HSURF : un microprocesseur facilement testable pour des applications à haute sûreté de fonctionnement

Christian Jay

THESE

Présentée par

Christian JAY

pour obtenir le grade de DOCTEUR

de L' UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

Spécialité : Microélectronique

"HSURF"

UN MICROPROCESSEUR FACILEMENT TESTABLE POUR DES APPLICATIONS A HAUTE SURETE DE FONCTIONNEMENT

Soutenue le 23 Juin 1986 devant la commission d'examen :

Président :
G. SAUCIER

Examinateurs :
J.P. AUCLAIR
J.F. CROCE-SPINELLI
F. JUTAND
G. NOGUEZ

These preparée au sein du **Laboratoire circuits et systèmes**

UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

Année universitaire 1982-1983

Président de l'Université : M. TANCHE

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

(RANG A)

SAUF ENSEIGNANTS EN MEDECINE ET PHARMACIE

PROFESSEURS DE 1ère CLASSE

ARNAUD Paul	Chimie organique
ARVIEU Robert	Physique nucléaire I.S.N.
AUBERT Guy	Physique C.N.R.S.
AYANT Yves	Physique approfondie
BARBIER Marie-Jeanne	Electrochimie
BARBIER Jean-Claude	Physique expérimentale C.N.R.S. (labo de magnétisme)
BARJON Robert	Physique nucléaire I.S.N.
BARNOUD Fernand	Biosynthèse de la cellulose-Biologie
BARRA Jean-René	Statistiques - Mathématiques appliquées
BELORISKY Elie	Physique
BENZAKEN Claude (M.)	Mathématiques pures
BERNARD Alain	Mathématiques pures
BERTRANDIAS Françoise	Mathématiques pures
BERTRANDIAS Jean-Paul	Mathématiques pures
BILLET Jean	Géographie
BONNIER Jean-Marie	Chimie générale
BOUCHEZ Robert	Physique nucléaire I.S.N.
BRAVARD Yves	Géographie
CARLIER Georges	Biologie végétale
CAUQUIS Georges	Chimie organique
CHIBON Pierre	Biologie animale
COLIN DE VERDIERE Yves	Mathématiques pures
CRABBE Pierre (détaché)	C.E.R.M.O.
CYROT Michel	Physique du solide
DAUMAS Max	Géographie
DEBELMAS Jacques	Géologie générale
DEGRANGE Charles	Zoologie
DELOBEL Claude (M.)	M.I.A.G. Mathématiques appliquées
DEPORTES Charles	Chimie minérale
DESRE Pierre	Electrochimie
DOLIQUE Jean-Michel	Physique des plasmas
DUCROS Pierre	Cristallographie
FONTAINE Jean-Marc	Mathématiques pures
GAGNAIRE Didier	Chimie physique

.../...

GASTINEL Noël	Analyse numérique - Mathématiques appliquées
GERBER Robert	Mathématiques pures
GERMAIN Jean-Pierre	Mécanique
GIRAUD Pierre	Géologie
IDELMAN Simon	Physiologie animale
JANIN Bernard	Géographie
JOLY Jean-René	Mathématiques pures
JULLIEN Pierre	Mathématiques appliquées
KAHANE André (détaché DAFCO)	Physique
KAHANE Josette	Physique
KOSZUL Jean-Louis	Mathématiques pures
KRAKOWIAK Sacha	Mathématiques appliquées
KUPTA Yvon	Mathématiques pures
LACAZE Albert	Thermodynamique
LAJZEROWICZ Jeannine	Physique
LAJZEROWICZ Joseph	Physique
LAURENT Pierre	Mathématiques appliquées
DE LEIRIS Joël	Biologie
LLIBOUTRY Louis	Géophysique
LOISEAUX Jean-Marie	Sciences nucléaires I.S.N.
LOUP Jean	Géographie
MACHE Régis	Physiologie végétale
MAYNARD Roger	Physique du solide
MICHEL Robert	Minéralogie et pétrographie (géologie)
MOZIERES Philippe	Spectrométrie - Physique
OMONT Alain	Astrophysique
OZENDA Paul	Botanique (biologie végétale)
PAYAN Jean-Jacques (détaché)	Mathématiques pures
PEBAY PEYROULA Jean-Claude	Physique
PERRIAUX Jacques	Géologie
PERRIER Guy	Géophysique
PIERRARD Jean-Marie	Mécanique
RASSAT André	Chimie systématique
RENARD Michel	Thermodynamique
RICHARD Lucien	Biologie végétale
RINAUDO Marguerite	Chimie CERMAV
SENGEL Philippe	Biologie animale
SERGERAERT Francis	Mathématiques pures
SOUTIF Michel	Physique
VAILLANT François	Zoologie
VALENTIN Jacques	Physique nucléaire I.S.N.
VAN CUTSEN Bernard	Mathématiques appliquées
VAUQUOIS Bernard	Mathématiques appliquées
VIALON Pierre	Géologie

PROFESSEURS DE 2ème CLASSE

ADIBA Michel	Mathématiques pures
ARMAND Gilbert	Géographie

.../...

AURIAULT Jean-Louis	Mécanique
BEGUIN Claude (M.)	Chimie organique
BOEHLER Jean-Paul	Mécanique
BOITET Christian	Mathématiques appliquées
BORNAREL Jean	Physique
BRUN Gilbert	Biologie
CASTAING Bernard	Physique
CHARDON Michel	Géographie
COHENADDAD Jean-Pierre	Physique
DENEUVILLE Alain	Physique
DEPASSEL Roger	Mécanique des fluides
DOUCE Roland	Physiologie végétale
DUFRESNOY Alain	Mathématiques pures
GASPARD François	Physique
GAUTRON René	Chimie
GIDON Maurice	Géologie
GIGNOUX Claude (M.)	Sciences nucléaires I.S.N.
GUITTON Jacques	Chimie
HACQUES Gérard	Mathématiques appliquées
HERBIN Jacky	Géographie
HICTER Pierre	Chimie
JOSELEAU Jean-Paul	Biochimie
KERCKOVE Claude (M.)	Géologie
LE BRETON Alain	Mathématiques appliquées
LONGEQUEUE Nicole	Sciences nucléaires I.S.N.
LUCAS Robert	Physiques
LUNA Domingo	Mathématiques pures
MASCLE Georges	Géologie
NEMOZ Alain	Thermodynamique (CNRS - CRTBT)
OUDET Bruno	Mathématiques appliquées
PELMONT Jean	Biochimie
PERRIN Claude (M.)	Sciences nucléaires I.S.N.
PFISTER Jean-Claude (détaché)	Physique du solide
PIBOULE Michel	Géologie
PIERRE Jean-Louis	Chimie organique
RAYNAUD Hervé	Mathématiques appliquées
ROBERT Gilles	Mathématiques pures
ROBERT Jean-Bernard	Chimie physique
ROSSI André	Physiologie végétale
SAKAROVITCH Michel	Mathématiques appliquées
SARROT REYNAUD Jean	Géologie
SAXOD Raymond	Biologie animale
SOUTIF Jeanne	Physique
SCHOOL Pierre-Claude	Mathématiques appliquées
STUTZ Pierre	Mécanique
SUBRA Robert	Chimie
VIDAL Michel	Chimie organique
VIVIAN Robert	Géographie

ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

Directeur	: M. M. MERMET
Directeur des Etudes et de la formation	: M. J. LEVASSEUR
Directeur des Recherches	: M. J. LEVY
Secrétaire Général	: Mle M. CLERGUE

Professeurs de la 1ère Catégorie

COINDE	Alexandre	Gestion
GOUX	Claude	Metallurgie
LEVY	Jacques	Métallurgie
LOWYS	Jean-Pierre	Physique
MATHON	Albert	Gestion
RIEU	Jean	Mécanique-Résistance des matériaux
SOUSTELLE	Michel	Chimie
FORMERY	Philippe	Mathématiques Appliquées

Professeurs de 2ème catégorie

HABIB	Michel	Informatique
PERRIN	Michel	Géologie
VERCHERY	Georges	Matériaux
TOUCHARD	Bernard	Physique industrielle

Directeur de recherche

LESBATS	Pierre	Métallurgie

Maîtres de recherche

BISCONDI	Michel	Métallurgie
DAVOINE	Philippe	Géologie
FOURDEUX	Angeline	Métallurgie
KOBYLANSKI	André	Métallurgie
LALAUZE	René	Chimie
LANCELOT	Francis	Chimie
LE COZE	Jean	Métallurgie
THEVENOT	François	Chimie
TRAN MINH	Canh	Chimie

Personnalités habilitées à diriger des travaux de recherche

DRIVER	Julian	Métallurgie
GUILHOT	Bernard	Chimie
THOMAS	Gérard	Chimie

Professeur à l'UER de Sciences de Saint-Etienne

VERGNAUD	Jean-Maurice	Chimie des Matériaux & chimie industrielle

****

Je tiens à remercier :

Madame G. SAUCIER, professeur à l'I.N.P-Grenoble, qui me fait l'honneur d'assurer la présidence du jury. Je tiens par la même occasion à lui exprimer toute ma reconnaissance pour m'avoir accueilli dans son laboratoire.

Monsieur G. NOGUEZ, professeur à l'Institut de Programmation (PARIS VI), d'avoir accepté d'être rapporteur de cette thèse.

Monsieur F. JUTAND, professeur à l'Ecole Nationale Supérieure de Télécommunications (Paris), d'avoir accepté d'être rapporteur de cette thèse.

Monsieur J.P. AUCLAIR, Chef de la division VZA de la S.N.C.F., qui est pour une grande part à l'origine de cette étude et dont j'ai pu apprécier les remarques au cours d'une collaboration fructueuse avec ses services.

Monsieur J.F. CROCE-SPINELLI, directeur du développement et du plan à la Compagnie des Signaux et d'Entreprises Electriques qui me fait l'honneur de participer à ce jury.

Je remercie également tous les membres du Laboratoire Circuits et Systèmes et tout particulièrement P. GENESTIER pour sa collaboration dans la réalisation de ce travail.

et, pour terminer, je voudrais remercier Claudine qui grâce à son aide morale et matérielle m'a permis de mener à terme ce travail.

RESUME :

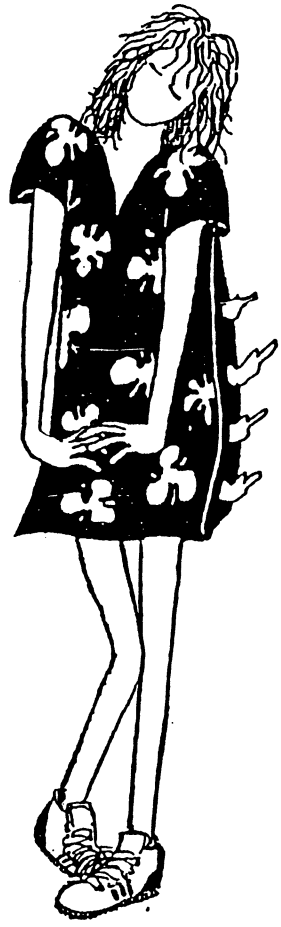
L'objet de l'étude est la conception d'un composant du type microprocesseur pour des applications à haute sûreté de fonctionnement. Le circuit conçu est caractérisé par une faible latence d'erreur, par des facilités de test en fin de fabrication et par des facilités d'implantation de mécanismes de détection d'erreurs en ligne (codage et vérification de l'information traitée).

Partant d'un jeu d'instructions spécifiques à l'application (domaine des automatismes logiques), on propose une architecture permettant d'éxécuter ledit jeu d'instructions et disposant de facilités de test en fin de conception et au cours de la vie du circuit. L'observabilité et la contrôlabilité du composant représentent une partie importante de l'étude. Après examen critique de plusieurs méthodes permettant de faciliter le test (en ligne et hors ligne) du circuit, un choix est réalisé afin d'intégrer dans l'architecture du microprocesseur les dispositifs nécessaires à la mise en œuvre de certaines d'entre elles.

Ce microprocesseur à été partiellement réalisé en technologie C.MOS dans le cadre du "Circuits-Multi-Projets".

MOTS-CLES :

MICROPROCESSEUR, CIRCUIT SPECIFIQUE, CIRCUIT INTEGRE, TEST-EN-LIGNE, TEST-HORS-LIGNE, CONTROLABILITE, OBSERVABILITE, SURETE DE FONCTIONNEMENT.



à ma puce ...

INTRODUCTION

L'introduction dans les systèmes de contrôle à haute sûreté de fonctionnement de circuits programmables à très haute intégration (V.L.S.I) pose des problèmes très critiques.

Les applications concernées sont essentiellement le transport terrestre (Train, Métro), l'avionique (Equipements embarqués, Pilotes automatiques), le spatial, le nucléaire (Contrôle de processus de centrale nucléaire) et certains domaines de contrôle de processus industriel.

Le concepteur de tels systèmes éprouvera toujours certaines difficultés avant d'acquérir la confiance nécessaire à l'utilisation de composants nouveaux dans des applications mettant en jeu des vies humaines (Transports), des investissements élevés (Spatial) ou risquant d'apporter des nuisances graves consécutives à un fonctionnement erroné. Pour de telles applications, il convient de prouver en fin de conception la conformité du système de contrôle aux spécifications initiales, en fin de fabrication l'intégrité du matériel (Composants sans défaillance à l'instant initial) et, au cours de son utilisation, la sécurité du système (non émission de commandes dangereuses par suite de défaillance du dispositif).

Pour atteindre ce dernier objectif, les architectures utilisées sont des architectures redondantes. L'application est implémentée sur 2, 3 ou 4 "Systèmes indépendants", et une comparaison des sorties permet d'améliorer la sécurité. Une implémentation "simplex", même munie de tests en ligne puissants pourra très difficilement être considérée comme un dispositif à haute sécurité.

La réalisation d'un système à haute sûreté de fonctionnement nécessite une évaluation minutieuse de sa sécurité en vue de certification, par exemple, or cette évalu ation se heurte pour les circuits intégrés actuels du type

microprocesseur à des difficultés spéciales. Ces dernières sont dues à la difficulté d'assurer un test efficace et sûr de ces dispositifs. Dans [BEL 84], l'auteur développe différentes stratégies relatives au test de tels composants. Il apparaît vue leur complexité, qu'un test exhaustif (test par identification) est inenvisageable. De même, un test par distinction élaboré sur une description structurelle du circuit sur laquelle est définie un ensemble de pannes s'avère d'une mise en oeuvre délicate. En effet, cette méthode demande une très bonne connaissance du circuit à son niveau logique, électrique et même parfois au niveau du dessin des masques (souvent in accessible à l'utilisateur). Elle demande aussi d'avoir une bonne connaissance de la modélisation des pannes au niveau technologique, ce qui est impossible actuellement dans les technologies N.MOS et C.MOS [WAD 78] [TIM 83].

De plus, si l'on trouvait une modélisation suffisante des défaillances au niveau des transistors ou de la porte, l'élaboration de tests détectant ces défaillances au niveau d'un dispositif comprenant plusieurs dizaines de milliers de transistors serait d'une complexité prohibitive.

On peut donc se demander à juste raison, si une application à haute sûreté de fonctionnement implantée sur des dispositifs qui sont testés en ligne et hors ligne de façon imparfaite pourra être l'objet d'une certification valable.

Considérons à titre d'exemple l'architecture du poste d'aiguillage informatisé (PAI) [SEV 84], implémenté sur 2 microprocesseurs 68000 (figure 1).

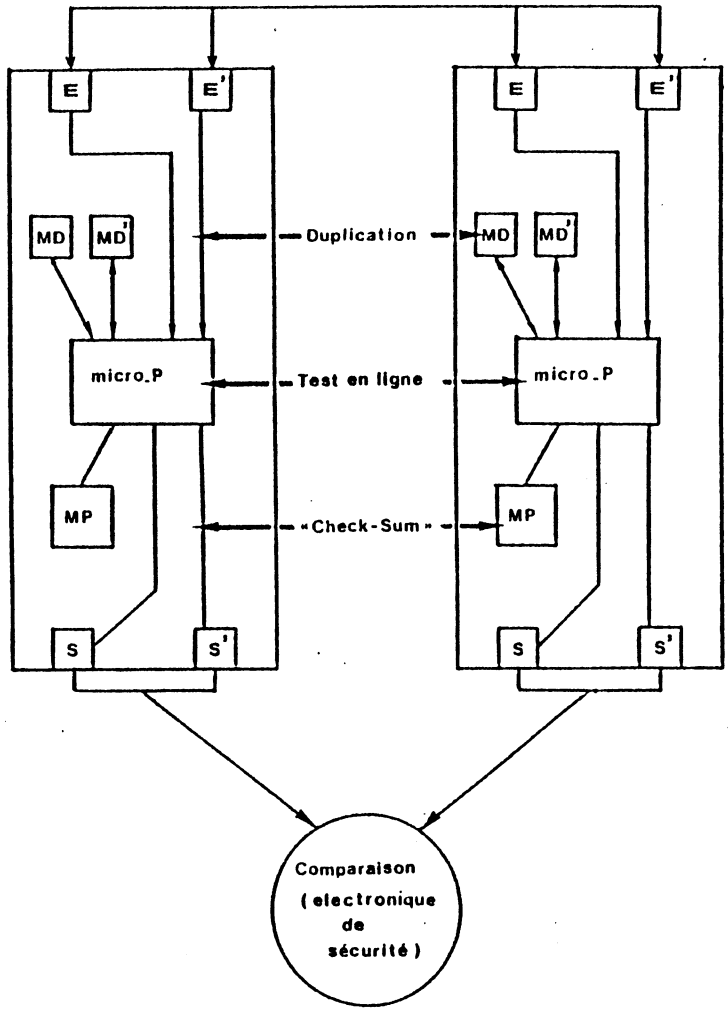


Figure 1

Le P.A.I. est un système basé sur une statégie de tolérance aux pannes multi-niveaux.

i) Il est composé de 2 unités identiques isolées (Duplex) dont les résultats (émission de commandes) sont constamment comparés à l'aide d'un comparateur de sécurité réalisé en électronique hybride.

ii) Chaque unité contient des dispositifs internes de détection d'erreurs.

Au sein de chaque dispositif la sécurité est implémentée par des techniques de redondance ; on a une duplication des mémoires de données (MD et MD'), des entrées (E et E') et des sorties (S et S') dont les valeurs sont comparées.

Le programme d'application est stocké dans une mémoire programme (MP) dont le contenu est périodiquement testé par vérification de propriétés invariantes ("check-sum"). Le microprocesseur est partiellement testé par l'utilisation d'un détecteur de codes invalides, par l'activation de "chiens de garde" et par l'exécution régulière de séquences de test (test en oisiveté).

L'évaluation de l'efficacité des stratégies adoptées dans cette réalisation se heurte à l'utilisation de V.L.S.I. et pose le problème du manque d'informations concernant leurs modes de défaillances (difficultés d'observation vu la complexité du problème, absence d'hypothèses de pannes réalistes,...). La méthode d'évaluation devra donc soit éliminer ces inconnues par prise en compte d'hypothèses pessimistes, soit les faire influer dans la détermination du taux d'insécurité de l'application.

L'approche développée dans [PIL 82] va dans le sens de la deuxième solution, et son application au poste d'aiguillage informatisé donne les résultats de la figure 2.

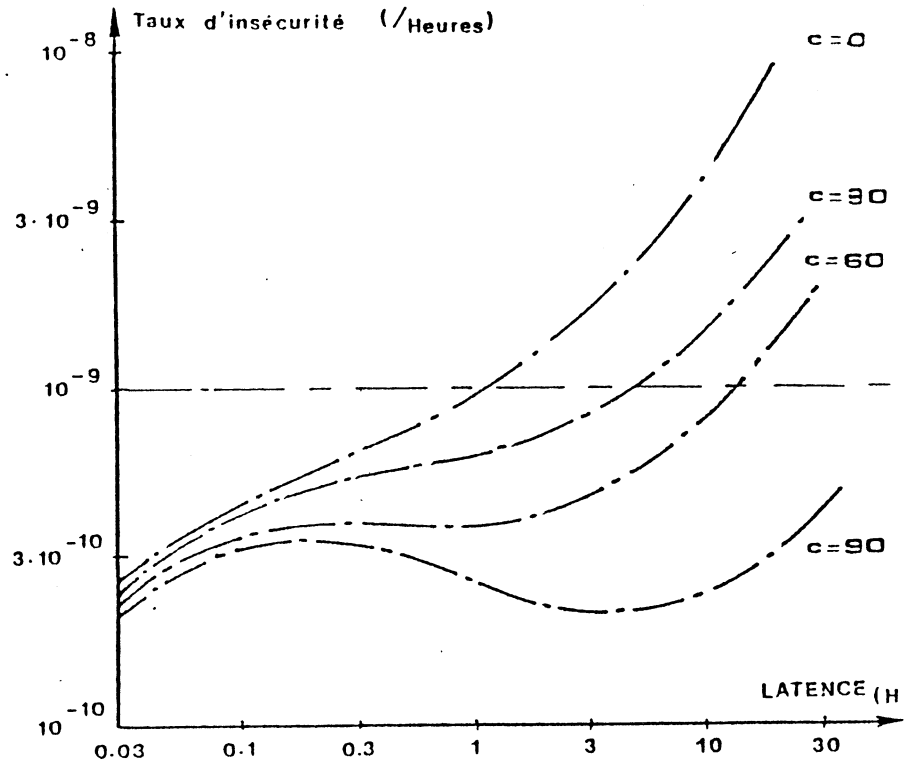


Figure 2

Ces courbes montrent l'évolution du taux d'insécurité du P.A.I. en fonction de deux paramètres mal connus qui sont le taux de couverture du test en ligne du microprocesseur (autotest par déroulement d'un programme de test totalement exécuté toutes les trois heures) et la latence des pannes non détectées par ces programmes de test (ces pannes engendreront une erreur dans le programme d'application au bout de H heures, H étant cette latence).

L'examen de ces résultats permet de définir facilement quelles sont les caractéristiques attendues pour un microprocesseur assurant le contrôle d'une application à haute sûreté de fonctionnement ; on pourrait par exemple exiger :

- Que le taux de couverture des programmes de test soit au moins de 60 %.
- Que la latence d'une panne pouvant échapper à ces programmes soit majorable en fonction de la fréquence d'exécution du programme de test (soit par exemple pour le PAI : H $<$ 2 heures).

Le développement de circuits spécifiques destinés aux environnements à haute sécurité devra donc avoir pour objectif principal d'obtenir des produits dans lesquels toute panne sera susceptible d'être détectée rapidement, soit par le programme d'application associé à des dispositifs de test en ligne puissant, soit par l'exécution de programmes de test spécifiques destinés à vérifier certaines ressources précises sous-utilisées par les programmes d'application ou dont l'intégrité est vitale (test en ligne discontinu pendant des périodes d'oisiveté de l'application).

Si nous prenons l'exemple du 68000 utilisé dans l'application du poste d'aiguillage informatisé, on sera gêné pour diverses raisons :

- L'application met en jeu des fonctions classiques des automatismes logiques ce qui représente une toute petite proportion des possibilités opérationnelles de ce microprocesseur (environ 30%). En particulier, les possibilités d'interruption, sauvegarde du contexte et la plupart des instructions de branchement ne sont pas utilisées. On peut donc craindre à juste raison qu'une panne reste cachée fort longtemps et que cette latence nous mette dans une situation délicate de double panne (dans les deux dispositifs).

- Le test périodique (test en oisiveté) sera difficile à mettre en oeuvre. S'il n'est pas trop délicat de balayer les instructions non utilisées par le programme d'application, il n'en sera pas de même pour la logique d'interruption, d'interface asynchrone. Il est d'ailleurs à noter que de tels dispositifs poseront toujours de gros problèmes aux concepteurs d'applications à haute sécurité ; en effet, des mécanismes tels que les interruptions ont la capacité d'influer sur le déroulement de programme d'application d'une façon non contrôlable temporellement, ainsi des dispositifs de test en ligne basés sur la vérification des flots d'instructions ou sur la mise en place de "chiens de garde" seront délicats à mettre en oeuvre.

Il faut aussi rappeler que la couverture des tests sera pénalisée par la méconnaissance des hypothèses de défaillances et de la structure interne des circuits.

A la lueur des constatations précédentes, il apparaît donc nécessaire de développer un composant spécifique doté d'un maximum de dispositifs facilitant son immersion dans les applications demandant une grande sûreté de fonctionnement.

Quel "cahier des charges" pour un tel microprocesseur ?

La discussion précédente nous enseigne que :

1) Le microprocesseur ne doit pas être surdimensionné et être bien adapté à l'application qu'il contrôle :

Dans les domaines de contrôle de processus, il est inutile de disposer de systèmes dont seulement un faible pourcentage des capacités est utilisé, il ne faut pas non plus que du fait de sa simplicité, et de son manque de moyen de traitement, l'utilisateur soit contraint de développer des programmes d'application longs, compliqués, et donc impossibles à valider. Nous nous efforcerons donc de doter notre circuit d'un jeu d'instructions lui conférant les capacités de traitement d'un microprocesseur classique de 16 bits tout en étant adapté aux automatismes logiques travaillant en temps réel (comme le poste d'aiguillage informatisé).

2) Le microprocesseur doit avoir une latence de panne faible :

C'est le premier objectif pour un circuit à haute sureté de fonctionnement. Toute panne doit être détectée en un temps court, soit par le programme d'application, soit par un programme de test.
Tout sera donc mis en oeuvre pour faciliter la mise en place d'une stratégie de test en ligne efficace.

Afin de définir quels sont les dispositifs les mieux appropriés, nous allons, en un premier temps, discuter les différentes méthodes ayant fait l'objet de recherches dans ce domaine. Nous discuterons successivement les approches consistant à intégrer des dispositifs d'autotest dans les circuits, à intégrer des dispositifs de test dans le logiciel d'application, ou à déléguer les opérations de vérification à un coprocesseur supplémentaire.

a) Mise en place de dispositifs de test en ligne intégrés au processeur :

- Une approche codant les informations circulant à l'intérieur du circuit est proposée dans [MAR 78], [ASH 77], [SMI 83]. Un décodeur permet ainsi de détecter toute erreur intervenant dans l'information traitée.

- Dans [DIA 75] et Iyengar [IYE 82], est étudiée la détection de pannes dans les unités de contrôle microprogrammées. L'ensemble des états est partitionné et une fonction permet d'évaluer, à partir d'un état courant et de "clés" stockées dans le microprogramme, un ensemble d'états suivants possibles. La comparaison de cette évaluation avec l'état réellement obtenu permet de détecter les pannes survenant dans cette partie du circuit.

- Courtois [COU 79], [COU 81] et Courtois et Marchal [MAR 82] étudient divers mécanismes de détection d'erreurs (code opération invalide, détecteur d'adresse erronnée, détecteur de protection mémoire,...) résultant de collages permanents de lignes de bus ou d'un mauvais fonctionnement du compteur programme pour les microprocesseurs 6800 et 68000.

De telles méthodes sont basées sur une modification profonde de la structure des microprocesseurs. Leur principe utilise des techniques de codages [WAK 78], [BOS 82] (code de parité, code de Hamming...) ne

permettant souvent que la détection d'erreurs simples, la mise en place de dispositifs détectant les erreurs doubles ou triples devenant extrêmement compliquée. Leur implémentation résulte donc en une augmentation importante de la surface de silicium utilisée pour implanter les organes de stockage des divers codes et leurs dispositifs d'évaluation ; ceci va faire décroître le rendement, la fiabilité et la testabilité des composants pour une couverture de panne insuffisante.

On peut s'interroger sur le bien-fondé de telles méthodes, pour une utilisation dans des applications à haute sûreté de fonctionnement. En effet, un accroissement de la surface du circuit implique une diminution de sa fiabilité et un risque de défaillance plus important (plus de transistors implique plus de points susceptibles de tomber en panne).

b) Les dispositifs de test sont intégrés au logiciel :

Diverses approches, utilisant des dispositifs intégrés non à l'architecture, mais à son logiciel d'application, ont été développées :

- Dans [BEL 82], les auteurs modélisent le processeur par un graphe ; ce qui leur permet de définir des tests intégrés aux applications exécutées par le processeur, et de détecter les erreurs pouvant apparaître sur les entrées du circuit.

- Des techniques sont proposées dans [AYA 79] et [KAN 75] pour intégrer au logiciel d'application un "observateur" qui permet de vérifier la bonne exécution du programme.

- Parhami, Metze et Mili [PAR 77] et [MET 81] ont, pour leur part, développé des règles permettant l'écriture de programmes s'auto-testant lors de leur exécution.

Toutes ces méthodes impliquent une modification, non négligeable, des programmes d'application. Ceci accroît donc le temps de calcul, ainsi que l'espace mémoire occupé. Ils ont cependant l'avantage de ne pas demander de modifications au niveau de l'architecture du circuit.

Leur mise en oeuvre impose toutefois de modéliser les erreurs pouvant survenir, soit au niveau du composant, ce qui n'est pas aisé du fait de la méconnaissance du circuit et des hypothèses de fautes dans les technologies utilisées, soit au niveau du logiciel, ce qui est encore certainement plus délicat à obtenir ; les erreurs au niveau logiciel étant virtuellement impossible à modéliser ou à énumérer.

c) Méthodes consistant à décharger la "responsabilité" du test sur un autre processeur ou circuit spécialisé (Watchdog processor) :

- Mc Cluskey et Namjoo [MCC 82a] et [NAM 83] considèrent l'ajout d'un processeur détectant les pannes, logicielles, ou matérielles, causant un accès mémoire intempestif.

- Dans [NAM 82], [SRI 82] et [SHE 83b], l'analyse de signature, technique permettant le compactage des informations circulant dans le circuit, a été utilisée. Les informations étant codées, le résultat attendu peut être comparé à la signature obtenue.

- Chavade et Crouzet [CHA 82], s'appuyant sur le travail de Crouzet et Landrault [CRO 80] ont développé un circuit permettant à l'usager de réaliser des systèmes auto-test à base de microprocesseurs dupliqués.

Cette approche ne requiert pas de modifications profondes, ni du circuit au niveau matériel, ni du logiciel d'application. Cependant, la complexité de l'application se trouve accrue par la présence d'un "coprocesseur" ou de dispositifs supplémentaires destinés à la vérification du flot d'informations. Un tel coprocesseur devra, si l'on désire superviser de façon correcte le déroulement d'une application, posséder une capacité de traitement pratiquement équivalente au processeur contrôlant l'application elle-même. Les problèmes de validation du processeur principal se trouvent donc reportés sur la validation du coprocesseur de contrôle. D'autre part, dans des applications où l'on dispose déjà d'une architecture redondante (duplex, triplex...), est-il vraiment indispensable d'ajouter des dispositifs complexes pour assurer la vérification du bon fonctionnement de l'application ?

Dans l'ensemble des méthodes présentées précédemment, seules celles citées dans les parties b et c semblent présenter un réel intérêt dans l'optique de la réalisation d'un composant du type microprocesseur destiné à un environnement à haute sûreté de fonctionnement.

On s'aperçoit que l'on a intérêt à ne pas trop compliquer l'architecture du circuit ou de l'application (par utilisation de composants nouveaux qui s'avéreront générateurs de problèmes). Cependant, il apparaît indispensable de faciliter la tâche du programmeur en lui offrant des dispositifs permettant de mettre en place au niveau logiciel une stratégie de test efficace ; pour ce faire, les approches citées dans b), combinées à celles développées par Shen et Shuette dans [SHE 83b] qui consistent à réaliser la compaction des informations circulant dans le système paraissent représenter un bon compromis entre la complexité ajoutée à l'application, sa simplicité de mise en oeuvre et les impératifs de testabilité d'un environnement à haute sûreté de fonctionnement.

L'objet de cette thèse consiste à proposer un microprocesseur à faible latence de pannes : "HSURF", adapté aux environnements à haute sûreté de fonctionnement. Ceci est réalisé par l'implantation de dispositifs permettant de faciliter toutes les opérations de test en ligne et en oisiveté.

A) HSURF est un microprocesseur facilitant le test en ligne

L'architecture interne du microprocesseur HSURF a été étudiée afin d'être compatible avec ce qui a été énoncé précédemment. C'est ainsi qu'une analyse de signature, réalisée sélectivement sur les divers bus, permettra d'accroître la visiblilité interne du circuit.
L'utilisateur peut alors, suivant l'organisation du système utilisant HSURF:

- Soit comparer la signature des deux circuits fonctionnant en parallèle de façon synchrone.

- Soit comparer périodiquement les signatures obtenues au cours de l'exécution des programmes d'application à des valeurs précalculées (cas d'un fonctionnement duplex asynchrone par exemple).

B) HSURF est un microprocesseur facilitant le test en oisiveté

Suivant l'application dans laquelle HSURF fonctionne, le test en ligne permet dans la plupart des cas de réaliser un test complet des ressources internes au microprocesseur. Cependant, la plupart des hypothèses faites sur l'évaluation des systèmes à haute sûreté de fonctionnement ont en commun de considérer une latence de panne à distribution indépendante du temps : l'environnement doit être analogue à celui que délivrerait un testeur aléatoire. Or, dans la réalité, ce cas de figure est assez rare. Par exemple, le programme d'atterissage d'un pilote automatique d'avion n'est sollicité qu'une fois par vol ; de même, un système de commande d'aiguillages de chemins de fer rencontre des configurations périodiques de circulation des trains de période 24 heures (ou plus lorsque certaines voies d'une gare ne sont utilisées que très rarement).

On sera donc souvent contraint, pour pallier cette difficulté, et pour éviter que des ressources de l'application restent inutilisées pendant de longues périodes, de développer une stratégie de test en ligne discontinue (appelé test en oisiveté), permettant :

- soit de générer les configurations d'environnement se produisant naturellement trop rarement en activant les organes sous-utilisés.

- soit de tester de la façon la plus complète possible les organes ou opérateurs sous-utilisés. La possibilité de réaliser un test exhaustif de ces organes, étant, bien entendu, l'idéal.

Le microprocesseur HSURF possède, dans le but de rendre plus aisé ce test en oisiveté, un ensemble d'opérateurs testable le plus complètement, facilement et rapidement possible. Pour cela, tout opérateur est rendu totalement accessible : les registres d'entrée et de sortie de ces opérateurs peuvent être modifiés ou observés, à tout moment, au cours de l'exécution de programmes de test spécifiques à l'opérateur étudié. Cet impératif

d'accessibilité a demandé l'implantation d'instructions de test spéciales à HSURF :

- test des registres de la partie contrôle (RI, micro-RI, micro-CO...)

- test des opérateurs (U.A.L., MULT)

C) HSURF, un composant testé à l'instant t=0

Les évalutations de systèmes à haute sûreté de fonctionnement sont basées sur l'hypothèse importante suivante :

"Le composant est sans défaillance à l'instant T=0"

Il est donc nécessaire de réaliser en fin de fabrication (T=0) un ensemble de tests complets et efficaces. Or, par manque de stratégie favorisant leur réalisation, et vu leur longueur et leur complexité, ceux-ci se trouvent souvent abrégés au maximum. Il devient alors délicat de considérer l'hypothèse comme vérifiée.

L'implantation au sein de HSURF d'un automate de test spécialisé permet de réaliser la vérification d'un ensemble minimum de points critiques (Réalisation d'un test hors-ligne dans un environnement de test spécifique). La validation de ce noyau autorise alors l'utilisation des différents dispositifs déja prévus pour le test en oisiveté et en ligne permettant ainsi la vérification de l'ensemble du circuit.

Le fait d'avoir intégré des opérateurs spécialement étudiés en vue d'un test facile et rapide rend le temps de test en fin de fabrication acceptable, et permet de valider ce circuit pour des applications à haute sûreté de fonctionnement.

II- SCHEMA GENERAL

DU MICROPROCESSEUR HSURF

II - SCHEMA GENERAL DU MICROPROCESSEUR

La structure du microprocesseur HSURF découle des exigences énoncées précédemment.

II.1 - ORGANISATION MATERIELLE

La partie opérative est organisée autour d'une structure à 2 bus de 16 bits auxquels ont accès 30 registres indépendants et accessibles individuellement. Ces registres se divisent en deux groupes :

- 16 registres accessibles en mode normal, c'est-à-dire avec les instructions classiques de chargement, rangement, opérations arithmétiques, logiques...

  Il y a : 8 registres généraux
  8 registres spécialisés

- 14 registres accessibles en mode test, c'est-à-dire avec des instructions spécialisées de chargement et de rangement.

Les opérateurs arithmétiques et logiques sont des opérateurs spéciaux facilement testables.

L'ensemble des informations circulant sur les bus peuvent être signées à des fins d'observation.

II.2 - ORGANISATION LOGICIELLE

Le jeu d'instructions du microprocesseur HSURF possède les opérations de base de tout microprocesseur classique de 16 bits auxquelles sont ajoutées de nombreuses possibilités de traitement pouvant porter aussi bien sur des mots de 16 bits que sur des parties de mots ou des bits isolés d'un mot (opérations logiques et transfert sur seulement des parties de mots par l'utilisation de masques).

De plus dans des applications temps réel destinées à contrôler l'évolution d'un ensemble de variables (les variables pouvant être des informations issues de capteurs), on agit souvent par comparaison entre l'ensemble des états nouveaux et des états anciens de leurs valeurs. Ainsi, dans l'application destinée aux postes d'aiguillages informatisés, les informations issues des capteurs (état de position d'un feu de signalisation, indicateur de présence d'un train sur une portion de voie,...) sont rangées dans des matrices. Le processus consiste ensuite à effectuer des modifications ou des comparaisons sur les éléments de ces matrices. En conséquence, le jeu d'instruction du microprocesseur HSURF offre la possibilité d'opérer directement sur des éléments de matrices bidimensionnelles.

Aux instructions précédentes, il faut ajouter un ensemble d'opérations spéciales destinées à faciliter le test du microprocesseur.

Organisation de la mémoire

La mémoire accessible par l'utilisateur est divisée en 16 pages contenant chacune 64k mots mémoire (65 536). Il faut donc 20 bits pour adresser un mot mémoire. Une adresse est composée :

- d'un numéro de page de 0 à 15 déterminant la page adressée (codé sur 4 bits).
- d'une adresse sur 16 bits déterminant le mot mémoire dans la page choisie.

III - JEU D'INSTRUCTION

DU MICROPROCESSEUR HSURF

III - JEU D'INSTRUCTION

Le microprocesseur HSURF offre un jeu d'instruction qui permet malgré sa simplicité de faciliter l'écriture d'un logiciel approprié aux applications haute sécurité.

Il comprend, outre les instructions classiques (arithmétiques, logiques, transfert, branchement, rupture de séquence), des instructions et des modes d'adressage adaptés aux besoins des applications de contrôle de processus.

- instructions avec masquage permettant de travailler sur des champs de bits.
- modes d'adressage spéciaux facilitant le travail direct sur des éléments de matrices.

De plus, les impératifs d'observabilité du processeur ont conduit à développer des instructions spécifiques pour le test.

III.1 - MODES D'ADRESSAGE

Seuls les modes d'adressage simples ont été retenus. Toutes les opérations à 2 opérandes s'effectuent suivant le format suivant :

DEST ← -- DEST op OPERANDE

Où DEST est l'un des 16 registres accessibles avec les instructions normales, et où OPERANDE peut être :

- un registre (pris dans le même ensemble que DEST)
- une constante immédiate
- un mot mémoire
- un élément de matrice

L'ensemble des différents modes d'adressage disponibles est le suivant :

Adressage inhérent : l'opérande est donné de façon implicite dans l'instruction.

Adressage relatif : l'opérande est un déplacement de 16 bits. Ce mode est utilisé seulement dans les instructions de branchement.

Adressage immédiat (I) : l'opérande est une constante.

Adressage direct (D) : l'adresse de l'opérande est donnée dans le mot suivant le code opération.

Adressage registre (R) : la zone adresse indique quel registre contient l'opérande.

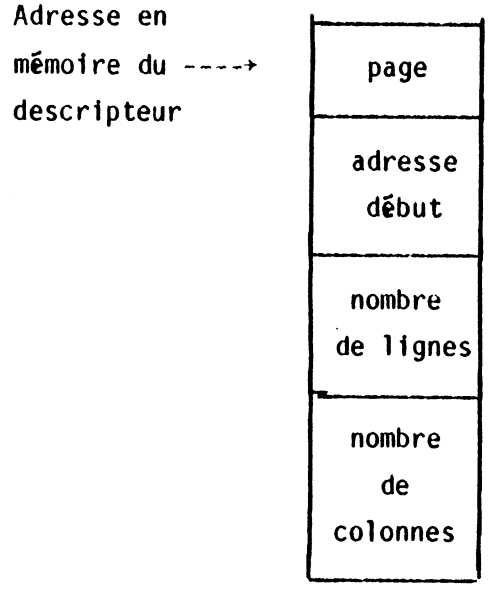
Adressage indirect par registre (IR) : un registre contient l'adresse de l'opérande.

• Modes d'adressage matrice :

Dans les applications de contrôle temps réel, la plupart des variables sont fonction de l'état de différents capteurs. Les valeurs de ces variables sont ensuite stockées dans différentes matrices (tableau de bits), l'une d'elle représentera par exemple l'état des capteurs à l'instant T, l'autre, l'état des capteurs à l'instant $T + \Delta T$.

Le traitement des différentes informations se résumant ensuite à de simples transformations ou comparaisons des différentes matrices. Des instructions spécifiques de traitement de tableaux booléens ont donc été implémentées dans le microprocesseur.

Les caractéristiques "statiques" des matrices sur lesquelles travaille le microprocesseur sont rangées en mémoire sous forme d'un enregistrement situé dans la même page que le programme d'application.



Une opération sur les éléments de matrice à la structure suivante :

OPMAT, Reg, Adr, Ligne, Colonne, (Rep-Masque)

où :

- Reg. = Registre général contenant le premier opérande, et où se trouvera le résultat.
- Adr. = Adresse du descripteur de la matrice
- Ligne = n° de la ligne où se trouve le deuxième opérande.
- Colonne = n° de la colonne où se trouve le deuxième opérande

- Rep-masque = n° du registre contenant le masque pour les opérations avec masquage.

Différents types d'adressage matrice ont été définis, chacun d'eux correspondant à une façon différente de passer les paramètres concernant le descripteur et les numéros de ligne et de colonne de l'élément de matrice invoqué par l'opération. Le descripteur d'adresse doit se trouver dans un des 2 registres spécialisés (RM0 ou RM1) ou être spécifié par adressage direct.

Le numéro de la ligne et de la colonne de l'élément utilisé par l'instruction peut être spécifié de façon immédiate ou par registre (on spécifie alors les numéros des registres qui contiennent les informations utiles).

Toutes les combinaisons possibles sont résumées dans le tableau suivant :

descripteur	n° ligne et colonne	symbole mnémonique
Direct	immédiat	DI
Registre	immédiat	RI
Direct	registre	DR
Registre	registre	RR

4 modes d'adressage supplémentaires ont été définis ; ils sont utilisés lors d'opérations entre un élément de matrice et une constante immédiate (Load et Store d'une constante immédiate dans une matrice ou comparaison d'un élément de matrice avec une constante immédiate). Ces modes d'adressage sont semblables à ceux définis précédemment ; on précise simplement un mot supplémentaire contenant la valeur immédiate. Leurs mnémoniques seront : DII, RII, DRI, RRI.

Une description détaillée des différents modes d'adressage du microprocesseur HSURF est donné en Annexe 1.

III.2 - JEU D'INSTRUCTIONS

III.2.1 - Instructions générales

Les tableaux suivants définissent les différents modes d'adressage pour les instructions arithmétiques, logiques et de transfert.

INSTRUCTIONS ARITHMETIQUES :

	MODE D'ADRESSAGE								
	I	D	R	IR	DI	RI	DR	RR	**
ADD	O	O	O	O	O	O	O	O	
ADDC	O	O	O	O	O	O	O	O	
SUB	O	O	O	O	O	O	O	O	
SUBC	O	O	O	O	O	O	O	O	
COMP	O	O	O	O	O	O	O	O	O
ASL	O	O	O	O	O	O	O	O	
ASR	O	O	O	O	O	O	O	O	
INC			O						
DEC			O						

INSTRUCTIONS LOGIQUES :

	MODE D'ADRESSAGE							
	I	D	R	IR	DI	RI	DR	RR
AND	O	O	O	O	O	O	O	O
OR	O	O	O	O	O	O	O	O
NAND	O	O	O	O	O	O	O	O
NOR	O	O	O	O	O	O	O	O
EOR	O	O	O	O	O	O	O	O
LSL	O	O	O	O	O	O	O	O
LSR	O	O	O	O	O	O	O	O
NOT			O					

INSTRUCTIONS DE TRANSFERT :

	MODE D'ADRESSAGE								
	I	D	R	IR	DI	RI	DR	RR	**
LOAD	O	O	O	O	O	O	O	O	
STORE		O	O	O	O	O	O	O	O
EXCH			O						

INSTRUCTIONS DE BRANCHEMENT, TRAITEMENT DE PILE :

	MODE D'ADRESSAGE							
	I	D	R	IR	DI	RI	DR	RR
JMP	O	O	O	O	O	O	O	O
JSR	O	O	O	O	O	O	O	O
RSR	Inhérent							
PUSH	O	O	O	O	O	O	O	O
PULL			O					
BR ★	Relatif							
CLC	Inhérent							
SEC	Inhérent							
NOP	Inhérent							

BR* : il existe deux types de branchements par bit du registre d'état RE.

III.2.2 - **Instructions spécialisées**

INSTRUCTIONS AVEC MASQUAGE :

Dans de nombreuses applications il est nécessaire de travailler seulement sur des bits isolés ou sur des portions de mots représentées par plusieurs champs de bits. Ce type d'opération est possible dans HSURF grâce aux instructions avec masquage.

Dans toutes les opérations du type :

DEST. ←-- DEST Op OPERANDE

Les seuls bits modifiés dans DEST sont les bits corespondants aux bits qui sont à "1" dans le registre masque.

Par exemple pour l'opération ET-MASQUE on a (Fig. 1) :

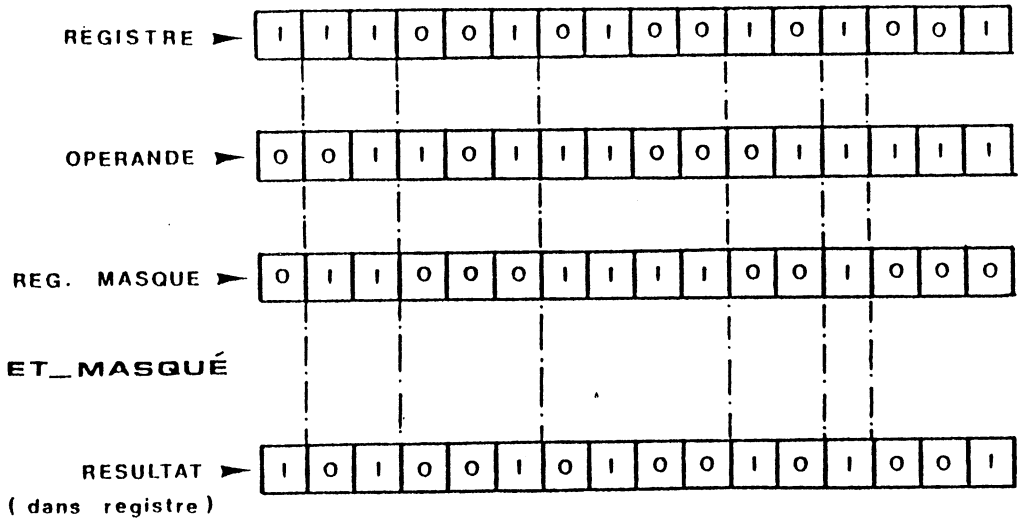


Figure 1

Deux drapeaux du registre d'état sont positionnés :

L = 1 Si tous les bits modifiés sont à "0"
H = 1 Si tous les bits modifiés sont à "1"

AND, OR, NAND, EOR, LOAD, STORE, COMP peuvent être utilisées avec l'option de masquage. Les modes d'adressage sont les mêmes qu'avec les instructions normales. Le masque est contenu dans l'un des 16 registres généraux, son numéro devant simplement être spécifié dans un mot à la fin du code instruction.

INSTRUCTION DE COMPARAISON DE DEUX MATRICES :

COMPARMAT Adr-enr1, Adr-enr2, Regcol, Regligne, Regmasque

où :

- Adr-enr1 est l'adresse de l'enregistrement contenant les caractéristiques statiques de la matrice n°1.
- Adr-enr2 est l'adresse de l'enregistrement contenant les caractéristiques statiques de la matrice n°2.
- Regcol est le numéro du registre qui contient le numéro de la colonne de l'élément à partir duquel il faut commencer la comparaison.
- Regligne est le numéro du registre qui contient le numéro de la ligne de l'élément à partir duquel il faut commencer la comparaison.
- Regmasque est le numéro du registre contenant le masque destiné à spécifier les bits ou les champs de bits du mot qu'il faut comparer.

L'instruction effectue une comparaison séquentielle de deux matrices à partir de l'élément spécifié MAT (i,j).

Résultat :

Un bit spécial du mot d'état indique si une différence a été détectée, et, dans le cas favorable, les registres Regligne et Regcol contiennent les coordonnées du premier élément différent.

Un redémarrage de l'instruction sans réinitialisation de ces registres permet de continuer la comparaison en séquence.

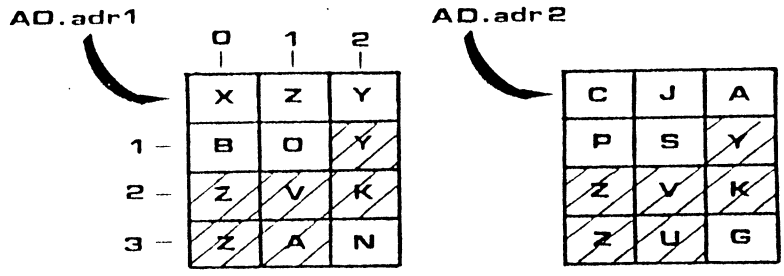
La figure 1 donne un exemple de l'exécution de cette instruction.

* Exemple d'exécution de l'instruction COMPARMAT (Fig. 2) :

COMPARMAT Adr1, ADr2, i, j, k

avec : contenu du registre n° i = 1
contenu du registre n° j = 1
contenu du registre masque n° k = "FFFF" (pas de masquage).

COMPARMAT adr1 adr2 i j k



1ier élément différent : MAT(3,1)

Résultat :

Le Bit E du registre d'état est mis à "1"
Le registre ligne n°i contient la valeur 3
Le rep. colonne n°j contient la valeur 1

Figure 2

INSTRUCTION DE RECHERCHE DANS UNE MATRICE :

Cette instruction a pour but de comparer séquentiellement une matrice avec le contenu d'un registre.

Le format de cette instruction est le suivant :

CHERCHMAT Adr-enr, Reg-col, Reg-ligne, Op-reg, Reg-masque

où :

Adr-enr est l'adresse de l'enregistrement contenant les caractéristiques statiques de la matrice.

Reg-col, Reg-ligne contiennent le numéro de la ligne et de la colonne du premier élément à partir duquel il faut commencer la comparaison.

Op-Reg est le registre opérande contenant la valeur à rechercher dans la matrice.

Reg-masque est le registre contenant le masque qui permet d'indiquer les bits sur lesquels la recherche (comparaison) sera effective.

Les résultats de cette instruction sont du même type que pour l'instruction COMPARMAT, la seule différence étant qu'après exécution les valeurs contenues dans les registres ligne et colonne sont les coordonnées dans la matrice du premier mot identique au contenu du registre Op-Reg.

INSTRUCTIONS DE TEST :

L'impératif principal du microprocesseur est d'être facilement testable, or, une telle nécessité impose :

- un accès direct à chaque point de mémorisation interne du microprocesseur.
- un accès direct aux entrées / sorties des différents opérateurs.

Au niveau des instructions, la solution a été de définir des instructions spécifiques de test devant permettre de réaliser les accès précédents.
Ces instructions sont les suivantes :

- LRS = Load Registre interne (accès direct à un registre)
- SRS = Store Registre interne

Ces deux instructions permettent d'accéder en lecture et en écriture à tous les registres internes du microprocesseur (Registres généraux et registres spécialisés tels que Compteur ordinal, Pointeur de pile, Registres tampon d'entrée de l'ual, du multiplieur,...)

- RAZ = Remise à zéro du registre de signature

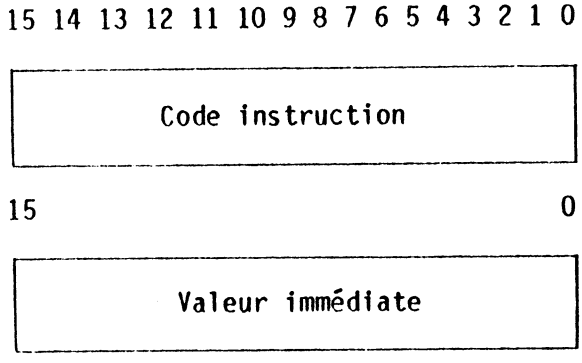
- AJS = Ajustement du contenu du registre de signature SIGN de la partie opérative.

Cette instruction permet de réaliser l'opération suivante :

Reg-SIGN ←-- Reg-SIGN ⊕ valeur immédiate.

sans créer de modification du contenu du registre d'état. Son but est d'ajuster le contenu du registre de signature à une valeur prédéterminée en fonction de sa valeur précédente (résultat d'une séquence de signature) et de

la valeur immédiate se trouvant dans le mot suivant le code instruction. On verra son utilité en détail dans la partie VI concernant le test en ligne du microprocesseur HSURF.
Sa structure est la suivante :



- **AJST** = Ajustement du contenu du registre de signature de la partie opérative suivi d'un test du contenu de ce registre avec la valeur "tout à zéro". Cette instruction réalise la même opération que l'instruction précédente et génère une alarme si après modification, le contenu du registre de signature n'est pas nul (cf. partie VI).

- **TRC** = Test des registres du contrôleur. Cette instruction permet d'écrire et de lire dans les registres internes de la partie contrôle dont l'accès en fonctionnement normal du microprocesseur est impossible. En effet, les registres internes au micro-contrôleur sont des registres qui sont utilisés de façon continue par la partie contrôle, ils contiennent des informations vitales sans lesquelles le séquencement et l'exécution des instructions est impossible. Cette instruction permet donc d'effectuer une séquence de lecture / écriture dans ces registres. Le format de cette instruction sera le suivant :

TRC Reg-Cont., Ad-Source, Ad-dest

où :

Reg-Cont : registre du contrôleur que l'on désire tester
Ad-Source : adresse en mémoire de la valeur que l'on veut stocker dans le registre à tester (lecture).

Ad-dest : adresse de la mémoire où l'on va aller stocker le résultat de la lecture du registre du contrôleur sous test (écriture).

L'exécution de l'instruction aura pour effet de valider un automate spécialisé qui effectuera la séquence d'opérations suivantes :

i) - Sauvegarde du contenu du registre du contrôleur à tester.
ii) - Opération d'écriture :
MEM ⌊Ad-Source⌋ ←-- Reg-Cont sous test
iii) - Opération de lecture :
⌊Reg-Cont sous test⌋ --→ MEM ⌊Ad-Dest⌋
iiii) - Restauration du contenu du registre sous test.

Après l'exécution de cette instruction, il est aisé de comparer le contenu de MEM ⌊Ad-Source⌋ avec celui de MEM ⌊Ad-Dest⌋.

- **TUAL** = test de l'UAL.

L'opération réalisée est la suivante :

R : = R op OP

Cette instruction donne les accès nécessaires au test exhaustif par tranche de l'Unité Arithmétique et Logique. Elle permet de tester les 16 configurations possibles d'instructions en entrée de chaque tranche de l'UAL.

Une description détaillée de chacune des instructions précédentes (codages, modification du registre d'état...) ainsi que l'affectation des numéros des différents registres permettant leur codage sont données en annexe 1.

IV - PARTIE OPERATIVE

DU MICROPROCESSEUR HSURF

IV - INTRODUCTION

Cette partie opérative est essentiellement une partie opérative régulière, facilement testable en ligne et hors ligne.

Afin de faciliter ces opérations sa réalisation répond aux exigences suivantes :

- structure répétitive en tranches de bits
- accès possible de façon indépendante à chaque registre interne
- possibilité d'appliquer des vecteurs de test à l'entrée des différents opérateurs internes par accès direct à leurs points d'entrée, et possibilité de lire le résultat du test par accès direct aux points de sortie.

Son architecture est organisée autour d'une structure à 2 bus (cf. Fig.1).

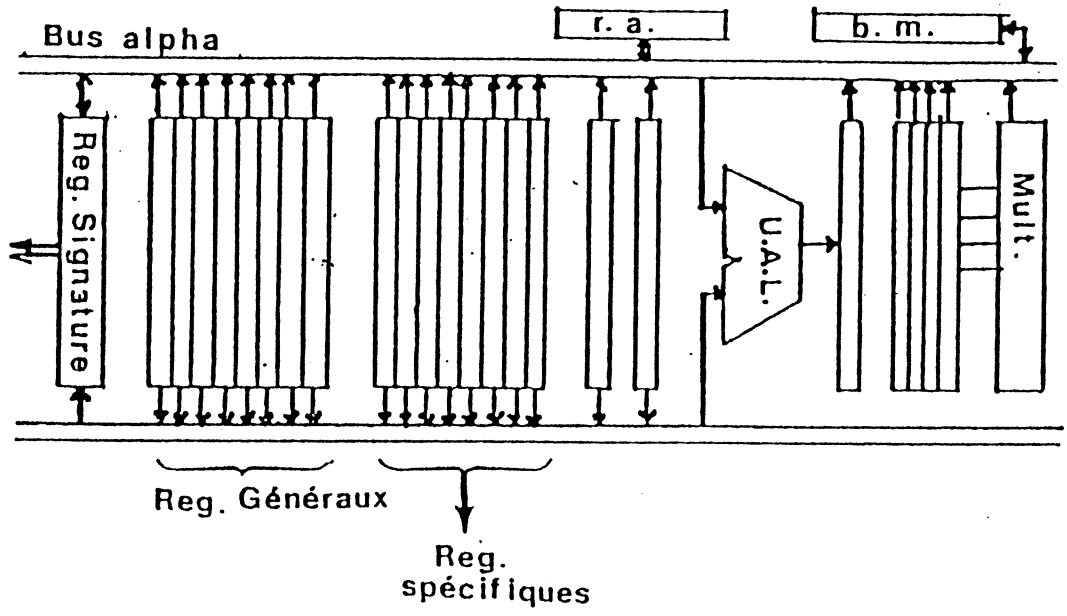


Figure 1

IV.1 - ELEMENTS GENERAUX DE LA PARTIE OPERATIVE

La partie opérative du microprocesseur est construite autour de 2 bus ALPHA et BETA d'une largeur de 16 bits. Tous les registres ont accès à l'un ou l'autre de ces bus de façon directe et indépendante.

IV.1.1 - Registres :

L'ensemble de ces registres est divisé en deux classes :

i) registres accessibles avec les instructions normales de lecture/écriture

* $R_0 \ldots R_7$: registres généraux
* CO : compteur ordinal
* PP : pointeur de pile
* RE : registre d'état
* RM_0, RM_1 : registres d'adresse des descripteurs de matrice
* RTIS, POL : registres du dispositf de signature
* RP_1: registre page

ii) registres accessibles de façon directe (lecture/écriture) avec les intructions de test

* RP_2 : registre de page (pour COMPARMAT)
* S : buffer d'adresse
* TM : buffer mémoire
* T_1, T_2: registres internes
* RI : registre d'instructions
* E_1, E_2 : buffers d'entrée de l'UAL
* A, B, C, D : buffers d'entrée du multiplieur
* RMA : registre masque
* SIGN : registre de signature

IV.1.2 - Les opérateurs de la partie opérative :

La réalisation d'opérateurs spéciaux tels que unité arithmétique et logique, multiplieur réalisant diverses fonctions plus ou moins complexes

pose des problèmes particuliers lorsqu'ils sont destinés à des systèmes dont la finalité principale est de disposer d'une grande facilité de test.

Dans des applications où plusieurs unités fonctionnent en paralèlle (Duplex, Triplex,...) et pour lesquelles une comparaison continue des sorties est réalisée (test en ligne continu), on aura déjà une très bonne couverture de test pour les opérateurs utilisés très fréquemment. Ainsi, une U.A.L. qui réalise des millions d'opérations par seconde et dont les résultats sont continuellement comparés pourra être considérée comme parfaitement testée pendant le déroulement de l'application ; malheureusement, ce type de stratégie paraissant comme une solution idéale ne peut être considéré comme viable.

En effet, dans la pratique, l'utilisation des différentes ressources du système est fonction de la structure du programme d'application, on ne sera donc jamais à l'abri du fait que pendant certaines périodes, des organes internes au microprocesseur (opérateurs de la partie-opérative) soient sous-utilisés. Il est donc indispensable de compléter notre stratégie de test en ligne continu par l'utilisation massive de programmes de test périodiques qui seront exécutés pendant les périodes d'oisiveté du programme d'application (test en ligne discontinu).

La mise en oeuvre de cette politique sera grandement facilitée si l'on bénéficie de modules dont le test complet est réalisable de façon très rapide.

La réalisation de tels opérateurs demande de prendre en compte les impératifs de testabilité dès l'étape de conception ; à partir d'une structure de base, on peut estimer la complexité de son test et, le cas échéant, étudier les modifications structurelles qui permettraient de réduire cette complexité. Notre politique consiste donc en un premier temps à définir pour chacun des opérateurs une stratégie de test complète puis à donner une séquence d'instructions détaillée permettant de réaliser un test le plus complet possible de chacune des cellules de cet opérateur.

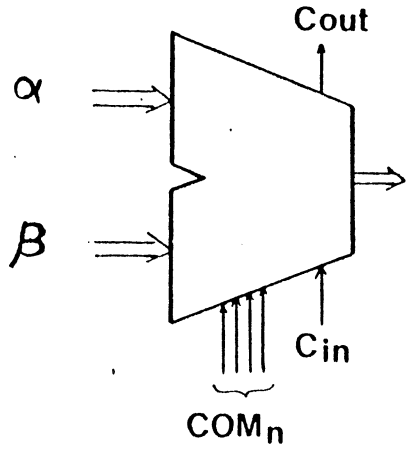
Afin de s'affranchir au maximum du problème délicat de l'élaboration d'hypothèses de pannes, nous essaierons de rendre possible pour chacun des opérateurs un test exhaustif ; Le test exhaustif d'un circuit combinatoire à n entrées consiste à appliquer au circuit les 2^n vecteurs d'entrée possibles.

Le test exhaustif détecte donc toutes les pannes qui transforment ce circuit en toute autre fonction combinatoire, il détecte toutes les pannes irredondantes de collage classiques simples et multiples ainsi que la plupart des pannes de court-circuit et de coupures étudiées dans [TIM 83] et [WAD 78] qui peuvent se modéliser comme des pannes de collage classique. Seules les fautes non classiques [WAD 78], telles que les "Stuck-open" transformant le circuit combinatoire en circuit séquentiel pourront ne pas être détectées. Ce problème pouvant être résolu par l'application de séquences de test particulières [CHA 83] [BAS 85].

IV.1.2.1 - L'Unité Arithmétique et Logique :

IV.1.2.1.1 - Fonction :

L'U.A.L. est réalisée par un opérateur combinatoire défini comme suit (Figure 2) :



α et β : entrées des opérandes

S : sortie du résultat

COM_n : commande de sélection de l'opération.

Figure 2

Les opérations réalisées par ce circuit sont les suivantes :

Mnémonique	Entrées	Entrées	Sortie S'
ADD	A	B	A plus B
ADDC	A	B	A plus B plus Retenue
SUB	A	B	A moins B
SUBC	A	B	A moins B moins Retenue
INC	A	X	A plus 1
DEC	A	X	A moins 1
NOT	A	X	$\bar{A}$
AND	A	B	A B
OR	A	B	A + B
NAND	A	B	$\overline{A\ B}$
NOR	A	B	$\overline{A + B}$
EOR	A	B	A + B

Où :

Retenue = Retenue entrante.

X signifie que le résultat S ne dépend pas de l'entrée β.

Il est à noter que les opérations de décalage arithmétique et logique ne sont pas réalisées par l'U.A.L. mais par l'un des 2 registres d'entrées de l'U.A.L organisé en registre à décalage.

Quatre drapeaux indicateurs d'état sont générés par l'U.A.L :

N : N = 1 le résultat est négatif
N = 0 le résultat est positif ou nul.

Z : Z = 1 le résultat est nul
Z = 0 le résultat est non nul

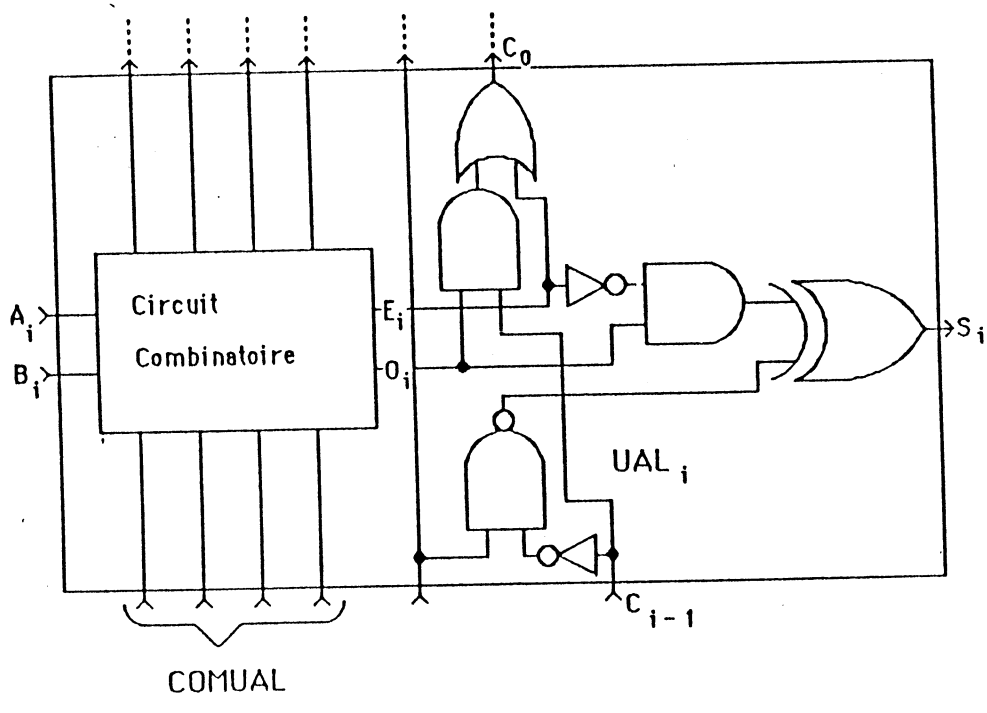
C : C=1 retenue sortante
C = 0 pas de retenue sortante

V : V = 1 l'opération arithmétique a provoqué un débordement
V = 0 pas de débordement

IV.1.2.1.2 - Structure de l'U.A.L.

L'unité arithmétique et logique est organisée autour d'une structure en tranche de bits (bit-slice), c'est-à-dire que l'opérateur 16 bits est réalisé par juxtaposition de 16 cellules de 1 bit.

La structure de base d'une cellule de 1 bit est la suivante (Figure 3) :



A_{0-15} : mot A
A_{0-15} : mot B
S_{0-15} : sortie del'U.A.L.
COMUAL : commande de l'opération réalisée par l'U.A.L

Figure 3

Le principe de l'opérateur est basé sur l'association d'un circuit de calcul permettant d'obtenir la sortie $S = f_1(E_i, O_i, C_{in})$ et la retenue sortante $Cout = f_2(E_i, O_i, C_{in})$, et d'un circuit combinatoire générant E_i et O_i en fonction des entrées A_i et B_i et du code de l'opération à réaliser.

IV.1.2.1.3 - Le test de l'U.A.L.

Si l'on considère l'opérateur U.A.L. de 16 bits dans son ensemble, nous avons la structure suivante (figure 4) :

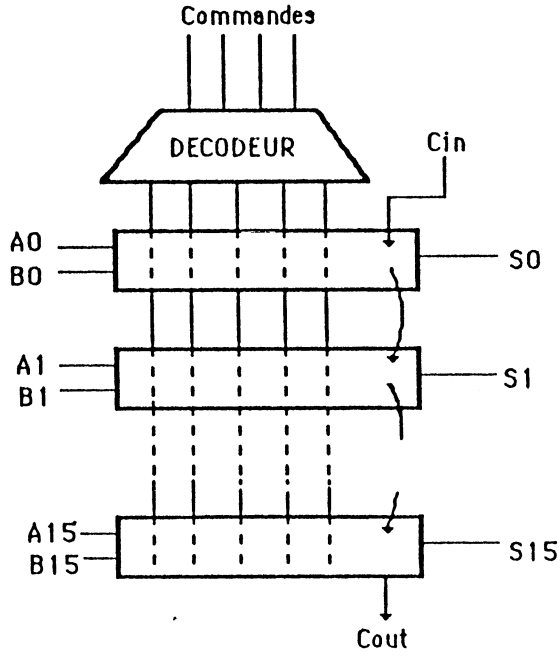


figure 4

L'impératif d'accessibilité totale aux opérateurs permet de définir l'ensemble des points du dispositif directement accessibles en entrée ("écriture") :

$$E = \{Commandes, A_{0-15}, B_{0-15}, Cin\}$$

et en sortie ("lecture") :

$$S = \{S_{0-15}, Cout\}$$

L'élaboration d'un test exhaustif sur un tel opérateur, s'avère inapplicable du fait du nombre beaucoup trop élevé d'entrées (37 entrées, ce qui conduit à 2^{37} vecteurs de test). Cependant, l'organisation en "tranches de bit" de cet élément permet de le considérer comme un "ILA" (Itérative Logic Array) et de développer une stratégie de test particulière à ce type de structure.

Les études s'intéressant au test et à la testabilité des ILAs sont nombreuses, [DIA 76], [PAR 81], [FRA 73]. Nous allons en rappeler les résultats afin de déterminer la politique de test de notre U.A.L. et le cas échéant les modifications à apporter à sa structure pour pouvoir mettre en oeuvre notre politique.

IV.1.2.1.4 - Le test des ILAs

IV.1.2.1.4.1 - Définition

Un ILA (iterative logic array) est défini comme un réseau linéaire de N cellules (fig 5) dont les entrées X_0 et Y_i sont directement contrôlables et dont les sorties $\hat{Y}_i$ et $\hat{X}$ sont directement observables

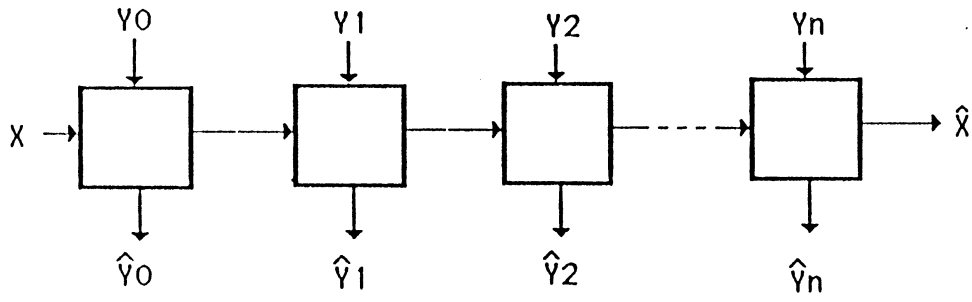


Figure 5

IV.1.2.1.4.2 - Modèle de panne :

Soit M un module combinatoire ou séquentiel synchrone dans un circuit C sous test. Soit z une fonction réalisée par M et soit s le nombre des états internes de M, si M est combinatoire alors s = 1.

Un mauvais fonctionnement F de M sera appelé panne de M si F change de façon permanente M en un module M^f qui réalise z^f avec $z^f \neq z$ et tel que le nombre d'états s^f de M^f n'est pas plus grand que s. Ainsi, une panne dans un module combinatoire peut affecter de façon arbitraire la table de vérité relative au module, mais ne peut pas le transformer en un circuit séquentiel. Pour détecter une telle panne, il est nécessaire et suffisant d'appliquer à ce module de n entrées les 2^n vecteurs d'entrées possibles. Ce modèle de panne, recouvre en fait l'ensemble des types de panne succeptible d'être décelées par une séquence de test exhaustif.

Il est aussi fait comme hypothèse, que seulement un module dans le circuit C est fautif en même temps. Cette hypothèse restrictive de "simple panne" sera justifiée si C est testé relativement souvent. Ceci affirme d'autant plus la nécessité de réaliser des tests fréquents pour chacun des modules du circuit, donc de disposer de modules "facilement" testables. L'élaboration de séquences de test couvrant les pannes multiples (plusieurs cellules défaillantes au même instant) est cependant envisageable [CHE85].

IV.1.2.1.4.3 - Stratégie de test des ILAs

Pour pouvoir réaliser le test d'un réseau logique itératif suivant les hypothèses précédentes, il faut pouvoir satisfaire aux deux conditions suivantes :

i) Il est possible d'appliquer à chacune des cellules de l'ILA une séquence de test exhaustif sur chacune de ces entrées (soit les 2^n valeurs possibles si la cellule a n entrées) quelle que soit sa position dans le réseau.

ii) Il est possible de propager le signal de sortie de n'importe quelle cellule sous test vers l'une des sorties primaires du réseau. ($\hat{Y}_i$ ou $\hat{X}$)

• Définitions :

Soit une cellule i d'un réseau itératif, cette cellule reçoit 2 types d'entrées :

- Un ensemble d'entrées "horizontales"

$$X_i = \{X_{ij},\ j\ (1,\ \ldots \text{nb. entrées})\}$$

- Un ensembles d'entrées verticales :

$$Y_i = \{Y_{ij'},\ j'\ (1,\ \ldots \text{nb. entrées})\}$$

Chaque entrée X_{ij} peut prendre une valeur dans l'ensemble $\widetilde{X}_{ij} = \{X_{ijk},\ k = 1,\ \ldots\}$ (pour un circuit fonctionnant en logique binaire, k pourra prendre 2 valeurs correspondant à l'état "0" et à l'état "1"). De même, sur chaque entrée $Y_{ij'}$, on peut avoir une valeur prise dans $\widetilde{Y}_{ij'} = \{Y_{ij'k'},\ k' = 1,\ \ldots\}$.

Une cellule i de ce type peut être caractérisée par une table d'états ayant une ligne pour chaque valeur de $\widetilde{X}_{ij}$ (état) et une colonne pour chaque valeur $\widetilde{Y}_{ij'}$. Le contenu de cette table représente la valeur des fonctions $\mathcal{X}(X_{ij},\ Y_{ij'})$ et $\mathcal{Y}(X_{ij},\ Y_{ij'})$ qui sont respectivement la fonction "état suivant" et la fonction "sortie".

A cette table d'états on peut associer un graphe ayant un état e_{ijk} pour chaque valeur X_{ijk} de $\widetilde{X}_{ij}$ et pour lequel une transition $T : X_{ijk1} \xrightarrow{Y_{ij'k'}} X_{ijk2} = \mathcal{X}(X_{ijk1},\ Y_{ij'k'})$ signifie qu'après l'entrée $(\widetilde{X}_{ij},\ \widetilde{Y}_{ij'}) = (X_{ijk},\ Y_{ij'k'})$, la sortie j de la cellule i (état suivant) a pour valeur X_{ijk2}.

Nous pouvons illustrer ceci par l'exemple suivant :

- L'I.L.A. est composé de cellules fonctionnant en logique binaire ($\widetilde{X}_{ij} = \{"0",\ "1"\}$, $\widetilde{Y}_{ij'} = \{"0",\ "1"\}$) et pour lesquelles les ensembles d'entrées X_i et Y_i sont des singletons (une seule propagation horizontale

et une seule entrée verticale par cellule).

Nous avons :

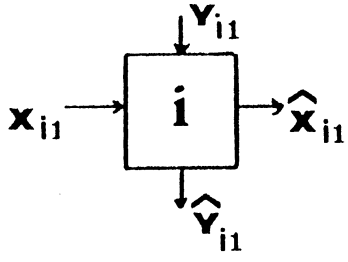
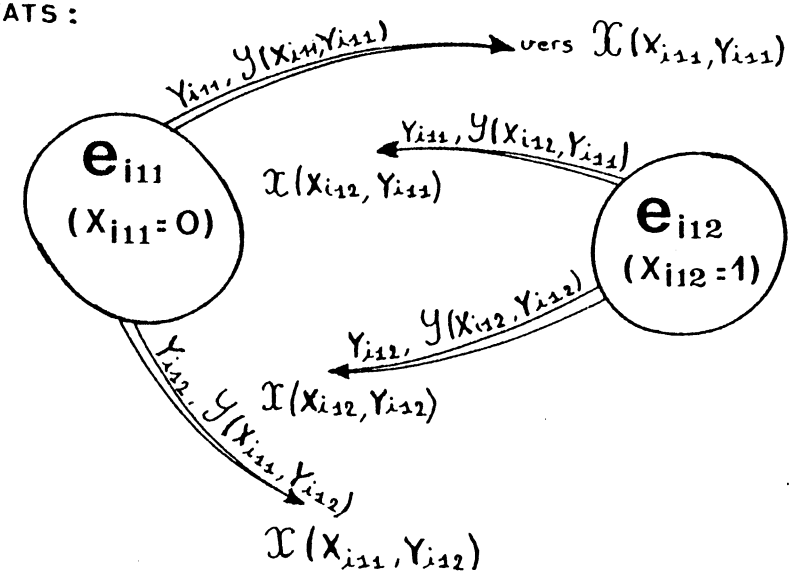


TABLE D'ETATS :

x_{i1} \ Y_{i1}	0	1
0	$\mathcal{X}(0,0);\mathcal{Y}(0,0)$	$\mathcal{X}(0,1);\mathcal{Y}(0,1)$
1	$\mathcal{X}(1,0);\mathcal{Y}(1,0)$	$\mathcal{X}(1,1);\mathcal{Y}(1,1)$

GRAPHE D'ETATS :



- Remarque :

  On peut par l'examen du graphe d'état (ou de la table d'état) associé à une cellule de base, déterminer si la condition i) relative au test des ILA's peut être vérifiée.

- Simplification des notations :

  Nous nous interressons dans le cadre de cette étude uniquement à des

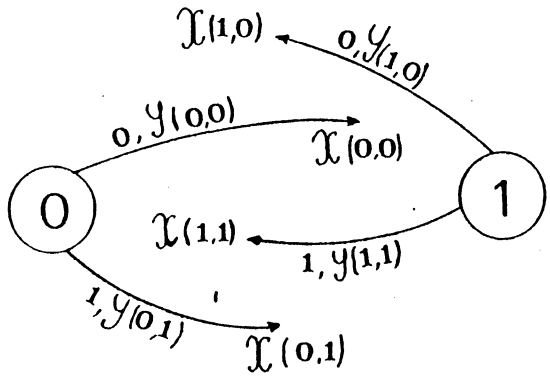
ILA's fonctionnant en logique binaire, nous aurons donc $\widetilde{X}_{ij}$ = {"0","1"} et $\widetilde{Y}_{ij'}$ = {"0","1"}. De plus, nous prendrons dans la suite des exemples de cellules n'ayant qu'une entrée suivant X (X_{ij}= X_{i1}= X) et une seule entrée suivant Y ($Y_{ij'}$= Y_{i1}= Y). Les états du graphe seront directement notés par les valeurs X_{ijk} associées (l'état e_{i11} associé au fait que X_{i11}= "0" sera noté 0, l'état e_{i12} associé à X_{i12}= "1" sera noté 1), de même les transitions $Y_{ij'k'}$ et les sorties $\mathcal{Y}(X_{ijk}, Y_{ij'k'})$ seront notées directement par leurs valeurs.

L'exemple précédant donnera donc les résultats suivants :

TABLE D'ETATS :

X \ Y	0	1
0	$\mathcal{X}(0,0); \mathcal{Y}(0,0)$	$\mathcal{X}(0,1); \mathcal{Y}(0,1)$
1	$\mathcal{X}(1,0); \mathcal{Y}(1,0)$	$\mathcal{X}(1,1); \mathcal{Y}(1,1)$

GRAPHE ASSOCIE :



- Etude des graphes associés aux cellules

Prenons par exemple la table et le gaphe d'état de la cellule hypothétique C1 représentée ci après (Figure 6) :

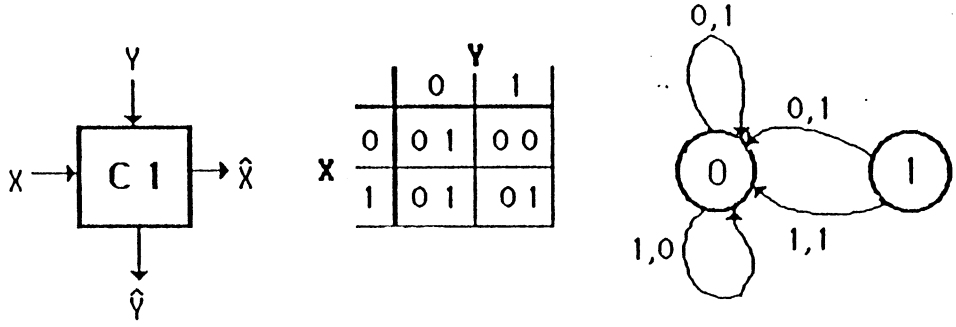


Figure 6

Avec un tel exemple, la condition i ne pourra pas être réalisée, en effet, il apparait que la cellule C ne peut générer que la valeur 0 sur sa sortie X, il sera donc impossible d'amener l'un des vecteurs de test (X=0, Y=0) ou (X=0, Y=1) sur les entrées de n'importe quelle cellule du réseau logique. Ceci se traduit au niveau du graphe d'état par le fait qu'il existe aucune

transition ayant pour destination l'état X = 1.

Soit maintenant la cellule C2 (Figure 7) :

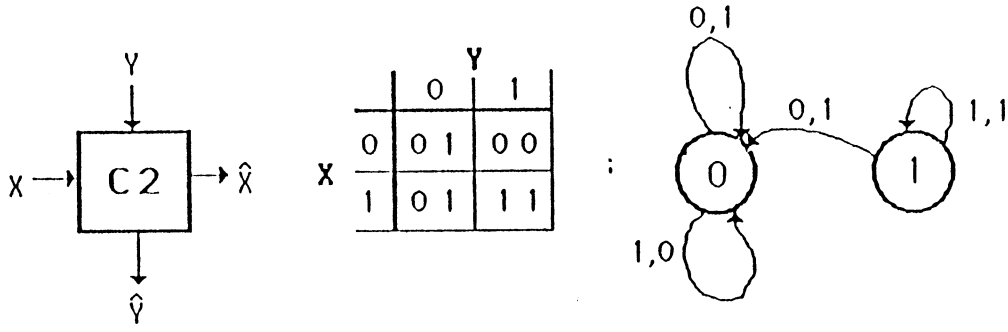
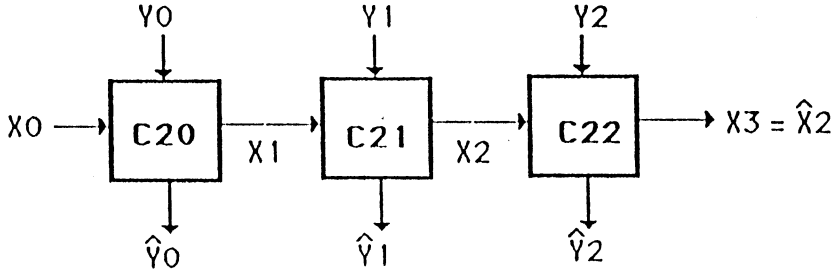


Figure 7

Nous voyons ici qu'il existe une transition arrivant sur l'état X = 1, nous pourrons donc vérifier la condition i relative au test des ILA.

Les vecteurs de test nécessaires pour tester par exemple un réseau de 3 cellules du type C2 seront les suivants (Figure 8) :



	X_0	Y_0	X_1	Y_1	X_2	Y_2	X_3
Test complet de C_{20}	0	0 → 0	0 → 0		0 → 0		
	0	1 → 0	1 → 0		1 → 0		
	1	0 → 0	ϕ → 0		ϕ → 0		
	1	1 → 1	0 → 0		ϕ → 0		
Vecteur d'entrée nécessaire sur la cellule C_{20} pour générer le test X_1=1 et Y_1=1 sur C_{21}	1	1 →	1	1	→ 1	0 → 0	
Vecteurs nécessaires sur C_{20} et C_{21} pour pouvoir faire un test exhaustif de C_{22}	1	1 →	1	1	→ 1	1 → 1	

Figure 8

Le test exhaustif par cellule du réseau peut donc être effectué, le nombre de vecteurs de test nécessaires est cependant fonction du nombre de cellules dont est composé l'ILA ; ceci pouvant devenir rapidement un inconvénient si le nombre d'entrées de la cellule est important.

Exemple 3 (Figure 9) :

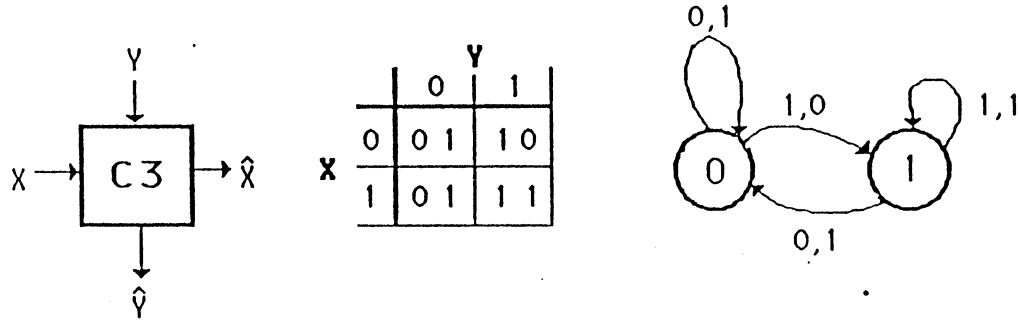


Figure 9

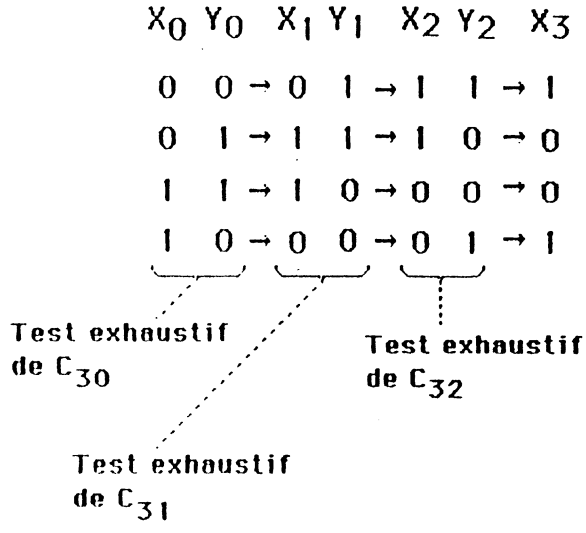
On vérifie ici aussi, que le réseau de cellules du type C3 pourra également être testé.

Le graphe précédent présente cependant la particularité supplémentaire suivante :

Pour chaque transition $t = X_{ijk1} \xrightarrow{Y_{ij'k'}} X_{ijk2}$ il existe une séquence d'entrée permettant de repasser de l'état X_{ijk2} dans l'état X_{ijk1}. Un tel graphe est appelé RT-graphe (Repeatable Transition) dans [SRI 81] de plus, s'il est possible de parcourir tous les états et les arcs de ce graphe sans repasser deux fois par le même arc, l'I.L.A. dont la cellule de base correspond à un tel graphe est testable avec un nombre de vecteurs indépendant du nombre de cellules du réseau. (On dira alors le réseau C-testable).

L'ensemble des vecteurs de test d'un tel réseau peut être obtenu très simplement en partant d'un état e_i du graphe , et en parcourant toutes les transitions jusqu'à revenir au même état e_i du graphe d'état de la fonction.

Soit par exemple à tester un réseau de trois cellules du type C3, nous aurons :



La suite des valeurs consécutives prises par y_i est celle des valeurs prises par y_{i-1} après une rotation de une position vers le haut, la génération des vecteurs de test s'averera donc très simple.

Remarque : un graphe tel que le suivant peut aussi être considéré comme un RT graphe (Figure 10) :

Figure 10

IV.1.2.1.5 - Application à l'Unité Arithmétique et Logique :

Nous considérons que chaque cellule de notre UAL est constituée par l'association d'une cellule de base et du décodeur. Nous remarquons que plusieurs lignes de commandes (au nombre de 4) sont communes à toutes les cellules. Pour simplifier l'étude il est commode de considérer l'opérateur comme plusieurs ILAs (ici 2^4 = 16) réalisant chacun une fonction bien définie.

Nous sommes ainsi en face d'ILAs simples à une dimension avec un accès direct à la sortie de chaque cellule. L'étude de chacune des fonctions réalisables par l'UAL fait apparaître que pour plusieurs opérations, les graphes d'état associés ne sont pas RT-graphe d'où un nombre de vecteurs de test grand (car dépendant du nombre de cellules n = 16)

Nous donnons ci-après les résultats pour l'opération NOR et INC (Figures 11 et 12) :

NOR	Ai.Bi = 00	01	10	11
Cin = 0	01	00	00	00
Cin = 1	01	10	10	10

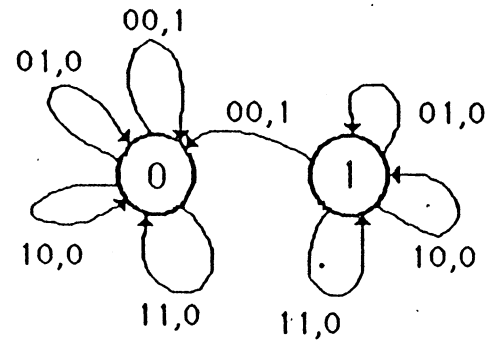


Figure 11

INC	Ai.Bi = 00	01	10	11
Cin = 0	00	00	01	01
Cin = 1	01	01	10	10

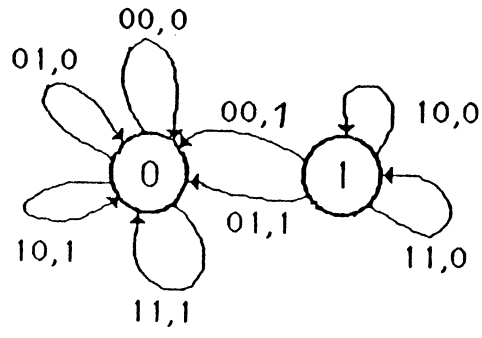


Figure 12

Ce problème se pose pour seulement 2 opérations arithmètiques INC et DEC alors qu'il se pose pour de nombreuses opérations logiques, or, il apparaît que pour les opérations logiques, l'entrée de retenue Cin n'a jamais (en temps de fonctionnement normal) d'influence sur le résultat de calcul. Une solution simple consiste donc à modifier la cellule de base de telle façon que Cin = Cout pour les opérations logiques.

Ainsi, pour toutes ces opérations nous aurons un graphe du type (Figure 13) :

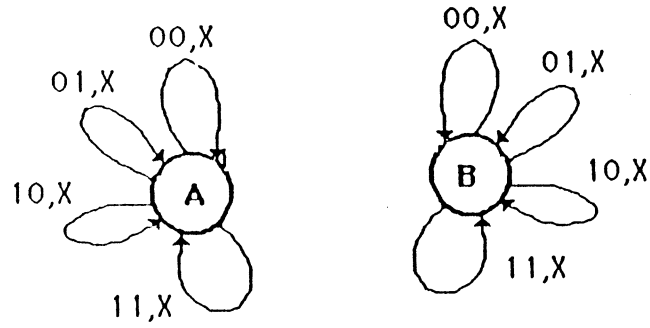


Figure 13

qui est RT graphe, d'où un nombre de vecteurs de test pour les opérations logiques indépendant du nombre de cellules de l'UAL.

Nous donnons dans la suite, le schéma détaillé d'une tranche de l'UAL, les valeurs des sorties pour toutes les entrées possibles et l'ensemble des vecteurs de test permettant une vérification complète d'une UAL de 16 bits.

IV.1.2.1.6 - Schéma logique :

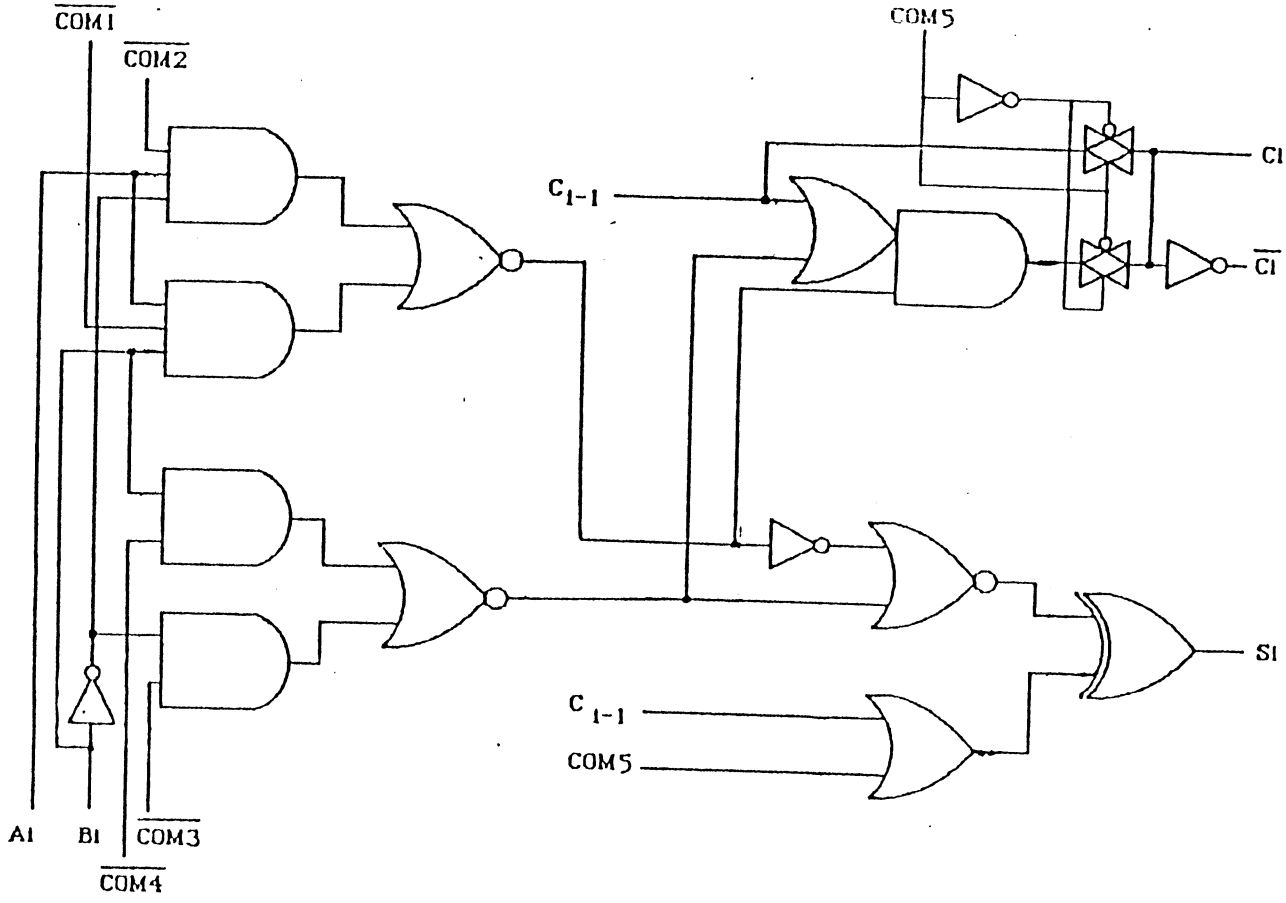


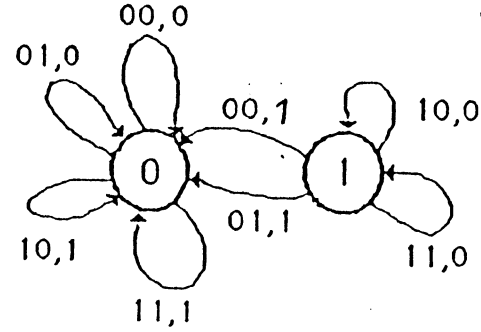
Figure 14
Schéma logique de 1 bit d'UAL (sans le décodeur)

CODE	C	A	B	E	O	S	C
INC	1	1	1	0	1	0	1
NOT	1	1	1	0	1	0	1
NOR	1	1	1	0	1	0	1
NAND	1	1	1	0	1	0	1
SUB	1	1	1	0	1	0	1
EOR	1	1	1	0	1	0	1
ADD	1	1	1	1	1	1	1
AND	1	1	1	1	1	1	1
OR	1	1	1	1	1	1	1
DEC	1	1	1	1	1	1	1
XXX	1	1	1	1	1	1	1
INC	1	1	0	0	1	0	1
NOT	1	1	0	0	1	0	1
NOR	1	1	0	0	1	0	1
NAND	1	1	0	1	1	1	1
SUB	1	1	0	1	1	1	1
EOR	1	1	0	1	1	1	1
ADD	1	1	0	0	1	0	1
AND	1	1	0	0	1	0	1
OR	1	1	0	1	1	1	1
DEC	1	1	0	1	1	1	1
XXX	1	1	0	1	1	1	1
INC	1	0	1	0	0	1	0
NOT	1	0	1	0	0	1	1
NOR	1	0	1	0	1	0	1
NAND	1	0	1	0	0	1	1
SUB	1	0	1	0	0	1	0
EOR	1	0	1	0	0	1	1
ADD	1	0	1	0	1	0	1
AND	1	0	1	0	1	0	1
OR	1	0	1	0	0	1	1
DEC	1	0	1	0	1	0	1
XXX	1	0	1	0	1	0	1
INC	1	0	0	0	0	1	0
NOT	1	0	0	0	0	1	1
NOR	1	0	0	0	0	1	1
NAND	1	0	0	0	0	1	1
SUB	1	0	0	0	1	0	1
EOR	1	0	0	0	1	0	1
ADD	1	0	0	0	0	1	0
AND	1	0	0	0	1	0	1
OR	1	0	0	0	1	0	1
DEC	1	0	0	0	1	0	1
XXX	1	0	0	0	1	0	1
INC	0	1	1	0	1	1	0
NOT	0	1	1	0	1	0	0
NOR	0	1	1	0	1	0	0
NAND	0	1	1	0	1	0	0
SUB	0	1	1	0	1	1	0
EOR	0	1	1	0	1	0	0
ADD	0	1	1	1	1	0	1
AND	0	1	1	1	1	1	0
OR	0	1	1	1	1	1	0
DEC	0	1	1	1	1	0	1
XXX	0	1	1	1	1	1	0
INC	0	1	0	0	1	1	0
NOT	0	1	0	0	1	0	0
NOR	0	1	0	0	1	0	0
NAND	0	1	0	1	1	1	0
SUB	0	1	0	1	1	0	1
EOR	0	1	0	1	1	1	0
ADD	0	1	0	0	1	1	0
AND	0	1	0	0	1	0	0
OR	0	1	0	1	1	1	0
DEC	0	1	0	1	1	0	1
XXX	0	1	0	1	1	1	0
INC	0	0	1	0	0	0	0
NOT	0	0	1	0	0	1	0
NOR	0	0	1	0	1	0	0
NAND	0	0	1	0	0	1	0
SUB	0	0	1	0	0	0	0
EOR	0	0	1	0	0	1	0
ADD	0	0	1	0	1	1	0
AND	0	0	1	0	1	0	0
OR	0	0	1	0	0	1	0
DEC	0	0	1	0	1	1	0
XXX	0	0	1	0	1	0	0
INC	0	0	0	0	0	0	0
NOT	0	0	0	0	0	1	0
NOR	0	0	0	0	0	1	0
NAND	0	0	0	0	0	1	0
SUB	0	0	0	0	1	1	0
EOR	0	0	0	0	1	0	0
ADD	0	0	0	0	0	0	0
AND	0	0	0	0	1	0	0
OR	0	0	0	0	1	0	0
DEC	0	0	0	0	1	1	0
XXX	0	0	0	0	1	0	0

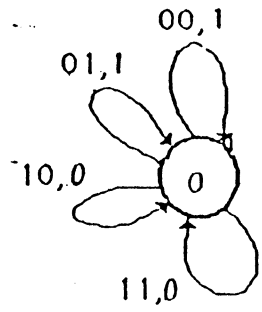
Table de fonctionnement de 1 bit de l'UAL

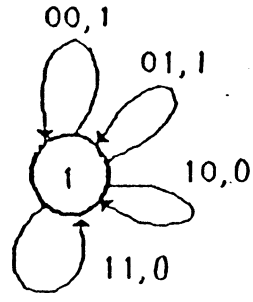
IV.1.2.1.7 - Graphes associés aux différentes opérations et vecteurs de test correspondants :

INC	00	01	10	11
0	00	00	01	01
1	01	01	10	10

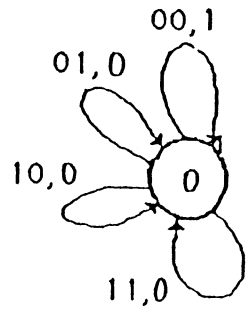


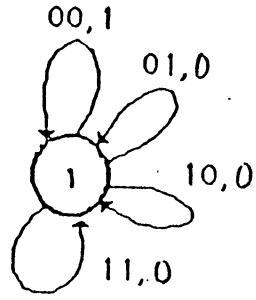
NOT	00	01	10	11
0	01	01	00	00
1	11	11	10	10



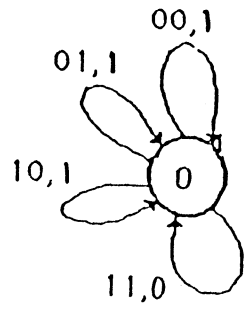


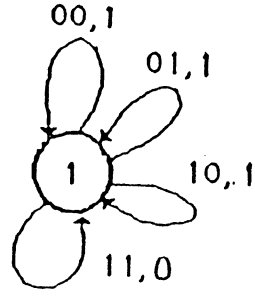
NOR	00	01	10	11
0	01	00	00	00
1	11	10	10	10



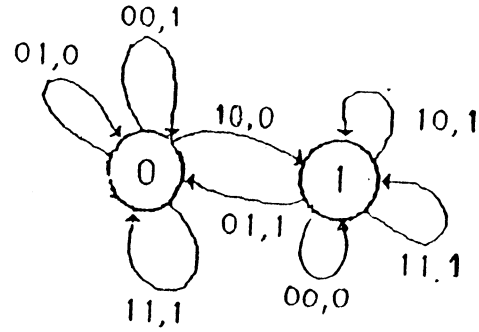


NAND	00	01	10	11
0	01	01	01	00
1	11	11	11	10

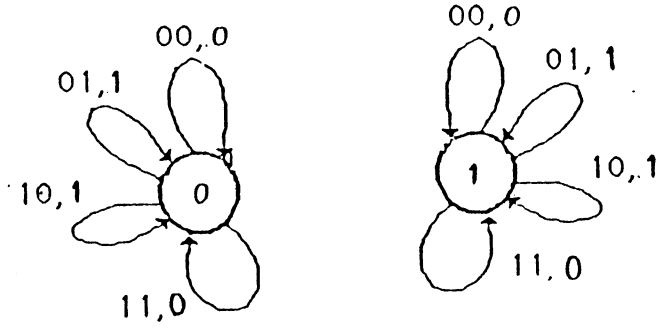




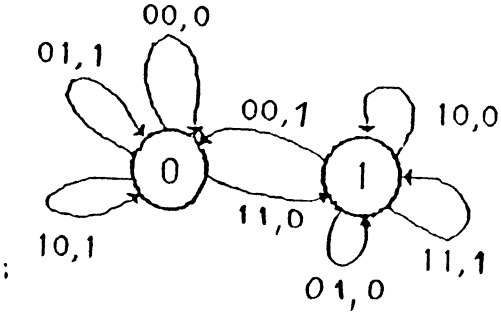
SUB	00	01	10	11
0	01	00	10	01
1	10	01	11	10



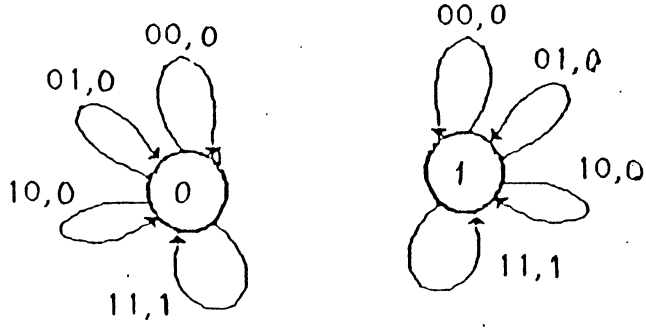
EOR	00	01	10	11
0	00	01	01	00
1	10	11	11	10



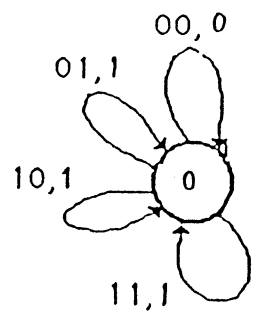
ADD	00	01	10	11
0	00	01	01	10
1	01	10	10	11

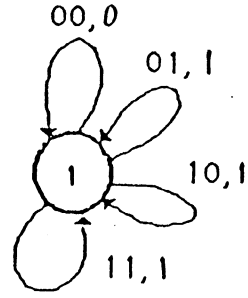


AND	00	01	10	11
0	00	00	00	01
1	10	10	10	11

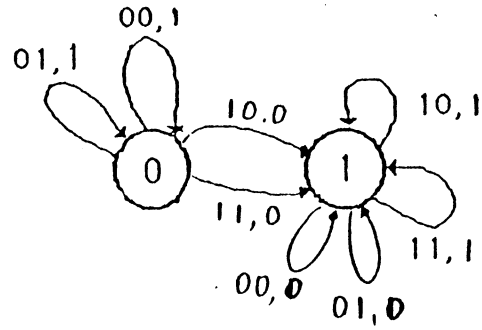


OR	00	01	10	11
0	00	01	01	01
1	10	11	11	11

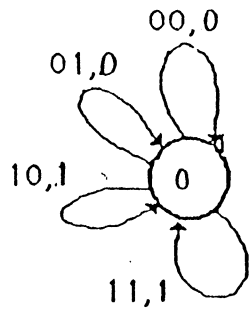


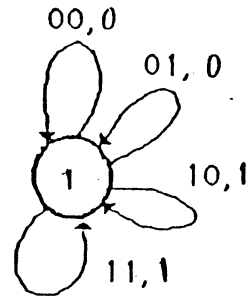


DEC	00	01	10	11
0	01	01	10	10
1	10	10	11	11



XXX	00	01	10	11
0	00	00	01	01
1	10	10	11	11





VECTEURS DE TEST :

ET_LOGIQUE :

Cin	Ai	Bi	Si	Cout
0	0000H	0000H	0000H	0
0	0000H	FFFFH	0000H	0
0	FFFFH	0000H	0000H	0
0	FFFFH	FFFFH	FFFFH	0
1	0000H	0000H	0000H	1
1	0000H	FFFFH	0000H	1
1	FFFFH	0000H	0000H	1
1	FFFFH	FFFFH	FFFFH	1

OU_LOGIQUE :

Cin	Ai	Bi	Si	Cout
0	0000H	0000H	0000H	0
0	0000H	FFFFH	FFFFH	0
0	FFFFH	0000H	FFFFH	0
0	FFFFH	FFFFH	FFFFH	0
1	0000H	0000H	0000H	1
1	0000H	FFFFH	FFFFH	1
1	FFFFH	0000H	FFFFH	1
1	FFFFH	FFFFH	FFFFH	1

XXX :

Cin	Ai	Bi	Si	Cout
0	0000H	0000H	0000H	0
0	0000H	FFFFH	0000H	0
0	FFFFH	0000H	FFFFH	0
0	FFFFH	FFFFH	FFFFH	0
1	0000H	0000H	0000H	1
1	0000H	FFFFH	0000H	1
1	FFFFH	0000H	FFFFH	1
1	FFFFH	FFFFH	FFFFH	1

NOT :

Cin	Ai	Bi	Si	Cout
0	0000H	0000H	FFFFH	0
0	0000H	FFFFH	FFFFH	0
0	FFFFH	0000H	0000H	0
0	FFFFH	FFFFH	0000H	0
1	0000H	0000H	FFFFH	1
1	0000H	FFFFH	FFFFH	1
1	FFFFH	0000H	0000H	1
1	FFFFH	FFFFH	0000H	1

NOR :

Cin	Ai	Bi	Si	Cout
0	0000H	0000H	FFFFH	0
0	0000H	FFFFH	0000H	0
0	FFFFH	0000H	0000H	0
0	FFFFH	FFFFH	0000H	0
1	0000H	0000H	FFFFH	1
1	0000H	FFFFH	0000H	1
1	FFFFH	0000H	0000H	1
1	FFFFH	FFFFH	0000H	1

NAND :

Cin	Ai	Bi	Si	Cout
0	0000H	0000H	FFFFH	0
0	0000H	FFFFH	FFFFH	0
0	FFFFH	0000H	FFFFH	0
0	FFFFH	FFFFH	0000H	0
1	0000H	0000H	FFFFH	1
1	0000H	FFFFH	FFFFH	1
1	FFFFH	0000H	FFFFH	1
1	FFFFH	FFFFH	0000H	1

EOR :

Cin	Ai	Bi	Si	Cout
0	0000H	0000H	0000H	0
0	0000H	FFFFH	FFFFH	0
0	FFFFH	0000H	FFFFH	0
0	FFFFH	FFFFH	FFFFH	0
1	0000H	0000H	0000H	1
1	0000H	FFFFH	FFFFH	1
1	FFFFH	0000H	FFFFH	1
1	FFFFH	FFFFH	FFFFH	1

ADD :

Cin	Ai	Bi	Si	Cout
0	6C6CH	5A5AH	C6C6H	0
0	3636H	2D2DH	6363H	0
0	1B1BH	9696H	B1B1H	0
0	8D8DH	4B4BH	D8D8H	0
1	C6C6H	A5A5H	6C6CH	1
1	6363H	D2D2H	3636H	1
1	B1B1H	6969H	1B1BH	1
1	D8D8H	B4B4H	8D8DH	1

SUB :

Cin	Ai	Bi	Si	Cout
0	6C6CH	C6C6H	A5A5H	0
0	3636H	6363H	D2D2H	0
0	1B1BH	B1B1H	6969H	0
0	8D8DH	D8D8H	B4B4H	0
1	C6C6H	6C6CH	5A5AH	1
1	6363H	3636H	2D2DH	1
1	B1B1H	1B1BH	9696H	1
1	D8D8H	8D8DH	4B4BH	1

INC :

Cin	Ai	Bi	Si	Cout
0	0000H	0000H	0001H	0
0	0000H	FFFFH	0001H	0
0	FFFDH	0000H	FFFEH	0
0	FFFDH	FFFFH	FFFEH	0
1	0002H	0000H	0003H	0
1	0002H	FFFFH	0003H	0
1	FFFFH	0000H	0000H	1
1	FFFFH	FFFFH	0000H	1
1	0003H	0003H	0004H	0
1	0003H	0007H	0004H	0
1	0007H	0007H	0008H	0
1	0007H	000FH	0008H	0
1	000FH	000FH	0010H	0
1	000FH	001FH	0010H	0
1	001FH	001FH	0020H	0
1	001FH	003FH	0020H	0
1	003FH	003FH	0040H	0
1	003FH	007FH	0040H	0
1	007FH	007FH	0080H	0
1	007FH	00FFH	0080H	0
1	00FFH	00FFH	0100H	0
1	00FFH	01FFH	0100H	0
1	01FFH	01FFH	0200H	0
1	01FFH	03FFH	0200H	0
1	03FFH	03FFH	0400H	0
1	03FFH	07FFH	0400H	0
1	07FFH	07FFH	0800H	0
1	07FFH	0FFFH	0800H	0
1	0FFFH	0FFFH	1000H	0
1	0FFFH	1FFFH	1000H	0
1	1FFFH	1FFFH	2000H	0
1	1FFFH	3FFFH	2000H	0
1	3FFFH	3FFFH	4000H	0
1	3FFFH	7FFFH	4000H	0
1	7FFFH	7FFFH	8000H	0
1	7FFFH	FFFFH	8000H	0

DEC :

Cin	Ai	Bi	Si	Cout
0	0000H	0000H	FFFFH	0
0	0000H	FFFFH	FFFFH	0
0	0001H	0000H	0000H	1
0	0001H	0000H	0000H	1
1	FFFFH	0000H	FFFEH	1
1	FFFFH	FFFFH	FFFEH	1
1	FFFEH	0000H	FFFDH	1
1	FFFEH	FFFFH	FFFDH	1
0	FFFCH	FFFCH	FFF8H	1
0	FFFCH	FFF8H	FFFBH	1
0	FFF8H	FFF8H	FFF7H	1
0	FFF8H	FFF0H	FFF7H	1
0	FFF0H	FFF0H	FFEFH	1
0	FFF0H	FFE0H	FFEFH	1
0	FFE0H	FFE0H	FFDFH	1
0	FFE0H	FFC0H	FFDFH	1
0	FFC0H	FFC0H	FFBFH	1
0	FFC0H	FF80H	FFBFH	1
0	FF80H	FF80H	FF7FH	1
0	FF80H	FF00H	FF7FH	1
0	FF00H	FF00H	FEFFH	1
0	FF00H	FE00H	FEFFH	1
0	FE00H	FE00H	FDFFH	1
0	FE00H	FC00H	FDFFH	1
0	FC00H	FC00H	FBFFH	1
0	FC00H	F800H	FBFFH	1
0	F800H	F800H	F7FFH	1
0	F800H	F000H	F7FFH	1
0	F000H	F000H	EFFFH	1
0	F000H	E000H	EFFFH	1
0	E000H	E000H	DFFFH	1
0	E000H	C000H	DFFFH	1
0	C000H	C000H	BFFFH	1
0	C000H	8000H	BFFFH	1
0	8000H	8000H	7FFFH	1
0	8000H	0000H	7FFFH	1

Remarque :

XXX représente l'opération réalisée par l'U.A.L. lorsque le champ "Code-Opération" de l'instruction TUAL contient le codage des instructions de décalage. Ce type d'opération étant réalisé directement au niveau du registre tampon E1 d'entrée de l'UAL et étant testé lors de la phase de test des registres.

Le test "exhaustif par tranche" de l'U.A.L. de 16 Bits nécessite donc :

(14 x 8) + (2 x 36) = 184 Vecteurs.

IV.1.2.2 - Le dispositif de masquage :

IV.1.2.2.1 - fonction

Afin de pouvoir effectuer des opérations sur des bits ou des parties de mots de 16 bits, il existe un dispositif situé à la sortie de l'UAL permettant de masquer certains bits de l'opérande.

Le schéma général de l'UAL et du dispositif de masquage (D.M.) est donné ci-dessous (Figure 15) :

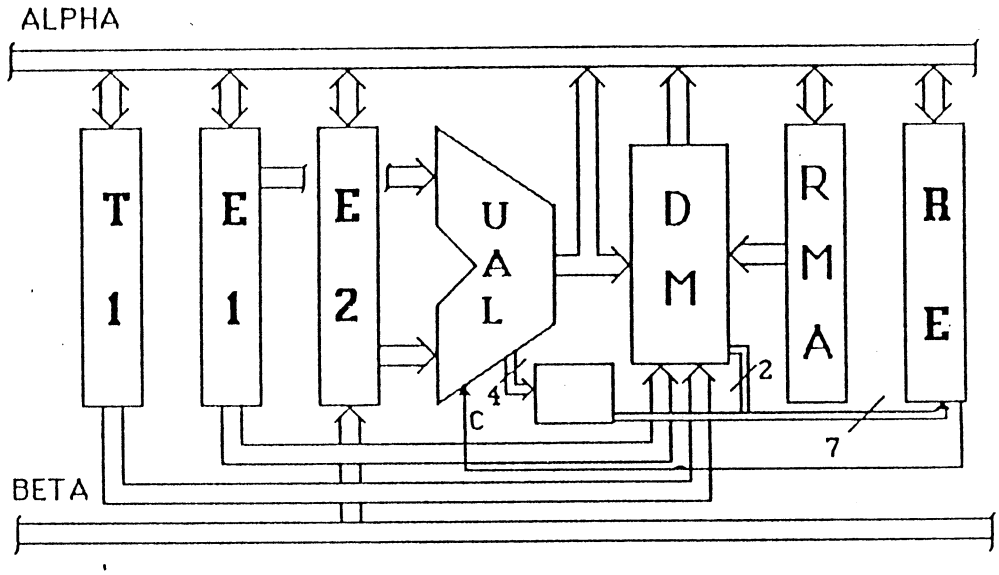


Figure 15

Le masque est un mot de 16 bits rangé dans un registre spécialisé (registre masque RMA). Ce registre est initialisé automatiquement au cours de l'exécution des instructions masquées.

Pour toutes les opérations masquées, seuls les bits de l'opérande correspondants à des bits à "1" dans le registre masque seront affectés par l'opération. Dans le cas d'une opération U.A.L (ANDM, ORM, NANDM, NORM, EORM et COMP) E_1 contient le contenu du registre à modifier, le mot disponible en sortie du dispositif de masquage (DM) est constitué par les bits sortant de l'UAL (bits modifiés) correspondants aux bits à "1" dans le registre masque et par les bits provenant de E_1 (bits modifiés) correspondant aux bits à "0" dans le registre masque.

Dans le cas d'une opération de transfert (LOADM et STOREM), la source des bits modifiés n'est plus SUAL mais le contenu de T_1 qui est un registre tampon :

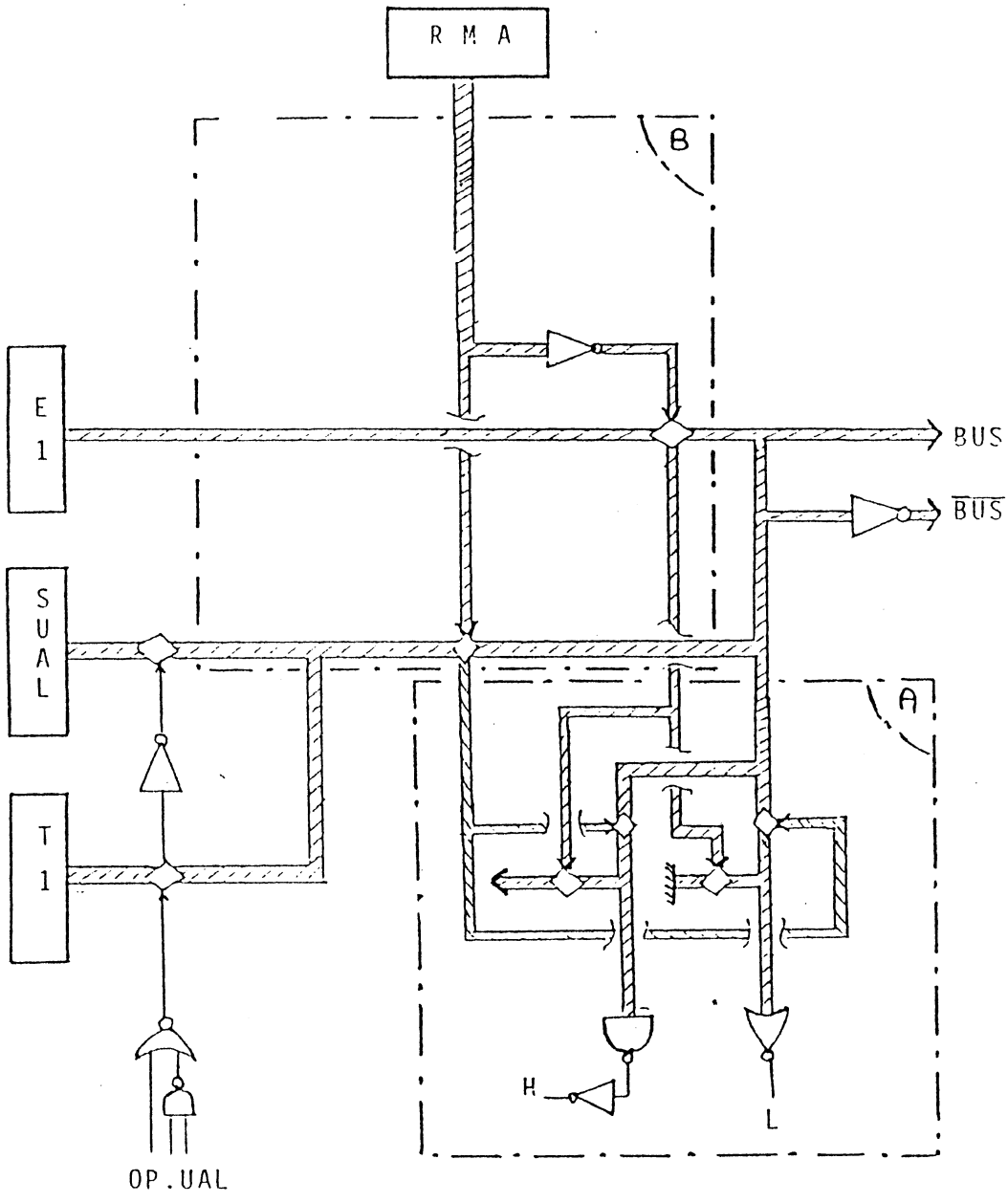
* dans le cas de LOADM, T_1 contient l'opérande et E_1 le contenu du registre à modifier.

* dans le cas de STOREM, T_1 contient le contenu du registre et E_1 le contenu du mot mémoire à modifier. Pour réaliser cette instruction, il est indispensable d'effectuer un cycle RMW (Read Modify Write).

Le choix entre les sources "Sortie UAL" et "Registre T_1" est réalisé par une commande provenant de la ROM indiquant si l'instruction donne lieu à une opération UAL.

IV.1.2.2.2 - Structure

Le schéma de principe du dispositif est donné ci-après (Figure 16) :



(Pour les détails A et B voir annexe 4)

Figure 16

Deux drapeaux indicateurs d'état sont générés à la sortie du dispositif de masquage :

H : H = 1 si tous les bits non masqués sont à 1
H = 0 sinon

L : L = 1 si tous les bits non masqués sont à 0
L = 0 sinon

IV.1.2.3 - **Le multiplieur**

IV.1.2.3.1 - Fonction :

Comme il a été indiqué au niveau du jeu d'instruction, le microprocesseur HSURF offre des possibilités de travail sur des matrices bidimensionnelles.

Une matrice est caractérisée par un ensemble d'informations statiques (nombre de ligne, nombre de colonnes, adresse de l'élément MAT) disponibles dans un descripteur. L'adresse d'un élément quelconque de cette matrice est obtenue en réalisant l'opération suivante :

Adr = Adr-elt (0,0) + (n° ligne * nombre-colonne) + n° colonne.

L'utilisation des instructions portant sur des matrices nécessite des calculs fréquents d'adresses d'éléments, il est donc indispensable de pouvoir réaliser la fonction multiplication rapidement.

Les multiplieurs utilisant l'UAL du microprocesseur et réalisant des additions plus décalage ont été rejetés du fait de leur temps d'exécution pouvant parfois être pénalisant, et du fait de la nécessité d'implanter un microprogramme contrôlant l'algorithme de multiplication. En effet, ce type de fonctionnement impose de nombreux échanges entre la partie contrôle et la partie opérative (test de parité de l'opérande, test de fin de multiplication,...) et pose le problème délicat du test des communications entre ces deux parties.

L'opération de multiplication sera donc réalisée à l'aide d'un opérateur combinatoire cellulaire [SHE 83] [SHE 84] présentant l'avantage d'être rapide et répondant à notre politique générale de test qui consiste à réaliser des opérateurs observables et testables de façon aussi indépendante que possible du reste du microprocesseur.

IV.1.2.3.2 - Structure du multiplieur

L'algorithme utilisé est basé sur la méthode des additions partielles.

Par exemple :

```
      1 0 1 0
    * 1 1 0 0
    ---------
      1 0 1 0
  + 0 0 0 0
    ---------
    0 1 0 1 0
+ 1 0 1 0
  -----------
  1 1 0 0 1 0
+ 1 0 1 0
  -----------
1 0 0 0 0 0 1 0
```

Une telle suite d'additions peut facilement être réalisée par un circuit combinatoire cellulaire dont les éléments de base sont des additionneurs complets (fig.17) organisés comme suit :

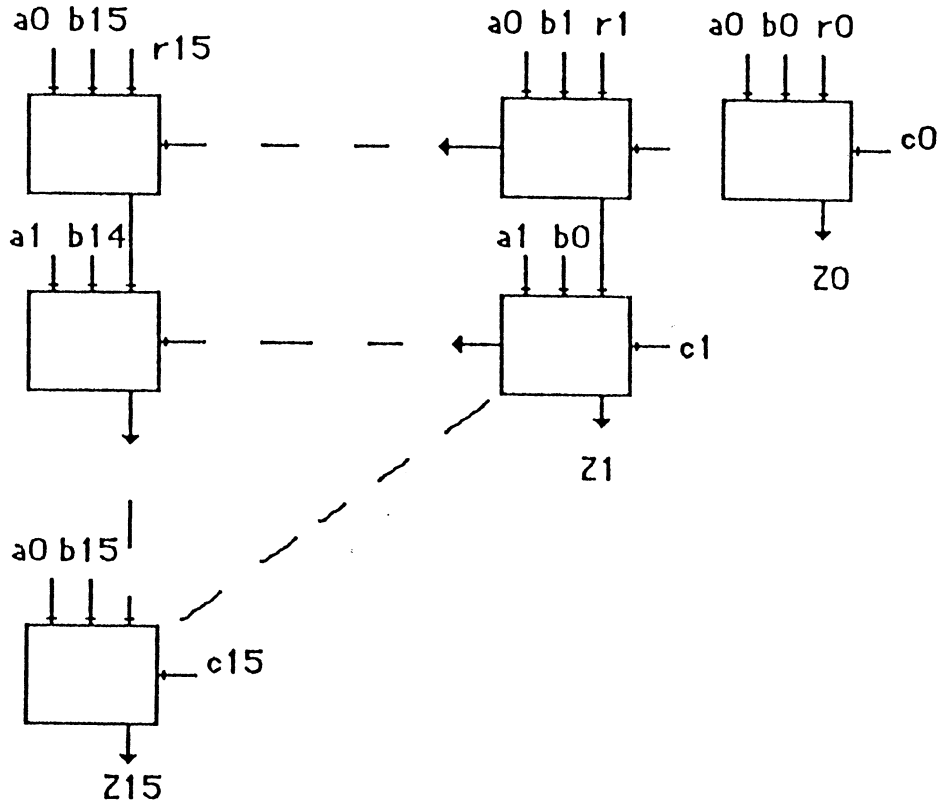


Fig. 17

Une telle organisation présente l'avantage de réaliser directement l'opération A * B + C + D, soit le calcul complet de l'adresse d'un élément de la matrice.

Le résultat de la multiplication de 2 nombres de n bits est un nombre de 2 * n bits, or, une restriction du microprocesseur HSURF est de limiter l'encombrement de chaque matrice à une page (soit 2^{16} éléments) au maximum. La valeur finale résultant d'un calcul d'adresse pour un élément de matrice doit toujours (en fonctionnement normal) pouvoir être codée sur 16 bits. Il apparaît donc qu'il n'est pas indispensable d'intégrer l'ensemble du multiplieur, mais seulement les cellules permettant de réaliser le calcul sur les 16 bits de poids faible. Les cellules réalisant le calcul des poids forts sont remplacées par un détecteur de débordement.

La structure de l'opérateur de multiplication de HSURF est donc la suivante (figure 18)

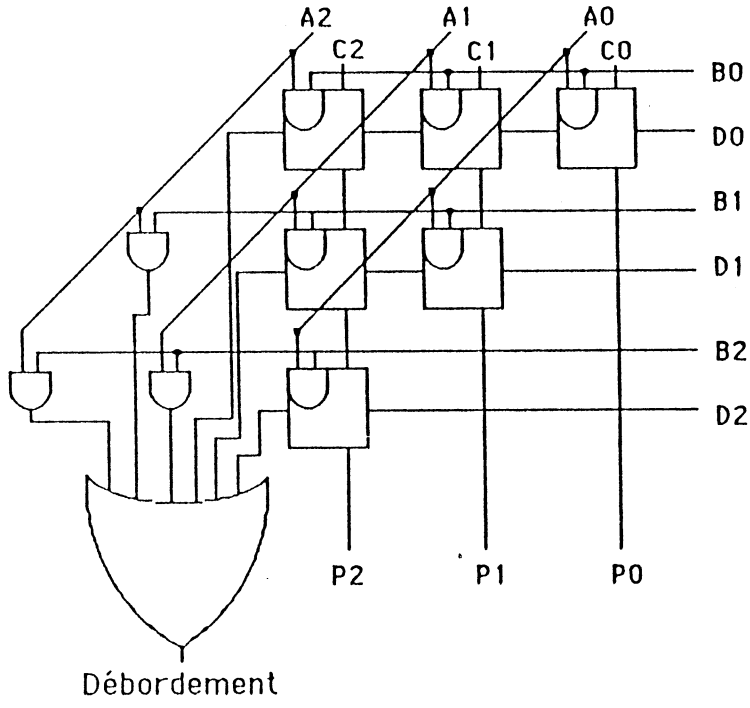


figure 18

Le multiplieur complet utiliserait 16 x 16 = 256 cellules alors, que le multiplieur réduit de HSURF n'utilise que 136 cellules (16+15+...+1 = 136)

IV.1.2.3.3 -·Le dispositif de détection de débordement

Lors d'une opération de multiplication avec notre opérateur réduit, il y a débordement si au moins l'une des deux conditions suivantes est réalisée :

1) l'une des retenues sortantes d_{out} d'une des cellules verticales de gauche est à 1

2) $a_1 b_n + a_2(b_n+b_{n-1}) + a_3(b_n+b_{n-1}+b_{n-2}) + \ldots + a_n(b_n+b_{n-1}+\ldots)$

$$= \sum_{i=1}^{n} a_i \cdot \left(\sum_{j=1}^{i} b_{n-j+1} \right) \text{ est à 1.}$$

Ceci peut être réalisé comme sur la figure précédente, soit comme sur la figure 19 suivante :

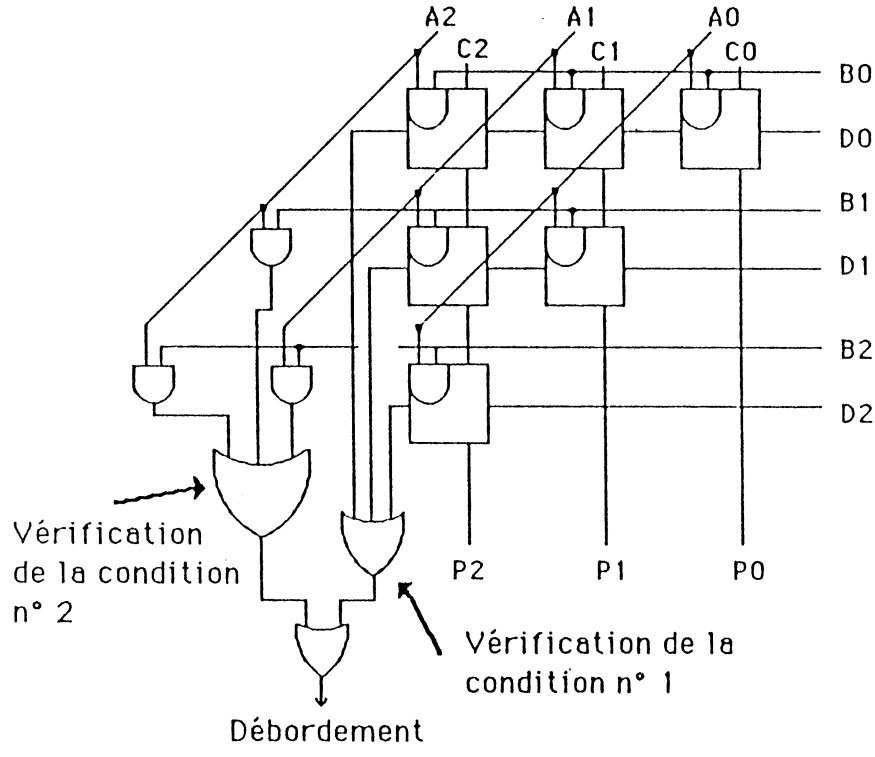


Figure 19

Pour des facilités d'implantation, le dispositif de débordement a été réalisé de façon plus cellulaire comme représenté figure 20.

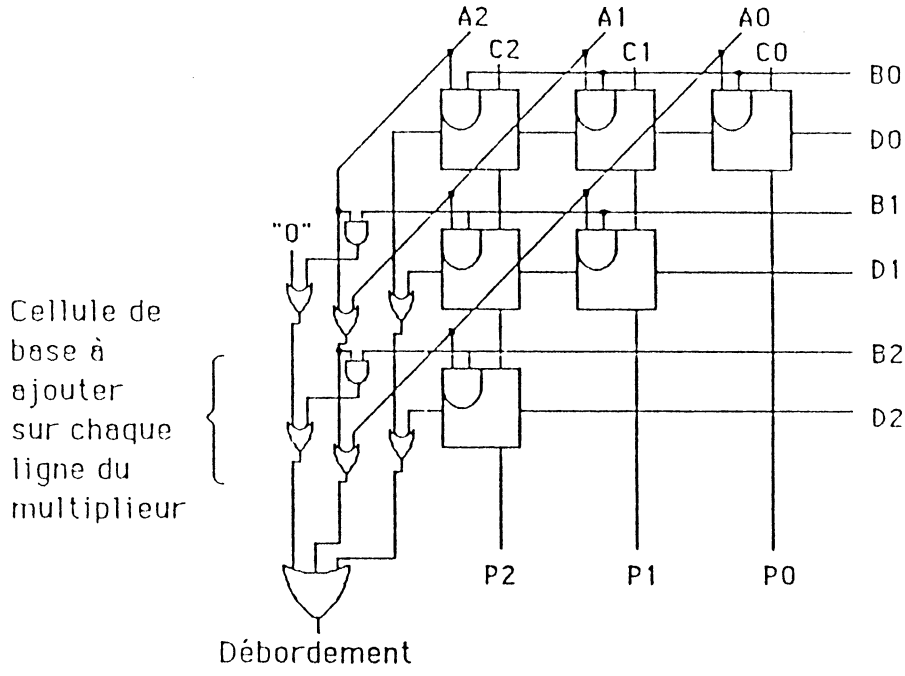


Figure 20

IV.1.2.3.4 - Test du multiplieur :

Pour les mêmes raisons que pour l'UAL (nb d'entrées primaires du multiplieur = 4 x 16 = 64) un test exhaustif de l'ensemble de l'opérateur s'avère inapplicable. Le multiplieur peut cependant être considéré comme un ILA bidimentionnel un test exhaustif par cellules avec les mêmes hypothèses que pour l'UAL peut donc être réalisé. Dans [DEC 85] une étude détaillée concernant le test de cet opérateur montre qu'il faut 241 vecteurs de test pour appliquer à chacune des 136 cellules les 16 entrées possibles (2^4) et pour propager les résultats vers les sorties primaires du circuit.
Il serait possible, comme il l'a été fait pour l'UAL de modifier la structure de base d'une cellule de façon à rendre le multiplieur C-testable, le

résultat de cete étude est proposé dans [SHE 84], on dispose alors d'un opérateur permettant de réaliser la multiplication et dont le test peut être accompli avec seulement 16 vecteurs. Malheureusement, la modification de la cellule entraîne une modification de sa table de vérité qui impose lors de l'étape de multiplication d'avoir Rin = Cin = 0. L'opération effctuée par l'opérateur est alors A * B, l'un des intérets principaux de notre multiplieur (calcul d'adresse A + B + C + D immédiat) est donc perdu. Nous avons donc préféré conserver la structure ainsi que les caractéristiques logiques de la cellule de base.

L'ensemble des 241 vecteurs de test l'opérateur de multiplication sont donnés en annexe 2.

IV.1.2.4 - Le dispositif de signature

IV.1.2.4.1 - Principe

Rappelons que le microprocesseur HSURF est destiné à garantir une faible latence de panne ce qui doit se traduire par un test efficace et répété de l'ensemble de ses ressources internes. Nous venons de voir que pour faciliter cette tâche, l'ensemble des opérateurs de la partie opérative a été conçu de façon à bénéficier d'un test rapide et efficace.
Malgré cela, il est indispensable de pouvoir vérifier de façon continue. (test en ligne continue) le bon fonctionnement du microprocesseur et ceci nécessite la vérification d'une très grande quantité d'information difficilement compatible avec le déroulement normal du programme d'application.

Une approche permettant de concilier la rapidité de test et l'importante quantité d'information à traiter consiste à réaliser une compaction de l'ensemble des informations à vérifier.

On utilise généralement le schéma suivant (Figure 21) :

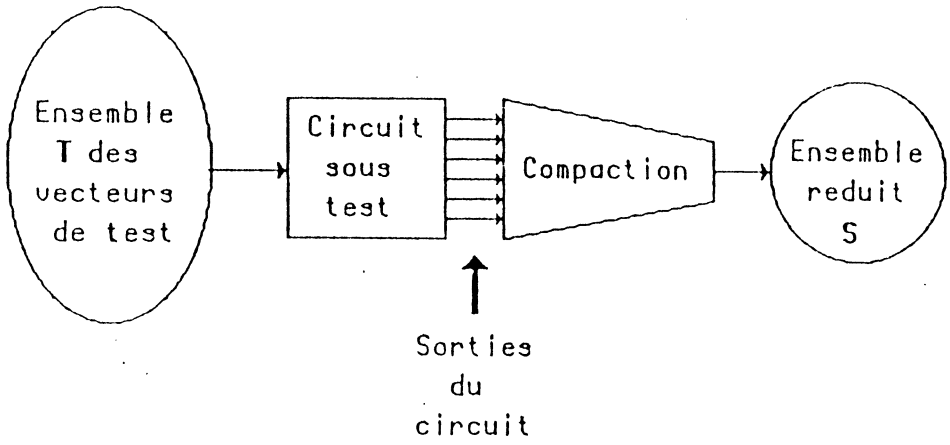


Figure 21

Le contenu de l'ensemble S est en quelque sorte un "résumé" aussi représentatif que possible de l'ensemble des événements qui se sont manifestés à la sortie du circuit sous test.

Le contenu de cet ensemble représente la signature des informations résultantes de la séquence de test. De nombreuses méthodes on été proposées pour obtenir cette signature ; nous pouvons citer par exemple les méthodes consistant à compter les transitions 0→1 ou 1→0 à la sortie du circuit sous test [HAY 76], [VEN 80], [SAV 80], les méthodes vérifiant les coefficients de walsh [SUS 81] [MUZ 82], les méthodes fondées sur la division polynomiale [DAV 78], [LTW 84], [HAS 84].

Nous nous intéressons ici à cette dernière méthode dont nous allons rappeler le principe et évaluer l'efficacité.

La figure 22 donne un arrangement typique du dispositif permettant d'effectuer une division polymiale série.

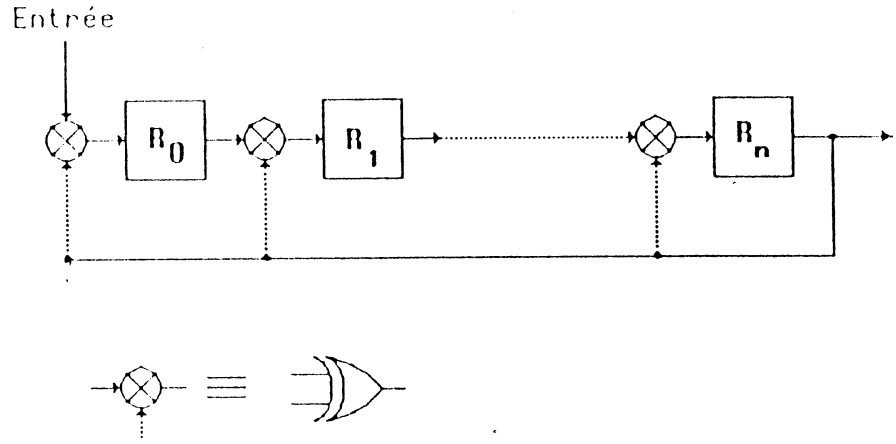


Figure 22

Il est constité d'un registre à décalage dont la sortie peut être rebouclée en différents points d'entrées des registres.

La série de bits en entrée constitue les différents coefficients d'un polynome (dividende) qui va être lors des successifs décalages à droite divisé par un polynome diviseur dont les coeficients sont fixés par la présence ou non des différents rebouclages. La sortie S est représentative du quotient de la division, et, en fin d'opération, le contenu des différents registres du dispositif de signature représente le reste de cette division ; c'est en fait la signature.

Le défaut de ce système est de ne réaliser que la signature d'un flot de données série ; nous étendons donc son principe à un dispositif comportant plusieurs entrées [cf fig 23] nous permettant de réaliser le compactage de données en parallèle.

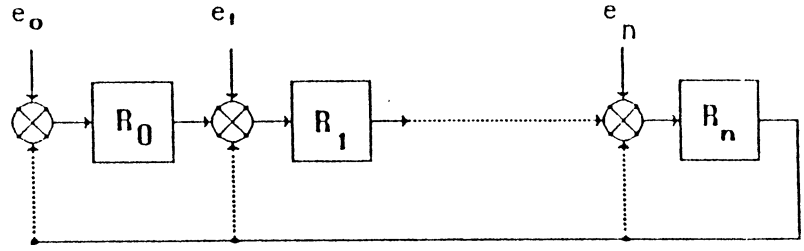


Figure 23

Les différents rebouclages possibles permettent de définir les coefficients du polynôme diviseur de la façon suivante :

$$\text{Pol div} = x^n + R_{n-1}x^{n-1} + R_{n-2}x^{n-2} + \ldots + R_i x^i + \ldots + R_1 x + 1$$

où :

$R_i = 1$ si la sortie du point mémoire R_n est connectée à la porte Xor présente à l'entrée du point mémoire Ri $(0 < i < n)$

$R_i = 0$ sinon.

* Equivalence entre la signature parallèle et le signature série :

On apprend dans [HAS 82] qu'un registre de division parallèle auquel on applique les suites d'entrées $e_0 \ldots e_{n-1}$ correspondantes aux coeff. des polynômes $M_0(x)$ à $M_{n-1}(x)$, se comporte de la même façon qu'un dispositif de compaction série à l'entrée duquel on aurait appliqué le polynôme

$$M(x) = M_0(x) + x.\ M_1(x) + \ldots + x^{n-1}.\ M_{n-1}(x)$$

On peut éclairer ceci par le schéma suivant (Figure 24) :

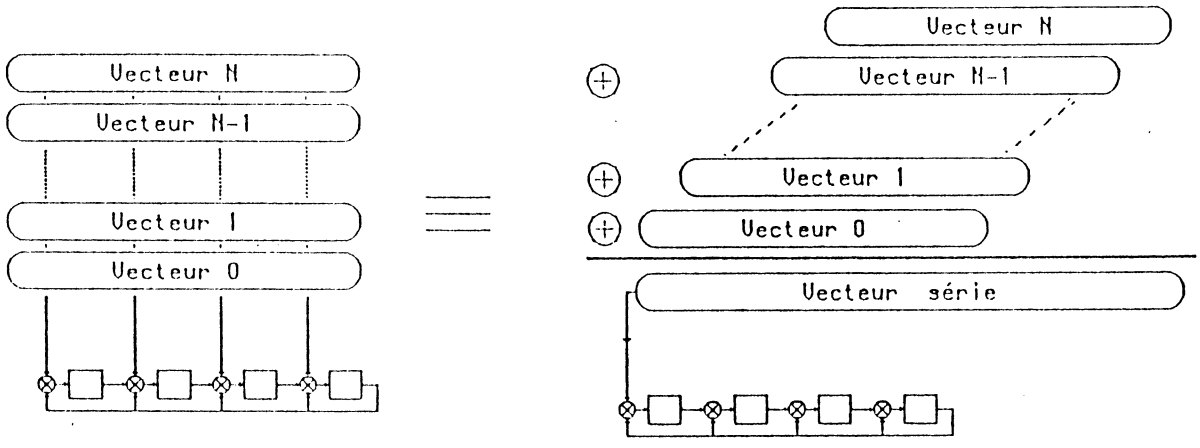


Figure 24

Un tel dispositif, parait donc tout à fait adapté pour réaliser la compaction d'une suite d'information.

L'impératif principal de l'étape de compaction est de réduire une grande suite de données, en conservant le maximum d'informations sur les évènements s'étant manifesté en sortie du circuit sous test.

Nous allons maintenant évaluer l'efficacité d'un tel dispositif de compaction. Le taux de couverture dans de tels mécanismes dépend de :

i - L'efficacité avec laquelle la séquence de Stimuli d'entrée du circuit sous test va détecter les pannes.

ii) Du taux de pannes, mises en évidence par les stimuli d'entrée et masquées par l'opération de compaction.

Le premier point concerne le problème classique de la génération de vecteurs de test efficaces.

Le microprocesseur HSURF ayant fait l'objet lors de sa conception d'études spécifiques tendant à rendre aisée la génération de ces vecteurs de test, nous ne considérerons pas ce problème ici.

Le deuxième point demande quand à lui une étude plus détaillée de la méthode de compaction par division polynômiale. Un dispositif réalisant le test par analyse de signature peut être schématiquement représenté comme suit (Figure 25) :

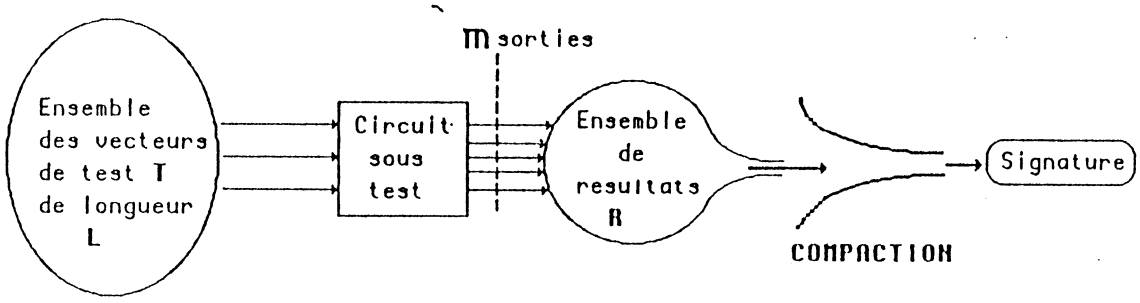


Figure 25

Soit T l'ensemble des L stimuli de test ; l'application de cet ensemble à l'entrée du circuit, va générer un ensemble de vecteurs de sortie R (réponses du circuit) qui sera en fait une matrice contenant L vecteurs de longueur m où m est nb. de sorties du circuit sous test.

Soit F l'ensemble de toutes les pannes possibles du circuit, on a un sous ensemble Fd de F représentant toutes les pannes que l'ensemble T de stimulis de test peut mettre en évidence, et on a un sous ensemble Fm de Fd qui contient les pannes masquées par l'opération de compaction.

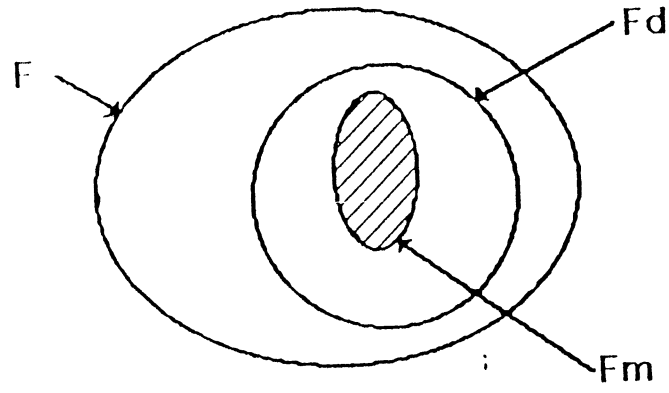


Figure 26

Soit E l'ensemble de toutes les matrices d'erreur pouvant être obtenues à la sortie du circuit sous test. On a card (E) $=2^{m.L}-1$ car on peut réaliser $2^{m.L}$ matrices de dimension m x L différentes, et, parmi toutes, une seule correspond à un fonctionnement correct du circuit (les $2^{m.L}-1$ autres caractérisant une erreur).

Nous avons le schéma suivant (Figure 27) :

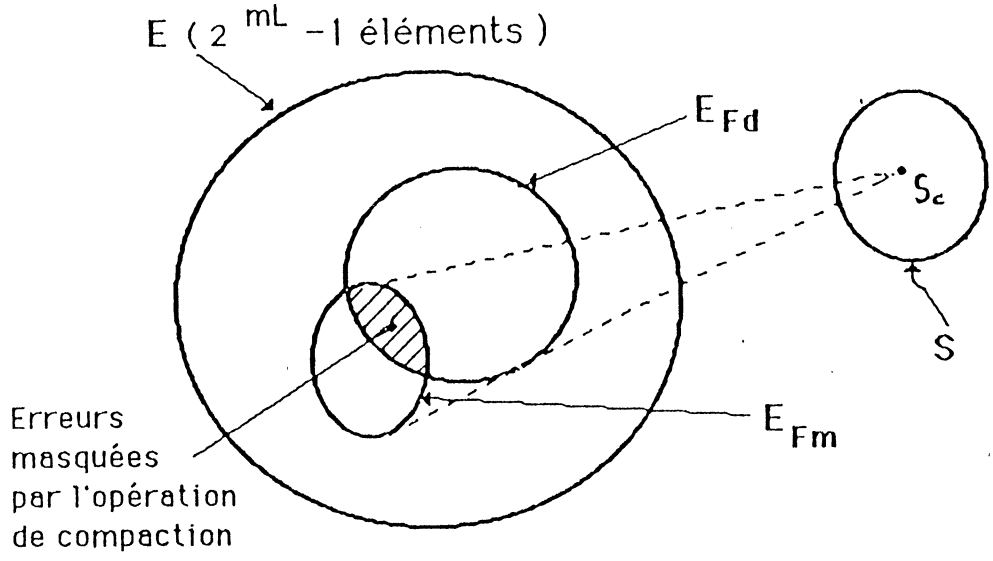


Figure 27

où

E_{Fd} : ensemble des matrices d'erreurs correspondants aux pannes Fd détectées par la séquence de test.

E_{Fm} : ensemble des matrices d'erreurs masquées, car donnant après compaction la même signature que la matrice résultat du circuit juste.

S : ensemble de toutes les signatures réalisables sur m bits (card (S) = 2^m).

Sc : signature résultant du test d'un circuit correct.

La quantité $\frac{|E_{fm}|}{|E|}$ donne la probabilité de masquer une erreur en effectuant la compaction.

HYPOTHESE : On admet que l'opération de compaction par division polynômiale réalise une transformation uniforme de l'ensemble E dans l'ensemble S,

On peut dire que :

$\left| E_{fm} \right| = \frac{2^{mL}}{2^m}$, sachant que dans cette numération, on compte la matrice résultant d'un circuit juste.

Donc, l'ensemble des matrices d'erreurs donnant la même signature que la matrice "circuit juste" est composé de $\frac{2^{mL}}{2^m} - 1$ éléments.

d'où :

$$\text{Probalité de masquage} = \frac{2^{mL} / 2^m - 1}{2^{mL} - 1} \xrightarrow{L \text{ grand}} \frac{1}{2^m}$$

On voit donc, que la probabilité de masquage est fonction du nb de bits (nb. de sorties du circuit) sur lequel on effectue la division. Il faut aussi noter que les erreurs qui sont réellement masquées sont en fait les erreurs correspondant à $E_{fm} \cap E_{fd}$ et, rien ne nous protège contre le fait que dans certains cas de figure l'ensemble E_{fm} soit significatif devant l'ensemble E_{fd}, et même, que $E_{fd} \subseteq E_{fm}$ dans quel cas, la totalité des fautes à priori détectables seraient masquées ; ceci dépend du polynôme diviseur.

Peut on diminuer cette probabilité d'erreurs ?

A) Une première solution, consiste à augmenter la longueur du message représentant la signature.

Nous avons alors le schéma suivant (Figure 28) :

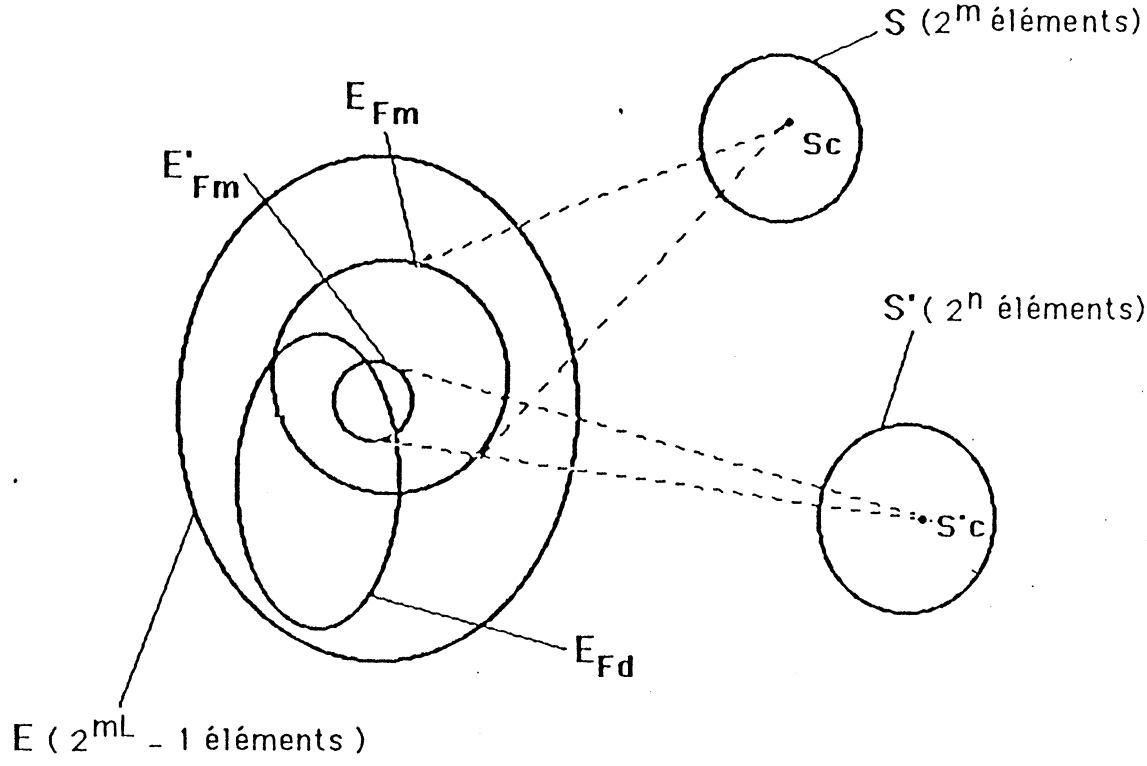


Figure 28

où :

S' : ensemble des signatures possibles sur n bit (card S' = 2^n)

et

E'_{fm}: ensemble des matrices d'erreurs donnant après compaction la même signature que la signature "circuit juste".

$$\text{on a : } \left|E'_{fm}\right| = \frac{2^{ml}}{2^n} \quad \text{d'où} \quad \text{prob} = \frac{1}{2^n}$$

Au niveau matériel, cette méthode consiste à ajouter des éléments de mémorisation dans l'ensemble de la chaine des registres à décalage ; le surcoût au niveau de la surface de silicium sera donc proportionnel au nombre d'éléments ajoutés.

B) Réalisation de plusieurs séries de test et de compaction

Cette approche consiste soit à stocker plusieurs signatures lors du compactage de la séquence de sortie, soit à réaliser plusieurs séries de tests avec des stimuli d'entrée différents et à effectuer plusieurs compactions.

Nous avons le schéma suivant (Figure 29 et 30) :

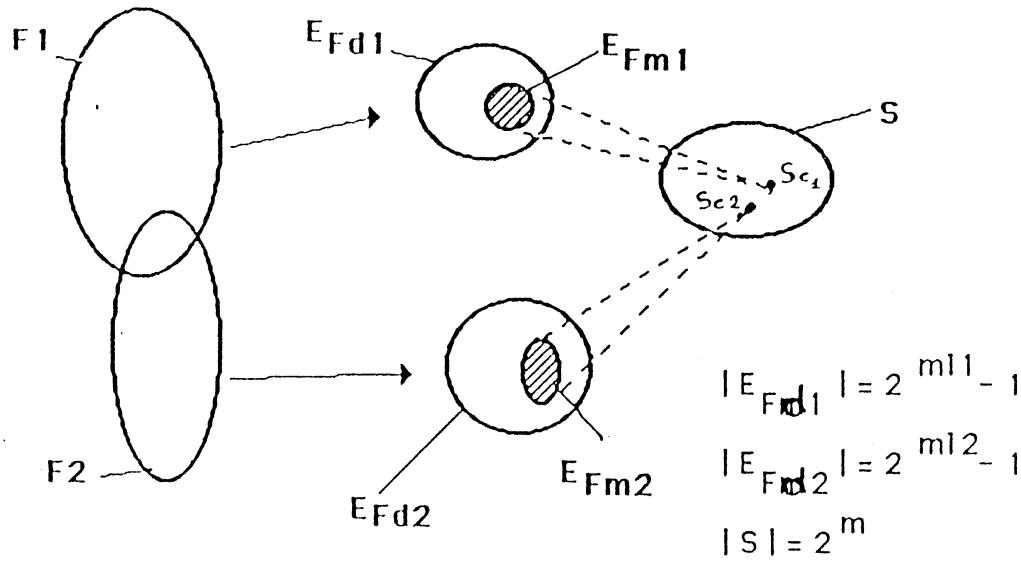


Figure 29

F_1 : ensemble des fautes décelables par T_1

F_2 : ensemble des fautes décelables par T_2

E_{fmi} ensemble des matrices d'erreurs masquées par la compaction et donnant la signature "circuit juste" Sci

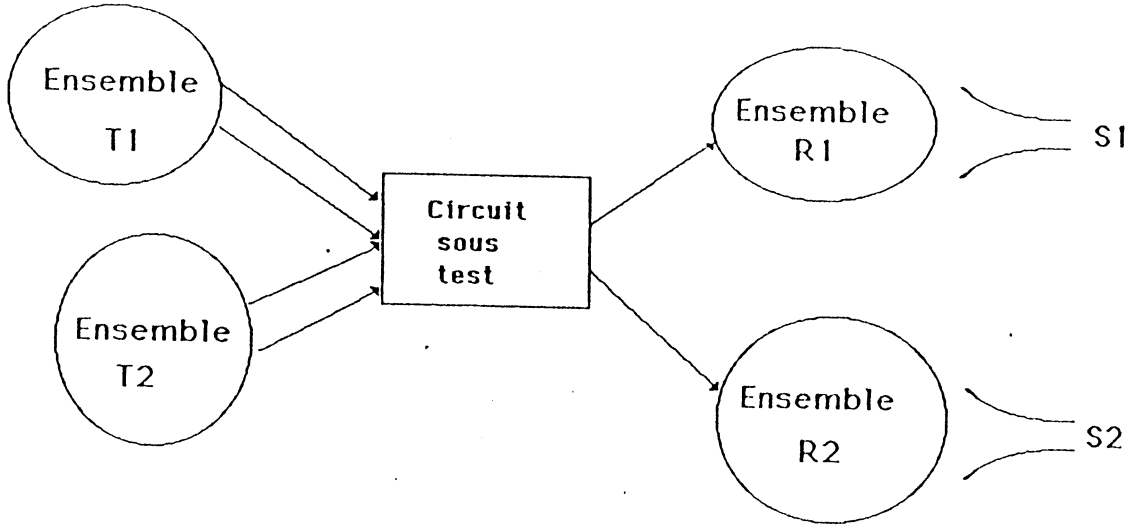


Figure 30

soit Sc_1 la signature correspondant à la transformation de l'ensemble réponse R_1 pour le "circuit juste", on a $\frac{2^{ml_1}}{m}$ matrices d'erreurs qui vont donner la même signature (idem pour Sc_2), donc, le nombre total de matrices d'erreurs donnant après compaction Sc_1 ou Sc_2 sera le produit de ces 2 quantités soit :

$$2^{ml_1 - m} \times 2^{ml_2 - m} = 2^{m(l_1 + l_2) - 2m}$$

Si pour pouvoir comparer avec la méthode précédente, on décide ne pas effectuer la compaction sur un nombre plus grand d'élément, on va choisir

$$l_1 + l_2 = L$$

donc :

$$2^m (l_1 + l_2) - 2m = 2^{mL} - 2m$$

Le nb d'erreurs masquées par le compactage sera donc $2^{mL} - 2m - 1$ d'où, la prob. de masque d'erreur :

$$\frac{2^{mL} - 2m - 1}{2^{mL} - 1} \sim \frac{2^m (L - 2)}{2^{mL}} \text{ si L grand}$$

$$= \frac{1}{2^{2m}}$$

On s'aperçoit ici que la valeur de l_1 et l_2 ne sont liées par aucune relation. L'amélioration de la prob. de masquage d'erreurs provient du fait que certaines pannes dévoilées par la séquence de test T_1 (début de la séquence de test T) étaient masquées par la suite des opérations de compaction correspondant à la séquence de test T_2 (fin de séquence de test T) Le fait de saisir des "échantillons" de signature au cours de l'opération de compactage permet donc de masquer moins d'erreurs. (l'idéal étant bien sur obtenu si l'on stocke la signature à chaque pas de compactage, ce qui revient à vérifier les réponses à tous les vecteurs de test, donc à ne pas bénéficier du compactage !)

Dans [HAS 84] est développée l'approche consistant à effecter différentes séquences de compactage.

Notons quand même que les erreurs qui étaient masquées par le dispositif de base et qui ne sont détectées par aucune des sous-séquences de vecteurs de test T_i resteront toujours masquées ; le seul moyen permettant de mettre en évidence de telles classes de pannes est de modifier les caractéristiques du polynôme diviseur.

C) Réalisation de différents compactages avec un polynôme diviseur différent

Dans cette technique, on va réaliser à partir d'un même ensemble de vecteurs de test T plusieurs compactions en utilisant des polynômes diviseurs de caractéristiques différentes.

Nous avons le schéma suivant (Figure 31) :

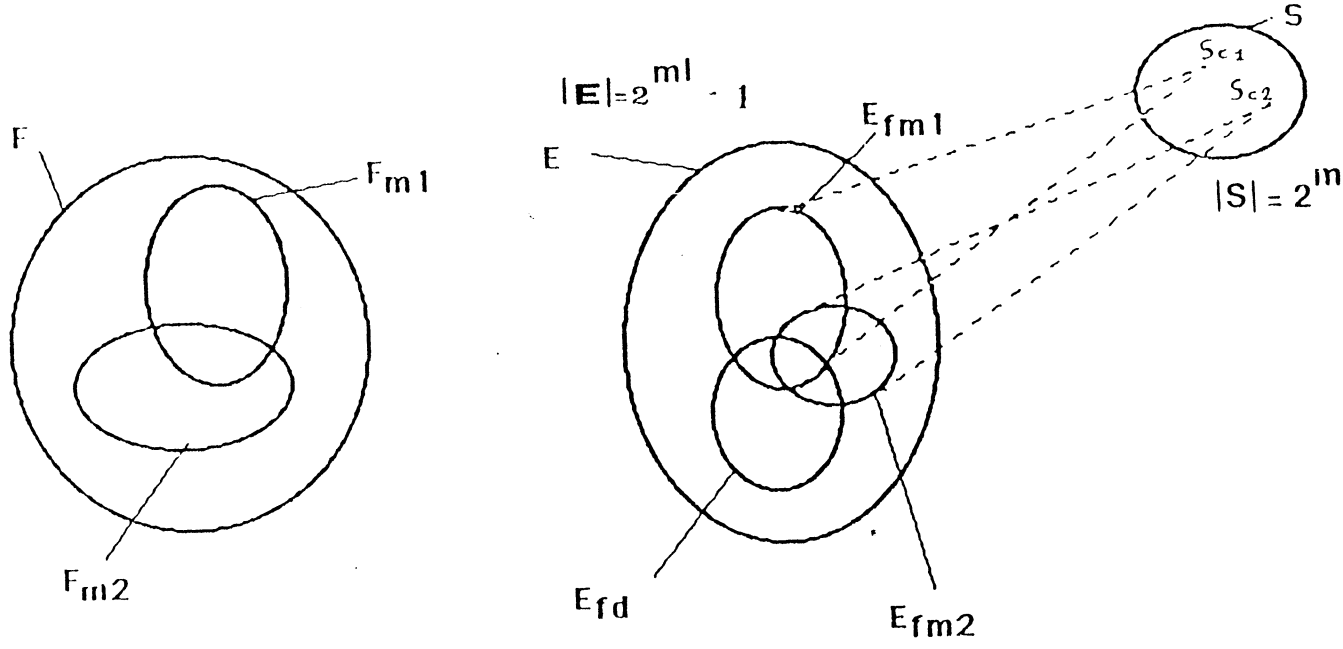


Figure 31

F : ensemble des pannes détectées par T.

F_{mi} : Ensemble des pannes masquées par la compaction avec le polynome P_i.

E : ensemble de toutes les matrices d'erreurs possibles

E_{fd} : ensemble des matrices d'erreurs détectables par la séquence de test T.

E_{fmi} : ensemble des erreurs masquées par la compaction avec $P_i(x)$.

De la même façon que précédemment, on obtient :

$$\text{prob. masquage d'erreurs} \simeq \frac{1}{2^{2m}}$$

Dans [SMI 80], le théorème 1 nous apprend que si s(x) est la signature générée par la compaction des entrées m(x) avec le polynôme diviseur p(x), pour une erreur correspondant au polynome d'erreur e(x), m(x) et m(x) + e(x) donneront la même signature S(x) si et seulement si e(x) est un multiple de P(x).

En fait, cela signifie que les vecteurs générés par un circuit faux seront masqués par la compaction si et seulement si ils sont divisibles par P(x). L'idéal, serait donc de trouver différents polynômes diviseurs $p_i(x)$ dont la population des erreurs masquées soit disjointe. Malheureusement, pour l'instant, ceci est impossible.

Conclusion :

Cette étude nous permet donc de constater que l'amélioration de l'efficacité de la compaction par division polynomiale repose sur trois points :

i) Augmentation de la taille de la signature (du registre à décalages) ce qui entraîne une augmentation du nombre de bascules, d'où une augmentation de la surface de silicium.

ii) Augmentation du nombre de vecteurs de test et stockage de plusieurs signatures.

iii) Compaction d'une même séquence de test à l'aide de différents polynômes diviseurs.

La première solution réalise une amélioration au détriment de la surface de silicium utilisée. En effet, si l'on veut obtenir par cette méthode une amélioration équivalente au méthodes ii) et iii) (passage de $P(x) = \frac{1}{2^n}$ à $P(x) = \frac{1}{2^{2m}}$) , il faut que n = 2m ; ce qui revient à doubler le nombre de registres du système de compaction, soit à doubler la surface de silicium utilisée pour la réalisation de ce dispositif.

D'autre part, le traitement (stockage + comparaison) d'une signature de longueur $n \neq m$ (où m est la largeur du chemin de données du microprocesseur) ne serait pas très facile à mettre en oeuvre.

Nous avons donc préféré implanter au niveau de la partie opérative du microprocesseur HSURF un dispositif permettant de réaliser une signature de longueur égale à 16 bits avec des facilités quand au choix du polynôme diviseur.

La stratégie d'utilisation d'un tel dispositif, sera dictée par les méthodes ii) et iii), c'est-à-dire , que l'on stockera au cours de la séquence de test plusieurs signatures qui pouront être réalisées avec différents polynômes diviseurs.

Les caractéristiques de ces polynômes pourront d'ailleurs être déterminées par une étude préalable des suites de données à compacter (lesquelles sont connues avec détail pour des séquences de test spécifiques et pour de nombreux programmes d'application) et par une simulation du compactage de ces différents flots d'information à l'aide d'un simulateur de signature tel le SIGLYZER (SIgnature anaLYZER) présenté par Shridar dans [SHR 82].

IV.1.2.4.2 - Le dispositif de signature programmable de HSURF

Dans le microprocesseur HSURF, toutes les informations circulant sur les bus (alpha et beta) peuvent être signées. On dispose d'un registre diviseur polynomial appelé SIGN implanté au niveau de la partie opérative comme suit (Figure 32) :

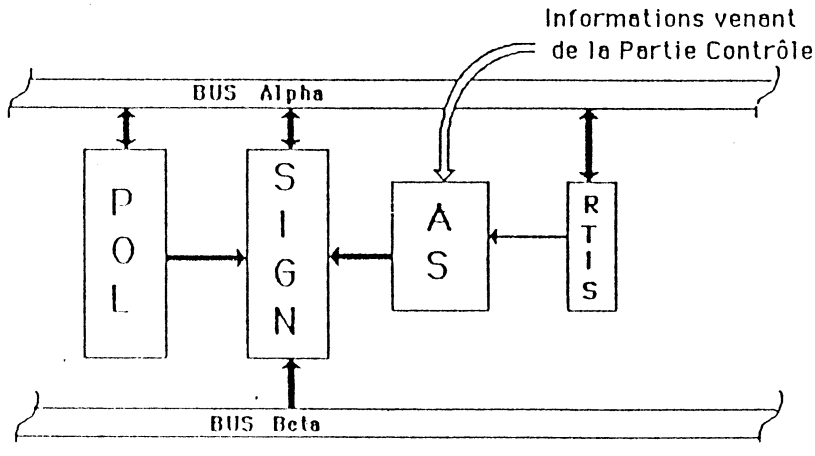


Figure 32

Afin de permettre une grande souplesse d'utilisation du dispositif et un choix facile du polynôme de division, deux registres spécialisés (POL et RTIS) permettent à l'utilisateur de définir deux paramètres :

A) Le polynôme caractéristique de la division peut être modifié à tout moment en changeant le contenu du registre POL. L'utilisateur pourra ainsi très facilement effectuer différentes signatures sur un même flot de données. Le rebouclage du dernier bit du registre sera autorisé sur la ième porte ou exclusif si le bit n° i du registre POL est à "0", sinon il n'y aura pas rebouclage (coefficient $R_i = 0$)

B) Le type des informations circulant sur les bus peut être programmé. Le contenu du registre RTIS (registre du type d'informations signées) permet en effet d'autoriser ou non la prise en compte de certaines informations circulant sur le bus pour effectuer la signature.

Le registre contient 8 bits dont la signification est la suivante (Figure 33) :

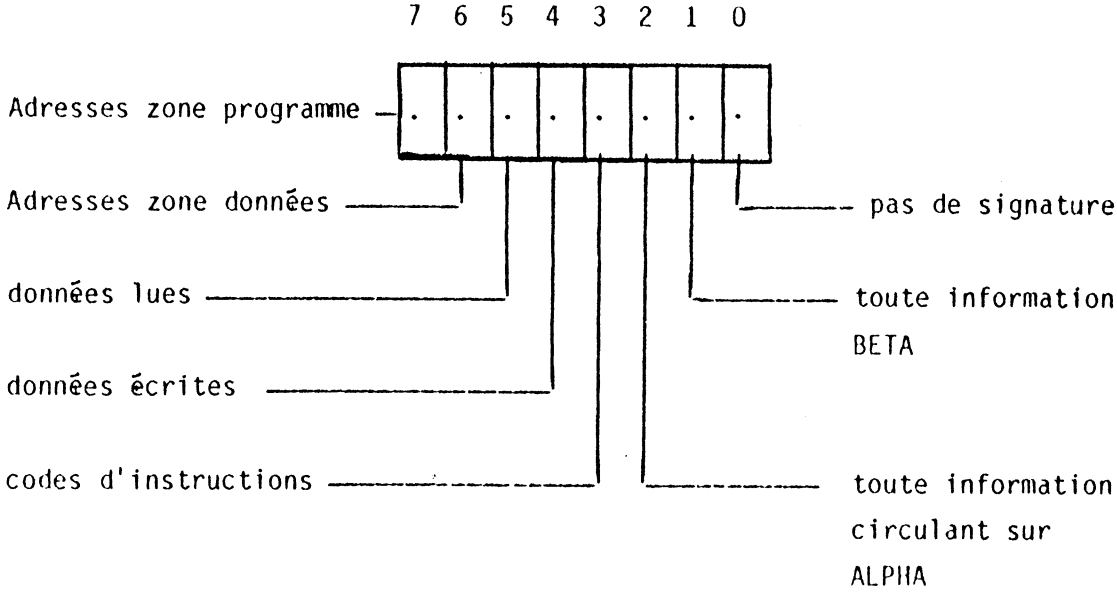


Figure 33

Le bit "pas de signature" à "1" interdit la signature quoi qu'il y ait dans les autres bits du registre.
Les bits 1 et 2 à "1" inhibent tous les autres bits de RTIS (sauf le bit 0).
En dehors de ces deux cas, les 32 configurations possibles des bits 7 à 3 sont utilisables.

L'utilisateur peut ainsi choisir par exemple de ne signer que les codes opération circulant sur le bus de façon à vérifier le bon séquencement d'un programme ou de certaines parties de programme indépendemment des données traitées.

Le dispositif AS visible sur le schéma de principe du dispositif de signature est un circuit logique autorisant ou non l'opération de compactage en fonction du contenu du registre RTIS et des commandes issues de la partie contrôle.

Tous les registres de la partie opérative devant être accessibles individuellement, il est possible de charger et de lire le regisre de signature sans signer ; de plus, l'utilisateur peut remettre à zéro ou ajuster le contenu de ce registre à l'aide d'instructions spécialisées.

Il existe donc quatre possibilités différentes permetant d'accéder au registre SIGN :

- chargement et signature
- chargement sans signature.
- remise à zéro.
- ajustement de la valeur du registre.

Remarque importante :

Toute opération dont l'un ou plusieurs des opérandes est un des registres de programmation du dispositif de signature (POL et RTIS) ne sera pas signée

Le schéma logique du registre SIGN est donné fig. 34
Le schéma logique du dispositif d'autorisation de signature est donné figure 35

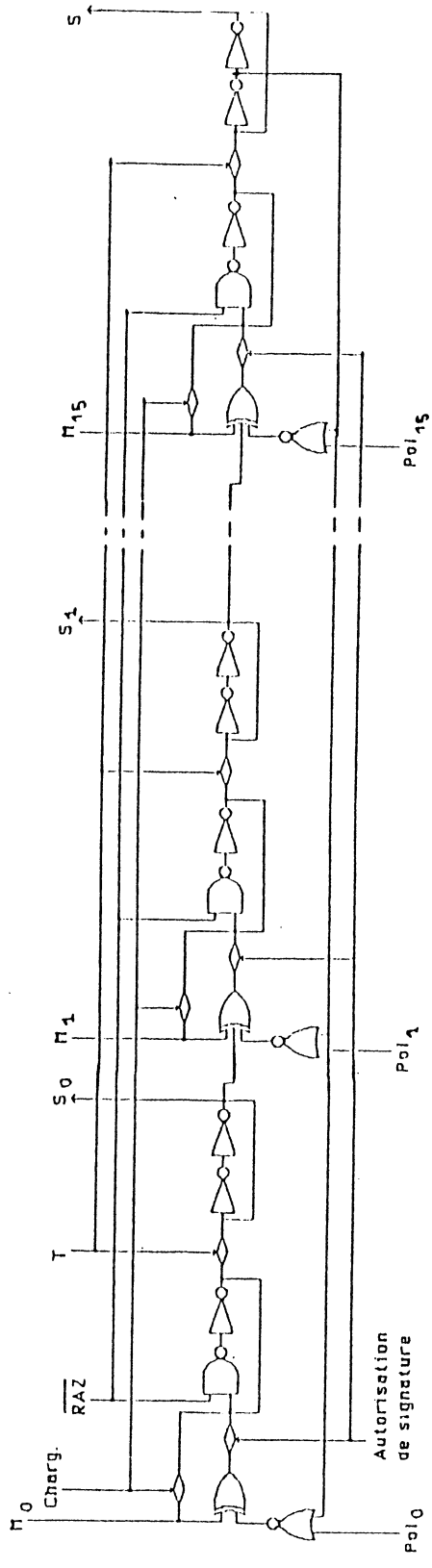


Fig. 34 : registre de signature

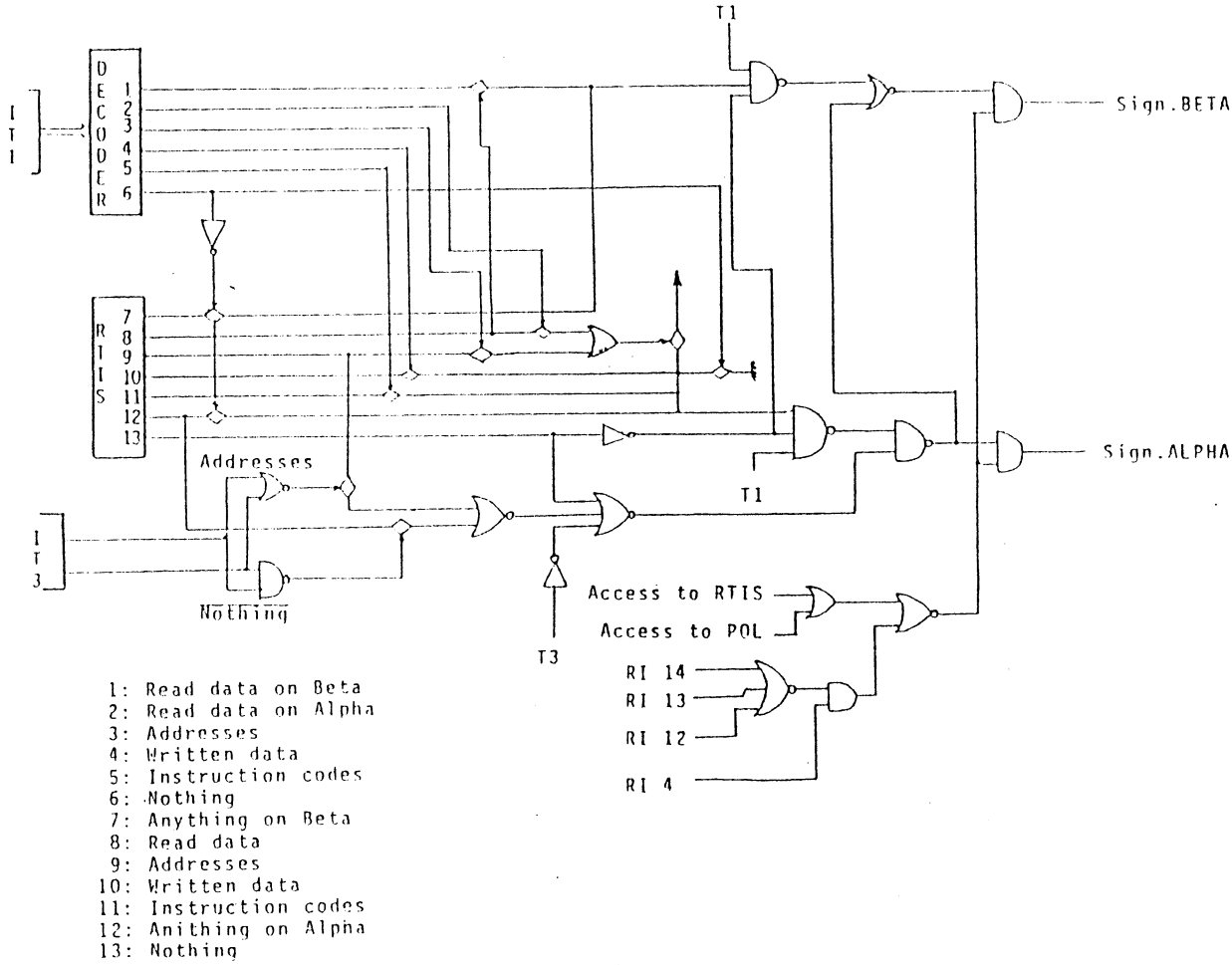


Fig. 35 : Dispositif d'autorisation de signature

Paralèllement à l'implantation de ce dispositif au sein du microprocesseur HSURF, un circuit autonome de signature basé sur les principes précédents destiné à être immergé dans toutes les les applications à microprocesseur a été développé par le Laboratoire Circuits et Systèmes (Réalisation sur réseau prédiffusé). Ce circuit permet d'effectuer la compaction de huit sorties (ou d'un multiple de 8 par mise en cascade de plusieurs circuits) avec possibilité de choix des caractéristiques du polynôme diviseur par simple

programmation d'un registre interne. Une présentation détaillée de ce composant est donnée dans [JAY 85].

IV.2 - Architecture générale de la Partie Opérative :

On donne figure 36 une représentation globale de l'architecture de la partie opérative du microprocesseur HSURF, sur laquelle on peut distinguer :

* l'ensemble des registres tous accessible de façon directe en lecture écriture via le bus Alpha
* L'Unité Arithmétique et Logique et le dispositif de masquage
* Le multiplieur cablé
* Le dispositif de signature.

On trouvera aussi dans le tableau 1 un récapitulatif de tous les registres de la partie opérative ainsi que leur mode d'accès :

- accès direct à l'aide des isntructions normales ou de test du microprocesseur

- accès seulement par l'intermédiaire des instructions de test.

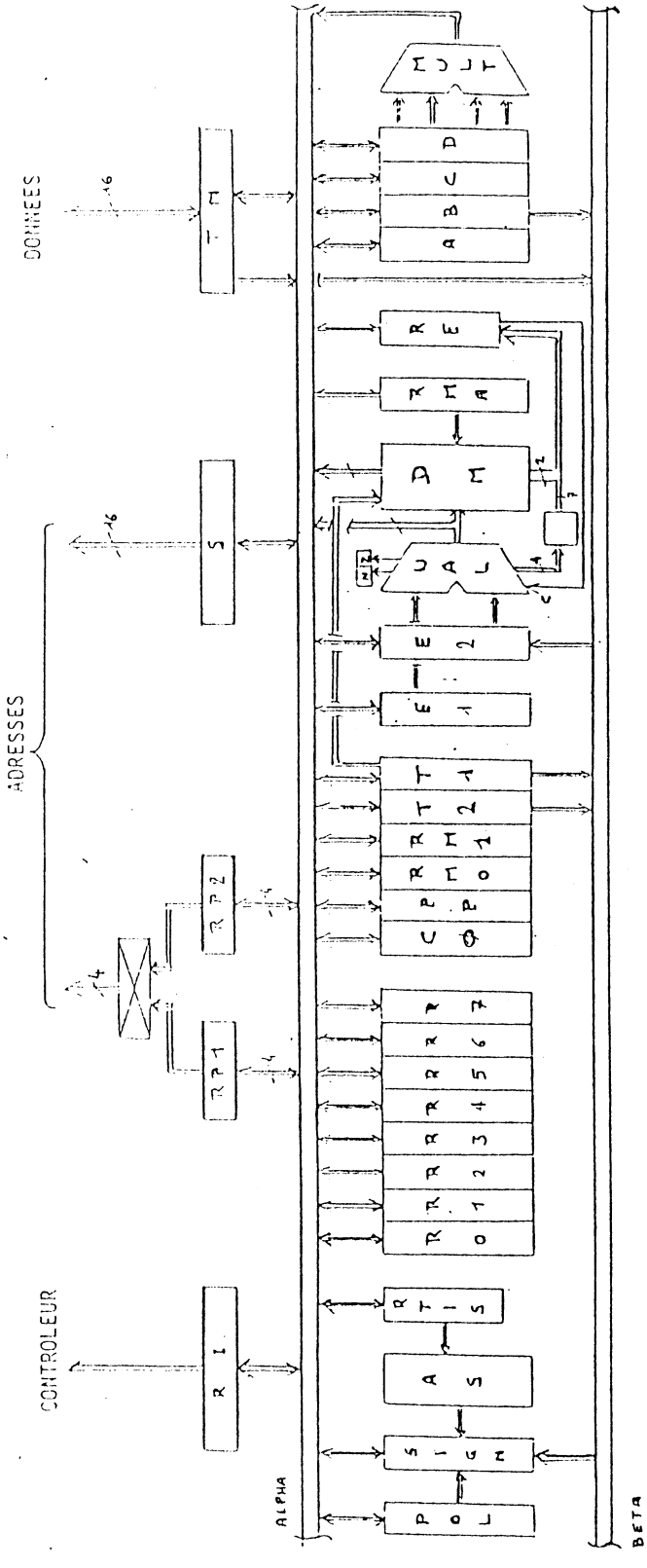


Figure 36 : Structure générale de la partie opérative

IV.3 - Réalisation de la partie opérative

Avant de développer la partie relative à la fabrication proprement dite des différents opérateurs de la P.O. du microprocesseur, il peut être utile, de soulever le problème relatif à la réalisation de circuits intégrés dans un environnement de recherche . En effet, la fabrication d'un circuit intégré est d'un coût relatif extrêmement élevé lorsqu'il s'agit de petites séries ou de circuits prototypes tels que les circuits de recherche ou d'enseignement. Le seul moyen accessible pour mener à bien la réalisation de tels circuits dans un environnement universitaire ou de recherche et de les fabriquer ensemble sur un même "wafer" afin de partager les frais fixes de réalisation (circuit multi projet [CMP]).

L'une des contraintes d'un tel système est d'imposer que tous les circuits soient réalisés dans la même technologie, or, à l'heure actuelle deux technologies semblent prépondérantes (N-MOS ou C-MOS), ainsi, pour ne pas avantager ou défavoriser l'une ou l'autre, les sessions de fabrication du CMP sont pour l'instant alternées : une fabrication C-MOS, une fabrication N-MOS,...avec une période d'environ 6 mois, ce qui représente pour une technologie donnée une étape de fabrication annuelle (deux, dans les meilleurs cas). Si l'on ajoute à ce fait la carence en outils de CAO efficaces adaptés à la technologie (vérificateur de régles de dessin, extracteur de schéma électrique...) on s'aperçoit qu'il n'est pas toujours aisé de faire une recherche appliquée dans un domaine tel que la conception de circuits intégrés.

Réalisation des éléments de la partie opérative

On présente ici, à titre d'exemple, l'approche suivie pour la réalisation du dessin des différents opérateurs de la Partie Opérative. La génération présentée ici est basée sur deux principes généraux simples :

i) le principe architectural simple constitué par l'existence de 2 doubles bus différentiels servant à connecter les divers registres et les opérateurs.
ii) le principe topologique simple qui consiste à définir des tranches standarts de hauteur fixe, définies au départ pour la technologie choisie et pour un nombre de lignes fixes supportant les alimentations, les bus et les connections de services internes à la partie opérative.

IV.3.1 - Structure d'une tranche

Chaque tranche de bit comprend :

- une ligne d'alimentation V_{DD}
- une ligne de masse GND
- deux lignes α et β (bus α et bus β)
- deux lignes $\overline{\alpha}$ et $\overline{\beta}$ (bus complementés)
- deux lignes de services, pouvant servir à des connexions internes au niveau de la P.O.

La technologie utilisée est la technologie CMOS 2 microns à deux niveaux d'aluminium du CMP 1985. L'ensemble des régles technologiques est donnée en annexe [3].

La présence de deux niveaux d'aluminium nous a permis de réaliser les lignes fixes de chaque brique à l'aide du 2ième niveau d'aluminium, laissant ainsi le niveau 1 d'aluminium entièrement libre pour les connexions internes à la brique, pour les connexions spécifiques locales entre deux briques voisines ou mieux pour les commandes verticales arrivant de la partie contrôle.

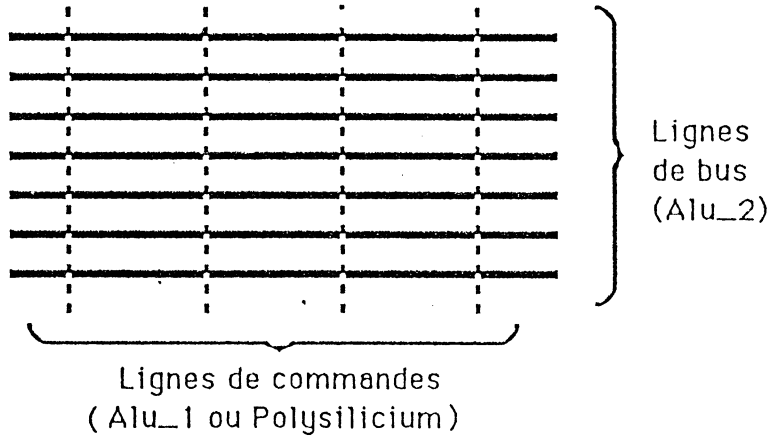


figure 1

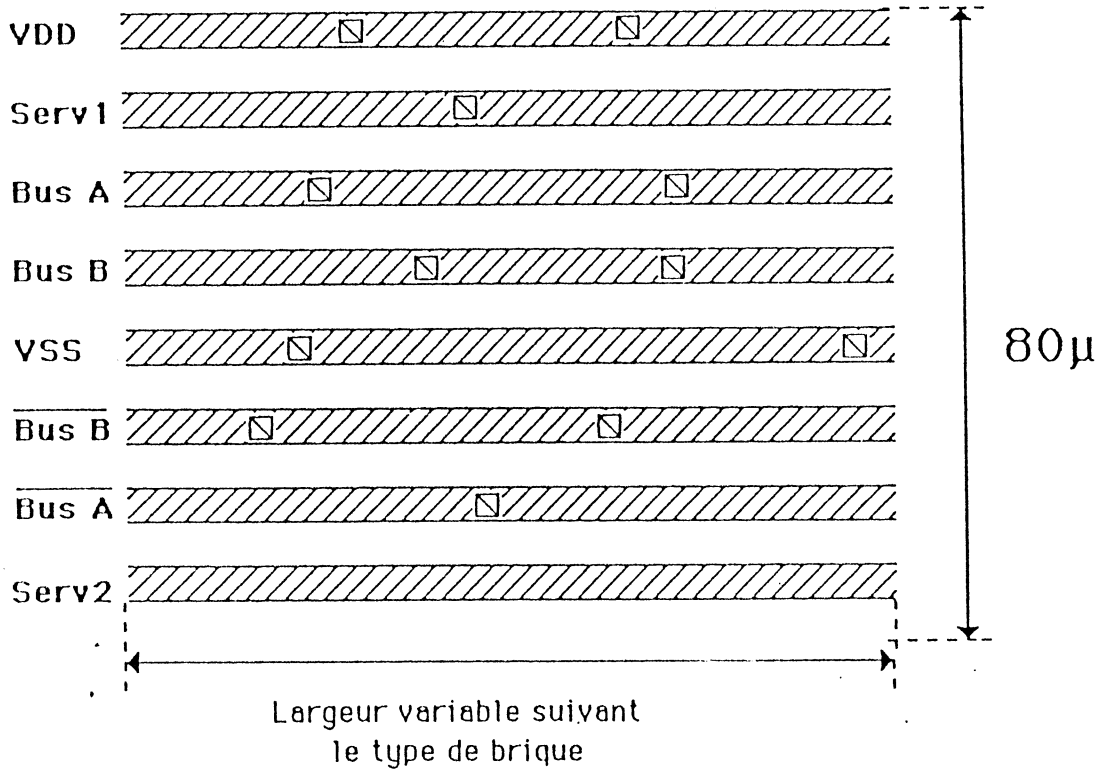
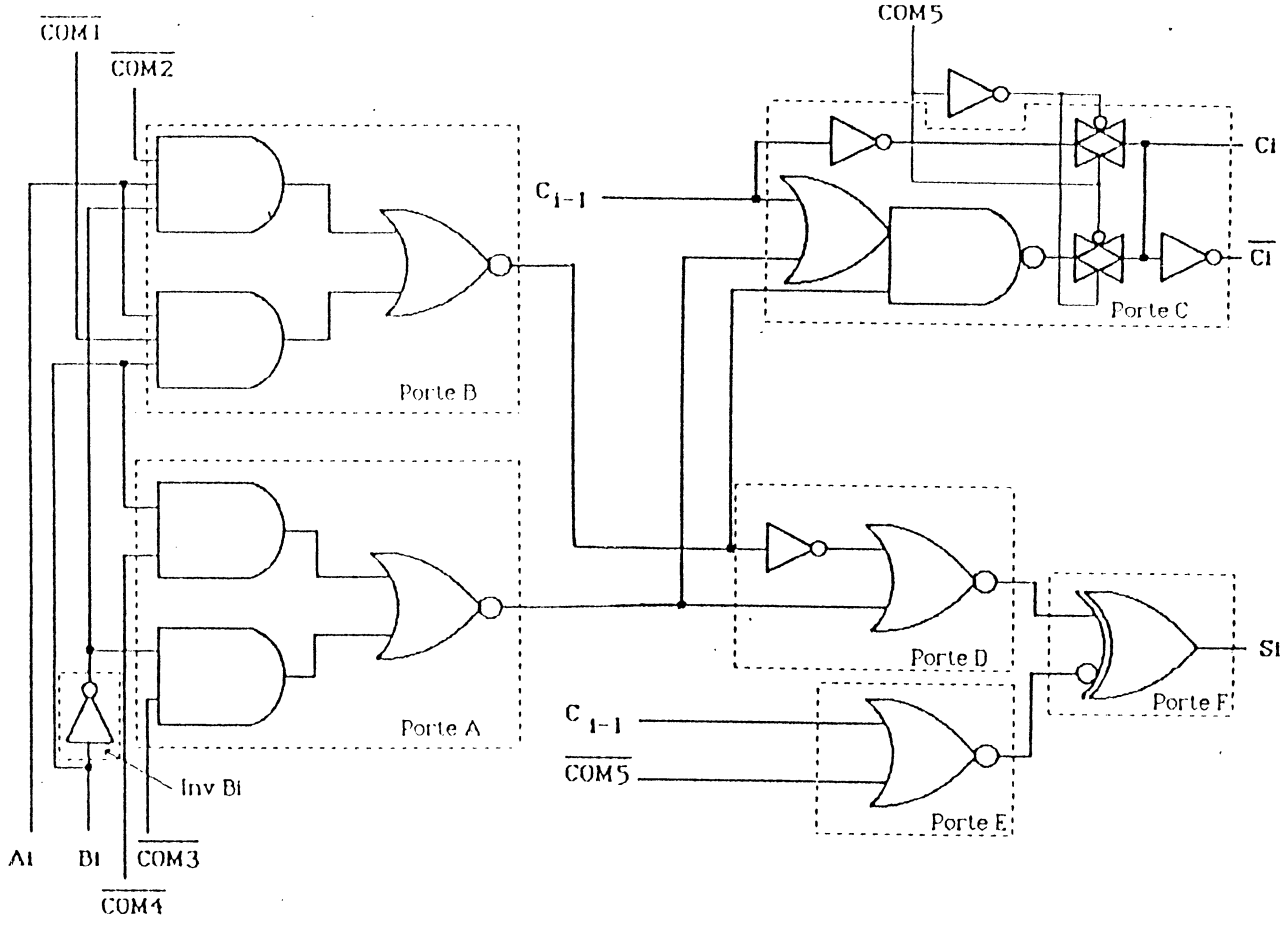


Figure 2

Disposition des lignes dans une tranche (niveau Alu 2)

IV.3.2 - Réalisation de l'U.A.L

Nous avons déjà donné une description de l'UAL au niveau logique dans le chapitre précédant. Afin de faciliter sa réalisation et sa simulation électrique, le schéma a été partitionné en portes complexes comme suit (Figure 3) :



L'UAL de 16 bits sera donc réalisée par l'association de cellules de base suivant l'organisation suivante (Figure 4) :

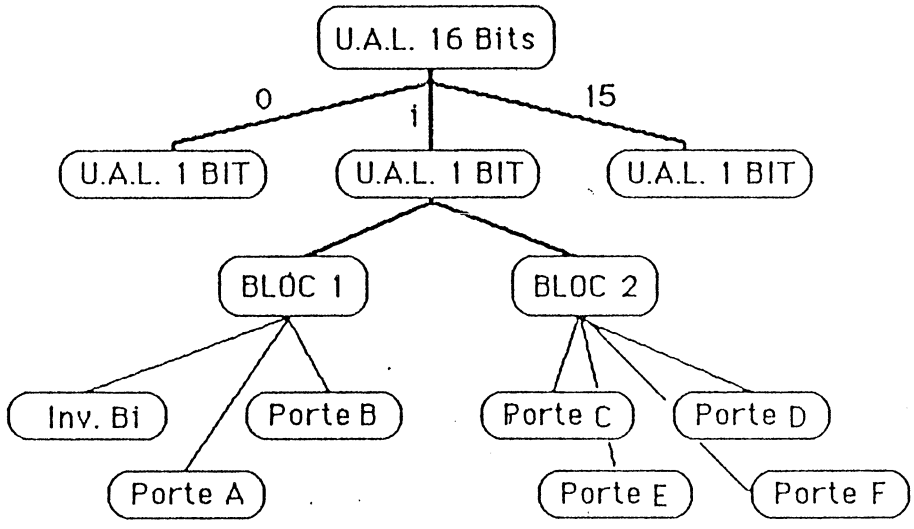


Figure 4

IV.3.3 - Implantation

Une cellule de base de l'Unité Arithmétique et Logique (sans son décodeur de commande) a été implantée. On donne figure 5 une représentation du dessin au niveau masque de cette cellule. Le niveau technologique permettant de

véhiculer les entrées, sorties et commandes sont les suivantes :

- BUS

  Vdd, Vss, α, $\bar{\alpha}$, β, $\bar{\beta}$ → Alu 2, 5μ.

- Commandes :

  COM 1 à COM → Alu 1, 4μ.
  COM 5 → Poly, 6μ.

- Entrées :

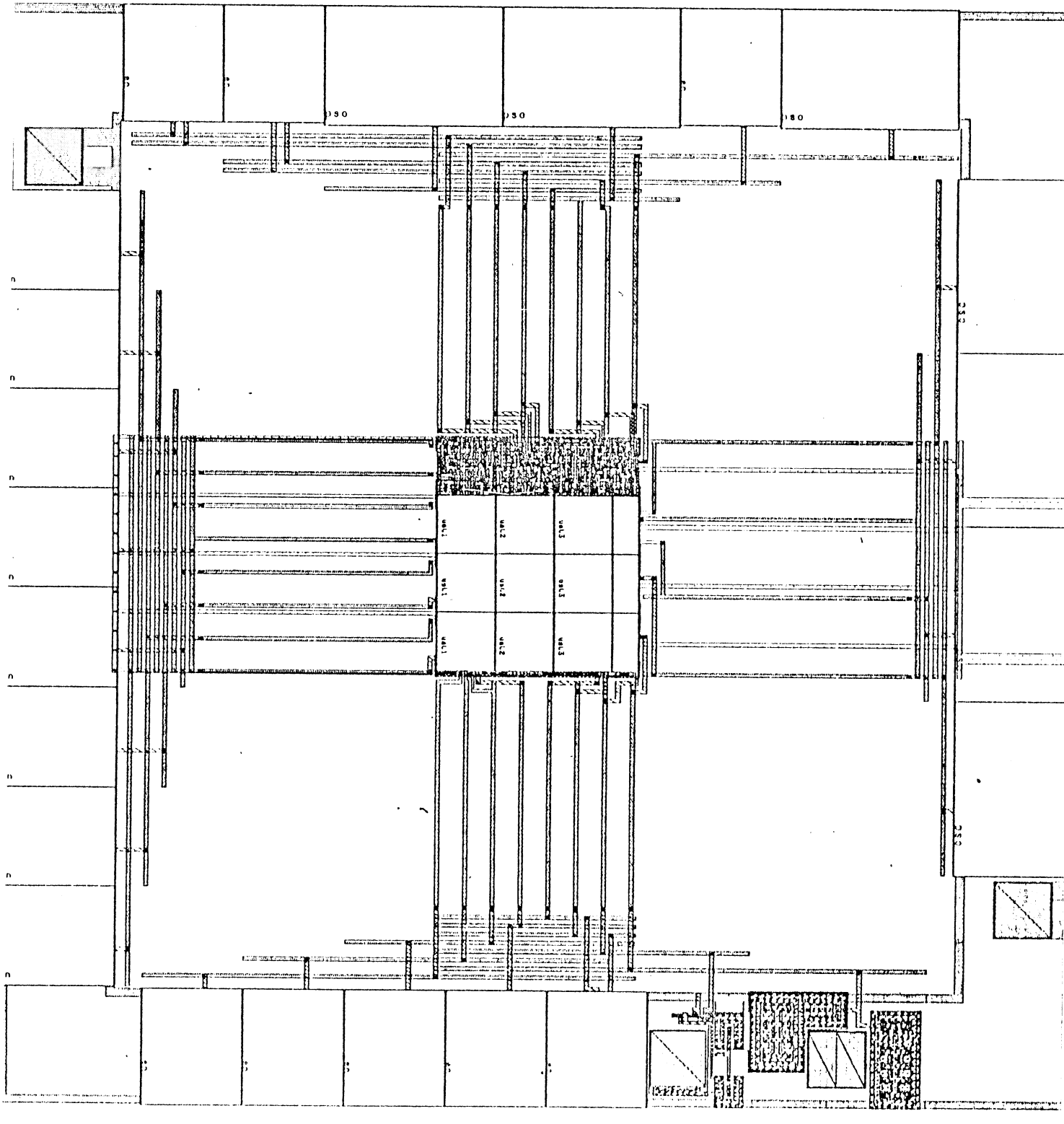
  Ai → Alu 2, 5μ.
  Bi → Poly, 6μ.

- Sorties :

  Si → diffusion N, 6μ et bus α Alu 2, 5μ.
  vers le disponible masquage.

* Réalisation d'un prototype

Un prototype du circuit a été réalisé. Le circuit conçu est une U.A.L 4 bits à laquelle nous avons relié différents points de test dans le but de vérifier son fonctionnement. On donne dans les pages suivantes une représentation du circuit qui sera réalisé dans le cadre du CMP-CMOS 1985.

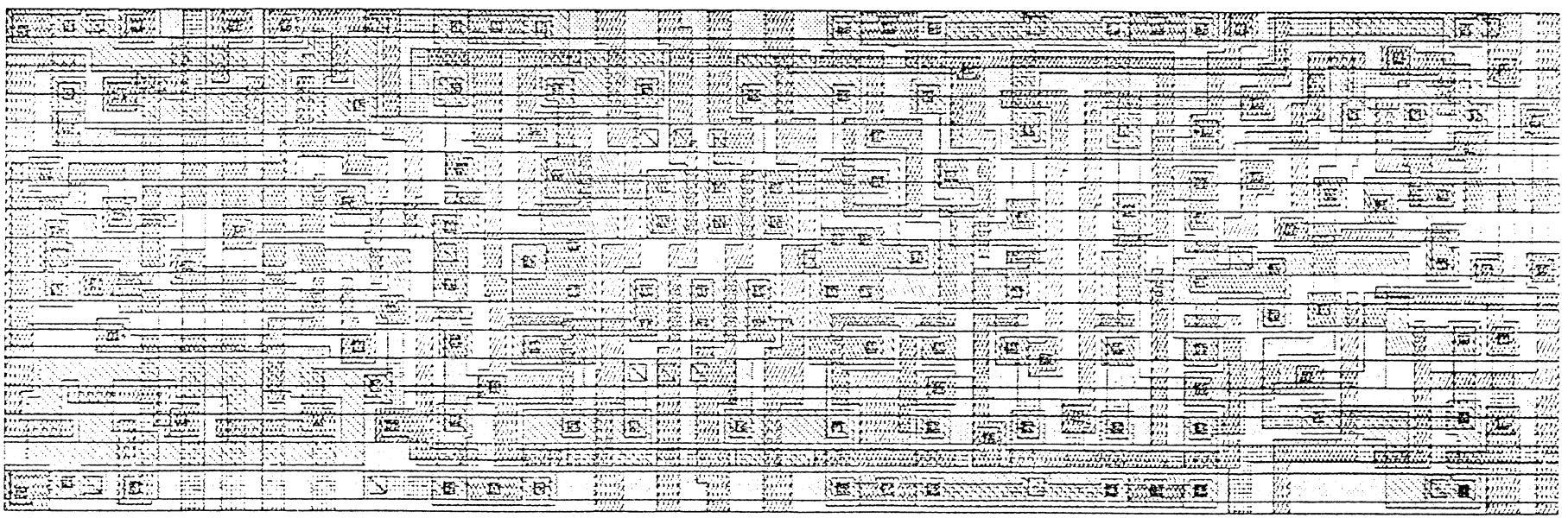


NOM:ual1b Echelle: 20 lam. par pouce

NIVEAUX md mb mc mp mm ms me mv

LUCIE : GPR100 Version TIM3 Fev 1985

Détail d'une tranche (1 Bit) de l'U.A.L. du prototype réalisé au CMP.



NUMBER OF VECTORS 0004727

IV.3.4 - Le dispositif de masquage

Ce dispositif n'a pas été implanté au niveau du silicium, on donne son schéma au niveau électrique ci-après (Figure 5).

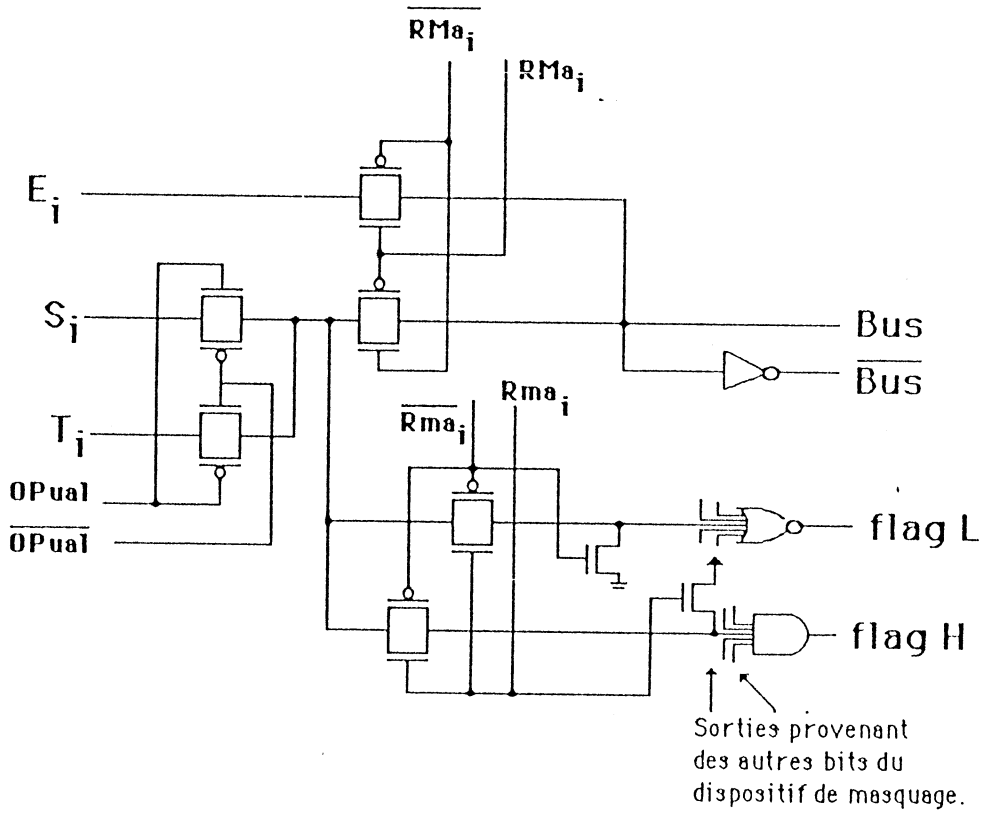


Figure 5

IV.3.4 - Le multiplieur

Comme vu précédemment, la cellule de base doit réaliser les fonctions suivantes :

$$Cout = Ai.Bj.(Cin + Din) + Cin.Din$$

$$Dout = Ai.Bj \oplus Cin \oplus Din$$

Pour faciliter l'implantation, la cellule a été partitionnée en différents blocs (Figure 6).

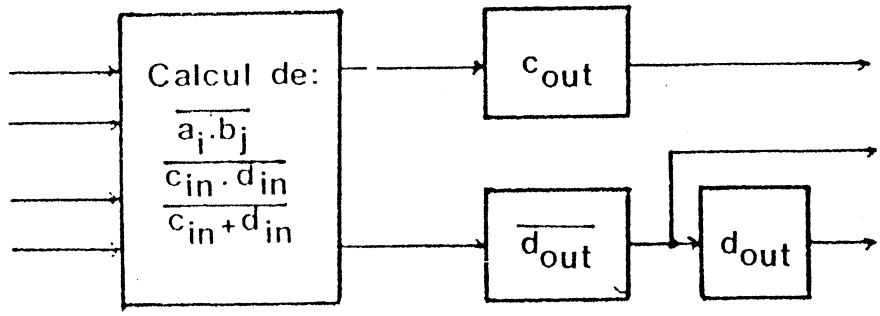
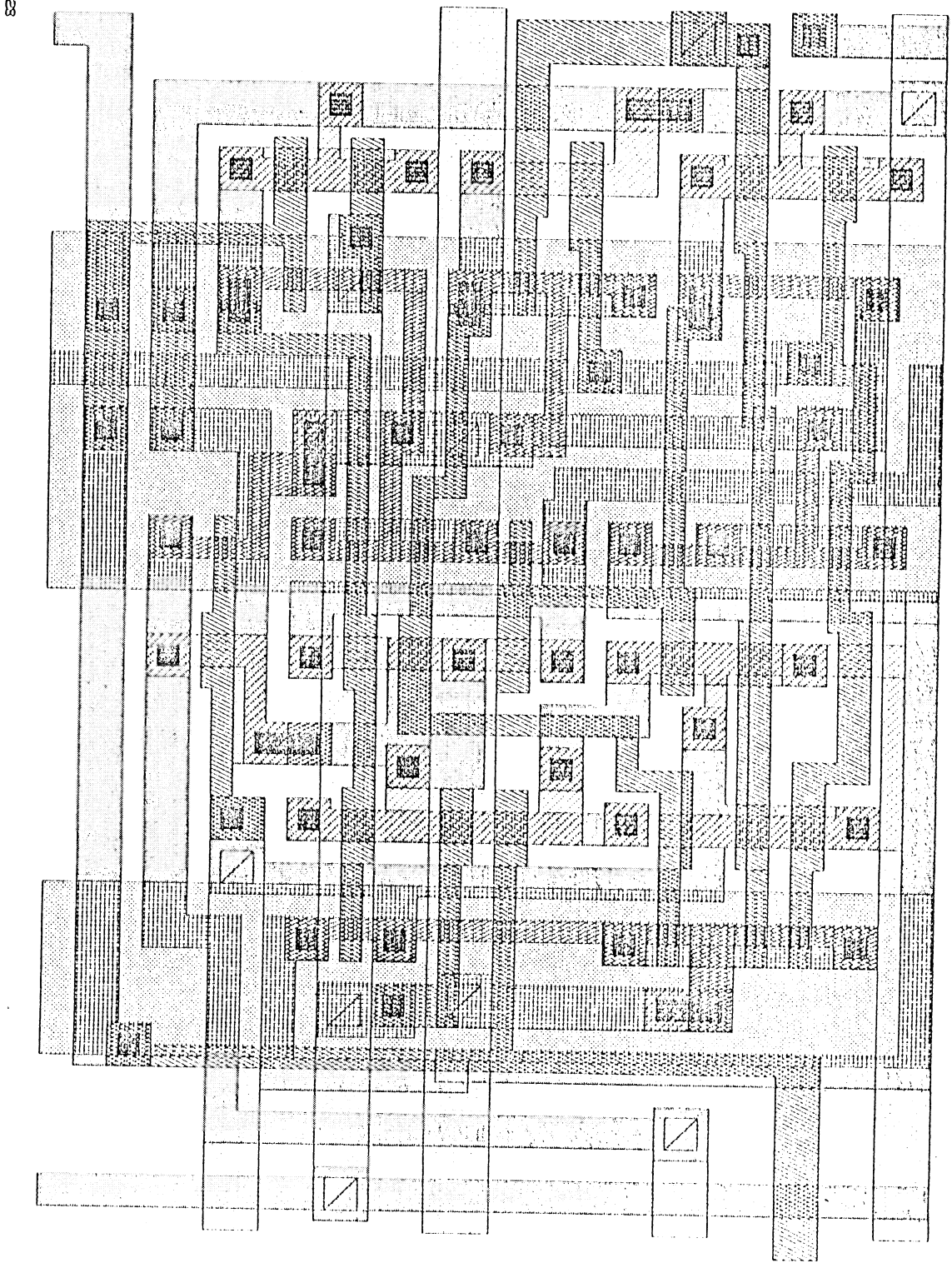


Figure 6

Un prototype du circuit consistant en un multiplieur 4Bits x 4Bits a été proposé au CMP-CMOS.

On trouvera ci-après une représentation de la cellule de base au niveau du masque.


NUMBER OF VECTORS 0003083

V - PARTIE CONTROLE DU MICROPROCESSEUR HSURF

V - INTRODUCTION

Le rôle de la partie contrôle d'un microprocesseur consiste d'une part à lire, décoder, exécuter les instructions et d'autre part à assurer la gestion des échanges avec l'extérieur à travers les broches de contrôle du circuit.

La réalisation du contrôleur de HSURF s'est orientée vers une réalisation microprogrammée qui permet de bénéficier des avantages suivants :

- facilité de conception,
- facilité d'implantation du fait de la régularité de la structure (par rapport à la logique aléatoire de la solution cablée),
- possibilité d'utiliser des programmes de génération automatique
- implementation facile d'algorithmes d'interprétation compliqués,
- facilité de mise en oeuvre des dispositifs de test.

V.1 - PRINCIPE DE LA MICROPROGRAMMATION

L'algorithme d'interprétation du jeu d'instructions du microprocesseur est décrit sous la forme d'un microprogramme qui sera stocké dans une ROM. A chaque état de l'algorithme correspond une micro-instruction qui contient deux types d'informations :

- des informations sur les commandes à envoyer à la P.O.
- des informations nécessaires au calcul de l'adresse de la micro-instruction suivante.

La partie contrôle microprogrammée comporte donc les éléments suivants :

- une ROM qui contient les commandes à générer pour la P.O. et les informations utiles au séquencement des micro-instructions.

- un SEQUENCEUR qui, à partir des informations données dans la micro-instruction, calcule l'adresse de la micro-instruction suivante.

- Dans le cas de l'utilisation de sous-programmes : une PILE DE SAUVEGARDE de la micro-adresse.

- Un MICRO-COMPTEUR ORDINAL contenant l'adresse de la micro-instruction courante.

- Un REGISTRE DE MICRO-INSTRUCTION dans lequel se trouve le contenu de la micro-instruction en cours d'exécution.

Ceci est représenté par le schéma suivant (Figure 1) :

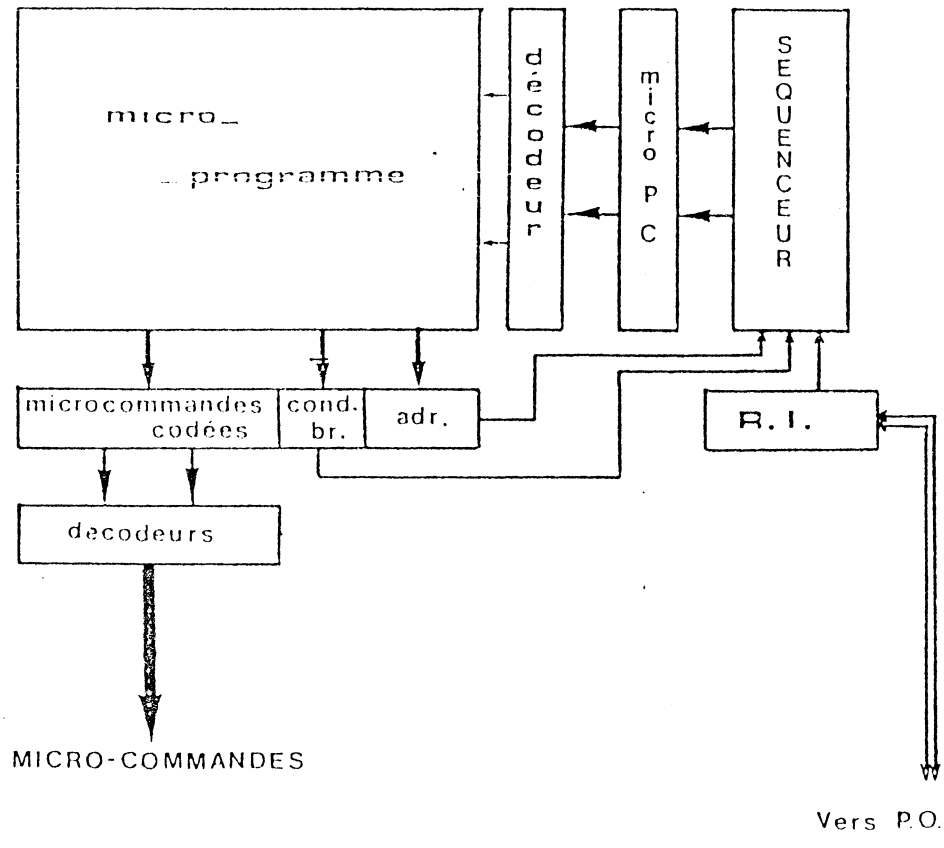


Figure 1

V.2 - GENERATION DU CONTENU DE LA R.O.M. ET DU CONTROLEUR

L'obtention d'une partie contrôle microprogrammée à partir de sa description algorithmique demande certaines adaptions au niveau de cette description afin de pouvoir générer le contenu de la ROM et du dispositif de séquencement :

- La notion d'état de l'algorithme est remplacée par la notion d'adresse de la micro-instruction qui implémente cet état.

- Le format de la micro-instruction est défini : il permettra de traduire l'algorithme déja connu sous forme d'organigramme en un microprogramme.

V.3 - TEST DE LA PARTIE CONTROLE

Avec une architecture équivalente à celle de la figure précédente, le test de la partie contrôle ne peut être réalisé que de façon indirecte par observation des modifications crées au sein de la partie opérative (ensemble de ressources contrôlées). Le test d'un contrôleur à travers les ressources qu'il contrôle a été étudié dans [ROB 79]. Il s'avère que quelles que soient les méthodes employées, une telle stratégie sera toujours délicate à mettre en oeuvre, le nombre de vecteur de test (longueur des programmes de test écrit dans le langage de la machine) devenant rapidement très important même pour détecter des défaillances simples.

Dans [THA 80] on donne un exemple de la complexité de génération de programmes permettant de détecter seulement des défauts dans le registre instruction et dans sa logique de décodage. Certains autres types de défaillances tel qu'un séquencement incorrect ou exécution partielle d'instruction demandent des programmes de test d'une complexité telle que leur génération s'avère encore plus délicate voire impossible. De plus si l'observation des résultats sur les ressources contrôlées (partie opérative) fait apparaitre une défaillance, il sera très difficile de localiser si l'organe créateur du défaut se situe au niveau de la partie contrôle, de la logique d'interface partie contrôle/partie opérative ou de la partie opérative ; le diagnostic d'erreur s'avérera donc très délicat.

Il apparait donc impératif, pour réaliser facilement un test complet du contrôleur, de pouvoir "converser" directement avec ce dernier. L'ajout d'accès d'observation et de contrôle directement au niveau de la partie contrôle permettra donc de rendre son test aussi indépendant que possible des ressources de la partie opérative.

V.4 - DISPOSITIFS PERMETTANT DE FACILITER LE TEST DE LA PARTIE CONTROLE

L'observation du schéma général (figure 1) permet de définir les points où circulent des informations stratégiques et dont l'observation facilitera le test du contrôleur.

Ces informations sont de deux types :

- les données de commandes : signaux à destination de la partie opérative (microcommandes)

- les données de contrôle et de séquencement permettant le calcul de l'adresse suivante dans la ROM de microcommande (microadresse).

L'accès aux informations présente en ces deux points permet :

i) Une vérification du bon séquencement des micro-instructions (accès au niveau du microcompteur ordinal)

ii) Une vérification de la validité des commandes envoyées à la partie opérative. Ici, l'accès sera donné après les décodeurs de microcommandes permettant ainsi de vérifier l'ensemble registre micro instruction (μRI) et logique de décodage.

En fonctionnement normal (exécution d'un programme) la quantité d'information circulant en ces nouveaux points d'observation est beaucoup trop importante pour que l'on puisse en réaliser une vérification continue (vérification de chacune des microadresses ou microcommandes), il a donc été choisi de réaliser une compaction du flot de données en utilisant le même principe que pour la Partie Opérative (Registres de signature par division polynomiale).

Le contenu des deux registres de la partie contrôle sera accessible par l'utlisateur de façon directe de la même façon que pour l'ensemble des registres de la partie opérative. Cet impératif d'accessibilité s'applique aussi à tous les registres internes du contrôleur (RI, μCO, μRI).

L'ensemble des dispositifs permettant de faciliter l'observation et le test d'une partie contrôle microprogrammée de HSURF sont résumés dans la figure suivante (Figure 2).

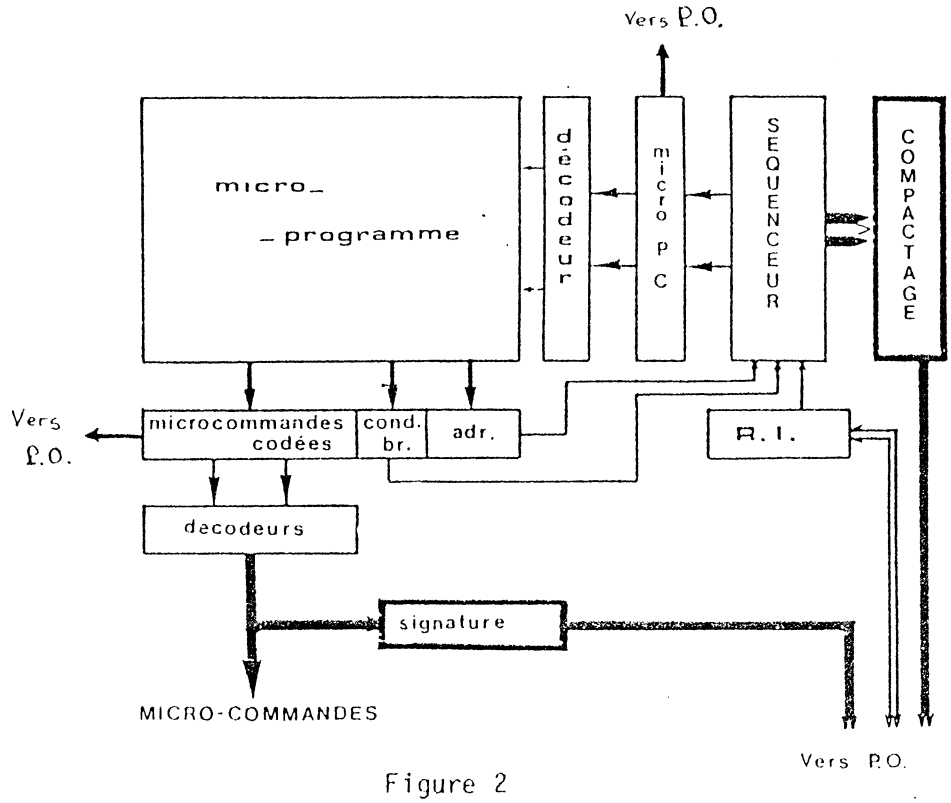


Figure 2

V.5 - REALISATION DE LA PARTIE CONTROLE DE HSURF

V.5.1 - Algorithme d'interprétation du jeu d'instruction de HSURF

Le graphe de contrôle de HSURF est donné en annexe 4. Il comprend 140 états. Sa structure générale est représentée figure 3.

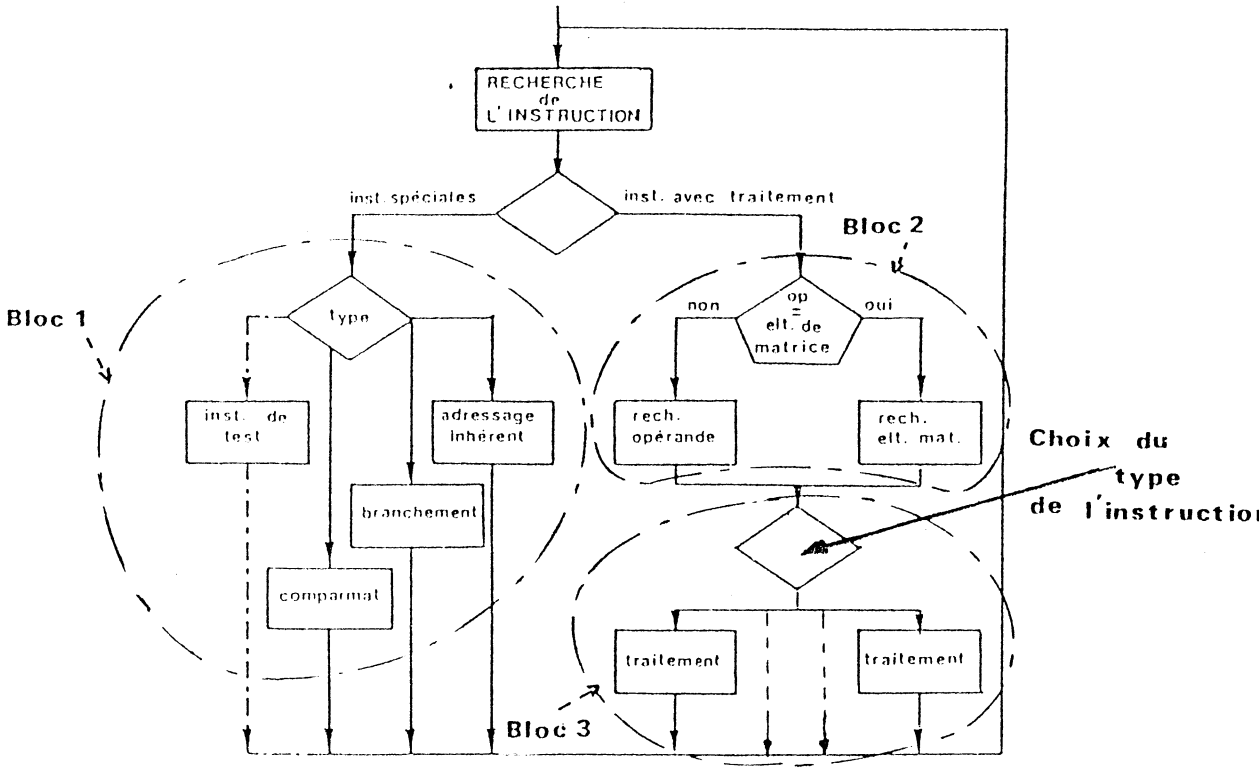


Figure 3

Le bloc 1 traite les instructions avec un opérande implicite, les instructions de test, de branchement et l'instruction COMPARMAT

Le bloc 2 réalise la recherche des opérandes pour les instructions classiques de traitement de tableau.

Le bloc 3 assure l'exécution des opérandes logiques arithmétiques et de transfert.

L'utilisation de niveaux de sous microprogramme permet d'éviter la répétition d'une même micro-instruction et par suite de diminuer leur nombre.

V.5.2 - Implantation du contrôleur de HSURF

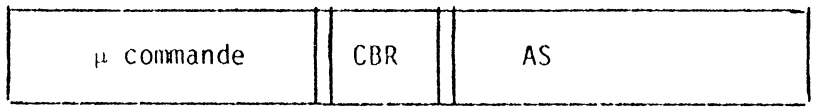
Le schéma général du contrôleur de HSURF est donné figure Y. Il est organisé autour des ressources suivantes.

V.5.2.1 - ROM

- Le contrôleur de HSURF est organisé autour d'une microprogrammation horizontale. A chaque état de l'automate de contrôle correspond donc une ligne de la ROM. Dans chaque ligne on dispose de deux zones, l'une contient les microcommandes à destination de la partie opérative, l'autre les informations nécessaires au séquencement des micro-instructions (champs AS et CBR) : champ adresse suivante et condition de branchement.

- Structure d'une micro-instruction

  La structure d'une micro-instruction est la suivante :



Micro-séquencement

- champ adresse suivante (A.S) : Ce champ est utilisé dans le cas d'un branchement inconditionnel ou d'un saut à un sous microprogramme, il contient l'adresse de la micro-instruction qui doit être exécutée après l'exécution de la micro-instruction courante. Le nombre de micro-instruction étant de 140, nous avons besoin d'un champ de 8 bits.

- champ condition de branchement (CBR) : ce champ est utilisé pour sélectionner la source de l'adresse de la micro-instruction suivante. En fonction de son codage, on sélectionnera soit le champ AS, soit la sortie du PLA de branchement, soit la sortie d'un des PLA de décodage (PLA A1 ou A2).

- champ microcommande : le contenu de ce champ indique les différentes commandes à envoyer à destination de la partie opérative : commande d'accès aux registres, commandes d'opération de l'UAL, ... (cf. Annexe 4).

V.5.2.2 - Micro-contrôleur de séquencement

Dans le cas qui nous intéresse (microprocesseur), l'algorithme de contrôle est complexe et comporte beaucoup de branchements, or, l'utilisation de "circuits d'éclatement" (mapping) s'avère délicate à mettre en oeuvre quand le nombre de test (et donc de "mapping") et important (le résultat aboutissant à un très mauvais remplissage de la ROM).

Nous avons donc opté pour des solutions donnant une plus grande souplesse dans l'implantation du microprogramme. Les séquenceurs seront réalisés à l'aide de PLAs, ce qui de plus permettera d'utiliser des outils de synthèse et de compaction automatiques.

Il existe deux solutions :

V.5.2.2.1 - Micro-contrôleur à PLA unique :

Le micro-contrôleur est réalisé à l'aide d'un seul PLA qui traite tous les branchements. Il reçoit donc en entrée toutes les informations nécessaires au séquencement :

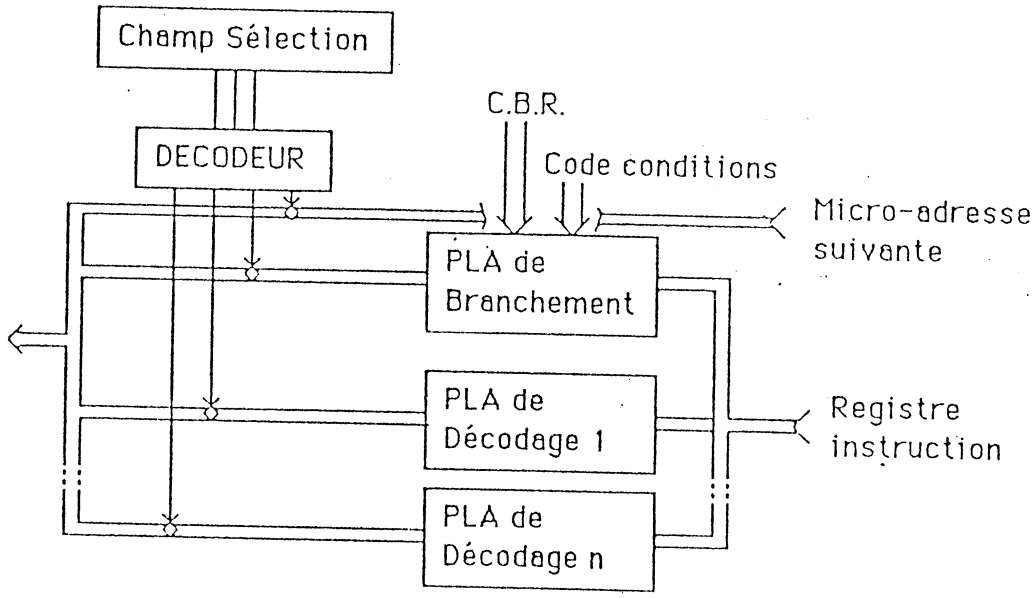
- Champs adresse suivante et conditions de branchement (CBR) de la micro-instruction.

- Bits à tester dans le registre instruction RI.

- Code conditions (registre d'état).

En sortie, il génère l'adresse de la micro-instruction suivante ainsi que les commandes de la pile s'il y en a une.

V.5.2.2.2 - Séquenceur MULTI - PLAs



Ici, le but est d'être indépendant de l'adresse courante pour avoir moins d'entrées dans les PLAs. Un tel séquenceur se divise en deux blocs principaux :

- Un ou plusieurs PLAs de décodage qui réalisent essentiellement les éclatements sur les bits du Registre d'Instruction vers les différentes branches des blocs de l'organigramme.

- Un PLA de branchement qui réalise les branchements locaux dans l'organigramme, en particulier les branchements résultant du contenu des bascules d'état. Chaque test étant repéré de façon unique par son numéro, il n'est pas nécessaire d'utiliser ici le champ micro-adresse suivante.

Il faut rajouter un champ SEL dans la partie séquencement de la micro-instruction pour sélectionner la source de l'adresse de la micro-instruction suivante (un des PLAs ou le champ adresse suivante de la micro-instruction courante).

Principe de fonctionnement

La partie séquencement de la micro-instruction sert à générer la micro-adresse suivante :

- soit par le PLA de branchement si on a des résultats de la P.O. ou des conditions particulières à tester.

- soit par un des PLAs de décodage qui fournit l'adresse du microprogramme de traitement à partir du code de l'instruction.

- soit en prenant directement le champ adresse suivante de la micro-instruction ceci pour un branchement inconditionnel.

La solution multi-plas permet d'obtenir pour chaque "sous-plas" un remplissage meilleur (d'où une meilleure utilisation de la surface de silicium). C'est cette solution qui a été retenue pour la réalisation du microprocesseur HSURF.

Le découpage des divers plas est en relation directe avec l'organisation de l'organigramme de contrôle (cf fig 3), nous avons :

* PLA A1 il réalise le groupe des premiers branchements juste après la séquence de recherche de l'instruction suivante (Fetch) et avant les blocs 1 et 2.

* PLA A2 : il réalise l'aiguillage situé en amont du bloc 3 (Choix du type de l'instruction).

* PLA de branchement : son rôle est de réaliser les aiguillages locaux (branchement à l'intérieur des blocs) résultant d'opérations de test exécutées dans la P.O. (branchements fonction des différents comptes-rendus et indicateurs de la partie opérative).

Le contenu complet de ces 3 PLAs est donné en annexe 4.

Ainsi, l'adresse de la micro-instruction suivante aura pour origine possible :

* L'un des 3 PLA ou
* le champ AS de la micro-instruction courante.

V.5.2.3 - Pile de microcontrôleur

Afin de réduire le nombre de micro-instruction (donc de ligne dans la ROM de micro programme), il est fait appel à des sous programmes permettant de ne coder qu'une seule fois un ensemble de micro-instructions exécutées à plusieurs reprises en différents points de l'organigramme de contrôle.

L'implantation de niveaux de sous microprogrammes implique l'utilisation d'une pile afin de stocker les microadresses de retour. L'organisation du microprogramme dans HSURF ne nécessite que 2 niveaux au maximum d'appel de sous microprogrammes, il nous a donc été possible de réaliser la pile de façon cablée, comme indiqué ci-après (Figure 5).

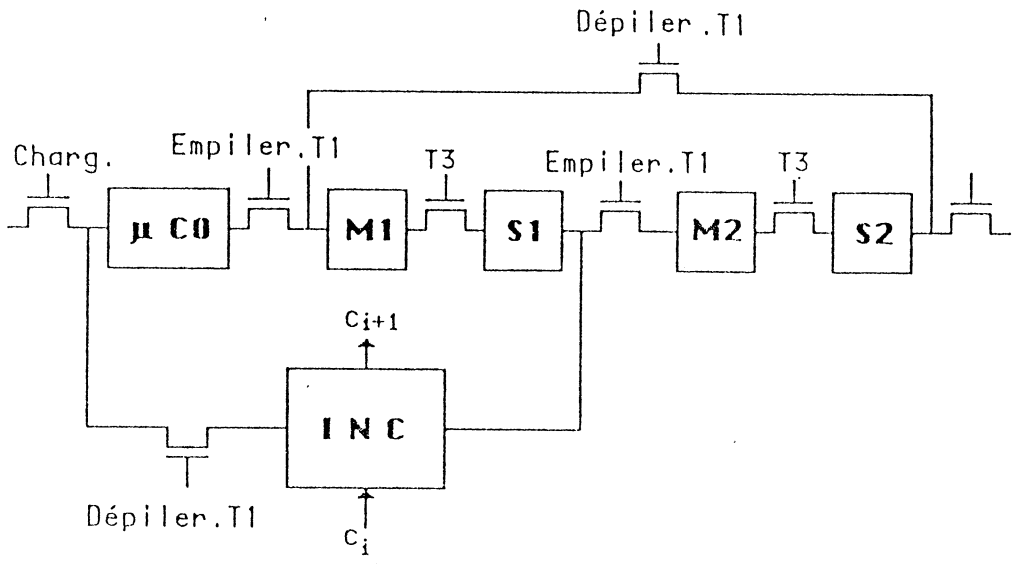


Figure 5
Schéma de la pile câblée

Le principe de cette pile est basé sur celui d'un registre à décalagte bidirectionnel. Lors d'une commande d'empilement, on opère un transfert des bits du micro compteur ordinal vers un registre de sauvegarde (cette opération est équivalente à effectuer un décalage à droite des bits du micro compteur ordinal, le chargement du registre maitre est réalisé pendant T1 et celui du registre esclave pendant T3). Lors d'une commande de dépilement, une opération de décalage à gauche est effectuée. Une particularité du dispositif implanté permet de réaliser l'incrémentation du contenu de la base de la pile pendant la phase de dépilement (retour d'un sous microprogramme), le microcompteur ordinal est automatiquement chargé avec l'adresse + 1 présenté au moment de l'appel.

V.5.2.4 - Micro-compteur ordinal (μCO)

Sa taille est donnée par le nombre de micro-instructions, ici elle sera de 8 bits.

V.5.2.5 - Registre de micro-instruction

Sa taille est identique à la largeur de la ROM du microprogramme, soit 14 bits. (8 pour le champ AS et 6 pour le champ CBR) pour la partie microséquencement et 26 pour la partie micrommandes.

V.6 - CONCLUSION

Il apparait que le contrôleur de HSURF ne présente pas d'originalité quant à son architecture générale. Les efforts fait lors de sa conception ont surtout contribués à satisfaire un impératif d'accessibilité et d'observabilité meilleur que pour la plupart des composants courants du type microprocesseur.

L'objectif principal était de permettre un test de cette partie de la façon la plus indépendante possible de la partie opérative. Pour arriver à un tel objectif, nous avons ajouté à la partie contrôle les points d'observation précisés précédemment sur la figure 2. Parallèlement à ces modifications matérielles, il a été nécessaire de prévoir des facilités au niveau logiciel se traduisant par l'ajout des instructions spécifiques suivantes :

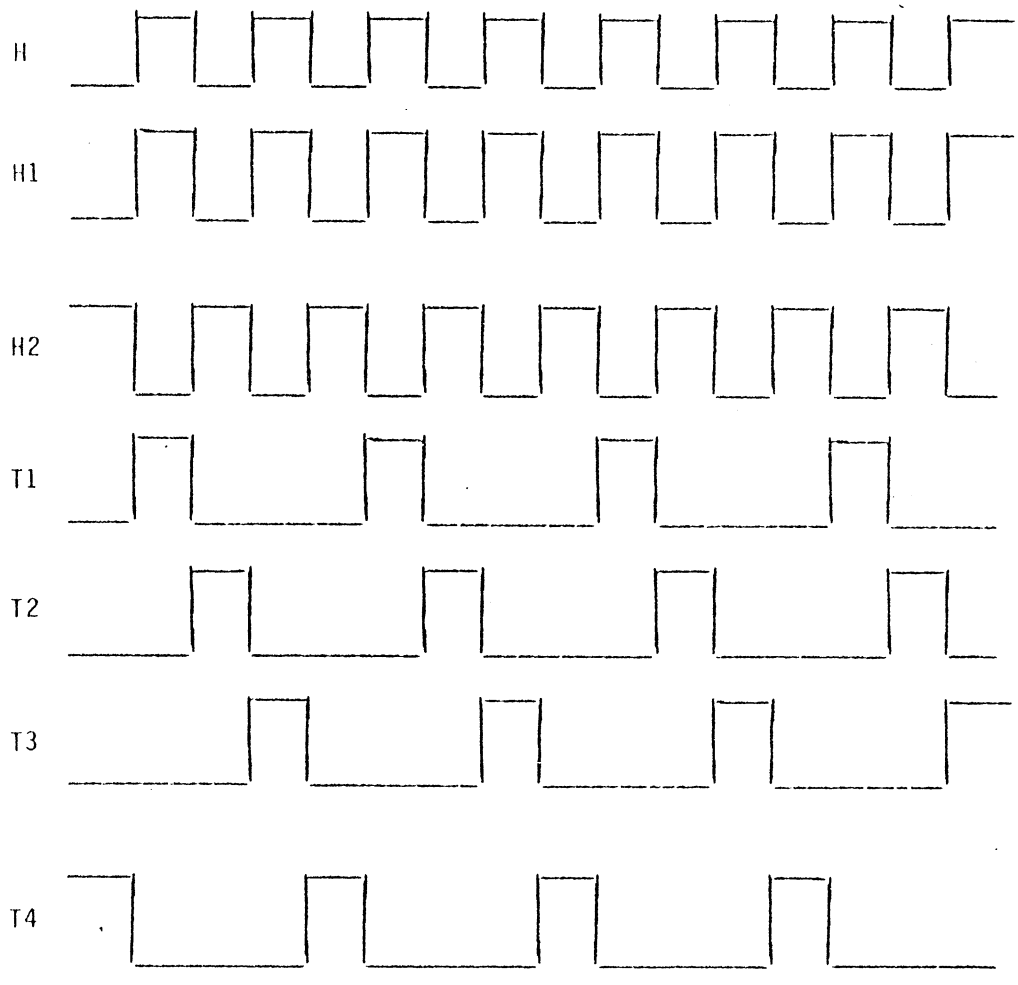
- Instruction RAZ : Remise à zéro des registres de signature.
- Instruction TRC : Test des registres du controleur.
- Instructions LRS (Load) et SRS (Store) destinées aux registres spéciaux du microprocesseur non accessible directement par les instructions normales de chargement et de lecture.

VI - TEMPORISATION DU MICROPROCESSEUR HSURF

VI - INTRODUCTION

Afin de pouvoir mieux comprendre les différents schémas logiques concernant les dispositifs de compaction et l'interface de commande P.C./P.O., nous donnons dans ce paragraphe une description des différents signaux d'horloge nécessaires au séquencement des micro-instructions au sein de HSURF.

A partir d'une horloge externe de base H, nous générons les signaux internes suivants :



Les signaux H1 et H2 sont sans recouvrement.

VI.1 - DEROULEMENT DES MICRO-INSTRUCTIONS

Un cycle d'exécution d'une micro-instruction se réalise en quetre phases (T1 à T4);

Au niveau de la partie contrôle, il se décompose en deux cycles d'horloge :

- Un cycle de tracersée des PLAs de séquencement et du décodeur de micro-adresse (T1 et T2).
- Un cycle de traversée de la ROM et des décodeurs de micro-commandes (T3 et T4).

Chaque cycle d'horloge se décompose lui même en deux phases :

- Une phase de précharge (T1 ou T3).
- Une phase de traversée des blocs fonctionnels (T2 ou T4).

Chacune des deux parties du microprocesseur (P.C. et P.O.) envoie et reçoit des informations. La P.C. envoie les microcommandes à la P.O., elle reçoit les compte-rendus de la P.O. et échange des informations avec l'extérieur. La P.O. envoie les compte-rendus à la P.C. et reçoit les micro-commandes de la P.C..

Les informations échangées avec l'extérieur sont : les signaux ALARME et RESET, les signaux de synchronisation pour les opérations de lecture/écriture et d'échanges avec les périphériques (signal VMA, R/W).

Les opérations réalisées au niveau de la partie opérative et de la partie contrôle durant les quatres phases T1 à T4 sont résumées page suivante.

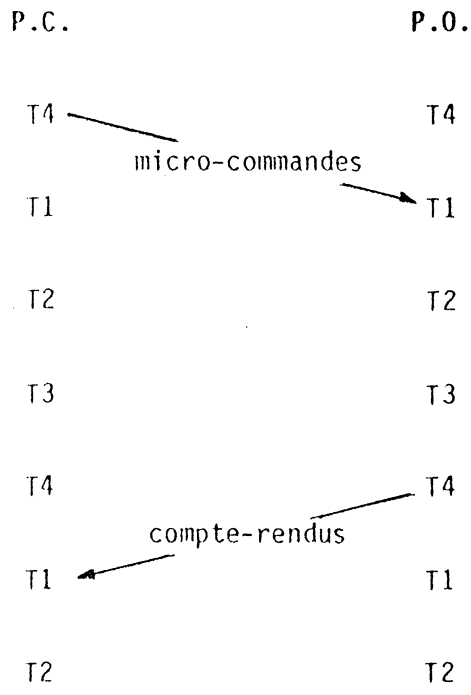
PARTIE CONTROLE		PARTIE OPERATIVE
Prise en compte éventuelle des indicateurs N et Z de la partie opérative, du signal RESET ou ALARME. Calcul de l'adresse de la micro-instruction suivante		Prise en compte des micro-commandes venant de la partie contrôle. Début de l'exécution des micro-commandes.

En T1, échange des informations entre la partie contrôle et la partie opérative.

Pendant les trois phases suivantes, la partie opérative et la partie contrôle travaillent séparément.

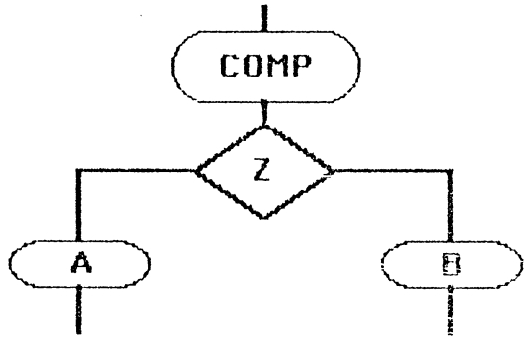
Traversée de la ROM qui délivre une nouvelle micro-instruction et traversée des décodeurs de micro-commandes. A la fin de T4, les nouvelles micro-commandes sont disponibles.		Poursuite et fin de l'exécution des micro-commandes. A la fin de T4, les nouvelles valeurs des indicateurs N et Z sont disponibles.

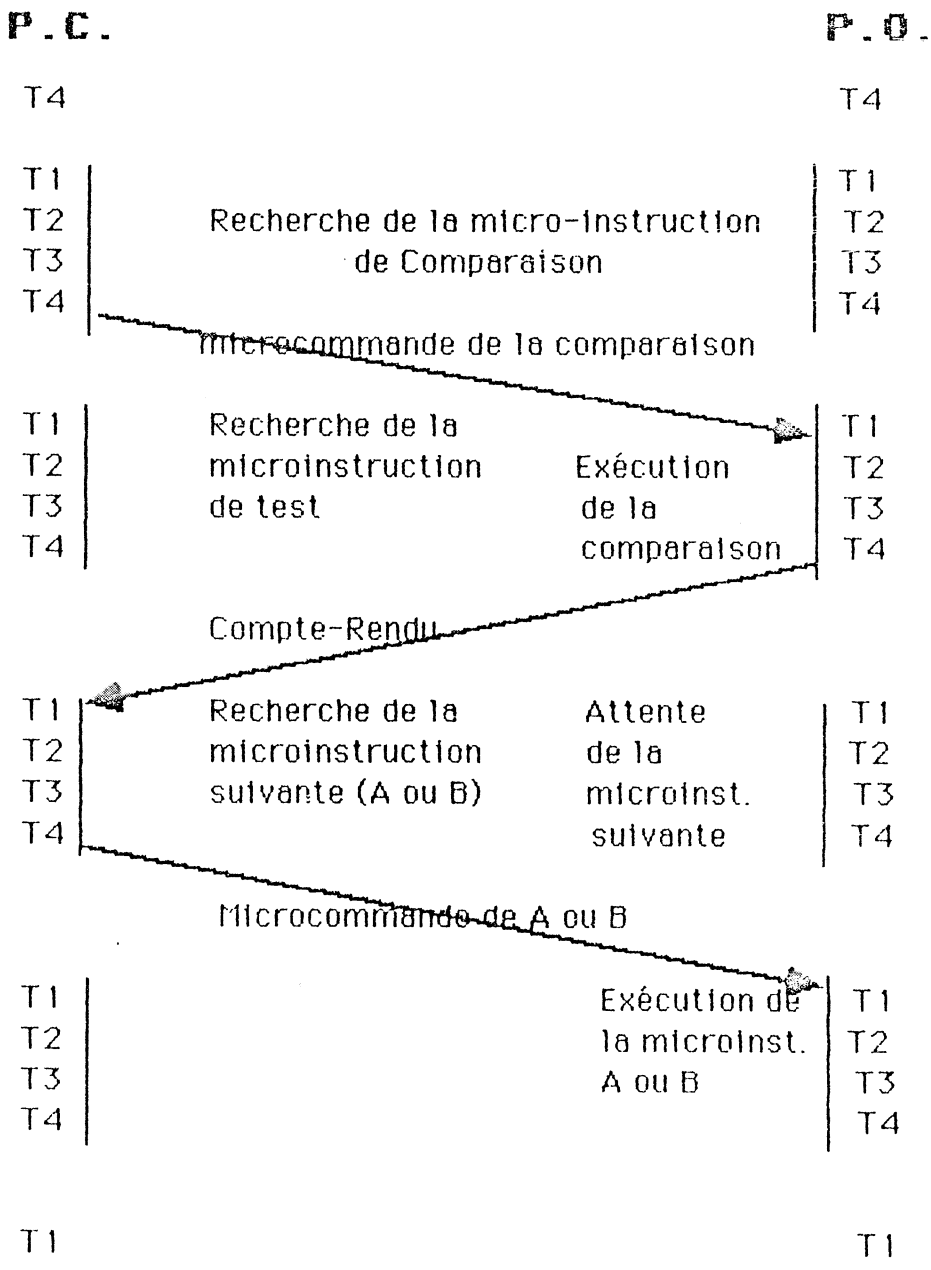
Dans le diagramme suivant, les flèches représentent les échanges entre la partie contrôle et la partie opérative :



Une commande envoyée pendant une phase T4 par la partie contrôle est reçue pendant la phase T1 suivante par la partie opérative. De même, le compte rendu envoyé pendant la phase T4 suivante par la partie opérative sera reçu par la partie contrôle pendant la phase T1 suivante.

L'exécution d'une micro-instruction sans demande de cycle mémoire et sans multiplication se déroulera comme indiqué dans la page suivante.


COMP
Z
A
B


P.C.
P.O.
T4
T4
T1
T2
T3
T4
Recherche de la micro-instruction
de Comparaison
T1
T2
T3
T4
Microcommande de la comparaison
T1
T2
T3
T4
Recherche de la
microinstruction
de test
Exécution
de la
comparaison
T1
T2
T3
T4
Compte-Rendu
T1
T2
T3
T4
Recherche de la
microinstruction
suivante (A ou B)
Attente
de la
microinst.
suivante
T1
T2
T3
T4
Microcommande de A ou B
T1
T2
T3
T4
Exécution de
la microinst.
A ou B
T1
T2
T3
T4
T1
T1

Cycles mémoire

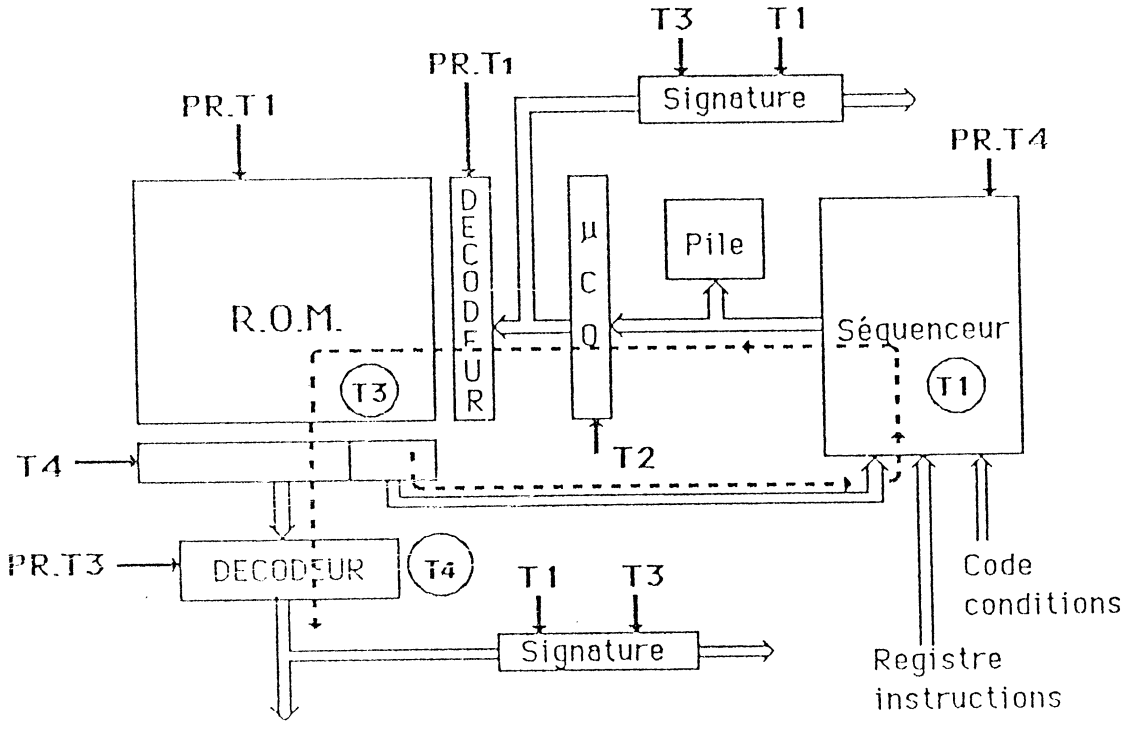
Le cycle mémoire pourra être demandé dès que les registres concernés sont correctement initialisés (registre S pour les opérations de lecture et registres s et TM pour les opérations d'écriture). Les signaux vma (Valid memory Address) sont générés.

Multiplication

L'opérateur MULT effectue en permanence l'opération de multiplication avec les données présentes dans les registres A, B, C et D de la partie opérative. Dès que ces registres contiennent les valeurs voulues, la multiplication désirée commence. Le résultat de cette multiplication sera transféré dans le registre destination pendant la phase T3 de la même manière que pour une opération UAL.

VI.2 - TEMPORISATION DE LA PARTIE CONTROLE

Le schéma suivant représente la partie contrôle temporisée :



Les blocs fonctionnels sont préchargés durant la phase précédant leurs modifications avec de nouvelles informations. Ainsi, les PLAs du séquenceur sont préchargés pendant la phase T4 et la nouvelle micro-adresse est calculée pendant T1. Le décodeur et la ROM sont préchargés en T1. Dès que le registre micro-compteur ordinal est chargé (en T2), l'information peut traverser le décodeur puis la ROM (fin de T2 et T3).

Le registre de micro-instructions est chargé au début de T4. Durant cette phase T4, les champs de micro-commandes sont décodés, les microcommandes sont disponibles à la fin de T4. La partie opérative peut alors les utiliser pendant la phase T1 suivante.

Signature des informations circulant dans la P.C.

Les registres de signature contiennent des éléments de mémorisation maitre/esclave. Deux phases sont donc nécessaires pour réaliser l'étape de division polynomiale. Celle ci peut être effectuée dès que l'information à compacter est disponible.

• Signature des micro-adresses :

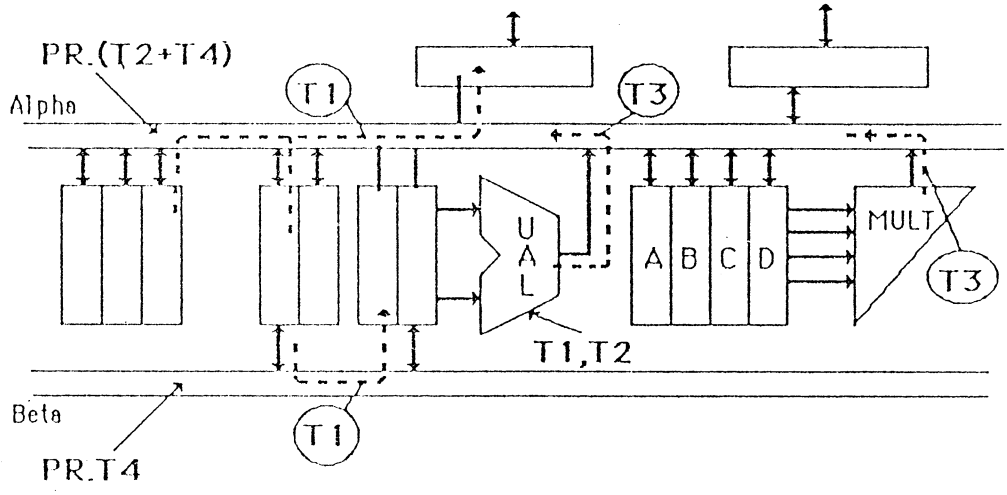
La micro-adresse est disponible à partir de T2 (chargement du registre micro-compteur ordinal), les maitres seront donc chargés avec le reste de la division pendant la phase T3, les esclaves pendant la phase T1 suivante.

• Signature des micro-commandes :

Les micro-commandes sont disponibles à partir de T1 (traversé des décodeurs en T4 précédente), les maitres sont chargés pendant la phase T1, les esclaves pendant T3.

VI.3 - TEMPORISATION DE LA P.O.

Le schéma suivant représente la partie opérative temporisée :



Les indicateurs d'état N et Z sont mis à jour en T4, il sont donc disponibles pour le contrôleur pendant la phase T1 suivante.

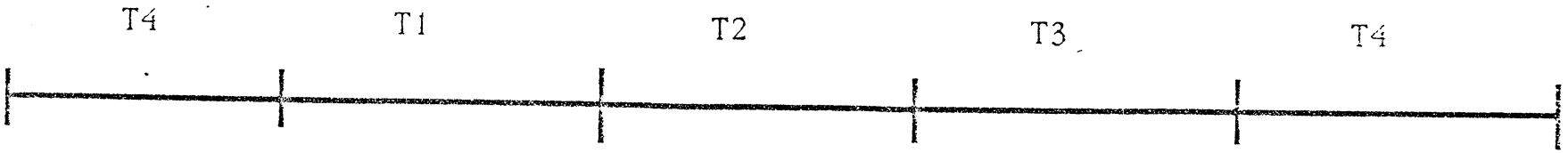
- Compaction des informations internes à a P.O. :

Elle est réalisée dès que l'information à signer est sur le bus; Les maitres sont chargés soit pendant T1 ou T3, les esclaves pendant la phase suivante.

Remarques

i) dans le cas d'un transfert entre registres, les phases T2 et T4 sont vides.

ii) dans le cas ou la micro-instruction est une comparaison suivie d'un test sur le compte-rendu, la P.O. attend les microcommandes de la P.C. pendant deux cycles d'horloge comme indiqué page suivante.



P.C.

- Précharge des PLAs
- Sorties ROM latchées (Ecr. dans Reg µ.Inst)
- Traversée des décodeurs de µ.Commande

- Précharge des décodeurs et de la ROM
- Traversée du séquenceur

- Sorties du séquenceur latchées (Ecr. dans Reg. µ.CO)
- Traversé du décodeur ROM

- Précharge des décodeurs de µ.Commandes
- Traversée de la ROM

- Précharge des PLAs
- Sorties ROM latchées
- Traversée des décodeurs de µ.Commandes

COMPTE-RENDUS

COMPTE-RENDUS

P.O.

- Précharge des Bus

- Précharge du Bus ALPHA

- Précharge des Bus

TRANSFERT ENTRE REGISTRES

- Lecture du ou des registres sources
- Ecriture du ou des registres destination

OPERATION U.A.L.

- Indicateurs d'état latchés

- Lecture du ou des registres opérandes
- Ecriture des registres E1 et E2

- OPERATION U.A.L.

- Lecture du résultat sur le bus ALPHA
- Ecriture dans le registre destination

- Indicateurs d'état latchés

VII - LE TEST HORS-LIGNE DE HSURF

VII - HSURF EST UN MICROPROCESSEUR COMPLETEMENT TESTE A L'INSTANT t=0

C'est la première condition à laquelle doivent satisfaire les composants utilisés dans la réalisation d'applications nécessitant une haute sûreté de fonctionnement. En effet, toutes les opérations de validation et de certification de systèmes assurant le contrôle d'applications immergées dans un environnement à haut risque se fondent sur l'hypothèse de base suivante :

Les composants sont sans défaillance à l'instant t=0

De ce fait, il apparaît indispensable de disposer de circuits dont le test en fin de fabrication soit complet tout en restant compatible avec des contraintes temporelles de production. Lors de la conception du microprocesseur HSURF les études de testabilité ont conduit à intégrer au sein du composant des dispositifs spéciaux permettant de favoriser les opérations de test. Ainsi, l'intégration d'un automate spécialisé indépendant du reste de la partie contrôle s'est avéré indispensable pour faciliter la réalisation des séquences de test hors-ligne.

Dans ce paragraphe, nous développons dans un premier temps la stratégie des opérations de test en fin de fabrication du composant, puis, dans un second temps, nous détaillons la structure matérielle de l'automate de test spécialisé présent au sein de HSURF.

Les opérations de test développées dans la suite étant du type "test hors-ligne" nous considérons que nous disposons d'un dispositif automatique de test [ATE 83] permettant de contrôler les différentes étapes du test. Pour chacune de ces étapes, nous indiquons les ressources internes du composant en cours de vérification, le type des opérations à effectuer (type de données à émettre au circuit, type de données à vérifier) et le type de ressources nécessaires au contrôle de l'opération en cours.

VII.1 - STRATEGIE

Le test hors ligne du microprocesseur HSURF est un test du type GO-NO-GO couramment utilisé dans les opérations de validation de circuits en fin de fabrication. Nous aurons :

- ~ arrêt du test dès qu'une défaillance apparait.
- ~ possibilité de diagnostic pour analyse éventuelle de défaillance.

Le test se déroulera suivant la stratégie "Start-Small" consistant en :

- ~ partitionnement du circuit en blocs.
- ~ test de chacun des blocs soit par accès direct, soit par utilisation de certains blocs déjà testés.

VII.2 - DEROULEMENT DU TEST

On peut considérer que le test se déroule en deux phases distinctes :

- ~ Durant la première phase, on va vérifier l'intégrité d'un ensemble minimum de ressources internes à HSURF. L'accès à ces différentes ressources sera réalisé sous contrôle de l'équipement automatique de test par l'intermédiaire de l'automate spécialisé intégré au sein de HSURF.

- ~ La deuxième phase consistera à vérifier les autres ressources du circuit pour lesquelles l'accès sera réalisé par l'intermédiaire du "noyau minimal" précédemment validé.

La figure 1 représente l'organigramme du test de HSURF par cette méthode.

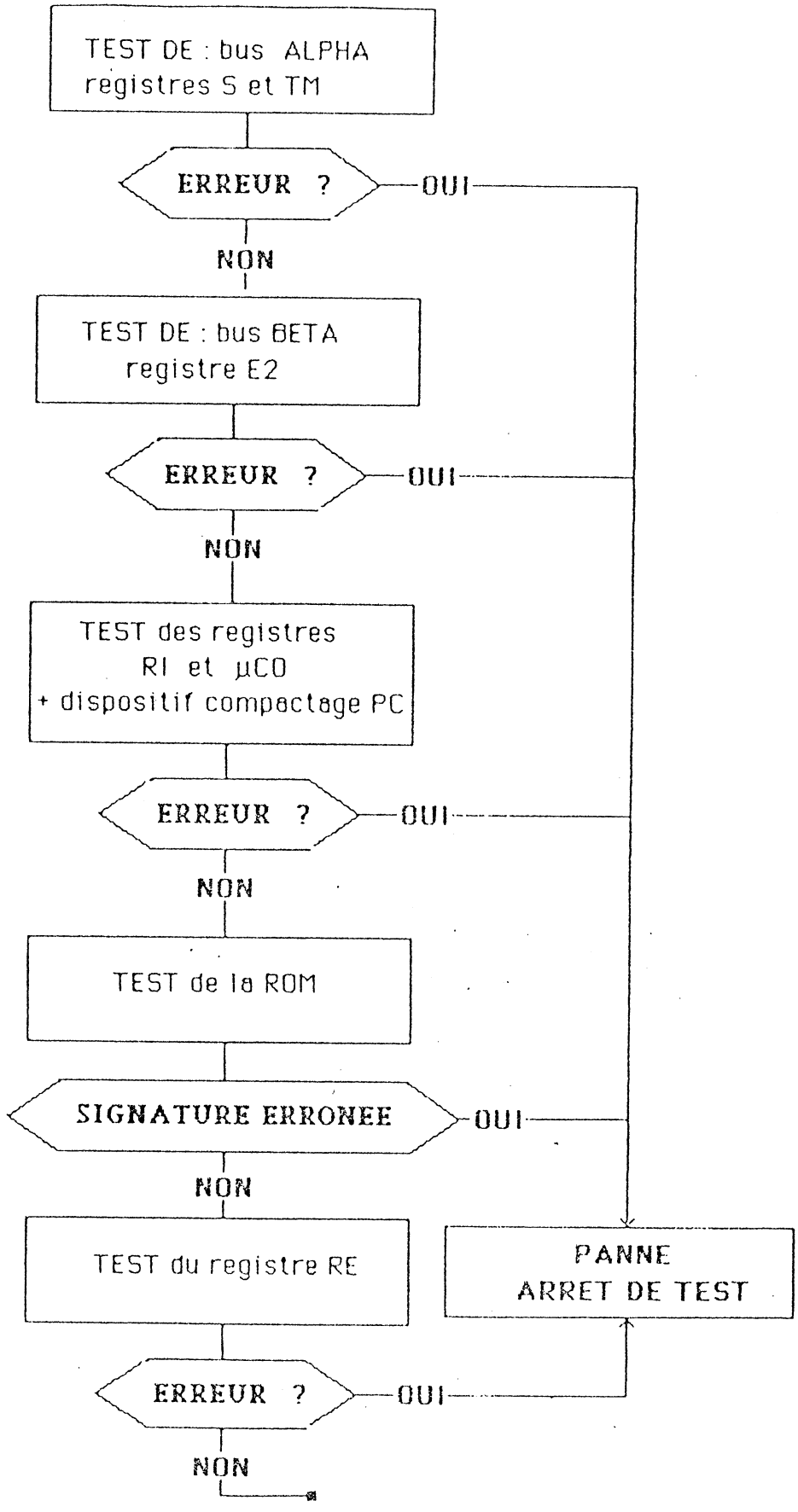

TEST DE : bus ALPHA
registres S et TM
ERREUR ?
OUI
NON
TEST DE : bus BETA
registre E2
ERREUR ?
OUI
NON
TEST des registres
RI et µCO
+ dispositif compactage PC
ERREUR ?
OUI
NON
TEST de la ROM
SIGNATURE ERRONEE
OUI
NON
TEST du registre RE
PANNE
ARRET DE TEST
ERREUR ?
OUI
NON

Figure 1

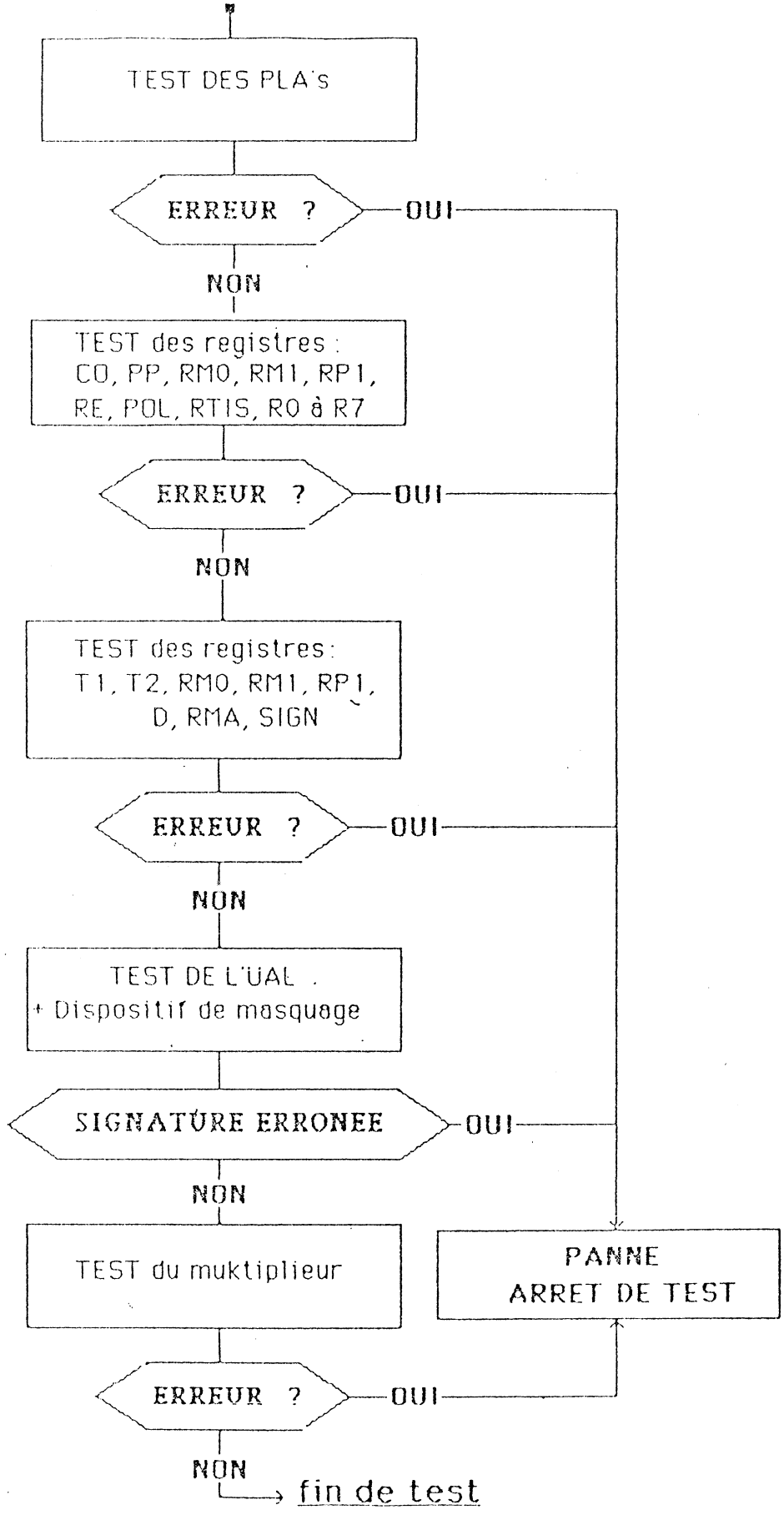


Figure 1 (suite)

VII.3 - PHASE 1 - Test sous contrôleur spécial

VII.3.1 - Etape n°1 (Fig. 2)

Ressources testées : Bus ALPHA, registres S et TM.

Stratégie de test : Réalisation d'un test du type mémoire vive par écriture d'une suite de "1" et de "0" dans les registres à tester. Les données transitent par le bus ALPHA. Une donnée est lue sur le bus de donnée externe, elle est stockée dans le registre TM, elle est transmise au registre S via le bus ALPHA et elle est recueillie sur le bus adresse externe. Ce test est répété avec autant de vecteurs de test que nécessaire pour détecter toute panne pouvant apparaître dans ce bloc.

Contrôle du test : Le contrôle du test est effectué par l'équipement automatique de test et par l'automate spécialisé interne au microprocesseur.

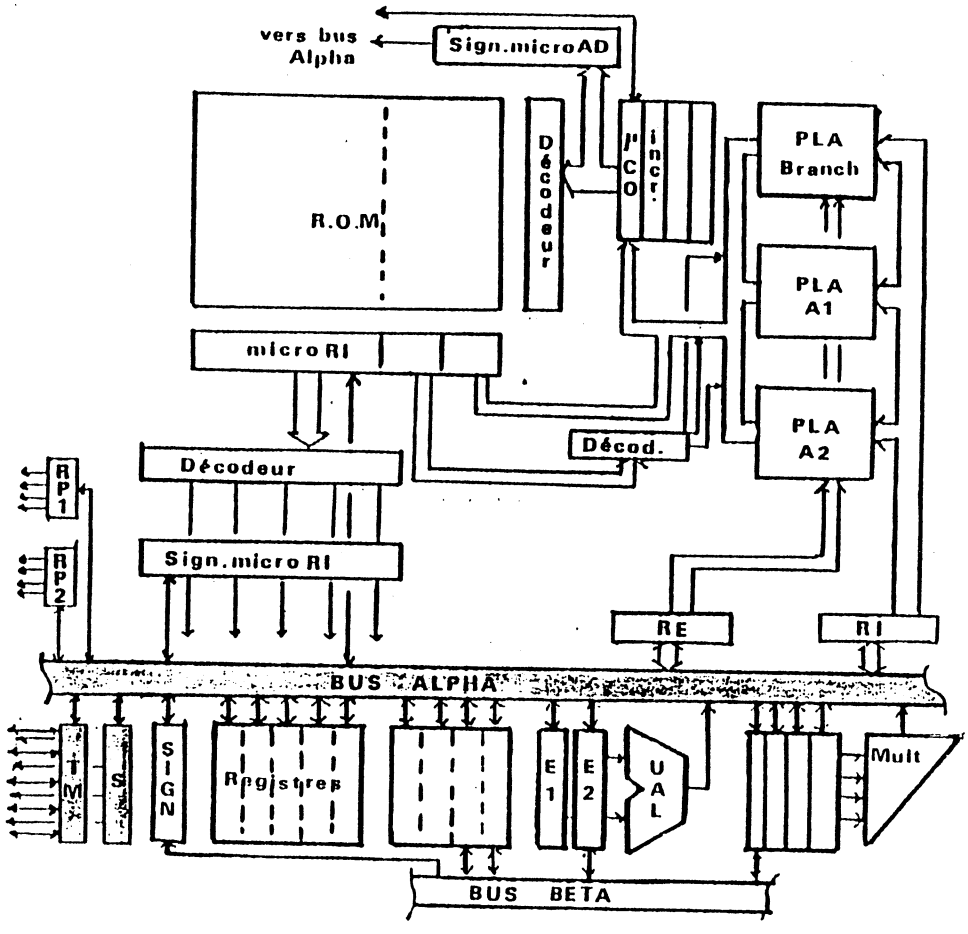


Figure 2

VII.3.2 - Etape n°2 (Fig. 3)

Ressources testées :Bus BETA, registre E2.

Stratégie de test : Comme pour l'étape précédente, on réalise un test du type mémoire vive. Une donnée est lue sur le bus donnée externe, stockée dans le registre TM, transmise au registre E2 via le bus BETA, stockée dans le registre S via le bus ALPHA puis recueillie sur le bus adresse externe.

Contrôle du test : Du même type que pour l'étape n°1.

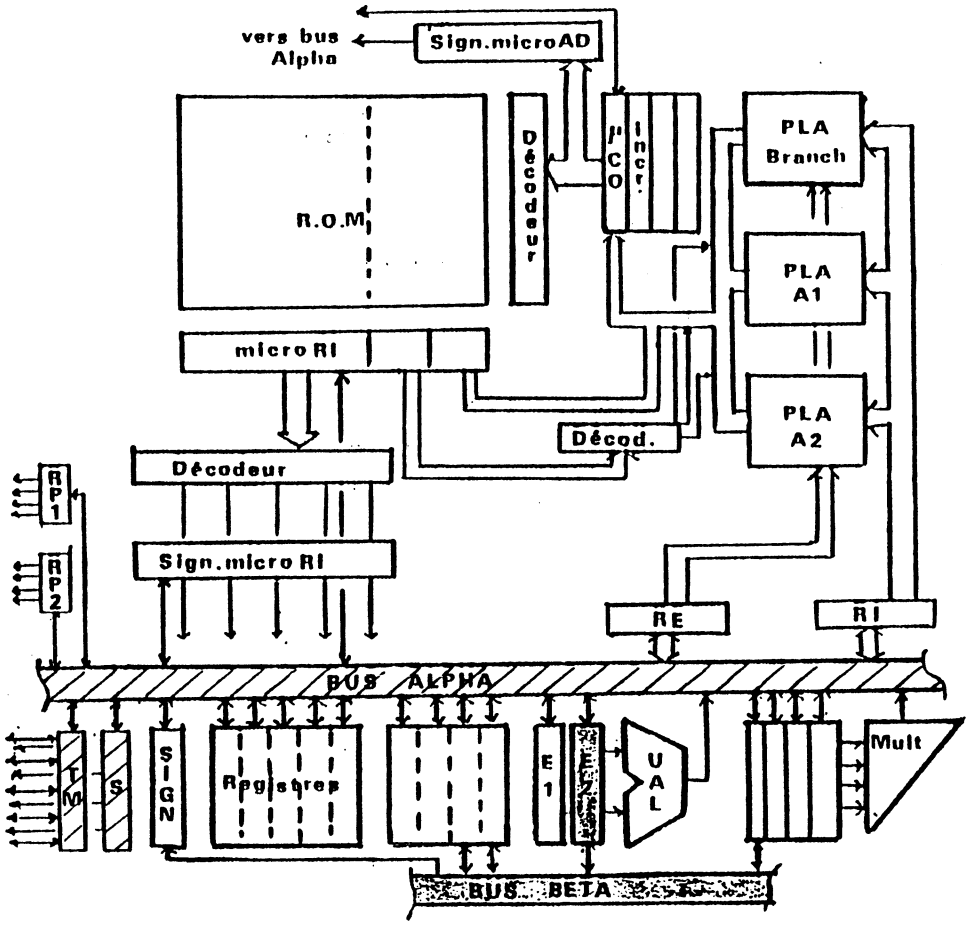


Figure 3

VII.3.3 - Etape n°3 (Fig.4)

Ressources testées : Registre RI.

Stratégie de test : Test du type mémoire vive. Une donnée est lue sur le bus donnée externe, elle est écrite dans RI via le registre TM et le bus ALPHA, enfin après avoir transité par le registre S, elle est recueillie sur le bus adresse externe.

Contrôle du test : Equipement automatique de test plus automate spécialisé.

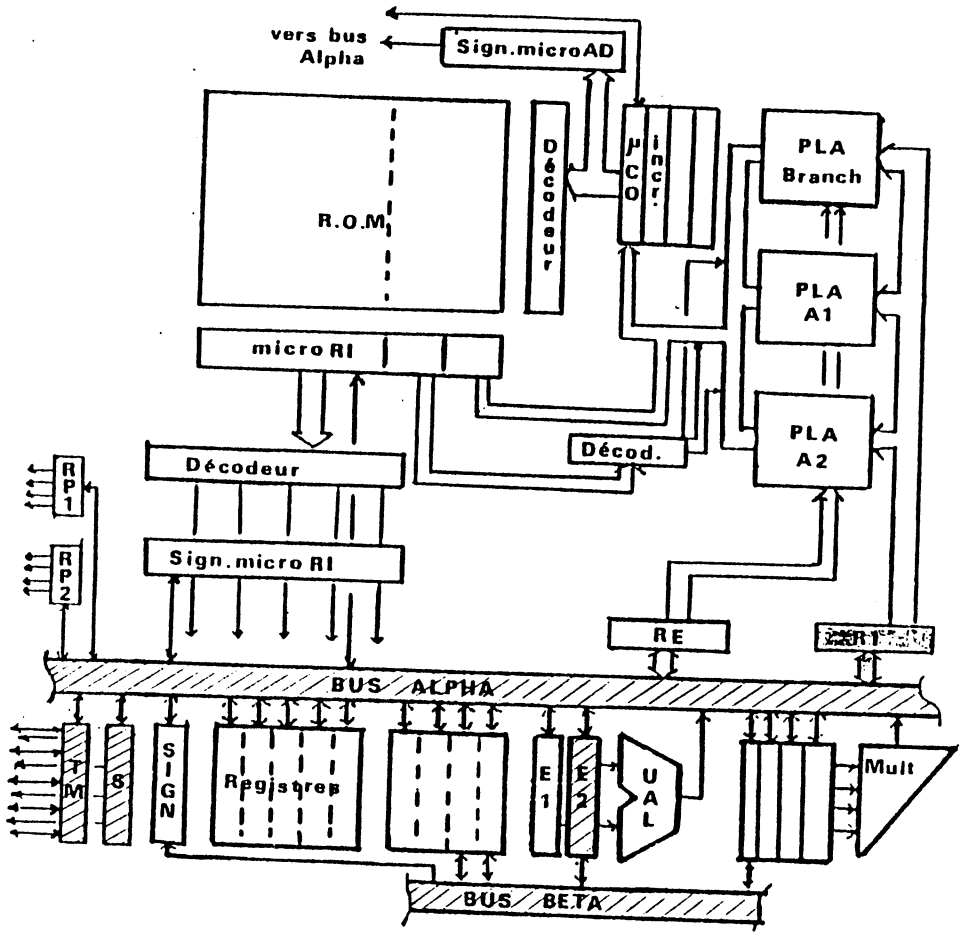


Figure 4

VII.3.4 - Etapes n°4 et n°5 (Fig.5)

Ressources testées : Registre micro-RI, Registre micro-CO, Dispositifs de compactage associés.

Stratégie de test : Test du type mémoire vive. Ces deux registres peuvent être testés l'un après l'autre dans un ordre indifférent. Une donnée est lue sur le bus donnée externe, elle est écrite dans le registre sous test via le bus ALPHA. On réalise ensuite la signature du contenu du registre à l'aide du dispositif de signature associé au registre sous test. Lorsque tous les vecteurs de test ont été envoyés, le contenu du registre de signature est dirigé vers le bus adresse externe soit en une fois pour le registre SIGN-MICRO-CO (16 bits), soit en plusieurs fois par morceaux pour le registre SIGN-MICRO-RI (6x16 bits). Une comparaison de la signature obtenue avec la valeur exacte atendue est réalisée.

Contrôle du test : Equipement automatique de test plus automate spécialisé.

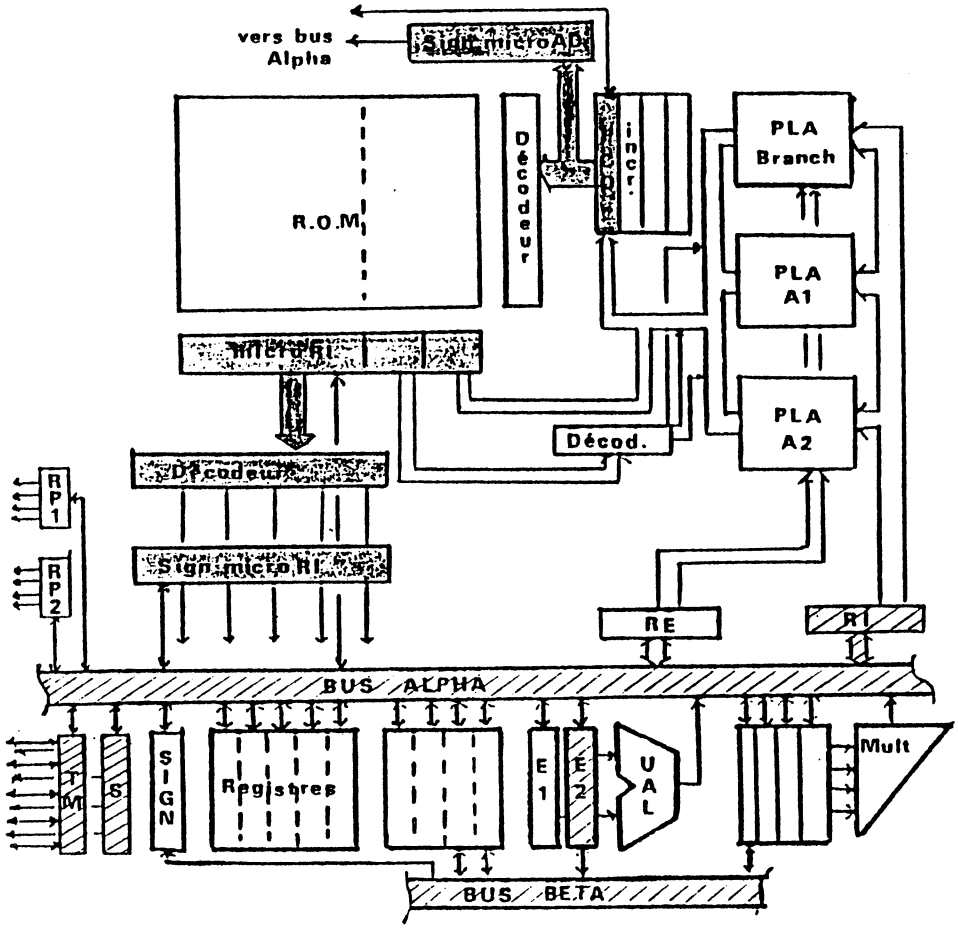


Figure 5

VII.3.5 - Etape n°6 (Fig.6)

Ressources testées : ROM de µProgramme, ensemble pile + incrémenteur.
Stratégie de test : La ROM est lue intégralement. Afin d'accélérer l'étape de test, les sorties de la ROM sont compactées. Le registre de micro-CO est initialisé à "0" ainsi que les registres de signature. La ROM est lue, les registres micro-CO et micro-RI sont signés, l'opération est répétée sur l'ensemble de la ROM. L'incrémentation du registre micro-CO est réalisée par des séquences d'empilement/ dépilement dans la pile de micro-CO. En fin de test, on compare les valeurs compactées aux valeurs exactes.

Afin de permettre une reconfiguration de la ROM (si cela a été prévu), il est possible de recueillir le contenu de chaque mot de la ROM sur le bus adresse externe. Ce genre de test, trop long, ne sera envisagé qu'en cas de nécessité de localisation du défaut.
Contrôle du test : Equipement automatique de test plus automate spécialisé.

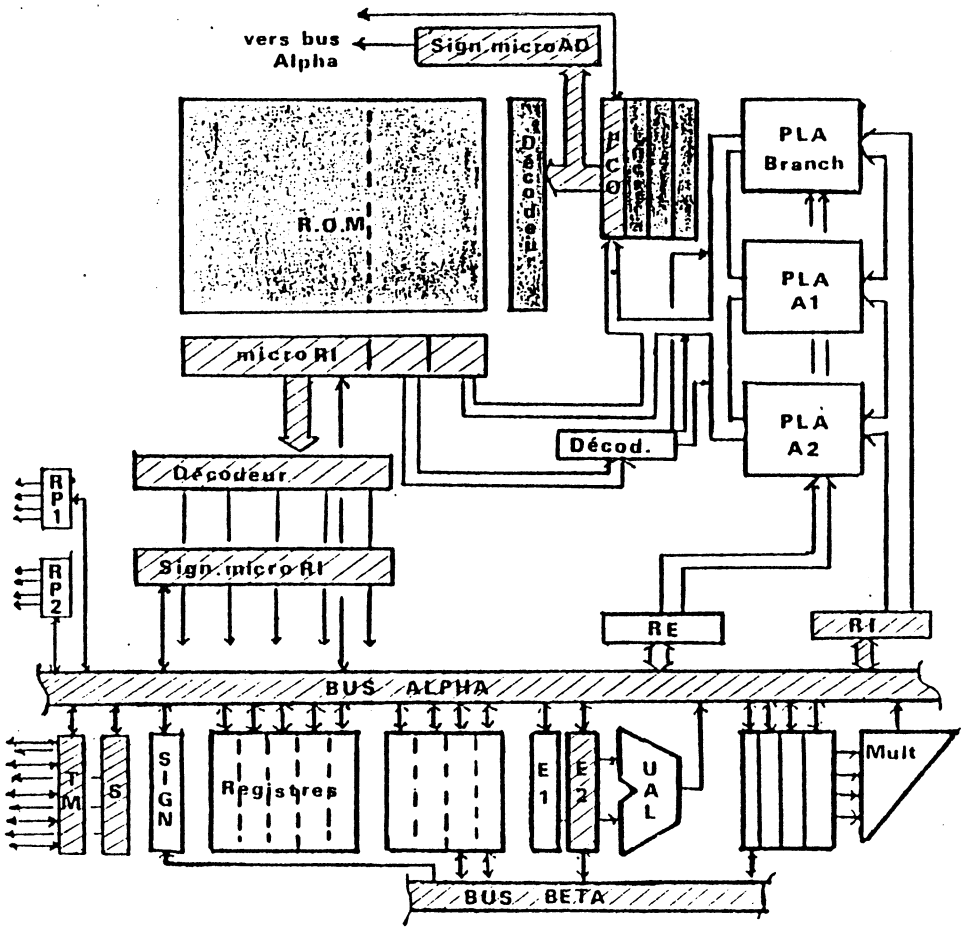


Figure 6

VII.3.6 - Etape n°7 (Fig. 7)

Ressources testées : registre RE.

Stratégie de test : un test du type mémoire vive est effectué. On vérifie la capacité de mémorisation des cellules du registre RE. Une donnée est lue sur le bus donnée externe puis écrite dans le registre RE. La donnée est ensuite stockée via le bus ALPHA dans le registre S puis envoyée sur le bus adresse externe.

Contrôle du test : automate spécialisé, équipement automatique de test.

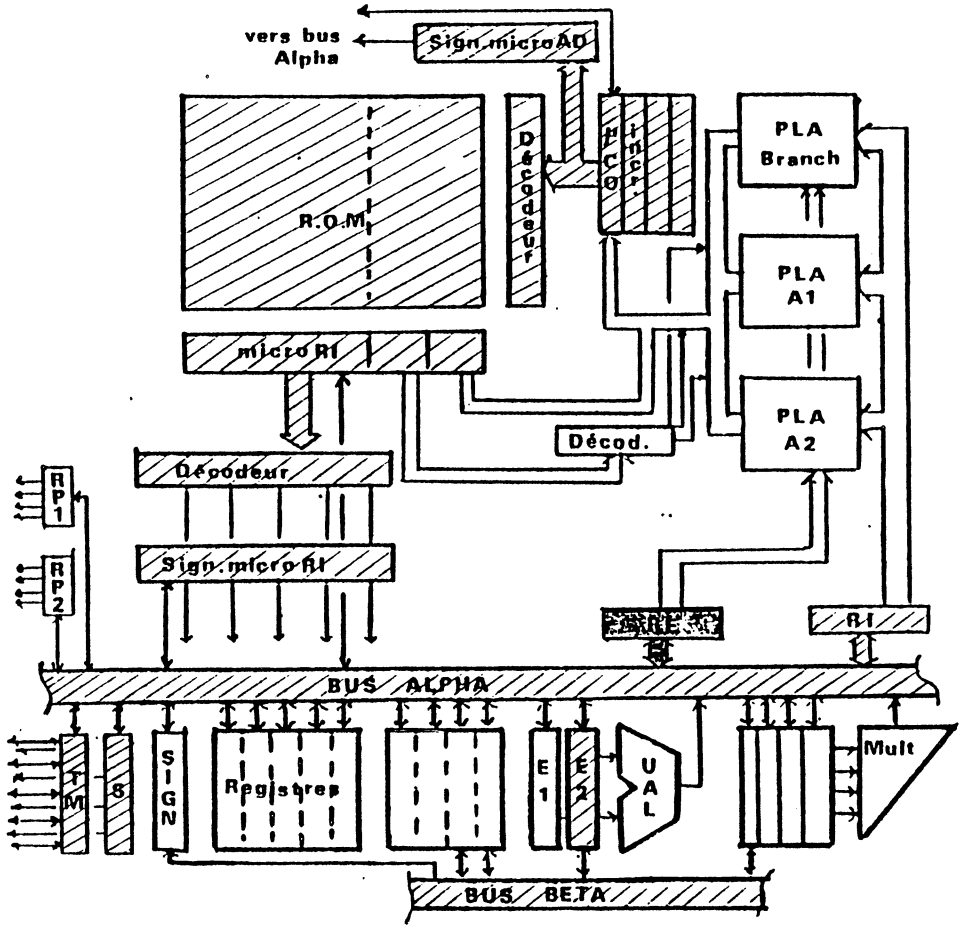


Figure 7

VII.3.7 - Etapes n°8 à n°10 (Fig. 8)

Ressources testées : PLA A1, PLA A2, PLA BRANCH.

Stratégie de test : Des vecteurs de test sont calculés pour chaque PLA. Une méthode de calcul pour les vecteurs de test de PLA en fonction du domaine d'entrée accessible est détaillée dans [DEC 85]. Le registre de signature SIGN-MICRO-CO est initialisé à zéro, les vecteurs de test sont chargés dans le registre RI (et RE), la sortie du PLA en cours de test est sélectionnée puis sa valeur est compactée via le registre μCO. Trois programmes sont successivement utilisés pour chacun des PLAs. Comme pour la ROM, si une reconfiguration du PLA a été prévu, une lecture du registre μCO peut être effectuée après l'envoi de chaque vecteur de test.

Contrôle du test : automate spécialisé, équipement automatique de test.

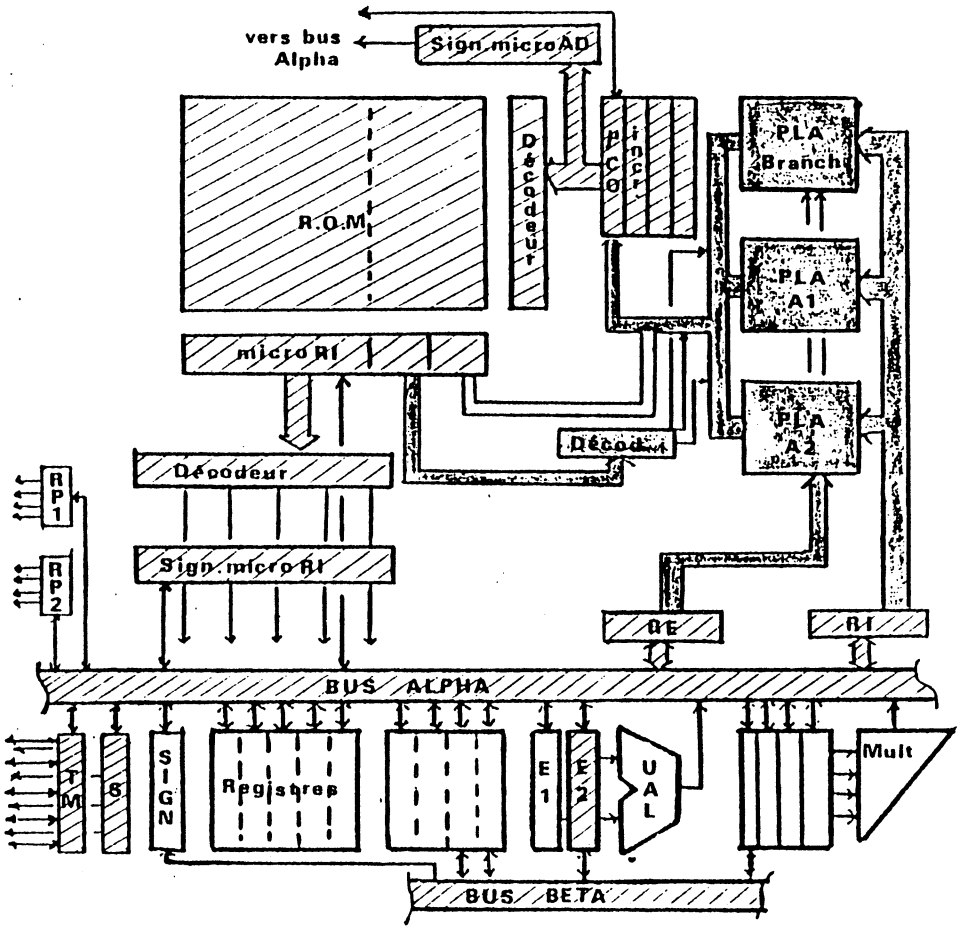


Figure 8

VII.4 - PHASE 2 - Test sous contrôleur de HSURF

VII.4.1 - Etape n°11 à 25 (Fig. 9)

Ressources testées : registre CO
registre PP
registre RMO
registre RM1
registre RP1
registre POL
registre RTIS
registres RO à R7

Stratégie de test : Réalisation d'un test du type mémoire vive par écriture d'une suite de "1" et de "0" dans les registres à tester. Les données transitent par le bus ALPHA.

Contrôle du test : Le noyau minimum du microprocesseur étant testé, c'est le contrôleur de HSURF qui est utilisé pour contrôler ce test.

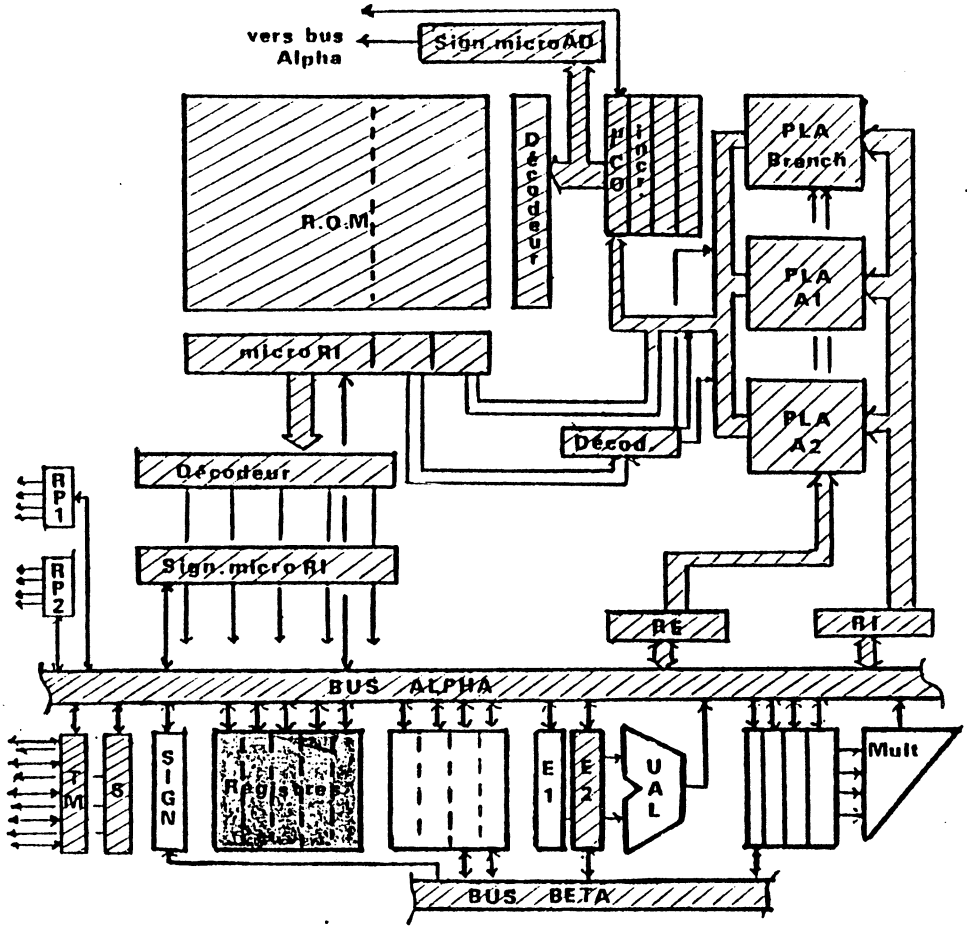


Figure 9

VII.4.2 - Etape n°26 à 35 (Fig. 10)

Ressources testées : registre T1
registre T2
registre E1
registre RP2
registres A,B,C,D
registre RMA
registre SIGN

Stratégie de test : Réalisation d'un test du type mémoire vive par écriture d'une suite de "1" et de "0" dans les registres à tester. Les données transitent par le bus ALPHA. Ici, l'accès aux différents registres est réalisé à l'aide des instructions spécifiques de test LRS et SRS.

Contrôle du test : Le noyau minimum du microprocesseur étant testé, c'est le contrôleur de HSURF qui est utilisé pour contrôler ce test.

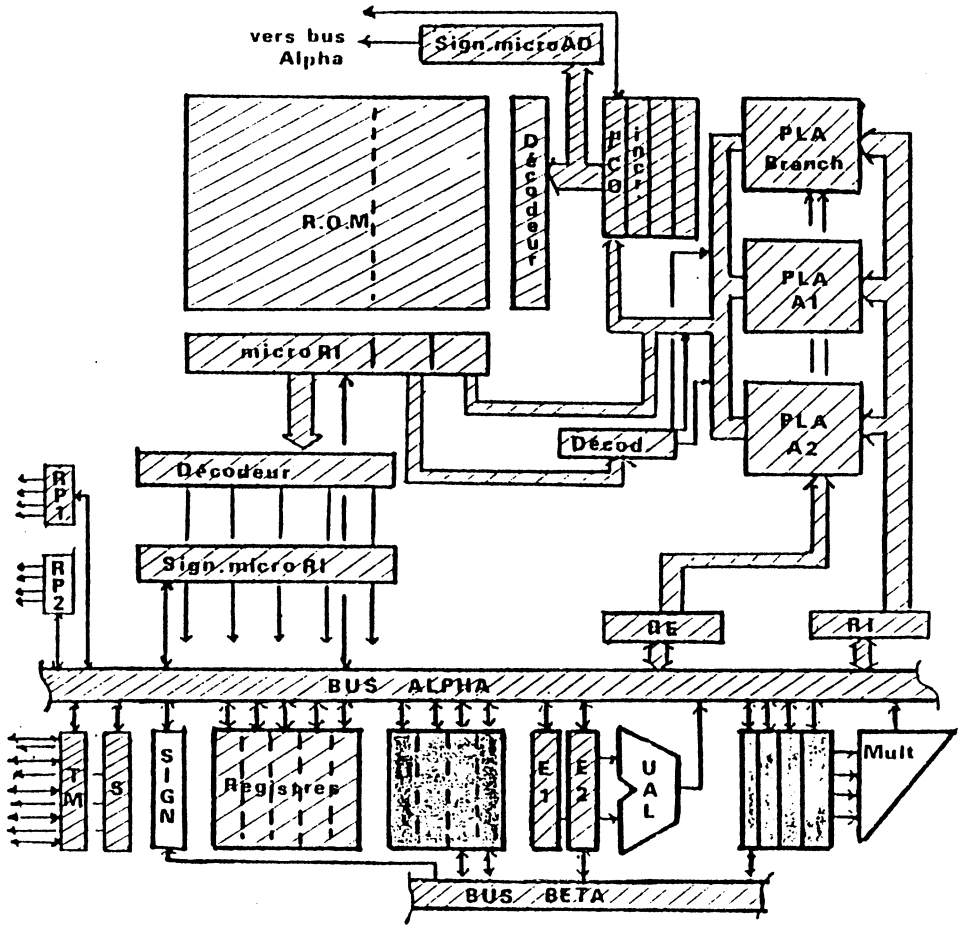


Figure 10

VII.4.3 - Etape n°36 (Fig. 11)

Ressources testées : registre DM, Unité arithmétique et Logique, dispositif de masquage, dispositif de signature.

Stratégie de test : l'unité arithmétique et logique est composée de 16 blocs. On réalise un test exhaustif de chaque bloc avec les vecteurs de test calculés lors de la conception de cet opérateur (cf. Chap. IV-B) Un programme de test est écrit afin d'amener les vecteurs de test jusqu'à l'UAL et le dispositif de masquage. Les registres E1 et E2 sont initialisés avec les vecteurs de test, l'opération est exécutée, le "contenu" du bus ALPHA est signé. En fin d'étape, on compare la valeur de la signature avec une valeur exacte précalculée.

Contrôle du test : le contrôleur de HSURF est utilisé pour contrôler ce test, on fait appel ici à l'instruction de test spécialisée TUAL.

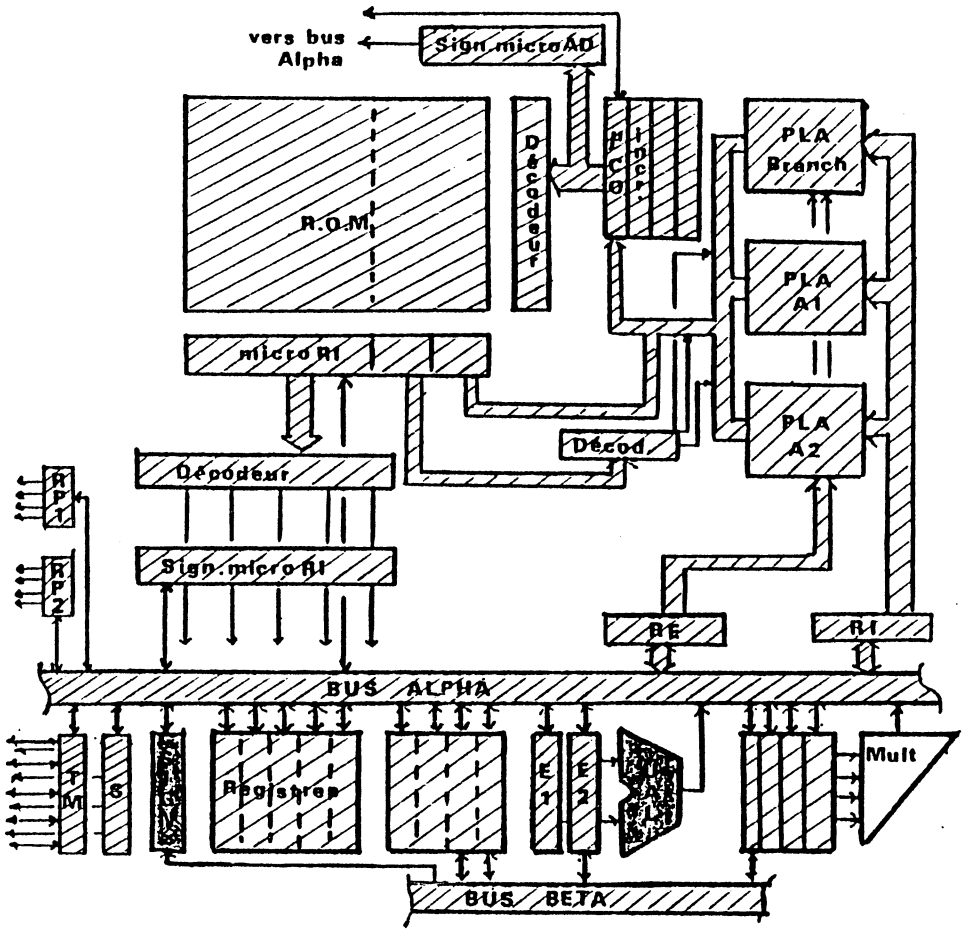


Figure 11

VII.4.4 - Etape n°37 (Fig. 12)

Ressources testées : multiplieur (MULT).

Stratégie de test : Un test exhaustif de chaque cellule du multiplieur est réalisé (cf. l'ensemble des données de test en annexe XX). Un programme de test est écrit afin d'amener les vecteurs de test jusqu'au multiplieur. Les registres A, B, C, D sont initialisés avec les vecteurs de test, l'opération est exécutée, le "contenu" du bus ALPHA est signé. En fin d'étape, on compare la valeur de la signature avec une valeur exacte précalculée.

Contrôle du test : le contrôleur de HSURF qui utilisé pour contrôler ce test.

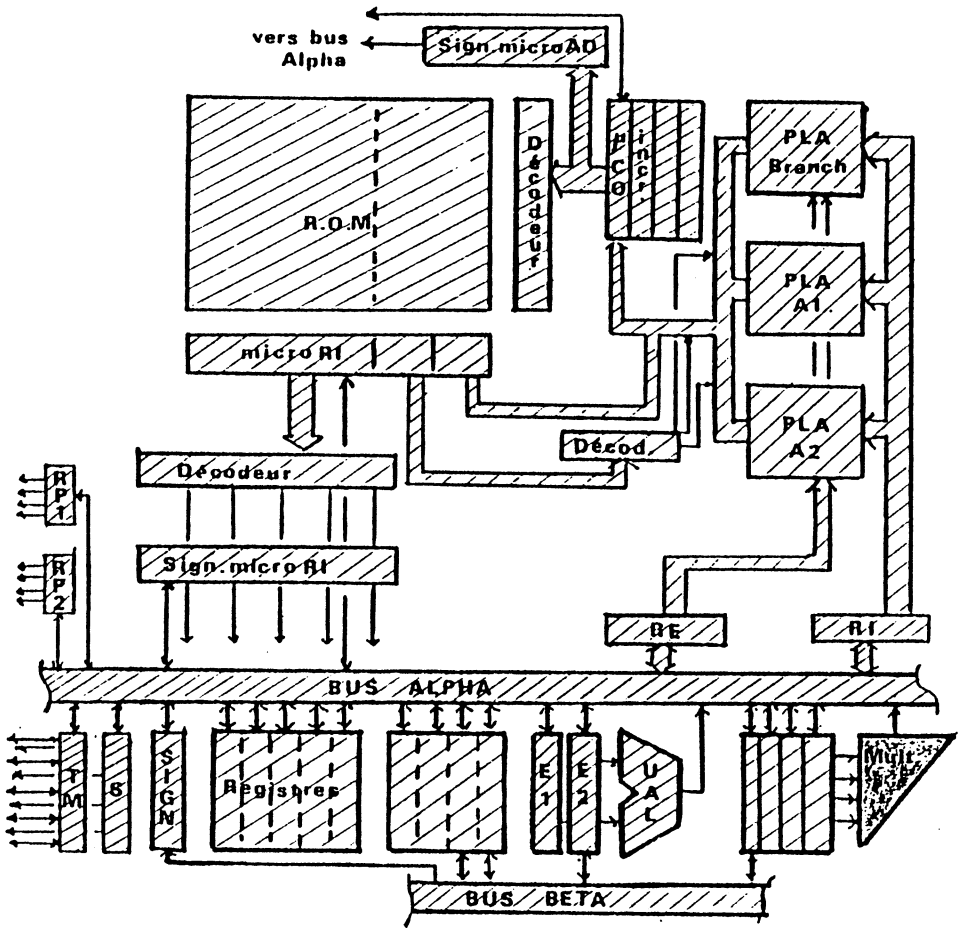


Figure 12

VII.5 - L'AUTOMATE DE TEST

VII.5.1 - Définition

Un automate d'état fini est défini par un ensemble d'états et de transitions. A chaque état est associé un ensemble de commandes pouvant être vide. A chaque transition est associée une condition autorisant ou non le passage à l'état successeur.

Un automate d'état fini peut être décrit à l'aide de symboles représentant les états et les transitions (graphe de contrôle interprété).

- Chaque état est représenté par un cercle contenant l'identificateur de l'état symbolisé. L'un des états, appelé état initial, est symbolisé par un cercle double. C'est dans cet état que l'automate doit se trouver lors de son initialisation.

- Les états sont reliés entre eux par des arcs coupés par des traits représentant les transitions.

Exemple (Figure 13) :

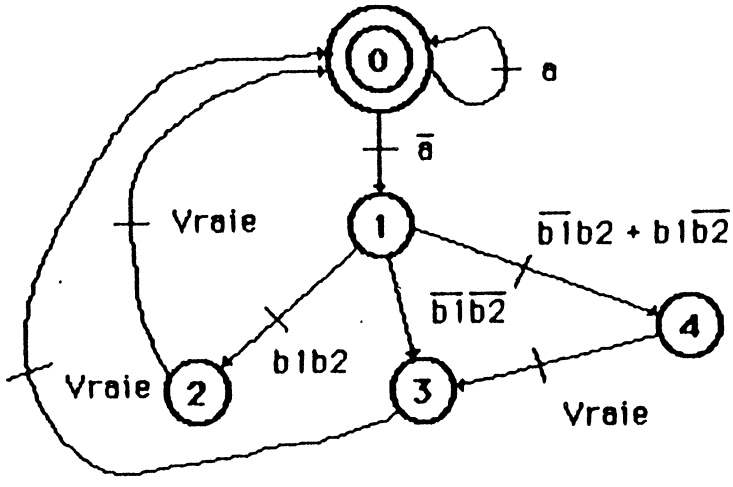


Figure 13

Remarque : la condition "vraie" étant toujours réalisée, la transition est franchie chaque fois que l'automate est dans un des états qui la précède (ici les états 2, 3 ou 4).

VII.5.2 - le dispositif interne de test de HSURF

Il est possible de définir un automate réalisant la phase 1 des séquences de test hors-ligne décrites précédemment. Le rôle de cet automate sera d'émettre les commandes nécessaires à l'exécution du test et de contrôler le séquencement des opérations. Cet automate peut être réalisé à l'aide d'un PLA générant les commandes et l'adresse de l'état suivant. L'état courant de l'automate est mémorisé par un ensembles de bascules (cf. Figure 14).

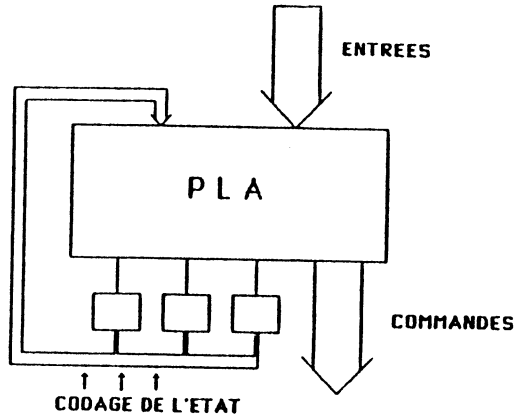


Figure 14

Les figures 15 à 22 donnent une représentation du graphe de contrôle interprété de l'automate de test de HSURF pour chacune des étapes décrites précédemment.

Il est composé de :

- 9 entrées :
  - 6 entrées codant l'adresse de l'état courant (A_0 à A_5).
  - 1 entrée de synchronisation (TEST).
  - 3 bits de codage indiquant le numéro de l'étape à réaliser ($DATA_0$ à $DATA_2$).
- 45 sorties :
  - 6 commandes représentant l'adresse de l'état suivant (S_0 à S_5).
  - 36 commandes à destination des points internes à HSURF à contrôler lors du test (C_0 à C_{35}).
  - 3 bits de codes permettant de réaliser l'autotest du PLA. Le code choisi est un code de Berger modifié [MAK 82] Il indique pour tout vecteur d'entrée, le nombre de sorties à 1. Ce nombre étant toujours inférieur à 8, seuls trois bits sont nécessaire pour réaliser ce codage.

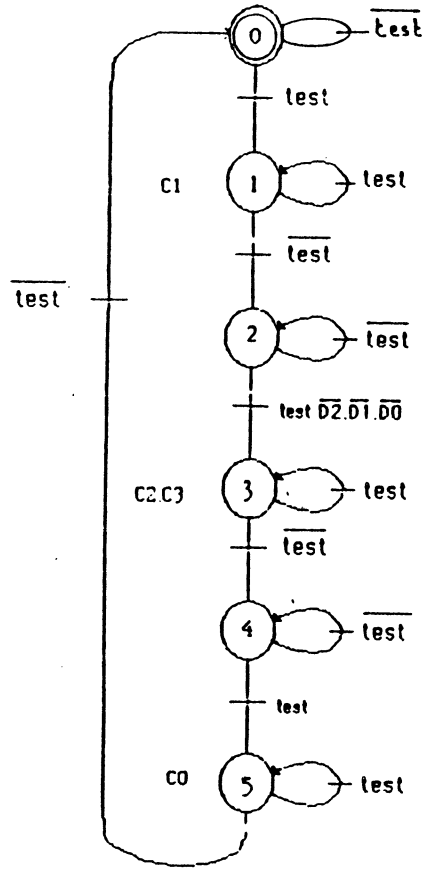


Figure 15

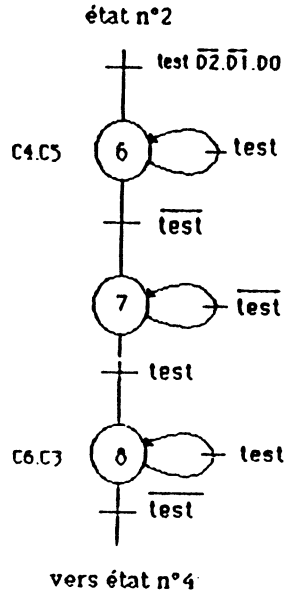


Figure 16

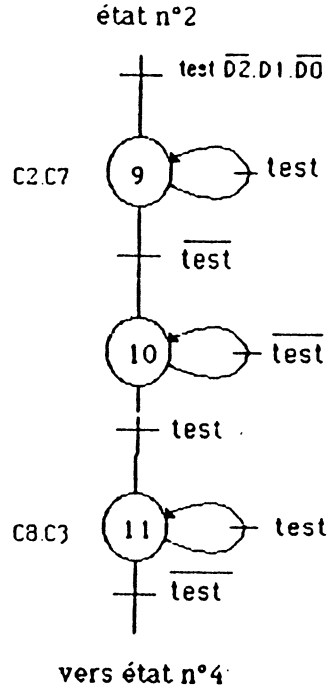


Figure 17

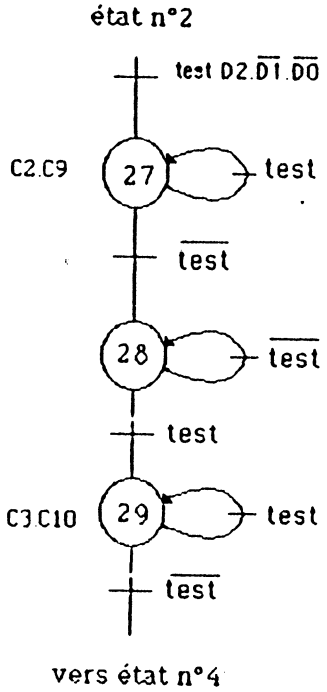


Figure 18

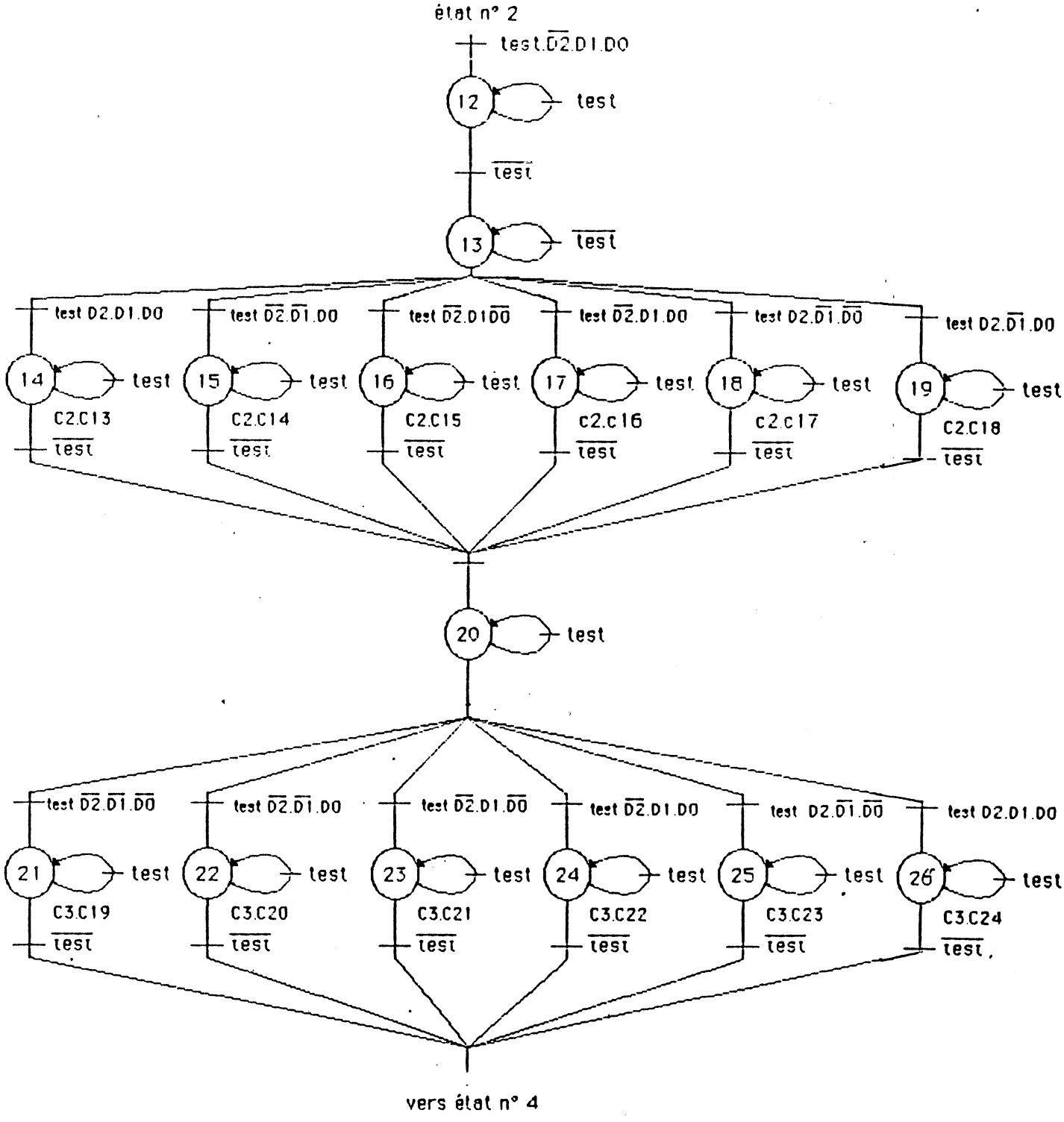

état n° 2
test.D2.D1.D0
12
test
test
13
test
test D2.D1.D0
test D2.D1.D0
test D2.D1.D0
test D2.D1.D0
test D2.D1.D0
test D2.D1.D0
14
15
16
17
18
19
test
test
test
test
test
test
C2.C13
C2.C14
C2.C15
C2.C16
C2.C17
C2.C18
test
test
test
test
test
test
20
test
test D2.D1.D0
test D2.D1.D0
test D2.D1.D0
test D2.D1.D0
test D2.D1.D0
test D2.D1.D0
21
22
23
24
25
26
test
test
test
test
test
test
C3.C19
C3.C20
C3.C21
C3.C22
C3.C23
C3.C24
test
test
test
test
test
test
vers état n° 4

Figure 19

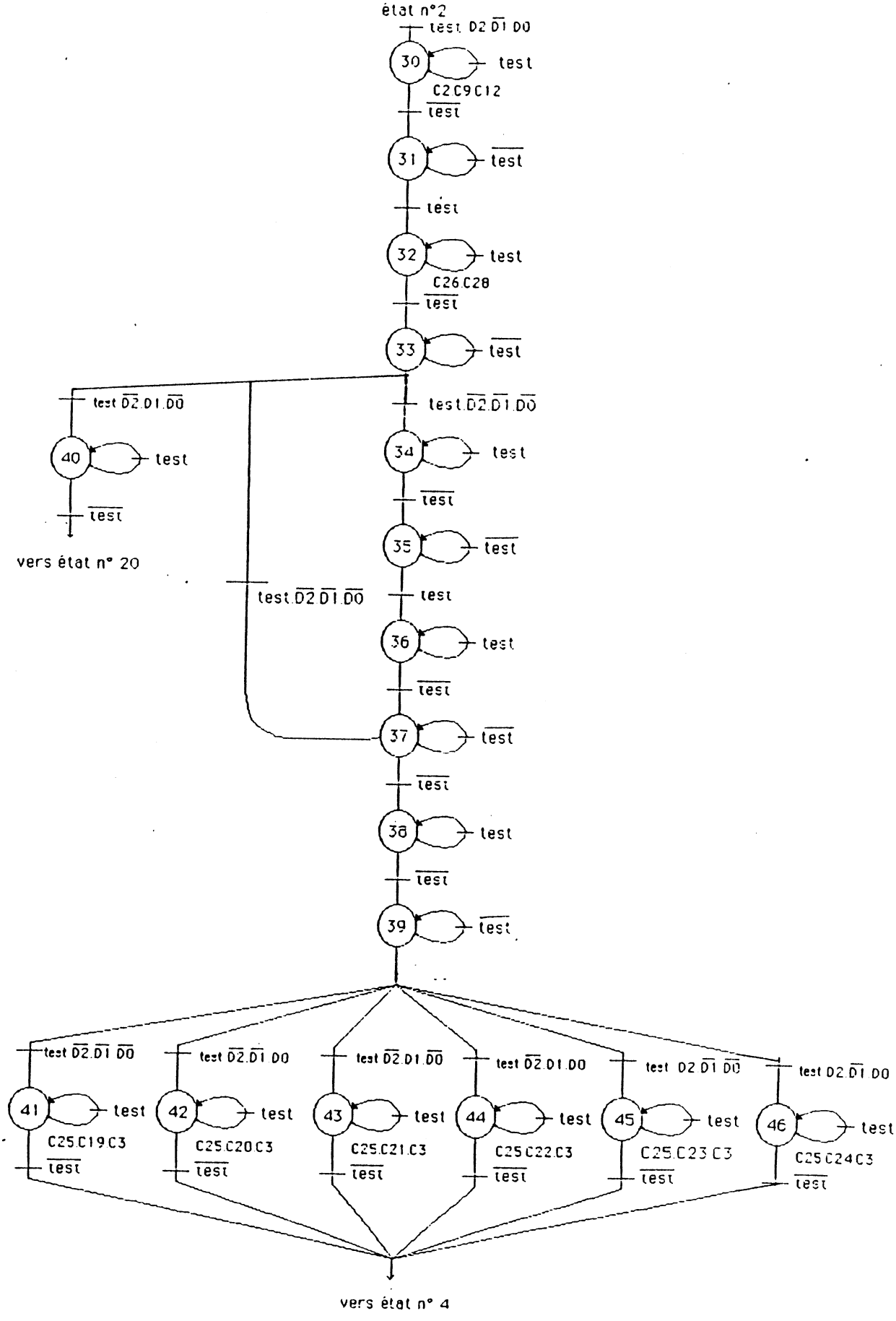

état n°2
test D2 D1 D0
30
test
C2 C9 C12
test
31
test
test
32
test
C26 C28
test
33
test
test D2 D1 D0
40
test
test
vers état n° 20
test D2 D1 D0
34
test
test
35
test
test
36
test
test
37
test
test
38
test
test
39
test
test D2 D1 D0
41
test
C25 C19 C3
test
test D2 D1 D0
42
test
C25 C20 C3
test
test D2 D1 D0
43
test
C25 C21 C3
test
test D2 D1 D0
44
test
C25 C22 C3
test
test D2 D1 D0
45
test
C25 C23 C3
test
test D2 D1 D0
46
test
C25 C24 C3
test
vers état n° 4

Figure 20

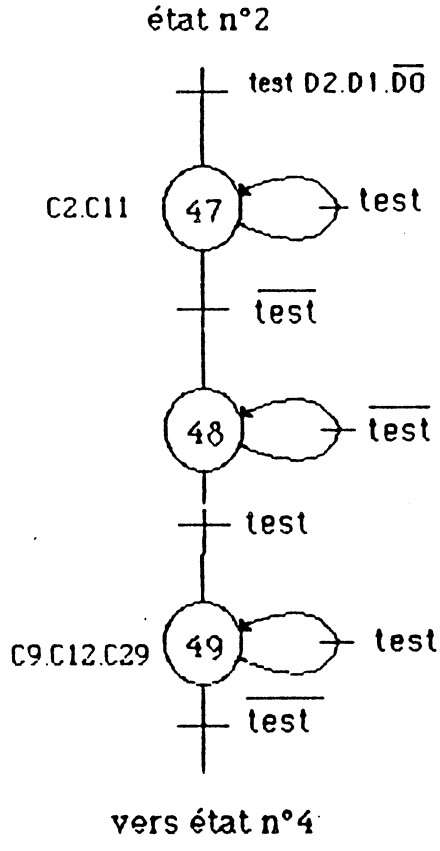

état n°2
test D2.D1.$\overline{D0}$
C2.C11
47
test
$\overline{test}$
48
$\overline{test}$
test
C9.C12.C29
49
test
$\overline{test}$
vers état n°4

Figure 21

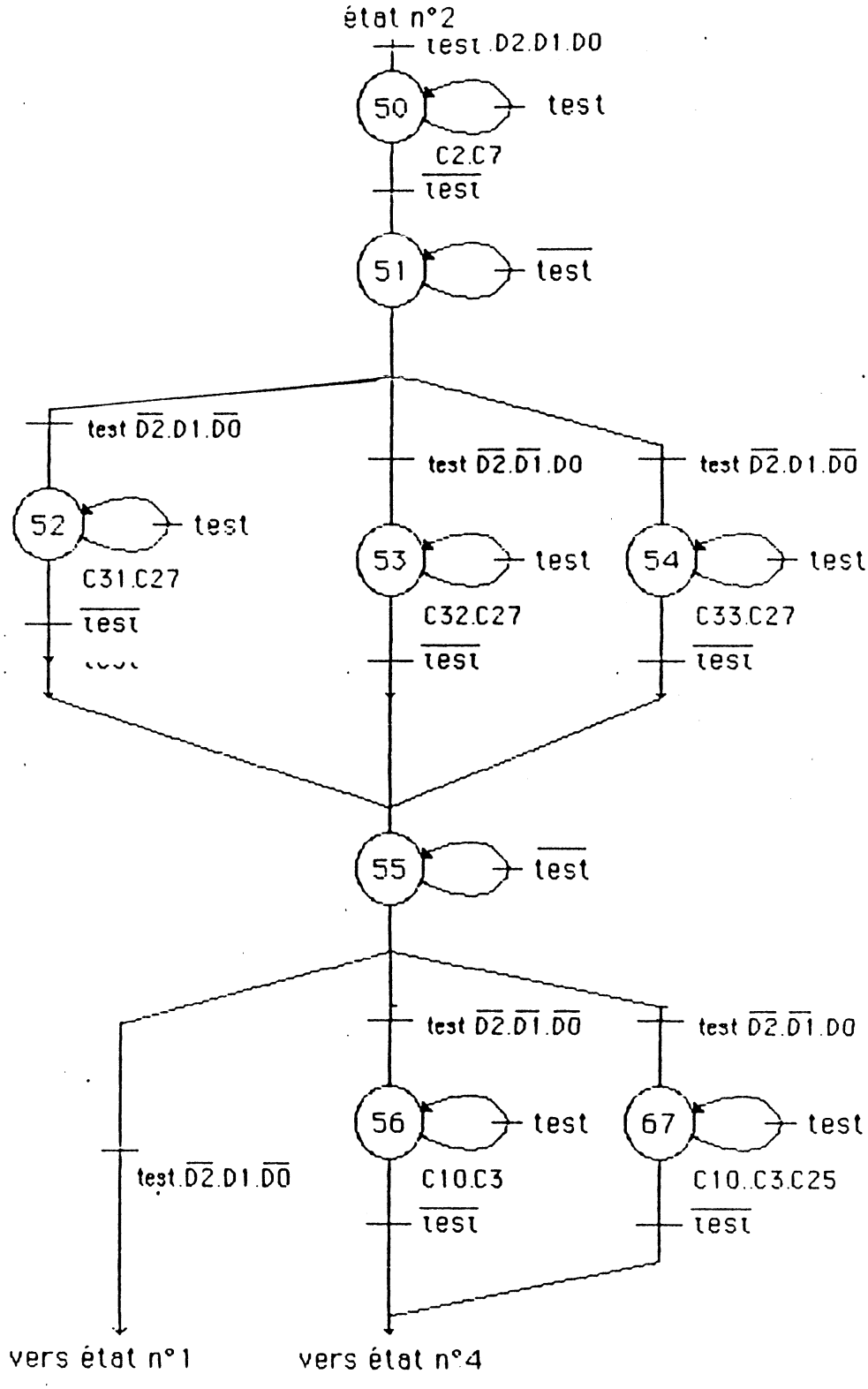

état n°2
test.D2.D1.D0
50
test
C2.C7
test
51
test
test D2.D1.D0
test D2.D1.D0
test D2.D1.D0
52
test
53
test
54
test
C31.C27
test
C32.C27
test
C33.C27
test
55
test
test D2.D1.D0
test D2.D1.D0
56
test
67
test
test.D2.D1.D0
C10.C3
test
C10.C3.C25
test
vers état n°1
vers état n°4

Figure 22

SIGNIFICATION DES DIFFERENTES COMMANDES GENEREES PAR L'AUTOMATE DE TEST :

C0 = ACQ et Ecriture du contenu du registre S sur le bus externe.
C1 = Chargement du registre TM avec la valeur présente sur le bus externe.
C2 = [TM] → Bus Alpha ([TM] signifie : contenu du registre TM)
C3 = Bus Alpha → S
C4 = [TM] → Bus Beta
C5 = Bus Beta → E2
C6 = [E2] → Beta
C7 = Bus Alpha → RI
C8 = [RI] → Bus Alpha
C9 = Bus Alpha → Micro-CO
C10 = [Micro-CO] → Bus Alpha
C11 = Bus Alpha → RE
C12 = [RE] → Bus Alpha
C13 = Bus Alpha → Micro-RI(0)
C14 = Bus Alpha → Micro-RI(1)
C15 = Bus Alpha → Micro-RI(2)
C16 = Bus Alpha → Micro-RI(3)
C17 = Bus Alpha → Micro-RI(4)
C18 = Bus Alpha → Micro-RI(5)
C19 = [Micro-RI(0)] → Bus Alpha
C20 = [Micro-RI(1)] → Bus Alpha
C21 = [Micro-RI(2)] → Bus Alpha
C22 = [Micro-RI(3)] → Bus Alpha
C23 = [Micro-RI(4)] → Bus Alpha
C24 = [Micro-RI(5)] → Bus Alpha
C25 = Opération de signature
C26 = ROM[Micro-CO] → Micro-RI
C27 = Signature Micro-CO
C28 = Signature Micro-RI
C29 = RAZ. SIGN-Micro-CO
C30 = RAZ SIGN-Micro-RI
C31 = Sortie PLA1 → Micro-CO
C32 = Sortie PLA2 → Micro-CO
C33 = Sortie PLA3 → Micro-CO
C34 = Empiler
C35 = Dépiler

Au niveau extérieur (circuit vu par l'utilisateur), la présence de l'automate se traduit par l'ajout de 5 broches d'entrée/sortie (cf. Fig. 23) :

- Entrée des 3 bits de contrôle ($DATA_0$ à $DATA_2$)
- Entrée du signal de synchronisation (TEST)
- Sortie d'un signal ACQ indiquant la fin de l'étape de test (C_{35}).

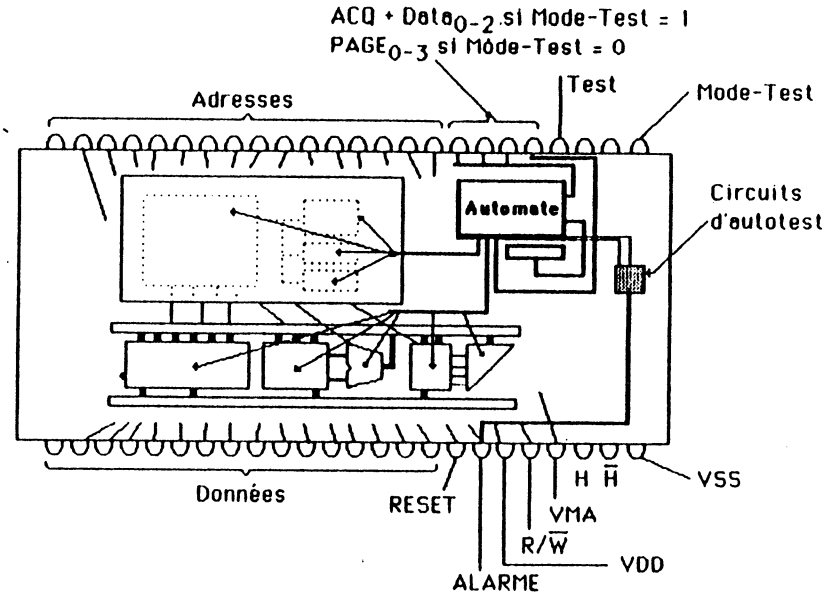


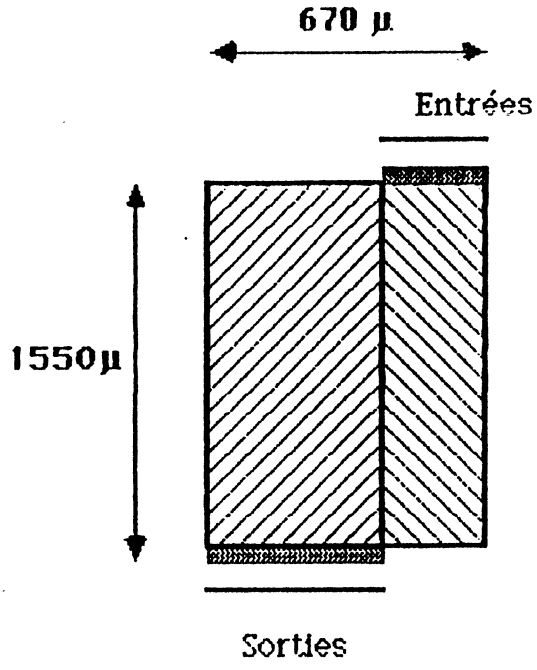
Figure 23

VII.5.3 - Réalisation de l'automate de test

La réalisation matérielle de l'automate de test interne à HSURF c'est déroulée de façon entièrement automatique à l'aide d'un logiciel de génération de parties contrôle. Cette chaîne d'aide à la conception permet à partir du graphe de contrôle interprété de générer les équations booléennes correspondant aux différentes commandes et au calcul de l'adresse suivante.

Une minimisation globale des fonctions booléennes est réalisée. Les résultats obtenus sont ensuite utilisés par un générateur de PLAs (Dessin des masques pour la technologie CMOS utilisée).

Les résultats obtenus pour le contrôleur de test de HSURF sont les suivants :



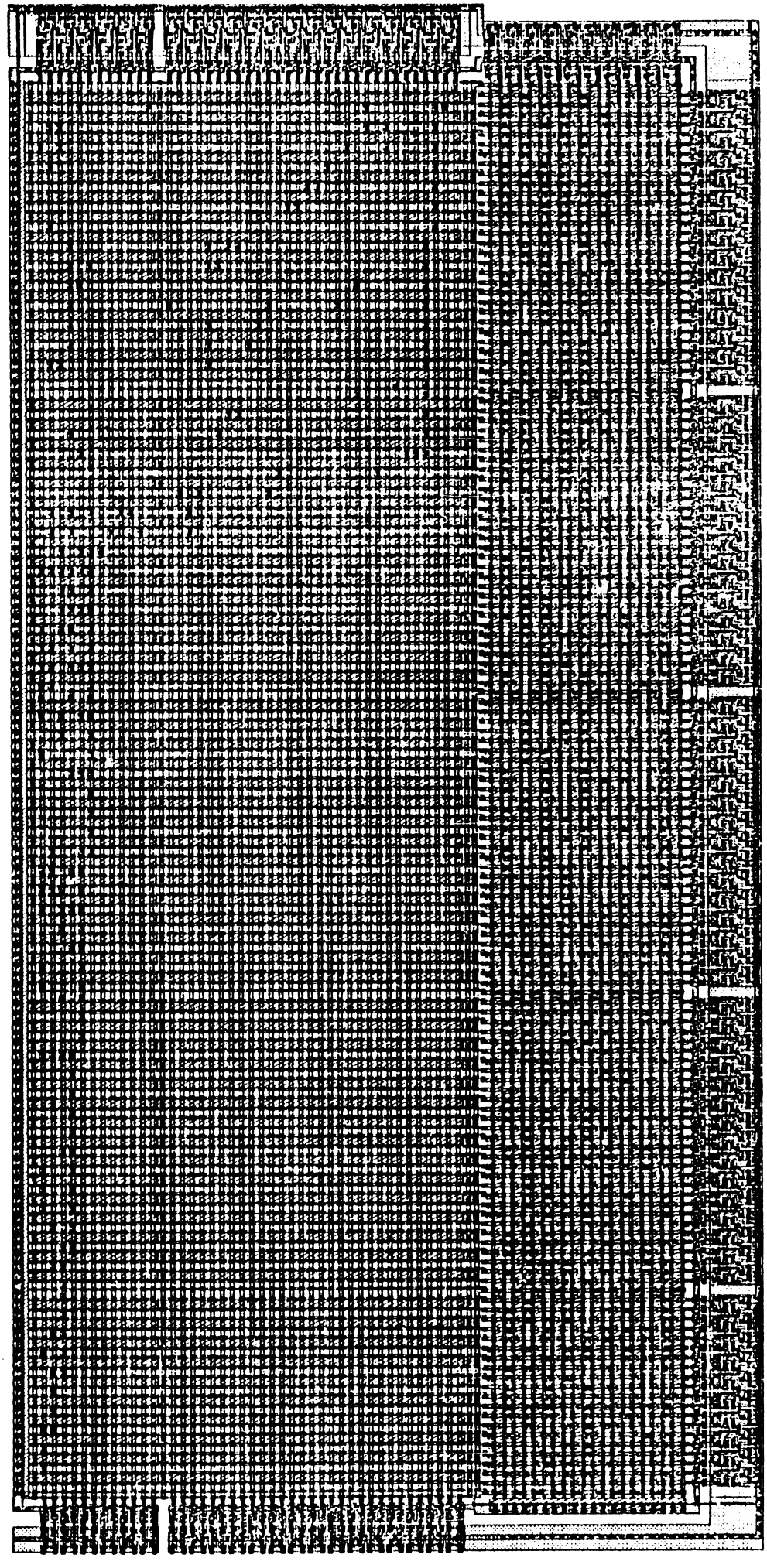
Nombre d'entrées : 10
Nombre de sorties : 42
Nombre de monomes : 139

Surface du PLA = 1.03 mm 2

Surface estimée de l'automate = 1.7 mm2
(PLA + Bascules état suivant + Connexions)

Surface estimée du controleur de HSURF = 7mm2

➡ **25% de la surface du contrôleur**



VII.5.4 - Autre possibilité de réalisation

La réalisation précédente impose d'intégrer l'ensemble de l'automate au sein de HSURF, ce qui représente une augmentation non négligeable de la surface de silicium utilisée. Une autre approche consiste à intégrer dans le microprocesseur seulement la partie de l'automate réalisant la génération des commandes.

Nous avons alors le schéma suivant (Figure 24) :

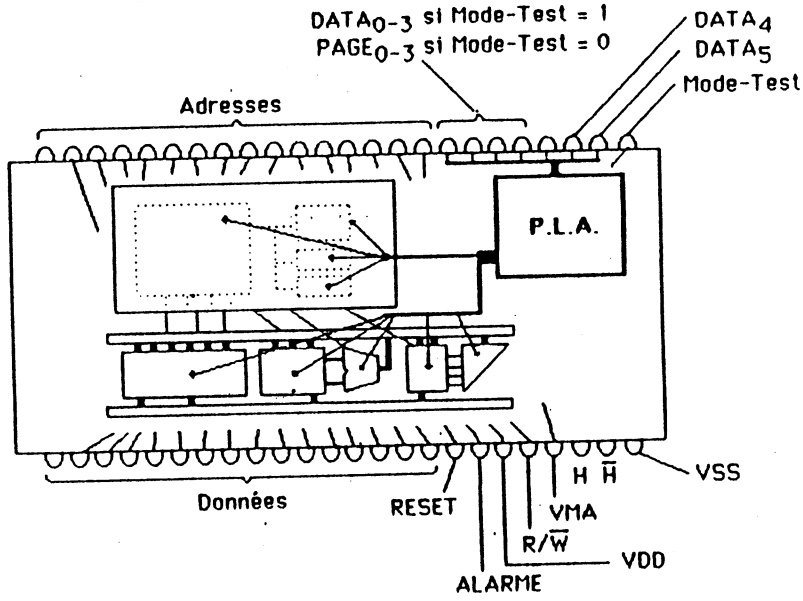
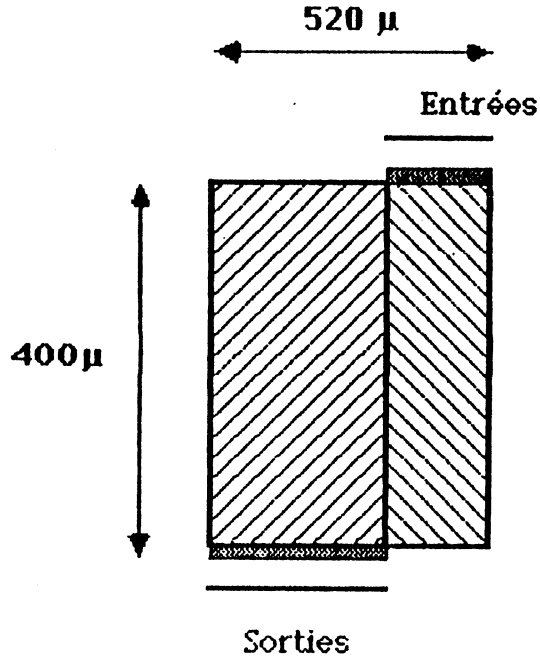


Figure 24

Le contrôle de séquencement du test est alors réalisé directement de l'extérieur par l'équipement automatique de test. Cette solution permet de bénéficier d'une plus grande souplesse au niveau des opérations de test réalisées (Dans ce cas, le graphe de contrôle de "l'automate de test" n'est plus figé ; il peut être programmé directement sur l'équipement automatique de test) et nécessite moins de surface de silicium. Le choix de cette méthode demande l'ajout au niveau externe de 6 broches d'entrée/sortie supplémentaires destinées à indiquer le numéro de la commande a générer.

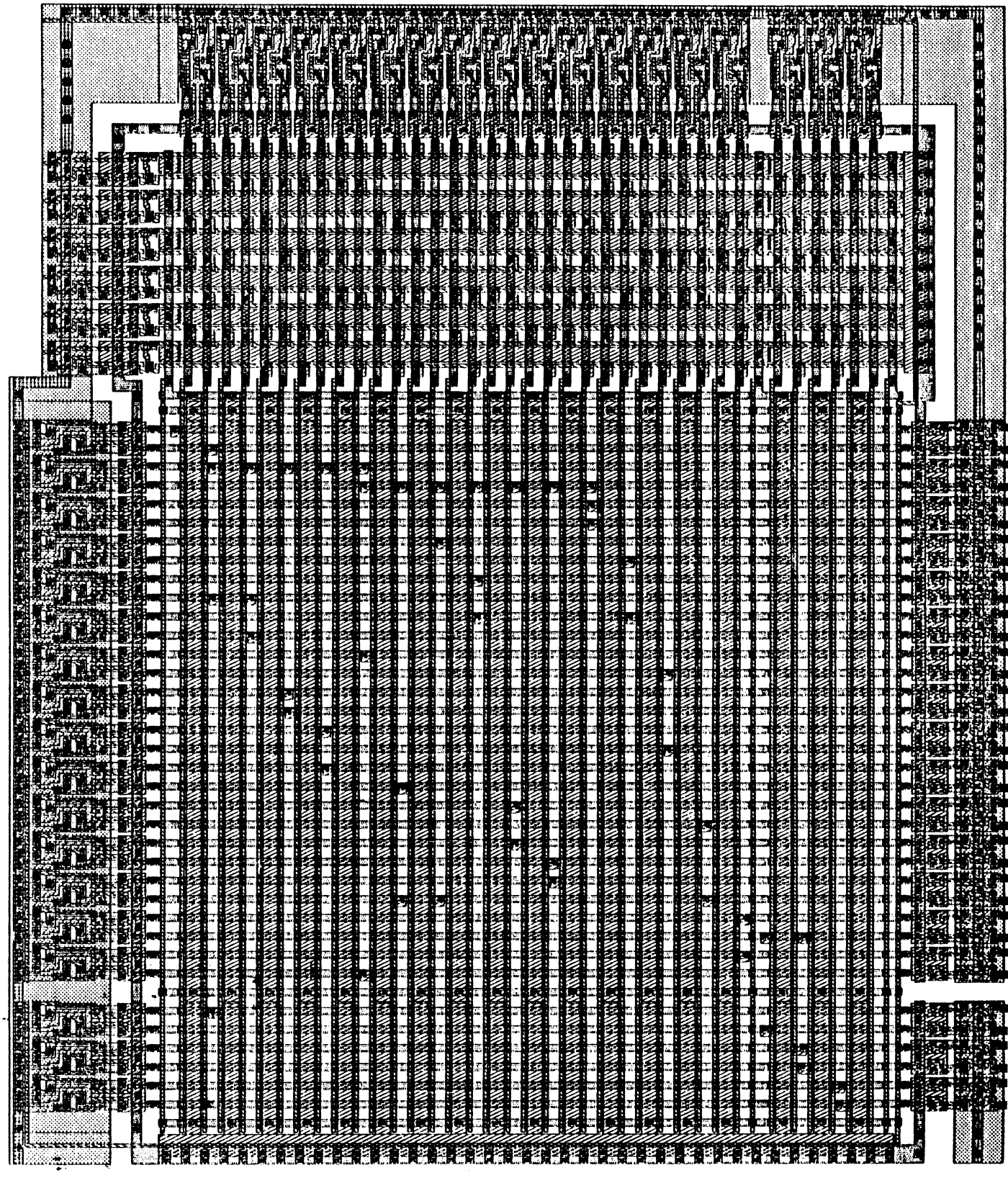
Les résultats d'une telle implantation sont donnés page suivante.



Nombre d'entrées : 6
Nombre de sorties : 36
Nombre de monomes : 36

Surface du PLA = 0.21 mm ^{2}
Surface estimée de l'automate = 0.3 mm2
Surface estimée du contrôleur de HSURF = 7mm2

➡ **4.5% de la surface du contrôleur**


NUMBER OF VECTORS 6629185

VII.6 - CONCLUSION

Le test hors-ligne du microprocesseur HSURF impose de disposer d'un accès direct en un nombre important de ses points internes. La solution proposée consiste à intégrer au sein même du circuit un dispositif permettant de réaliser ces acces. Deux solutions ont été étudiées, l'une se traduisant par l'intégration d'un automate complet de test (génération des commandes et des adresses nécessaires au séquencement des opérations de test), l'autre par l'intégration de seulement la partie génération des commandes. La réalisation des PLAs de l'une et l'autre des solutions nous enseigne que le meilleur rapport efficacité/surface est obtenu avec la deuxième solution. Il est cependant important de noter que cette constatation est étroitement liée au fait que dans notre cas le nombre de commandes nécessaires est assez faible (39 commandes différentes) et, qu'il nous est possible de disposer d'un acces direct en paralelle à toutes les entrées du pla permettant de les générer sans augmenter de façon trop importante le nombre de broches du circuit (2 broches supplémentaires).

VIII - LE TEST EN LIGNE DE HSURF

VIII HSURF EST UN MICROPROCESSEUR FACILITANT LES OPERATIONS DE TEST EN LIGNE

Le microprocesseur HSURF dispose d'une latence de panne faible, c'est en effet l'un des objectifs principaux demandés à un circuit destiné à un environnement à haute sûreté de fonctionnement. La structure et les dispostifs de test hors-ligne décrits dans le paragraphe précédent nous permettent de disposer d'un composant sans faute à l'instant t=0 ; il apparaît maintenant indispensable de détecter toute panne pouvant survenir lors du fonctionnement de l'application. Le microprocesseur dispose donc d'une architecture et de dispositifs logiciels permettant de faciliter les opérations de test en ligne.

En fait, on peut distinguer :

- Le test en ligne discontinu qui est réalisé périodiquement pendant les périodes d'oisiveté du programme d'application. Il consiste généralement à dérouler des programmes de test vérifiant la conformité de certaines ressources particulières au système. Ces tests sont en général du type déterministes (les opérandes sont prédéterminés d'après la structure de l'organe à tester).

- Le test en ligne continu qui se déroule sans interruption "apparente" du programme d'application et sans exercer d'influence sur le process en cours d'éxécution.

De nombreux travaux [KAN 75], [PAR 77], [AYA 79], [MET 81], [SRI 82], [SHE 83b] se sont interessés aux mécanismes de détection en ligne. Ces mécanismes regroupés sous le terme de test en ligne sont étudiés, soit pour

assurer la détection à moindre coût que la réplication, soit pour compléter l'efficacité de la détection dans un système dupliqué.

Dans [SAU 85] on apprend que l'ensemble de ces mécanismes est développée à partir des deux concepts suivants :

- Notion de système de référence (ASR : Authorative System Reference).
- Notion de niveau d'abstraction (ou d'univers).

La notion de bon fonctionnement d'un système est établie par rapport à une référence constituée d'un ensemble de spécifications ; cette référence appelée système de référence peut être spécifiée à différents niveaux (niveau application consistant à décrire ce que doit réaliser le système du point de vue de l'utilisateur, niveau informationnel décrivant comment l'application est réalisée par un ensemble d'algorithmes, niveau logique s'interessant à la structure et à la fonctionnalité des blocs logiques matériels, niveau physique décrivant la réalisation physique des entités logiques).

A partir de ces concepts, on définit la notion "d'évènement indésirable" [AVI 82] qui suivant le niveau auquel il est caractérisé sera défini en terme de défaut, panne ou erreur [LAP 82]. Le mécanisme de détection peut donc être caractérisé par le niveau auquel il s'applique, c'est à dire par le niveau des évènements indésirables qu'il détecte. Une détection parfaite au niveau i serait obtenu si l'on pouvait comparer parfaitement et instantannement l'état du système décrit au niveau i à son ASRi. Cette perfection n'est pas réalisable en pratique (impossiblité de définir formellement un ASR et d'observer entièrement un système). La technique de réplication représente une approche de cette perfection : les différentes unités constituent des images de l'ASR et, on estime moyennant certaines précautions lors de la conception (isolation physique des unités ...) qu'elles ne peuvent être affectées simultanément par les mêmes évènements indésirables.

Cette approche parait interressante mais dans certains cas, on préferera faire l'étude de dispositifs de test en ligne plus facilement implantables (mais moins performants et aux capacités de détection imparfaite) soit pour éviter une réplication paraissant coûteuse eût égard aux objectifs de détection, soit pour compléter la technique de réplication : l'usage de tests en ligne à tous niveaux permet une détection rapide de certains évènements indésirables avant que ceux ci ne contrarient les sorties. Il s'agit de détecter les pannes "cachées" dans une unité afin d'éviter les évènements de double panne risquant de faire échouer la détection et d'entrainer l'émission de commandes dangereuses.

En fait, [SAU 85] nous enseigne que quelque soit la manière dont elle est implantée, l'opération de test en ligne consiste toujours à vérifier la validité d'une "propriété invariante" toujours vérifiée pour tout état correct du système (mais qui malheureusement peut l'être aussi dans certains états incorrects).

La suite de ce paragraphe consistera donc à définir les différentes stratégies de test en ligne utilisées avec le microprocesseur HSURF ainsi que les dispositifs additionnels permettant leurs mises en oeuvre en fonction des propriétés invariantes que l'on désire vérifier.

VIII-1 - LE TEST EN LIGNE DISCONTINU DE HSURF :

Ici, on s'intéresse à des séquences de test destinées à vérifier la conformité de certains opérateurs internes à HSURF (U.A.L., multiplieur, capacité de mémorisation des différents registres ...). Pour ce faire, on définit un ensemble de programmes et de vecteurs de test destinés à chacuns de ces organes. Ces programmes sont fixes (stockés en ROM), non modifiés au cours de leur exécution et tels que la suite des instructions, des données et des adresses invoquées lors de leur déroulement (dans le cas d'un fonctionnement normal) est unique et connu à l'avance. Ainsi, si l'on réalise une compaction des différentes informations circulant sur les bus internes du microprocesseur pendant l'exécution de ces programmes, nous avons la

certitude d'obtenir un résultat unique et calculable à l'avance (ce résultat constitue en fait l'invariant de l'étape de test).

La stratégie consiste donc, pour les opérations de test discontinu en ligne à définir des programmes de test spécifiques pour chacuns des éléments du circuit concernés et à prédéfinir la valeur de la signature obtenue lors de leur exécution. L'invariance est obtenue aussi bien au niveau de la compaction de la suite des données (codes opérations + opérandes) représentant les programmes qu'au niveau des adresses invoquées au cours de l'exécution de ces programmes.

Au niveau de HSURF, la mise en oeuvre d'une telle stratégie de test s'avère relativement simple du fait de la structure du microprocesseur (accessibilité, contrôlabilité), et des études préalables réalisées lors de la conception des différents opérateurs de la partie opérative. De plus, on aura la possibilité de signer soit la suite des codes opérations du programme, soit le flot des adresses utilisées lors de l'exécution de ce dernier ,toutes les facilités concernant la programmation du dispositif de signature étant intégrées à HSURF :

- Choix du type d'information compactée.
- Choix du polynome diviseur.

On pourra par exemple avoir la structure de programme suivante :

1) PHASE D'INITIALISATION
   du dispositif de signature

   - REG_SIGN := RAZ
   - REG_POL := Val-init1
   - REG_RTIS := Val-init2

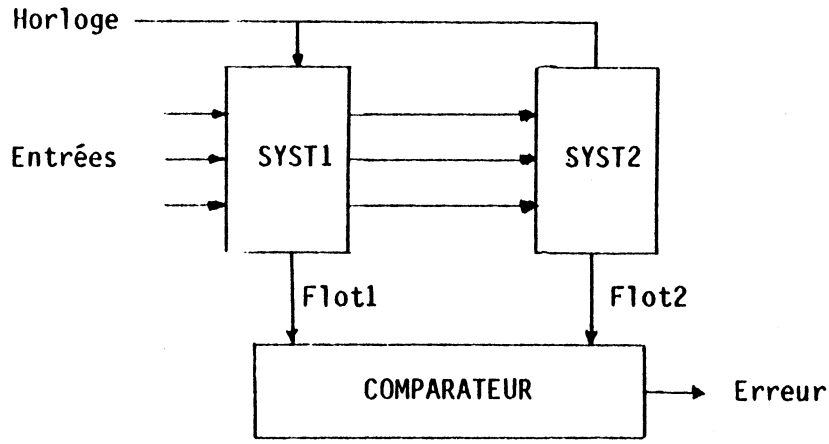
2) EXECUTION du programme de test

3) VERIFICATION du contenu de REG_SIGN, soit par comparaison avec une valeur précalculée (comparaison réalisée par le microprocesseur et envoie d'alarme si une différence est détectée), soit par stockage de la valeur de REG_SIGN à une adresse correspondant à un comparateur externe.

VIII-2 - LE TEST EN LIGNE CONTINU DE HSURF

Ici, la stratégie utilisée sera dépendante de l'architecture du système sur lequel on implante l'opération de test.

i) Le système est composé de plusieurs unités (duplex, triplex ...) qui sont synchronisées au top d'horloge près. Nous nous trouvons alors dans le cas idéal de la réplication totale, on peut donc envisager de réaliser une comparaison continue du flot de données au sein de chaque unité, l'invariant étant ici représenté par le fait que les informations circulant dans chaque système est identique (FLOT-SYST.1 Δ FLOT-SYST.2 = Θ où a Δ b est une opération réalisant la différence entre les termes a et b et où Θ représente un invariant de valeur nulle). Nous pouvons par exemple envisager le schéma suivant :



Une comparaison continue des flots de données peut ne pas être indispensable, on pourra alors réaliser une compaction au sein de chaque unité et vérifier l'égalité de la signature en des points de rendez-vous.

ii) Soit le système est "simplex" (dispositif à sécurité réduite), soit les différentes unités ne sont pas synchronisées (ce cas de figure étant très fréquemment rencontré dans les applications à haute sûreté de fonctionnement où un asynchronisme entre systèmes est intentionnellement réalisé afin d'améliorer l'immunité aux parasites pouvant survenir sur les entrées. C'est le cas du Poste d'Aiguillage

Informatisé de la S.N.C.F.), soit les logiciels implantés dans chaque unité sont différents.

Dans ce cas, les écoulements dans les différents programmes d'application des différentes unités n'ont aucune raison d'être équivalents. De plus, l'exécution d'un programme au niveau d'une unité (ou du système si l'on dispose d'une architecture simplex) est fonction des données reçues en entrée du système. On ne peut donc pas à priori savoir quel chemin sera emprunté lors de la vie du système, il est donc impossible d'envisager de réaliser une compaction du flôt de données internes au système et d'en précalculer la signature.

Nous allons donc dans un premier temps définir un ensemble de règles et de transformations à apporter au logiciel implanté soit au niveau du système dans le cas d'une architecture simplex, soit au niveau de chaque unité dans une arhitecture "multiplex" afin de pouvoir disposer de propriétés invariantes indispensables à l'élaboration de notre politique de test en ligne.

VIII-2-1 - Définitions

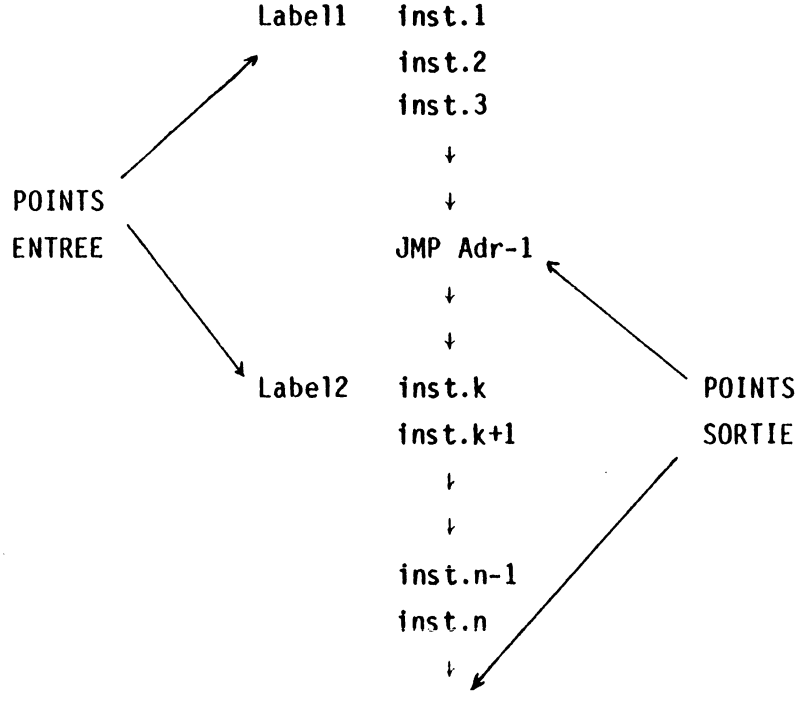
Nous considérerons dans la suite qu'un programme d'application écrit pour le microprocesseur HSURF est réalisé par la mise en séquence de deux types d'instructions :

- Les instructions "normales" ou de branchement (ou de saut) inconditionnel.
- Les instructions de branchement ou de déséquencement conditionnel (rupture de séquence) pouvant entrainer lors de l'éxécution du programme un déséquencement dans la suite des instructions invoquées.

Définition 1 : Une **séquence** sera définie comme une suite d'instructions consécutives. Deux instructions sont dites consécutives si elles sont stockées en des adresses mémoire consécutives ou si elles sont source ou destination d'une instruction de branchement ou de saut inconditionel (il existe un chemin d'écoulement et un seul pour parcourir une séquence).

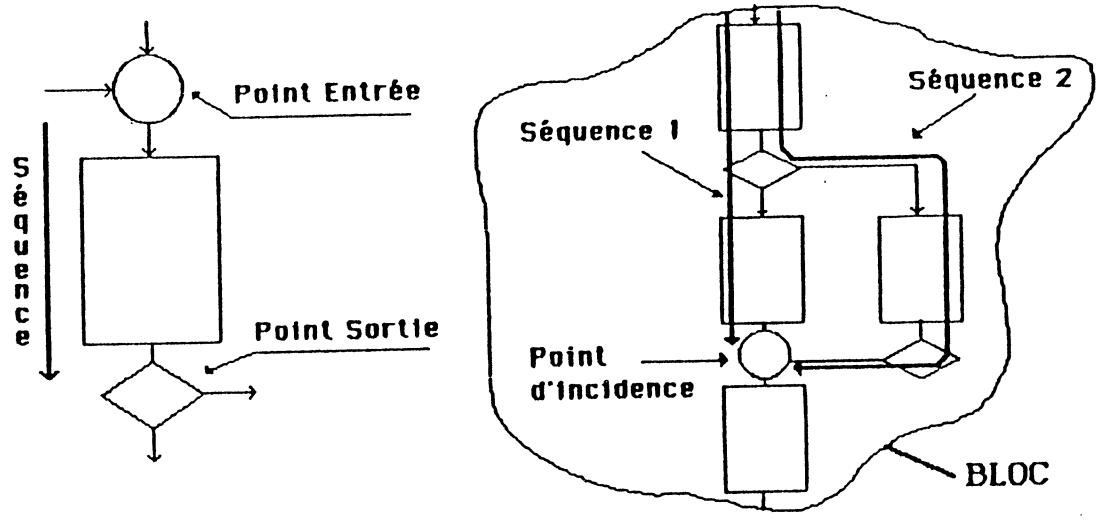
Définition 2 : Un **bloc** sera défini comme la concaténation de séquences d'instructions de rupture de séquence. Un bloc sera accessible par un ensemble de points d'entrée et, on pourra en sortir par un ensemble de points de sortie tels que tous les chemins d'écoulement dans ce bloc (dans le cas d'une exécution correcte du programme) permettent d'aller d'un point entrée à un point sortie.

Exemple :



Définition 3 : On définit un **point d'incidence** comme un point d'entrée pouvant être atteint après l'exécution d'au moins deux séquences différentes.

Nous pouvons éclairer ceci par les schémas suivants :



Définition 4 : On définit par C une fonction de compactage qui s'applique sur une séquence de données, telle que :

C(Séquence-1) = VAL1
C(Séquence-2) = VAL2 avec : VAL1 = VAL2
si Séquence-1 = Séquence-2

Proposition 1 : L'application de la fonction C sur la suite des codes opération d'une séquence donne une valeur et une seule, et il est possible de précalculer cette valeur si l'on a connaissance de la fonction et de la séquence.

En effet, par définition, il n'existe qu'un chemin d'écoulement pour aller du début à la fin d'une séquence, la suite des codes opération rencontrés est donc invariante, donc, l'application de la fonction C sur cet invariant donnera une valeur fixe et une seule.

APPLICATION DIRECTE

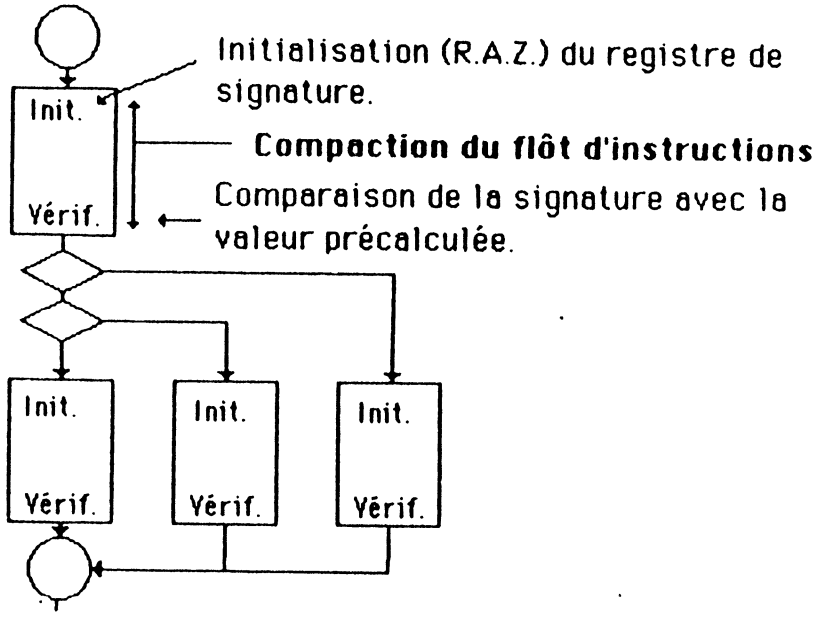
nous pouvons d'après la proposition 1) définir dès à présent une politique envisageable pour réaliser le test en ligne des programmes d'application :

1) La fonction C est réalisée par l'opération de dvision polynomiale.

2) Le programme d'application est partitionné en un ensemble de séquences et, on précalcule pour chacunes d'elles le résultat de la compaction.

3) Pour chacune des séquences précédentes, on initialise le dispositif de signature, on réalise la compaction des codes opération et on compare la signature obtenue à la valeur attendue.

L'application de cette stratégie pourrait par exemple donner :



Parmi les inconvénients de cette méthode, nous pouvons citer :

- On ne réalise que la vérification du bon déroulement de chacune des séquences sans vérifier si leur enchainement est correct (pas de vérification du séquencement global du programme d'application).

- Il est nécessaire d'ajouter pour chaque séquence des instructions correspondantes à la phase d'initialisation du dispositif de compaction et à la vérification de la validité de la signature obtenue, ce qui peut se traduire dans certains cas par une forte augmentation de logiciel.

Définition 4 : Un bloc B sera dit C-Equivalent si et seulement si C(B) = Valeur-fixe quel que soit le chemin d'écoulement pris pour traverser ce bloc.

Soit par exemple, un bloc ayant i flots d'écoulement possibles correspondant à l'exécution des suites d'instructions Suite-1 à Suite-i, nous avons :

$$C(Bloc) = C(Suite\text{-}1) = C(Suite\text{-}2) = \ldots = C(Suite\text{-}i)$$

$$\equiv$$

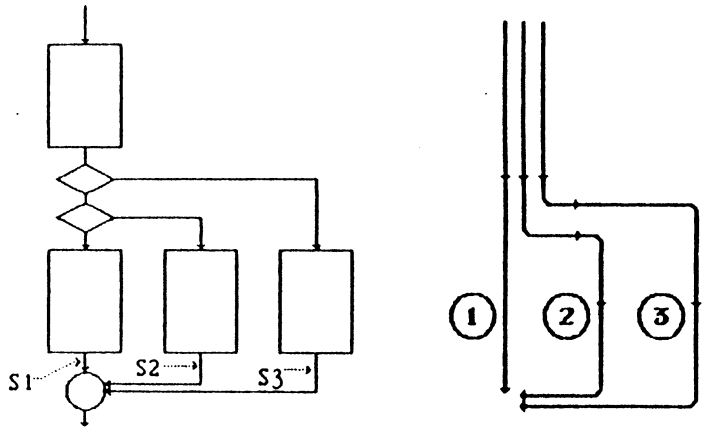
Bloc est C-Equivalent

Définition 5 : On définit une instruction d'ajustement AJ(X) telle qu :

$$AJ(Val1, Val2) = Val3.$$

Proposition 2 : Il est possible de rendre tout bloc C-Equivalent en insérant dans certains de ses chemins d'écoulement une instruction d'ajustement.

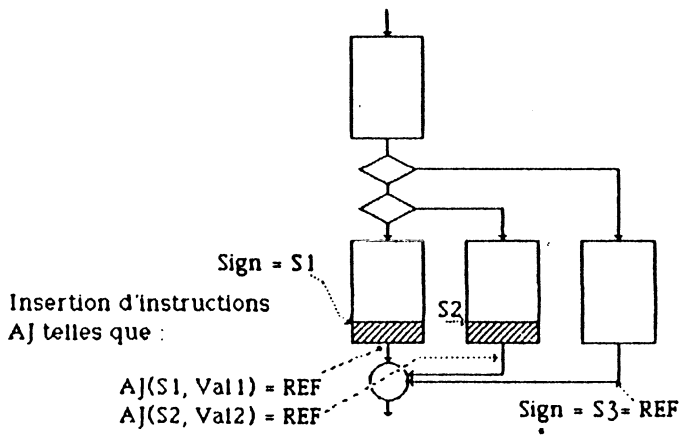
En effet, soit par exemple le bloc hypothétique suivant, nous avons :



Ce bloc à 3 chemins d'écoulement possibles, si on prend comme référence la signature obtenue en parcourant le chemin n° 3, il suffit d'insérer une instruction d'ajustement dans le chemin n° 1 et dans le chemin n° 2 telles que :

C(chemin n°1) = C(chemin n°2) = C(chemin n°3) = REF.

Nous aurons alors :



Théorème : Il est possible de rendre C-Equivalent tout programme d'application constitué par une association de blocs en rendant chacuns de ses blocs C-Equivalent.

Une stratégie de test en ligne pourra donc consister à rendre le programme d'application C-Equivalent et, à vérifier pour chacun de ses blocs la correction de la signature.

VIII-2-2 - Mise en oeuvre pratique avec HSURF

Ici, la fonction de compaction sera la division polynomiale, l'opération réalisée sera soit la compaction des codes opération du programme d'application, soit la compaction des adresses de la zone programme (la programmation du registre RTIS de HSURF permet de définir facilement le type des informations signées). L'instruction d'ajustement sera AJS (ou AJST) réalisant l'opération OU-exclusif entre le contenu du registre de signature et une valeur immédiate :

$$\text{Reg-SIGN} := \text{Reg-SIGN} \oplus \text{Val-Imm.}$$

Mode d'utilisation des instructions d'ajustement

Nous donnons dans la suite pour différentes structures de contrôle de langage de haut niveau couramment utilisées ("SI cond FAIRE action1 SINON action2", "TANTQUE cond FAIRE action", "REPETER action JUSQU'A cond.") une stratégie applicable pour insérer les instructions d'ajustement de façon à rendre les blocs C-Equivalents puis, nous en tirons une règle permettant de résoudre le problème pour n'importe quelle structure.

Structure SI condition ALORS action1 SINON action2 (Figure 1)

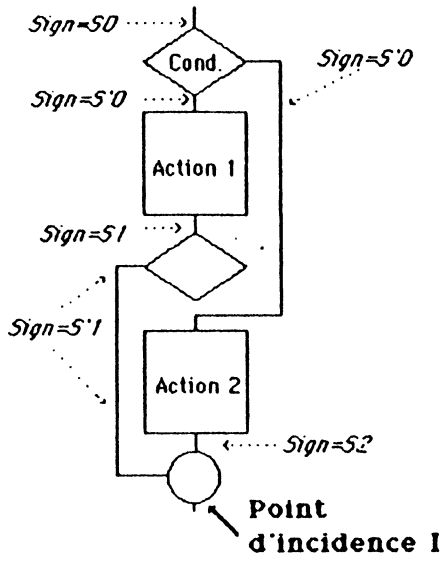


Figure 1

Pour une telle structure, la compaction de codes instruction donne les valeurs de signature S_0, puis S'_0 (après compactage du code opération correspondant à l'instruction de branchement conditionnel), puis S_1 et S'_1 ou S_2 suivant que l'on a exécuté (et signé) les séquences correspondant à l'action 1 ou à l'action 2.

Il apparait un problème au point d'incidence I, pour le résoudre, deux solutions sont envisageables :

1) Ajouter une instruction d'ajustement à la fin de la séquence correspondant à l'action 1 afin que la signature suivant le chemin A donne S_2 au lieu de S'_1 (cf. Figure 1a).

2) Ajouter une instruction d'ajustement à la fin de la séquence correspondant à l'action 2 afin que C("Chemin B") = S'_1 au lieu de s_2 (cf. Figure 1b).

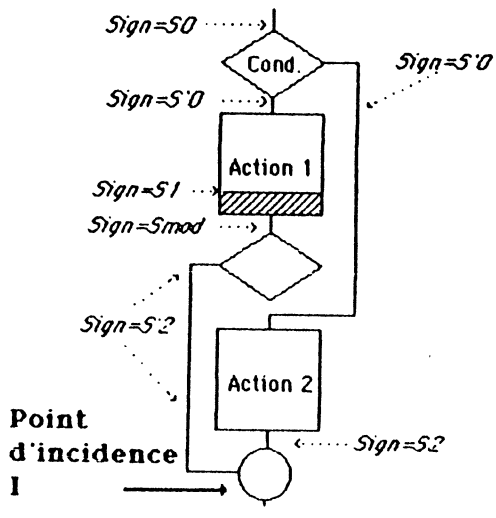


Figure 1a

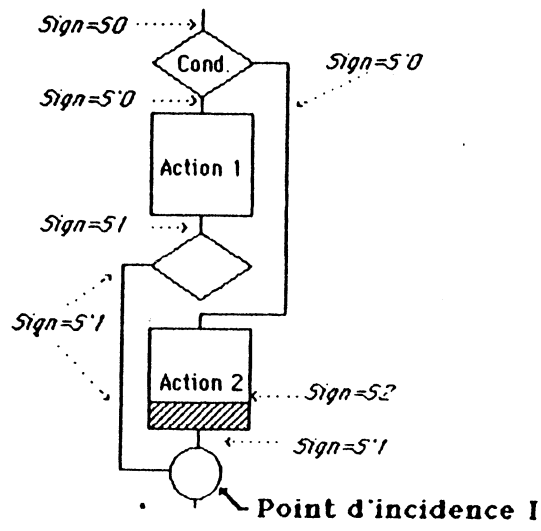


Figure 1b

On s'aperçoit que la solution n°1 demande de réaliser l'ajustement en tenant compte du fait que l'instruction de branchement inconditionnel sera signé aprés l'opération d'ajustement. On devra donc calculer une valeur VAL_1 telle que :

$$AJ(S_1, VAL_1) = S_{mod} \text{ avec } C(S_{mod}, inst\_branc\_incond) = S_j$$

où : $C(S_i, INST_j)$ représente le résultat de la compaction de la suite d'instructions ayant donnée S_i concaténée avec l'instruction $INST_j$.

La solution n°2 demande quant à elle de calculer la valeur VAL_2 telle que l'on ait :

$$AJ(S_2, VAL_2) = S'_1.$$

Structure SI Condition ALORS Action (Figure 2a)

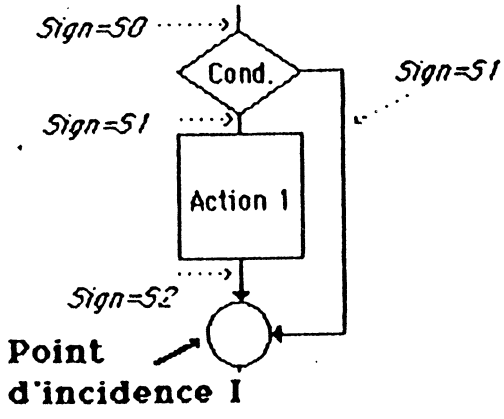


Figure 2a

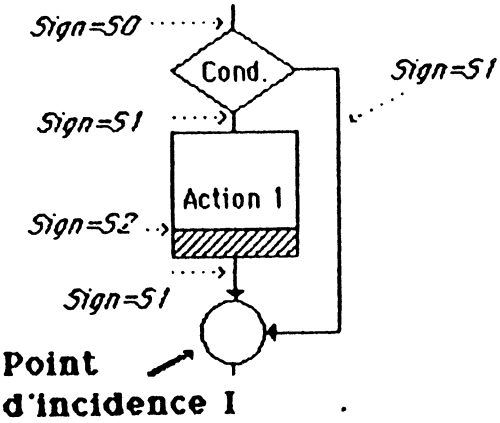


Figure 2b

Ici, le conflit au point d'incidence I peut se résoudre en insérant l'instruction d'ajustement juste avant le point I à la fin de la séquence d'actions (Figure 2b).

Structure REPETER Action JUSQU'A Condition (Figure 3a)

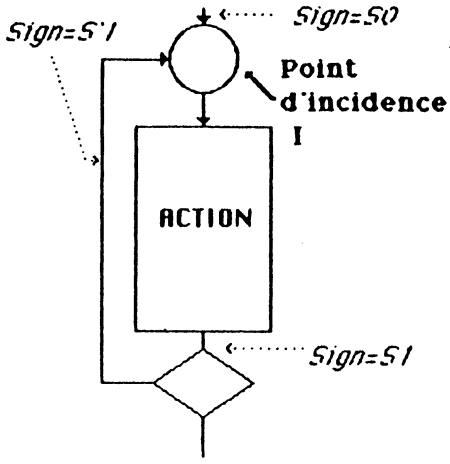


Figure 3a

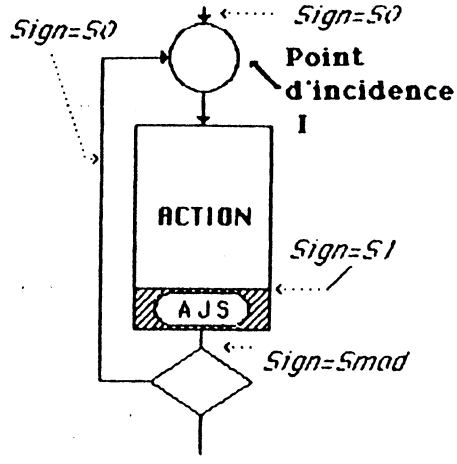


Figure 3b

Même problème au point I, il suffit ici d'ajouter une instruction d'ajustement à la fin de la séquence d'instructions (Figure 3b).

Structure TANTQUE Condition FAIRE Action (Figure 4a)

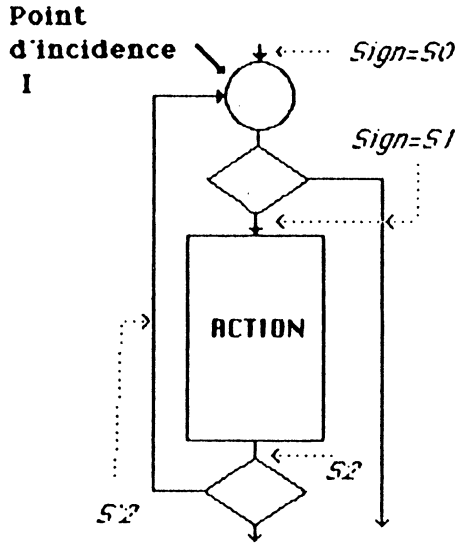


Figure 4a

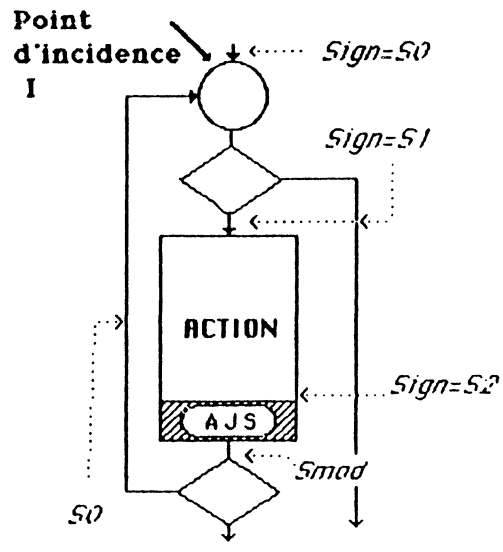


Figure 4b

La solution apparait en Figure 4b, il faudra faire attention ici lors du calcul de la valeur immédiate à associer à l'instruction d'ajustement (même problème que pour la structure SI-SINON solution n°1).

Structure CAS : SI $Cond_1$ FAIRE $Action_1$
SI $Cond_2$ FAIRE $Action_2$

SI $Cond_i$ FAIRE $Action_i$

Les résultats sont donnés dans les figure 5a et 5b suivantes :

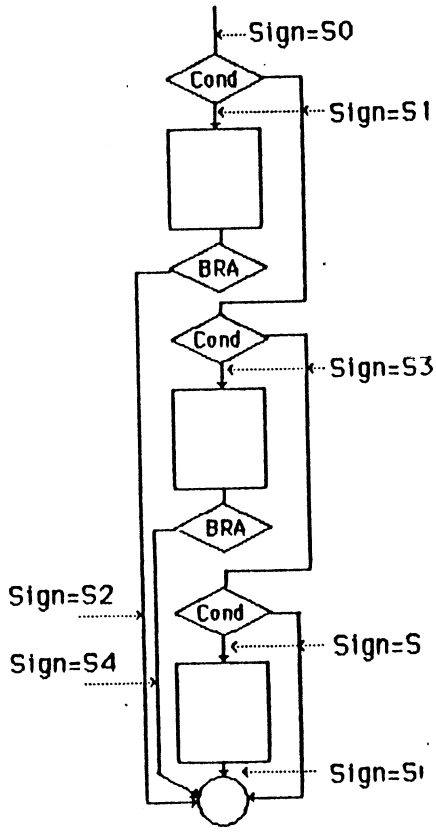


Figure 5a

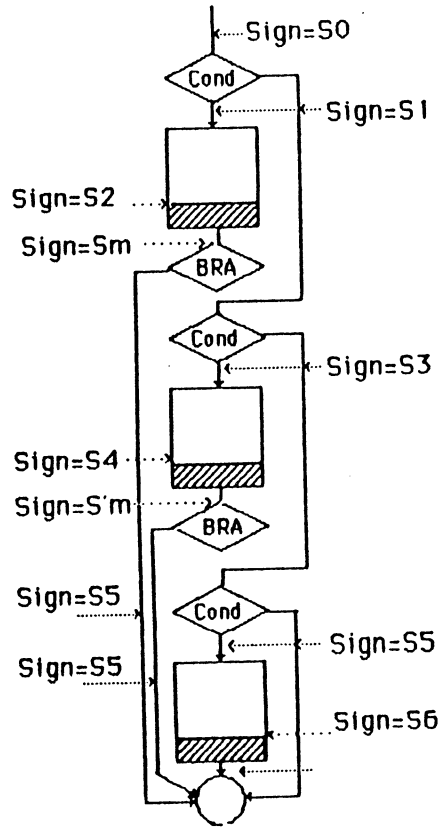


Figure 5b

Remarques : On s'aperçoit par les exemples précédants, qu'il est toujours possible de rendre un bloc, dont la structure est calquée sur une structure de langage de haut niveau, C-Equivalent. Il apparait que si il existe n chemins d'écoulement possibles pour traverser le bloc, il suffit d'insérer n-1 instructions d'ajustement pour rendre ce bloc C-Equivalent ; cependant, il n'existe pas à priori (à moins de savoir reconnaître la structure de langage haut

niveau utilisée dans le programme d'application) de méthode permettant de décider où l'on doit insérer l'instruction d'ajustement :

- Directement avant le point d'incidence (cas des structures SI/SINON Solution 2, SI).

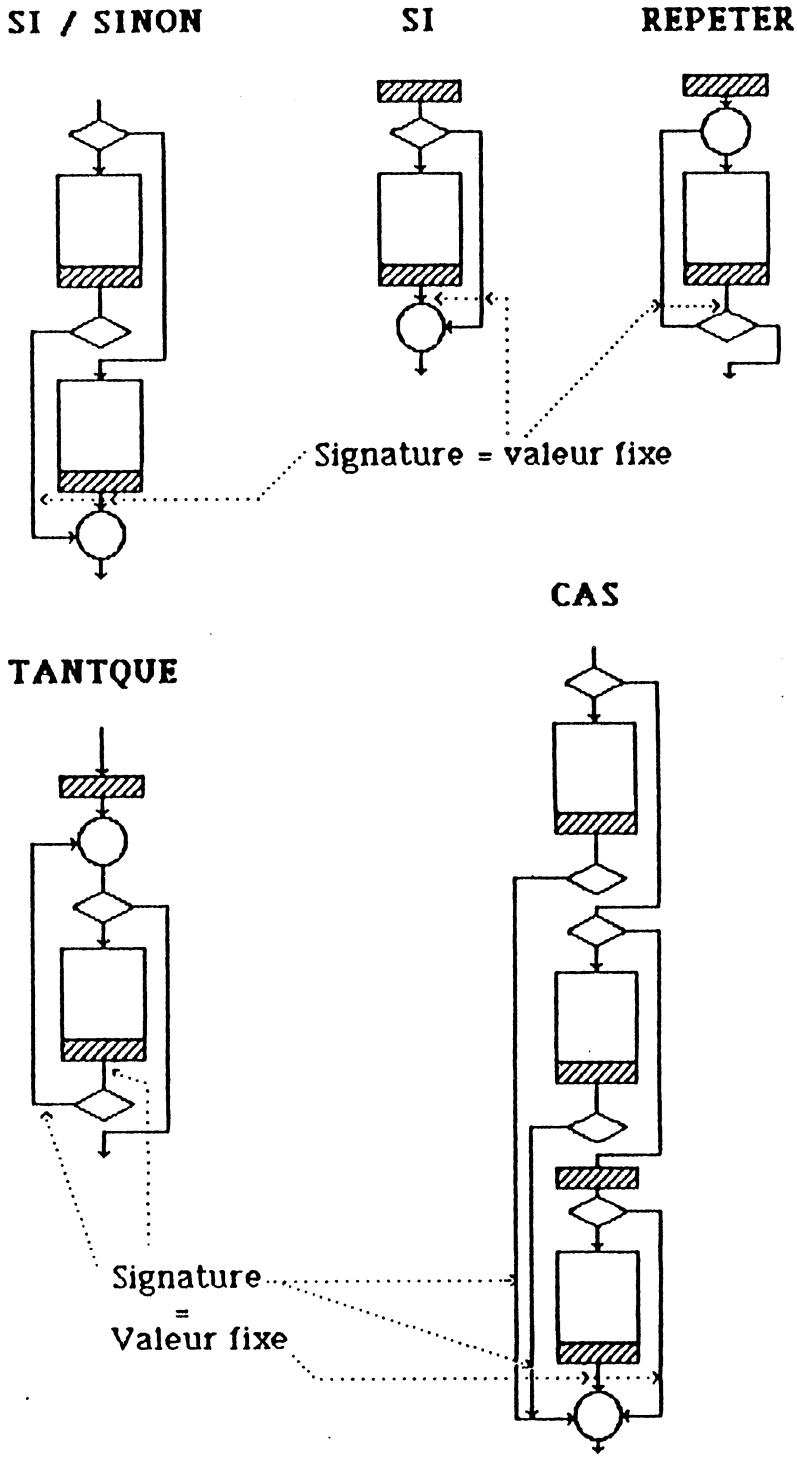
- Avant l'instruction de branchement ou de saut précédant le point d'incidence (cas des structures SI/SINON Solution 1, REPETER/JUSQU'A).

- Soit avant les uns et les autres (cas de la structure CAS).

De ce fait, il n'apparait pas évident de développer un dispositif logiciel automatique réalisant la modification du programme d'application et calculant les valeurs des signatures et des valeurs immédiates qu'il faut associer aux instructions d'ajustement de façon à rendre le programme C-Equivalent.

Afin de rendre l'opération plus cohérente, nous prendrons comme stratégie d'insérer systématiquement des instructions d'ajustement sur chacuns des chemins d'écoulement arrivant à un point d'incidence. Cette approche, légerement plus couteuse en logiciel (le nombre d'instructions d'ajustement ajoutées est plus important car si le bloc peut être traversé par n chemins d'écoulements, il faudra ajouter n instructions contre n-1 dans la solution précédante) nous permet d'envisager un traitement automatique du programme d'application dès la phase d'assemblage de façon à rendre la signature en chaque point d'écoulement invariante.

Les résultats de l'application de cette méthode aux structures précédantes sont donnés dans la page suivante.


SI / SINON
SI
REPETER
Signature = valeur fixe
CAS
TANTQUE
Signature
=
Valeur fixe

VIII-2-3 - Utilisation des instructions AJS et AJST

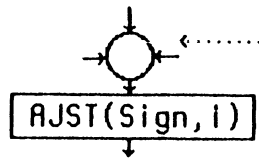
Nous avons vu que l'insertion des instructions d'ajustement permettait de rendre la signature invariante en tous les points d'incidence du programme. La valeur de cette dernière sera fonction des valeur immédiates associées aux instructions d'ajustement de chaque chemin d'écoulement. Le fait que cette valeur puisse être fixée par l'utilisateur nous permet d'envisager l'utilisation en certains points de l'instruction AJST dont la fonction est de réaliser un ajustement suivi d'une comparaison du contenu du registre de signature avec la valeur "tout à zéro" ; il suffit en effet de précalculer les valeurs immédiates de telle façon que la signature aux points d'incidence soit nulle, l'utilisation d'instructions AJST au lieu d'instructions AJS en ces points réalisera alors en une seule étape l'ajustement et la vérification de la signature.

Une autre approche peut consister à numéroter chaque point d'incidence du programme et à initialiser les valeur immédiates associées aux instructions d'ajustement de façon à ce que la signature au point d'incidence i ait la valeur i. L'insertion en ces points d'une instruction supplémentaire AJST telle que :

AJST(Signature-au-point-n°i,VALi) = "0"

permet alors de vérifier automatiquement la bonne exécution du programme d'application ; Cette stratégie permet en plus de vérifier la cohérence du séquencement entre les différents blocs du programme (cf. exemple ci-dessous).

Si on choisit de donner à la signature une valeur correspondant au n° dincidence, il suffit d'ajouter après chaque point d'incidence :



Point d'incidence n° : i.

. l'instruction AJST (ou AJS) réalise un ou-exclusif bit à bit entre le contenu du registre de signature et la valeur immédiate. Avec cette stratégie, le calcul de la valeur immédiate associée à l'instruction AJST devient alors évident : la signature au point n°i a normalement pour valeur i, il suffit donc de donner à VALi la valeur i pour que la disjonction donne la valeur "tout à zéro".

VIII-2-4 - Un logiciel de transformation de programmes d'application

Un logiciel permet de modifier de façon automatique tout programme d'application de façon à le rendre C-Equivalent et à permettre la mise en oeuvre de notre politique de test en l'gne continu. A partir du programme d'application écrit en en langage assembleur, nous générons un nouveau texte source contenant les instructions d'ajustement et les valeurs immédiates associées nécessaires. La modification du programme d'application est réalisée avant la phase d'assemblage du programme. L'opération se déroule en trois étapes distinctes :

- Recherche des points d'incidence
- Insertion des instructions d'ajustement
- Calcul de la signature des différentes séquences et calcul des valeurs immédiates à associer aux instructions d'ajustement.

La mise en oeuvre de notre stratégie de test en ligne continu s'avère donc relativement simple. Il faut cependant noter que quelle que soit la solution choisie, elle devra être complétée par la mise en place dans le système de dispositifs tels que "chien de garde temporel" destinés à éviter le problème des "boucles infinies" (ce type de vérification n'étant pas réalisé directement avec les méthodes présentées précédemment).

VIII-3 - CONCLUSION

La présence dans le microprocesseur HSURF des instructions spécialisées AJS et AJST, et du dispositif de compaction (avec toutes les facilités de programmation qui lui sont associées) facilite la réalisation des opérations de test en ligne. Une des stratégies d'utilisation de ces dispositifs a été développée ; elle permet de réaliser un test en ligne continu efficace, ne necessite pas l'ajout d'un important matériel supplémentaire et s'avère d'une mise en oeuvre relativement simple du fait que le programme d'application peut être modifié de façon automatique. Pour ce qui est du test en ligne discontinu, l'accessibilité totale à tous les points internes du microprocesseur et, la présence dans ce dernier d'opérateurs conçus de façon à faciliter leur test permettent de développer des routines de test consises et efficaces.

Enfin, nous signalerons, que le dispositif de compaction autonome que nous avons cité au chapitre IV.1 [JAY 85] dispose de fonctions dont l'effet est équivalent aux instructions AJS ou AJST. L'utilisation de ce circuit dans des systèmes contrôlés par un autre type de microprocesseur que HSURF permet donc d'envisager l'application de stratégies de test en ligne continu équivalentes à celles présentées dans ce chapitre.

IX - CONCLUSION GENERALE

L'utilisation de composants du type microprocesseur dans des applications à haute sûreté de fonctionnement a révélée des dificultés particulières quant à la fiabilité et à la validation des circuits mis en oeuvre dans de tels systèmes.

Nous avons montré que le problème concernant la validation nécessite de disposer de composants ayant une latence de panne faible et pouvant être reconnus sains à l'instant t=0.

Pour atteindre cet objectif, un ensemble de techniques et de méthodes sont proposées ; elles ont essentielement pour but d'améliorer l'observabilité et la contrôlabilité du circuit. Ceci se traduit au niveau architectural, par la réalisation d'une partie opérative régulière ("tranche de bit") disposant d'opérateurs spécialement étudiés en vue d'un test complet et rapide, par l'implantation au sein de la partie opérative et de la partie contrôle de dispositifs d'observation (compaction) et par l'intégration dans le circuit d'un automate de test permettant d'en contrôler les points clés. De même, au niveau logiciel, et apparut un nouveau type d'instructions destinées à faciliter les opérations de test du circuit (LRS, SRS, TUAL ...) ainsi que du logiciel d'application (AJS, AJST).

L'ensemble de ces techniques et méthodes est appliquable lors de la réalisation de n'importe quel microprocesseur. La méthodologie proposée pour la conception des opérateurs "facilement testables" de la partie opérative de HSURF est aisement transposable à tous types d'opérateurs dont la structure est régulière.

Aujourd'hui, la moindre application de contrôle utilise un microprocesseur, les problèmes posés par l'exemple particulier du poste d'aiguillage informatisé deviennent donc à des degrés plus ou moins importants des problèmes quotidiens. La nouvelle génération de circuits V.L.S.I. devra donc disposer d'architectures facilitant leur test.

Il faut cependant garder à l'esprit que l'ajout de dispositifs destinés à faciliter le test du circuit ne doit pas se traduire par une augmentation de la surface du composant telle qu'elle puisse entrainer une dégradation de sa fiabilité. Une attention toute particulière devra donc être portée dès les étapes de conception, de façon à réaliser un compromis entre le bénéfice apporté par les dispositifs de test et l'accroissement de la complexité du circuit ; pour le cas de HSURF, 10 à 15% de surface de silicium supplémentaire est un bon compromis.

REFERENCES

[ABO 84] **E.M. ABOULHAMID, E. CERNY**
"Built-in testing of One-Dimensionnal Unilateral Iterative Arrays"
IEEE Trans. on Comp., Vol C-33, Juin 1984, pp. 560-564.

[ARC 84] **E. ARCHAMBEAU, E.J Mc CLUSKEY**
"Fault coverage of pseudo exhaustif testing"
Proc FTCS 14, Juin 1984, pp. 141-145.

[ARC 85] **E. ARCHAMBEAU**
"Test fonctionnel des circuits intégrés digitaux"
Thèse I.N.P-Grenoble, Octobre 1985.

[ASH 77] **M. ASHJEE, S. REDDY**
"On totally self-checking checkers for separable codes"
IEEE Trans. on Comp., Vol C.20, Août 1977, pp. 737-745.

[ATE 83] **S. SADIER**
"Les équipements automatiques de test"
Formation Continue I.N.P-Grenoble sur le test et la testabilité des circuits complexes. 25-29 Avril 1983. Fascicule 1.

[AVI 82] **"A. AVIZIENIS**
"The Four Universe Information System Model for the study of fault tolerance"
Proc. FTCs 12, Juin 1982, pp. 6-13

[AYA 79] **J. AYACHE, P. AZEMA, M. DIAZ**
"Observers : A concept for on-line detection of control errors in concurrent systems".
Proc. FTCS 9, Juin 1979, pp. 79-86.

[BAS 85] **D. BASCHIERA, B. COURTOIS**
"Testing s-open of cmos gates and networks"
Rapport de recherche IMAG n° 533, Mai 1985.

[BEL 82] C. BELLON, G. SAUCIER
"Protection against external errors in a dedicated system"
IEEE Trans. on Comp., Vol. C-31, Avril 1982, pp. 311-317.

[BEL 84] C. BELLON
"Le test fonctionnel de circuits intégrés complexes"
Thèse d'état, USMG/INPG, Octobre 1984.

[BOS 82] B. BOSE, T.R.N. RAO
"Theory of unidirectionnal Error Correcting/Detecting codes"
IEEE Trans. on Comp., Vol C-31, June 1982, pp. 520-530.

[CER 83] E. CERNY & All
"Built in testing of pI testable iterative arrays"
Proc. FTCs 13, Juin 1983, pp. 33-36.

[CHA 82] J. CHAVADE, Y. CROUZET
"The P.A.D. : A self checking LSI circuit for fault detection in micro-computer"
Proc. FTCS 12, Juin 1982, pp. 55-62.

[CHA 83] R. CHANDRAMOULI
"On testing stuck-open faults"
Proc. FTCS 13, Juin 1983, pp. 258-265.

[CHE 85] W.T. CHENG, J.M. PATEL
"A minimum test set for multiple-fault detection in simple carry adders"
ICCD 1985, pp. 435-438

[CMP] F. ANCEAU
"The french MPC project"
Université Paris VI, Juin 1981.

[COU 79] **B. COURTOIS**
"Some results about the efficiency of simple mechanisms for the detection of microcomputer malfunctions"
Proc. FTCS 9, Juin 1979, pp. 71-74.

[COU 81] **B. COURTOIS**
"A methodology for on line testing of microprocessors"
proc. FTCS 11, Juin 1981, pp. 272-274.

[CRO 80] **Y. CROUZET, C. LANDRAULT**
"Design of a self checking Detection processor"
Proc FTCS 10, Octobre 1980, pp. 275-277.

[DAV 78] **R. DAVID**
"Feedback shift repister testing"
Proc. FTCS 8, Juin 1978, pp. 103-107.

[DEC 85] **P. DE CHOUDENS**
"Test intégré de processeur facilement testable"
Thèse I.N.P-Grenoble Nov. 1985

[DIA 75] **M. DIAZ, J. MOREIRA DE SOUZA**
"design of self checking microprogrammed controls"
Proc FTCS 5, Juin 1975, pp. 137-142.

[DIA 76] **F. DIAS**
"Truth table verification of an iterative logic array"
IEEE Trans on Comp., Vol C25, n°6, Juin 1976, pp. 605-613.

[FRI 73] **A.D. FRIEDMAN**
"Easily testable iterative systems"
IEEE Trans on Comp., Vol C22, decembre 1973, pp. 1061-1064.

[GAJ 82] **D.O. GAJSKI**
"The structure of a silicon compiler"
ICCD 82, pp. 272-276.

[HAR 78] **H. HAROUF, A. FRIEDMAN**
"Efficiently designed self checking for any m-out-of-n code"
IEEE Trans. on Comp., Vol C.27, Juin 1978, pp. 482-490.

[HAS 82] **S.Z. HASSAN, D.J. LU, E.J. Mc CLUSKEY**
"Parallel signature analyzers, detection capability and extensions"
CRC technical report, n° 82-20, Dec 1982, pp. 1.6.

[HAS 84] **S.Z. HASSAN, E.J. Mc CLUSKEY**
"Increased fault coverage through multiple signature"
proc FTCS 14, Juin 1984, pp. 354-360.

[HAY 76] **J.P. HAYES**
"Transition count testing of combinational logic circuit"
IEEE Trans. on Comp., Vol C.25, Juin 1976, pp. 613-620.

[IYE 82] **S. IYENGAR, L. KINNEY**
"Concurrent testing of flow of control in simple microprogrammed units"
Proc ITC, Novembre 1982.

[JAY 84] **C. JAY, G. SAUCIER, P. GENESTIER**
"A testable microprocessor for process control"
ICCD 1984, pp. 284-289.

[JAY 85] **C. JAY, P. CHAISEMARTIN**
"Conception d'un circuit de signature"
1er colloque national de circuits à la demande, Grenoble, Mai 1985 pp. 131-147.

[KAN 75] **J. KANE, S. YAU**
"Concurrent Software fault detection"
IEEE Trans. Software Engineering, Vol. SE-1, Mars 1975, pp. 87-99

[LAP 82] **J.C. LAPRIE, A. COSTES**
"Dependability : A unifying concept for reliable computing"
Proc. FTCS 12, Juin 1982, pp. 18-21

[LAT 79] **Bill LATIN**
"VLSI design methodology, the problem of the 80's for microprocessors design"
ITC 1979, pp. 548-549.

[LTW 84] **L.T. WANG, E.J. Mc CLUSKEY**
"Linear feedback shift register design for VLSI system testing"
Proc. FTCS 14, Juin 1984, pp. 360-365

[MAK 82] **G.P. MAK, J.A. ABRAHAM, E.S. DAVIDSON**
"The design of PLAs with concurent error detection"
Proc. FTCS 12, Juin 1982, pp. 303.310

[MAR 82] **P. MARCHAL, B. COURTOIS**
"On detecting the hardware failures disrupting programs in microprocessors"
Proc. FTCS 12, Juin 1982, pp. 249-256.

[MCC 82a] **E.J. Mc CLUSKEY, M. NAMJOO**
"Watchdog processors and capability checking"
Proc. FTCS 12, Juin 1982, pp. 245-248.

[MCC 82b] **E.J. Mc CLUSKEY**
"Verification testing"
Proc. 19ème DAC, pp. 495-500.

[MET 81] **G. METZE, A. MILI**
"Self checking programs : an axiomatization of program validation by executable assertions"
Proc. FTCS 11, Juin 1981, pp. 102-105.

[LAP 82] **J.C. LAPRIE, A. COSTES**
"Dependability : A unifying concept for reliable computing"
Proc. FTCS 12, Juin 1982, pp. 18-21

[LAT 79] **Bill LATIN**
"VLSI design methodology, the problem of the 80's for microprocessors design"
ITC 1979, pp. 548-549.

[LTW 84] **L.T. WANG, E.J. Mc CLUSKEY**
"Linear feedback shift register design for VLSI system testing"
Proc. FTCS 14, Juin 1984, pp. 360-365

[MAK 82] **G.P. MAK, J.A. ABRAHAM, E.S. DAVIDSON**
"The design of PLAs with concurent error detection"
Proc. FTCS 12, Juin 1982, pp. 303.310

[MAR 82] **P. MARCHAL, B. COURTOIS**
"On detecting the hardware failures disrupting programs in microprocessors"
Proc. FTCS 12, Juin 1982, pp. 249-256.

[MCC 82a] **E.J. Mc CLUSKEY, M. NAMJOO**
"Watchdog processors and capability checking"
Proc. FTCS 12, Juin 1982, pp. 245-248.

[MCC 82b] **E.J. Mc CLUSKEY**
"Verification testing"
Proc. 19ème DAC, pp. 495-500.

[MET 81] **G. METZE, A. MILI**
"Self checking programs : an axiomatization of program validation by executable assertions"
Proc. FTCS 11, Juin 1981, pp. 102-105.

[MUZ 82] **J.C. MUZIO, D.M. MILLER**
"Spectral techniques for faults detections"
Proc. FTCS 12, Juin 1982, pp. 297-302.

[NAM 82] **M. NAMJOO, E.J. Mc CLUSKEY**
"Watchdog processors and capability checking"
Proc. FTCS 12, Juin 1982, pp. 245-248.

[NAM 83] **M. NAMJOO**
"CEREBUS 16° : An architecture for a general purpose watchdog processor"
Proc. FTCS 13, juin 1983 pp. 216-219.

[PAR 77] **B. PARHAMI**
"The concept of Self Checking programs"
Proc. FTCS 7, juin 1977 pp. 216

[PAR 81] **R. PARTHASARATHY, M.R. SUDHAKAR**
"A testable Design of iterative logic Array"
IEEE Trans comp. Vol C30 n°11, Nov 1981 pp 833-841

[PIL 82] **E. PILAUD**
"Conception et Validation de Systèmes informatiques à haute sûreté de fonctionnement"
Thèse de docteur ingénieur I.N.P-Grenoble, novembre 1982.

[ROB 79] **C. ROBACH**
"Test et testabilité des systèmes informatiques"
Thèse d'état, I.N.P-Grenoble, juin 1979.

[SAU 84] **G. SAUCIER, C. JAY**
"Conception d'un microprocesseur Specialisé de Sécurité"
4ème Colloque Int. Fiabilité et maintenabilité
Perros Guirec 1984 pp 561-567

[SAU 85] G. SAUCIER, E. PILAUD
"Le concept de test en ligne"
Onde électrique, Mai-Juin 1985, Vol. 65, pp. 41-48

[SAV 80] J. SAVIR
"Syndrome testable design of Conbinational Circuits"
IEEE Trans. on Comp. Vol C-29, n° 6 juin 1980 pp. 442-451

[SEV 84] M. SEVESTRE
"Validation d'un système de sécurité feroviaire à base de microprocesseur"
4ème Colloque Int. Fiabilité et Maintenance.
Perros Guirrec 1984 pp. 586-589

[SHE 83a] J.P SHEN ; F.J FERGUSON
"Easily testable array multipliers"
Proc. FTCS 13 Milan Juin 1983 pp. 37-40

[SHE 83b] J. SHEN, M. SHUETTE
"One line Self-Monitoring using Signature Instruction Streams"
Proc. ITC Nov 1983

[SHE 84] J.P. SHEN ; F.P. FERGUSON
"The Design of easily testable VLSI array multiplier"
IEEE Trans. on Comp. Vol C33 n°6 juin 1984 pp. 554-560

[SHM 82] M.E. SHMID & All
"Upset exposure by means of abstraction Verification"
Proc. FTCS 12 juin 1982 pp. 237-244

[SHR 82] T. SHRIDAR & All
"Analysis and Simulation of paralell signature analyzer"
ITC 1982 pp. 656-661

[SMI 80] **J.F SMITH**
"Measures of the effectiveness of fault Signature analysis"
IEEE Trans. on Comp, vol C29 juin 1980 pp. 510-514

[SMI 83] **J. SMITH, P. LAMP**
"A Theory of Totally Self Checking System design"
IEEE Trans. on Comp Sept. 1983 pp. 831-844

[SPICE] "Spice User Manual" : un simulateur électrique.

[SRI 81] **T. SRIDHAR, J.P. HAYES**
"A fonctionnal approach to testing Bit Sliced Microprocessors"
IEEE Trans. on Comp. vol C30 n°8 Aout 1981 pp. 563- 571

[SRI 82] **T. SRIDHAR, S. THATTE**
"Concurrent checking of program flow in VLSI processors"
Proc. ITC 1982, pp. 191 - 199

[SUS 81] **A.K. SUSKIND**
"Testing by verifying Walsch coefficients"
Proc. FTCS 11, juin 1981, pp. 206-208

[THA 80] **S.M. THATTE, J.A. ABRAHAM**
"test generation for microprocessor"
IEEE Trans. on Comp., Vol C-29, Juin 1980, pp. 429-441.

[TIM 83] **C. TIMOC**
"Logical Models of physical failures"
ITC 1983, pp. 546-553

[VER 86] **A. VERGIS, K.STEIGLITZ**
"Testability conditions for bilateral arrays of combinational cells"
IEEE Trans. on Comp., Vol C-35, Janvier 1986, pp. 13-23

[VEN 80] **C.S VENKATRAMAN, K.K SALUJA**
"Transition count testing of sequential machines"
Proc FTCS 10, Oct 1980, pp. 167-172

[WAD 78] **R.L WADSACK**
"Technology dependent logic faults"
Compcon's 78, 1978, pp. 124-129

[WAK 78] **J.F WAKERLY**
"Error detecting Codes, Self Checking Circuits and applications"
Elsevier, North Holland New-York 1978

[YAO 80] **S. YAO**
"An approach to concurent flow checking"
IEEE Trans Software Engineering. Vol SE-6 n°2,
Mars1980, pp.126-137

TABLE DES MATIERES

I - INTRODUCTION GENERALE 3

II - SHEMA GENERAL DE HSURF 17

II.1 - Organisation matérielle 18
II.2 - Organisation logicielle 18

III - JEU D'INSTRUCTION DU MICROPROCESSEUR HSURF 21

III.1 - Modes d'adressage 23
III.2 - Jeu d'instructions 27

III.2.1 - Instructions générales 27
III.2.2 - Instructions spécialisées 29

IV - PARTIE OPERATIVE DE HSURF 37

IV.1 - Eléments généraux de la partie opérative 40

IV.1.1 - Registres 40
IV.1.2 - Opérateurs de la partie opérative 40

IV.1.2.1 - L'unité arithmétique et logique 42
IV.1.2.2 - Le dispositif de masquage 66
IV.1.2.3 - Le multiplieur 69
IV.1.2.4 - Le dispositif de signature 76

IV.2 - Architecture générale de la partie opérative 97
IV.3 - Réalisation de la partie opérative 99

V - PARTIE CONTROLE DE HSURF 111

V.1 - Principes de la microprogrammation 113
V.2 - La génération du contenu de la R.O.M. et du controleur..... 115
V.3 - Le test de la partie contrôle 115
V.4 - Les dispositifs permettant de faciliter le test de la partie contrôle 116
V.5 - Réalisation de la partie contrôle 118

V.5.1 - Algorithme d'interprétation du jeu d'instruction de HSURF 118
V.5.2 - Implantation du contrôleur de HSURF 119

V.6 - Conclusion 125

VI - TEMPORISATION DE HSURF 127

VI.1 - Déroulement des micro-instructions 130
VI.2 - Temporisation de la partie contrôle 134
VI.3 - Temporisation de la partie opérative 136

VII - LE TEST HORS-LIGNE DE HSURF 139

VII.1 - Stratégie 142
VII.2 - Déroulement du test 142
VII.3 - PHASE 1 : Test sous contrôleur spécial 145
VII.4 - PHASE 2 : Test sous contrôleur de HSURF 152
VII.5 - L'automate de test 156
VII.6 - Conclusion 171

VIII - LE TEST EN LIGNE DE HSURF 173
VIII.1 - Test en ligne discontinu 177
VIII.2 - Test en ligne continu 179
VIII.3 - Conclusion 195

IX - CONCLUSION GENERALE 197

REFERENCES 201

ANNEXES :

A1- Codes opérations du microprocesseur HSURF 219
A2- Vecteurs de test du multiplieur de HSURF 267
A3- Règles de dessin CMP-C.MOS 1 µ 271
A4- Détails de la partie contrôle de HSURF 277

Annexe 1

Codes opération du microprocesseur HSURF

ANNEXE-1.1

Toutes les opérations à deux opérandes se font entre un des 16 registres accessibles en dehors des instructions de test (ou valeur immédiate pour COMP et STORE avec des éléments des matrice), et un opérande qui est :

- un registre
- un opérande immédiat
- un opérande mémoire
- un élément de matrice

MODES D'ADRESSAGE

La plupart des 32 opérations arithmétiques et logiques peuvent être utilisées avec 12 modes d'adressage (8 pour les matrices et 4 "classiques").

Les 21 branchements travaillent en adressage relatif uniquement.

Les instructions pour le test et COMPARMAT sont à part.

Remarques :

- Dans le cas des opérations masquées, un troisième opérande contenant le numéro du registre servant de masque (sur 4 bits) sera nécessaire. Il sera placé à la fin de l'instruction.

- Les registres dont les numéros sont donnés sur trois bits (registres opérandes), sont un des registres généraux R0 à R7.

- Les registres dont les numéros sont données sur quatre bits (registre destination, registres masque, registres numéro de ligne ou de colonne), sont parmi les seize registres accessibles à l'utilisateur en dehors des instructions de test :

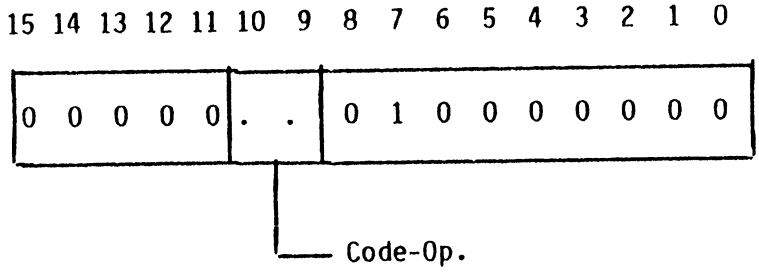
CO, RE, PP, RP1, RMO, RM1,POL,RTIS, ainsi que R0 à R7

ANNEXE-1.2

MODES D'ADRESSAGE "CLASSIQUES"

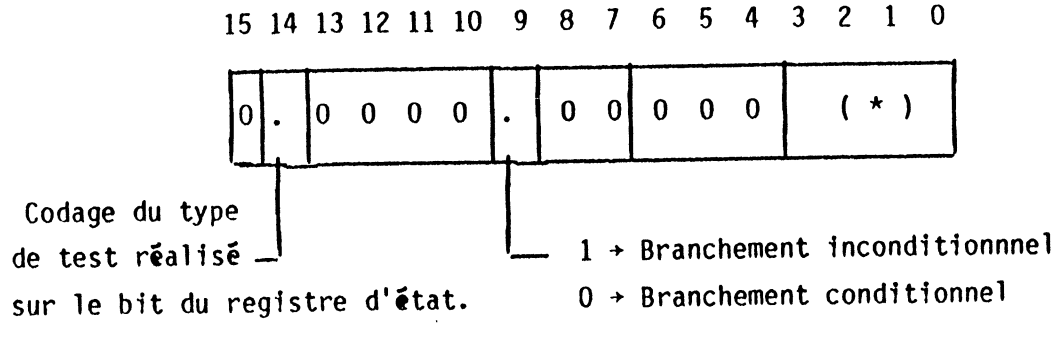
INHERENT :

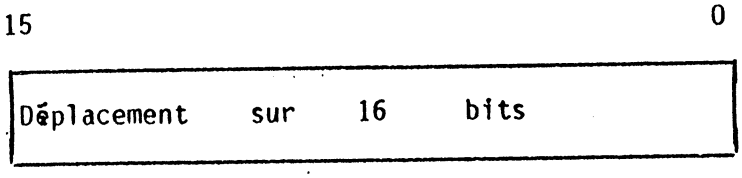
L'opérande est implicitement désignée par le code opération. Ce mode d'adressage n'est utilisé que par RSR, CLC, SEC,NOP.



RELATIF :

L'opérande est un déplacement sur 16 bits donné en complément à 2 dans le mot mémoire suivant celui du code de l'instruction. Ce mode d'adressage ne sert qu'aux instructions de branchement.



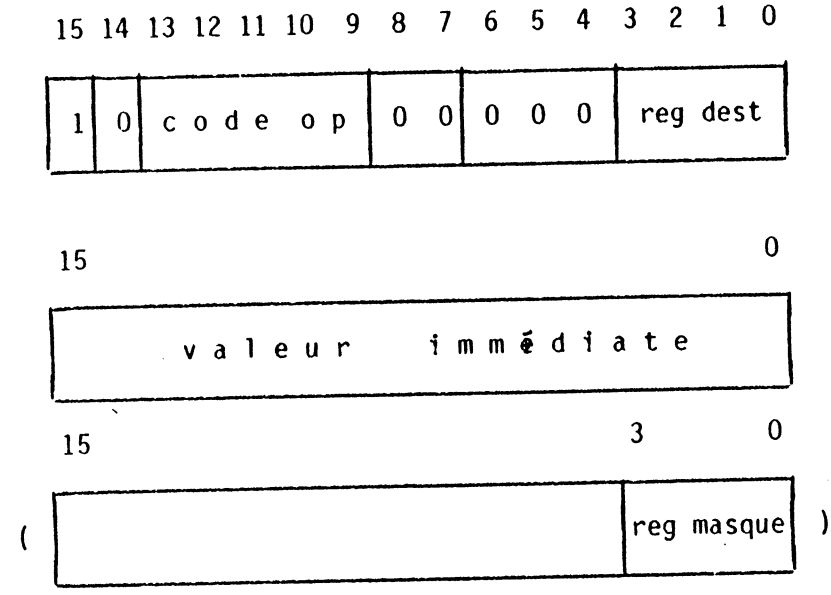


(*) : numéro du bit testé dans le registre d'état dans le cas d'un branchement conditionnel (10 possibilités).

OPERANDE IMMEDIAT

Ce mode et les suivants sont utilisés dans pratiquement toutes les instructions arithmétiques, logiques et de transfert.

L'opérande est ici une valeur située dans le deuxième mot de l'instruction.



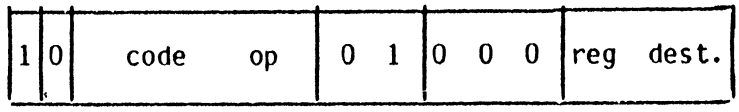
req dest est le numéro du registre destination de l'opération.

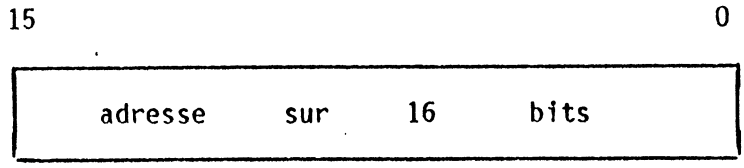
req masque est le numéro du registre utilisé comme masque.

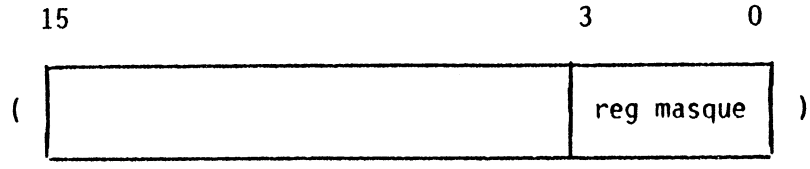
ANNEXE-1.4

DIRECT :

Le mot suivant le code d'instructon contient l'adresse de l'opérande sur 16 bits.

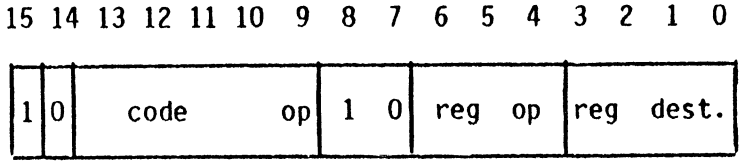


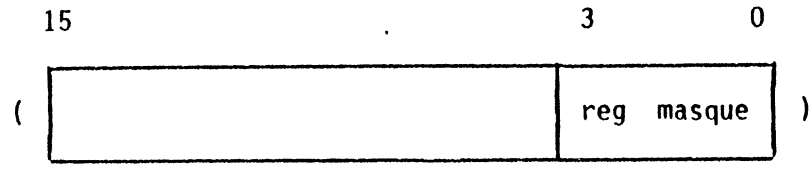




REGISTRE :

L'opérande est ici dans un registre dont le numéro est donné dans les bits 6 à 4 du mot contenant le code de l'instruction. Ce registre est un des 8 registres généraux.



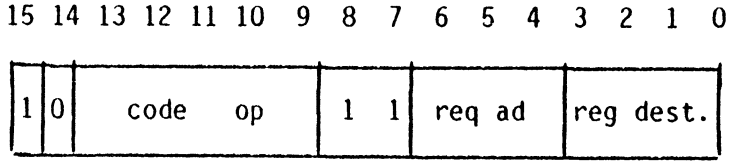


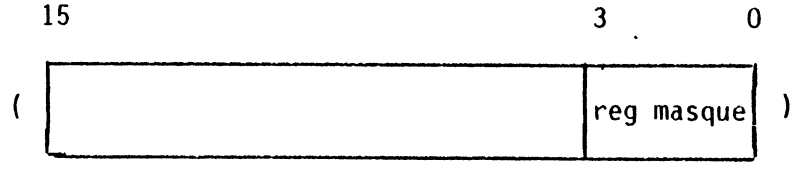
req op est le numéro du registre opérande.

ANNEXE-1.5

INDIRECT PAR REGISTRE :

L'adresse de l'opérande est dans le registre dont le numéro est donné dans le champ rep ad de l'instruction.





ANNEXE-1.6

MODES D'ADRESSAGE DES ELEMENTS DE MATRICE

Ces modes d'adressage permettent d'accéder individuellement aux éléments (mots de 16 bits) de tableaux bidimensionnels occupant au plus une page.

Il en existe 8 dont 4 ne sont utilisés que par deux instructions (STORE et COMP).

Une matrice n'est pas dans la même page que le programme et les autres variables. Pour y accéder, on dispose d'un descripteur parmi les variables du programme. L'accès à un élément de matrice se fera donc en donnant l'adresse du descripteur concerné. Les différents modes d'adressage correspondent à différentes façons de donner ces paramètres.

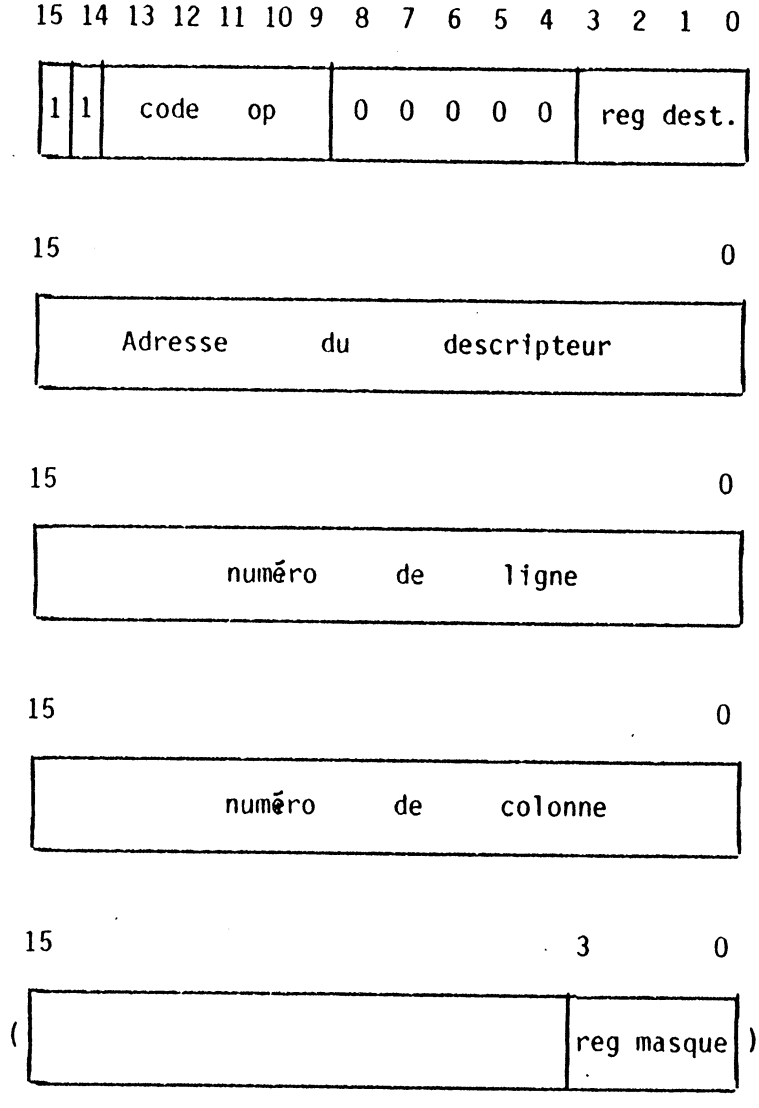
Dans le code de l'instruction, les bits 8 à 6 déterminent le mode d'adressage.

DESCRIPTEUR	NUMEROS DE LIGNE ET DE COLONNE	SYMBOLE MNEMONIQUE
Direct	Valeur immédiate	DI
Registre	Valeur immédiate	RI
Direct	Registre	DR
Registre	Registre	RR

ANNEXE-1.7

DIRECT, VALEUR IMMEDIATE

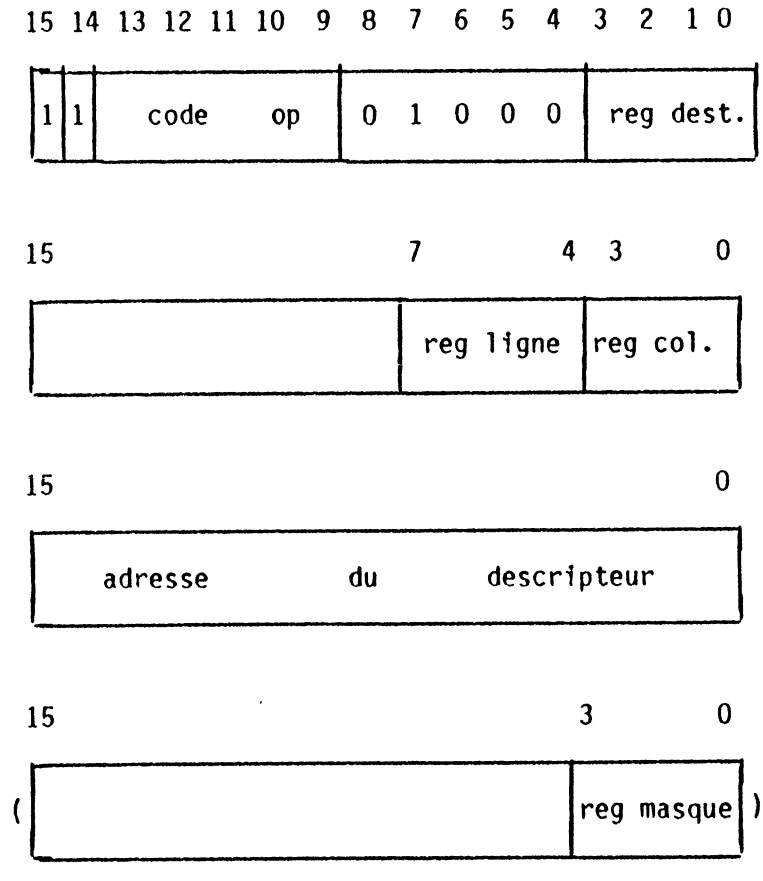
Les trois paramètres sont donnés dans les mots suivant le code de instruction.



ANNEXE-1.8

DIRECT, REGISTRE

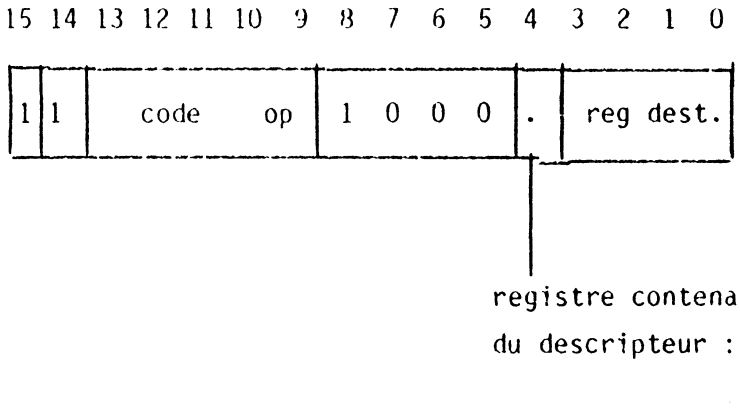
Les numéros de ligne et de colonne de l'élément adressé sont dans deux registres dont les numéros sont donnés dans le mot suivant le code de l'instruction. L'adresse du descripteur est donnée dans le mot suivant.

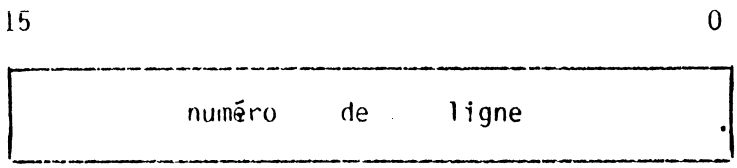


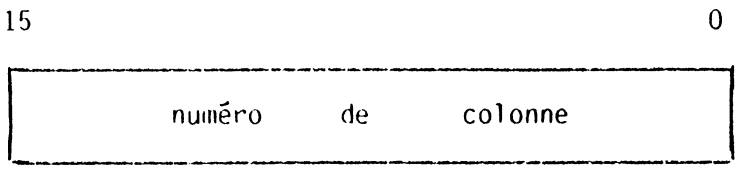
ANNEXE-1.9

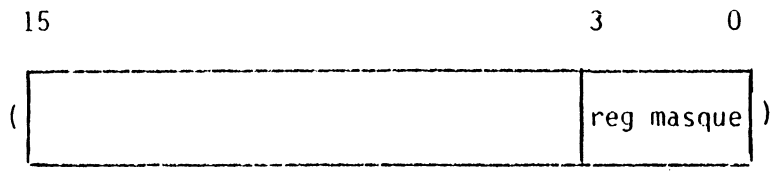
REGISTRE, VALEUR IMMEDIATE

L'adresse du descripteur est dans un des deux registres matrice RM0 ou RM1. Les numéros de ligne et de colonne de l'élément visé sont dans les deux mots suivant le code de l'instruction.





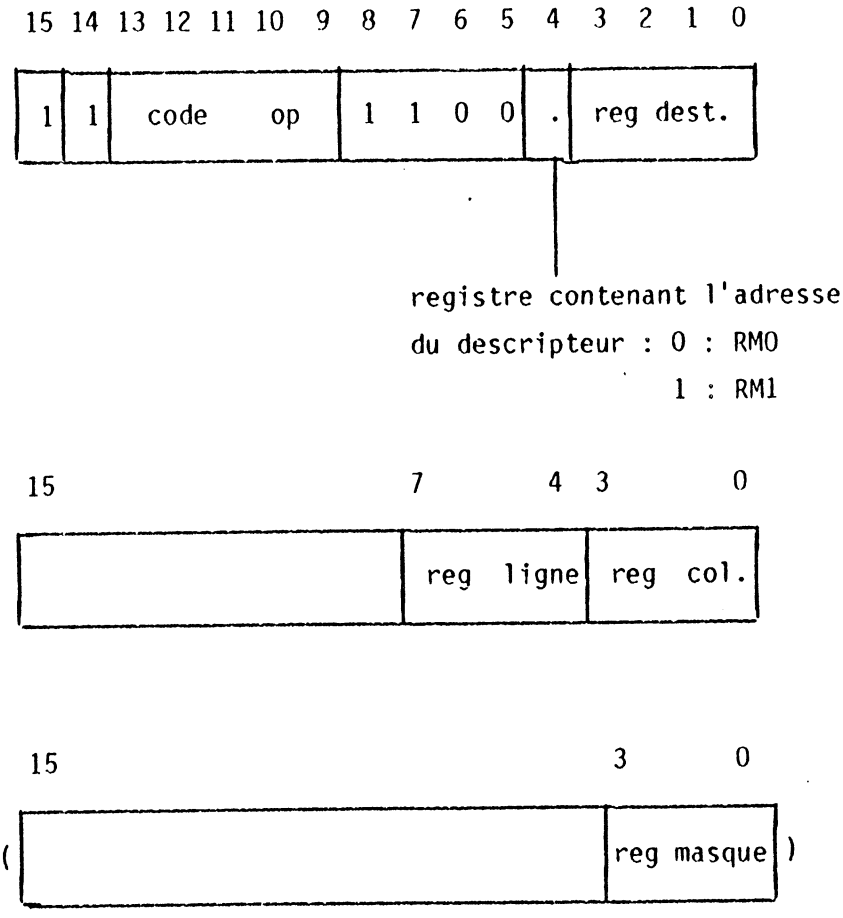




ANNEXE-1.10

REGISTRE, REGISTRE

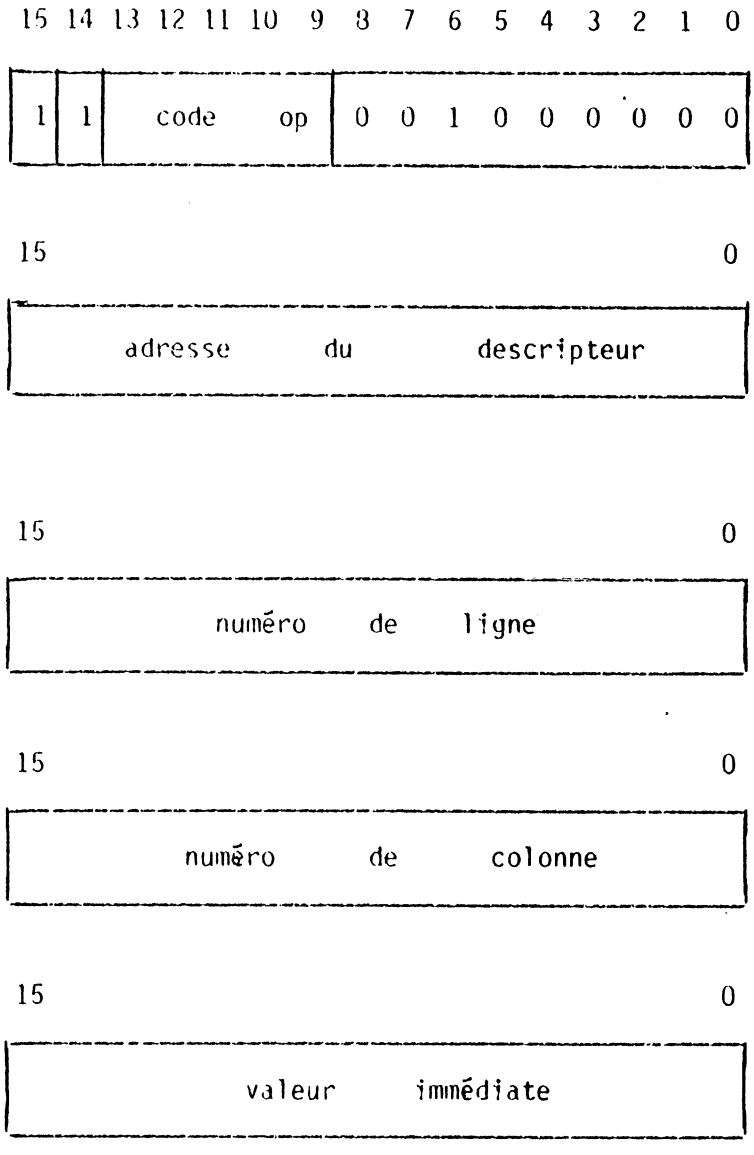
L'adresse du descripteur est dans RM0 ou RM1. Les numéros de ligne et de colonne sont dans des registres dont les numéros sont donnés dans le mot suivant le code de l'instruction.



Les quatres modes d'adressage suivants ne sont utilisés que par STORE et COMP (pour travailler sur une valeur immédiate). Ils sont similaires à ceux ci-dessus avec en plus un mot pour la valeur immédiate. Ils ne sont pas utilisables avec des opérations masquées.

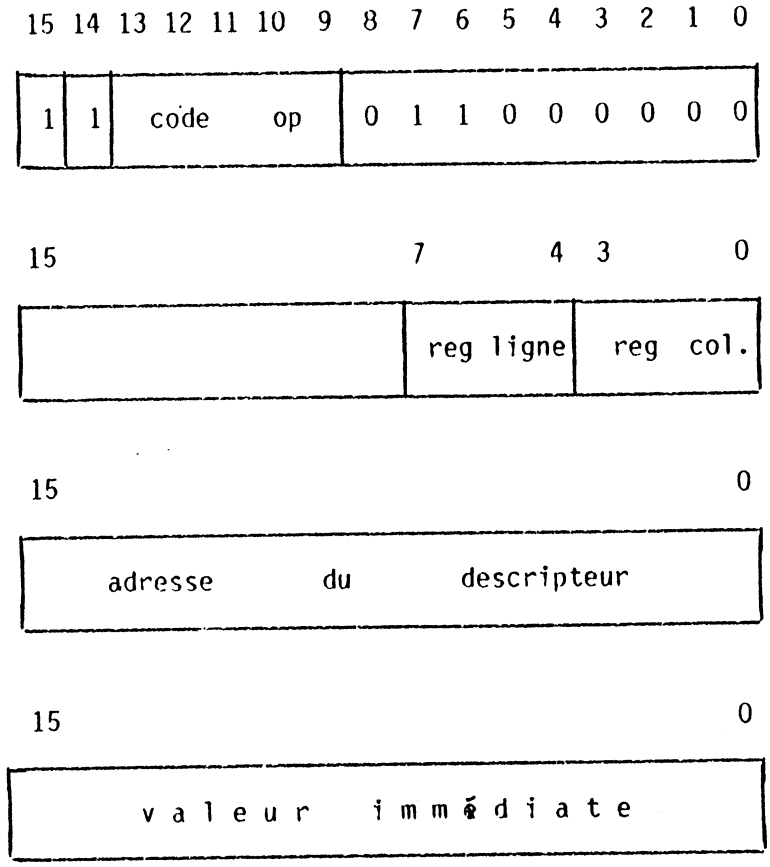
ANNEXE-1.11

DIRECT, VALEUR IMMEDIATE - VALEUR IMMEDIATE :



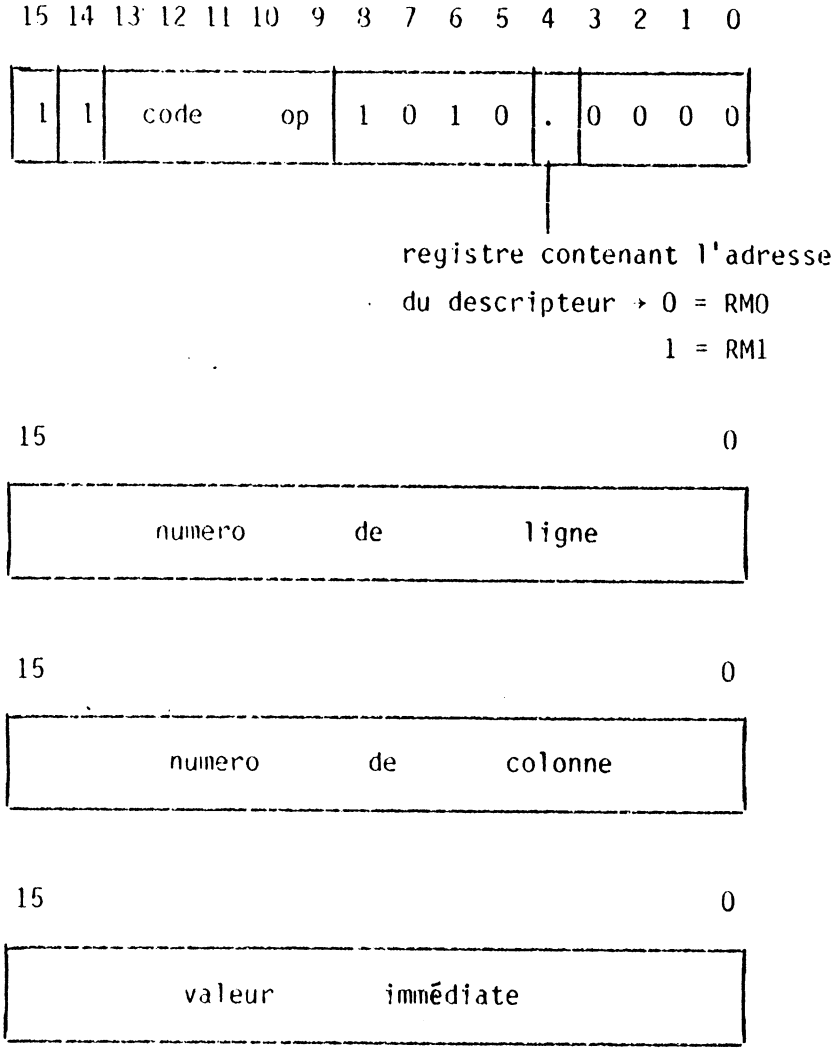
ANNEXE-1.12

DIRECT, REGISTRE - VALEUR IMMEDIATE :



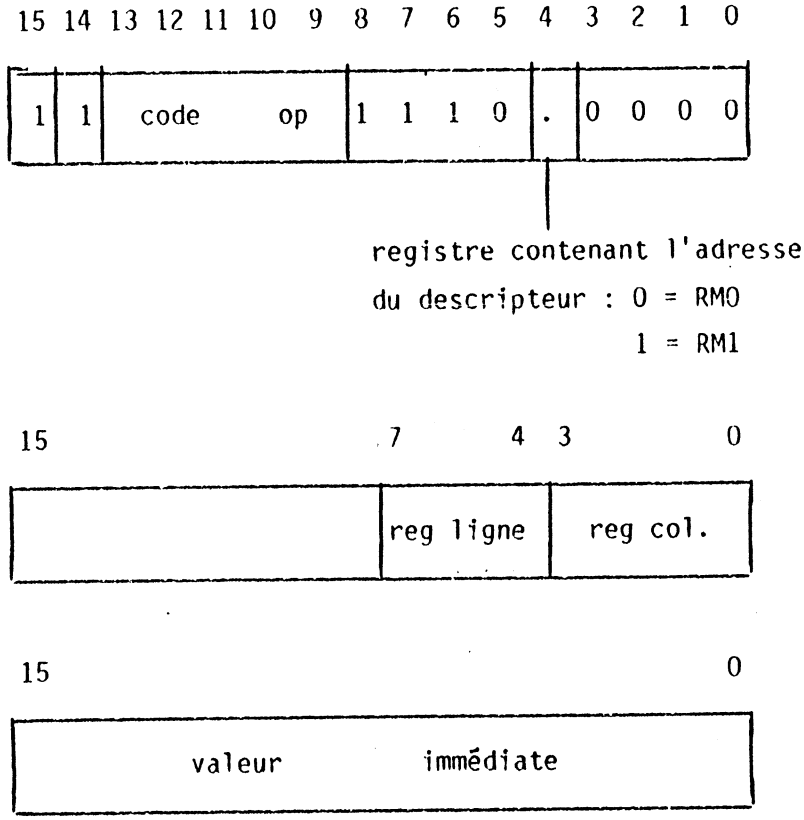
ANNEXE-1.13

REGISTRE, VALEUR IMMEDIATE - VALEUR IMMEDIATE



ANNEXE-1.14

REGISTRE, REGISTRE - VALEUR IMMEDIATE :



INSTRUCTIONS

Un distingue cinq types d'instructions :

- Arithmétiques et logiques :

  Addition, soustraction, opérations logiques de base (et, ou, nand, nor, eor), décalages, opérations de transfert load, store, exchange), opérations masquées et de comparaison.

ANNEXE-1.15

- Branchement :

Il en existe 21, 2 par bit du registre d'état (branchement si 0 et branchement si 1), plus le branchement inconditionnel (BRA).

- Instructions avec opérande inhérent :

CLC. SEC qui agissent sur le bit de la retenue dans le registre d'état.
RSR retour de sous-programme.

- Rupture de séquence et manipulation de pile :

Saut inconditionnel, saut à un sous-programme, empiler, dépiler.

- Instruction de comparaison de deux matrices :

COMPARMAT.

- Instruction pour le test :

Elles permettent l'écriture et la lecture de TOUS les éléments de mémorisation du microprocesseur ainsi que la sélection des informations à signer.

ANNEXE-1.16

NOTATION :

Les registres destination sont notés "R".
Les opérandes sont notés "Op".
L'affectation du registre destination est notée : ":="

exemple :

Une opération à 2 opérandes (reg. dest. reçoit reg. dest. oper opérande) sera notée :

R := R oper Op

Un X dans un bit du registre d'état signifie que celui-ci est modifié par l'instruction.

Un ? indique que la valeur du bit est indéterminée après exécution de l'opération (résultat sans interêt fonction de la valeur des opérandes).

ATTENTION : Quand le registre destination est RE, on ne prend pas en compte les bascules d'état venant de l'UAL.

CODAGE DU CODE OPERATION :

- L'indication : CODE-OP = $XXXX_H$ signifie que le code de l'opération est $XXXX_H$ en hexadecimal.

- L'indication : Code-Op = xxxxx signifie qu'il faut remplacer, en fonction du mode d'adressage choisi, l'indication "code op" des pages précédentes (bits 13 à 9 du registre instruction) par la valeur binaire xxxxx pour obtenir le codage de l'instruction.

ANNEXE-1.17

INSTRUCTIONS ARITHMETIQUES, LOGIQUES ET DE TRANSFERT :

* Instructions utilisant les modes d'adressage :

OPERANDE IMMEDIAT, DIRECT, REGISTRE, INDIRECT PAR REGISTRE et pour les éléments de matrice :

DIRECT, VALEUR IMMEDIATE ; DIRECT, REGISTRE ; REGISTRE, VALEUR IMMEDIATE ; REGISTRE, REGISTRE.

- Instructions ARITMETIQUES :

ADD : addition sans retenue

R := R + Op

Code-Op = 00000

indicateur d'état :

E	L	H	W	T	G	V	N	Z	C
X	X	X	X	X	X	X	X	X	X

ADDC : addition avec retenue

R := R + OP + C

Code-Op = 00010

indicateur d'état :

E	L	H	W	T	G	V	N	Z	C
X	X	X	X	X	X	X	X	X	X

ANNEXE-1.18

SUB : soustraction sans retenue

R := R - Op

Code-Op = 00001

indicateur d'état :

E	L	H	W	T	G	V	N	Z	C
X	X	X	X	X	X	X	X	X	X

SUBC : soustraction avec retenue

R := R - Op - C

Code-Op = 00011

indicateur d'état :

E	L	H	W	T	G	V	N	Z	C
X	X	X	X	X	X	X	X	X	X

ANNEXE-1.19

- Instructions LOGIQUES :

AND : et logique

R := R et Op

Code-Op = 00111

indicateur d'état :

E	L	H	W	T	G	V	N	Z	C
X	X	X	X	X	X	X	X	X	X

OR : ou logique

R := R ou Op

Code-Op = 01000

indicateur d'état : idem AND

NAND : non_et logique

R := R et Op

Code-Op = 01001

indicateur d'état : idem AND

NOR : non_ou logique

R := R ou Op

Code-Op = 01010

indicateur d'état : idem AND

ANNEXE-1.20

EOR : ou exclusif

R := R + Op

Code-Op = 00110

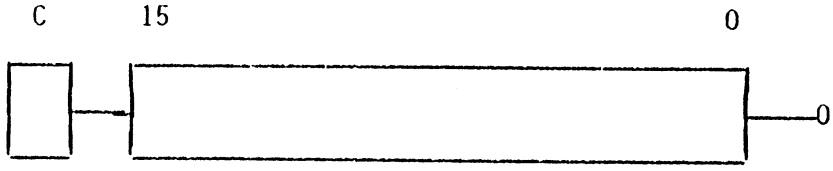
indicateur d'état : idem AND

ANNEXE-1.21

- Instructions de DECALAGE :

On dispose d'instructions de décalages arithmétiques et logiques permettant de décaler un registre de plusieurs positions.
Le nombre de positions dont on décale est contenu dans l'opérande Op.

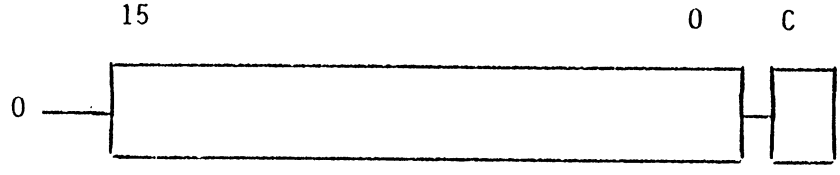
LSL : décalage logique à gauche d'un registre (R)



ode-Op = 01110

indicateur d'état : C est modifié

LSR : décalage logique à droite d'un registre

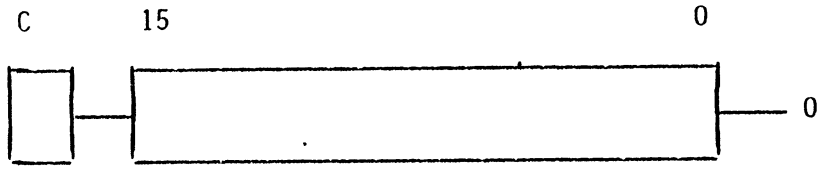


Code-Op = 01100

indicateur d'état : C est modifié

ANNEXE-1.22

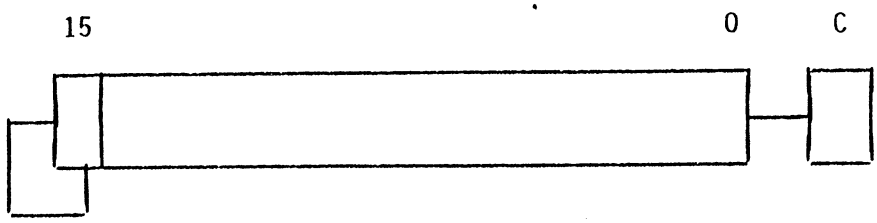
ASL : décalage arithmétique à gauche d'un registre



Code-Op = 01111

indicateur d'état : C est modifié

ASR : décalage arithmétique à droite d'un registre



Code-op = 01101

indicateur d'état : C est modifié

ANNEXE-1.23

- Instruction de COMPARAISON :

COMP : comparaison d'un registre (R) ou une valeur immédiate (pour les éléments de matrice uniquement) et un registre ou un mot mémoire (variable, élément de matrice) ou une valeur immédiate (Op).

Code-Op = 11011

indicateur d'état :

E	L	H	W	T	G	V	N	Z	C
X	X	X	X	X	X	X	X	X	

Remarque :

Cette instruction peut, en plus de ceux cités plus haut, utiliser (pour des éléments de matrice) les modes d'adressage suivants :

DIRECT, VALEUR IMMEDIATE - VALEUR IMMEDIATE
DIRECT, REGISTRE - VALEUR IMMEDIATE
REGISTRE, VALEUR IMMEDIATE - VALEUR IMMEDIATE
REGISTRE, REGISTRE - VALEUR IMMEDIATE

ANNEXE-1.24

Instructions de TRANSFERT :

LOAD : chargement d'un registre

R := Op

Code-Op = 11100

Indicateur d'état :

E	L	H	W	T	G	V	N	Z	C
							X	X	

ANNEXE-1.25

STORE : rangement d'un registre ou d'une valeur immediate (pour les éléments de matrice uniquement) en mémoire (opérande mémoie ou élément de matrice) ou dans un registre

Op := R (ou VALEUR IMMEDIATE)

Code-Op = 10011

indicateur d'état :

E	L	H	W	T	G	V	N	Z	C
								X	X

Remarque : MODES D'ADRESSAGE

* Comme COMP, cette instruction peut également uitliser les modes d'adressages suivants pour le rangement d'une valeur immédiate dans un élément de matrice :

DIRECT, VALEUR IMMEDIATE - VALEUR IMMEDIATE
DIRECT, REGISTRE - VALEUR IMMEDIATE
REGISTRE, VALEUR IMMEDIATE - VALEUR IMMEDIATE
REGISTRE, REGISTRE - VALEUR IMMEDIATE

* Le mode d'adressage OPERANDE IMMEDIAT n'est pas utilisable avec cette instruction. Son emploi est détecté comme un code invalide.

ANNEXE-1.26

- Instructions MASQUEES :

Les instructions masquées permettent de travailler directement sur des bits d'un mot ou sur une partie de mots.

Le masque est un des 16 registres accessibles en dehors du mode test par l'utilisateur. Son numéro sera contenu dans un mot mémoire suivant immédiatement celui ou ceux de l'instruction.

Le jeu d'instructions du microprocesseur comporte 8 instructions masquées :

- 5 instructions logiques

- 1 instruction de comparaison

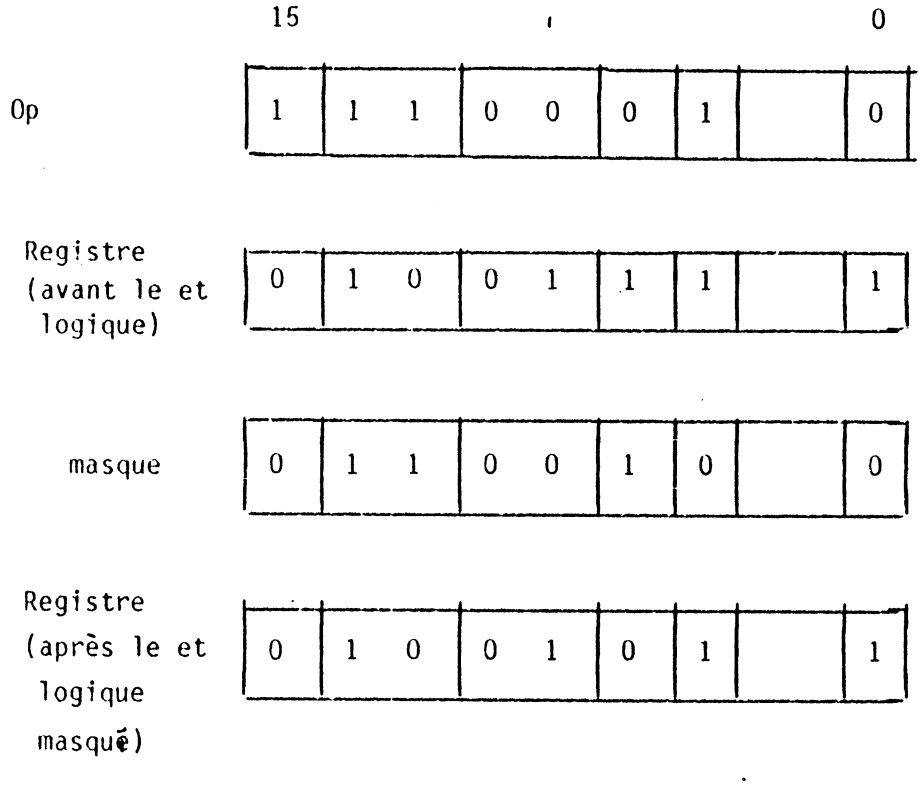
- 2 instructions de transfert

ANNEXE-1.27

Instructions masquées LOGIQUES :

L'opération effectuée est la suivante :
(R := R oper Op) masqué.

Par exemple, dans le cas où oper est le et logique :



ANNEXE-1.28

Les cinq instructions logiques masquées sont :

ANDM : et logique masqué
(R := R et Op) masqué
Code-Op = 10111

indicateur d'état :

E	L	H	W	T	G	V	N	Z	C
	X	X	X	X	X	X	X	X	X

ORM : ou logique masqué
(R := R ou Op) masqué
Code-Op = 11000

indicateur d'état : idem ANDM

NANDM : non_et logique masqué
(R := R et Op) masqué
Code-Op = 11001

indicateur d'état : idem ANDM

NORM : non_ou logique masqué
(R := R ou Op) masqué
Code-Op = 11010

indicateur d'état : idem ANDM

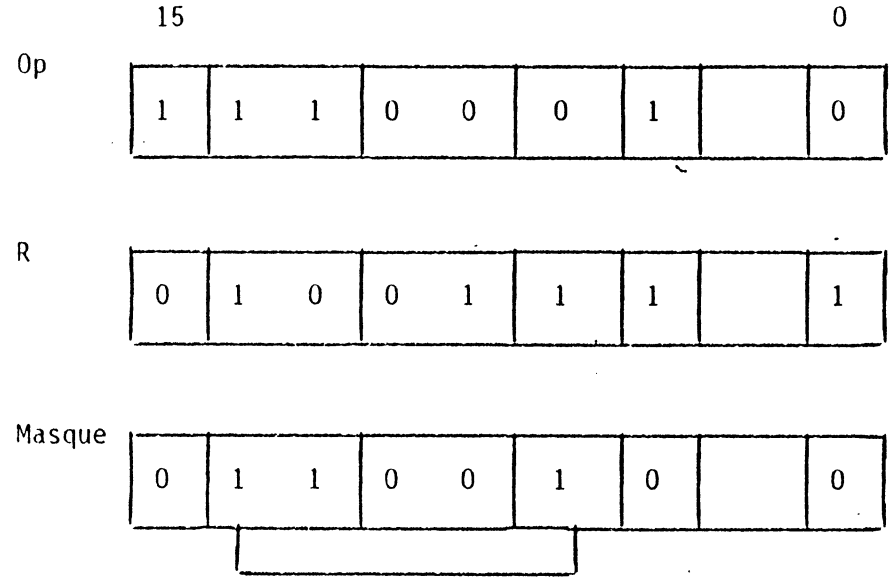
EORM : ou exclusif masqué
(R := R + Op) masqué
Code-Op = 10110

indicateur d'état : idem ANDM

Instruction de COMPARAISON

COMPM : comparaison masquée

Code-Op = 10101



Si tous les bits comparés sont égaux : Z = 1

sinon : Z = 0

ANNEXE-1.30

Instruction de TRANSFERT

LOADM : changement d'un registre
(R := Op) masqué

	15						9		0
Op	1	1	1	0	0	0	1		0
Masque	0	1	1	0	0	1	0		0
R	0	1	0	0	1	1	1		1
R (après le LOAD)	0	1	1	0	1	0	1		1

Code-Op = 10100

Indicateur d'état

E	L	H	W	T	G	V	N	Z	C
	X	X					X	X	

STOREM : rangement d'un registre
(Op := R) masqué

	15						9		0
Op	1	1	1	0	0	0	1		0
Masque	0	1	1	0	0	0	1		0
R (avant le STOREM)	0	1	0	0	1	1	0		1
Op (après le STOREM)	1	1	0	0	0	0	0		0

Code-Op = 10000

Indicateur d'état :

E	L	H	W	T	G	V	N	Z	C
	X	X					X	X	

Instructions n'utilisant que le mode d'adressage REGISTRE :

INC : incrémentation d'un registre

R := R + 1

Code-Op = 00100

indicateur d'état

E	L	H	W	T	G	V	N	Z	C
X	X	X	X	X	X	X	X	X	X

Remarque :

Le champ registre opérande du code de l'instruction n'est pas utilisé.

DEC : décrémentation d'un registre

R := R - 1

Code-Op = 00101

Indicateur d'état :

E	L	H	W	T	G	V	N	Z	C
X	X	X	X	X	X	X	X	X	X

Remarque : la même que pour INC

ANNEXE-1.33

NOT : complément logique d'un registre

R := R

Code-Op = 01011

Indicateur d'état :

E	L	H	W	T	G	V	N	Z	C
	X	X	?	?	?	?	X	X	?

Remarque : la même que pour INC

EXCH : échange entre deux registres

R := =: reg Op

Code-Op = 11101

Indicateur d'état :

E	L	H	W	T	G	V	N	Z	C
							X	X	

Il est modifié en fonction du contenu de reg Op. Ceci n'est pas vrai si R est le registre d'état RE. Si reg Op est le registre d'état RE, son contenu sera celui de R, MAIS les bits correspondant à N et Z seront modifiés lors du transfert de reg Op (ancienne valeur) vers R.

ANNEXE-1.34

INSTRUCTIONS DE BRANCHEMENT :

. L'opérande Op est dans ce cas un déplacement.
Aucune de ces instructions ne modifie le registre d'état.

BRA : branch always
branchement inconditionnel
CO := CO + Op
CODE-OP = 0200_H

BCC : branch if carry clear
branchement si la retenue est à 0
CO := CO + Op si C = 0 ; CO := CO + 1 si C = 1
CODE-OP = 0000_H

BCS : branch if cary set
branchement si la retenue est à 1
CO := CO + Op si C = 1 ; CO := CO + 1 si C = 0
CODE-OP = 4000_H

BEQ : branch if equal
branchement si égal
CO := CO + Op si Z = 1 ; CO := CO + 1 si Z = 0
CODE-OP = 4011_H

BNE : branch if not equal
branchement si différent
CO := CO + Op si Z = 0 ; CO := CO + 1 si Z = 1
CODE-OP = 0001_H

BVC : branch if overflow clear
branchement si pas de débordement
CO := CO + Op si V = 0 ; CO := CO + 1 si V = 1
CODE-OP = 0003_H

BVS : branch if overflow set
branchement si débordement
CO := CO + Op si V = 1 ; CO:= CO + 1 si V = 0

CODE-OP = 4003_H

BMI : branch if minus
branchement si moins
CO := CO + Op si N = 1 ; CO := CO + 1 si N = 0
CODE-OP = 4002_H

BPL : branch if plus
branchement si plus
CO := CO + Op si N = 0 ; CO := CO + 1 si N = 1
CODE-OP = 0002_H

BHI : branch if higher
branchement si supérieur
CO := CO + Op si C + Z = 0
CO := CO + 1 si C + Z = 1
CODE-OP = 0004_H

BLS : branch if less
branchement si inférieur
CO := CO + Op si C + Z = 1
CO := CO + 1 si C + Z = 0
CODE-OP = 4004_H

BGE : branch if greater or equal
branchement si = 0
CO := CO + Op si N + V = 0
CO := CO + 1 si N + V = 1
CODE-OP = 0005_H

BLT : branch if lower
branchement si 0
CO := CO + Op si N + V = 1
CO := CO + 1 si N + V = 0
CODE-OP = 4005_H

BGT : branch if greater

branchement si 0

CO := CO + 1 si $\underline{Z + (N + V) = 0}$

CO := CO + 1 si $Z + (N + V) = 1$

CODE-OP = 0006_H

BLE : branch if lower or equal

branchement si = 0

CO := CO + Op si $\underline{Z + (N + V) = 1}$

CO := CO + 1 si $Z + (N + V) = 0$

CODE-OP = 4006_H

BLH : branch if L high

branchement si L = 1

CO := CO + Op si $\underline{L = 1}$; CO := CO + 1 si L = 0

CODE-OP = 4008_H

BLL : branch if L low

branchement si L = 0

CO := CO + Op si $\underline{L = 0}$; CO := CO + 1 si L = 1

CODE-OP = 0008_H

BHH : branch if H high

branchement si H = 1

CO := CO + Op si $\underline{H = 1}$; CO := CO + 1 si H = 0

CODE-OP = 4007_H

BHI : branch if H low

branchement si H = 0

CO := CO + Op si $\underline{H = 0}$; CO := CO + 1 si H = 1

CODE-OP = 0007_H

BEH : branch if E high

branchement si E = 1

CO := CO + Op si $\underline{E = 1}$; CO := CO +1 s E = 0

CODE-OP = 4009_H

BEL : branch if E Low

branchement si E = 0

CO := CO + Op si $\underline{E = 0}$; CO := CO + 1 si E = 1

CODE-OP = 0009_H

ANNEXE-1.37

INSTRUCTIONS AVEC OPERANDE INHERENT :

CLC : clear carry

mise à zéro de la retenue

C := 0

CODE-OP = 0480_H

indicateur d'état :

E	L	H	W	T	G	V	N	Z	C
									X

SEC : set carry

mise à un de la retenue

C := 1

CODE-OP = 0680_H

indicateur d'état :

E	L	H	W	T	G	V	N	Z	C
									X

NOP : non opération

passage en séquence à l'instruction suivante

CODE-OP = 0080_H

ANNEXE-1.38

INSTRUCTIONS DE RUPTURE DE SEQUENCE et de MANIPULATION DE PILE

* Instructions utilisant les modes d'adressage :

OPERANDE IMMEDIAT, DIRECT, REGISTRE, INDIRECT PAR REGISTRE et pour les éléments de matrice :

DIRECT-VALEUR IMMMEDIATE, DIRECT-REGISTRE, REGISTRE-VALEUR IMMEDIATE, REGISTRE-REGISTRE.

Remarque : Dans les instructions suivantes, Op à la signification d'une adresse.

JMP : saut inconditionnel

CO := Op

Code-Op = 11111

JSR : saut à un sous-programme

```
PP := PP - 1      ) sauvegarde en pile de
                  )
MEM (PP) := CO    ) l'ancienne valeur de CO
                  )
CO := Op          ) chargement de CO avec l'adresse du sous-programme
```

Code-Op = 10001

PUSH : empiler

Cette instruction met l'opérande en sommet de pile

PP := PP - 1
MEM (PP) := Op

Code-Op = 10010

Instructions sur REGISTRE uniquement :

PULL : dépiler

Le sommet de pile est mis dans le registre dont le numéro est donné dans le champ REG DEST du code de l'instruction.

R := MEM (PP)
PP := PP + 1

Code-Op = 11110

Instructions avec adressage INHERENT :

RSR : retour de sous-programme

CO := MEM (PP)) restauration de CO
PP := PP + 1)

CODE-OP = 0280_H

ANNEXE-1.40

INSTRUCTIONS POUR LE TEST

Ces instructions sont au nombre de cinq :

LRS : chargement d'un registre interne avec un opérande direct.

R := Op

On entend par registres internes ceux qui ne sont pas accessibles avec les instructions normales LOAD et STORE (c.a.d ceux qui ne sont accessibles qu'en mode test) : T1, T2, E1, E2, RP2, S, TM, A, B, C, D, RMA, SIGN, registres de la partie contrôle (P1, P2 (pile câblée) , registres de signature, registre de l'automate séparé, ainsi que les différentes bascules (N, Z, L, RESET, ALARME de code invalide) qui seront regroupées en un registre pour ces instructions. Le bit 4 du code de l'instruction à 1 indique que l'on travaille sur un des registres de signature. Dans ce cas l'instruction n'est pas signée.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	1	0	0	num. de reg.				

15	0
adresse sur 16 bits	

SRS : Sortie d'un registre interne dans un mot mémoire (opérande direct)

Op := R

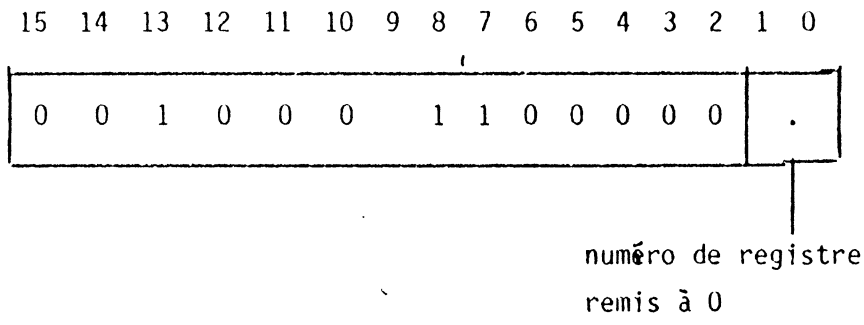
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	1	1	0	num. de req.				

15	0
adresse sur 16 bits	

ANNEXE-1.41

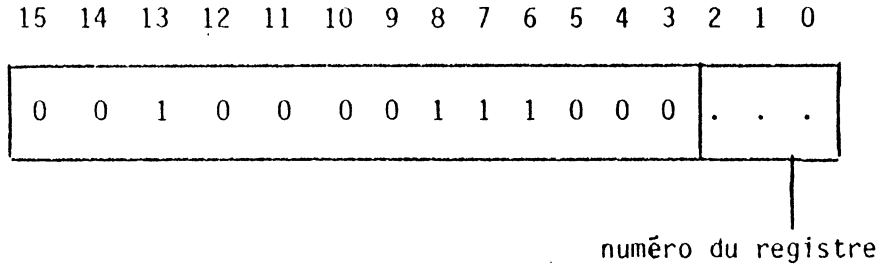
RAZ : Remise à zéro d'un registre de signature

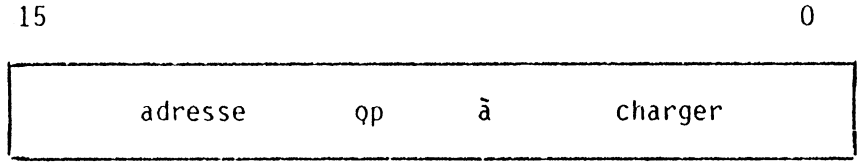
Cette instruction sert à charger "0" dans les différents registres de signature.

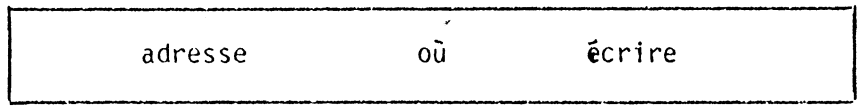


TRC : Test d'un registre du contrôleur

Son action consiste en l'écriture puis la lecture d'une valeur (donnée comme opérande direct) dans le registre testé.







TUAL : test de l'UA.L

R := R ap Op

Cette instruction donne les accès nécessaires au test exhaustif de l'UAL.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	Code Op				1	1	reg Op			reg dest			

AJS : Ajustement du contenu du registre de signature de la partie opérative.

REG-SIGN := REG-SIGN + Valeur Immédiate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0

CODE-OP = 4180_H

AJST : Equivalent à AJS mais, l'opération est suivie d'un test comparant le contenu du registre de signature avec la valeur "tout à zéro". si REG-SIGN est différent de "zéro" alors il y a émission d'un signal sur la sortie ALARME et mise du microprocesseur dans un état d'attente.

CODE-OP = $41C0_H$

ANNEXE-1.43

Affectation des numéros des registres :

Pour les instructions LRS et SRS :

Numéro	registre	Code binaire
0	T1	0 0 0 0 0
1	T2	0 0 0 0 1
2	E1	0 0 0 1 1
3	E2	0 0 0 1 1
4	RP2	0 0 1 0 0
5	S	0 0 1 0 1
6	TM	0 0 1 1 0
7	A	0 0 1 1 1
8	B	0 1 0 0 0
9	C	0 1 0 0 1
10	D	0 1 0 1 0
11	bascules N, Z, RESET, ALARME	0 1 0 1 1
12	Pile : le sommet P1 étant en poids forts, et P2 en poids faibles. Les bits 7 et 15 ne sont pas utilisés	0 1 1 0 0
14	Registre de l'automate de TRC et VDR	0 1 1 1 0
16	Registre de Signature des micro-adresses.	1 0 0 0 0
17	SIGN	1 0 0 0 1
18	Registre de signature des micro commandes décodées (SGC) pris par mor	1 0 0 1 0
à	ceaux de 16 bits au maximum, avec 18 correspondant aux bits de poids	à
23	faibles.	1 0 1 1 1

ANNEXE-1.44

Pour l'instruction RAZ

Le numéro du registre se place dans les bits 1 et 0 du code de l'instruction.

0 0 : SIGN
0 1 : Registre de signature des micro-adresses
1 0 : Registre de signature des commandes décodées

Pour l'instruction TRC

(bits 1 et 0 du code de l'instruction)

Numéro	registre
0 0 0 0	RI
1 0 0 1	micro-CO
2 0 1 0 à 7 1 1 1	Portion n°i du registre micro-instruction

ANNEXE-1.45

Pour toutes les AUTRES instructions

Numéro	Registre
0	R0
à	.
7	R7
8	C0
9	PP
10	RM0
11	RM1
12	RP1
13	RE
14	POL
15	RTIS

Remarque :

Quand un registre est codé sur trois bits, c'est un des huits registres généraux.

Annexe 2

Vecteurs de test du multiplieur de HSURF

LISTE DES VECTEURS DE TEST DU MULTIPLIEUR
CELLULAIRE DU MICROPROCESSEUR HSURF

A=	B=	C=	D=
FFFF	5555	0000	AAAA
FFFF	AAAA	FFFF	5555
AAAA	FFFF	5555	5555
5555	FFFF	AAAA	5555
AAAA	FFFF	0000	AAAA
5555	FFFF	0000	AAAA
0000	0000	0000	0000
0000	0000	FFFF	0000
0000	FFFF	FFFF	0000
FFFF	0000	0000	0000
FFFF	0000	FFFF	0000
FFFF	FFFF	FFFF	0000
5555	FFFF	5555	0000
AAAA	FFFF	AAAA	FFFF
0000	0000	FFFF	FFFF
0000	0000	FFFF	0002
0000	0000	FFFF	0004
0000	0000	FFFF	0008
0000	0000	FFFF	0010
0000	0000	FFFF	0020
0000	0000	FFFF	0040
0000	0000	FFFF	0080
0000	0000	FFFF	0100
0000	0000	FFFF	0200
0000	0000	FFFF	0400
0000	0000	FFFF	0800
0000	0000	FFFF	1000
0000	0000	FFFF	2000
0000	0000	FFFF	4000
0000	0000	FFFF	8000
0001	0000	0000	FFFF
0002	0000	0001	FFFF
0000	0000	0003	0002
0000	0000	0007	0002
0000	0000	000F	0002
0000	0000	001F	0002
0000	0000	003F	0002
0000	0000	007F	0002
0000	0000	00FF	0002
0000	0000	01FF	0002
0000	0000	03FF	0002
0000	0000	07FF	0002
0000	0000	0FFF	0002
0000	0000	1FFF	0002
0000	0000	3FFF	0002
0000	0000	0007	0004
0000	0000	000F	0004
0000	0000	001F	0004
0000	0000	003F	0004
0000	0000	007F	0004
0000	0000	00FF	0004
0000	0000	01FF	0004
0000	0000	03FF	0004
0000	0000	07FF	0004
0000	0000	0FFF	0004
0000	0000	1FFF	0004
0000	0000	3FFF	0004
0000	0000	7FFF	0004
0000	0000	000F	0008
0000	0000	001F	0008
0000	0000	003F	0008
0000	0000	007F	0008
0000	0000	00FF	0008
0000	0000	01FF	0008
0000	0000	03FF	0008
0000	0000	07FF	0008
0000	0000	0FFF	0008
0000	0000	1FFF	0008
0000	0000	3FFF	0008
0000	0000	7FFF	0008
0000	0000	001F	0010
0000	0000	003F	0010
0000	0000	007F	0010
0000	0000	00FF	0010
0000	0000	01FF	0010
0000	0000	03FF	0010
0000	0000	07FF	0010
0000	0000	0FFF	0010
0000	0000	1FFF	0010
0000	0000	3FFF	0010
0000	0000	7FFF	0010
0000	0000	003F	0020
0000	0000	007F	0020
0000	0000	00FF	0020
0000	0000	01FF	0020
0000	0000	03FF	0020
0000	0000	07FF	0020
0000	0000	0FFF	0020
0000	0000	1FFF	0020
0000	0000	3FFF	0020
0000	0000	7FFF	0020
0000	0000	007F	0040
0000	0000	00FF	0040
0000	0000	01FF	0040
0000	0000	03FF	0040
0000	0000	07FF	0040
0000	0000	0FFF	0040
0000	0000	1FFF	0040
0000	0000	3FFF	0040
0000	0000	7FFF	0040
0000	0000	00FF	0080
0000	0000	01FF	0080
0000	0000	03FF	0080
0000	0000	07FF	0080
0000	0000	0FFF	0080
0000	0000	1FFF	0080
0000	0000	3FFF	0080
0000	0000	7FFF	0080
0000	0000	01FF	0100
0000	0000	03FF	0100
0000	0000	07FF	0100
0000	0000	0FFF	0100
0000	0000	1FFF	0100
0000	0000	3FFF	0100
0000	0000	7FFF	0100

A=	B=	C=	D=
0000	0000	03FF	0200
0000	0000	07FF	0200
0000	0000	0FFF	0200
0000	0000	1FFF	0200
0000	0000	3FFF	0200
0000	0000	7FFF	0200
0000	0000	07FF	0400
0000	0000	0FFF	0400
0000	0000	1FFF	0400
0000	0000	3FFF	0400
0000	0000	7FFF	0400
0000	0000	0FFF	0800
0000	0000	1FFF	0800
0000	0000	3FFF	0800
0000	0000	7FFF	0800
0000	0000	1FFF	1000
0000	0000	3FFF	1000
0000	0000	7FFF	1000
0000	0000	3FFF	2000
0000	0000	7FFF	2000
0003	0000	0003	0002
0007	0000	0007	0002
000F	0000	000F	0002
001F	0000	001F	0002
003F	0000	003F	0002
007F	0000	007F	0002
00FF	0000	00FF	0002
01FF	0000	01FF	0002
03FF	0000	03FF	0002
07FF	0000	07FF	0002
0FFF	0000	0FFF	0002
1FFF	0000	1FFF	0002
3FFF	0000	3FFF	0002
7FFF	0000	7FFF	0002
0003	0000	0007	0004
0007	0000	000F	0004
000F	0000	001F	0004
001F	0000	003F	0004
003F	0000	007F	0004
007F	0000	00FF	0004
00FF	0000	01FF	0004
01FF	0000	03FF	0004
03FF	0000	07FF	0004
07FF	0000	0FFF	0004
0FFF	0000	1FFF	0004
1FFF	0000	3FFF	0004
3FFF	0000	7FFF	0004
0003	0000	000F	0008
0007	0000	001F	0008
000F	0000	003F	0008
001F	0000	007F	0008
003F	0000	00FF	0008
007F	0000	01FF	0008
00FF	0000	03FF	0008
01FF	0000	07FF	0008
03FF	0000	0FFF	0008
07FF	0000	1FFF	0008
0FFF	0000	3FFF	0008
1FFF	0000	7FFF	0008
3FFF	0000	7FFF	0008
0003	0000	001F	0010

A=	B=	C=	D=
0007	0000	003F	0010
000F	0000	007F	0010
001F	0000	00FF	0010
003F	0000	01FF	0010
007F	0000	03FF	0010
00FF	0000	07FF	0010
01FF	0000	0FFF	0010
03FF	0000	1FFF	0010
07FF	0000	3FFF	0010
0FFF	0000	7FFF	0010
0003	0000	003F	0020
0007	0000	007F	0020
000F	0000	00FF	0020
001F	0000	01FF	0020
003F	0000	03FF	0020
007F	0000	07FF	0020
00FF	0000	0FFF	0020
01FF	0000	1FFF	0020
03FF	0000	3FFF	0020
07FF	0000	7FFF	0020
0003	0000	007F	0040
0007	0000	00FF	0040
000F	0000	01FF	0040
001F	0000	03FF	0040
003F	0000	07FF	0040
007F	0000	0FFF	0040
00FF	0000	1FFF	0040
01FF	0000	3FFF	0040
03FF	0000	7FFF	0040
0003	0000	00FF	0080
0007	0000	01FF	0080
000F	0000	03FF	0080
001F	0000	07FF	0080
003F	0000	0FFF	0080
007F	0000	1FFF	0080
00FF	0000	3FFF	0080
01FF	0000	7FFF	0080
0003	0000	01FF	0100
0007	0000	03FF	0100
000F	0000	07FF	0100
001F	0000	0FFF	0100
003F	0000	1FFF	0100
007F	0000	3FFF	0100
00FF	0000	7FFF	0100
0003	0000	03FF	0200
0007	0000	07FF	0200
000F	0000	0FFF	0200
001F	0000	1FFF	0200
003F	0000	3FFF	0200
007F	0000	7FFF	0200
0003	0000	07FF	0400
0007	0000	0FFF	0400
000F	0000	1FFF	0400
001F	0000	3FFF	0400
003F	0000	7FFF	0400
0003	0000	0FFF	0800
0007	0000	1FFF	0800
000F	0000	3FFF	0800
001F	0000	7FFF	0800
0003	0000	1FFF	1000

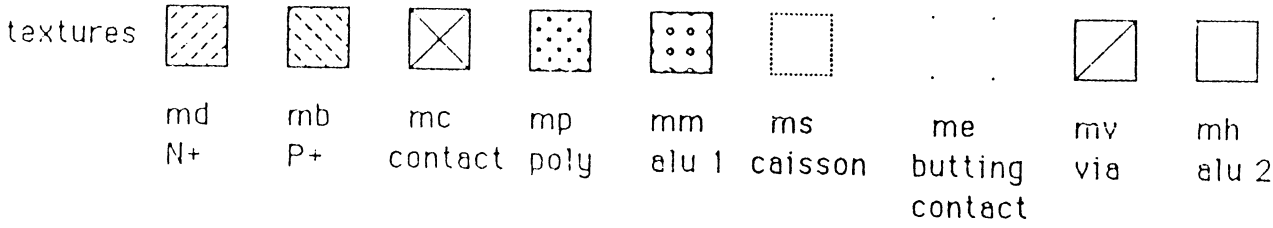
ANNEXE-2.2

```
A= 0007  B= 0000  C= 3FFF  D= 1000
A= 000F  B= 0000  C= 7FFF  D= 1000
A= 0003  B= 0000  C= 3FFF  D= 2000
A= 0007  B= 0000  C= 7FFF  D= 2000
A= 0000  B= FFFF  C= 0000  D= 0000
A= FFFF  B= FFFF  C= 0000  D= FFFF
A= 0000  B= 0000  C= 0000  D= FFFF
A= 0000  B= 0000  C= 0001  D= FFFF
```

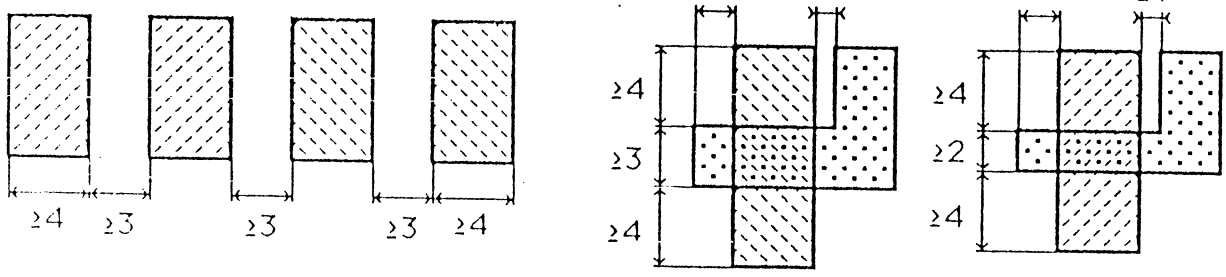
Annexe 3

Règles de dessin
CMP C.MOS 1 µ

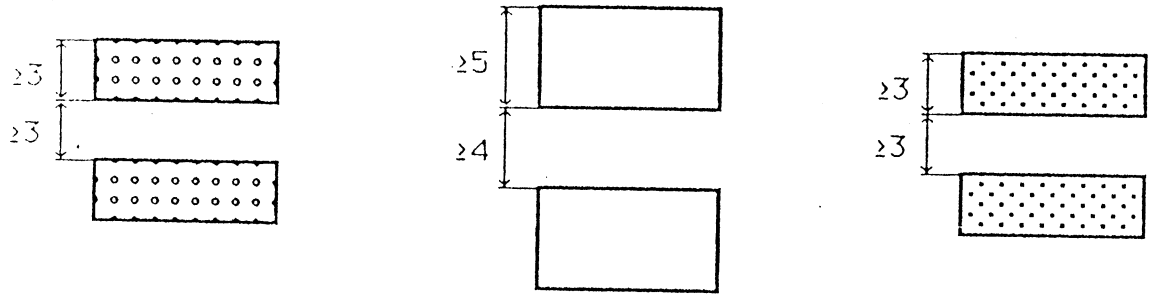
CMP
REGLES CMOS FINES



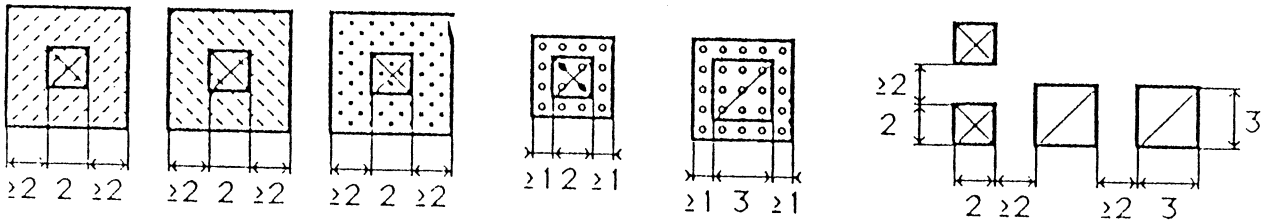
oxydes minces



interconnexions



contacts

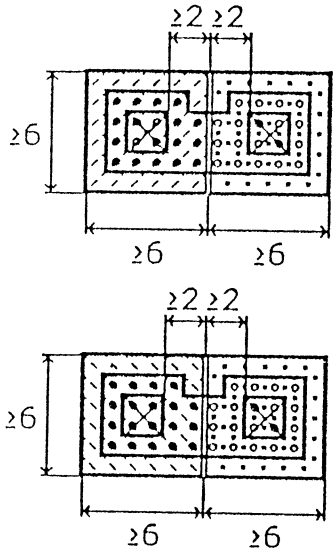


ANNEXE-3.2

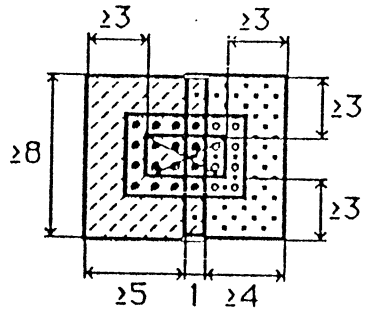
Règles CMOS fines
Janvier 1985

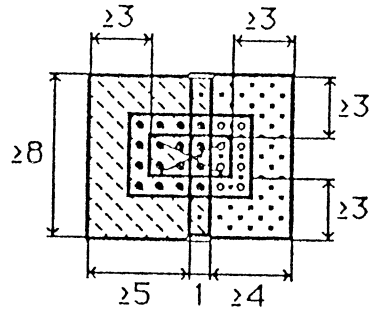
textures

md mb mc mp mm ms me mv mh

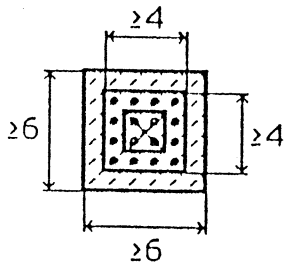


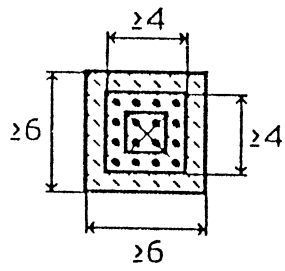
ponts métalliques

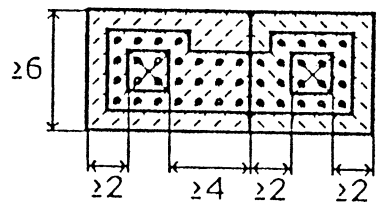




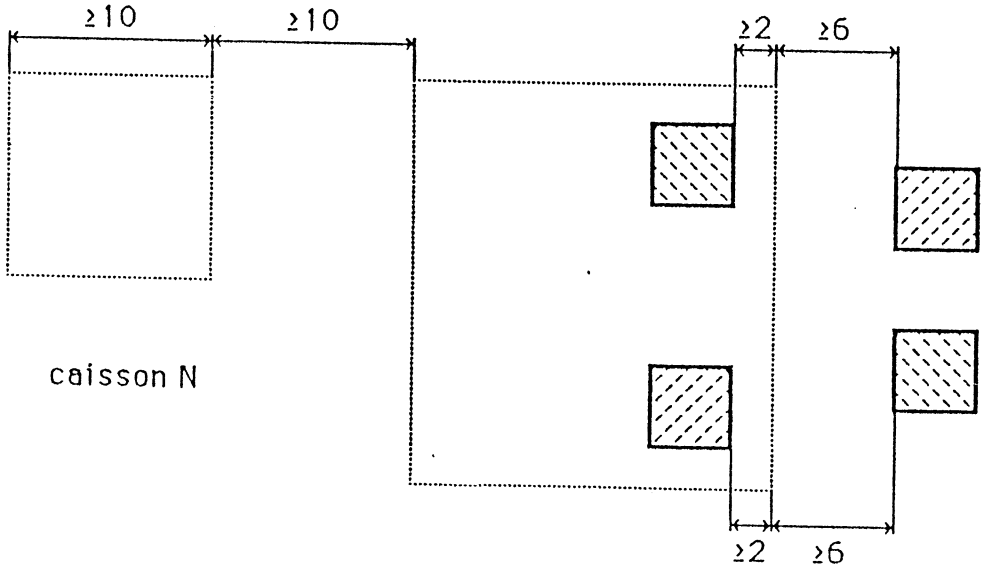
polarisation du substrat







près du contact drain/source
d'un transistor

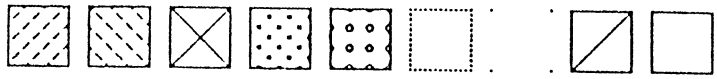


caisson N

ANNEXE-3.3

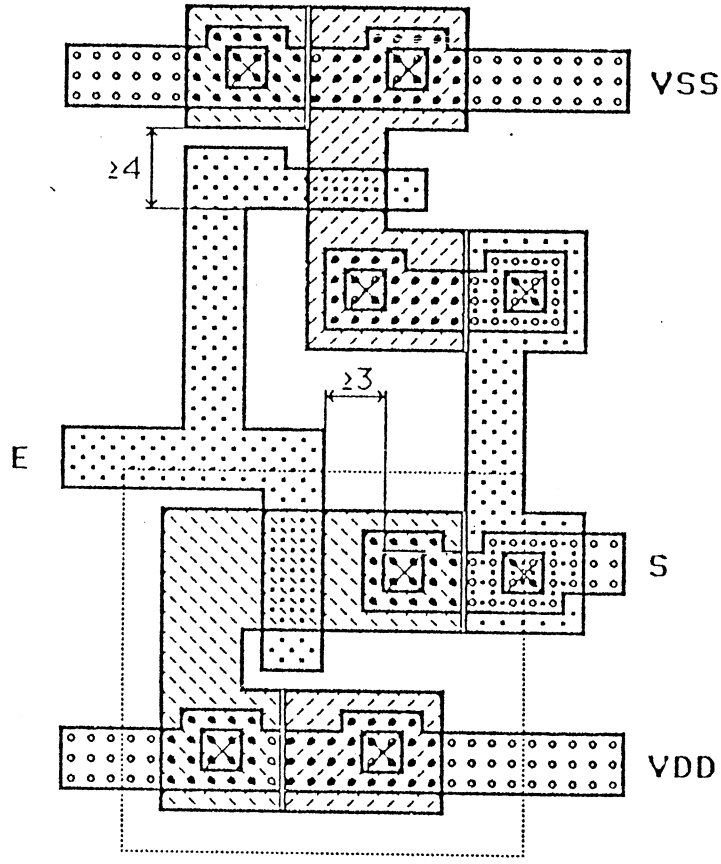
Règles CMOS fines
Janvier 1985

textures



md mb mc mp mm ms me mv mh

exemple : l'inverseur

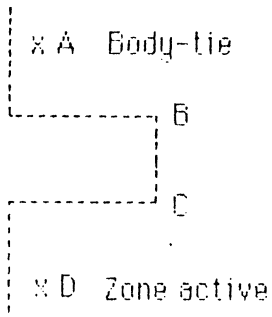


CMP
REGLES CMOS FINES

JAN.85

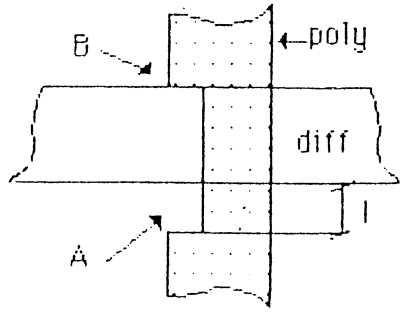
NOTES

- Les contacts enterrés ne sont pas autorisés
- Le caisson dessiné est un caisson N.
- La distance entre deux "métal-1" est normalement 3 lambdas. Toutefois elle peut être réduite à 2 lambdas, sur une longueur maximum de 8 lambdas et seulement aux alentours des contacts ou des vias.
- La distance entre un contact sur diffusion et une grille de transistor doit être de 3 lambdas.
- Un contact poly-diffusion sera réalisé normalement avec deux trous de contact. Toutefois il peut être réalisé avec un seul trou de contact, à condition que les débordements de la diffusion et du poly autour du contact soient de 3 lambdas, et non plus de 2 lambdas.
- Emplacement des body-tie : Il est recommandé que les body-tie soient situés de telle façon qu'il n'y ait pas de "zone active" (diffusion et grilles de transisors) qui soit distante de plus de 75 lambdas d'un body-tie (NB : distance à compter en suivant le chemin résistif; exemple : cas d'un rétrécissement du caisson :



La distance à considérer est ABCD et non AD).

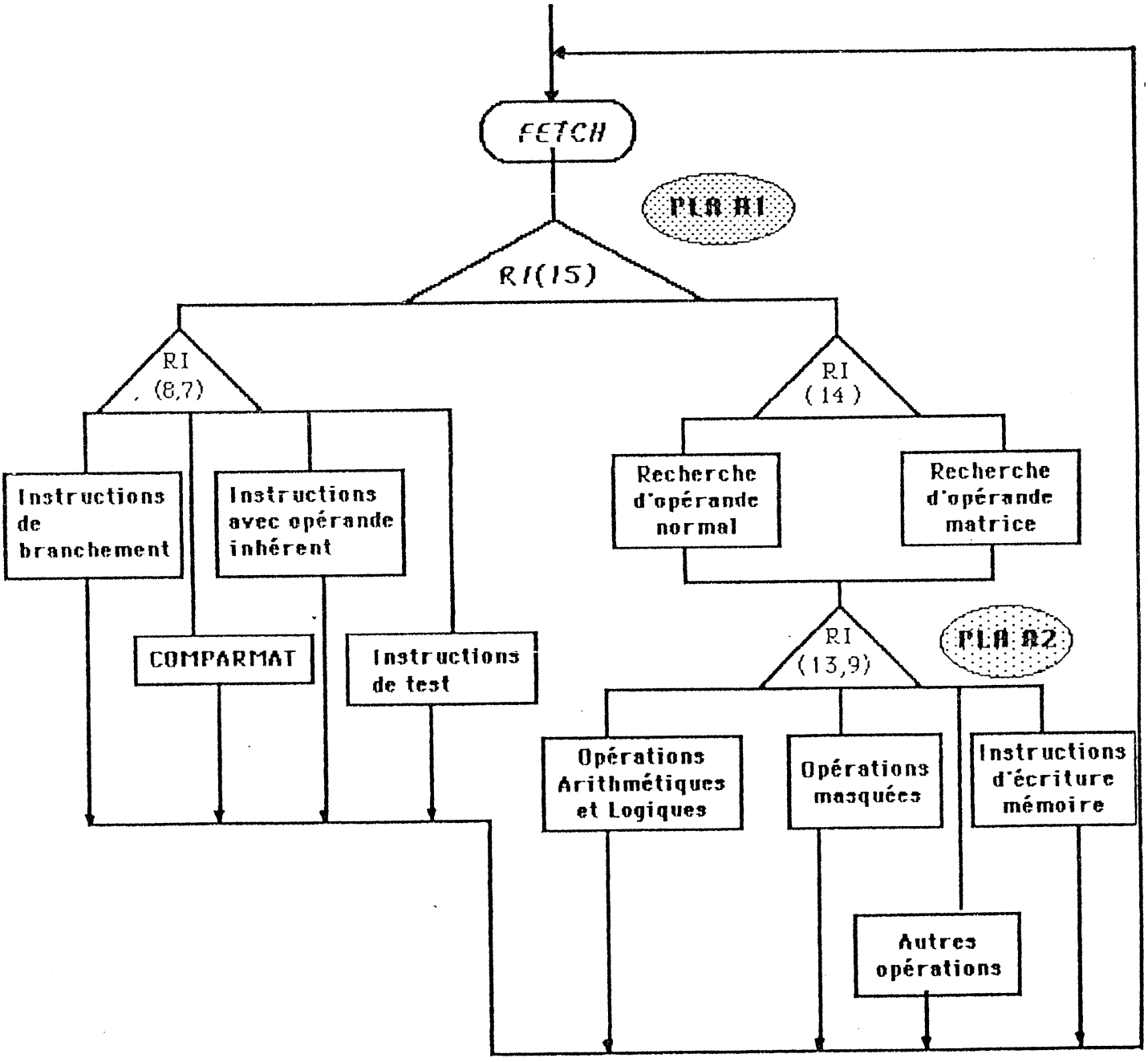
- La distance comprise entre une diffusion P (respectivement N) d'un body-tie, et le bord opposé d'une grille de transistor N (respectivement P) doit être de 4 lambdas (cas d'une grille de longueur minimale et d'un body-tie situé à proximité immédiate de la grille parallèlement à la largeur w de la grille -voir inverseur page 3).
- L'augmentation de la largeur du poly au sortir d'un transistor doit être faite de la manière A et non de la manière B.



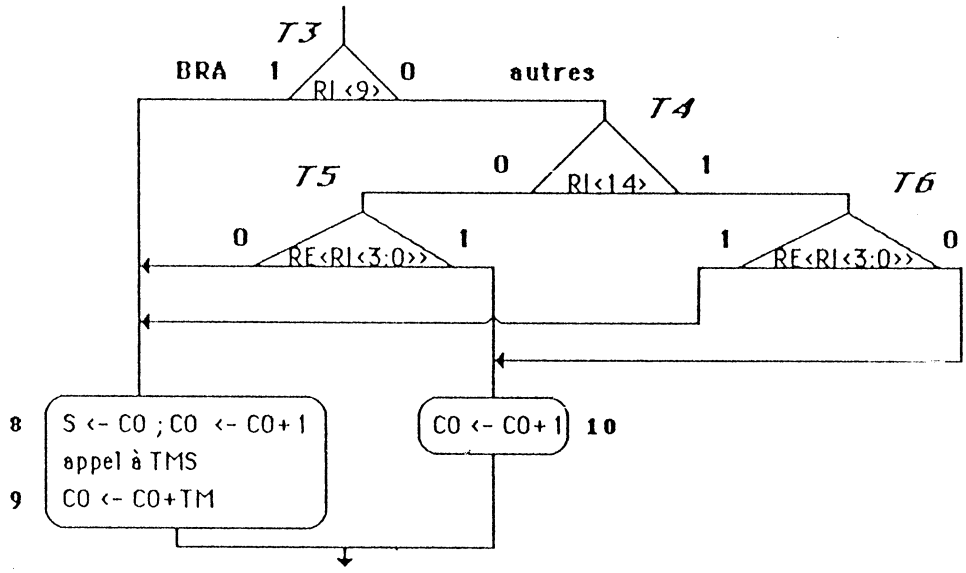
Annexe 4

Détails de la partie contrôle de HSURF

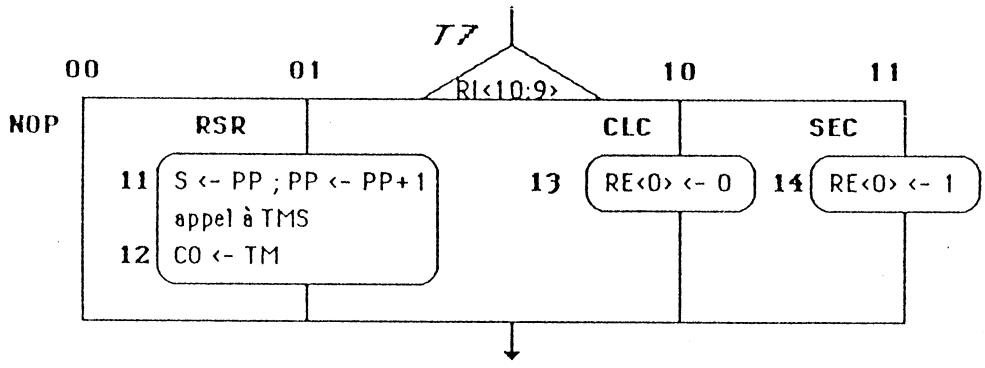
Graphe de contrôle du microprocesseur HSURF


FETCH
PLA R1
RI(15)
RI
(8,7)
RI
(14)
Instructions de branchement
Instructions avec opérande inhérent
Recherche d'opérande normal
Recherche d'opérande matrice
COMPARMAT
Instructions de test
RI
(13,9)
PLA R2
Opérations Arithmétiques et Logiques
Opérations masquées
Instructions d'écriture mémoire
Autres opérations

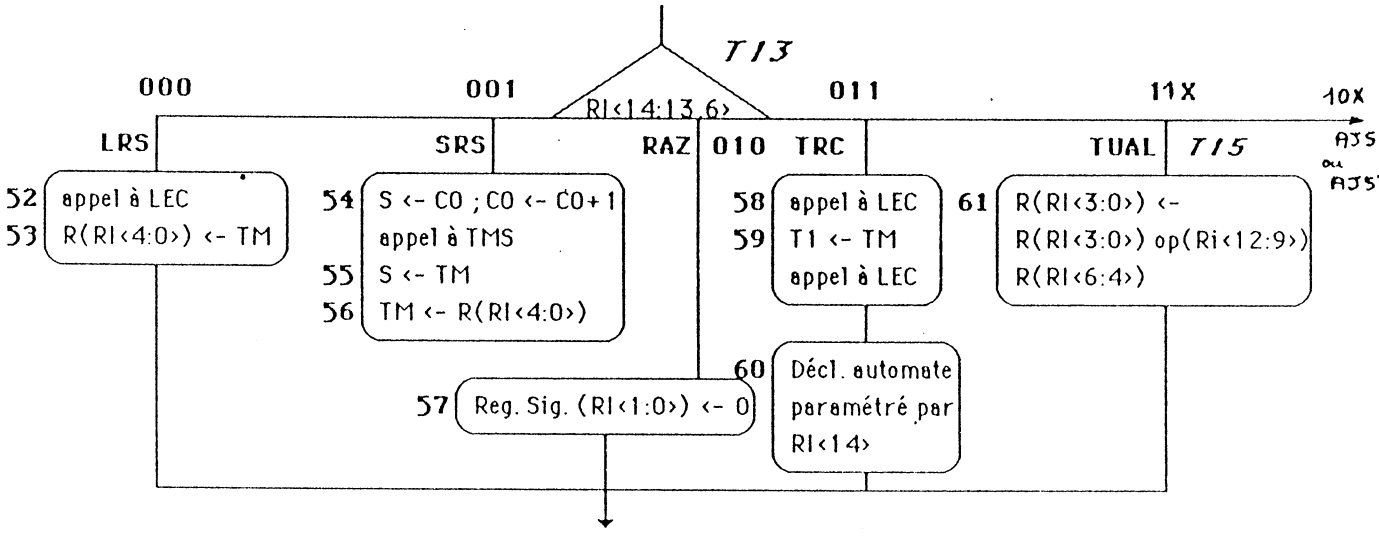
INSTRUCTIONS DE BRANCHEMENT



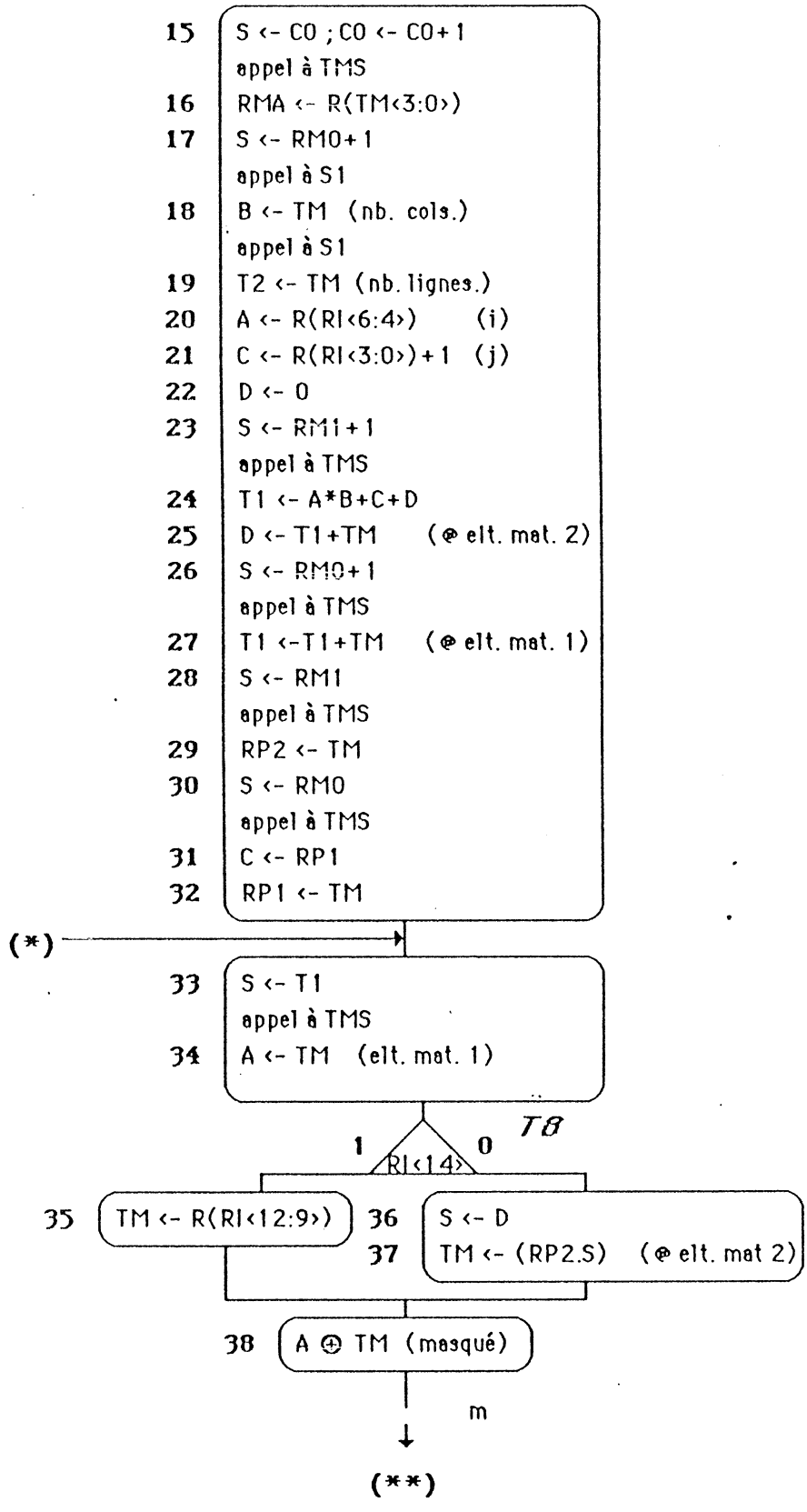
INSTRUCTIONS AVEC OPERANDE INHERENT

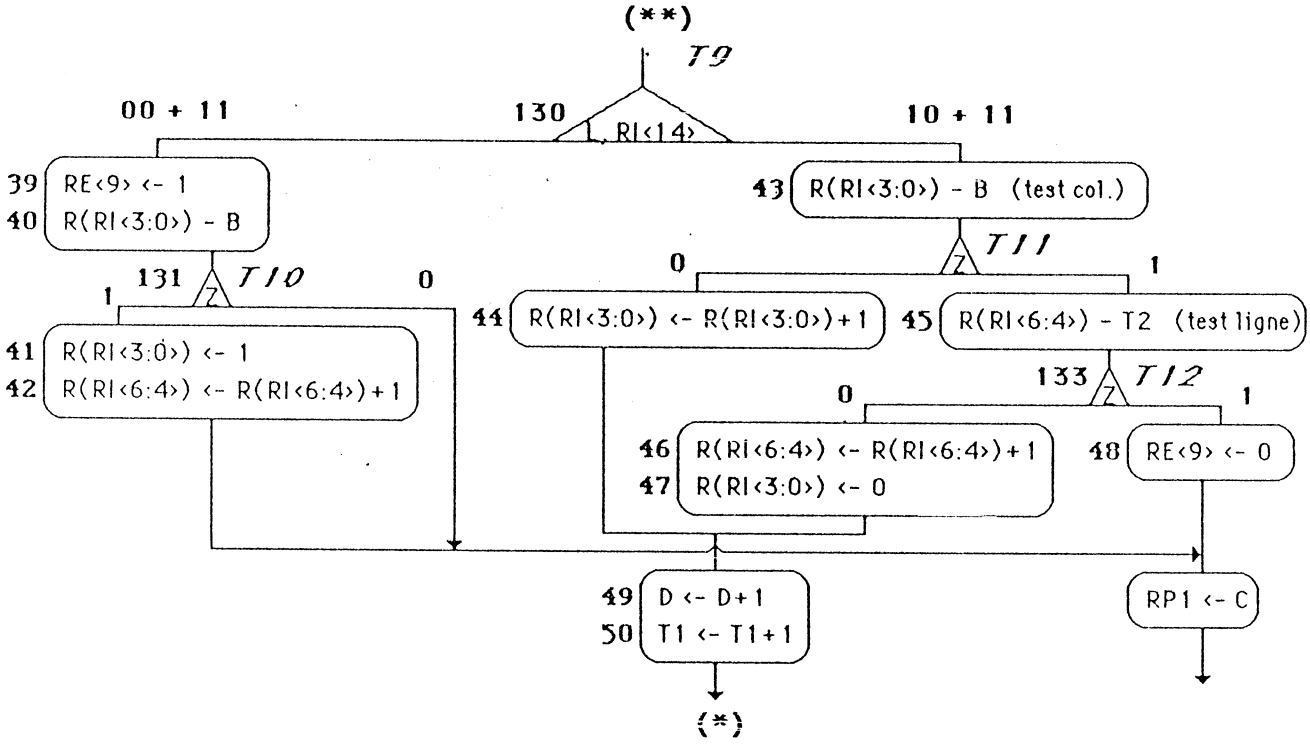


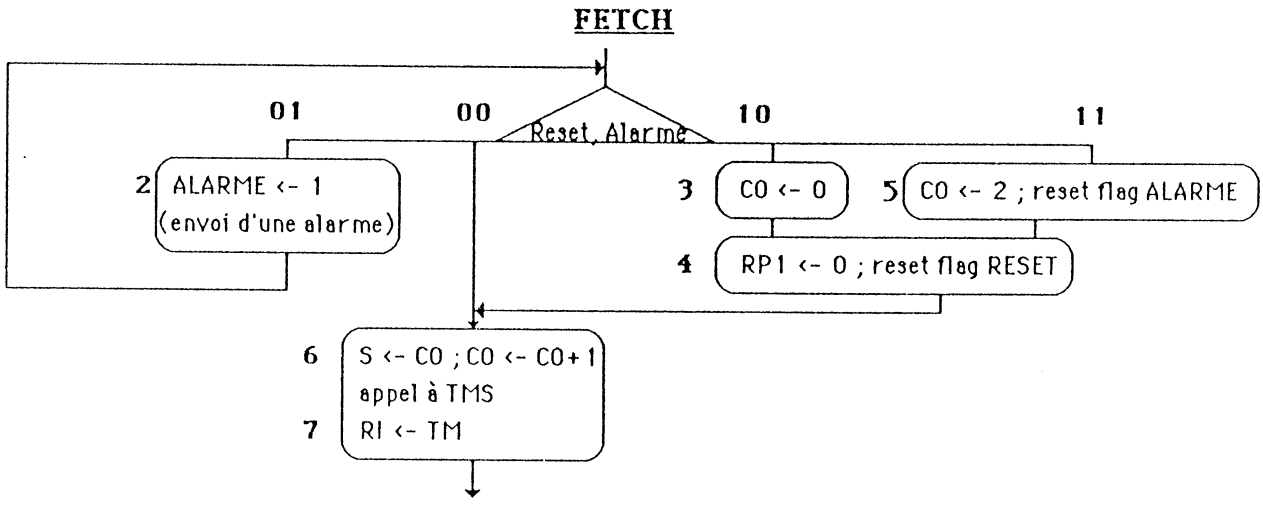
INSTRUCTIONS DE TEST



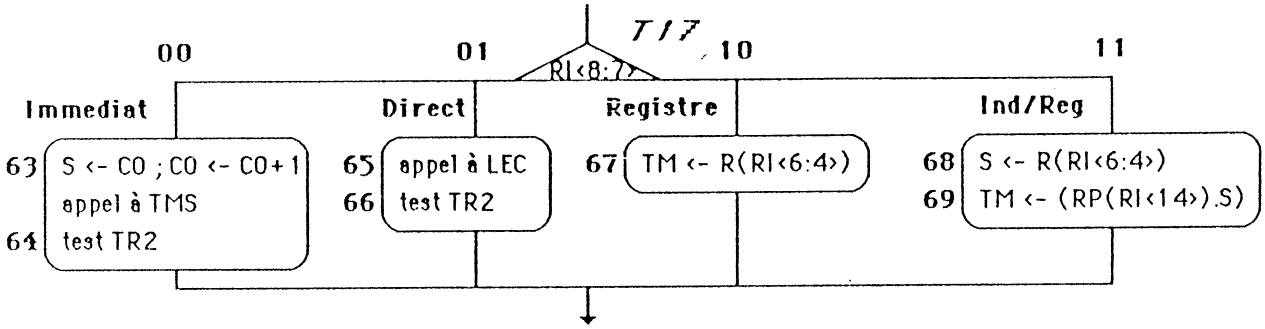
COMPARMAT


15 S <- C0 ; C0 <- C0+1
appel à TMS
16 RMA <- R(TM<3:0>)
17 S <- RM0+1
appel à S1
18 B <- TM (nb. cols.)
appel à S1
19 T2 <- TM (nb. lignes.)
20 A <- R(RI<6:4>) (i)
21 C <- R(RI<3:0>)+1 (j)
22 D <- 0
23 S <- RM1+1
appel à TMS
24 T1 <- A*B+C+D
25 D <- T1+TM (@ elt. mat. 2)
26 S <- RM0+1
appel à TMS
27 T1 <-T1+TM (@ elt. mat. 1)
28 S <- RM1
appel à TMS
29 RP2 <- TM
30 S <- RM0
appel à TMS
31 C <- RP1
32 RP1 <- TM
(*)
33 S <- T1
appel à TMS
34 A <- TM (elt. mat. 1)
TB
1
RI<14>
0
35 TM <- R(RI<12:9>)
36 S <- D
37 TM <- (RP2.S) (@ elt. mat 2)
38 A ⊕ TM (masqué)
m
(**)

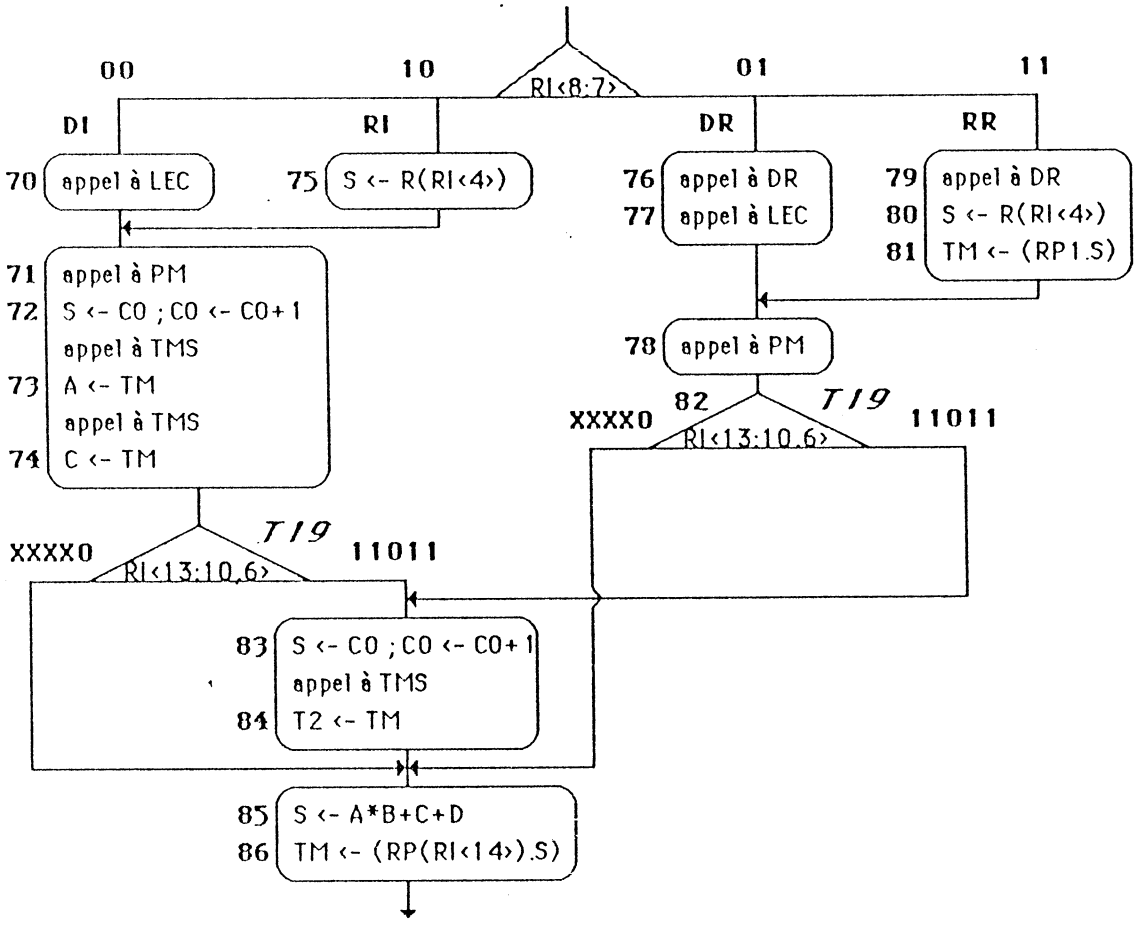

(**)
T9
130
RI<14>
00 + 11
10 + 11
39 RE<9> <- 1
40 R(RI<3:0>) - B
131
T10
Z
1
0
41 R(RI<3:0>) <- 1
42 R(RI<6:4>) <- R(RI<6:4>)+1
43 R(RI<3:0>) - B (test col.)
T11
0
1
44 R(RI<3:0>) <- R(RI<3:0>)+1
45 R(RI<6:4>) - T2 (test ligne)
133
T12
46 R(RI<6:4>) <- R(RI<6:4>)+1
47 R(RI<3:0>) <- 0
48 RE<9> <- 0
49 D <- D+1
50 T1 <- T1+1
RP1 <- C
(*)


FETCH
01
00
Reset, Alarme
10
11
2 ALARME <- 1
(envoi d'une alarme)
3 C0 <- 0
5 C0 <- 2 ; reset flag ALARME
4 RP1 <- 0 ; reset flag RESET
6 S <- C0 ; C0 <- C0+1
appel à TMS
7 RI <- TM

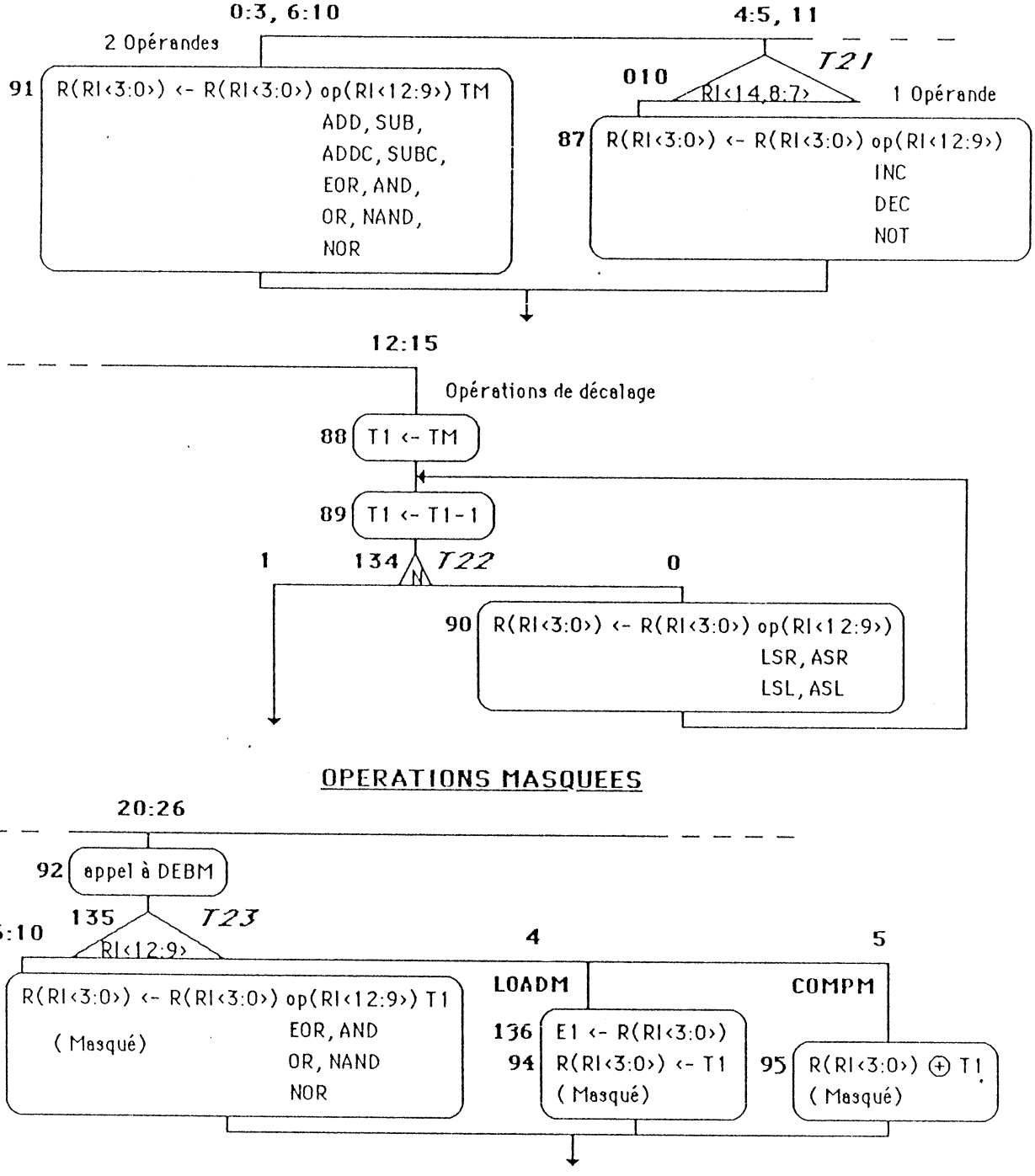
RECHERCHE D'UN OPERANDE



RECHERCHE D'UN ELEMENT DE MATRICE

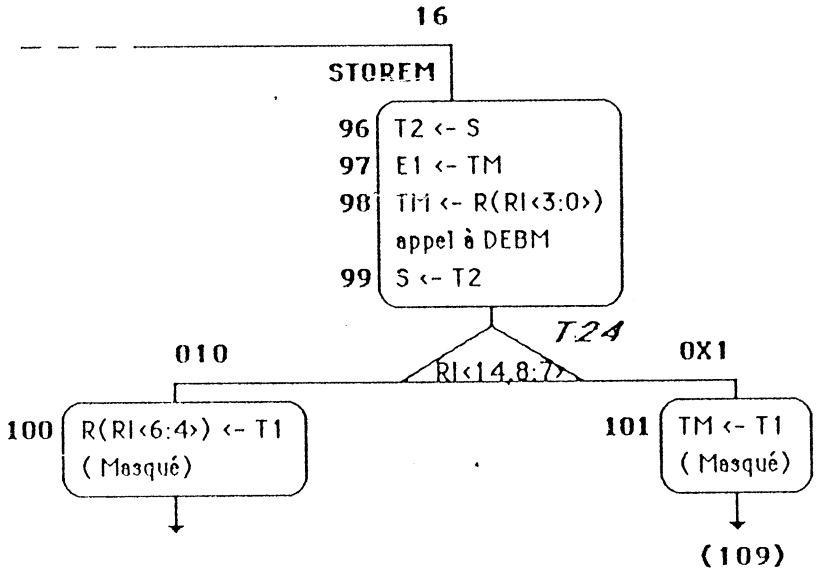


OPERATIONS ARITHMETIQUES ET LOGIQUES

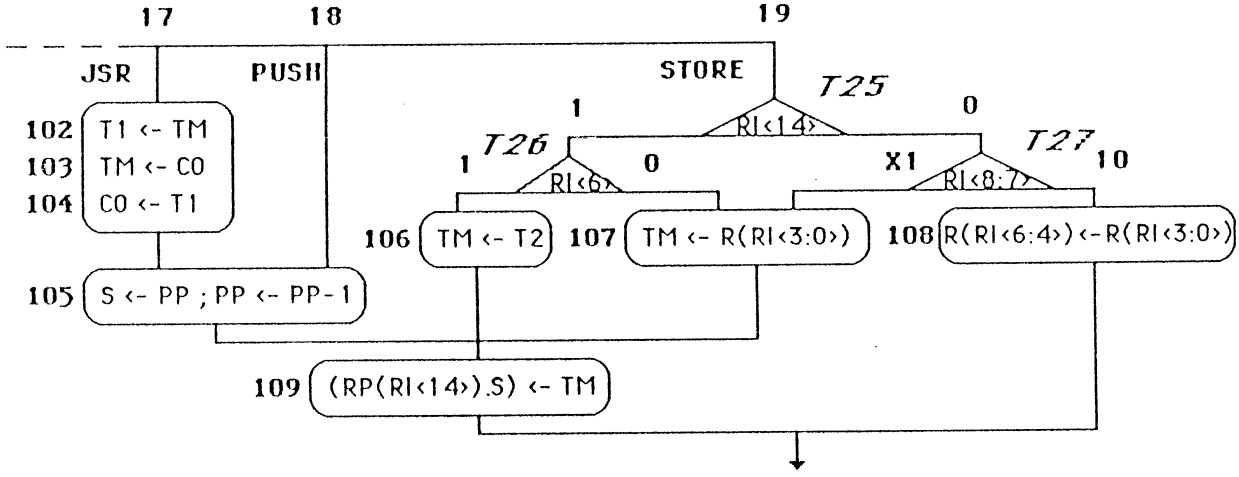


OPERATIONS MASQUEES

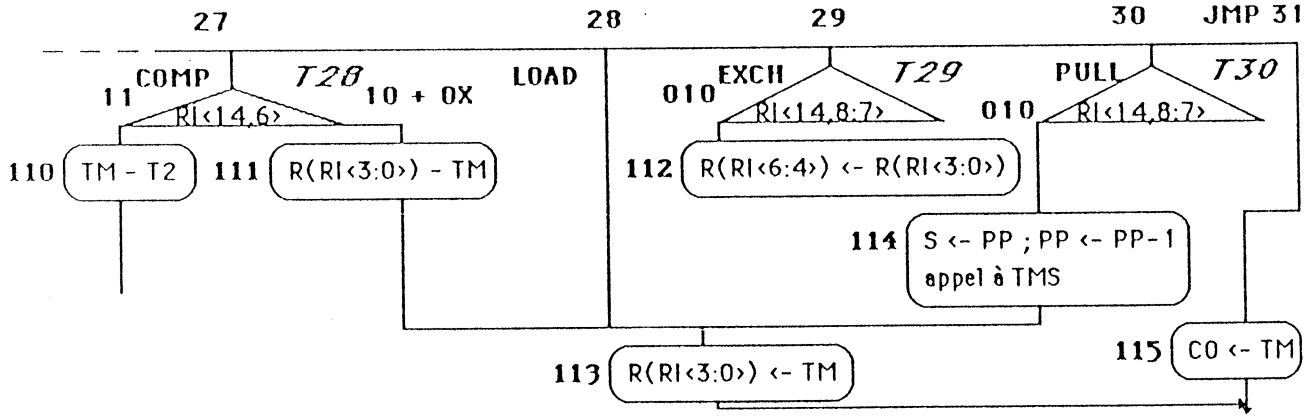
ANNEXE-4.8



INSTRUCTIONS D'ECRITURE MEMOIRE



AUTRES OPERATIONS



Sous - Microprogrammes

TMS :	lecture d'un mot mémoire avec RP1	TM <- (RP1.S)	116
MS :	lecture du mot programme suivant	S <- CO ; CO <- CO+1 saut à TMS	117
LEC :	lecture d'un opérande direct	S <- CO ; CO <- CO+1 appel à TMS S <- TM saut à TMS	118 119
S1 :	lecture du mot mémoire suivant	S <- S+1 saut à TMS	120
DEBM:	initialisation d'une opération masquée	T1 <- TM appel à MS RMA <- R(TM<3:0>)	121 122
DR :	chargement des numéros de ligne et de colonne d'un élément de matrice	S <- CO ; CO <- CO+1 appel à TMS A <- R(TM<7:4>) C <- R(TM<3:0>)	123 124 125
PM :	recherche et chargement des paramètres pour une opération d'adressage d'un élément de matrice.	RP2 <- TM appel à S1 D <- TM appel à S1 B <- TM	126 127 128

Contenu des différents PLAs du contrôleur de HSURF

ANNEXE-4.11

P L A A 1

Tests réalisés : TR1 = T1, T2, T3, T4, T5, T6, T7, T13, T14, T15, T16, T17, T18

	ENTREES RI											SORTIES	
Branch.	15	14	13	12	11	10	9	8	7	6	paramétr. bit RE	n° de µ.inst	Adresse
BRA	0						1	0	0				
Autre branch.	0	1					0	0	0		1	8	8
	0	0					0	0	0		0		
	0	0					0	0	0		1	10	10
	0	1					0	0	0		0		
NOP	0					0	0	0	1			1	2
RSR	0					0	1	0	1			11	11
CLC	0					1	0	0	1			13	13
SEC	0					1	1	0	1			14	14
COMPARMAT	0							1	0			15	16
LRS	0	0	0							0		52	54
SRS	0	0	0							1		54	57
RAZ	0	0	1							0		57	60
TRC	0	0	1							1		58	61
TUAL	0	1	0	0	0	0	0					62	65
	0	1	1	0	0	0	0					61	64
	0	1	0	0	0	0	1					62	65
	0	1	1	0	0	0	1					61	64
	0	1	0	0	0	1	0					62	65
	0	1	1	0	0	1	0					61	64
	0	1	0	0	0	1	1					62	65
	0	1	1	0	0	1	1					61	64
	0	1	0	0	1	0	0					62	65
	0	1	1	0	1	0	0					61	64
	0	1	0	0	1	0	1					62	65
	0	1	1	0	1	0	1					61	64
	0	1	0	0	1	1	0					62	65
	0	1	1	0	1	1	0					61	64
	0	1	0	0	1	1	1					62	65
	0	1	1	0	1	1	1					61	64
	0	1	0	1	0	0	0					62	65
	0	1	1	1	0	0	0					61	64
	0	1	0	1	0	0	1					62	65
	0	1	1	1	0	0	1					61	64
	0	1	0	1	0	1	0					62	65
	0	1	1	1	0	1	0					61	64
	0	1	0	1	0	1	1					62	65
	0	1	1	1	0	1	1					61	64
Immediate	1	0						0	0			63	66
Direct	1	0						0	1			65	68
Register	1	0						1	0			67	70
Ind/Reg	1	0						1	1			68	71
DI	1	1						0	0			70	73
RI	1	1						0	1			75	78
DR	1	1						1	0			76	80
PP	1	1						1	1			79	84

ANNEXE-4.12

P L A A 2

Tests réalisés : TR2 = T20, T21, T25, T26, T27, T28, T29 T30

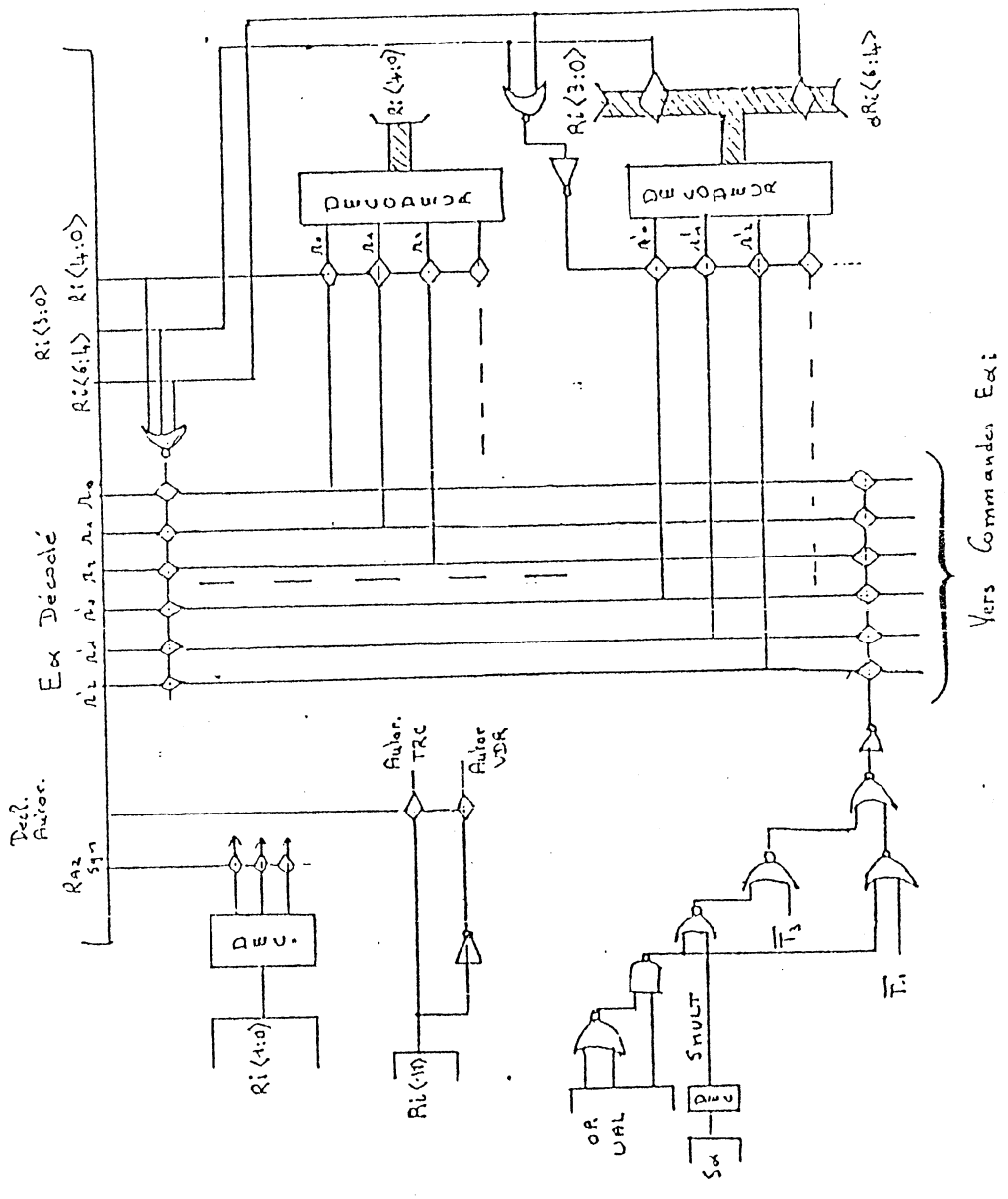
	ENTREES ↓ RI									SORTIES ↓	
Branch	14	13	12	11	10	9	8	7	6	m-instr.	Adresse
INC	0	0	0	1	0	0	1	0		87	90
DEC	0	0	0	1	0	1	1	0			
NOT	0	0	1	0	1	1	1	0			
LSR		0	1	1	0	0				88	91
ASR		0	1	1	0	1					
LSL		0	1	1	1	0					
ASL		0	1	1	1	1					
ADD		0	0	0	0	0				91	95
SUB		0	0	0	0	1					
ADDC		0	0	0	1	0					
SUBC		0	0	0	1	1					
EOR		0	0	1	1	0					
AND		0	0	1	1	1					
OR		0	1	0	0	0					
NAND		0	1	0	0	1					
NOR		0	1	0	1	0					
EORM		1	0	1	1	0				92	96
ANDM		1	0	1	1	1					
ORM		1	1	0	0	0					
NANDM		1	1	0	0	1					
NORM		1	1	0	1	0					
LOADM		1	1	0	1	1					
COMPM		1	0	1	0	1					
STOREM		1	0	0	0	0				96	101
JSR		1	0	0	0	1				102	107
PUSH		1	0	0	1	0				105	110
STORE	1	1	0	0	1	1			1	106	111
	1	1	0	0	1	1			0	107	112
	0	1	0	0	1	1		1			
	0	1	0	0	1	1	1	0		108	113
COMP	1	1	1	0	1	1			1	110	115
	1	1	1	0	1	1			0	111	116
	0	1	1	0	1	1					
LOAD		1	1	1	0	0				113	119
EXCH	0	1	1	1	0	1	1	0		112	117
PULL	0	1	1	1	1	0	1	0		114	118
JMP		1	1	1	1	1				115	120

BRANCH PLA : Tests réalisés : T8, T9, T10, T11, T12, T19, T23, T22, T24, T0

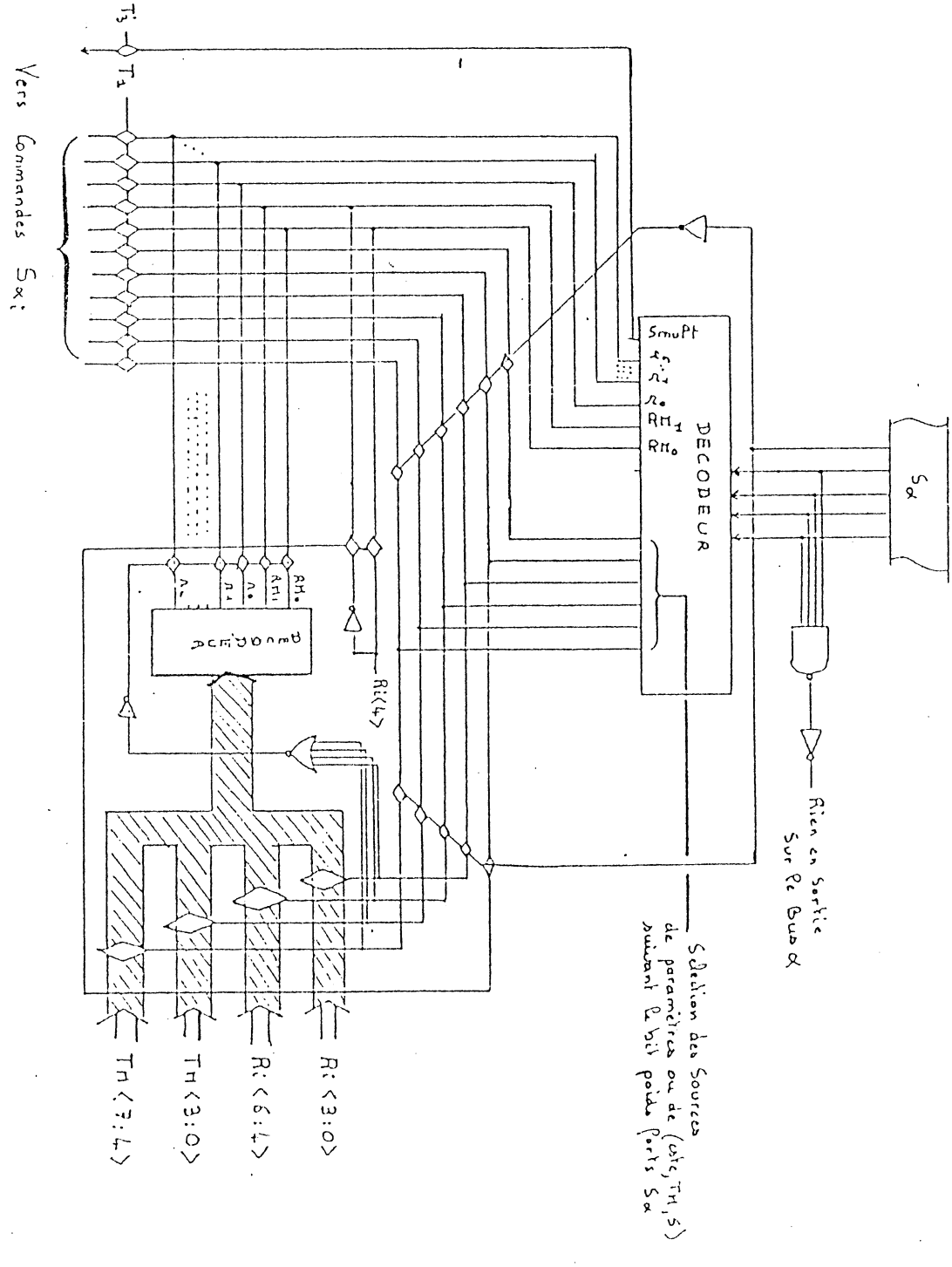
Test	ENTREES RI 14	13	12	11	10	9	8	7	6	RESET	ALARME	N	Z	L	CBR 1	2	3	4	5	SORTIES PUSH	PULL	n° de µ.inst.	Adresse
T0										0	0				0	0	0	0	0	0	0	6	5
										0	1				0	0	0	0	0	0	0	2	0
										1	0				0	0	0	0	0	0	0	3	3
										1	1				0	0	0	0	0	0	0	5	1
T8	0														0	0	0	0	1	0	0	36	36
	1														0	0	0	0	1	0	0	35	35
T9														0	0	0	0	1	0	0	0	39	40
														1	0	0	0	1	0	0	0	43	45
T10													1		0	0	0	1	1	0	0	41	43
													0		0	0	0	1	1	0	0	51	56
T11													1		0	0	1	0	0	0	0	45	48
													0		0	0	1	0	0	0	0	44	47
T12													1		0	0	1	0	1	0	0	48	51
													0		0	0	1	0	1	0	0	46	49
T19		1	1	0	1				1						0	0	1	1	0	0	0	83	86
									0						0	0	1	1	0	0	0	85	88
T22												1			0	0	1	1	1	0	0	1	2
												0			0	0	1	1	1	0	0	90	94
T23 EOR		0	1	1	0										0	1	0	0	0	0	0	93	98
AND		0	1	1	1																		
OR		1	0	0	0																		
NAND		1	0	0	1																		
NOR		1	0	1	0																		
LOADM		0	1	0	0										0	1	0	0	0	0	0	136	121
COMPM		0	1	0	1										0	1	0	0	0	0	0	95	100
T24	0						1	0							0	1	0	0	1	0	0	100	105
	0							1							0	1	0	0	1	0	0	101	106
CALL SP															1	0			0	1	0		
RET SP															1	0			1	0	1		

Schéma logique des sous-ensembles de génération des signaux de commande de lecture et d'écriture dans les registres de la Partie Opérative

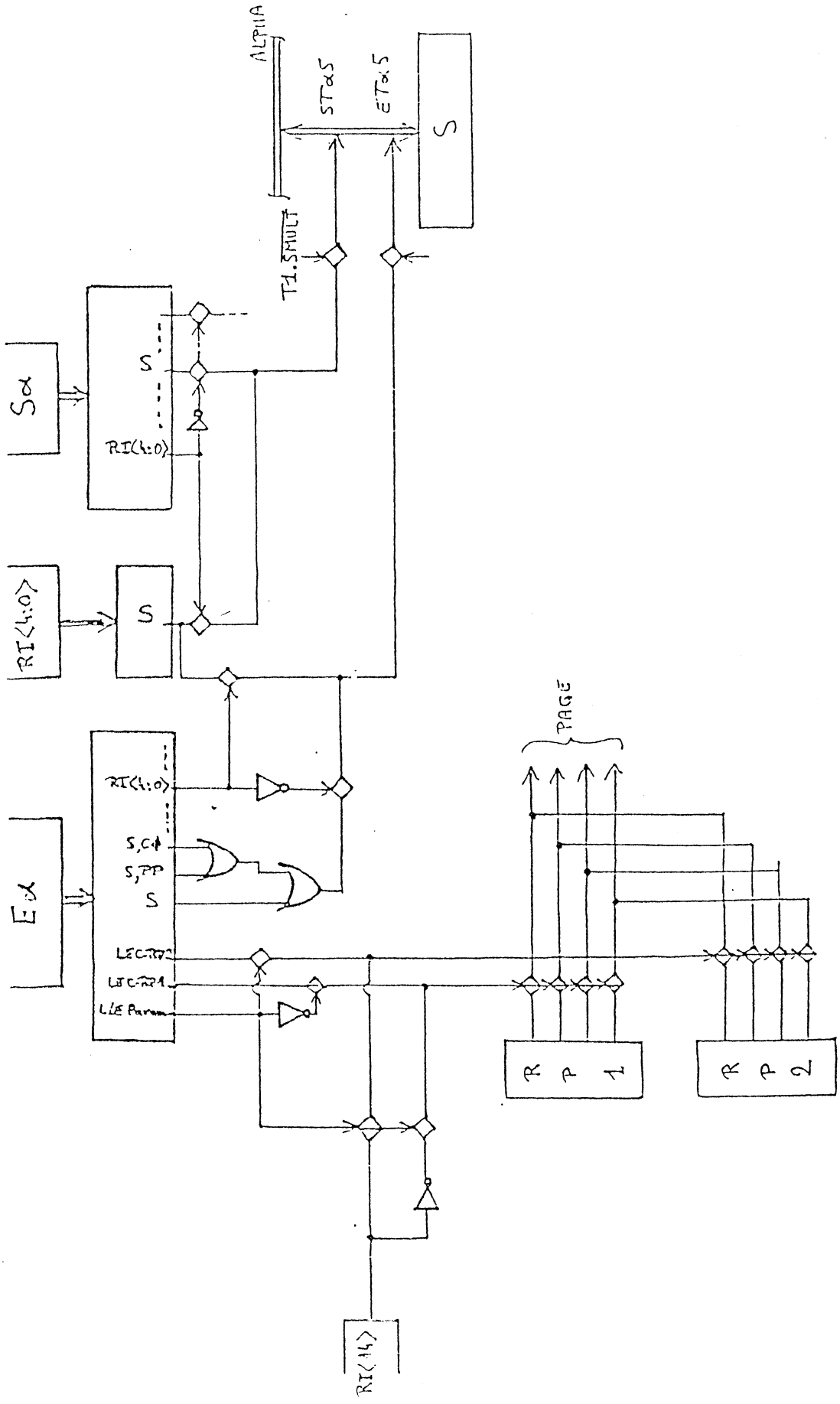
ANNEXE-4.15


Ri<4:0>
Ri<3:0>
dRi<6:4>
Ri<3:0>
Ri<6:4>
Ri<4:0>
Ex Décodé
Vers Commandes Exi
Autor. TRC
Autor VSR
Decl. Autor.
RAZ Sgn
Ri<1:0>
Ri<11>
OR UAL
SHIFT
Sx
T3
T1

ANNEXE-4.16


T3
T1
Vers Commandes Sxi
DECODEUR
Sα
Rien en Sortie Sur Pc Bus α
Sélection des Sources de paramètres ou de (code, TH, S) suivant le bit poids fort Sα
Ri<4>
Ri<3:0>
Ri<6:4>
TH<3:0>
TH<7:4>

ANNEXE-4.17


ALPHA
STOS
ETOS
S
T1.SMULT
Sα
S
RI(4:0)
RI<4:0>
S
RI(4:0)
Eα
S,C1
S,PP
S
LEC-PP
LEC-P1
LIE Param
PAGE
R P 1
R P 2
RI(14)

AUTORISATION DE SOUTENANCE

DOCTORAT 3ème CYCLE, DOCTORAT-INGENIEUR, DOCTORAT USMG

Vu les dispositions de l'arrêté du 16 avril 1974,

Vu les dispositions de l'arrêté du 5 juillet 1984,

Vu les rapports de M. NOGUEZ G.

M. DUTAND F.

M. JAY Christian est autorisé

à présenter une thèse en vue de l'obtention du nouveau doctorat

mention microélectronique

Grenoble, le 12 JUIN 1988

Le Président de l'Université Scientifique

et Médicale

M. TANCHE

RESUME :

L'objet de l'étude est la conception d'un composant du type microprocesseur pour des applications à haute sûreté de fonctionnement. Le circuit conçu est caractérisé par une faible latence d'erreur, par des facilités de test en fin de fabrication et par des facilités d'implantation de mécanismes de détection d'erreurs en ligne (codage et vérification de l'information traitée).

Partant d'un jeu d'instructions spécifiques à l'application (domaine des automatismes logiques), on propose une architecture permettant d'éxécuter ledit jeu d'instructions et disposant de facilités de test en fin de conception et au cours de la vie du circuit. L'observabilité et la contrôlabilité du composant représentent une partie importante de l'étude. Après examen critique de plusieurs méthodes permettant de faciliter le test (en ligne et hors ligne) du circuit, un choix est réalisé afin d'intégrer dans l'architecture du microprocesseur les dispositifs nécessaires à la mise en œuvre de certaines d'entre elles.

Ce microprocesseur à été partiellement réalisé en technologie C.MOS dans le cadre du "Circuits-Multi-Projets".

MOTS-CLES :

MICROPROCESSEUR, CIRCUIT SPECIFIQUE, CIRCUIT INTEGRE, TEST-EN-LIGNE, TEST-HORS-LIGNE, CONTROLABILITE, OBSERVABILITE, SURETE DE FONCTIONNEMENT.