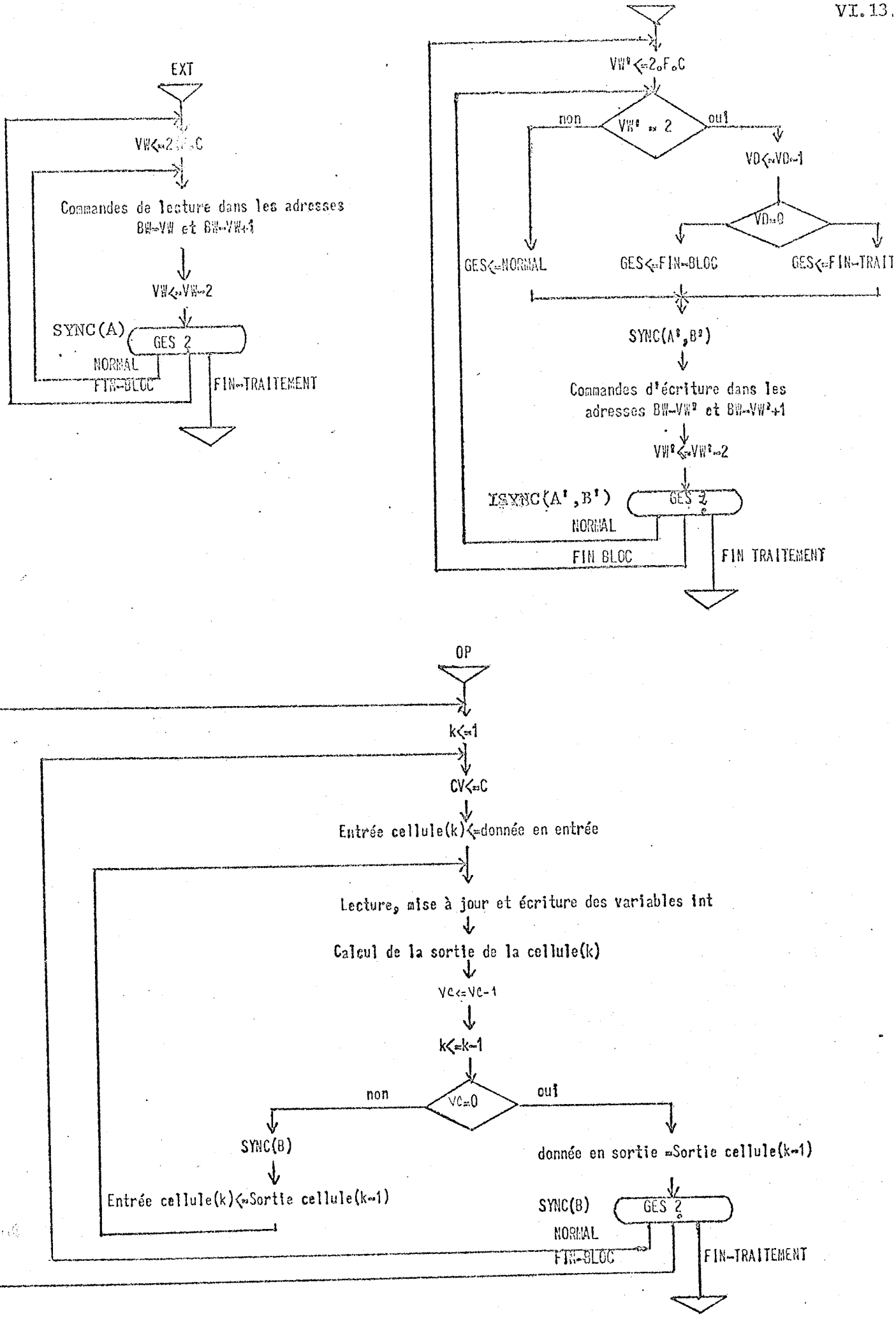


Figure VI.5 : ORGANIGRAMMES DECRIVANT L'ALGORITHME DE FILTRAGE NUMERIQUE LI

```
N° de
 m.i.

256 BRANCH ('XXXX')                    *C Branchement à l'adresse 'XXXX' en
                                          fin de m.p

257 DOP (    , R2, R6)                 *C Point d'entrée du m.p. EXT
258 ST (R5, R6, RADLEC)
259 ST (R6, R1, R6)
260 ST (R5, R6, RADLEC)
261 ST (R6, R1, R6)
262 DOP (    , R1, R1); SYNC(A)
263 BRANCH (R9); INC=8
```

a) Microprogramme EXT

```
N° de
 m.i.

286 BRANCH ('ZZZZ')                    *C Branchement à l'adresse 'ZZZZ' en
                                          fin de m.p.

287 DOP (    , R2, R7)                 *C Point d'entrée du m.p. RAN
288 ADD ('010A', RO, R9)
289 EXEC ('0002', R7, R14)
290 EXEC (R8, R1, R14); DZ + 4         *C Sans branchement relatif si fin
                                          du bloc-filtre

291 DOP (    , R14, R8); EZ + 2        *C Avec branchement relatif si fin
                                          du filtrage

292 ST (R9, R1, R9); INC + 2
293 ADD ('0108', RO, R9)
294 DOP (    , R1, R1); SYNC(A',B')
295 ST (R5, R7, RADECR)
296 ST (R7, R1, R7)
297 ST (R5, R7, RADECR)
298 ST (R7, R1, R7); ISYNC (A', B')
299 BRANCH (R9); INC + 22
```

b) Microprogramme RAN

Figure VI.6 : MICROPROGRAMMES POUR EXT ET RAN - FILTRAGE NUMERIQUE LINEAIRE

```
N° de
 m.i.

264 BRANCH ('YYYY')                    *C Branchement à l'adresse 'YYYY'
                                          en fin de m.p.
265 DOP (   , R3, RCOMPT)              *C Point d'entrée du m.p. OP
266 DOP (   , RTE1, R10); DAE
267 DOP (   , FLEC, R11)
268 MULT (R11, ML); DZRCOMPT + 1
269 DOP (   , FLEC, R12)
270 ADD (R10, PFORT, R10)
271 MULT (R12, ML); DZRCOMPT + 1
272 ST (R13, R1, R13); POSRCC
273 ADD (R10, PFORT, R10); GAS
274 MULT (R10, ML); DZRCOMPT + 1
275 DOP (   , R10, FECR)
276 DOP (   , PFORT, R10)
277 MULT (R11, ML); DZRCOMPT + 1
278 DOP (   , R11, FECR)
279 ADD (R10, PFORT, R10)
280 MULT (R12, ML); EZ + 3           *C Branchement relatif si fin d'un
                                          filtre
281 DOP (   , FLEC, R11); SYNC (B)
282 ADD (R10, PFORT, R10); DZRCOMPT = 14
283 DOP (   , R4, R13); DZRCOMPT + 1
284 ADD (R10, PFORT, RTS1); GAS
285 BRANCH (R9); SYNC(B)
```

Figure VI.7 : MICROPROGRAMME POUR OP - FILTRAGE NUMERIQUE LINEAIRE

b) Transformée de Fourier rapide [18]

La transformée de Fourier rapide (TFR) est un algorithme permettant de calculer efficacement la transformée de Fourier discrète (TFD) d'une série de N nombres complexes Xk, k = 0 à N - 1 :

$$Ar = \sum_{k=0}^{N-1} Xk \; W_N^{-rk}$$

$$r = 0 \text{ à } N - 1; \; W_N = \exp\left(\frac{2\pi j}{N}\right); \quad j = \sqrt{-1}$$

Une légère modification de l'algorithme TFR nous permet de calculer l'inverse de la transformée de Fourier discrète TFD^{-1} :

$$Xk = \frac{1}{N} \sum_{r=0}^{N-1} Ar \; W_N^{rk}$$

$$k = 0 \text{ à } N - 1$$

Il existe deux types d'algorithmes principaux de TFR : décimation en temps et décimation en fréquence. Nous considérons ici la microprogrammation d'un algorithme du deuxième type pour calculer la TFR d'une série de $N = 2^n$ nombres complexes. Cet algorithme travaille "sur place", c'est-à-dire qu'il utilise une seule zone de travail Z d'une capacité de 2.N mots* dans la mémoire principale. Initialement les nombres Xk y sont rangés dans l'ordre naturel, mais les coefficients Ar résultant de la transformation s'y trouveront en ordre inversé binaire.

---

* Chaque nombre complexe nécessite deux mots contigus : un pour la partie réelle, l'autre pour la partie imaginaire.

La TFR est exécutée en n étapes successives ($n = \log_2 N$). A chaque étape on exécute N/2 fois un ensemble d'opérations appelé "papillon" :

$$a' = a + b$$

$$b' = (a - b) \cdot W_N^{-\alpha}$$

a et b nombres complexes pris de la zone de travail Z

α entier réel fonction de l'étape et de la position de a et b dans la zone de travail. $\alpha = 0$ à $\frac{N}{2} - 1$.

Les résultats a' et b' se substituent, respectivement, à a et b dans la zone de travail.

Les différentes valeurs $W_N^{-\alpha} = \cos \frac{2\pi\alpha}{N} - j \, SIN \frac{2\pi\alpha}{N}$ * sont obtenus à partir d'une table de cosinus : C, contenue en mémoire principale et constitu de M + 1 valeurs différentes ($M = 2^m \geq N/4$) :

$$\text{Cos} \left(\frac{\pi \alpha}{2 M} \right) \qquad \alpha = 0, 1, \ldots, M$$

Enfin, comme toutes les opérations propres à un "papillon" sont exécutées en arithmétique fixe, il est possible que certaines d'entre-elles provoquent un débordement. Dans ce cas, le mécanisme des alarmes internes (c.f. IV.5) détecte cette condition anormale et fait exécuter un recadrage (division par deux de toutes les données de la zone Z).

---

* $W_{4.M}^{-\alpha} = \text{Cos} \left[\frac{\pi}{2.M} \alpha \right] - j \, \text{Cos} \left[\frac{\pi}{2.M} (M - \alpha) \right] \quad \alpha = 0, 1, \ldots, M$

$W_{4.M}^{-\alpha} = - \text{Cos} \left[\frac{\pi}{2.M} (2.M - \alpha) \right] - j \, \text{Cos} \left[\frac{\pi}{2.M} (\alpha - M) \right] \quad \alpha = M+1, M+2, \ldots 2.M$

$W_N^{\alpha} = W_{4.M}^{4.\alpha.M/N}$

L'ensemble des deux opérations : TFR et recadrage, est décomposé en trois tâches :

TEXT : - Commande des lectures dans la table C
- Commande des lectures dans la zone Z
- Gestion du séquencement de certaines opérations de la tâche TOP
- Gestion des débordements

TOP : - Exécution des "papillons"
- Le cas échénat, recadrage des données de la zone de travail Z
- Gestion de l'exécution de l'ensemble de l'algorithme

TRAN : - Commande des écritures dans la zone Z

L'exécution de ces trois tâches met en oeuvre 20 registres, un mot de la mémoire locale et 2.N + M + 1 mots de la mémoire principale :

	Notes	Valeur initiale	Valeur finale
R0	constante	0	0
R1	constante	2.N+2	2.N+2
R2	pointeur de la table de cosinus (P.COS)	M	M
R3	incrément du pointeur P.COS (INC.COS)	-4M/N	2M
R4	base de la zone de travail Z(B.DON)	adresse du premier mot qui suit la zone Z	sans changement
R5	pointeur en lecture de Z (P.DON)	N+2	2.N-1
R6	compteur du nombre d'étapes. Séparation entre deux nombres pour un papillon (SEP.DON)	N-1	0
R7	pointeur en écriture de Z (P.DON)	N+2	2.N-1
R8 à R14	registres de travail du m.p. OP	-	-
R15	gestion d'exécution des tâches (SAUT)	256	279
EXTENSION et RCOMPT	registres de travail du m.p. OP	-	-
RCODE	code d'opération ADD (, ,);DAE	'3939'	'3939'
RPRIOR	les bits (9:15) de ce registre contiennent une partie de l'adresse de branchement en cas de débordement	RPRIOR(9:15)= 1110001	sans changement
ML(0)	compteur du nombre de cadrages(CD)	0	nombre de divisions par 2 des éléments de Z
Mémoire principale	zone de travail Z B.DON-2.N à B.DON-1	Xk ordre naturel	Ar ordre 'bit inversé'

Les couples de sémaphores A-A', B-B' et C-C' doivent avoir été initialisés à la valeur 0.

Les figures VI.8, VI.9 et VI.10 représentent les organigrammes des trois tâches : TEXT, TOP et TRAN qui composent l'algorithme de TFR. Les figures VI.10, VI.11 et VI.12 représentent les microprogrammes associés à chacune de ces tâches. Ils sont respectivement constitués de 41*, 61 et 15 microinstructions, soit 117 microinstructions pour programmer l'ensemble de l'algorithme. Un "papillon" est exécuté en 19 ou 21 CE (7,125 ou 7,875 η sec) selon que l'on doit ou non renouveler la valeur du coefficient W_N. Une transformée de Fourier discrète portant sur 1024 nombres complexes est exécutée en 37,28 + 2,80 d m. sec, d étant le nombre de cadrages effectués.

---

* Nous n'avons pas compté les microinstructions propres au microprogramme de gestion des alarmes internes, car celui-ci est commun à tous les algorithmes du réseau.

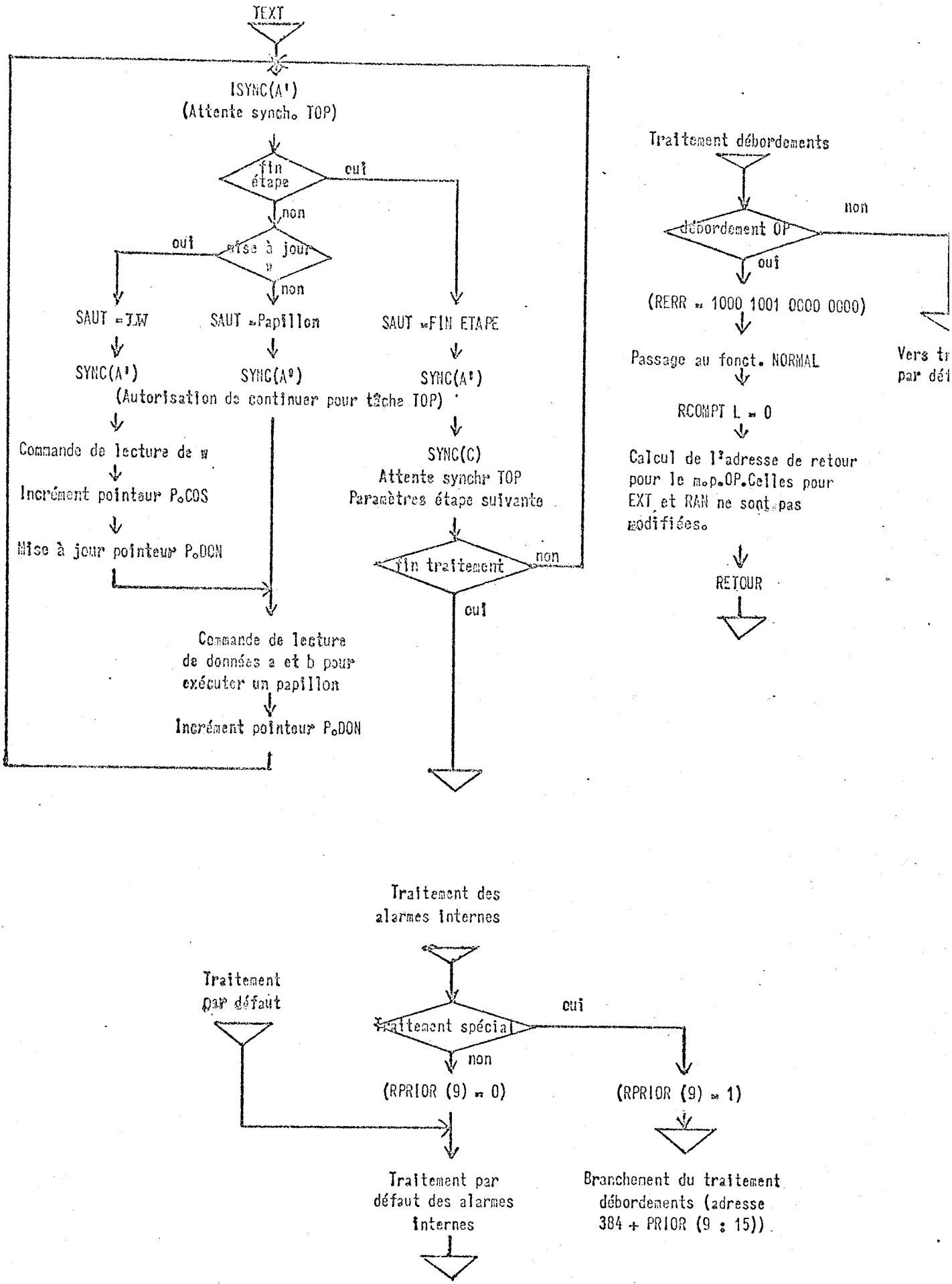


Figure VI.8 : ORGANIGRAMME DE LA TACHE TEXT DE L'ALGORITHME FFT

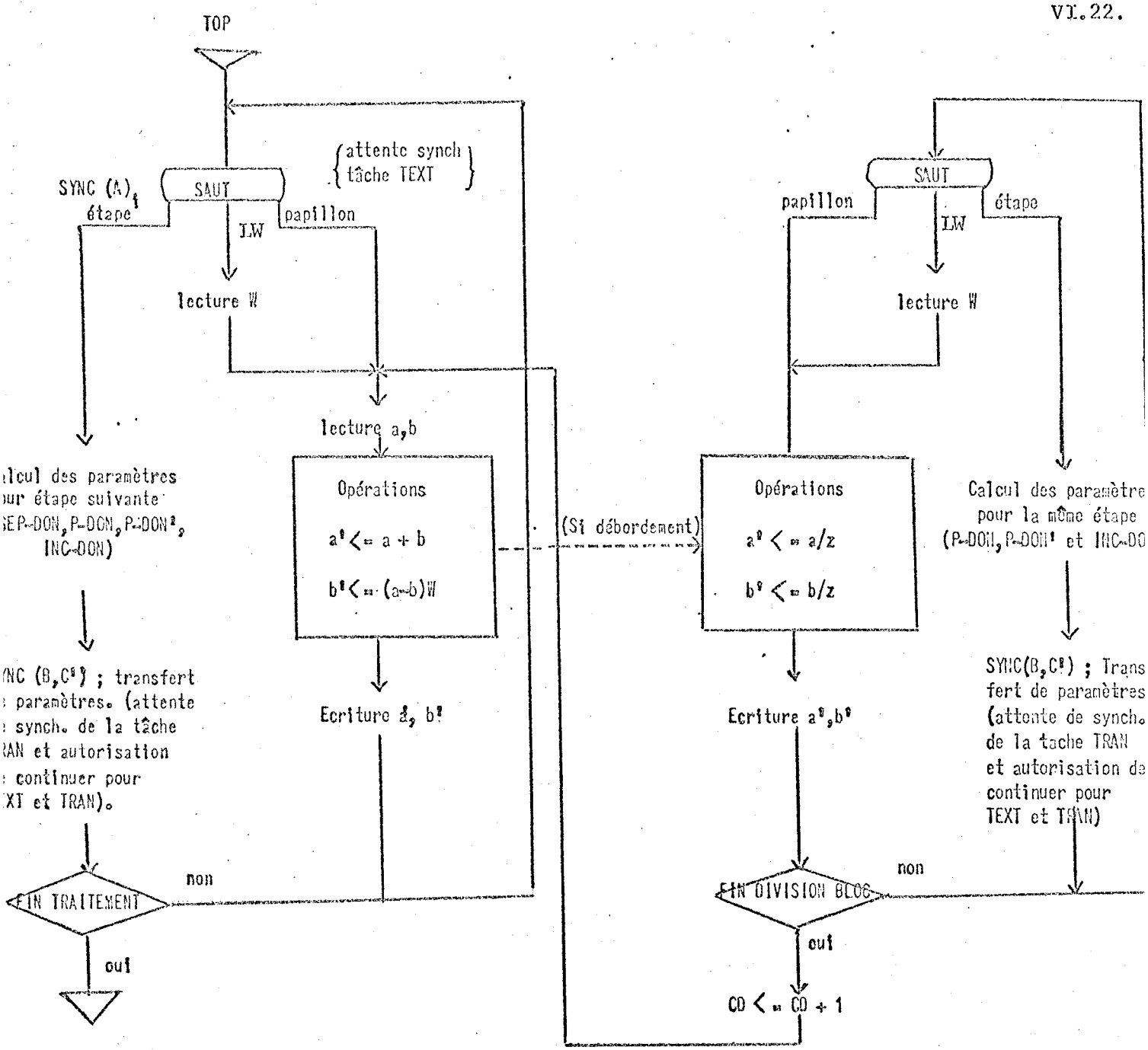


Figure VI.9 : ORGANIGRAMME DE LA TACHE TOP DE L'ALGORITHME FFT

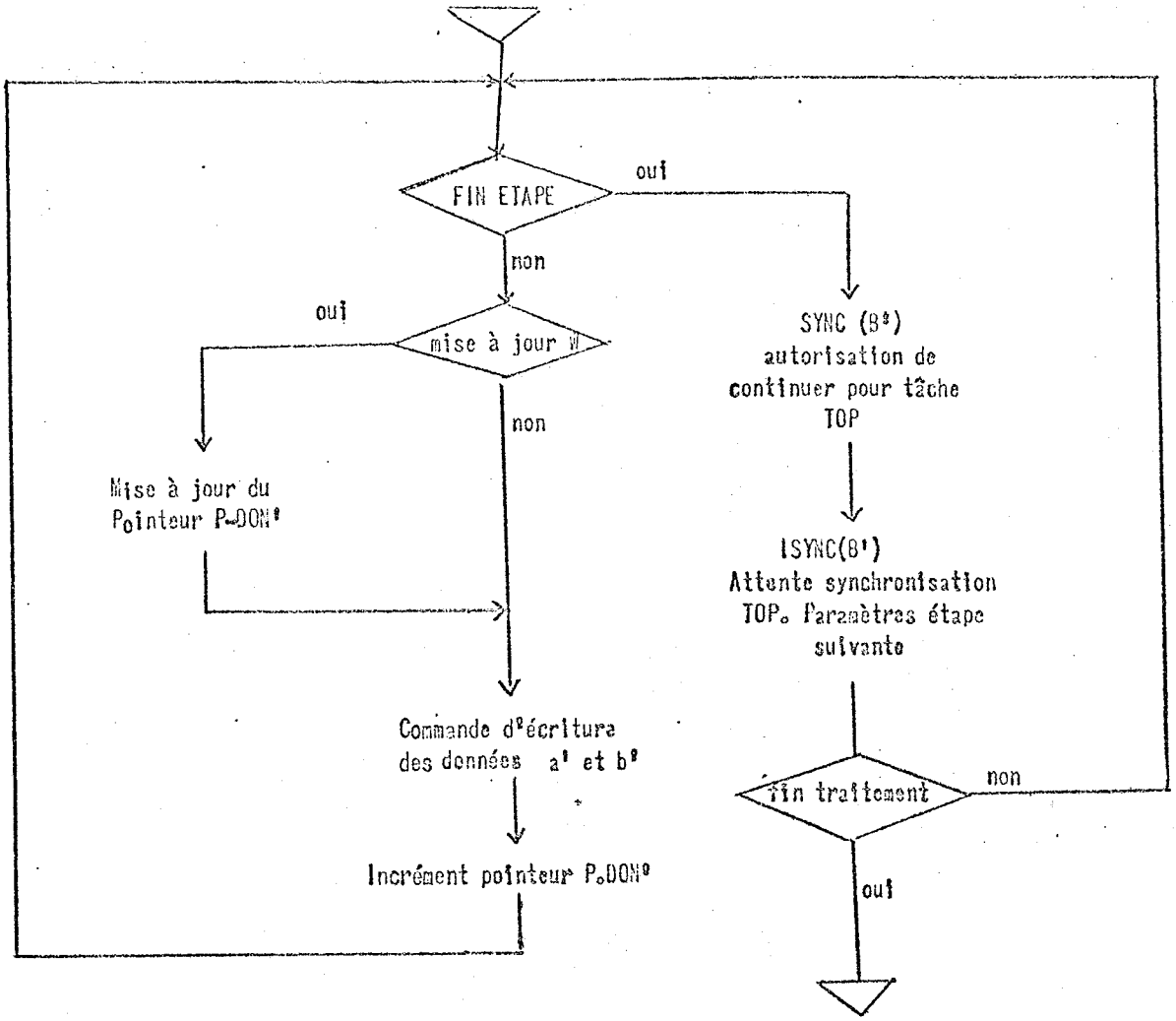


Figure VI.10 : ORGANIGRAMME DE LA TACHE TRAN DE L'ALGORITHME FFT

N° de m.i.

```
321 ET ('FFFB', R15, R15)                    *C Point d'entrée du micro-
322 ADD (R2, R3, R2); POSRCC                    programme EXT
323 ET (RO, RO, RO); GZ + 3
324 OU ('0001', R15, R15)
325 ST (RO, R3, R3); PZ - 3
326 ST ('vvvv', R2, RADLEC)                  *C 'vvvv' : adresse du premier
                                                élément de la table COSINUS
327 ET (RO, RO, RO); SYNC(A')                *C Libération de OP
328 ADD ('vvvv', R2, RADLEC)                 *C 'vvvv' : adresse du dernier
                                                élément de la table COSINUS
329 ST (R4, R5, RADLEC); INC + 3
330 OU ('0004', R15, R15)
331 ST (R4, R5, RADLEC); SYNC(A')            *C Libération de OP
332 ADD ('FFFF', R5, R5)
333 ST (R4, R5, RADLEC)
334 ST (R5, R6, R5)
335 ST (R4, R5, RADLEC)
336 ADD ('FFFF', R5, R5)
337 ST (R4, R5, RADLEC); ISYNC(A')          *C Attente OP
338 ST (R5, R6, R5); POSRCC
339 ET (RO, RO, RO); GZ - 9
340 ADD (R5, R1, R5); DZ - 19                *C Test de fin d'exécution d'une
                                                étape
341 ADD ('0117', RO, R15)                    *C Fin d'exécution d'une étape
342 ET (RO, RO, RO); SYNC(A')                *C Libération de OP
343 ET (RO, RO, RO); SYNC(B)                 *C Attente des paramètres pour
                                                l'étape suivante
344 DOP (  , R6, R6); POSRCC
345 ET (RO, RO, RO); DZ - 23                 *C Test de fin de traitement
346 BRANCH ('XXXX')                          *C Fin de traitement. Branchement
                                                à l'adresse 'XXXX'.
```

a) Un microprogramme principal

Figure VI.11 : MICROPROGRAMMES POUR EXT-TFR

N° de m.i.

```
497 CONJ ('FGFF', RERR, RO)          *C Point d'entrée du m.p. gesti
                                        débordements
498 DOP (  , RO, RO); GLS
499 DOP (  , RO, RO); POSRCC
500 ST (RO, RO, RCOMPT); EZ + 2
501 BRANCH ('0012')                  *C L'erreur détectée n'est pas u
                                        débordement OP. Retour à la
                                        gestion par défaut
502 NORM                             *C Passage au fonctionnement nor
503 DOP (  , RCONTX, RCONTX)         *C Décalage de la file des vecte
                                        d'état
504 ET ('01FF', RCONTX, EXTENSION)
505 ET ('FE00', RCONTX, RO)
506 ADD ('0149', EXTENSION, EXTENSION);DAE
507 OU ('0004', EXTENSION, EXTENSION)
508 OU (RO, EXTENSION, RCONTX)       *C Ecriture dans FVE du nouveau r
                                        d'état OP
509 DOP (  , RCONTX, RCONTX)         *C Décalage de la file FVE
510 DOP (  , RCOMPT, RO)
511 RETOUR                ; INC-1
```

b) Microprogramme de gestion de débordements

```
4 ET ('0040', PRIOR, RO)             *C Point d'entrée de la tâche EXT
                                        pour la gestion des alarmes
                                        internes
5 DOP (  , RO, RO); POSRCC
6 ADD ('01C0, PRIOR, RO)
7 ET (RO, RO, RO); INC + 9
16 ET (RO, RO, RO); EZ + 2
17 BRANCH (RO)
18 Gestion par défaut des alarmes internes
```

c) Microprogramme de gestion des alarmes internes

Figure VI.11. (Suite) : MICROPROGRAMMES POUR EXT - TFR

```
256 ST (RO, FLEC, R8); INC + 2          *C Lecture - COSINUS
257 ADD (RO, FLEC, R8)                  *C Lecture COSINUS
258 ST (RO, FLEC, R9); INC + 2          *C Lecture - SINUS. Pour TFR-1
                                         changer ST pour ADD
259 DOP (  , FLEC, R11); INC + 3
260 DOP (  , FLEC, R10); INC - 1
261 DOP (  , FLEC, R10); INC - 2
262 DOP (  , FLEC, R12)
263 ST (R10, R12, R13); POSRCC          *C Si débordement, diviser par
                                         deux le bloc de données; (1)
264 MULT (R13, R8)
265 ADD (R10, R12, FECR); POSRCC        *C /  /  /  /  (2)
266 DOP (  , PFORT, R14)
267 MULT (R13, R9)
268 DOP (  , FLEC, R10); POSRCC
269 DOP (  , PFORT, R13)
270 ST (R11, R10, R12)                  *C /  /  /  /  (3)
271 MULT (R12, R9)
272 ADD (R11, R10, FECR)                *C /  /  /  /  (4)
273 ST (R14, PFORT, R11)
274 MULT (R12, R8)
275 ADD (R11, R11, FECR)                *C /  /  /  /  (5)
276 ADD (R13, PFORT, R11)
277 ADD (R11, R11, FECR)                *C /  /  /  /  / (6)
278 BRANCH (R15); SYNC(A)               *C Point d'entrée du N.P. OP.
                                         Test branchement
279 ADD ('0100', RO, R15)               *C Fin exécution d'une étape;
                                         calcul paramètres suivants
280 CONJ (RO, R6, EXTENSION); DAE
281 ST (RO, R3, R3); GLS
282 ADD (R1, EXTENSION, R5)
283 CONJ(RO, EXTENSION,R6); SYNC(B', C)  *C Attente RAN; Libération EXT
                                         et RAN
284 DOP (  , R6, R6); POSRCC
285 ADD (R1, EXTENSION, R7); DZ - 7      *C Test fin de traitement
286 BRANCH ('YYYY')                     *C Fin de traitement.
                                         Branchement à l'adresse 'YYYY'
```

a) Microprogramme principal

Figure VI.12 : MICROPROGRAMMES POUR OP-TFR

N° de m.i.

```
287 DOP (  , FLEC, EXTENSION); INC + 2          *C Fausse lecture cosinus
288 DOP (  , FLEC, EXTENSION)                   *C Fausse lecture cosinus
289 DOP (  , FLEC, EXTENSION); INC + 3          *C Fausse lecture sinus
290 DOP (  , FLEC, EXTENSION); INC - 20         *C Fin traitement débordements
                                                   (3), (4), (5) et (6)
291 ET (RO, RO, RO); INC + 1
292 DOP (  , FLEC, EXTENSION); DZRCOMPT + 2
293 ET (RO, RO, RO); INC - 34                   *C Fin traitement débordement
                                                   (1) et (2)
294 EXEC (RO, EXTENSION, FECR); DZRCOMPT + 3
295 DOP (  , FLEC, EXTENSION)
296 DOP (  , FLEC, EXTENSION); INC - 6
297 EXEC (RO, FLEC, FECR)
298 EXEC (RO, FLEC, FECR); DZRCOMPT + 1
299 EXEC (RO, FLEC, FECR); DZRCOMPT + 16
300 DOP (  , R10, FECR); DAE                     *C Début traitement déborde-
                                                   ments (1) et (3)
301 DOP (  , R11, FECR); DAE                    *C Début traitement
                                                   débordements (2) et (4)
302 DOP (  , R12, FECR)                         *C Début traitement
                                                   débordement (5)
303 ADD ('0001', ML, ML)                        *C Début traitement
                                                   débordement (6)
304 EXEC ('FFFA', R1, RCOMPT)
305 ADD (R12, R10, R11)                         *C Reconstitution de d/2
306 EXEC (R10, RO, R10); DEB - 7
307 EXEC (R11, RO, R11)
308 EXEC (R13, RO, R13)
309 EXEC (R14, RO, R14); INC + 6
310 ADD ('0100', RO, R15)
311 ST (RO, R3, R3)
312 CONJ (RO, R6, EXTENSION)
313 ADD (R1, EXTENSION, R5); SYNC(B', C)
314 ADD (R1, EXTENSION, R7)
315 DOP (  , R15, R12); SYNC(A)
316 BRANCH (R12); INC + 31                       *C Test branchement
```

b) Microprogramme de cadrage

Figure VI.12 (suite) : MICROPROGRAMMES POUR OP - TFR

N° de m.i.

```
347 ST (R4, R7, RADECR)                    *C Point d'entrée du
                                              microprogramme RAN
348 ADD ('FFFF', R7, R7)
349 ST (R4, R7, RADECR)
350 ST (R7, R6, R7)
351 ST (R4, R7, RADECR)
352 ADD ('FFFF', R7, R7)
353 ST (R4, R7, RADECR)
354 ST (R7, R6, R7); POSRCC
355 ET (RO, RO, RO); GZ - 8
356 ADD (R7, R1, R7); DZ - 9             *C Test de fin d'exécution d'une
                                              étape
357 ET (RO, RO, RO); SYNC(C')             *C Fin d'exécution d'une étape
358 ET (RO, RO, RO); ISYNC(C')            *C Attente des paramètres pour
                                              l'étape suivante
359 DOP (  , R6, R6); POSRCC
360 ET (RO, RO, RO); GZ - 3               *C Test de fin de traitement
361 BRANCH ('ZZZZ')                       *C Fin de traitement; branchement à
                                              l'adresse ZZZZ
```

Figure VI.13 : MICROPROGRAMME POUR RAN - TFR

- C O N C L U S I O N -

Dans le cadre de ce travail nous avons montré que l'efficacité de la technique "pipe-line" est grandement améliorée quand elle est appliquée à la définition de systèmes multiprocesseurs plutôt que monoprocesseurs. Dans les monoprocesseurs "pipe-line", du fait du problème de la dépendance, l'accroissement maximal théorique de la puissance de calcul est très optimis par rapport à celui obtenu dans la pratique. Dans les multiprocesseurs "pipe-line" décrits dans la première partie de cette thèse, le problème de dépendance étant éliminé, le gain réel en puissance de calcul peut atteindre le maximum théorique.

Nous avons étudié la mise en oeuvre des outils de synchronisation au sein de ces structures. Ces outils dérivés plus ou moins directement des sémaphores et primitives P et V, donnent lieu à des systèmes de synchronisat très souples et d'une compréhension relativement facile pour l'utilisateur.

Le réseau d'opérateurs adapté au traitement du signal décrit dans deuxième partie de ce mémoire a été développé à partir d'un multi-microproce seur en barillet "pipe-line". A l'état actuel de l'étude, ce réseau est pratiquement défini : les circuits logiques et leur découpage sont établis, leur fonctionnement logique a été testé par une simulation en langage CASSANDRE. Nous pouvons dire que tout est prêt pour une réalisation technologique, réalisation que nous souhaitons vivement car, comme le montre les quelques exemples d'application exposés au sixième chapitre, les solutio apportées par cette structure sont prometteuses.

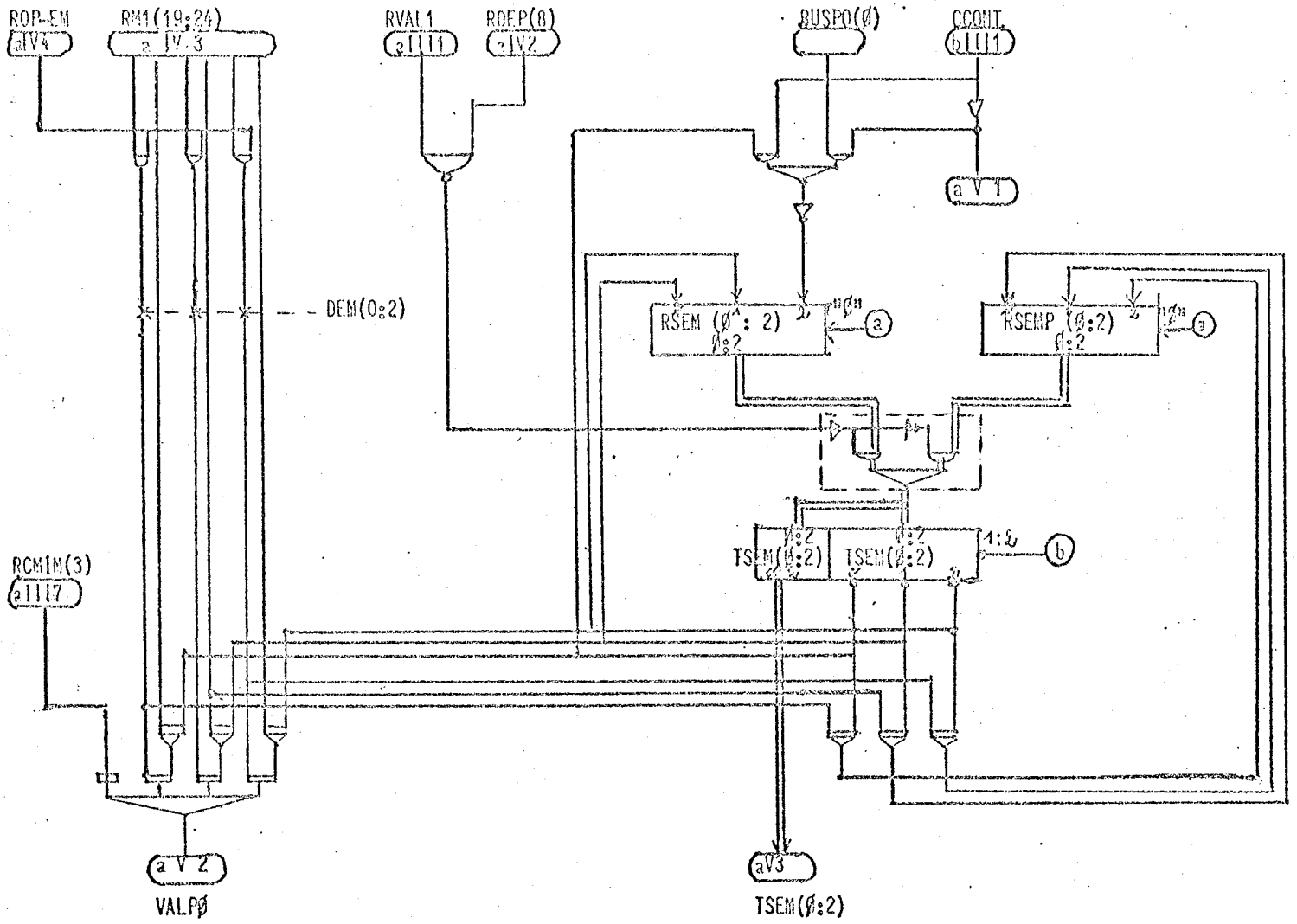
- A P P E N D I C E -

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

Nous avons établi un programme de simulation en langage CASSANDRE [16] décrivant le fonctionnement logique du réseau d'opérateurs. Ce programme respecte assez fidélement toutes les fonctions logiques et tou les éléments de mémorisation. Il est constitué de 7 "unités" principales : MIMEM, REGE, ARIT, ETAT, MULT, MEM et PRIN, les six premières décrivent chacune des six parties exposées au paragraphe V.2., la dernière pour établ les connexions entre ces parties.

Pour illustrer la correspondance entre ces unités et les circuits du réseau, considérons à titre d'exemple les circuits nécessaires à la réalisation des trois couples de sémaphores parallèles A-A', B-B' et C-C' (c.f. IV.4) représentés figure A.1.a et les instructions CASSANDRE les décrivant (figure A.1.b).

Les figures A.2 à A.8 représentent les programmes des sept "unité mentionnées plus haut.



a) Diagramme logique

```
VALPO : = RCMIM(3).-/+(DEM(0:2).(RM1(10) & RM1(12) & RM1(14)) & TSEM(0:2));
DEM(0:2) : = 'S1'ROPSEM'ALORS'RM1(9)&RM1(11)&RM1(13)'SINON'000;
<H(0)> RSEM(0:2)<=TSEM(1:2)&('S1'CCONT'ALORS'BUSP(0)'SINON'TSEM(0)),
       RSEMP(0:2) <= *R|1|(DEM(0:2) # TSEM(0:2));

< H(1)>  TSEM(0:2) <= 'S1'RDEP(8). RVAL1'ALORS'RSEMP(0:2)'SINON'RSEM(0:2);
```

b) Description en langage CASSANDRE

Figure A.1. : DIAGRAMME LOGIQUE DES SEMAPHORES PARALLELES ET DESCRIPTION EN LANGAGE CASSANDRE

```
                    NI(0:29),CO(0:7),CONTEX(0:15),VAL1,ERRMM);
'HORLOGEMERE'H,HRAZ;
'REGISTRE'RVAL1,RCSAUT,RCCONT,RPO(2:15),RPOINT(0:8,0:2),RDEP(2:8),
          RDO(0:15),RSEMP(0:2),RSEM(0:2),RCMIM,RTMIM,MIM(0:29,0:511),
          RADMIN(0:8),RMI(0:25),TSEM(0:2),ROPSEM,RCMIM(1:3),RT(1:2),
          RMIN(0:29),RCRMIN,TUAL9(0:9),DECODE(0:7,0:15);
'SIGNAL'DEM(0:2);
'EXTERNE'UAL9((0:8),(0:8),(0:3),,;(0:9));
MI:=MIM(,$(RADMIN));
CO:=DECODE(,$(-RMI(0)&RMI(1:3)));
CONTEX:=TSEM&0000&RPOINT(,0);
VAL1:=RVAL1;
ERRMM:=TUAL9(0);
DEM:='SI'ROPSEM'ALORS'RMI(19)&RMI(21)&RMI(23)'SINON'000;
UAL9('SI'RCSAUT.RVAL1+RCCONT'ALORS'RPO(7:15)'SINON'RPOINT(,1),
     RDEP(2)&RDEP(2)&RDEP,(TEST.RVAL1)&00&(TEST.RVAL1),0,
     -(RVAL1.-RCSAUT.-TEST));
<H(0)>RVAL1<=VALE.-RCMIM(3).-/+(DEM.(RMI(20)&RMI(22)&RMI(24))#TSEM),
      RCSAUT<=CO(0),
      RCCONT<=CCONT,
      RPO<=BUSPO(2:15),
      RPOINT(,1)<=RPOINT(,0),
      RDEP<=RMI(19:25),
      RDO(0:3)<=BUSDO(0:3),
      'SI'RT(2)'ALORS'RDO(4:15)<=BUSDO(4:15),
      RSEMP<=*R|1|(DEM#TSEM),
      RSEM<=TSEM(1:2)&('SI'CCONT'ALORS'BUSPO(0)'SINON'TSEM(0)),
      RCMIM<=CO(7).RT(2),
      RTMIM<=-ADRTO2;
<H(1)>'SI'RCMIM(2).RADMIN(0)'ALORS'MIM(,$(RADMIN))<=RMIN,
      'SI'-RCMIM(2)'ALORS'RADMIN<=('SI'RCMIM(1)'ALORS'1&RDO(8:15)
                                   'SINON'TUAL9(1:9)),
      RMI(0:3)<=MI(0:3),
      RMI(19:25)<=MI(19:25),
      TSEM<='SI'RDEP(8).RVAL1'ALORS'RSEMP'SINON'RSEM,
      RPOINT(,0)<=RPOINT(,2),
      ROPSEM<=-/+(MI(14:17)&-MI(18)),
      RCMIM(3)<=RCMIM(2)+RCMIM(3).RT(2),
      RT<=(-/+RT)&RT(1);
<H(3)>RCMIM(1:2)<=(RCMIM.-RTMIM.RVAL1)&(RCMIM(1).-RCMIM(2)),
      'SI'RCMIM.RTMIM.RVAL1.-RCRMIM'ALORS'RMIM<=-RPO&-RDO,
      RPOINT(,2)<=TUAL9(1:9),
      RCRMIM<=RCMIM.RTMIM.RVAL1.-RCRMIM;
<H(4)>TUAL9<=UAL9(,,,,;*);
'POUR'I=0'A'7'DEBUT'<HRAZ>DECODE(I,I)<=1;'FIN';
<HRAZ>RVAL1<=0,
      RT<=00,
      TUAL9(0)<=0,
      RCMIM(1)<=0;

'UNITE'UAL9(PO(0:8),DO(0:8),S(0:3),N,CO;F(0:9));
'SIGNAL'A(0:9),B(0:9),X(0:9),Y(0:9),REP(0:10),Z(0:9);
A:=0&PO;
B:=DO(0)&DO;
X:=-(A+('SI'S(0)'ALORS'B'SINON'Z)+('SI'S(1)'ALORS'-B'SINON'Z));
Y:=-(A.(('SI'S(2)'ALORS'-B'SINON'Z)+('SI'S(3)'ALORS'B'SINON'Z)));
REP:=*D|1|(0&X+0&Y.REP)&CO;
F:='SI'N'ALORS'X+-Y'SINON'(-X.Y)=REP(1:10);
```

FIGURE A.2. UNITE MIMEM: MEMOIRE PRINCIPALE ET CIRCUITS POUR SA GESTION. SEMAPHORES PARRALLELES.

```
'UNITE'REGE(H(0:4),HRAZ,CCONT,TO(0:15),MI(0:29),VAL1,CO1;
            BUSPO(0:15),BUSDO(0:15),ADRDO(0:4),ADRTO(0:4),ZERO);
'HORLOGE'HERE'H,HRAZ;
'REGISTRE'TCOMPT(8:15),TTO(0:15),RMI(0:29),RG(0:15,0:15),RCOMPT(0:15),
          RT(0:2),RATO(0:4,0:1),MEM(0:15,0:255),RPU(0:1),RECCOM(0:1),
          RECMEM(0:1);
'SIGNAL'Z(0:15),REP(0:15),SECMEM;
BUSPO:='SI'CCONT'ALORS'Z
       'SINON'('SI'RMI(14)'ALORS'RMI(15)&RMI(15:29)
                'SINON'RG(,$(RMI(20:29))));
BUSDO:='SI'-RMI(9)'ALORS'RG(,$(RMI(10:13)))
       'SINON'('SI'-RMI(10).-RMI(11).RMI(12)
                'ALORS'('SI'RMI(13)'ALORS'MEM(,$(TCOMPT))'SINON'RCOMPT)
                'SINON'Z);
ADRDO:=(RMI(9).-RMI(10))&(RMI(9).RMI(10))&RMI(11:13);
ADRTO:=(RMI(4).-RMI(5))&(RMI(4).RMI(5))&RMI(6:8);
ZERO:=/.RCOMPT;
REP:=*D|1|(RCOMPT.REP)&RPU(1);
SECMEM:=RMI(4).-RMI(5).-RMI(6).RMI(7).RMI(8).RT(1);
<H(0)>'SI'RT(0)'ALORS'TCOMPT<=RCOMPT(8:15),
       'SI'RT(2)'ALORS'TTO<=TO;
<H(1)>RMI(0)<=MI(0),
      RMI(4:29)<=MI(4:29),
       'SI'RATO(0,1)'ALORS'RG(,$(RATO(1:4,1)))<=TO,
       RCOMPT<='SI'RECCOM(1)'ALORS'TO'SINON'RCOMPT#REP,
       RT<=*R|-1|RT;
<H(2)>RATO(2:4,1)<=RATO(2:4,0);
<H(3)>'SI'RECMEM(1)'ALORS'MEM(,$(TCOMPT))<=TTO,
       RPU(1)<=RPU(0).VAL1,
       RATO(0:1,1)<=(RATO(0,0).VAL1)&RATO(1,0),
       RECCOM(1)<=RECCOM(0).VAL1,
       RECMEM(1)<=RECMEM(0).VAL1;
<H(4)>RATO(,0)<=((-RMI(0)+CO1).-RMI(4))&RMI(5:8),
       RPU(0)<=-SECMEM.-RMI(14)./.RMI(15:18).RT(1),
       RECCOM(0)<=RMI(4).-RMI(5).-RMI(6).RMI(7).-RMI(8).RT(1),
       RECMEM(0)<=SECMEM;
<HRAZ>RMI(4)<=0,
       RMI(9)<=0,
       RT<=100;
```

FIGURE A.3. UNITE REGE: REGISTRES GENERAUX. MEMOIRE LOCALE.

```
'UNITE'ARIT(H(0:4),HRAZ,ADRDO(0:4),CO1,ZERO,BUSPO(0:15),BUSDOE(0:15),
            ADRTO(0:4),CCONT,MI(0:23),VAL1;
            BUSDOS(0:15),TO(0:15),TEST,CONTEX(0:15),ERRDEB);
'HORLOGEMERE'H,HRAZ;
'REGISTRE'RTEST,RPO(0:15),RDO(0:15),RS(0:3),RM,RC16,RCREXT,RCOPT(0:4),
          ROPARI,RCRCOD(0:1),RCC(0:3,0:3),REXTEN(0:15),RCODE(0:15),
          RMI(0:23),RT(0:2),RERRDE,RCOEXT(0:1),TSELEC,TSUAL0,TTO(0:15),
          RPARM(0:4),TGAUCH,TDROIT,RMSQD,RPCC,DECODE(0:15,0:15);
'SIGNAL'Z(0:15),OPSIM,X(0:1),W(0:2),DEBOR,SUAL(0:16),C0,
        SZERO,XUAL(0:15);
'EXTERNE'UAL17((0:16),(0:16),(0:3),,;(0:16),,);
BUSDOS:='SI'ADRDO(0).ADRDO(2).-ADRDO(3)
        'ALORS'('SI'ADRDO(4)'ALORS'REXTEN'SINON'RCODE)'SINON'Z;
TO:=('SI'TSELEC'ALORS'-TSUAL0'SINON'TTO(0))&TTO(1:15);
TEST:=RTEST;
CONTEX(3:6):=RCC(,0);
ERRDEB:=RERRDE;
OPSIM:=/.-RMI(14:18);
X:=(OPSIM+-CO1)&(CO1.(OPSIM+REXTEN(15)));
W:='SI'X(1)'ALORS'RCODE(13:15)
   'SINON'('SI'X(0)'ALORS'RMI(1:3)'SINON'RCODE(5:7));
DEBOR:=(RPARM(1)+RPARM(2)).-RPARM(0);
UAL17(-RPO(0)&-RPO,-RDO(0)&-RDO,RS,RM,RC16;SUAL,C0,SZERO);
'POUR'J=0'A'15'DEBUT'XUAL(J):=-SUAL(16-J);'FIN';
<H(0)>RTEST<='SI'RMI(14)'ALORS'0
            'SINON'/+(DECODE(,$(RMI(15:18))).(001&RCC(1,0)&-RCC(1,0)&
                                              /.-RCC(0:1,0)&-RCC(0,0)&
                                              /+RCC(0:1,0)&RCC(0,0)&
                                              RCC(2:3,0)&1111&
                                              -(RT(1).ZERO))),
      RPO<=BUSPO,
      RDO<=BUSDOE,
      RS<=-(W(0)+W(1).W(2))&(W(0)+W(1))&W(0)&(-W(0)+W(1).W(2)),
      RM<=W(1),
      RC16<=W(0).W(2)+-W(2).RCC(3,0),
      RCREXT<=ADRTO(0).ADRTO(2).-ADRTO(3).ADRTO(4),
      RCOPT<='SI'-(OPSIM+CO1)'ALORS'00000
           'SINON'('SI'X(0)'ALORS'RMI(19:23)
                  'SINON'('SI'X(1)'ALORS'RCODE(8:12)
                         'SINON'RCODE(0:4))),
      ROPARI<=CO1+-RMI(0),
      RCRCOD(0)<=ADRTO(0).ADRTO(2).-ADRTO(3).-ADRTO(4),
      RCC(,1)<='SI'CCONT'ALORS'BUSPO(3:6)'SINON'RCC(,0);
<H(1)>REXTEN<='SI'RCOEXT(0)'ALORS'('SI'RCOEXT(1)'ALORS'TO
                                  'SINON'REXTEN(1:15)&TDROIT)
             'SINON'('SI'RCOEXT(1)'ALORS'TGAUCH&REXTEN(0:14)
                    'SINON'REXTEN),
      'SI'RCRCOD(1)'ALORS'RCODE<=TO,
      RCC(,0)<=RCC(,3),
      RMI(0:3)<=MI(0:3),
      RMI(14:23)<=MI(14:23),
      RT<=*R|-1|RT;
<H(2)>RERRDE<=DEBOR.-RMSQD;
<H(3)>RCOEXT<=((RCREXT+RCOPT(2).-RCOPT(4)).ROPARI.VAL1)&
              ((RCREXT+RCOPT(2).RCOPT(4)).ROPARI.VAL1),
      RCC(,3)<='SI'RPCC'ALORS'-RPARM(2)&RPARM(3)&DEBOR&
                              (-RPARM(0).RPARM(4))
                'SINON'RCC(,2);
<H(4)>TSELEC<=/.RCOPT(2:4),
      TSUAL0<=SUAL(0),
      TTO(0)<=-(RCOPT(2).RCOPT(4)).
              ('SI'RCOPT(2)+/.RCOPT(3:4)
               'ALORS'('SI'RCOPT(4)'ALORS'REXTEN(15)'SINON'-SUAL(2))
               'SINON'('SI'RCOPT(4)'ALORS'XUAL(0)'SINON'-SUAL(1))),
      TTO(1:14)<='SI'/+RCOPT(2:3)
                 'ALORS'('SI'RCOPT(4)'ALORS'-SUAL(1:14)
                        'SINON'-SUAL(3:16))
                'SINON'('SI'RCOPT(4)'ALORS'XUAL(1:14)
                       'SINON'-SUAL(2:15)),
      TTO(15)<=(RCOPT(2)+-RCOPT(3)+RCOPT(4)).
              ('SI'RCOPT(2)+/.RCOPT(3:4)
               'ALORS'('SI'RCOPT(4)'ALORS'-SUAL(15)'SINON'REXTEN(0))
               'SINON'('SI'RCOPT(4)'ALORS'XUAL(15)'SINON'-SUAL(16))),
      RPARM<=RM&SUAL(0:1)&SZERO&C0,
      TGAUCH<=-SUAL(16),
      TDROIT<=-SUAL(1).RCOPT(3),
      RCRCOD(1)<=RCRCOD(0).ROPARI.VAL1,
      RMSQD<=RCOPT(0)+-(ROPARI.VAL1),
      RPCC<=RCOPT(1).ROPARI.VAL1,
      RCC(,2)<=RCC(,1);
'POUR'I=0'A'15'DEBUT'<HRAZ>DECODE(I,I)<=1;'FIN';
<HRAZ>RT<=100,
      RERRDE<=0,
      RMSQD<=1;

'UNITE'UAL17(PO(0:16),DO(0:16),S(0:3),M,C16;F(0:16),C0,REZERO);
'SIGNAL'X(0:16),Y(0:16),REP(0:17),Z(0:16);
X:=-(PO+('SI'S(0)'ALORS'DO'SINON'Z)+('SI'S(1)'ALORS'-DO'SINON'Z));
Y:=-(PO.(('SI'S(2)'ALORS'-DO'SINON'Z)+('SI'S(3)'ALORS'DO'SINON'Z)));
REP:=*D|1|(0&X+0&Y.REP)&C16;
F:='SI'M'ALORS'X+-Y'SINON'(-X.Y)=REP(1:17);
C0:=REP(1);
REZERO:=/.F;
```

FIGURE A.4. UNITE ARIT: UNITE ARITHMETIQUE ET LOGIQUE.REGISTRES ASSOCIES.CALCUL DES TESTS.

```
'UNITE'ETAT(H(0:4),HRAZ,APPELS(2:8),ADRDO(0:4),CONTEX(0:15),TO(0:15),
            ERRDEB,ERRF,ERRAD,CO(0:7),ADRTO(0:4),ERRHH,
            VAL1;
            BUSPO(0:15),BUSDO(0:15),VALS,CCONT,REPONS(2:8),HALTE);
'HORLOGEHERE'H,HRAZ;
'REGISTRE'REXC(0:15),RREPON,RHALTE,RCONTX(0:15,0:15),RTRP,RRZERR(0:1),
          ROPCON(0:2),RTCON(0:1,0:1),RECREX(0:2),RECOTX(0:1),
          RCONTX(0:15,0:3),REFF,RCCONT,RM3,RT(0:2),RERR(0:15),RERRTO,
          RPCON(0:3),RERRHH,RERRCO,
          RTA(0:2),RENTRE,RETOUR,RTR,RHTPRI,RRZ,RRU;
'SIGNAL'AX(0:8),TPRIOR(0:7),CODEXT(0:2),Z(0:15),SP1,X(0:15),ERREUR,
        ALINEF,APEXEF,SM3,XRZERR,M3(0:3),REP(0:3),VIDE,ERRCO;
AX:=0&*0|-1|(6&(APPELS.REXC(2:8))&0+AX);
TPRIOR:='SI'RHTPRI'ALORS'AX(0:7)#AX(1:8)'SINON'TPRIOR;
CODEXT:='SI'-REFF'ALORS'-(TPRIOR(1)+TPRIOR(2)+TPRIOR(3))&
                        -(TPRIOR(1)+TPRIOR(4)+TPRIOR(5))&
                        -(TPRIOR(2)+TPRIOR(4)+TPRIOR(6))'SINON'000;
BUSPO:='SI'-RCCONT'ALORS'Z
       'SINON'('SI'-RM3'ALORS'RCONTX(,3)
                'SINON'0000000&/+CODEXT(0:1)&00&CODEXT&RT(0)&RT(2)&0);
BUSDO:='SI'ADRDO(0).-ADRDO(2).(-ADRDO(3).-ADRDO(4)+RT(0))
       'ALORS'('SI'ADRDO(3)'ALORS'('SI'ADRDO(4)'ALORS'REXC
                                    'SINON'RCONTX(,3))
               'SINON'('SI'ADRDO(4)'ALORS'RERR'SINON'CONTEX))
       'SINON'Z;
VALS:=-RCCONT.-REFF.-RERRTO;
CCONT:=RCCONT;
REPONS:='SI'RREPON'ALORS'TPRIOR(1:7)'SINON'0000000;
HALTE:=RHALTE;
SP1:=RCCONT+REFF;
X:='SI'SP1'ALORS'('SI'RM3'ALORS'CONTEX'SINON'HCONTX(,$(RPCON)))
   'SINON'TO;
ERREUR:=RERRHH+RERRCO+ERRDEB+ERRF+ERRAD;
ALINEF:=RERRTO.-REFF.-ROPCON(2).RTA(2);
APEXEF:=/+TPRIOR.-ROPCON(2).-RECREX(2)./.REXC(0:1).-RERR(0).RPCON(0).
        -REFF.RTA(2);
SM3:=RENTRE+ALINEF+APEXEF+RM3.RTA(2);
XRZERR:=RRZERR(1).-RERRTO;
M3:=SM3&CM3&&0&SM3;
REP:=*0|1|(RPCON.M3+RPCON.REP+M3.REP)&SP1;
VIDE:=RPCON(0).RPCON(2);
ERRCO:=ROPCON(1).('SI'/.RTCON(,1)'ALORS'-RPCON(0)'SINON'VIDE);
<H(0)>REXC(0)<=-RRZ.(RRU+('SI'RECREX(1)'ALORS'X(0)'SINON'REXC(0))),
      'SI'RECREX(1)'ALORS'REXC(1:15)<=X(1:15),
      RREPON<='SI'RREPON'ALORS'-RT(2)'SINON'RCCONT.-REFF.-RERRTO.RT(2),
      RHALTE<=RHALTE+ERREUR.(RERR(0).-RERRTO+RTRP),
      'SI'RM3'ALORS'HCONTX(,$(RPCON))<=RCONTX(,3),
      RTRP<='SI'RTRP'ALORS'-ROPCON(2)'SINON'-RPCON(0).-RPCON(1).RERRTO,
      RRZERR(0)<=CO(6).RT(0),
      ROPCON(0)<=CO(7).RT(0),
      RTCON(,0)<=ADRTO(2:3),
      RECREX(0)<=ADRTO(0).-ADRTO(2).ADRTO(3).ADRTO(4).RT(0),
      RECOTX(0)<=ADRTO(0).-ADRTO(2).ADRTO(3).-ADRTO(4).RT(0),
      'SI'RECOTX(1)'ALORS'RCONTX(,0)<=X;
<H(1)>REFF<='SI'REFF'ALORS'-RTA(2)'SINON'ROPCON(2)+ALINEF,
      RCCONT<='SI'RCCONT'ALORS'-(-ALINEF.RTA(2))
              'SINON'RETOUR+ALINEF+APEXEF,
      RM3<='SI'RM3'ALORS'-RTA(2)'SINON'SM3,
      'SI'RECOTX(1)'ALORS'RCONTX(,3)<=RCONTX(,2),
      RT(0)<=RT(2),
      RT(2)<=-RT(0).-RT(2),
      RERR(0)<='SI'RERR(0)'ALORS'-XRZERR'SINON'ERREUR,
      RERR(1)<='SI'RERR(1)'ALORS'-XRZERR'SINON'ERREUR.RERR(0),
      RERR(2)<='SI'RERR(2)'ALORS'-XRZERR'SINON'ERREUR.RERR(1),
      'SI'-RERR(0)'ALORS'RERR(3:15)<=RTA(2)&RTA(0)&RERRHH&RERRCO&ERRDE
                                      ERRF&ERRAD&Z(10:15),
      RERRTO<='SI'RERRTO'ALORS'-(RCCONT.RM3.REFF)'SINON'ERREUR,
      RPCON<=(RPCON#M3)#REP;
<H(2)>RERRHH<=ERRHH,
      RERRCO<=ERRCO,
      ROPCON(2)<=ROPCON(1).-ERRCO,
      RTA(0)<=RTA(2),
      RTA(2)<=-RTA(0).-RTA(2),
      RECREX(2)<=RECREX(1),
      RENTRE<=ROPCON(1).-ERRCO./.RTCON(,1),
      RETOUR<=ROPCON(1).-ERRCO.-RTCON(0,1),
      'SI'RECOTX(1)'ALORS'RCONTX(,2)<=RCONTX(,1),
      RTR<=RTCON(1,1);
<H(3)>RRZERR(1)<=RRZERR(0).VAL1,
      ROPCON(1)<=ROPCON(0).VAL1,
      RTCON(,1)<=RTCON(,0),
      RECREX(1)<=RECREX(0).VAL1,
      RECOTX(1)<=RECOTX(0).VAL1+SP1,
      'SI'RECOTX(1)'ALORS'RCONTX(,1)<=RCONTX(,0);
<H(4)>RHTPRI<=(-RCCONT+REFF).-RREPON.RT(0),
      RRZ<=RCCONT.-REFF.-RERRTO.RT(2),
      RRU<=RTR.RETOUR;
<HRAZ>REXC<=Z,
      RREPON<=0,
      RHALTE<=0,
      RTRP<=0,
      REFF<=1,
      RCCONT<=1,
      RM3<=1,
      RT<=100,
      RERR(0:2)<=000,
      RERRTO<=0,
      RPCON<=1111,
      RERRHH<=0,
      RERRCO<=0,
      RTA<=100,
      RETOUR<=0,
      ROPCON(1)<=0,
      RECREX(1)<=0,
      RRZ<=1,
      RRU<=0;
```

FIGURE A.5. UNITE ETAT: GESTION DES APPELS EXTERNES ET DES ALARMES INTERNES. FILE DE SAUVEGARDE AUTOMATIQUE DES VECTEURS D'ETAT.

```
'UNITE'MULT(H(0:4),HRAZ,ADRDO(0:4),CO7,VAL1,BUSPO(0:15),BUSDOE(0:15),
            ADRTO2;
            BUSDOS(0:15),VALS);
'HORLOGEMERE'H,HRAZ;
'REGISTRE'RPO(0:15),RDO(0:15),RDEMOP,RCODE,RT(0:2),RPFORT(0:15),
          RPFAIB(0:15),RMPX,RK(0:15),RAUX,ROC,RTA1;
'SIGNAL'Z(0:15),DS,SMUL(0:23),SROC;
'EXTERNE'MUL((0:8),(0:15),(0:15);(0:23));
DS:=RT(1)./.(ADRDO(1:3)=101);
BUSDOS:='SI'DS'ALORS'('SI'ADRDO(4)'ALORS'RPFORT'SINON'RPFAIB)'SINON'Z;
VALS:=-(RAUX.(DS+RT(1).CO7));
MUL('SI'RMPX'ALORS'RPO(0:8)'SINON'RPO(8:15)&0,RDO,RK;SMUL);
SROC:=RAUX+('SI'ROC'ALORS'-RT(1)'SINON'RDEMOP.RT(2).VAL1);
<H(0)>'SI'RT(1).-RAUX'ALORS'(RPO<=BUSPO,
                             RDO<=BUSDOE,
                             RDEMOP<=CO7,
                             RCODE<=ADRTO2);
<H(1)>RT<=*R|-1|RT,
      'SI'RTA1.ROC
      'ALORS'(RPFORT<='SI'RCODE'ALORS'SMUL(0:15)'SINON'RPO,
              RPFAIB<='SI'RCODE'ALORS'SMUL(16:23)&RPFAIB(0:7)'SINON'RDO);
<H(3)>'SI'RTA1.ROC'ALORS'(RMPX<=1,
                          RK<=SMUL(0:15));
<H(4)>RAUX<=RCODE.('SI'RAUX'ALORS'-RT(2)'SINON'RDEMOP.RT(2).VAL1),
      'SI'-SROC'ALORS'(RMPX<=0,
                       RK<=Z),
      ROC<=SROC,
      RTA1<=RT(0);
<HRAZ>RCODE<=0,
      RT<=100,
      RAUX<=0,
      ROC<=0;

'UNITE'MUL(P(0:8),DO(0:15),K(0:15);S(0:23));
'SIGNAL'D(0:16),X(0:15,0:4);
'EXTERNE'M((0:2),(0:16),(0:15);(0:15),(0:1));
D:=DO(0)&DO;
X(,0):=K;
'POUR'I=0'A'3'DEBUT'
     M|I|(P(6-2.I:8-2.I),D,X(,I);X(,I+1),S(22-2.I:23-2.I));'FIN';
S(0:15):=X(,4);

'UNITE'M(Y(0:2),X(0:16),K(0:15);S(0:15),T(0:1));
'SIGNAL'P(0:17),XA(0:17),D(0:17),R(0:17),SX(0:17),Z(0:17);
P:=K(0)&K(0)&K;
XA:='SI'Y(1)#Y(2)'ALORS'X(0)&X
    'SINON'('SI'-Y(0).Y(1).Y(2)+Y(0).-Y(1).-Y(2)'ALORS'X&0'SINON'Z);
D:='SI'Y(0)'ALORS'-XA'SINON'XA;
R:=*D|1|(P.D+P.R+D.R)&Y(0);
SX:=(P#D)#R;
S:=SX(0:15);
T:=SX(16:17);
```

FIGURE A.6. UNITE MULT: MULTIPLIEUR RAPIDE.

```
'UNITE'MEM(H(0:4),HRAZ,ADRTO(0:4),ADRLO(0:4),FADM,ADADM(0:15),
            DOADME(0:15),VAL1,TO(0:15),CO5,DEMADM;
            DOP(0:15),VALP,ERRAD,ERRF,DODANS(0:15),LECADM,ERRADM,
            DUSADM,DEBADM,FINADM);
'HORLOGEMERE'H,HRAZ;
'REGISTRE'ROPER,MEMT(0:33,0:15),RAD(0:15,1:2),ROPLEC,ROPECR,TERRAD(1:2),
           ROPF(0:3),REXLEC,RRAECR,RT(0:2),RCOMPT(0:4),ROCBUS(1:4),
           TFINOP(1:3),RINTP(1:2),RINMEM,RDEADM,ROCMEM(1:4),RRZ,RERRAD,
           RERRF(1:2),RM4,RJ(1:4),RUOC(1:4),RDAC,RCHAR(1:2),RFINF(1:2),
           RPH(0:3);
'SIGNAL'EXLEC,OPLEC,OPECR,RAECR,TOPER,SMEMT(0:33),MLIBRE,BFUN,BAD(0:15),
         BDONE(0:15),Z(0:15),BDONS(0:15),DEMREF,ZERO(1:2),
         SFILE(0:15,1:2),COMPT(0:2,1:2),SCOMPT(0:3),JETON(1:4),
         JOCMEM(1:4),SRZ,SERRF;
'EXTERNE'MP('HORLOGE'(0:4),'HORLOGE',,,(0:15),(0:15),(0:3),;(0:15),),
          GESFI('HORLOGE'(0:4),,(0:15),,,;(0:15),(0:2)),
          AD4((0:3),(0:3),;(0:3));
EXLEC:=/.(ADRTO=10001).RT(0);
OPLEC:=/.(ADRLO=10001).RT(1);
OPECR:=/.(ADRTO=10001).RT(1);
RAECR:=/.(ADRTO=10001).RT(2);
TOPER:='SI'ROPF(1)'ALORS'TOPER'SINON'ROPER;
SMEMT:=MEMT(,$(RCOMPT(0:3)));
MLIBRE:=-/+ROCMEM;
BFUN:=ROCBUS(2)+ROCBUS(3).FADM;
'SI'ROCBUS'ALORS'(BAD:=RAD(,1);  BDONE:=Z)
                 (BAD:=RAD(,2);  BDONE:=SFILE(,2))
                 (BAD:=ADADM;    BDONE:=DOADME)
                 (BAD:=Z;        BDONE:=Z);
MP(H,HRAZ,-MLIBRE,BFUN,BAD,BDONE,RPH,ROCBUS(4);BDONS,DEMREF);
ZERO:=-/+COMPT(,1)&-/+COMPT(,2);
GESFI|1|(H,TFINOP(1)+RCOMPT(4).TOPER,'SI'ROPF(3)'ALORS'SMEMT(1:16)
          'SINON'BDONS,RRZ,ROPLEC.VAL1+RCOMPT(4).-TOPER.-ZERO(1),
          TFINOP(1)+RCOMPT(4).TOPER.SMEMT(0);SFILE(,1),COMPT(,1));
GESFI|2|(H,ROPECR.VAL1+RCOMPT(4).TOPER,'SI'ROPF(3)'ALORS'SMEMT(18:33)
          'SINON'TO,RRZ,TFINOP(2)+RCOMPT(4).-TOPER.-ZERO(2),
          ROPECR.VAL1+RCOMPT(4).TOPER.SMEMT(17);SFILE(,2),COMPT(,2));
DOP:='SI'OPLEC'ALORS'SFILE(,1)'SINON'Z;
VALP:=-/+((EXLEC.(COMPT(0,1)+RINTP(1)))&(OPLEC.(ZERO(1)+RINMEM))&
          (OPECR.(COMPT(0,2)+RINMEM))&(RAECR.(ZERO(2)+RINTP(2)))&
          (CO5.RT(1).RINMEM));
ERRAD:=RERRAD;
ERRF:=RERRF(2);
DODANS:=BDONS;
LECADM:=TFINOP(3);
ERRADM:=/+BAD(0:7);
DUSADM:=ROCBUS(3);
DEBADM:=RDEADM;
FINADM:=-ROCMEM(3);
AD4(RCOMPT(0:3),RM4&RM4&00,RCOMPT(4);SCOMPT);
JETON:=RJ+*R|1|-RUOC.(*R|1|RJ+*R|2|-RUOC.(*R|2|RJ+*R|3|-RUOC));
JOCMEM:=ROCMEM+RUOC.JETON.(MLIBRE&MLIBRE&MLIBRE&MLIBRE).(-TERRAD&11);
SRZ:=ROPF(1).RDAC.TOPER.-ROPF(2);
SERRF:='SI'ROPER'ALORS'-/+RCOMPT(0:1)'SINON'/.RCOMPT(0:1);
<H(0)>ROPER<=ADRTO(2),
      'SI'RCOMPT(4).-TOPER'ALORS'MEMT(,$(RCOMPT(0:3)))<=-ZERO(1)&
                                   SFILE(,1)&-ZERO(2)&SFILE(,2),
      'SI'RCHAR'ALORS'(RAD(,1)<=TO)
                      (RAD(,2)<=TO),
      ROPLEC<=OPLEC,
      ROPECR<=OPECR,
      TERRAD<='SI'/+TO(0:7)'ALORS'RCHAR'SINON'00,
      ROPF(0)<=CO5.RT(0),
      REXLEC<=EXLEC,
      RRAECR<=RAECR;
<H(1)>RT<=*R|-1|RT,
      RCOMPT(0:3)<=SCOMPT,
      RCOMPT(4)<=ROPF(3).-RCOMPT(4),
      ROCBUS<=JOCMEM,
      TFINOP<='SI'RPH(2)'ALORS'ROCBUS(1:3).(11&BFUN)'SINON'000,
      RINTP<=RUOC(1:2)+(ROPF(1)&ROPF(1))*
             ROCMEM(1:2).(/.COMPT(1:2,1)&(-COMPT(1,2).COMPT(2,2))),
      RINMEM<=ROPF(1),
      RDEADM<=RUOC(3).JETON(3).MLIBRE;
<H(2)>ROCMEM<=-ROCMEM.JOCMEM+ROCMEM.-(RPH(3)&RPH(3)&RPH(3)&RPH(3)),
      ROPF(3)<='SI'ROPF(3)'ALORS'-RFINF(2)'SINON'/.ROPF(1:2),
      RRZ<=SRZ,
      RERRAD<=/+TERRAD,
      RERRF(2)<=RERRF(1),
      RM4<=SRZ+TOPER.RFINF(1),
      'SI'RPH(3)'ALORS'RJ<=*R|-1|RJ;
<H(3)>ROPF(1:2)<=('SI'ROPF(1)'ALORS'-RFINF(1)'SINON'ROPF(0).VAL1.-SERRF)
                 &(ROPF(1).RDAC),
      RUOC<=-RUOC.((REXLEC.VAL1)&(RRAECR.VAL1)&DEMADM&DEMREF)+
             RUOC.-((TERRAD+ROCBUS(1:2).(RPH(1)&RPH(1)))&
                   (ROCBUS(3:4).(RPH(2)&RPH(2)))),
      RDAC<=/.-RUOC(1:2),
      RCHAR<='SI'VAL1'ALORS'REXLEC&RRAECR'SINON'00,
      RFINF(2)<=RFINF(1),
      RERRF(1)<=ROPF(0).VAL1.SERRF;
<H(4)>RPH<='SI'MLIBRE'ALORS'0000'SINON'1&*D|-1|RPH,
      RFINF(1)<=/.(RCOMPT(2:3)&-RCOMPT(4));
<HRAZ>RT<=100,
      RCOMPT<=00000,
      ROCBUS<=0000,
      RDEADM<=0,
      ROCMEM<=0000,
      ROPF(1:3)<=000,
      RRZ<=1,
      RERRAD<=0,
      RERRF<=00,
      RM4<=0,
      RJ<=1000,
      RUOC<=0000,
      RCHAR<=00,
      RPH<=0000,
      RFINF(1)<=0;

'UNITE'MP(H(0:4),HRAZ,OCMEM,FUN,AD(0:15),DONE(0:15),PH(0:3),DEBREF;
          DONS(0:15),DEMREF);
'HORLOGEMERE'H,HRAZ;
'REGISTRE'MEM(0:15,0:255),RD(0:31);
'SI'-/+AD(0:7)'ALORS'DONS:=MEM(,$(AD));
DEMREF:=RD(0);
<H(4)>'SI'PH(1).FUN.-/+AD(0:7)'ALORS'MEM(,$(AD))<=DONE,
      RD<=*R|1|RD;
'POUR'I=0'A'30'DEBUT'<HRAZ>RD(I)<=0;'FIN';
<HRAZ>RD(31)<=1;

'UNITE'GESFI(H(0:4),CHFILE,EFILE(0:15),RZ,SM,SP;SFILE(0:15),COMPT(0:2));
'HORLOGEMERE'H;
'REGISTRE'FILE(0:15,0:15),RENTRE(0:1),RSORT(0:1),RCOMPT(0:2),TM,TP;
'SIGNAL'SENTRE(0:3),SSORT(0:3),SCOMPT(0:3);
'EXTERNE'AD4((0:3),(0:3),;(0:3));
SFILE:=FILE(,$(RSORT));
COMPT:=RCOMPT;
AD4|1|(00&RSORT,0&RCOMPT,0;SENTRE);
AD4|2|(00&RSORT,0000,TM;SSORT);
AD4|3|(0&RCOMPT,0&TM&TM&TM,TP;SCOMPT);
<H(0)>'SI'CHFILE'ALORS'FILE(,$(RENTRE))<=EFILE,
      RENTRE<=SENTRE(2:3);
<H(1)>RSORT<=SSORT(2:3),
      RCOMPT<='SI'RZ'ALORS'000'SINON'SCOMPT(1:3);
<H(2)>TM<=SM,
      TP<=SP;

'UNITE'AD4(A(0:3),B(0:3),C;D(0:3));
'SIGNAL'R(0:3);
R:=*D|1|(A.B+A.R+B.R)&C;
D:=(A#B)#R;
```

FIGURE A.7. UNITE MEM: MEMOIRE PRINCIPALE ET CIRCUITS POR SA GESTION. ACCES DIRECT. FILES FLEC ET FECR ET CIRCUITS DE SAUVEGARDE

A

```
'UNITE'PRIN(HH,FADH,ADADH(0:15),DOADME(0:15),DEMADH;
            HALTE,DOADMS(0:15),LECADH,ERRADH,BUSADH,DEBADH,FINADH);
'HORLOGEMERE'HH;
'HORLOGE'H(0:4),HRAZ;
'REGISTRE'RH(0:4),RAZ,APPEL(2:8);
'SIGNAL'MI(0:29),CO(0:7),XCONTE(0:15,0:3),VAL1,ERRMM,XBUSPO(0:15,1:2)
        XBUSDO(0:15,1:5),ADRDO(0:4),ADRTO(0:4),ZERO,XVALO(2:5),CCONT,
        REPONS(2:8),TO(0:15),TEST,ERRDEB,ERRAD,ERRF,VALO,BUSPO(0:15),
        BUSDO(0:15),CONTEX(0:15);
'EXTERNE'MIMEM('HORLOGE'(0:4),'HORLOGE',,,(0:15),,(0:15),;
                (0:29),(0:7),(0:15),,),
         REGE('HORLOGE'(0:4),'HORLOGE',,(0:15),(0:29),,;
              (0:15),(0:15),(0:4),(0:4),),
         ETAT('HORLOGE'(0:4),'HORLOGE',(2:8),(0:4),(0:15),(0:15),,,,
              (6:7),(0:4),,;
              (0:15),(0:15),,,(2:8),),
         ARIT('HORLOGE'(0:4),'HORLOGE',(0:4),,,(0:15),(0:15),(0:4),,
              (0:23),;
              (0:15),(0:15),,(0:15),),
         MULT('HORLOGE'(0:4),'HORLOGE',(0:4),,,(0:15),(0:15),;(0:15),
         MEM('HORLOGE'(0:4),'HORLOGE',(0:4),(0:4),,(0:15),(0:15),,
             (0:15),,;
             (0:15),,,,(0:15),,,,,);
MIMEM(H,HRAZ,TEST,VALO,BUSPO,CCONT,BUSDO,ADRTO(2);
      MI,CO,XCONTE(,0),VAL1,ERRMM);
REGE(H,HRAZ,CCONT,TO,MI,VAL1,CO(1);
     XBUSPO(,1),XBUSDO(,1),ADRDO,ADRTO,ZERO);
ETAT(H,HRAZ,APPEL,ADRDO,CONTEX,TO,ERRDEB,ERRF,ERRAD,CO(6:7),ADRTO,
     ERRMM,VAL1;
     XBUSPO(,2),XBUSDO(,2),XVALO(2),CCONT,REPONS,HALTE);
ARIT(H,HRAZ,ADRDO,CO(1),ZERO,BUSPO,BUSDO,ADRTO,CCONT,MI(0:23),VAL1;
     XBUSDO(,3),TO,TEST,XCONTE(,3),ERRDEB);
MULT(H,HRAZ,ADRDO,CO(7),VAL1,BUSPO,BUSDO,ADRTO(2);
     XBUSDO(,4),XVALO(4));
MEM(H,HRAZ,ADRTO,ADRDO,FADM,ADADM,DOADME,VAL1,TO,CO(5),DEMADM;
    XBUSDO(,5),XVALO(5),ERRAD,ERRF,DOADMS,LECADM,ERRADM,BUSADM,DEBADM
    FINADM);
VALO:=XVALO(2).XVALO(4).XVALO(5);
BUSPO:=XBUSPO(,1)+XBUSPO(,2);
BUSDO:=XBUSDO(,1)+XBUSDO(,2)+XBUSDO(,3)+XBUSDO(,4)+XBUSDO(,5);
CONTEX:=XCONTE(,0)+XCONTE(,3);
<HH>RH<='SI'RAZ'ALORS'10000'SINON'('SI'HALTE'ALORS'RH'SINON'*R|-1|RH)
    RAZ<=0,
    H:='SI'RAZ+HALTE'ALORS'00000'SINON'RH,
    HRAZ:=RAZ,
    APPEL<=APPEL.-REPONS;
```

FIGURE A.8. UNITE PRIN: CONNEXIONS.

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

[1] F. ANCEAU,
"Microprogrammed system for tasks management".
NATO Advanced Summer Institute On Microprogramming. St-Raphaël.
Août-Septembre 1971.

[2] F. ANCEAU,
"Synchronisation dans les systèmes hiérarchisés de processus-processeu
Congrès AFCET - GRENOBLE.
Novembre 1972.

[3] G.S. TJADEN et M.J. FLYNN,
"Detection and parallel execution of independant instructions".
IEEE Transactions on Computers, Vol. C-19, n° 10.
Octobre 1970.

[4] D.W. ANDERSON, F.J. SPARACIO et R.M. TOMASULO,
"The IBM system/360 model 91 : machine philosophy and instruction handling".
IBM Journal. Janvier 1967.

[5] B. GOLD, I.L. LEBOW, P.G. Mc HUGH et C.M. RADER,
"The FDP, a fast programmable signal processor".
IEEE Transactions on computers, Vol. C-20, n° 1.
Janvier 1971.

[6] J.L. ROSENFELD et R.D. VILLANI,
"Micro-multiprocessing : an approach to multiprocessing at the level of very small tasks".
IEEE Transactions on Computers, Vol. C-22, n° 2.
Février 1973.

[7] J.M. KURTZBERG et R.D. VILLANI,
"A balanced pipellining approach to multiprocessing on an instruction stream level".
IEEE Transactions on Computers, Vol. C-22, n° 2.
Février 1973.

[8] E. MITRANI,
"Réseau d'opérateurs adapté au traitement du signal".
Journées d'Etudes IRIA. Structures résultant d'un groupement de Processeurs - Saint-Pierre de Chartreuse.
22-23 Novembre 1973.

[9] C. BRUNET,
"Etude d'une structure biprocesseur banalisé".
Thèse soutenue au Laboratoire d'Automatique de Grenoble INPG
7 Mars 1974.

[10] E.W. DIJKSTRA,
"Cooperating sequential processes".
Tech. University Eindhoven - EWD - 123 - The Netherlands.
Septembre 1965.

[11] P. ULLMAN,
"Coopération entre tâches et dispatching".
EMP Fontainbleau.
Janvier 1971.

[12] E.W. DIJKSTRA,
"The structure of the multiprogramming system".
Communications of the ACM, Vol. 11, n° 5.
Mai 1968.

[13] H.J. SAAL et W.E. RIDDLE,
"Communicating semaphores".
Stanford Linear Accelerator Center. Stanford, CA. 94305.
Computer science Department.
Décembre 1970.

[14] S.S. PATIL,
"Limitations and capabilities of Dijkstra's semaphore primitives for coordination among processes".
Computation Structures Group MEMO 57.
Massachussetts Institute of Technology. Project MAC.
Février 1971.

[15] E. METRANI,
"Etude d'un réseau d'opérateurs adapté au traitement du signal".
Rapport des travaux de Thèse. Note IETI-MCTE/72-1882/EM/JQ.
24 Mai 1972.

[16] J. MERMET,
"Définition du Langage CASSANDRE".
Thèse soutenue à Grenoble - IMAG - INPG.
Mars 1970.

[17] S. SANKAR,
"Conception des filtres numériques et analyse des erreurs résultant de leur réalisation en arithmétique fixe".
Thèse soutenue à Grenoble - IMAG - INPG.
20 Mars 1974.

[18] W. COCHRAN, J. COOLEY, D. FAVIN, H. HELMS, R. KAENEL, W. LANG, G. MALING
D. NELSON, C. RADER et P. WELCH,
"What is the fast Fourier transform ?".
Proceedings of the IEEE, Vol. 55, n° 10.
Octobre 1967.