



HAL
open science

Méthodologie de test de processeurs : impacts sur la conception

Chantal Robach

► **To cite this version:**

Chantal Robach. Méthodologie de test de processeurs : impacts sur la conception. Modélisation et simulation. Institut National Polytechnique de Grenoble - INPG; Université Joseph-Fourier - Grenoble I, 1975. Français. NNT : . tel-00286244

HAL Id: tel-00286244

<https://theses.hal.science/tel-00286244>

Submitted on 9 Jun 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THESE

présentée à

UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE
INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

POUR OBTENIR LE GRADE DE
DOCTEUR-INGENIEUR
spécialité Informatique

Chantal ROBACH

**Méthodologie de Test
de Processeurs.
Impacts sur la Conception.**

Thèse soutenue le 29 mai 1975 devant la commission d'examen

Président : Monsieur J. KUNTZMANN
Madame G. SAUCIER
Examineurs : Monsieur R. BEAUFILS
Monsieur J. GUALINO
Monsieur M. SCHWOB

UNIVERSITE SCIENTIFIQUE
ET MEDICALE DE GRENOBLE

INSTITUT NATIONAL POLYTECHNIQUE
DE GRENOBLE

M. Michel SOUTIF

Présidents

M. Louis NEEL

M. Gabriel CAU

Vice-Présidents

MM. Lucien BONNETAIN

Jean BENOIT

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

PROFESSEURS TITULAIRES

MM.	ANGLES D'AURIAC Paul	Mécanique des fluides
	ARNAUD Paul	Chimie
	AUBERT Guy	Physique
	AYANT Yves	Physique approfondie
Mme	BARBIER Marie-Jeanne	Electrochimie
MM.	BARBIER Jean-Claude	Physique expérimentale
	BARBIER Reynold	Géologie appliquée
	BARJON Robert	Physique nucléaire
	BARNOUD Fernand	Biosynthèse de la cellulose
	BARRA Jean-René	Statistiques
	BARRIE Joseph	Clinique chirurgicale
	BEAUDOING André	Clinique de Pédiatrie et Puériculture
	BERNARD Alain	Mathématiques Pures
Mme	BERTRANDIAS Françoise	Mathématiques Pures
MM.	BEZES Henri	Pathologie chirurgicale
	BLAMBERT Maurice	Mathématiques Pures
	BOLLIET Louis	Informatique (IUT B)
	BONNET Georges	Electrotechnique
	BONNET Jean-Louis	Clinique ophtalmologique
	BONNET-EYMARD Joseph	Pathologie médicale
	BOUCHERLE André	Chimie et toxicologie
	BOUCHEZ Robert	Physique nucléaire
	BOUSSARD Jean-Claude	Mathématiques appliquées
	BRAVARD Yves	Géographie
	CABANEL Guy	Clinique rhumatologique et hydrologie
	CALAS François	Anatomie
	CARLIER Georges	Biologie végétale
	CARRAZ Gilbert	Biologie animale et pharmacodynamie
	CAU Gabriel	Médecine légale et toxicologie
	CAUQUIS Georges	Chimie organique
	CHABAUTY Claude	Mathématiques Pures
	CHARACHON Robert	Clinique Oto-Rhino-Laryngologique
	CHATEAU Robert	Thérapeutique (Neurologie)
	CHIBON Pierre	Biologie animale
	COEUR André	Pharmacie chimique et chimie analytique
	CONTAMIN Robert	Clinique gynécologique
	COUDERC Pierre	Anatomie pathologique
	CRAYA Antoine	Mécanique
Mme	DEBELMAS Anne-Marie	Matière médicale
MM.	DEBERMAS Jacques	Géologie générale
	DEGRANGE Charles	Zoologie
	DELORMAS Pierre	Pneumo-Phtisiologie
	DEPORTES Charles	Chimie minérale
	DESRE Pierre	Métallurgie
	DESSAUX Georges	Physiologie animale
	DODU Jacques	Mécanique appliquée

MM.	DOLIQUE Jean-Michel	Physique des plasmas
	DREYFUS Bernard	Thermodynamique
	DRUCROS Pierre	Cristallographie
	DUGOIS Pierre	Clinique de dermatologie et syphiligraphie
	FAU René	Clinique neuro-psychiatrique
	GAGNAIRE Didier	Chimie physique
	GALLISSOT François	Mathématiques pures
	GALVANI Octave	Mathématiques pures
	GASTINEL Noël	Mathématiques appliquées
	GAVEND Michel	Pharmacologie
	GEINDRE Michel	Electroradiologie
	GERBER Robert	Mathématiques pures
	GERMAIN Jean-Pierre	Mécanique
	GIRAUD Pierre	Géologie
	JANIN Bernard	Géographie
	KAHANE André	Physique Générale
	KLEIN Joseph	Mathématiques pures
	KOSZUL Jean-Louis	Mathématiques pures
	KRAVTCHENKO Julien	Mécanique
	KUNTZMANN Jean	Mathématiques appliquées
	LACAZE Albert	Thermodynamique
	LACHARME Jean	Biologie végétale
	LAJZEROWICZ Joseph	Physique
	LATREILLE René	Chirurgie générale
	LATURAZE Jean	Biochimie pharmaceutique
	LAURENT Pierre-Jean	Mathématiques appliquées
	LEDRU Jean	Clinique médicale B
	LLIBOUTRY Louis	Géophysique
	LONGEQUEUE Jean-Pierre	Physique nucléaire
	LOUP Jean	Géographie
Mlle	LUTZ Elisabeth	Mathématiques pures
	MALGRANGE Bernard	Mathématiques pures
	MALINAS Yves	Clinique obstétricale
	MARTIN-NOEL Pierre	Seméiologie médicale
	MAZARE Yves	Clinique médicale A
	MICHEL Robert	Minéralogie et pétrographie
	MICOUJ Max	Clinique maladies infectieuses
	MOURIQUAND Claude	Histologie
	MOUSSA André	Chimie nucléaire
	MULLER Jean-Michel	Thérapeutique (néphrologie)
	NEEL Louis	Physique du solide
	OZENDA Paul	Botanique
	PAYAN Jean-Jacques	Mathématiques pures
	PEBAY-PEYROULA Jean-Claude	Physique
	RASSAT André	Chimie systématique
	RENARD Michel	Thermodynamique
	RINALDI Renaud	Physique
	DE ROUGEMONT Jacques	Neuro-chirurgie
	SEIGNEURIN Raymond	Microbiologie et hygiène
	SENGEL Philippe	Zoologie
	SIBILLE Robert	Construction mécanique
	SOUTIF Michel	Physique générale
	TANCHE Maurice	Physiologie
	TRAYNARD Philippe	Chimie générale
	VAILLANT François	Zoologie
	VALENTIN Jacques	Physique nucléaire
	VAUQUOIS Bernard	Calcul électronique
Mme	VERAIN Alice	Pharmacie galénique
MM.	VERAIN André	Physique
	VEYRET Paul	Géographie
	VIGNAIS Pierre	Biochimie médicale

PROFESSEURS ASSOCIES

MM.	CHEEKE John	Thermodynamique
	COPPENS Philip	Physique
	CORCOS Gilles	Mécanique
	CRABBE Pierre	CERMO
	GILLESPIE John	I.S.N.
	ROCKAFELLAR Ralph	Mathématiques appliquées

PROFESSEURS SANS CHAIRE

Mlle	AGNIUS-DELDORD Claudine	Physique pharmaceutique
	ALARY Josette	Chimie analytique
MM.	AMBROISE-THOMAS Pierre	Parasitologie
	BELORIZKY Elie	Physique
	BENZAKEN Claude	Mathématiques appliquées
	BERTRANDIAS Jean-Paul	Mathématiques pures
	BIAREZ Jean-Pierre	Mécanique
	BILLET Jean	Géographie
Mme	BONNIER Jane	Chimie générale
MM.	BOUCHET Yves	Anatomie
	BRUGEL Lucien	Energétique
	CONTE René	Physique
	DEPASSEL Roger	Mécanique des fluides
	GAUTHIER Yves	Sciences biologiques
	GAUTRON René	Chimie
	GIDON Paul	Géologie et Minéralogie
	GLENAT René	Chimie organique
	GROULADE Joseph	Biochimie médicale
	HACQUES Gérard	Calcul numérique
	HOLLARD Daniel	Hématologie
	HUGONOT Robert	Hygiène et Méd. Préventive
	IDELMAN Simon	Physiologie animale
	JOLY Jean-René	Mathématiques pures
	JULLIEN Pierre	Mathématiques appliquées
Mme	KAHANE Josette	Physique
MM.	KUHN Gérard	Physique
	LOISEAUX Jean	Physique nucléaire
	LUU-DUC-Cuong	Chimie organique
	MAYNARD Roger	Physique du solide
	PELMONT Jean	Biochimie
	PERRIAUX Jean-Jacques	Géologie et minéralogie
	PFISTER Jean-Claude	Physique du solide
Mlle	PIERY Yvette	Physiologie animale
MM.	RAYNAUD Hervé	Mathématiques appliquées
	REBECQ Jacques	Biologie (CUS)
	REVOL Michel	Urologie
	REYMOND Jean-Charles	Chirurgie générale
	RICHARD Lucien	Biologie végétale
Mme	RINAUDO Marguerite	Chimie macromoléculaire
MM.	ROBERT André	Chimie papetière
	SARRAZIN Roger	Anatomie et chirurgie
	SARROT-REYNAULD Jean	Géologie
	SIROT Louis	Chirurgie générale
Mme	SOUTIF Jeanne	Physique générale
MM.	VIALON Pierre	Géologie
	VAN CUTSEM Bernard	Mathématiques appliquées

MAITRES DE CONFERENCES ET MAITRES DE CONFERENCES AGREGES

MM.	AMBLARD Pierre	Dermatologie
	ARMAND Gilbert	Géographie
	ARMAND Yves	Chimie
	BARGE Michel	Neurochirurgie
	BEGUIN Claude	Chimie organique
Mme	BERIEL Hélène	Pharmacodynamique
M.	BOUCHARLAT Jacques	Psychiatrie adultes
Mme	BOUCHE Liane	Mathématiques (CUS)
MM.	BRODEAU François	Mathématiques (IUT B)
	BUISSON Roger	Physique
	BUTEL Jean	Orthopédie
	CHAMBAZ Edmond	Biochimie médicale
	CHAMPETIER Jean	Anatomie et organogénèse
	CHARDON Michel	Géographie
	CHERADAME Hervé	Chimie papetière
	CHIAVERINA Jean	Biologie appliquée (EFP)
	COHEN-ADDAD Jean-Pierre	Spectrométrie physique
	COLOMB Maurice	Biochimie médicale
	CORDONNIER Daniel	Néphrologie
	COULOMB Max	Radiologie
	CROUZET Guy	Radiologie
	CYROT Michel	Physique du solide
	DELOBEL Claude	M.I.A.G.
	DENIS Bernard	Cardiologie
	DOUCE Roland	Physiologie végétale
	DUSSAUD René	Mathématiques (CUS)
Mme	ETERRADOSSI Jacqueline	Physiologie
MM.	FAURE Jacques	Médecine légale
	FONTAINE Jean-Marc	Mathématiques pures
	GAUTIER Robert	Chirurgie générale
	GENSAC Pierre	Botanique
	GIDON Maurice	Géologie
	GRIFFITHS Michaël	Mathématiques appliquées
	GROS Yves	Physique (stag.)
	GUITTON Jacques	Chimie
	HICTER Pierre	Chimie
	IVANES Marcel	Electricité
	JALBERT Pierre	Histologie
	KOLODIE Lucien	Hématologie
	KRAKOWIAK Sacha	Mathématiques appliquées
Mme	LAJZEROWICZ Jeannine	Physique
MM.	LEROY Philippe	Mathématiques
	MACHE Régis	Physiologie végétale
	MAGNIN Robert	Hygiène et médecine préventive
	MARECHAL Jean	Mécanique
	MARTIN-BOUYER Michel	Chimie (CUS)
	MICHOULIER Jean	Physique (IUT A)
Mme	MINIER Colette	Physique
MM.	NEGRE Robert	Mécanique
	NEMOZ Alain	Thermodynamique
	PARAMELLE Bernard	Pneumologie
	PECCOUD François	Analyse (IUT B)
	PEFFEN René	Métallurgie
	PERRET Jean	Neurologie
	PHELIP Xavier	Rhumatologie
	RACHAIL Michel	Médecine interne
	RACINET Claude	Gynécologie et obstétrique
	RAMBAUD Pierre	Pédiatrie
Mme	RENAUDET Jacqueline	Bactériologie
MM.	ROBERT Jean-Bernard	Chimie-Physique

MM.	ROMIER Guy	Mathématiques (IUT B)
	SHOM Jean-Claude	Chimie générale
	STIEGLITZ Paul	Anesthésiologie
	STOEBNER Pierre	Anatomie pathologique
	VROUSOS Constantin	Radiologie

MAITRES DE CONFERENCES ASSOCIES

MM.	COLE Antony	Sciences nucléaires
	FORELL César	Mécanique
	MOORSANI Kishin	Physique

CHARGES DE FONCTIONS DE MAITRES DE CONFERENCES

MM.	BOST Michel	Pédiatrie
	CONTAMIN Charles	Chirurgie thoracique et cardio-vasculaire
	FAURE Gilbert	Urologie
	MALLION Jean-Michel	Médecine du travail
	ROCHAT Jacques	Hygiène et hydrologie

Fait à Saint Martin d'Hères, OCTOBRE 1974.

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

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

PROFESSEUR ASSOCIE

M. BOUDOURIS Georges	Radioélectricité
----------------------	------------------

PROFESSEURS SANS CHAIRE

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

MAITRES DE CONFERENCES

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

MAITRE DE CONFERENCES ASSOCIE

M. LANDAU Ioan Doré	Automatique
---------------------	-------------

CHARGE DE FONCTIONS DE MAITRES DE CONFERENCES

M. ANCEAU François	Mathématiques appliquées
--------------------	--------------------------

"Pour bien savoir les choses, il en faut savoir le détail;
et, comme il est presque infini, nos connaissances
sont toujours superficielles et imparfaites".

- La Rochefoucauld -

*A mes parents,
A mes frères,
A mes amis.*

AVANT-PROPOS

Le travail présenté dans cette thèse a été effectué au sein de l'équipe Mathématiques et Logique du Hardware de l'Ecole Nationale Supérieure d'Informatique et de Mathématiques Appliquées de Grenoble, dirigée par Monsieur le Professeur J. KUNTZMANN.

Nous tenons tout d'abord à exprimer notre profonde gratitude à Monsieur le Professeur J. KUNTZMANN pour la confiance qu'il nous a témoignée en nous accueillant au sein de l'ENSIMAG. Il nous fait l'honneur, malgré ses nombreuses occupations, de présider notre jury de thèse. Nous tenons à lui exprimer nos très vifs remerciements.

Nous sommes très honorée et très touchée de la présence de Monsieur le Professeur R. BEAUFILS, de l'Université Paul Sabatier à Toulouse. Nous le remercions très sincèrement d'avoir bien voulu accepter de faire partie de ce jury.

Que Monsieur SCHWOB, Chef du Département Test et Diagnostic de la CII et Monsieur GUALINO, Conseiller Scientifique au SESORI trouvent ici l'expression de notre gratitude pour leur présence à cette commission et l'intérêt qu'ils ont bien voulu accorder à nos travaux.

Chef de l'Equipe Mathématiques et Logique du Hardware, Madame G. SAUCIER, Maître de Conférences à l'ENSIMAG nous a toujours accordé sa confiance dans nos travaux. Madame SAUCIER fut notre premier guide et nous a apporté la formation sans laquelle ce travail n'aurait pas été possible. Nous tenons à lui témoigner ici notre reconnaissance pour ses nombreux conseils et sa contribution à l'élaboration de cette thèse.

De plus, nous ne saurions dissocier le contenu de cette étude et le cadre qui a permis de la mener à bien. Nous tenons à remercier les membres de l'Equipe Mathématiques et Logique du Hardware pour les nombreuses discussions critiques qui nous ont permis d'améliorer ce travail.

Nous ne saurions clore cet avant-propos sans exprimer nos très sincères et vifs remerciements à Madame J. CARRY pour sa gentillesse et sa compétence, à Monsieur S. RASOLONJATOVO ainsi qu'à Monsieur D. IGLESIAS et le Service de Reprographie qui ont assuré la réalisation matérielle de cette thèse.

Le SESORI a contribué matériellement, au titre du contrat CRI 73011 à la première phase de ces études. Nous remercions, à ce propos, Monsieur PIGNAL, Chargé du Groupement Calcul Electronique et Informatique du C.N.E.T. qui s'est intéressé à nos travaux et s'est fait notre porte-paroles auprès du SESORI.

TABLE DES MATIERES

* *
* *

CHAPITRE I

INTRODUCTION

I DEFINITIONS ET HYPOTHESES DE PANNES

- 1.1. Définitions
- 1.2. Hypothèses de pannes

II METHODES DE TEST AU NIVEAU COMPOSANT

- 2.1. Méthodes de génération de vecteurs de test dans le cadre d'une approche analytique
 - 2.1.1. Les approches déterministes
 - 2.1.2. L'approche aléatoire
- 2.2. Approches fonctionnelles
 - 2.2.1. L'approche fonctionnelle exhaustive
 - 2.2.2. L'approche fonctionnelle avec opérandes aléatoires
 - 2.2.3. méthodes d'identification d'automates

III COMPARAISON, EFFICACITE ET LIMITES DE CES METHODES

- 3.1. Approche déterministe/Approche aléatoire pour la génération des vecteurs de test
 - 3.1.1. Circuits de petite et moyenne complexité
 - 3.1.2. Circuits de grande complexité
- 3.2. Approche déterministe/Approches fonctionnelles
- 3.3. Efficacité relative des méthodes citées. Tableau récapitulatif
- 3.4. Problème de la composition au niveau système

IV PROPOSITION DE METHODES STRUCTURELLES

- 4.1. Caractéristiques des méthodes recherchées
- 4.2. Définition d'une méthode structurelle
- 4.3. Exemples

4.

4.3.1. *Exemple 1*

4.3.2. *Exemple 2*

4.3.3. *Exemple 3*

4.4. *Principes d'une méthode structurelle*

C H A P I T R E I I

I PROGRAMME DE TEST OU DE MAINTENANCE

II CARACTERISTIQUES GENERALES

III ETUDE DES DIFFERENTES APPROCHES DE BASE

- 3.1. *Approche de type "start-big"*
- 3.2. *Approche de type "start-small"*
- 3.3. *Approche de type "multiple clue"*
- 3.4. *Conclusion*

IV CARACTERISTIQUES D'UN PROGRAMME DE TEST EFFICACE

V METHODOLOGIE GENERALE DE TEST

- 5.1. *Problème d'initialisation*
 - 5.1.1. *Précisions sur la structure du hardware*
 - 5.1.2. *Problème du test du hardware de deuxième degré*
- 5.2. *Niveau macro-bloc*
 - 5.2.1. *Définition*
 - 5.2.2. *Ordre de test au niveau macro-bloc*
- 5.3. *Niveau micro-bloc*
 - 5.3.1. *Partition d'un macro-bloc en micro-blocs*
 - 5.3.2. *Analyse du chemin de données*
 - 5.3.3. *Analyse des niveaux de commande*
 - 5.3.4. *Problème du cheminement de l'information de test*
 - 5.3.5. *Traitement au niveau d'un micro-bloc*

VI IMPACTS SUR LA CONCEPTION

C H A P I T R E I I I

INTRODUCTION

I CLASSIFICATION DES AUTOMATES

1.1. *Définitions*

1.2. *Modélisation de l'automate. Complexité*

1.2.1. *Modélisation de l'automate*

1.2.2. *Complexité*

1.3. *Contraintes d'entrée*

1.3.1. *Pas de contraintes*

1.3.2. *Dépendance faible*

1.3.3. *Dépendance forte*

1.4. *Critères d'observabilité et de distinguabilité*

1.4.1. *Critère de distinguabilité : niveau intrinsèque*

1.4.2. *Critère d'observabilité : niveau global*

1.5. *Impacts de l'observabilité sur la distinguabilité*

1.5.1. *Structure de type 1*

1.5.2. *Structure généralisée déduite de la structure de type 1*

1.5.3. *Structure de type 2*

1.5.4. *Structure généralisée*

1.5.5. *Exemple*

II ETUDE D'AUTOMATES DE CONTROLE - ETUDE DE TYPE 1

2.1. *Classification de l'automate*

2.1.1. *Modélisation de l'automate - Complexité*

2.1.2. *Définition du quintuplet $\{X, Q, Z, \delta, \lambda\}$*

2.1.3. *Contraintes d'entrée*

2.1.4. *Observabilité*

2.2. *Méthodologie de test*

2.3. *Résultats pratiques*

- 2.3.1. *Algorithme réalisé*
- 2.3.2. *Structure hardware*
- 2.3.3. *Classification de l'automate*
- 2.3.4. *Tableau d'états associé*
- 2.3.5. *Résultats pratiques du test*

III ETUDE DE TYPE 2

- 3.1. *Classification de l'automate*
 - 3.1.1. *Modélisation de l'automate - Complexité*
 - 3.1.2. *Contraintes d'entrée*
 - 3.1.3. *Observabilité - Distinguable*
 - 3.1.4. *Impacts sur le choix d'une méthode de test*
- 3.2. *Méthodologie de test*
 - 3.2.1. *Remarque préliminaire*
 - 3.2.2. *Différents types de pannes*
 - 3.2.3. *Pannes sur les éléments de mémorisation*
 - 3.2.4. *Pannes sur les fils de commande*
 - 3.2.5. *Pannes sur le séquençement des phases*
 - 3.2.6. *Organisation générale du test*
- 3.3. *Résultats pratiques*
 - 3.3.1. *Algorithmes réalisés*
 - 3.3.2. *Structure hardware*
 - 3.3.3. *Classification de l'automate*
 - 3.3.4. *Résultats pratiques du test*

IV CONCLUSION

C H A P I T R E I V

INTRODUCTION

I SYNTHESE ET TESTABILITE

- 1.1. *Hypothèses préliminaires*
- 1.2. *Définitions*
- 1.3. *Principe général de l'algorithme*
 - 1.3.1. *Définition des paramètres*
 - 1.3.2. *Exposé général de l'algorithme*
 - 1.3.3. *Organigramme simplifié*
- 1.4. *Problèmes à résoudre*

II ETUDE ET RESOLUTION DES PROBLEMES PRELIMINAIRES

- 2.1. *Etablissement des critères d'adjonction de sorties supplémentaires*
 - 2.1.1. *Critères*
 - 2.1.2. *Remarque*
 - 2.1.3. *Rappels : définition d'un arbre de distinction*
 - 2.1.4. *Conclusion*
- 2.2. *Limite supérieure de μ*
- 2.3. *Propriété*

III CODAGE DES SORTIES SUPPLEMENTAIRES

- 3.1. *Précodage et codage*
- 3.2. *Etape α : précodage*
- 3.3. *Etape β : codage complet des sorties supplémentaires*

IV ORGANIGRAMME

- 4.1. *Organigramme*
- 4.2. *Exemple*

V ELABORATION DE LA SEQUENCE DE TEST D'UN AUTOMATE μ -TESTABLE

- 5.1. *Algorithme de recherche d'un chemin eulérien*

5.1.1. *Algorithme*

5.1.2. *Exemple*

5.2. *Algorithme de recherche du plus court chemin*

C H A P I T R E V

INTRODUCTION

I STRUCTURE HARDWARE DE L'UNITE CENTRALE

- 1.1. *Les blocs mémoires*
- 1.2. *Les unités arithmétiques et logiques*
- 1.3. *Les registres*
- 1.4. *Les bus*
- 1.5. *Les organes de liaison entre l'unité centrale et l'extérieur*

II CARACTERISTIQUES GENERALES

- 2.1. *Unité centrale*
 - 2.1.1. *Micro-machine*
 - 2.1.2. *Sous-ensembles*
- 2.2. *Format des informations - Représentation des nombres flottants*
- 2.3. *Les microinstructions - Leur structure*
 - 2.3.1. *La partie opérative*
 - 2.3.2. *Partie adressage de la mémoire microprogrammée*
 - 2.3.3. *En résumé*
- 2.4. *Parallélisme*

III MICROPROGRAMME DE TEST DE L'UNITE CENTRALE

- 3.1. *Facilités de test - Hardcore de 1er degré*
- 3.2. *Structure générale des microprogrammes de test*
 - 3.2.1. *Principe*
 - 3.2.2. *Structure du microprogramme de test*
 - 3.2.3. *Contrôle de validité de bon fonctionnement et utilisation*
- 3.3. *Structure détaillée des microprogrammes de test*
 - 3.3.1. *Test manuel (phase 1)*
 - 3.3.2. *Test automatique avec opérandes en mémoire morte (phase 2)*
 - 3.3.3. *Test automatique avec opérandes en mémoire centrale (phase 3)*

IV ETUDE DE DETAIL : LES UNITES ARITHMETIQUES ET LOGIQUES

- 4.1. *Structure de l'unité arithmétique et logique*
 - 4.1.1. *Le registre H*
 - 4.1.2. *Le registre N*
 - 4.1.3. *Le registre M*
 - 4.1.4. *Le registre RF*
 - 4.1.5. *L'additionneur ADD1*
 - 4.1.6. *L'unité de contrôle*
- 4.2. *Microprogrammation de l'unité arithmétique et logique*
 - 4.2.1. *Opérations sur les nombres entiers*
 - 4.2.2. *Opérations sur les nombres flottants et décalages*
- 4.3. *Méthodologie de test et ordonnancement*
 - 4.3.1. *Analyse du chemin de données*
 - 4.3.2. *Recherche d'un ordonnancement*
 - 4.3.3. *Analyse au niveau commandes*
- 4.4. *Traitement au niveau du micro-bloc*

V SPECIFICATIONS TECHNIQUES DU MICROPROGRAMME DE TEST

- 5.1. *Phase 1 : test manuel*
- 5.2. *Phase 2 : test automatique avec opérandes en mémoire morte*
- 5.3. *Phase 3 : test automatique avec opérandes en mémoire centrale*

INTRODUCTION

* *
*

L'intérêt porté au problème du test et de la maintenance des systèmes logiques s'est accru ces dernières années, en même temps que le développement des ordinateurs et de leur domaine d'applications.

Ce problème s'explique largement par l'utilisation croissante des ordinateurs pour des applications en temps réel telles que le contrôle de processus chimiques ou de réacteurs nucléaires où tout fonctionnement incorrect du calculateur peut être désastreux.

Les problèmes rencontrés pour l'élaboration de méthodes de test efficaces sont dûs essentiellement au manque de points de test dans les circuits LSI et au besoin de procédures de test performantes pour des réseaux logiques de plus en plus complexes.

On peut considérer qu'il y a trois politiques de protection face aux pannes :

α) techniques de haute fiabilité :

on range dans cette catégorie les techniques de masquage des pannes par redondance importante du matériel (structure K parni N)

Un élément est remplacé par n éléments identiques, cet élément n'ayant généralement pas de caractéristiques particulières face au test.

β) techniques d'adjonction du matériel en cours de synthèse en vue d'obtenir :

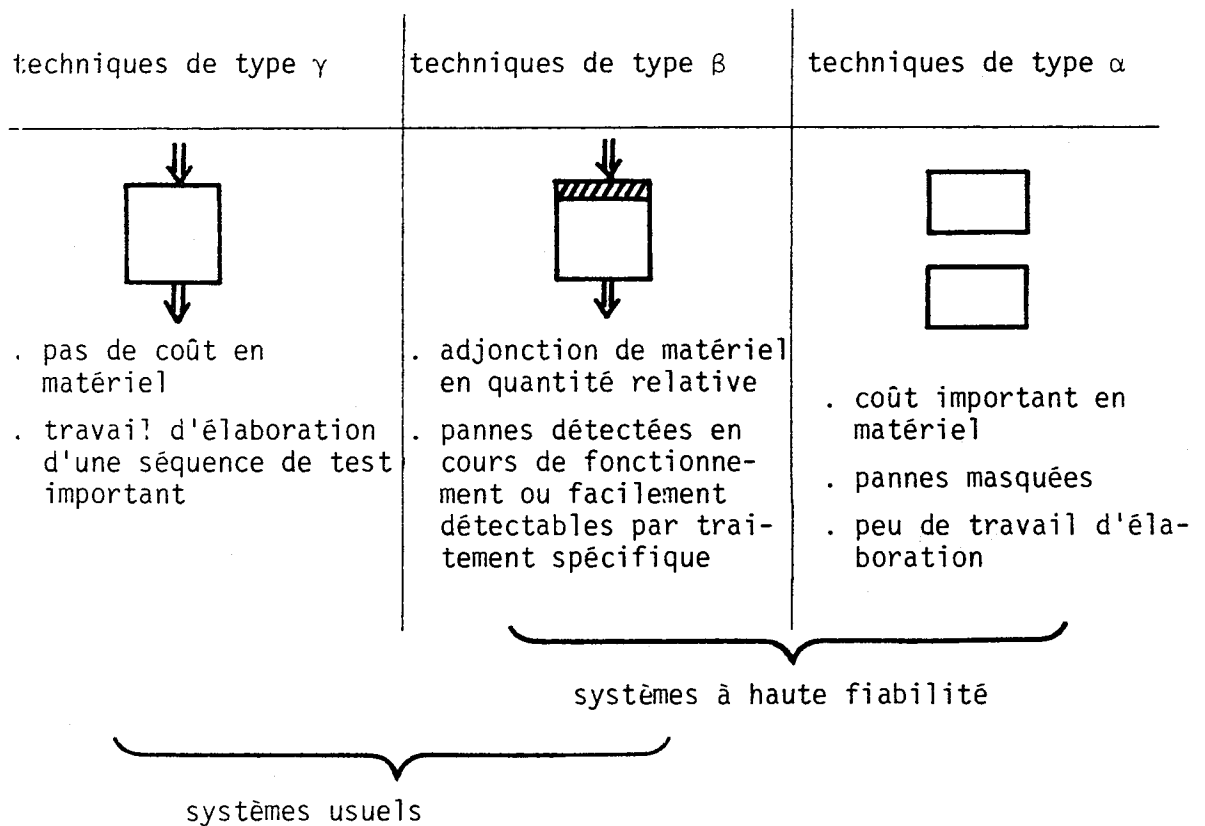
. soit une détection immédiate (et/correction) au niveau du module en cours de fonctionnement (circuit auto-test)

. soit une facilité plus grande de distinction (séquence de test très courte, circuit totalement testable)

. soit une détection partielle d'une certaine classe de pannes, au niveau transfert d'information (code correcteur, ...)

γ) protection face aux pannes sans adjonction de matériel :

il s'agit alors de détecter au mieux, et le plus vite possible, le plus grand nombre de pannes; on associe à chaque module hardware une séquence de test judicieuse et à l'ensemble du système un programme de test (ou de maintenance).



Dans l'étude présentée ici, on exposera des techniques de type γ (écriture de microprogrammes de test) et des techniques de type β (circuits facilement testables).

Le problème du test se pose à deux niveaux :

- niveau composant ou sous système
- niveau système.

Au niveau composant ou sous-système il existe un nombre important de méthodes de test; les résultats sont performants pour les circuits combinatoires mais les problèmes ne sont pas encore parfaitement résolus en ce qui concerne les circuits séquentiels; nous présenterons dans le chapitre I les principales méthodes caractéristiques et nous définirons les principes d'une méthode structurelle pour des circuits séquentiels caractérisant des automates de contrôle.

Au niveau système, les études actuelles sont beaucoup moins nombreuses et souvent spécifiques d'un ordinateur donné; notre objectif dans le chapitre II est de donner une méthodologie de test efficace pour de

grands systèmes en définissant un partitionnement du système et en introduisant des relations d'ordonnement.

Dans le chapitre III nous nous intéresserons au cas particulier des automates de contrôle; après avoir défini une classification des automates nous présenterons deux types généraux d'automates de contrôle pour lesquels une méthode de test efficace sera proposée (définition d'une méthode structurelle).

Dans la plupart des cas, l'élaboration d'une séquence de test pour des réseaux logiques nécessite des calculs importants. Il semble donc nécessaire de concevoir des circuits tenant compte de ces difficultés, c'est-à-dire d'introduire, au niveau de la synthèse, un critère lié à la testabilité du circuit.

L'approche proposée au chapitre IV conduit à la définition de systèmes facilement testables pour lesquels la séquence de test sera très simple à obtenir et à mettre en oeuvre.

Afin d'illustrer l'étude proposée aux précédents chapitres, on prendra un exemple pratique fourni par le geoprocesseur, ordinateur microprogrammé de moyenne importance pour lequel nous avons écrit un microprogramme de test. Le chapitre V illustrera donc les résultats des études précédentes à l'aide de cet exemple pratique.

CHAPITRE I

NIVEAU COMPOSANT OU SOUS-SYSTEME

* *
*

INTRODUCTION

Dans le domaine du composant ou sous-système, de nombreuses recherches ont été menées à bien et ont donné lieu à des méthodes de test élaborées.

Parmi ces méthodes un certain nombre trouvent peu d'applications pratiques et sont à l'heure actuelle d'un simple intérêt académique.

Nous exposerons les méthodes les plus connues et nous en donnerons dans ce chapitre, les caractéristiques générales, les avantages et les limites.

Une étude comparative nous permettra ensuite de définir le type de méthode le plus approprié à une classe donnée de circuits, d'une part intrinsèquement (structure du circuit), d'autre part compte tenu du contexte du circuit étudié (module appartenant à un système).

Nous proposerons ensuite des méthodes spécifiques et originales pour des parties caractéristiques d'un ordinateur et nous en définirons les principes. Il s'est en effet avéré nécessaire d'élaborer des méthodes spécialisées pour des circuits pour lesquels l'application de méthodes classiques n'était pas efficace ou donnait des séquences de test trop longues.

I – DEFINITIONS ET HYPOTHESES DE PANNES. [31], [61].

1.1 DEFINITIONS.

. Test : le problème du test est de déterminer des séquences d'entrée de longueur minimale ou pseudo-minimale qui, appliquées au réseau logique considéré, permettent de dire, au vu des sorties, si le système est ou non défaillant.

Le résultat d'un test permet de définir si le réseau logique est sans faute ou non et permet d'identifier la faute si elle existe : c'est ce qu'on appelle la détection.

La détection est généralement complétée par une localisation, aussi précise que possible, de la faute.

L'ensemble "détection + localisation" constitue le diagnostic.

. Les termes pannes, fautes ou erreurs sont utilisés, assez librement, pour se référer à toute déviation de l'opération attendue du système : soit la présence détectée d'une action non voulue, soit l'absence détectée d'une action voulue.

Les pannes pourront être le résultat d'erreurs de conception ou de fabrication ou être dûes à un mauvais fonctionnement d'éléments du système (diodes ou transistors défectueux, phénomène de diaphonie, court-circuits....) ou à des erreurs de programmation (non considérées ici).

1.2 HYPOTHESES DE PANNES. [31], [61].

. Les restrictions généralement adoptées sur le type de pannes pouvant être détectées et/ou localisées, sont les suivantes :

α) pannes logiques qui entraînent le collage d'une ligne (collage à zéro, collage à un).

β) le système reste logique sous l'hypothèse d'une panne.

γ) normalement, une seule panne est considérée à un moment donné (hypothèse de la panne unique).

ε) les fautes ne sont pas transitoires (ou intermittentes). En outre, certaines méthodes de test imposent leurs propres restrictions : par exemple,

la méthode de vérification d'automates exige que le tableau d'état du circuit sans faute soit fortement connexe et que le nombre d'états n'augmente pas sous l'hypothèse d'une panne.

. Le modèle de collage est celui qui est couramment adopté, et rares sont les solutions qui prennent en compte d'autres types de pannes tels que les court-circuits, les erreurs de câblage et les erreurs dues à l'environnement du système.

Un tel modèle considère uniquement les pannes statiques, les paramètres qui affectent le fonctionnement dynamique étant vérifiés par d'autres tests. Il suppose que les mécanismes d'erreur dans une porte résultent en un collage à 1 ou à 0 des entrées ou des sorties de la porte.

Parmi les fautes importantes non couvertes de façon adéquate par un tel modèle sont les court-circuits entre lignes physiquement adjacentes. Dans certaines technologies (RTL, DTL, ECL) ces fautes peuvent être modélisées par l'insertion d'une fonction ET ou d'une fonction OU entre les lignes coupées. Mais, même lorsque ce modèle est adéquat, la prise en compte de telles fautes introduit une grande complexité dans le test à cause du grand nombre de court-circuits possibles. On pourra réduire cette complexité en faisant intervenir la topologie du circuit.

. Avec un modèle de collage les pannes peuvent être décrites de façon convenable en utilisant des outils analytiques pour les circuits logiques, tels que l'Algèbre Booléenne ou ses extensions. Par conséquent, un tel modèle permet d'utiliser une représentation analytique tout en étant représentatif de beaucoup des pannes d'un système.

. Mais quel que soit le modèle de pannes choisi, on doit décider si l'hypothèse de panne simple est justifiable. Les tests basés sur les pannes uniques sont plus simples et plus courts que ceux considérant toutes les combinaisons possibles de pannes.

L'hypothèse de pannes simples, est basée sur l'estimation subjective que les pannes multiples ont une probabilité faible de ne pas être détectées par un test couvrant toutes les pannes simples.

Cependant, pour un équipement opérationnel, cette hypothèse a été justifiée : un test complet couvrant toutes les pannes simples détectera aussi la majorité des pannes multiples.

II – METHODES DE TEST AU NIVEAU COMPOSANT.

Les techniques décrites ici sont applicables à des circuits de taille moyenne, soit quelques centaines de portes.

L'étude présentée ne se veut pas exhaustive et nous considérerons seulement les méthodes les plus caractéristiques [5], [12], [31], [56], [61].

Au niveau composant ou sous-système on peut distinguer deux classes de méthodes [29].

c.) les méthodes proches du matériel qui prennent en compte la structure du circuit : approche analytique.

De telles méthodes, s'appuyant sur une analyse au niveau de la porte, sont conçues pour assurer que les composants hardware d'une unité opèrent correctement. Par conséquent, un test analytique assure que chaque porte ET donne une sortie égale à 1 si et seulement si toutes ses entrées ont la valeur logique 1; que chaque bascule peut être activée, désactivée ...; que les signaux d'horloge arrivent en temps opportun; ...

Dans le cadre de cette approche on considère deux méthodes de génération de vecteurs de test :

- les méthodes de propagation : D-algorithme, différence booléenne, ...
- l'approche aléatoire pure ou adaptative.

β) les méthodes indépendantes du hardware qui prennent uniquement en compte l'aspect fonctionnel du circuit : approche fonctionnelle.

Un tel test a pour but de vérifier que l'unité testée se comporte correctement, autrement dit qu'elle exécute bien sa tâche. On vérifiera par exemple qu'un compteur incrémente (ou décrémte) correctement; que le transfert de contrôle arrive sous les conditions requises; ...

Dans cette catégorie on trouvera :

- la vérification fonctionnelle avec opérandes aléatoires
- les méthodes d'identification d'automates.

2.1 METHODES DE GENERATION DE VECTEURS DE TEST DANS LE CADRE D'UNE APPROCHE ANALYTIQUE.

2.1.1 Les approches déterministes.

Les méthodes de sensibilisation du chemin de propagation sont

basées sur l'hypothèse du collage : il s'agit de partir d'un ensemble de pannes et de générer les vecteurs spécifiques qui testent cette panne; ce sont des approches qui, partant du schéma logique, sont analytiques et permettent de définir un ensemble de vecteurs de test minimal ou pseudo-minimal.

On peut distinguer trois phases :

- . manifestation de la panne : on considère une panne donnée (collage) à un endroit donné.
- . propagation de la panne : on propage la panne vers une sortie primaire à travers un chemin de sensibilisation c'est-à-dire un chemin tel que tout changement de la valeur logique du défaut le long de ce chemin se traduira par un changement correspondant à la sortie primaire.
- . phase de consistance : on détermine la valeur logique des entrées primaires du circuit de façon à permettre la propagation du défaut.

Cette méthode de sensibilisation du chemin de propagation a donné lieu à 2 méthodes principales : le D-algorithme et la différence booléenne.

α) le D-algorithme : il est caractérisé par la prise en compte de chemins multiples de propagation et il donne un test unique pour une panne donnée [54].

β) la différence booléenne : elle est caractérisée par une définition algébrique des chemins de propagation. Elle donne tous les tests possibles pour une panne donnée mais est très peu applicable pour les circuits séquentiels [59].

2.1.2 L'approche aléatoire. [41], [45]

Les combinaisons d'entrée sont, non plus fixées à l'avance, mais fournies par un générateur de séquences aléatoires; on recherche alors par simulation les défauts détectés; la simulation des circuits en présence de défauts permet d'une part d'établir les séquences de test et d'autre part d'établir, en partie, le dictionnaire des pannes avec les signaux de sortie correspondants (méthode de Monte Carlo).

Chaque configuration d'entrée qui détecte au moins une nouvelle faute est emmagasinée comme test.

On obtient le schéma suivant :

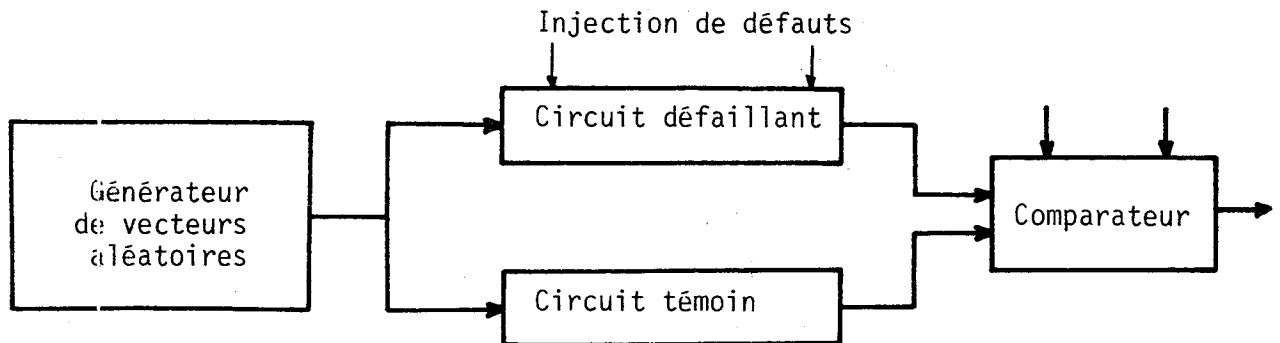


Fig. I.1.

2.2 LES APPROCHES FONCTIONNELLES.

2.2.1 L'approche fonctionnelle exhaustive.

Cette approche consiste à vérifier la fonction du circuit à tester en envoyant toutes les combinaisons d'entrée possibles.

Notons par exemple que le test fonctionnel exhaustif d'un additionneur 32 bits requiert 2^{64} opérations. Par conséquent un tel test n'est applicable que pour des circuits dont les entrées sont en nombre réduit et dont les fonctions sont très simples.

A l'heure actuelle, le test fonctionnel n'est appliqué qu'aux registres et aux bascules, dont les fonctions sont assez simples pour permettre un test fonctionnel complet.

Mais leur avantage consiste principalement en leur automatisation facile.

2.2.2 L'approche fonctionnelle avec opérandes aléatoires. [21], [22].

Cette approche consiste à appliquer au circuit, au moment du test effectif, des combinaisons d'entrée aléatoires. Dans ce cas, il s'agit de déterminer le nombre minimal de combinaisons d'entrée nécessaires pour assurer que tout défaut présent sera détecté avec une probabilité donnée.

Les entrées du circuit à tester et celles d'un circuit témoin réputé correct sont attaquées en parallèle par des signaux aléatoires, comme le montre la figure I.2.

Les réponses des deux circuits sont comparées; en cas de divergen-

ce, le circuit testé est déclaré défectueux.

Le problème est alors de fixer une durée maximale de test pour pouvoir attribuer à cette méthode un seuil de confiance donné.

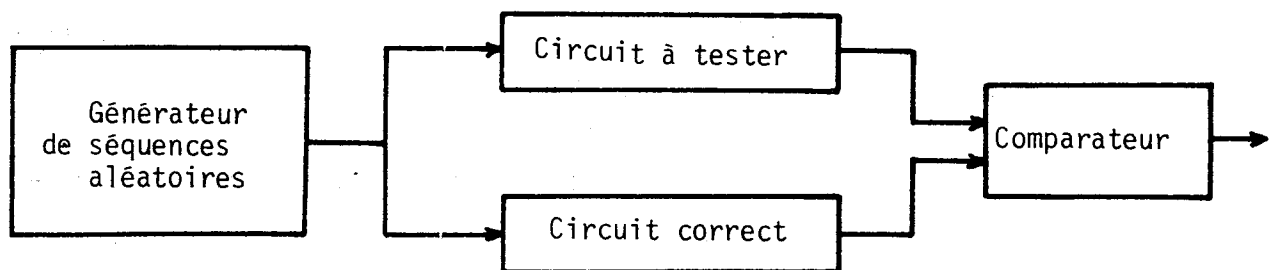


Fig. I.2.

2.2.3 Méthodes d'identification d'automates. [28], [34], [35].

Il s'agit, au niveau d'un module séquentiel de taille réduite, de vérifier son fonctionnement par une méthode de reconnaissance d'automates. On dispose uniquement de la description du fonctionnement du module (tableau d'état) sans connaître la réalisation physique.

La procédure d'identification est généralement la suivante :

- α) initialisation du système dans un état connu; cette phase utilise soit une séquence de synchronisation soit une séquence de positionnement ("homing sequence").
- β) vérification du nombre d'états et de toutes les transitions possibles entre états; cette phase utilise une séquence de distinction ("distinguishing sequence").

Cette méthode donne une sécurité plus grande (hypothèses moins restrictives sur le type de pannes) mais

- a) n'est applicable que dans certains cas bien déterminés et pour des automates de taille réduite.
- b) donne des séquences de test très longues.
- c) ne donne aucune indication de localisation.

III – COMPARAISON, EFFICACITÉ ET LIMITES DE CES METHODES.

3.1 APPROCHE DETERMINISTE / APPROCHE ALEATOIRE POUR LA GENERATION DES VECTEURS DE TEST. [1], [64], [67].

De nombreux résultats comparatifs ont été donnés sur l'efficacité respective d'une approche aléatoire et d'une approche déterministe (généralement, le D-algorithme ou ses extensions), sur des modules de complexité très diverse.

En effet, les méthodes probabilistes de génération de séquences de test ont souvent été opposées aux méthodes déterministes, essentiellement pour le test de circuits séquentiels.

Nous allons donner quelques résultats pour des complexités différentes et suivant divers paramètres (temps de test, pourcentage de pannes détectées par test, ...) :

3.1.1 Circuits de petite et moyenne complexité.

On considère dans ce paragraphe des circuits de petite complexité (jusqu'à 200 portes) ou de moyenne complexité (jusqu'à 1000 portes) et nous démontrerons l'intérêt de méthodes mixtes pour de tels circuits.

α) de façon générale, on peut noter le résultat suivant : concernant l'efficacité de détection des pannes (nombre de fautes détectées par unité de temps), on a montré que les méthodes aléatoires sont meilleures qu'une solution déterministe (en particulier, le D-algorithme) dans une première phase de la génération des tests, mais cette relation est inversée au-delà d'un certain seuil.

Cette inversion arrive, en pratique, pour un taux de pannes détectées de 70 % environ, pour des circuits de petite complexité [39].

β) de façon plus précise, on peut affiner la valeur de ce seuil suivant la structure du circuit à tester [1]. En effet, un paramètre décisif du choix d'une méthode de test est le nombre de niveaux de la logique testée, c'est-à-dire le nombre de portes d'une entrée primaire à une sortie primaire. Dès que le nombre de niveaux excède 8, un test aléatoire devient incomplet (temps trop long) et il devient préférable d'utiliser une méthode déterministe.

Par contre, pour 6 niveaux ou moins, une méthode déterministe est plus lente qu'une méthode aléatoire et ne fournit pas de meilleure détection.

La solution idéale est alors une méthode mixte, combinant les 2 types de méthodes : aléatoire et déterministe.

Le tableau suivant illustre ces résultats :

TABLEAU I.1.

Illiac IV Carte	Nombre de lignes	Nombre de niveaux	Temps de génération des vecteurs de test(s)		
			aléatoire	D-algorithme	mixte
BSA06	124	5	38	67	38
SGA03	112	6	63	88	64
FGSA	437	6	250	544	250
AMB09	89	7	79	65	49
CSA04	189	8	incomplet*	216	94
CLA11	128	9	incomplet*	125	114
CCB09A	115	10	incomplet*	137	146

* Le temps du programme de test ayant été limité à 15 mn, certaines cartes (plus de 8 niveaux) n'étaient pas entièrement testées par une méthode aléatoire.

γ) Conclusion : pour des circuits de petite et moyenne complexité on a les résultats suivants :

. supériorité d'une méthode aléatoire sur une méthode déterministe dans une première phase du test.

. solution idéale : méthode mixte combinant une méthode aléatoire et une méthode déterministe.

3.1.2 Circuits de grande complexité.

Pour des circuits de grande complexité, le nombre de test générés par une méthode aléatoire est très supérieur à celui généré par une méthode déterministe, pour détecter un nombre donné de pannes [64].

Le seuil à partir duquel une méthode aléatoire devient inefficace décroît en même temps que la complexité croît. La figure suivante illustre ce résultat :

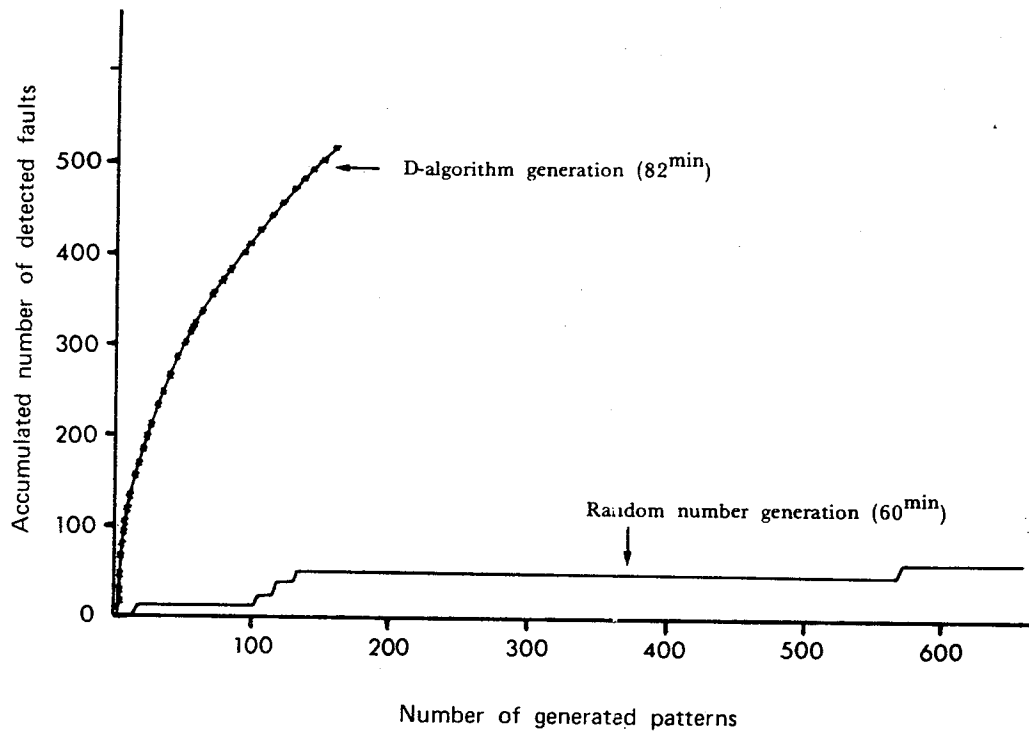


Fig. I.3.

Le D-algorithme fournit uniquement des vecteurs de test efficaces, c'est-à-dire testant à chaque fois de nouvelles pannes. Au contraire, dans le nombre de vecteurs de test générés par une méthode aléatoire, un faible pourcentage de ces vecteurs est efficace.

On peut donc conclure qu'au niveau de circuits de grande complexité, les méthodes aléatoires sont inefficaces.

3.2 APPROCHE DETERMINISTE / APPROCHES FONCTIONNELLES.

Peu de résultats ont été donnés concernant l'efficacité d'une approche déterministe comparativement à une approche fonctionnelle, tant il est évident qu'une méthode fonctionnelle (exhaustive ou avec opérandes aléatoires) donne un test très incomplet (approche fonctionnelle avec opérandes aléatoires) ou ne peut être employée que pour des circuits dont les fonctions sont très simples (approche fonctionnelle exhaustive).

3.3 EFFICACITE RELATIVE DES METHODES CITEES. TABLEAU RECAPITULATIF.

	Méthodes proches du hardware		Méthodes fonctionnelles	
	aléatoire	déterministe	aléatoire	identification d'automates
Complexité de l'unité testée	sous-système module	système sous-système module	système sous-système module	sous-système module
Nature du réseau	séquentiel combinatoire	séquentiel combinatoire	séquentiel combinatoire	séquentiel
Type de pannes	collage simple	collage simple (multiple, faute de câblage)		toute panne conservant le nombre d'états
Pourcentage de pannes détectées	environ 70%	80 à 100%	non défini	toute panne
Elaboration	facile	difficile (étude analytique)	facile	difficile
Localisation	très bonne	très bonne	aucune	aucune

TABLEAU I.2.

3.4 PROBLEME DE LA COMPOSITION AU NIVEAU SYSTEME.

Quelle que soit la méthode proposée, on s'intéressait jusqu'à présent au composant ou sous-système indépendamment de son contexte c'est-à-dire hors du système.

Nous considérons maintenant le critère le plus contraignant, à savoir : supposons les vecteurs élaborés au moyen de l'une quelconque de ces méthodes; la *composition* au niveau système (génération des vecteurs de test, observation des résultats de test) sera-t-elle plus ou moins facile selon la méthode choisie ?

Définition : on entend par problème de composition l'union des problèmes d'observabilité et de distinguabilité définis au Chap. II. § 5.3.

Composition = observabilité + distinguabilité

La réponse est claire : les approches analytiques ne prenant pas en compte l'aspect fonctionnel du circuit, présenteront de grosses contraintes de composition et aucune facilité n'est garantie, à l'encontre des méthodes fonctionnelles; malheureusement, pour ces dernières méthodes, l'une est très peu utilisable (identification d'automates), l'autre ne donne aucune garantie sur le taux de pannes détectées et la localisation.

Afin de résoudre ce problème et de tenir compte du contexte du circuit testé lorsqu'il est partie intégrante d'un système, on a été amené à élaborer et définir une méthode spécifique originale : la méthode structurelle.

Le problème de composition, inexistant au niveau du module lui-même, prend une importance primordiale lorsqu'on se place dans le contexte d'un système : en effet, au niveau du système, le module testé est rarement accessible ou observable directement et l'on doit tenir compte des modules à travers lesquels se font l'accès ou l'observation du module testé.

IV – PROPOSITION DE METHODES STRUCTURELLES.

On cherche des méthodes originales au niveau composant ou sous-système, tenant compte de son environnement : méthodes structurelles.

4.1 CARACTERISTIQUES DES METHODES RECHERCHEES.

On recherche :

α) la rigueur d'une méthode analytique vis-à-vis du taux de pannes détectées et de la localisation : on part de l'ensemble des pannes considérées à un niveau très fin (collages par exemple) et on cherche la manifestation de chaque panne.

β) on respecte l'aspect fonctionnel afin de garantir la commandabilité au niveau global.

Le point α) élimine les méthodes fonctionnelles; le point β) élimine les méthodes aléatoires et, dans certains cas, les méthodes de propagation classiques.

4.2 DEFINITION D'UNE METHODE STRUCTURELLE.

L'objectif est de combiner une approche analytique partant du matériel et la description fonctionnelle de ce module.

Un test basé sur cette méthode pourra s'exprimer comme suit :

$$T = \mathcal{F}(\{f_i(x_{ij})\}, \{X_k\})$$

Le test sera conc donné par :

- . un ensemble d'activations fonctionnelles $\{f_i(x_{ij})\}$
- . des paramètres d'entrées $\{X_k\}$

L'étude de base sera donc la suivante :

étude fonctionnelle + étude analytique

- contraintes sur les entrées
- contraintes sur la succession des états

4.3 EXEMPLES.

4.3.1 Exemple 1.

On cherche à réaliser la consistance d'un vecteur de données D et la composition de la valeur de commandes C .

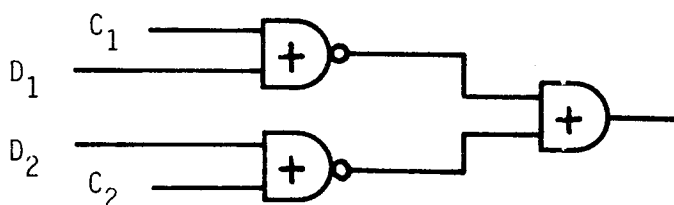


Fig. I.4.

Les entrées $C1$ et $C2$ étant des commandes d'entrée sur un bus, $D1$ et $D2$ des lignes de données (logique négative).

Lors de l'élaboration des vecteurs de test on pose immédiatement la contrainte $\overline{C1.C2} = 1$ (on élimine $C1 = C2 = 0$ soit deux voies d'accès au bus). On diminue ainsi le nombre de vecteurs de test (tous ceux contenant $\overline{C1.C2}$ n'étant plus considérés) par suite de l'interprétation fonctionnelle.

$\left\{ \begin{array}{l} \text{étude fonctionnelle soit la fonction d'accès au bus } f_i \\ \phantom{\text{étude fonctionnelle soit la}} + \\ \phantom{\text{étude fonctionnelle soit la}} \downarrow \\ \phantom{\text{étude fonctionnelle soit la}} \text{contraintes sur les entrées } C \\ \text{étude analytique} \end{array} \right.$

$$T = \mathcal{F}^*(f_i(C), D)$$

4.3.2 Exemple 2. |46|.

Cas de l'interconnexion de 2 circuits:

Soit une interconnexion de 2 modules dont les entrées ne sont pas indépendantes (cas pratiques fréquents au niveau des entrées de commande). Il se pose donc le problème de trouver les vecteurs de test au niveau des 2 modules interconnectés, ce qui amène des restrictions dans le choix des entrées ou même des impossibilités de détection pour certaines pannes.

Un exemple pratique a été fourni par un circuit du T 1600 (Télé-mécanique) et qui peut être représenté comme suit :

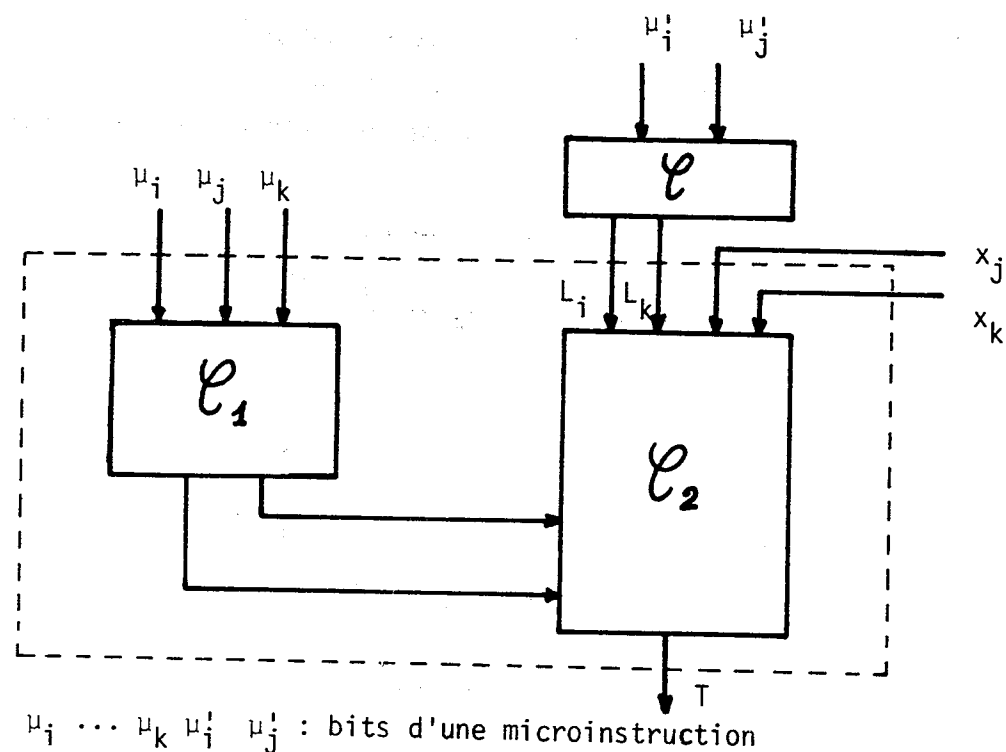


Fig. I.5.

Pour procéder au raccordement des vecteurs de test trouvés pour chacun des 2 sous-circuits \mathcal{L}_1 et \mathcal{L}_2 on doit tenir compte des contraintes de la microprogrammation.

Ce travail est rendu complexe et parfois impossible par le fait que les entrées (μ_i, μ_j, μ_k) ne sont pas indépendantes des entrées $(L_i, L_k, \mu'_i, \mu'_j)$ et l'on essaiera de tenir compte de ces relations de dépendance.

{

 étude fonctionnelle (fonction gérée par une microinstruction)

 +

 étude analytique

 }

↓

 contraintes sur les entrées $\mu_i \dots \mu_k \mu'_i \mu'_j$

$$T = \mathcal{G}(f_i(\mu_i \mu_j \mu_k \mu'_i \mu'_j), x_j x_k)$$

4.3.3 Exemple 3. [49].

Considérons les automates de contrôle : ce sont des réseaux qui ne traitent pas directement l'information et dont les sorties sont des commandes envoyées dans une partie opérative qui traite l'information.

Ces séquenceurs locaux réalisent un ou plusieurs algorithmes et leur test doit tenir compte de leur aspect fonctionnel.

L'exemple pratique rencontré peut être représenté comme suit :

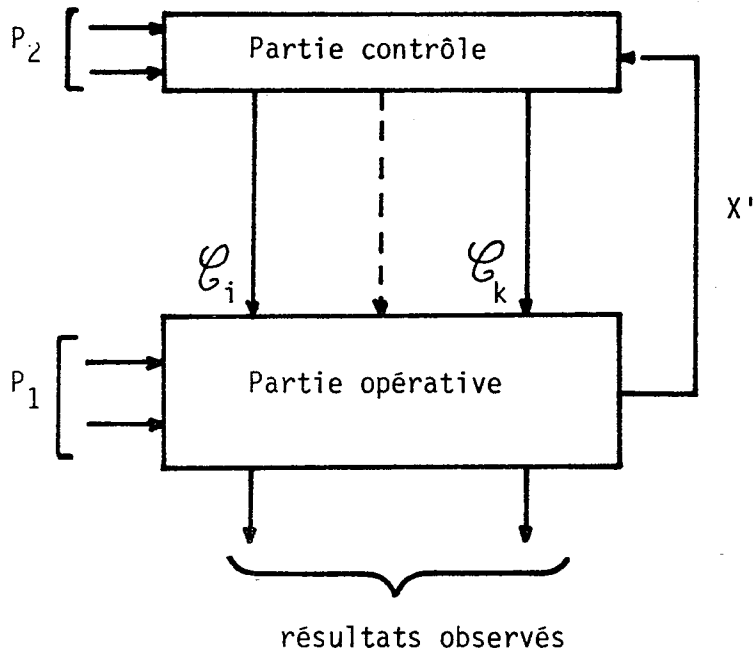


Fig. I.6.

la partie contrôle ne peut être testée qu'à travers la partie opérative et les paramètres d'entrée P_2 de la partie contrôle indiquent l'algorithme à exécuter (contraintes de déroulement de l'algorithme).

On prendra donc en compte, au moment du test, la signification fonctionnelle de ces paramètres d'entrée.

Dans ce cas, la prise en compte de l'aspect fonctionnel implique :

- . d'une part des contraintes sur les entrées P_2
- . d'autre part des contraintes sur la succession des états, dues au déroulement des algorithmes réalisés par cette partie contrôle.

$\left\{ \begin{array}{l} \text{étude fonctionnelle soit les fonctions } \{f_i\} \text{ réalisées par les} \\ \text{algorithmes} \\ + \\ \text{étude analytique} \end{array} \right.$

$\left. \begin{array}{l} \text{contraintes sur les entrées} \\ \text{contraintes sur la succession des} \\ \text{états} \end{array} \right\}$

$$T = \mathcal{F}(\{f_i(P_2), X'\}, P_1)$$

4.4 PRINCIPES D'UNE METHODE STRUCTURELLE.

Suivant le type de circuit considéré on adopte deux approches essentielles :

α) - on cherche des "activations fonctionnelles" globales, en mettant à jour, pas à pas, l'ensemble des pannes détectées, le balayage fonctionnel s'arrêtant lorsque l'ensemble des pannes considérées a été détecté. Les activations fonctionnelles sont choisies de façon à détecter, le plus rapidement possible, le plus grand nombre de pannes, ce choix se faisant à l'aide d'une analyse de la structure hardware du module.

Cette approche sera présentée en détail au Chapitre III.

β) - on part de l'étude des pannes à un niveau très fin, on étudie leurs manifestations fonctionnelles élémentaires et on en déduit les fonctions et les opérandes de test à envoyer pour détecter chacune de ces pannes; on remonte ensuite au niveau global en cherchant un ensemble de fonctions de test couvrant toutes les fonctions élémentaires. Cette approche est illustrée dans la méthode proposée pour les mémoires dynamiques à technologie MOS [18].

γ) Conclusion :

L'étude de base est l'étude des manifestations fonctionnelles des pannes fines.

Dans le premier cas, on part des fonctions du module avec analyse des manifestations de chaque panne. Dans le second cas, on part des manifestations fonctionnelles de chaque panne et on assure ensuite la commandabilité au niveau global.

De telles méthodes, et ces méthodes seulement, permettent d'atteindre, au niveau global, un taux de pannes détectées satisfaisant et une localisation précise des pannes.

CHAPITRE II

NIVEAU SYSTEME

* * *

I - PROGRAMME DE TEST OU DE MAINTENANCE.

La maintenance d'un ordinateur a pour objet :

- α) la vérification de la machine lors de la mise en route (test de conception et fin de fabrication)
- β) l'entretien régulier du système : test préventif avec arrêt du fonctionnement normal de la machine
- γ) le dépannage lors d'un fonctionnement défectueux : maintenance curative après un "message d'erreur"
- δ) une scrutation préventive de certaines parties du matériel sous-système c'est-à-dire ne nécessitant pas un arrêt de l'utilisation normale du système.

Le programme de maintenance d'un ordinateur consiste à vérifier la validité des différents organes de la machine ainsi que de leurs interconnexions. D'une manière générale, un programme ou micro-programme de test comporte un calcul répété en boucles, avec, entre chaque boucle, une séquence de comparaison permettant de vérifier que le résultat trouvé est correct.

Il peut porter, soit sur l'ensemble du système, soit, ce qui est préférable, sur des organes successifs. Les résultats du test sont ainsi d'autant plus faciles à analyser que le nombre d'éléments en jeu est réduit.

En outre, il ne faut pas oublier que l'un des buts essentiels de ces programmes de test est, sachant qu'une panne existe, de localiser le circuit en panne. Le programme de maintenance doit donc isoler, au maximum, l'élément fautif et faciliter l'observation de signaux caractéristiques à l'oscilloscope.

II – CARACTERISTIQUES GENERALES. [15], [27].

La définition d'un programme de test est fortement déterminée par son utilisation et son contexte. On peut distinguer 3 niveaux fondamentaux :

- Niveau conception : lorsqu'il s'agit du tout premier modèle d'un nouveau système. Ce test est caractérisé par les erreurs de conception et de fabrication, le mauvais fonctionnement des composants et les erreurs dans le programme de test lui-même. L'objectif d'un tel test est donc de vérifier la conception.

Des tests fonctionnels sont écrits pour déterminer si le système fonctionne en accord avec sa description ou ses spécifications fonctionnelles.

De tels tests ne sont en général pas sensibles à des changements dans le hardware du système et ceci est d'autant plus important que, dans cette phase du test, il y a un grand nombre de modifications.

En plus de ces tests fonctionnels, des tests aléatoires ou des tests de "pire cas" peuvent être développés.

- Niveau fin de fabrication : les exigences du test sont sensiblement les mêmes que pour un test de conception. Un tel test devra tenir compte d'erreurs multiples provenant de composants défectueux ou d'erreurs de fabrication.

- Niveau maintenance : les tests à ce niveau sont sensiblement différents et ce sont ceux auxquels on s'intéressera particulièrement. Un bon test de maintenance pourra en outre être utilisé au niveau fin de fabrication.

On fait l'hypothèse que le système est en condition de bon fonctionnement (système opérationnel) et que les pannes seront réparées dès qu'elles apparaissent. Par conséquent, on pourra admettre dans ce cas l'hypothèse de la panne simple.

III - ETUDE DES DIFFERENTES APPROCHES DE BASE.

On peut considérer 3 approches de base :

3.1 APPROCHE DE TYPE «START-BIG».

Une approche de ce type consiste en une vérification rapide du système avec des diagnostics plus sophistiqués au niveau du sous-système si nécessaire.

C'est en général une vérification fonctionnelle dont le but est de vérifier, dans un premier temps, que l'unité testée se comporte correctement.

Avantages et limites : dans des conditions de bon fonctionnement, cette méthode a l'avantage d'être rapide et, sous l'hypothèse de panne simple, elle fournit une localisation grossière et rapide (localisation au niveau sous-système A, B, ...).

Cependant elle peut devenir extrêmement complexe et présenter des aléas. De plus, elle n'est pas applicable dans le cas d'erreurs multiples.

Cette méthode est efficace lorsqu'on désire uniquement une vérification rapide et permet une pré-localisation rapide au niveau sous-système.

Un organigramme général de cette approche est le suivant, où A, B, C, ... représentent des sous-ensembles importants du système (découpage fonctionnel de haut niveau) :

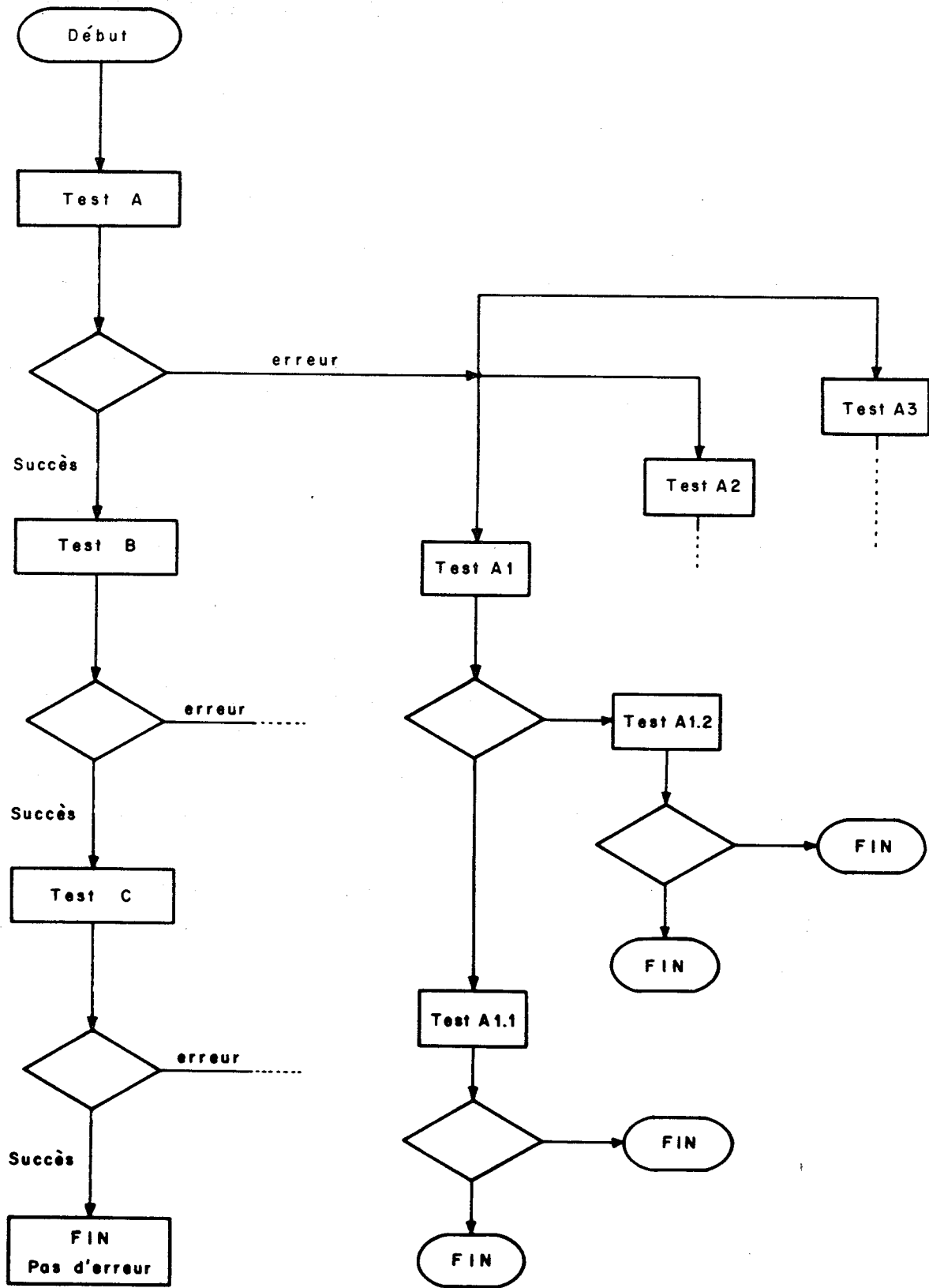


Fig. II.1.

Exemple de stratégie particulière à cette approche :

Dans l'histoire du test, cette stratégie est la première qui ait été utilisée.

L'approche générale était d'exécuter une instruction machine complexe (par exemple, MULTIPLIER) en utilisant des opérandes aléatoires. Les résultats étaient alors comparés avec ceux obtenus en utilisant une séquence équivalente d'instructions plus simples (par exemple ADDITIONNER et DECALER), mais avec les mêmes opérandes.

Si les résultats concordent, l'instruction complexe était considérée comme bonne, sinon comme défectueuse.

Mais un (ou plusieurs) ensembles d'opérandes aléatoires ou pseudo-aléatoires ne peuvent pas fournir un test complet pour un ensemble complexe tel qu'une U.A.L.

De plus, l'instruction sous test n'est presque jamais disjointe (du point de vue hardware) des instructions de la séquence équivalente.

Enfin, une non-concordance des résultats peut être due au mauvais fonctionnement d'une instruction de la séquence équivalente. Par conséquent la conclusion n'est pas nécessairement valable.

La difficulté avec ces tests est donc qu'ils sont généralement incomplets, et il n'est pas inhabituel pour de tels tests d'être limités à une détection de la moitié seulement des fautes possibles.

32 APPROCHE DE TYPE «START-SMALL». |3|, |44|.

Contrairement à l'approche précédente, c'est une approche séquentielle de "bootstrapping" : le premier test concerne la plus petite quantité de circuiterie possible et chaque test supplémentaire ajoute une petite partie de matériel à la partie précédemment testée.

Lorsqu'un test donné détecte une erreur, on fait l'hypothèse que le circuit défaillant est l'un des circuits ajoutés pour ce test et que la première erreur détectée est réparée avant de poursuivre les tests.

Avantages et limites : c'est une méthode dont l'élaboration est plus difficile que pour les précédentes approches mais elle présente l'avantage d'être efficace pour des erreurs multiples.

Cette approche est particulièrement bien adaptée pour un programme de maintenance et convient très bien pour détecter les pannes multiples à l'encontre des méthodes précédentes. Ce dernier point permet donc d'utiliser cette approche pour un test de fabrication.

Cette approche de test est de première importance dans les micro-diagnostics : en utilisant cette technique il est seulement nécessaire de comparer un résultat de test avec un résultat généré par le hardware déjà testé.

Un organigramme général de cette approche est le suivant :

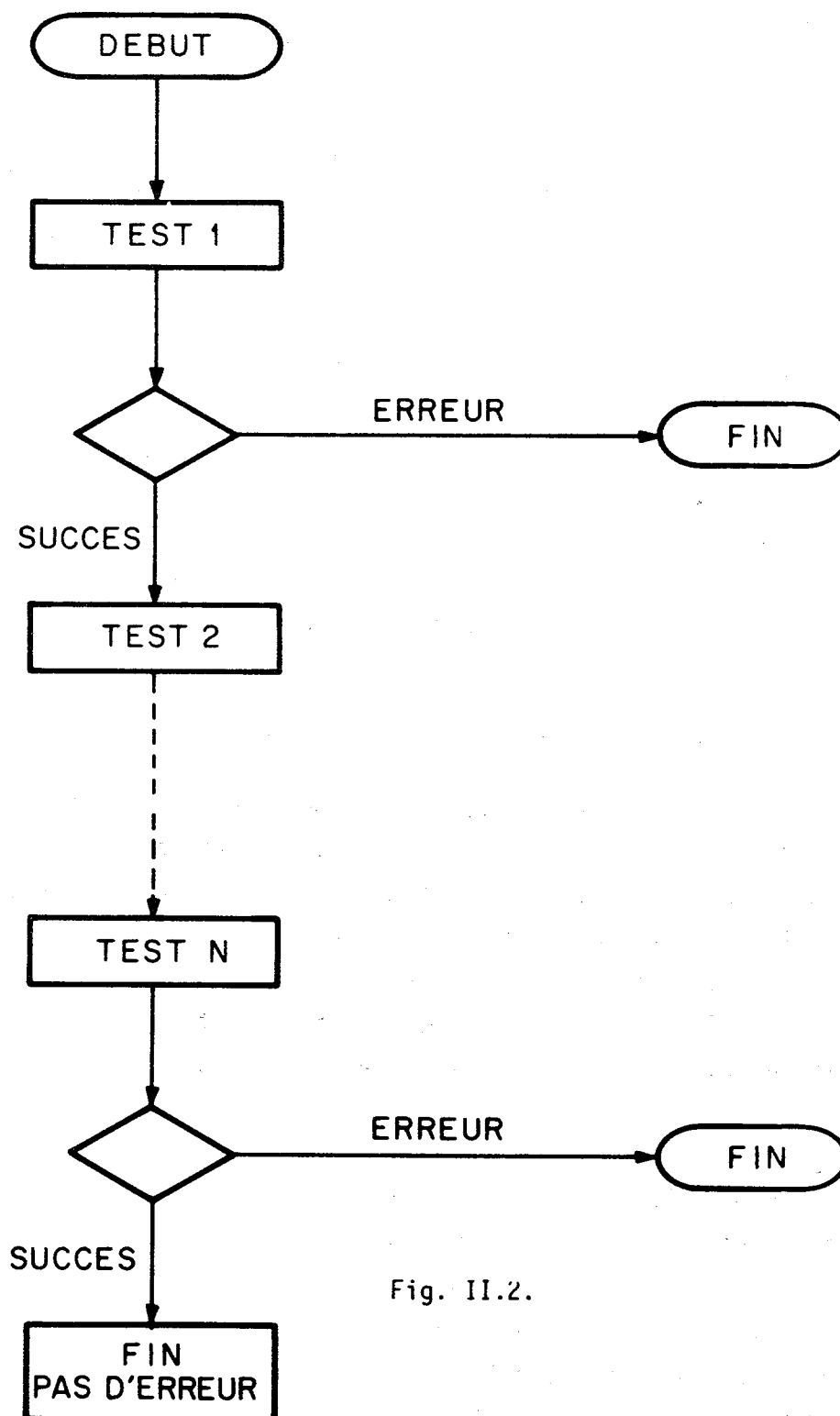


Fig. II.2.

Remarque : on pourra prévoir, sur les branches d'erreur, des arbres de localisation plus précise.

3.3 APPROCHE DE TYPE «MULTIPLE-CLUE».

Cette approche est du type précédent mais elle base son diagnostic sur l'analyse d'une série de résultats de test individuels : ce test est dit "combinatoire" puisque son déroulement ne dépend pas de la présence d'une erreur.

On active tous les tests, l'information d'erreur est enregistrée et le diagnostic se termine par l'analyse globale des informations d'erreu

Avantages et limites : sous l'hypothèse de panne simple, cette analyse est efficace. Mais en cas d'erreurs multiples, l'analyse devient extrêmement complexe si ce n'est impossible.

Une approche de ce type est utilisable pour un test de maintenance et peut servir à donner un meilleur diagnostic dans certaines parties du système.

Un organigramme général de cette approche est le suivant :

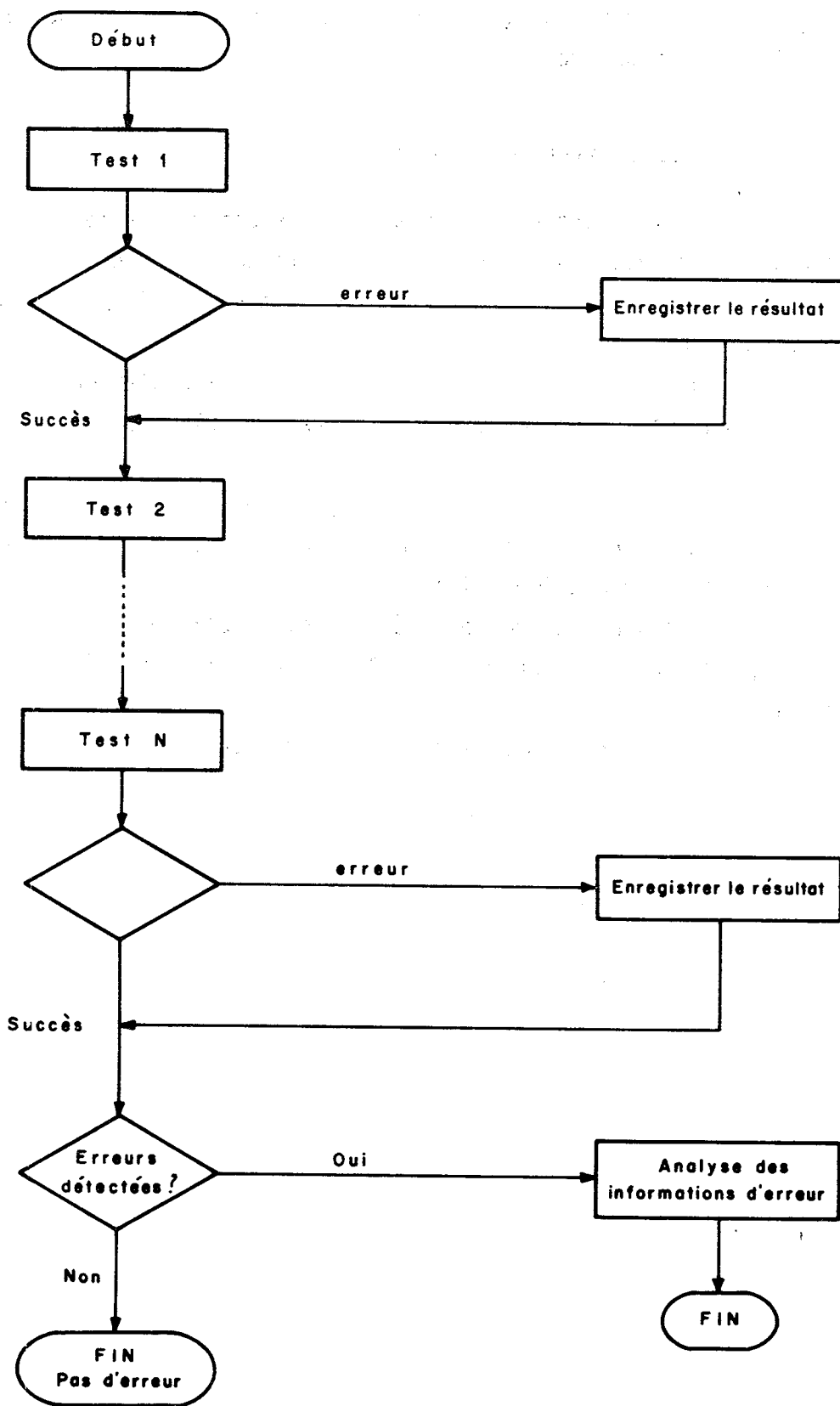


Fig. II.3.

3.4 CONCLUSION.

On peut donner le tableau récapitulatif suivant

	stratégie applicable		
	start-small	multiple clue	start-big
test de conception et de fabrication	X		
test de maintenance	X	X	
rapide vérification			X

Il apparaît clairement qu'une approche de type "start-small" est la plus adéquate pour le test d'un système par son application à un test de fabrication aussi bien que de maintenance : bien que d'élaboration plus difficile, elle est la plus efficace.

IV – CARACTERISTIQUES D'UN PROGRAMME DE TEST EFFICACE.

Indépendamment de l'approche choisie, il est important qu'un programme de test réponde aux critères suivants :

α) *test automatique* : on essaie d'éviter des manipulations humaines et un arrêt de la machine, en maintenance normale; c'est le problème d'initialisation du test.

β) *efficacité par rapport au taux de pannes* : ce critère nous conduit à proposer une partition fine en modules de base et à définir des méthodes spécifiques et originales au niveau de ces modules.

Les méthodes proposées devront avoir les mêmes performances (taux de pannes détectées; localisation) que les méthodes analytiques classiques.

On définira à cet effet deux niveaux de test : niveau macro-bloc, niveau micro-bloc.

γ) *localisation* : on cherche un ordonnancement de ces modules de base, vis-à-vis du test, permettant d'utiliser une approche de type "start-small" : un module donné ne sera testé qu'à travers des modules déjà testés (accès de l'information de test à ce module ou observation des résultats de test du module).

δ) *test, coût et rapidité* : l'ensemble des vecteurs de test devra être soigneusement élaboré (minimisation du nombre de vecteurs) de manière à réduire le problème de stockage des vecteurs de test et le temps d'exécution du programme de test.

On peut noter que, la qualité du test au niveau système dépendant des méthodes choisies au niveau sous-système, on choisira généralement des méthodes analytiques au niveau des modules définis. En effet, étant donnée la complexité des ensembles que l'on doit tester, il est nécessaire de profiter de l'aide que peut apporter la structure du système : si le temps d'élaboration de telles méthodes est important, le résultat par contre est performant.

V - METHODOLOGIE GENERALE DE TEST.

On rappelle que l'approche de base choisie est une approche de type "start-small".

5.1 PROBLEME D'INITIALISATION.

Rappelons que dans une approche de type "start-small", se pose le problème d'initialisation du test : où seront rangées les données du test et par quel moyen ? Comment déterminer l'ensemble minimal de circuits permettant de réaliser un premier test : en d'autres termes, qu'est-ce que le hardcore ?

On peut distinguer essentiellement 2 situations :

- α) au cours de la conception, le problème du test a été intégré : adjonction de matériel réduisant ou supprimant une intervention manuelle au niveau de l'initialisation du test.
- β) ce problème n'a pas été décisif au moment de la conception et l'initialisation requiert un arrêt et des manipulations du matériel.

. Le point α) est parfaitement illustré dans la technique de maintenance de la série IBM 360 (360/50). [11], [26].

Le hardcore, environ 10% du hardware supplémentaire, a été conçu à des fins de test uniquement et représente la fraction du processeur qui doit être opérationnelle pour dérouler les tests.

Ce hardware est prévu pour fournir une extrême facilité:

- pour remettre le processeur dans un état spécifié : initialisation (fonction Scan-In).
- pour mémoriser cet état (fonction Scan-Out).

L'exécution d'un "Scan-Out" fait que l'état des éléments de mémoire du processeur est mémorisé dans une zone spéciale de la mémoire centrale.

Le "Scan-In" est la fonction inverse : les données trouvées dans la zone spéciale de mémoire sont utilisées pour forcer le processeur dans un état donné.

Ce hardware est également prévu pour permettre au processeur de fonctionner par petits incréments : cela permet d'exécuter une petite partie d'un cycle instruction et ensuite d'analyser des résultats intermédiaires. On obtient ainsi un meilleur diagnostic que si l'instruction devait s'exécuter entièrement avant de pouvoir analyser les résultats.

Ainsi, par cette voie d'accès particulière, les programmes et les vecteurs de test d'un premier sous-ensemble sont chargés, ceci indépendamment de la mémoire morte; après le test des circuits de comparaison élémentaires, le test de la mémoire morte est initialisé de façon à ce qu'elle prenne le contrôle ultérieur.

. Nous nous plaçons ici dans le cas le plus défavorable (cas β) où il n'y a pas de facilité de test. [13].

L'exemple pratique présenté est un ordinateur microprogrammé de moyenne importance, sans adjonction prévue de matériel, malgré de grandes exigences de testabilité. [2].

5.1.1 Précisions sur la structure du hardware.

Dans une approche de ce type on distinguera deux sortes de hardware :

- α) hardware de premier degré :

Il s'agit du matériel spécialement mis en place au moment du test (intervention manuelle) et considéré par hypothèse comme bon (vérification spéciale préalable).

Cet ensemble, dans cette approche, comportera au minimum la mémoire morte où seront rangés les microprogrammes de test. Dans certains cas, si l'architecture le permet, on peut augmenter la rigueur et la finesse de la localisation en incluant dans ce hardware de 1er degré les circuits de comparaison (cas du Géoprocasseur : remplacement d'une UAL par une carte comparateur).

- β) hardware de deuxième degré :

Il s'agit d'un ensemble minimal du matériel usuel de l'ordinateur

réalisant à partir du hardcore de 1er degré les fonctions nécessaires à l'initialisation du test, soit :

- chargement et stockage des "premiers" vecteurs de test et des "premiers" résultats obtenus, si cette fonction n'est pas réalisée par le hardcore de 1er degré (par exemple l'ensemble minimal permettant d'élaborer des opérandes de test à partir du caractère valeur immédiate des micro-instructions, dans le cas d'une machine microprogrammée).
- comparaison des résultats attendus avec les résultats obtenus.
- observation des résultats (par exemple les circuits de visualisation).

5.1.2 Problème du test du hardcore de deuxième degré.

Si le premier objectif est toujours la rigueur du test (détection et localisation), ce hardcore de 2ème degré est testé en fonctionnement pas à pas avec vérification humaine (cas du Géoprocasseur).

Si l'on veut éviter ce fonctionnement pas à pas, on teste alors ce hardcore de 2ème degré, automatiquement, à travers un ensemble plus grand du matériel; en cas de défaillance seulement, la localisation étant impossible et le hardcore étant l'élément initial pour le test, on retourne alors à un fonctionnement en pas à pas pour le tester (cas du T 1600, Télé-mécanique). [46].

En conclusion, dans une approche start-small, l'initialisation requiert soit une implémentation spéciale du matériel (test automatique), soit une intervention manuelle (test non automatique).

5.2 NIVEAU MACRO-BLOC.

On rappelle qu'on définit deux niveaux de test : niveau macro-bloc, niveau micro-bloc.

5.2.1 Définition.

Un macro-bloc est défini par sa fonction : il s'agit d'un découpage fonctionnel de haut niveau.

Dans un ordinateur les macro-blocs seront des UAL, des mémoires, ...; dans une unité centrale ce découpage est classique et généralement fourni par les plans logiques du constructeur.

5.2.2 Ordre de test au niveau macro-bloc.

Une fois le hardware supposé bon, cet ordre est donné par les critères suivants :

- α) contraintes d'acheminement et de stockage des informations de test
Par exemple : - liaison périphérique (données venant de l'extérieur).
- mémoire centrale (stockage de données en grand nombre).
- registres de travail (stockage de résultats de test intermédiaires).
- β) contraintes de puissance de calcul
Par exemple : - UAL (traitement des données).
- γ) les macro-blocs restants sont testés dans un ordre indéterminé ou selon des critères plus spécifiques et non vitaux.

En définitive, une proposition d'ordonnement des macro-blocs vis-à-vis du test serait la suivante :

- 1) test liaison périphérique (un seul fonctionnement : périphérique → unité centrale, sélection d'un seul périphérique).
- 2) test mémoire centrale.
- 3) test mémoire locale (mémoire rapide composée de registres).
- 4) test d'une UAL au moins.
- 5) test du reste de la machine dans un ordre non défini ici.

5.3 NIVEAU MICRO-BLOC.

5.3.1 Partition d'un macro-bloc en micro-blocs.

Le micro-bloc constitue :

α) une entité matérielle : soit un sous-ensemble physique défini sur les plans logiques du système.

β) une entité fonctionnelle : l'entité matérielle définie en α) possède une fonction, à un niveau élémentaire, que l'on peut définir.

γ) surtout, une entité vis-à-vis du test : à ce sous-ensemble sera associé -préalablement ou non- un ensemble de vecteurs de tests, par une méthode bien spécifique à ce micro-bloc et indépendamment des méthodes choisies pour les autres micro-blocs.

En effet, il sera nécessaire de tenir compte, à un niveau très fin, de la structure du micro-bloc et de choisir ou d'élaborer les méthodes les plus adéquates. Ces méthodes seront choisies dans l'éventail de celles proposées au Chapitre I.

Dans certains cas, la partition pourra être triviale, c'est-à-dire qu'il n'y aura pas de découpage à un niveau fin et la méthode de test sera choisie directement au niveau macro-bloc (Exemple : bloc mémoire).

Remarque 1 : on distinguera essentiellement deux sortes de micro-blocs : les micro-blocs de traitement (additionneur, registres) et les micro-blocs de commande (séquenceur local).

Remarque 2 : pour une unité centrale, une partition en micro-blocs peut être obtenue à partir d'un langage de description de type CASSANDRE et dans le meilleur cas, elle est fournie par le concepteur.

Exemple : - *Découpage*

Un macro-bloc sera par exemple une UAL; à l'intérieur de ce macro-bloc les micro-blocs seront des registres, un additionneur (micro-blocs de traitement), un séquenceur de multiplication, une partie contrôle locale (micro-blocs de commande).

- *méthodes spécifiques*

Les méthodes de test dont les caractéristiques générales ont été données au Chapitre I sont spécifiques à chaque micro-bloc.

- méthodes de propagation classiques (registres).
- méthodes de propagation prenant en compte les propriétés de symétrie et de répétitivité (additionneur). [66]

- méthode de vérification d'automates (séquenceur de multiplication) [53]
- et surtout, méthode spécifique combinant une approche fonctionnelle et une étude analytique, dont les caractéristiques générales ont été données au chap. I. § IV. et dont l'étude de détail est présentée au Chapitre III (partie contrôle locale) [49].

5.3.2 Analyse du chemin de données. [33], [55]

On rappelle que l'objectif est de déterminer un ordre de test dans l'ensemble des micro-blocs de telle sorte que le test d'un micro-bloc se fasse en traversant uniquement des micro-blocs déjà testés : ceci, pour l'entrée des informations de test à partir de l'entrée du macro-bloc aussi bien que pour les résultats du test à la sortie du macro-bloc.

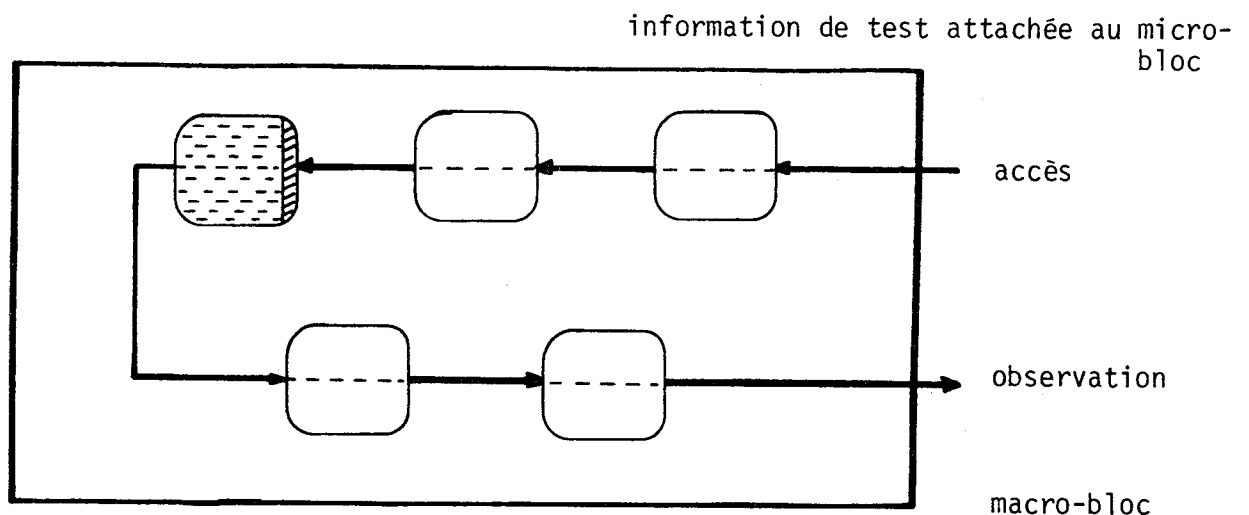


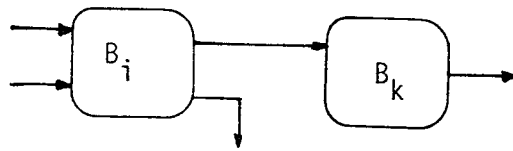
Fig. II.4.

A cet effet, on analyse préalablement le cheminement des informations et la structure des commandes. Pour cela, on distingue 2 étapes :

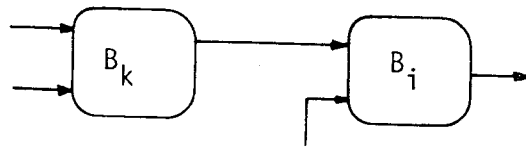
- analyse du chemin de données (micro-blocs de traitement)
- analyse des niveaux de commande (micro-blocs de commande).

Remarque :

On notera que :



si B_k n'est atteint qu'à partir de B_i alors B_k doit être testé après B_i .



si B_k n'est observable qu'à travers B_i alors B_k doit être testé après B_i .

a) Analyse du chemin de données.

Soit E l'ensemble des micro-blocs considérés :

$$E = \{B_{ij}\}$$

Soit $\mathcal{P}(E)$ l'ensemble des parties de E :

$$\mathcal{P}(E) = \{B_{i1}, B_{i2}, \dots, B_{ik}; B_{ij} \in E\}$$

Pour représenter des éléments de $\mathcal{P}(\mathcal{P}(E))$ on adopte la notation suivante :

. un élément de $\mathcal{P}(E) : \{B_{i1}, B_{i2}, \dots, B_{ik}\}$ sera représenté comme le monôme $B_{i1} \cdot B_{i2} \cdot \dots \cdot B_{ik}$

. un ensemble d'éléments de $\mathcal{P}(E)$ c'est-à-dire un élément de $\mathcal{P}(\mathcal{P}(E))$ sera représenté comme une somme de monômes :

$$\{B_{i1} \cdot B_{i2} \cdot \dots \cdot B_{ik}\} \cup \{B_{j1} \cdot B_{j2} \cdot \dots \cdot B_{jm}\}$$

et sera noté comme suit :

$$\alpha = B_{i1} \cdot B_{i2} \cdot \dots \cdot B_{ik} + B_{j1} \cdot B_{j2} \cdot \dots \cdot B_{jm}$$

On peut donc représenter tout élément de $\mathcal{P}(\mathcal{P}(E))$ comme une somme de monômes sur E :

$$\alpha = \sum_i \prod_j B_{ij}$$

On définit deux relations binaires internes sur $\mathcal{P}(\mathcal{P}(E))$ désignées par ρ_1 et ρ_2 .

Définition de ρ_1

ρ_1 ou relation de commandabilité définit l'accès de l'information à un micro-bloc B_k :

$$\text{soit } B_k \rho_1 (B_{11} \cdot \dots \cdot B_{1n} + \dots + B_{j1} \cdot \dots \cdot B_{jm})$$

$$B_k \rho_1 \alpha \text{ où } B_k \in E, \alpha \in \mathcal{P}(\mathcal{P}(E))$$

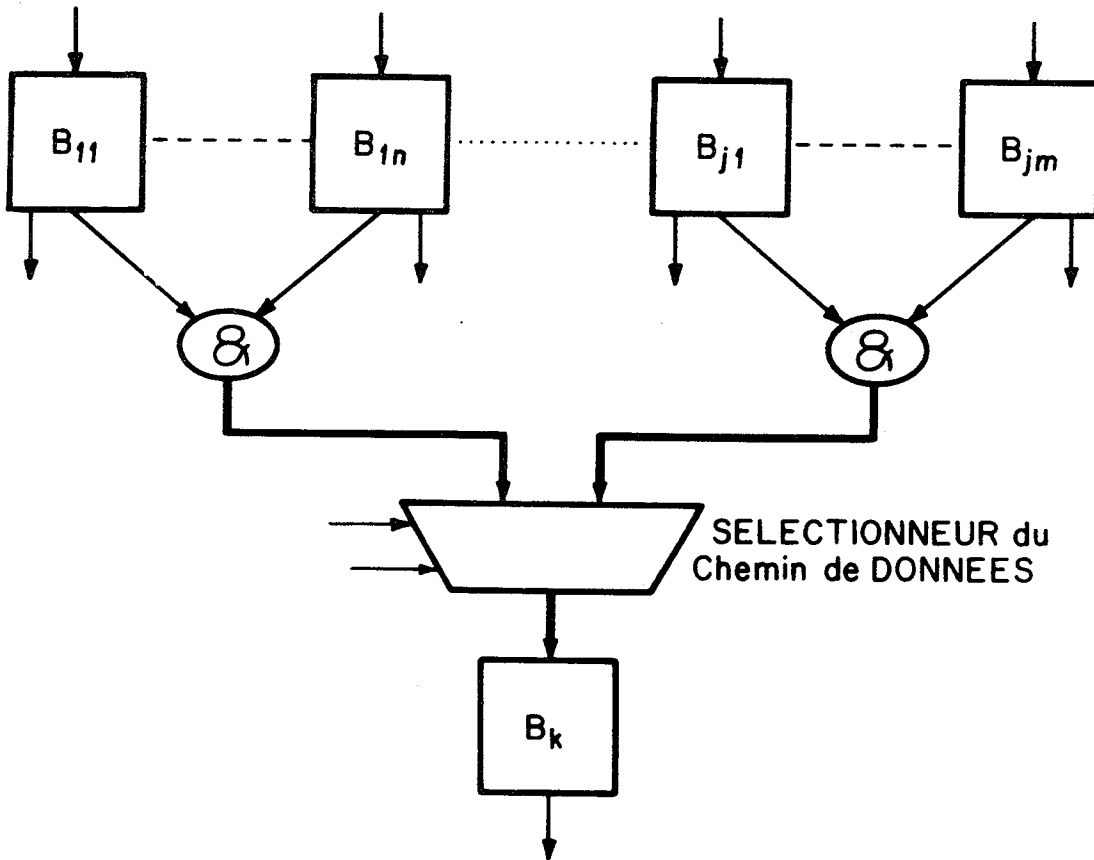


Fig. II.5.

l'entrée informationnelle de B_k est

- . soit la sortie de B_{11} et ... et B_{1n}
- . soit ...
- . soit la sortie de B_{j1} et ... et B_{jm}

Interprétation de ρ_1

$B_k \rho_1 (B_{11} \cdot \dots \cdot B_{1n} + \dots + B_{j1} \cdot \dots \cdot B_{jm})$
signifie que

- le test de B_k vient après - soit le test de B_{11} et ... et B_{1n}
 - soit ...
 - soit le test de B_{j1} et ... et B_{jm}

Définition de ρ_2

ρ_2 ou relation d'observabilité définit l'observation de l'information issue d'un micro-bloc B_k :

$$B_k \rho_2 (B_{11} \cdot \dots \cdot B_{1n} + \dots + B_{j1} \cdot \dots \cdot B_{jm})$$

soit $B_k \rho_2 B$ où $B_k \in E, B \in \mathcal{P}(\mathcal{P}(E))$

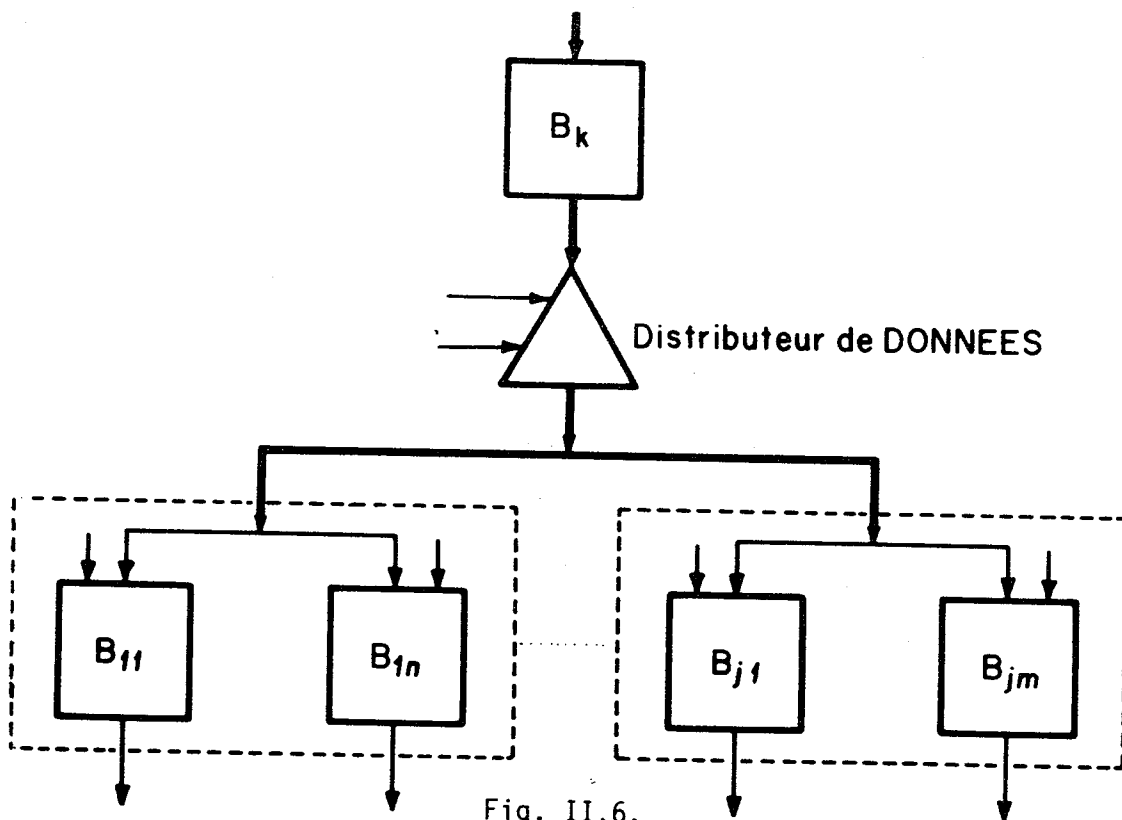


Fig. II.6.

La sortie informationnelle de B_k est :

- . soit l'entrée de B_{11} et ... et B_{1n}
- . soit ...
- . soit l'entrée de B_{j1} et ... et B_{jm}

Interprétation de ρ_2

$$B_k \rho_2 (B_{11} \cdot \dots \cdot B_{1n} + \dots + B_{j1} \cdot \dots \cdot B_{jm})$$

signifie que :

le test de B_k vient après - soit le test de B_{11} et ... et B_{1n}

- soit ...
- soit le test de B_{j1} et ... et B_{jm}

b) Définition du préordre de priorité \succcurlyeq .

Tout micro-bloc B_k est associé à 2 relations :

- . une relation ρ_1 entre ce micro-bloc B_k et ses antécédents directs : $B_k \rho_1 \alpha$
- . une relation ρ_2 entre ce micro-bloc B_k et ses descendants directs : $B_k \rho_2 \beta$

On combinera ces deux relations ρ_1 et ρ_2 pour obtenir une seule relation \succcurlyeq telle que :

$$B_k \succcurlyeq \alpha \cdot \beta$$

Interprétation de \succcurlyeq (ou préordre de priorité)

$$\alpha \succcurlyeq \beta : \text{le test de } \alpha \text{ ne peut pas venir avant le test de } \beta$$

$$\text{si } \beta = B_{11} \cdot \dots \cdot B_{1n} + \dots + B_{j1} \cdot \dots \cdot B_{jm}$$

$$\text{test de } \beta = \text{test de } B_{11} \text{ et } \dots \text{ et } B_{1n} \text{ ou test de } B_{j1} \text{ et } \dots \text{ et } B_{jm}$$

Propriétés de la relation \succcurlyeq

. La relation \succcurlyeq est bien un préordre qui vérifie en outre les axiomes suivants :

$$\text{Axiome 1 : } \begin{array}{l} \alpha \succcurlyeq \beta \\ \beta \succcurlyeq \gamma \end{array} \Bigg| \longrightarrow \alpha \succcurlyeq \beta \cdot \gamma$$

$$\text{Axiome 2 : } \alpha \succcurlyeq \alpha \cdot \gamma + \beta \longrightarrow \alpha \succcurlyeq \gamma + \beta$$

$$\text{Axiome 3 : } \alpha \succcurlyeq \beta \longrightarrow \alpha \cdot \gamma \succcurlyeq \beta \cdot \gamma$$

Les axiomes 2 et 3 impliquent la propriété suivante :

Propriété 1 :

$$\begin{array}{l} \alpha \succcurlyeq \beta \cdot \gamma \\ \beta \succcurlyeq \alpha \cdot \delta \end{array} \Bigg| \longrightarrow \alpha \cdot \beta \succcurlyeq \gamma \cdot \delta$$

Loi d'absorption : ce préordre de priorité vérifie la loi d'absorption :

$$\alpha \succcurlyeq \beta + \beta \cdot \gamma \implies \alpha \succcurlyeq \beta$$

Remarque : on peut noter qu'il y a compatibilité entre la relation de préordre \succcurlyeq et l'opération \bullet .

Définition de blocs d'indiscernabilité :

$$\begin{array}{l} \alpha \succcurlyeq \beta \cdot \gamma \implies \alpha \succcurlyeq \beta \\ \beta \succcurlyeq \alpha \cdot \delta \implies \beta \succcurlyeq \alpha \end{array} \Bigg| \implies \alpha \equiv \beta$$

on dira que α et β sont indiscernables et on notera $\alpha \equiv \beta$; $\varepsilon = (\alpha.\beta)$ constituera un bloc d'indiscernabilité : α et β seront indiscernables vis-à-vis de l'ordre de test et la localisation, impossible au niveau de α ou de β , se fera au niveau de ε .

Dans toute inéquation α et β seront remplacés par ε .

De plus,

$$\begin{array}{l} \alpha \succcurlyeq \beta \cdot \gamma \\ \beta \succcurlyeq \alpha \cdot \delta \end{array} \Bigg| \implies \begin{array}{l} \alpha \cdot \beta \succcurlyeq \gamma \cdot \delta \\ \alpha \equiv \beta \\ (\alpha \cdot \beta) = \varepsilon \end{array} \Bigg| \implies \varepsilon \succcurlyeq \gamma \cdot \delta$$

c) Système d'inéquations obtenu.

Soit n le nombre de micro-blocs considérés.

On obtient ainsi un système de n inéquations : pour chaque micro-bloc B_k on a une relation de préordre entre ce micro-bloc et ses antécédents et descendants directs.

$$B_k \succcurlyeq \alpha \quad \text{où } B_k \in E, \alpha \in \mathcal{P}(\mathcal{P}(E))$$

La résolution de ce système donnera un ordre de test sur l'ensemble des micro-blocs c'est-à-dire sur E ou $\mathcal{P}(E)$ dans le cas d'existence de blocs d'indiscernabilité.

d) Recherche d'un ordonnancement.

On cherche une partition des micro-blocs en couches ordonnées $\mathcal{C}_0, \dots, \mathcal{C}_i$ telles que tout micro-bloc d'une couche \mathcal{C}_k n'est testé qu'à travers un ensemble des micro-blocs des couches $\mathcal{C}_j : \{\mathcal{C}_j; j < k\}$ dont l'un au moins appartient à \mathcal{C}_{k-1}

$$B_i \in \mathcal{E}_k \iff \left. \begin{array}{l} \cdot B_k \text{ est supérieur à un élément au moins de } \mathcal{E}_{k-1} \\ \cdot \text{ tous les éléments inférieurs à } B_k \text{ sont éléments} \\ \text{de } \mathcal{E}_0, \dots, \mathcal{E}_{k-1} \end{array} \right\}$$

On définit l'ensemble \mathcal{B}_0^i comme l'ensemble des micro-blocs déjà ordonnés : les éléments de cet ensemble notés $B_{01}^i \dots B_{0k}^i$ ont les propriétés suivantes :

$$\cdot \alpha \succ_{\mathcal{B}_0^i} \beta \implies \alpha \succ \beta$$

B_{0j}^i est neutre par rapport à la multiplication.

$$\cdot \alpha \succ_{\mathcal{B}_0^i} \beta + \gamma \implies \alpha \succ_{\mathcal{B}_0^i} \beta$$

B_{0j}^i est absorbant par rapport à l'addition.

Au départ l'ensemble \mathcal{B}_0 est le hardcore (l'information est directement accessible ou observable à l'extérieur du macro-bloc).

e) **Algorithme de recherche d'un ordonnancement.**

A chaque pas k de l'algorithme on crée une couche \mathcal{E}_k telle que les micro-blocs de \mathcal{E}_k soient testés après les micro-blocs des couches $\mathcal{E}_1, \mathcal{E}_2 \dots \mathcal{E}_{k-1}$.

Considérons le pas i de l'algorithme : à ce stade on a défini les couches $\mathcal{E}_1 \dots \mathcal{E}_{i-1}$ et \mathcal{B}_0^i est l'ensemble des micro-blocs déjà ordonnés

$$\mathcal{B}_0^i = \{\text{micro-blocs des couches } \mathcal{E}_1, \dots, \mathcal{E}_{i-1}\}$$

• (i') les inéquations du système sont simplifiées selon les propriétés du préordre \succ .

• (i'') on cherche toute inéquation de la forme $B_k \succ_{\mathcal{B}_0^i} B_{0j}^i$ où $B_{0j}^i \in \mathcal{B}_0^i$

Tout micro-bloc B_k vérifiant une telle inéquation est classé dans la couche \mathcal{E}_i .

On remet alors à jour l'ensemble des éléments déjà ordonnés :

$$\mathcal{B}_0^{i+1} = \mathcal{B}_0^i \cup \{B_k \succ_{\mathcal{B}_0^i} B_{0j}^i\}$$

On passe ensuite au pas (i+1) de l'algorithme en considérant les inéquations des micro-blocs non encore ordonnés.

S'il n'existe pas d'inéquation $B_k \succ B_{0j}^i$, on passe en $\textcircled{i''''}$

$\textcircled{i''''}$ toute inéquation est donc du type :

$B_k \succ \alpha$ où les éléments de α n'appartiennent pas à \mathcal{B}_0^i

Il y aura au moins un bloc d'indiscernabilité.

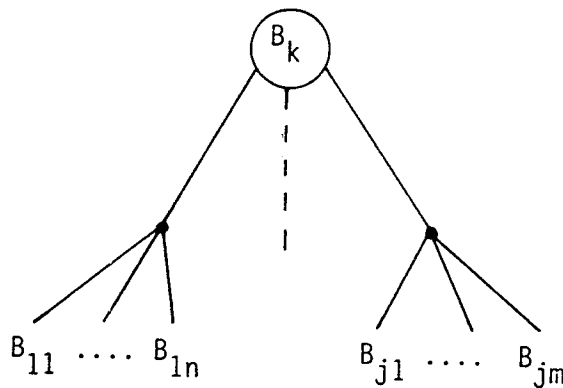
$\boxed{1}$ On construit un arbre d'indiscernabilité comme suit :

. on choisit une inéquation du système considéré :

$$B_k \succ B_{11} \cdot \dots \cdot B_{1n} + \dots + B_{j1} \cdot \dots \cdot B_{jm}$$

B_k est la racine de l'arbre

$B_{11}, \dots, B_{1n}, \dots, B_{j1}, \dots, B_{jm}$, sont les successeurs de B_k .



. on considère les inéquations relatives aux successeurs de B_k et on construit leurs successeurs de la même manière.

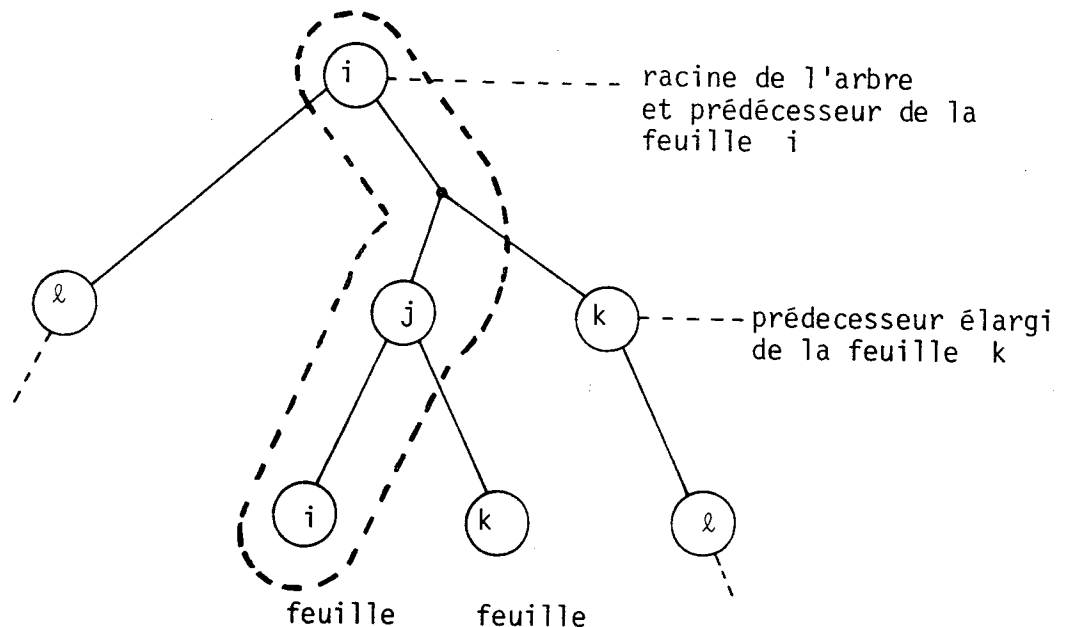
. un élément de cet arbre sera une feuille

- s'il est prédecesseur dans l'arbre : cet élément définit alors un bloc d'indiscernabilité; le bloc d'indiscernabilité créé sera constitué par cet élément et l'ensemble des éléments qui lui sont à la fois supérieurs et inférieurs.

- s'il est "prédécesseur élargi".

Définition : on appellera un élément : "prédécesseur élargi" s'il est prédécesseur dans une branche associée à l'opération de multiplication.

Exemple :



- les éléments \textcircled{i} et \textcircled{k} sont des feuilles
- la feuille \textcircled{i} définit le bloc d'indiscernabilité (i,j)

2 On note tous les blocs d'indiscernabilité d'ordre minimum c'est-à-dire contenant le nombre minimum d'éléments.

Pour chacun de ces blocs d'indiscernabilité BI_i on écrit le nouveau système d'inéquations obtenu.

. si pour l'un de ces systèmes, il n'existe pas d'inéquations du type $BI_i \succ \alpha$

où α est un polynôme des éléments non encore ordonnés alors BI_i est classé dans la couche \mathcal{C}_i et on met à jour l'ensemble des éléments déjà ordonnés

$$\mathcal{B}_0^{i+1} = \mathcal{B}_0^i \cup BI_i$$

On passe au pas $(i+1)$ de l'algorithme.

. sinon, c'est qu'il existe un bloc d'indiscernabilité plus grand.

On choisit l'un des blocs d'indiscernabilité et son système d'inéquations associé et on passe au pas (i''') de l'algorithme.

L'algorithme s'arrête lorsque tous les micro-blocs ont été ordonnés.

f) Exemple.

Soit le système d'inéquations, obtenu après simplification.

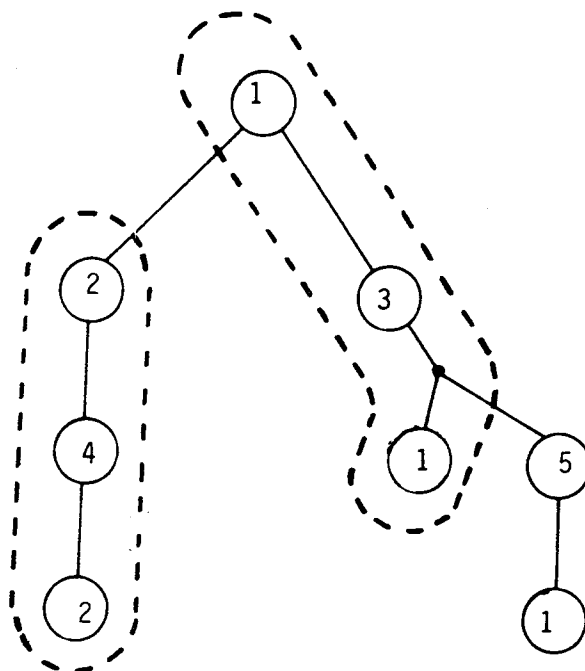
$$\left\{ \begin{array}{l} 1 \succcurlyeq 2+3 \\ 2 \succcurlyeq 4 \\ 3 \succcurlyeq 1.5 \\ 4 \succcurlyeq 2 \\ 5 \succcurlyeq 1 \\ 6 \succcurlyeq B_0 \end{array} \right. \quad \text{où } \mathcal{B}_0^1 = \{B_0\}$$

Pas 1 : $(1'')$ $6 \succcurlyeq B_0 \rightarrow$ l'élément 6 est classé dans la couche \mathcal{C}_1
et $\mathcal{B}_0^2 = \{B_0, 6\}$

Pas 2 : $(2')$ pas de simplifications

$(2'')$ il n'existe pas d'inéquation du type
 $B_k \succcurlyeq$ un élément de \mathcal{B}_0^2

$(2''')$ \square on construit l'arbre d'indiscernabilité ayant comme racine l'élément 1 :



② Les blocs d'indiscernabilité d'ordre minimum sont

$$\cdot \alpha = (1.3)$$

$$\cdot \beta = (2.4)$$

Le système d'inéquations devient :

Choix de $\alpha = (1.3.)$,	Choix de $\beta (2.4.)$
$\left\{ \begin{array}{l} \alpha \succcurlyeq 2+\alpha \\ 2 \succcurlyeq 4 \\ 4 \succcurlyeq 2 \\ \alpha \succcurlyeq \alpha.5 \\ 5 \succcurlyeq \alpha \end{array} \right.$	\vdots	$\left\{ \begin{array}{l} 1 \succcurlyeq \beta+3 \\ \beta \succcurlyeq \beta \\ 3 \succcurlyeq 1.5 \\ \beta \succcurlyeq \beta \\ 5 \succcurlyeq 1 \end{array} \right.$

Soit après simplification

$\left\{ \begin{array}{l} \alpha \succcurlyeq 2.5 \\ 5 \succcurlyeq \alpha \\ 2 \succcurlyeq 4 \\ 4 \succcurlyeq 2 \end{array} \right.$,	$\left\{ \begin{array}{l} 1 \succcurlyeq \beta+3 \\ 3 \succcurlyeq 1.5 \\ 5 \succcurlyeq 1 \end{array} \right.$
---	---	---

On remarque, dans le système de droite qu'il n'existe pas d'inéquations du type $\beta \succcurlyeq \dots$

$\beta = (2.4)$ est classé dans la couche \mathcal{C}_2

$$\text{et } \mathcal{B}_0^3 = \{B_0, 6, 2, 4\} = \{B_0, B_{01}, B_{02}, B_{03}\}$$

Pas 3 : le système d'inéquations est le suivant (après simplifications) :

$$\left\{ \begin{array}{l} 1 \succcurlyeq B_{02} \cdot B_{03} \\ 3 \succcurlyeq 1.5 \\ 5 \succcurlyeq 1 \end{array} \right.$$

$\mathcal{B}_0^4 = \{B_0, 6, 2, 4, 1\} = \{B_0, B_{01}, \dots, B_{04}\}$ l'élément 1 est classé dans la couche \mathcal{C}_3 et

Pas 4 : le système d'inéquations obtenu après simplifications est le suivant :

$$\left\{ \begin{array}{l} 3 \succcurlyeq 5 \\ 5 \succcurlyeq B_{04} \end{array} \right.$$

\mathcal{C}_4 et $\mathcal{B}_0^5 = \{B_0, 6, 2, 4, 1, 5\} = \{B_0, B_{01} \dots B_{05}\}$
 4" 5 $\succ B_{04} \rightarrow$ l'élément 5 est classé dans la couche

Pas 5 le dernier élément est l'élément 3 qui sera classé dans la couche \mathcal{C}_5 .

L'ordre de test est donc le suivant :

- . $6 \in \mathcal{C}_1$
- . $2, 4 \in \mathcal{C}_2$
- . $1 \in \mathcal{C}_3$
- . $5 \in \mathcal{C}_4$
- . $3 \in \mathcal{C}_5$

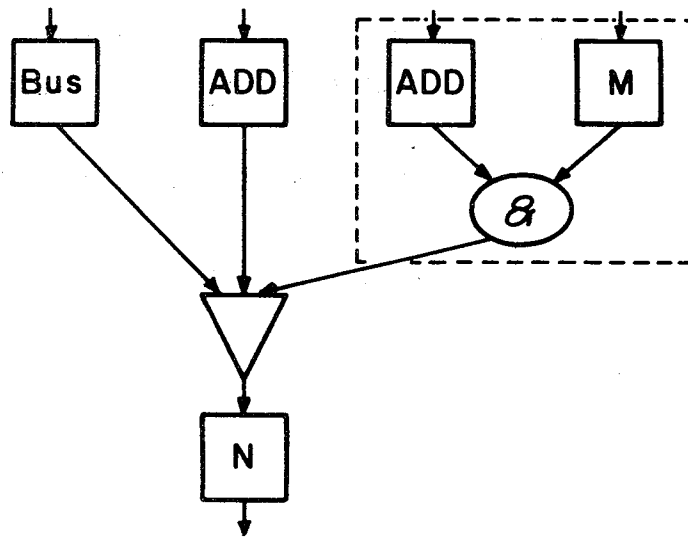
g) Exemple pratique.

Nous choisirons comme exemple l'UAL 1 du Géoprocasseur. La structure de cette UAL est donnée en Figure II.10.

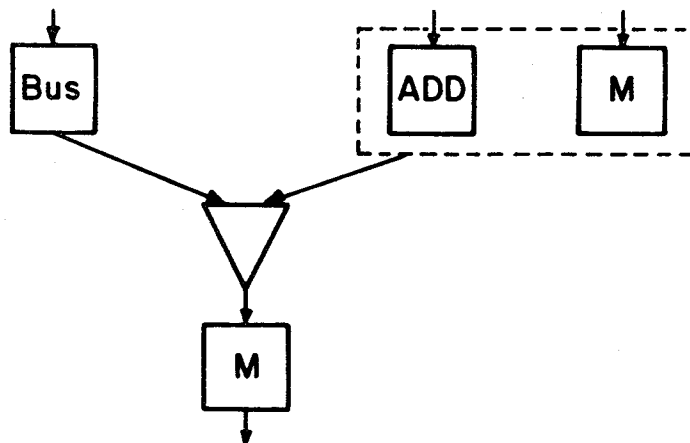
Les micro-blocs de traitement sont les registres H, N, M, l'additionneur. Les micro-blocs de commande sont le séquenceur de multiplication, l'unité de contrôle locale.

- Relations entre les micro-blocs de traitement

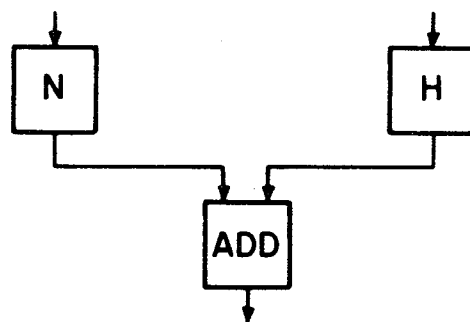
- . Relation de commandabilité : ρ_1



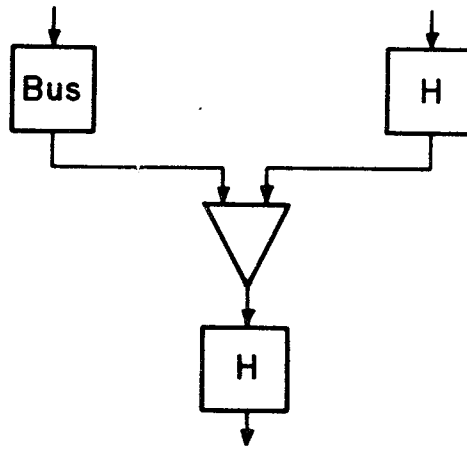
N est atteint à partir du bus ou de ADD ou de (ADD & M)



M est atteint à partir du bus ou de (ADD & M)

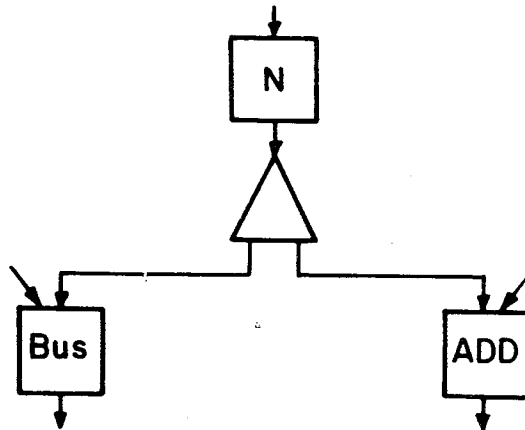


ADD est atteint à partir de H et N.

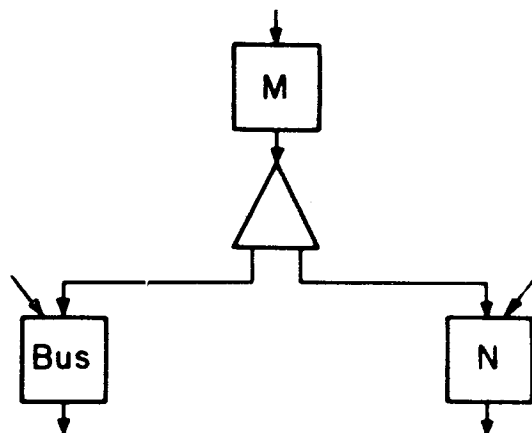


H est atteint à partir du bus ou de lui-même.

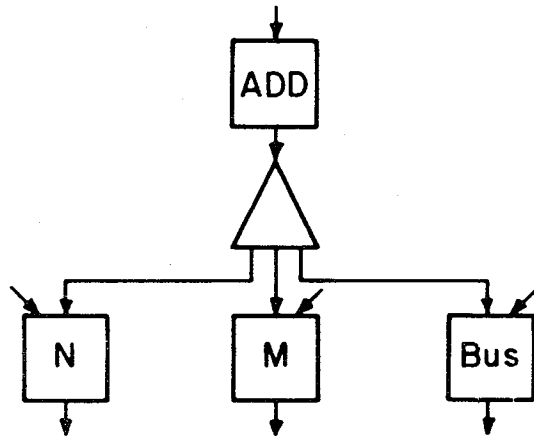
. Relation d'observabilité : ρ_2



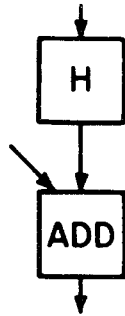
N est observable à travers le bus ou ADD



M est observable à travers le bus ou N



ADD est observable à travers N ou M ou le bus



H est observable à travers ADD.

-- Recherche d'un ordonnancement

On obtient le système d'inéquations suivant :

$$\left\{ \begin{array}{l}
 N > \text{bus} + \text{ADD} \\
 N > \text{bus} + \text{ADD} + \text{ADD.M} \\
 M > \text{bus} + N \\
 M > \text{bus} + M.\text{ADD} \\
 \text{ADD} > N + M + \text{bus} \\
 \text{ADD} > N.H \\
 H > \text{ADD} \\
 H > \text{bus}
 \end{array} \right.$$

Le système équivalent est le suivant :

$$\left\{ \begin{array}{l} N > \text{bus} + \text{ADD} \\ M > \text{bus} + N.M.\text{ADD} \\ \text{ADD} > N.H \\ H > \text{ADD} \end{array} \right.$$

L'application des règles de simplification permet d'obtenir le système final :

$$\left\{ \begin{array}{l} N > \text{bus} \\ M > \text{bus} \\ \text{ADD} > N.H \\ H > \text{ADD} \end{array} \right.$$

La chronologie de test pour les micro-blocs de traitement sera donc :

$$\left\{ \begin{array}{l} N, M \\ H \text{ et l'additionneur ADD} \end{array} \right.$$

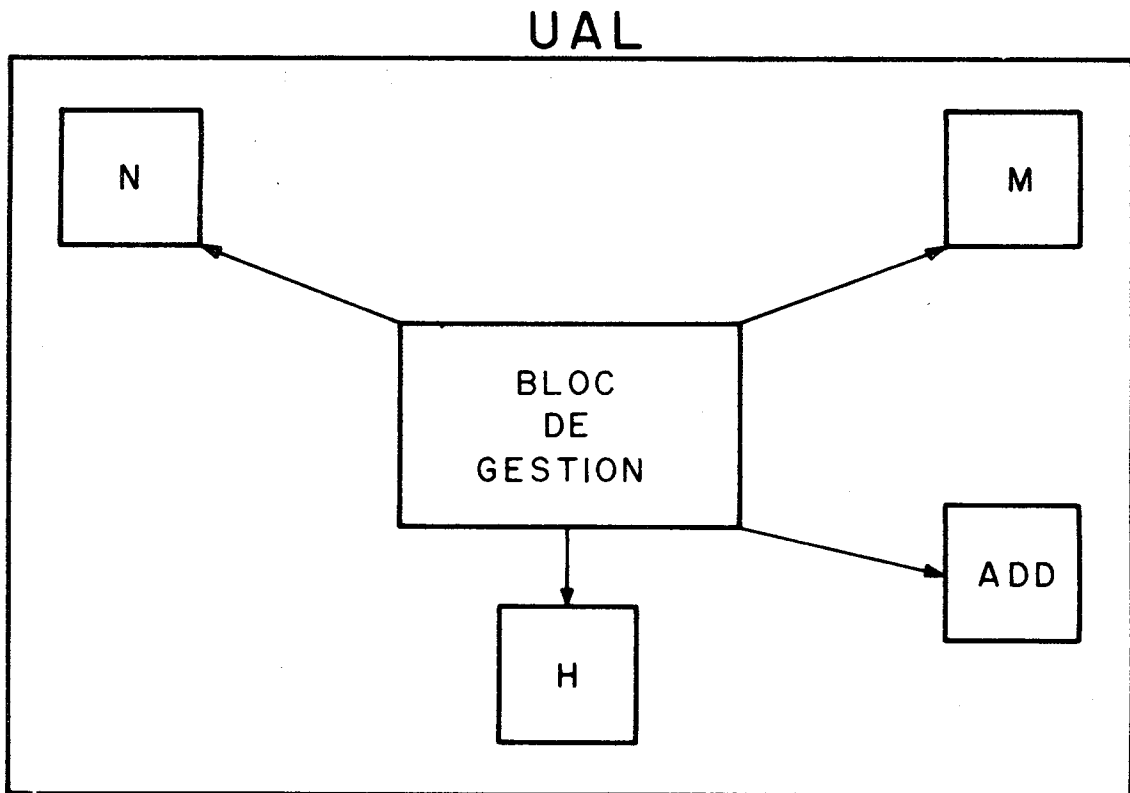
5.3.3 Analyse au niveau commandes.

Supposons l'ordonnancement des micro-blocs de traitement réalisé à l'aide de l'algorithme précédent. On cherche à ordonner les micro-blocs de commande par rapport aux micro-blocs de traitement.

On considère 2 cas :

- a) Dans un macro-bloc, il existe un micro-bloc de commande assurant la gestion de toutes les fonctions d'un sous-ensemble des micro-blocs appartenant à ce macro-bloc.

Exemple :

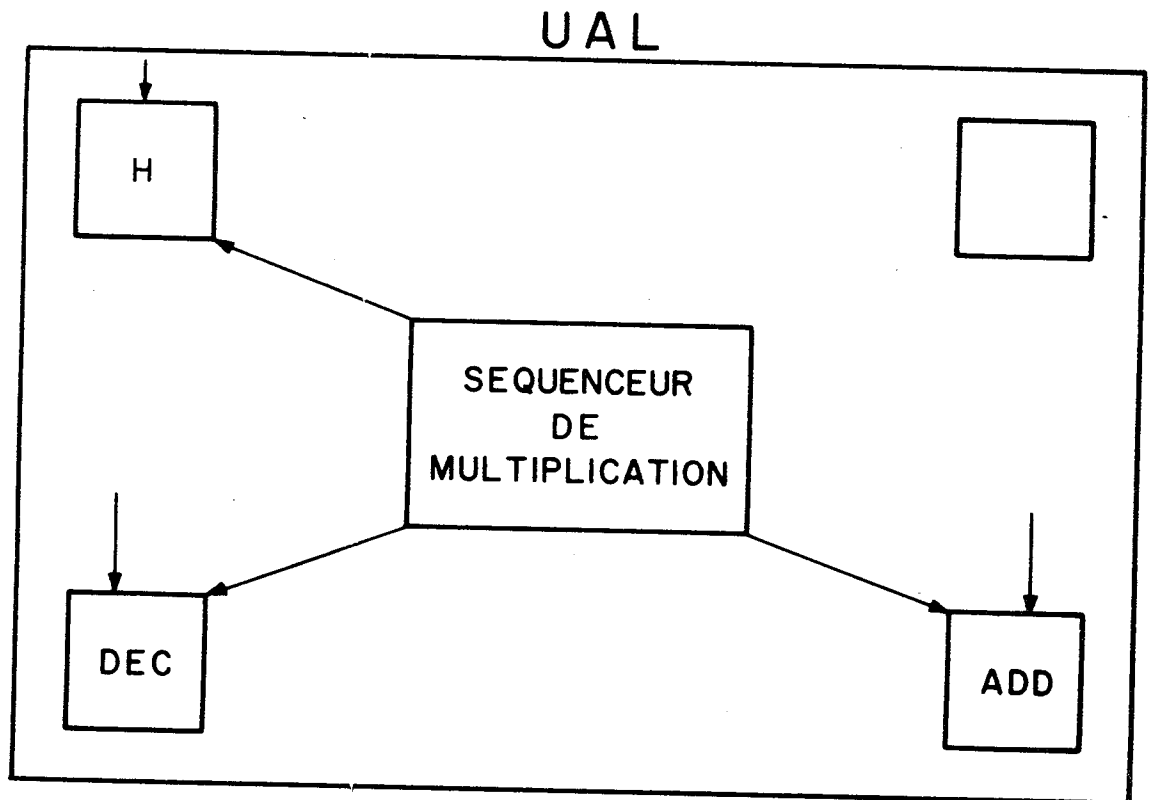


Le bloc de gestion de l'UAL I commande toutes les opérations réalisées par les micro-blocs N, M, H, ...

Dans ce cas, le micro-bloc de commande est testé en priorité, avant l'ensemble des micro-blocs gérés par lui, les micro-blocs qu'il gère (micro-blocs de traitement) ayant été ordonnés au préalable.

b) Dans un macro-bloc, il existe un micro-bloc de commande assurant la gestion d'une ou plusieurs fonctions d'un sous-ensemble des micro-blocs appartenant à ce macro-bloc; ces micro-blocs peuvent donc fonctionner indépendamment du contrôle local.

Exemple :



L'automate de gestion de la multiplication gère l'opération de multiplication effectuée par l'additionneur et la logique de décalage, mais ces deux dernières unités peuvent fonctionner par ailleurs (gérées directement par l'unité de gestion de l'UAL I).

Dans ce cas, le micro-bloc de commande est testé après les unités qu'il gère partiellement. Ces micro-blocs peuvent en effet être testés à l'aide des fonctions non gérées par ce micro-bloc de contrôle.

En résumé, dans l'ordonnancement général du test, les blocs de commande seront donc insérés immédiatement avant les blocs opératifs dont ils contrôlent tous les fonctionnements possibles et après les blocs opératifs pouvant fonctionner indépendamment d'eux (gestion partielle).

Remarque : pour le test de ces micro-blocs de commande, on s'attachera à vérifier la validité des commandes à travers les micro-blocs de traite-

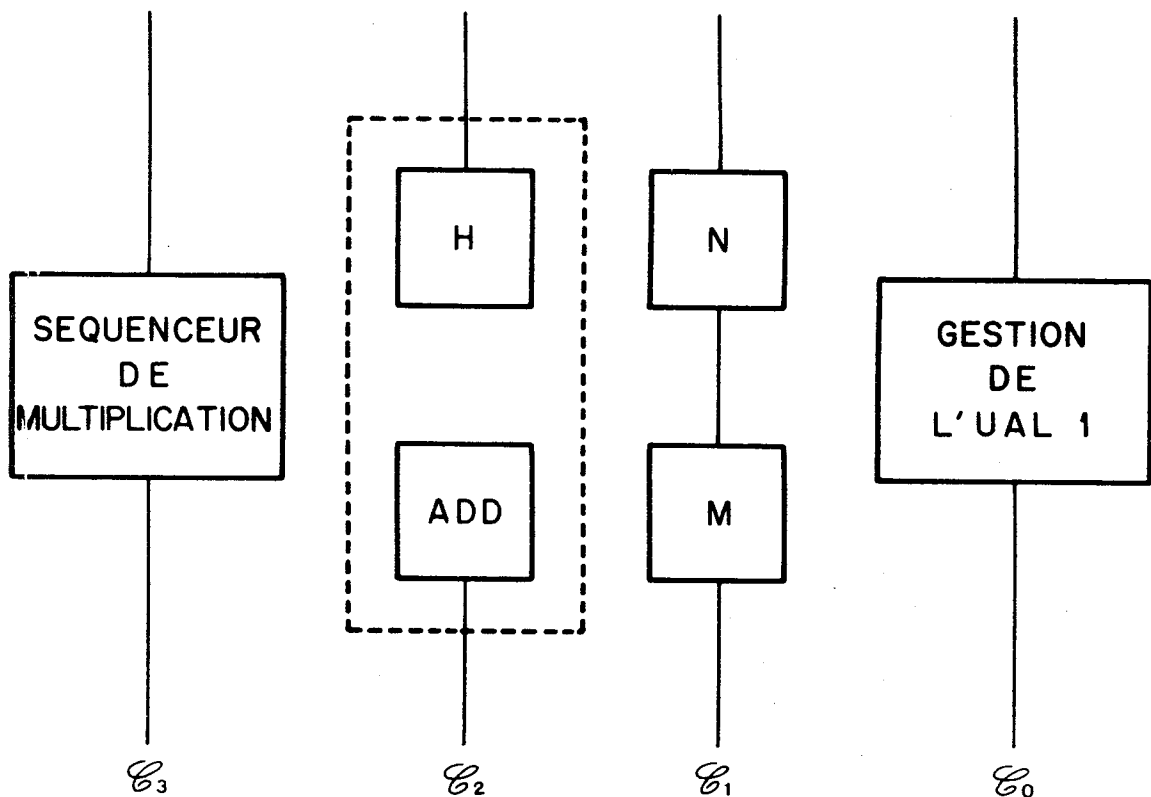
ment qu'elles activent : pour cela, on vérifiera que l'information correspondante passe ou non, indépendamment de son contenu.

Exemple :

Le bloc de gestion de l'UAL I est un micro-bloc de commande de type a) : il sera donc testé avant tous les micro-blocs qu'il gère soit H, N, M et l'additionneur.

Le séquenceur de multiplication est un micro-bloc de commande de type b) : il sera donc testé après les micro-blocs qu'il gère partiellement soit H, M et l'additionneur.

Pour l'ensemble de l'UAL I on obtient le schéma de test suivant :



Remarques :

α) Une sous-partition à l'intérieur d'une couche \mathcal{E}_i signifie que les tests de ces sous-blocs n'interfèrent pas (chemins disjoints) et que la localisation se fait d'emblée au niveau de chacun de ces sous-blocs.

Exemple : couche \mathcal{C}_1 ; sous-blocs N et M.

β) Deux blocs B_i et B_j associés à une même couche qui n'est pas partitionnée sont liés par la relation d'indiscernabilité

Exemple : couche \mathcal{C}_2 ; blocs ADD et H.

5.3.4 Problème du cheminement de l'information de test.

Le problème du cheminement de l'information de test se pose comme suit :

Considérons un bloc B_i de la couche \mathcal{C}_i . Cet élément est observable et/ou accessible à travers un bloc au moins B_{i-1} de la couche \mathcal{C}_{i-1} .

Il se pose donc deux problèmes :

- problème de masquage : le résultat d'un test peut être masqué s'il est transformé par les blocs B_{i-1}, \dots, B_0 .
- problème de consistance : l'information désirée à l'entrée de B_i peut ne pas être obtenue à la sortie de B_{i-1}, \dots, B_0 ou difficile à élaborer.

Un tel problème n'a pas été crucial dans l'exemple pratique étudié : profondeur de couches limitée, "transparence" des micro-blocs des couches \mathcal{C}_{i-1}, \dots ;




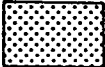
Définition : un micro-bloc est transparent s'il permet de laisser passer l'information sans effectuer de traitement dessus (ce qui est le cas par exemple pour les registres).

5.3.5 Traitement au niveau d'un micro-bloc.

On rappelle que l'information de test est attachée au micro-bloc par une méthode spécifique à ce micro-bloc et que lorsque les méthodes classiques ne s'avèrent pas efficaces, on élaborera des méthodes originales telles que décrites au Chap. I. § IV.

On donne ici une illustration des méthodes choisies pour les micro-blocs de l'exemple considéré (UAL du Géoprocasseur) (Fig. II.10.).

Le type de méthode de test employé pour chaque micro-bloc est indiqué comme suit :

- méthodes analytiques pures 
- méthode analytique prenant en compte les facteurs répétitivité et symétrie. 
- méthode fonctionnelle par vérification d'automates 
- méthodes spécifiques originales 

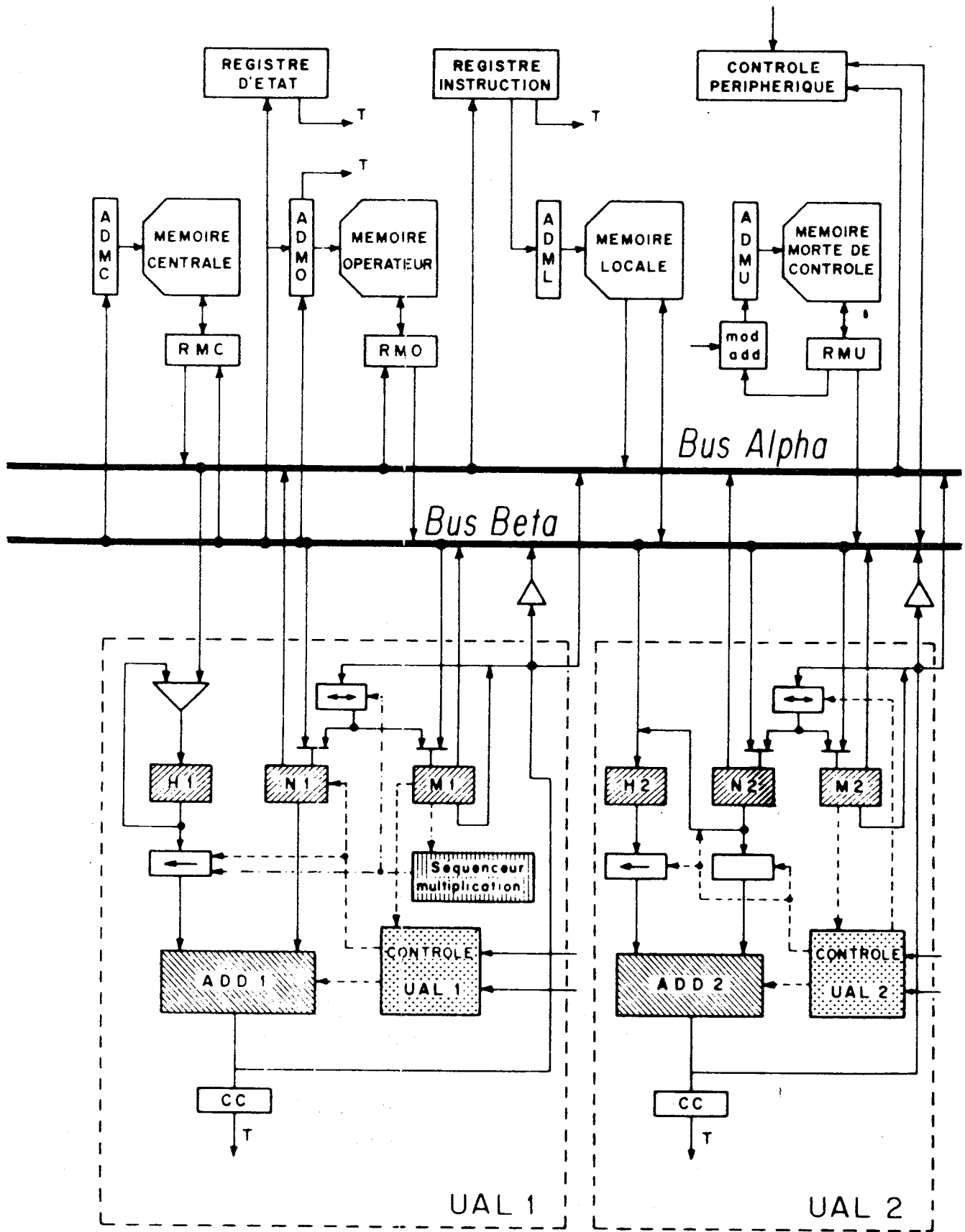


Fig. II.10.

VI – IMPACT SUR LA CONCEPTION.

De l'étude précédente il s'avère qu'un processeur pourra être facilement testable (rigueur du test, localisation), avec une adjonction de matériel minimum, si le problème du test est intégré à la conception [26], [63], [47] et en constitue un paramètre, cela à tous les niveaux, à savoir :

α) Contraintes d'architecture de haut niveau : choix et implémentation du hardware de 1er degré permettant d'appliquer ensuite une méthode de type start-small sans intervention manuelle (test automatique).

Le hardware sera donc la partie du système assumée sans faute ou qui sera testée par elle-même de façon à pouvoir être utilisée avec certitude comme base de toute la procédure de test ultérieure.

Il devra donc être minimisé et on se contentera d'ajouter le matériel indispensable pour éviter une intervention humaine (technique FLT, IBM : au minimum, voie d'accès et de chargement des micro-programmes de test).

β) Contraintes sur la description du système : on cherchera à avoir une banque de données efficace pour la description des circuits, définissant des entités fonctionnelles et physiques.

On cherchera donc à définir un partitionnement, à partir de la structure hiérarchisée, physique autant que fonctionnelle, du système, pour rendre efficace le test des sous-systèmes. Ceci impliquera une dépendance faible entre sous-systèmes.

γ) Contraintes sur la structure du chemin de données : au cours de la conception, on devra donc définir une partition à deux niveaux : macro-blocs, micro-blocs.

La partition en micro-blocs devra permettre, grâce à l'analyse du chemin de données et des niveaux de commande, d'établir facilement les relations d'ordonnement et de fournir un système d'inéquations, dont la solution soit unique sans création de blocs d'indiscernabilité; mais en cas d'existence de tels blocs on pourra concevoir le système de telle sorte que les micro-blocs indiscernables soient implantés sur une même carte.

Cette étude de la structure du chemin de données devra être complétée par une analyse de la "transparence" des blocs afin d'éviter les problèmes de cheminement (accès ou observation) de l'information.

δ) Contraintes sur les micro-blocs : il est nécessaire de pouvoir dissocier clairement les micro-blocs de traitement des micro-blocs de commande (modularité).

En outre, on devra prendre en compte la structure de ces blocs :

. micro-blocs de traitement :

- on définira de façon précise l'hypothèse des pannes
- on donnera ces méthodes spécifiques prenant en compte la commandabilité au niveau global
- on fera apparaître les facilités nécessaires à l'application ou l'élaboration de ces méthodes spécifiques.
- on fera intervenir les facteurs de symétrie, répétitivité, ...

. micro-blocs de commande :

- dans le cas d'automates simples, on imposera des contraintes sur les bascules d'état (existence de "distinguishing sequences" de longueur fixée)
- dans le cas d'automates complexes, on fera apparaître les facilités nécessaires à l'application des méthodes spécifiques.

Lorsque les points cités sont pris en compte au niveau de la conception, la méthodologie de test exposée dans ce chapitre donne lieu à des micro-programmes de test extrêmement efficaces sur un matériel donné.

On peut en déduire que si les critères fournis par cette méthodologie sont exploités lors de la conception d'un processeur, celui-ci pourra être très facilement testable (efficacité pouvant être évaluée de façon précise), pratiquement sans adjonction de matériel.

CHAPITRE III

METHODES DE TEST POUR DES AUTOMATES DE CONTROLE

* * *

INTRODUCTION.

On a vu que l'objectif, pour obtenir les méthodes de test les plus efficaces possible, est d'élaborer des méthodes au point de rencontre d'une approche fonctionnelle et d'une approche analytique : ce sont des méthodes structurelles (chapitre I § IV).

Les tests basés sur une méthode structurelle s'expriment par :

$$T = \mathcal{F} (\{f_i(x_{ij})\}, \{x_k\})$$

Le test est défini par :

- . un ensemble d'activations fonctionnelles $\{f_i(x_{ij})\}$
- . un ensemble de paramètres d'entrée $\{x_k\}$

Ce chapitre propose des méthodes structurelles pour des automate de contrôle; après avoir donné une classification des automates nous proposerons deux études de type général pour deux classes d'automates de contrôle.

I – CLASSIFICATION DES AUTOMATES.

1.1 DEFINITIONS. [19], [23], [35], [65].

Un automate d'état fini est caractérisé par un quintuplet :

$$A = \{X, Q, Y, \delta, \lambda\}$$

dans lequel :

$X = \{x_1, x_2, \dots, x_n\}$ est l'ensemble fini, non vide, des entrées primaires

$Q = \{q_1, q_2, \dots, q_p\}$ est l'ensemble fini, non vide, des états

$Y = \{y_1, y_2, \dots, y_m\}$ est l'ensemble fini, non vide, des sorties

δ est la fonction de transition d'état : $Q \times X \rightarrow Q$

λ est la fonction de sortie : $Q \times X \rightarrow Z$ (machine de Mealy)

$Q \rightarrow Z$ (machine de Moore)

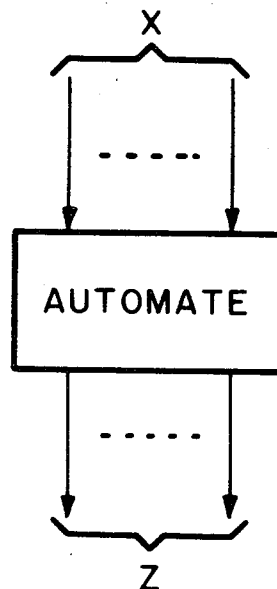


Fig. III.1.

On distinguera de façon plus précise les automates de contrôle définis par rapport au contexte : ce sont des automates dont un sous-ensemble (ou l'ensemble) des sorties sont des commandes envoyées vers une partie opérative traitant l'information du système global [23] :

$$Y = Z \cup \{C_i\}$$

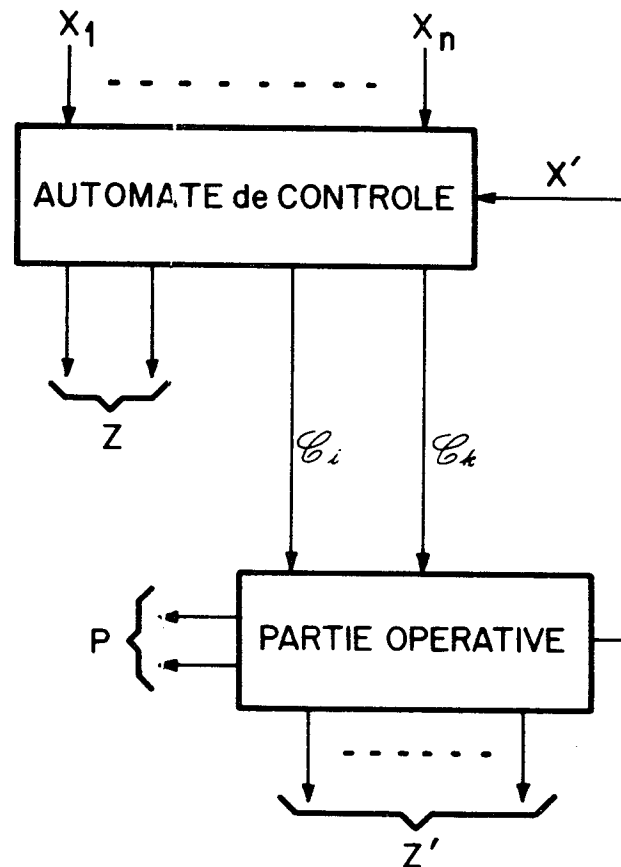


Fig. III.2.

L'ensemble {automate de contrôle + partie opérative} réalise des algorithmes (comportant un état final).

Pour chaque algorithme on peut considérer :

- 1) la fonction calculée par l'algorithme
- 2) la méthode de calcul de cette fonction (c'est-à-dire la définition, pour chaque ensemble d'entrées de l'algorithme, de la séquence d'actions à réaliser pour obtenir la fonction). Afin d'utiliser les concepts d'automates pour la description de processus algorithmiques il est nécessaire d'interpréter les signaux d'entrée et de sortie de l'automate, respectivement, comme :

- les signaux concernant l'information à traiter
- les actions élémentaires réalisées par l'algorithme

Un automate ayant un état final est appelé processeur discret d'information si une telle interprétation a été donnée. [23]

L'interprétation peut être définie comme suit :

Soit B l'ensemble informationnel dont les éléments b constituent l'information traitée par l'algorithme.

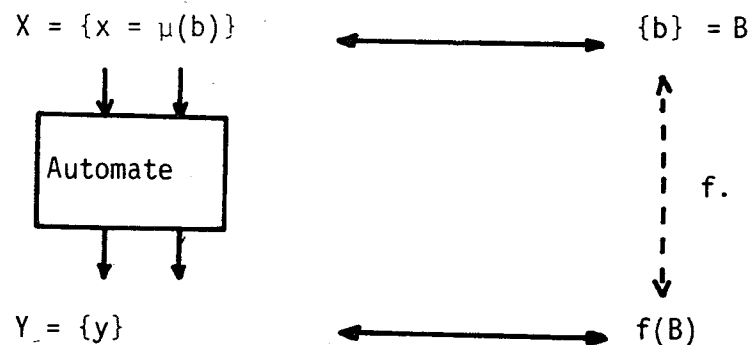
On définit une correspondance entre :

. les éléments de B et les signaux d'entrée de l'automate :

$$b \in B \leftrightarrow x = \mu(b) \in X$$

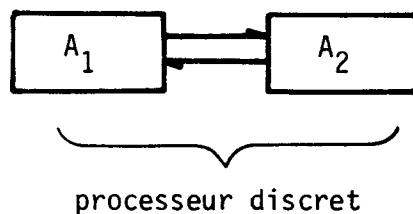
. les signaux de sortie de l'automate et les transformations de B :

$$y \in Y \leftrightarrow f(b)$$



$f(B)$ définit l'opération de l'automate.

On peut considérer l'opération d'un processeur discret dans le processus de calcul de la fonction f comme l'opération de 2 automates :



où A1 : automate de contrôle

A2 : partie opérative

La partie opérative, comportant des éléments de mémorisation, est appelée automate opérationnel (A2).

Nous nous intéresserons dans la suite de ce chapitre aux automates de contrôle (automates A1).

Ces automates auront deux types d'entrée :

- des entrées propres à l'automate : $\{X\}$
- des entrées provenant de la partie opérative (code, condition, ...) : $\{X'\}$

La partie opérative, gérée par l'automate de contrôle, pourra être constituée d'unités fonctionnelles indépendantes U_k , recevant des commandes, disjointes ou non, de l'automate de contrôle.

On aura donc le schéma général suivant :

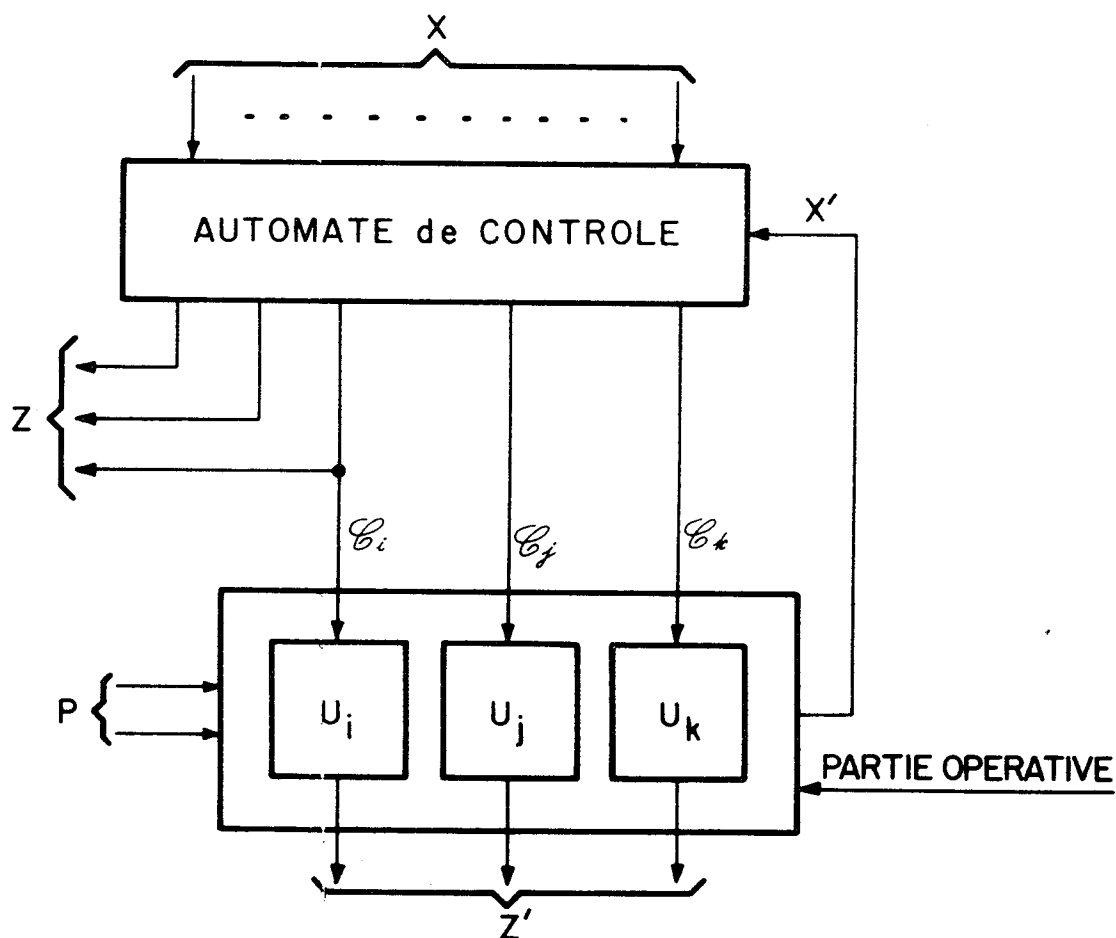


Fig. III.3.

$$\text{où } Z = \lambda(Q, X, X')$$

$$e_i = \lambda(Q, X, X')$$

Remarques :

. Les paramètres d'entrée P de la partie opérative ne sont pas nécessairement disjointes des entrées X de l'automate de contrôle :

$$P \cap X \text{ peut être non vide}$$

. $\mathcal{C}_i \cap \mathcal{C}_j$ peut être non vide

$\mathcal{C}_i, \mathcal{C}_j, \mathcal{C}_k \dots$: ensembles de commandes.

. Les paramètres d'entrée X' de l'automate sont fonction des entrées P de la partie opérative :

$$X' = f(P).$$

Pour de tels automates, le comportement interne sera caractérisé par :

- d'une part, les sorties Z de l'automate
- d'autre part, l'effet des commandes \mathcal{C}_i à travers les unités fonctionnelles U_k , c'est-à-dire les sorties Z' de la partie opérative :

$$Z' = \mathcal{F}(\mathcal{C}_i, U_i) = \mathcal{F}(\lambda(Q, X, X'), U_i)$$

Remarque : par (U_i) on entend l'état des éléments de mémorisation caractérisant la partie opérative c'est-à-dire l'état de l'automate opérationnel.

1.2 MODELISATION DE L'AUTOMATE – COMPLEXITE.

1.2.1 Modélisation de l'automate.

Les renseignements donnés par le concepteur sur un automate peuvent se situer à 2 niveaux :

. niveau fonctionnel :

- le seul renseignement consiste en la donnée de la fonction ou de l'algorithme réalisés par l'automate. Une telle donnée orientera le test de l'automate vers une vérification d'algorithme ou de fonctionnement.
- dans certains cas on possède une description plus formelle du circuit sous forme de tableau d'états c'est-à-dire que l'on a connaissance du quintuplet $\{X, Q, Y, \delta, \lambda\}$ (caractérisation des états). On est alors ramené pour le test au problème d'identification d'automates (checking experiment).

. niveau logique : on possède la description du réseau séquentiel (schéma logique) au niveau des portes et interconnexions logiques.

Une telle donnée oriente vers un test structurel, contrairement au cas précédent.

Toutefois, les renseignements fournis ne se situent généralement pas à un seul des niveaux définis; en pratique, l'automate de contrôle est donné par son schéma logique et décrit par les fonctions ou algorithmes qu'il gère.

Généralement, les méthodes de test considèrent soit l'aspect fonctionnel (checking experiment, vérification d'algorithme) soit la structure hardware (D-algorithme). Ayant une description de l'automate, à la fois structurelle et fonctionnelle, il semble judicieux d'utiliser (ou d'élaborer) des méthodes qui prennent en compte ces deux aspects.

1.2.2 Complexité

La complexité d'un automate peut se définir à chacun des niveaux cités :

. niveau fonctionnel :

- automate réalisant une famille d'algorithmes, algorithme complexe, ...
- nombre élevé d'entrées et/ou d'états, automates très incomplètement spécifiés, ...

. niveau logique : nombre élevé de portes, d'éléments de mémorisation, boucles de retour, points d'accès, ...

La complexité d'un automate, à quelque niveau que ce soit, déterminera profondément le choix d'une méthode de test.

1.3 CONTRAINTES D'ENTREE

Les contraintes sur les entrées de l'automate traduisent généralement les contraintes de l'algorithme implémenté.

On pourra distinguer en particulier 3 cas :

1.3.1 Pas de contraintes.

- les entrées X de l'automate sont indépendantes; elles peuvent s'enchaîner dans un ordre quelconque
- l'automate ne reçoit pas d'entrées issues de la partie opérative (fonctionnement autonome) : $X' = \emptyset$

1.3.2 Dépendance faible.

- les entrées X de l'automate ne sont pas indépendantes
- comme dans le cas précédent : $X' = \emptyset$ (fonctionnement autonome)

On dira qu'il y a dépendance faible lorsqu'on pourra définir des classes sur les entrées X , comme suit :

Considérons 2 entrées successives de l'automate : X_t et X_{t+1}

On définira deux ensembles de classes, l'un défini sur $\{X_t\}$ l'autre sur $\{X_{t+1}\}$:

- $E_1 = \{\mathcal{C}_i, \dots, \mathcal{C}_j\}$ tel que

$$\bigcap_i \mathcal{C}_i = \emptyset$$

$$\bigcup_i \mathcal{C}_i = \{X_t\}$$

- $E_2 = \{\mathcal{C}_\alpha, \dots, \mathcal{C}_\beta\}$ tel que

$$\bigcap_\alpha \mathcal{C}_\alpha = \emptyset$$

$$\bigcup_\alpha \mathcal{C}_\alpha = \{X_{t+1}\}$$

Il y aura dépendance faible si

$$X_t \in \mathcal{C}_i \Rightarrow X_{t+1} \in \mathcal{C}_\alpha \text{ ou à un sous-ensemble de } E_2$$

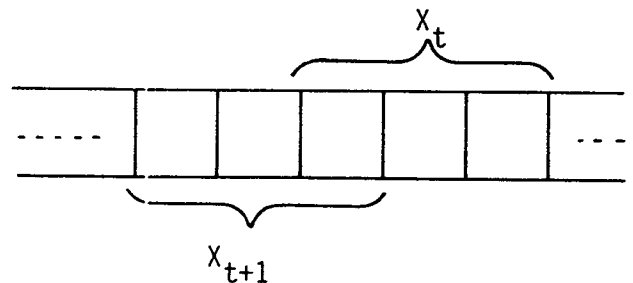
On peut donc noter que la dépendance faible est caractérisée par :

- une relation directe entre X_t et X_{t+1} (dépendance à court terme) : $X_{t+1} = g(X_t)$
- un fonctionnement autonome de l'automate : $\{X'\} = \emptyset$

Exemple :

Considérons un automate de contrôle réalisant la multiplication, suivant un algorithme de Booth.

Les entrées de l'automate sont les zones successives de 3 bits de l'opérande, se recouvrant sur 1 bit :



Le bit de poids fort de X_t (entrée à l'instant t) est égal au bit de poids faible de X_{t+1} (entrée à l'instant $t+1$).

Les entrées de l'automate sont les 8 combinaisons possibles de 3 bits.

On construira 2 ensembles E_1 et E_2 :

$$\begin{array}{l}
 E_1 \text{ défini sur } \{X_t\} \left\{ \begin{array}{l}
 \mathcal{E}_1 = \{000, 001, 010, 011\} \text{ est l'ensemble des entrées dont le bit} \\
 \text{de poids fort est égal à 0} \\
 \mathcal{E}_2 = \{100, 101, 110, 111\} \text{ est l'ensemble des entrées dont le bit} \\
 \text{de poids fort est égal à 1}
 \end{array} \right. \\
 \\
 E_2 \text{ défini sur } \{X_{t+1}\} \left\{ \begin{array}{l}
 \mathcal{E}_3 = \{000, 010, 100, 110\} \text{ est l'ensemble des entrées dont le bit} \\
 \text{de poids faible est égal à 0} \\
 \mathcal{E}_4 = \{001, 011, 101, 111\} \text{ est l'ensemble des entrées dont le bit} \\
 \text{de poids faible est égal à 1}
 \end{array} \right.
 \end{array}$$

Si l'on considère le processus d'enchaînement des entrées, on obtient :

$$\begin{array}{l}
 X_t \in \mathcal{E}_1 \Rightarrow X_{t+1} \in \mathcal{E}_3 \\
 X_t \in \mathcal{E}_2 \Rightarrow X_{t+1} \in \mathcal{E}_4
 \end{array}$$

1.3.3 Dépendance forte.

Il y aura dépendance forte si les paramètres d'entrée initiaux X de l'automate définissent un ensemble de familles de transitions.

On a alors 2 cas :

. soit $X'(P) = \emptyset$: fonctionnement autonome de l'automate.

Dans ce cas, les paramètres d'entrée X déterminent l'enchaînement global de l'algorithme. L'ensemble de familles de transitions défini par X est alors réduit à 1 seul élément

$X \rightarrow 1$ famille de transitions

. soit $X'(P) \neq \emptyset$; $X'(P)$ détermine une famille de transitions parmi l'ensemble de familles défini par X ; $X'(P)$ constitue le seul paramètre sur lequel on ait un pouvoir de décision.

La dépendance forte sera donc caractérisée par :

$X \rightarrow \{T_1, T_2, \dots, T_n\}$ (ensemble de chemins d'un algorithme ou ensemble de familles de transitions)

avec $T_1 = T_{11}, T_{12}, \dots, T_{1p}$

.....

$T_n = T_{n1}, T_{n2}, \dots, T_{nr}$

$X'(P) \rightarrow T_i = T_{i1}, \dots, T_{im}$ (choix d'un chemin d'un algorithme ou une famille de transitions)

Exemple :

Considérons l'automate réalisant l'algorithme d'addition-soustraction sur nombres flottants.

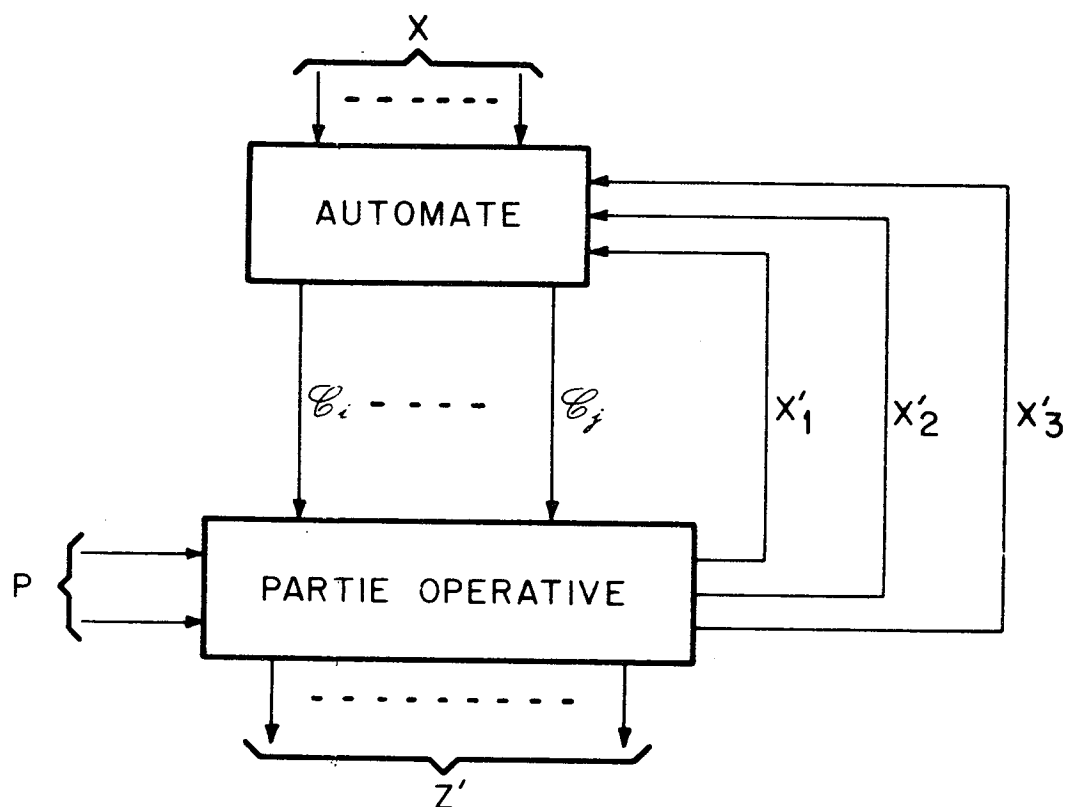


Fig. III.4.

. où X = paramètres d'entrée de l'automate : choix de l'algorithme effectué, choix de l'opération (par exemple l'addition)

. où $X' = \{x'_1, x'_2, x'_3\}$

avec x'_1 = signe de la différence des exposants

x'_2 = valeur du bit B

x'_3 = signe de l'exposant

L'algorithme réalisé par cet automate est le suivant :

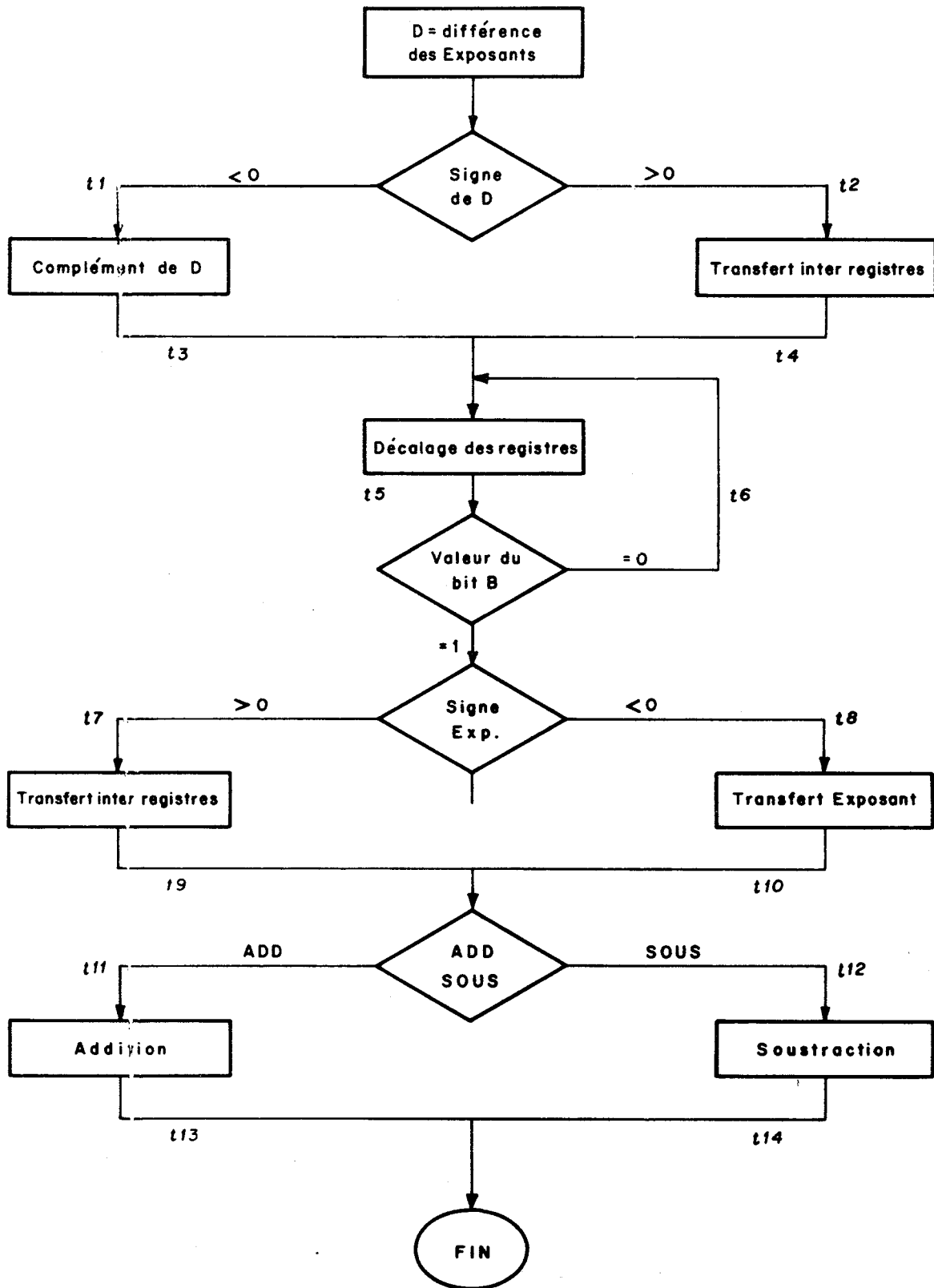


Fig. III.5.

$X \rightarrow$ ensemble de familles de transitions $T_1 \dots T_8$

$$\left\{ \begin{array}{l} T_1 = t_1 t_3 t_5 t_6^* t_7 t_9 t_{11} t_{13} \\ T_2 = t_1 t_3 t_5 t_7 t_9 t_{11} t_{13} \\ T_3 = t_1 t_3 t_5 t_6^* t_8 t_{10} t_{11} t_{13} \\ T_4 = t_1 t_3 t_5 t_8 t_{10} t_{11} t_{13} \\ T_5 = t_2 t_4 t_5 t_6^* t_7 t_9 t_{11} t_{13} \\ T_6 = t_2 t_4 t_5 t_7 t_9 t_{11} t_{13} \\ T_7 = t_2 t_4 t_5 t_6^* t_8 t_{10} t_{11} t_{13} \\ T_8 = t_2 t_4 t_5 t_8 t_{10} t_{11} t_{13} \end{array} \right.$$

T_1, \dots, T_8 définissent l'ensemble des chemins possibles de l'algorithme

$X'(P) \rightarrow T_i, 1 \leq i \leq 8$: 1 famille de transition

T_i définit un parcours déterminé de l'algorithme.

1.4 CRITERES D'OBSERVABILITE ET DE DISTINGUABILITE.

On peut considérer l'automate à 2 niveaux :

- niveau intrinsèque : on considère l'automate de contrôle seulement et on introduira à ce niveau la notion de distinguableté. Cette notion concerne l'identification des états de l'automate (fonction de transition d'états : δ)

$$\text{Etat } Q_i \xrightarrow{\delta} \{Z_i, \mathcal{C}_i\}$$

- niveau global : on considère l'automate dans son contexte (processeur discret d'information) et on introduira à ce niveau la notion d'observabilité.

Cette notion concerne la reconnaissance des sorties ou commande de l'automate à travers son contexte :

$$\begin{array}{l} \text{soit } Z_i' \text{ tel que : } \{\mathcal{C}_i\} \rightarrow \{Z_i'\} = \mathcal{F}(\mathcal{C}_i, U_i) \\ \text{soit } Z_i \text{ directement} \end{array}$$

On cherchera à définir une relation de caractérisation \mathcal{R}_C entre les commandes de l'automate et les sorties Z' de la partie opérative, ayant comme paramètres les unités fonctionnelles U_k .

1.4.1 Critère de distinguabilité : niveau intrinsèque.

$$Q_i \rightarrow (Z_i, \mathcal{C}_i)$$

Ce critère fait appel à la notion de "séquence de distinction" (distinguishing sequence).

Définition : une séquence d'entrée X_0 de l'automate est une séquence de distinction si la séquence de sortie $\{Z_i, \mathcal{C}_i\}$ produite par la machine est différente pour tout état initial de la machine.

Cas pratiques : les automates rencontrés en pratique sont souvent entièrement décodés : à chaque élément de mémorisation (bascule) du circuit séquentiel considéré correspond un ensemble de commandes $\{\mathcal{C}_i\}$, unique.

Dans ces cas là l'ensemble des commandes $\{\mathcal{C}_i\}$ caractérise de façon unique une bascule (soit encore, un état) et toute entrée de l'automate est une séquence de distinction.

1.4.2 Critère d'observabilité : niveau global.

On peut distinguer :

* Observabilité spatiale.

observabilité directe : on dira qu'un automate est directement observable lorsque ses sorties sont directement accessibles.

L'automate est caractérisé par :

$$\{Z_i; Z_i = \lambda(Q, X)\}; \{\mathcal{C}_i\} = \emptyset$$

observabilité indirecte : on dira qu'il y a observabilité indirecte lorsqu'un sous-ensemble non vide de sorties de l'automate n'est accessible qu'à travers une partie opérative (automate de contrôle).

L'automate est caractérisé par :

$$\{Z_i; Z_i = \lambda(Q, X)\}; \{\mathcal{C}_i; \mathcal{C}_i = \lambda(Q, X)\}$$

* **Observabilité temporelle.**

observabilité immédiate : on dira qu'il y a observabilité immédiate lorsque les sorties ou commandes de l'automate sont observables pendant le cycle de base où sont validées les commandes.

Soit l'instant t :

$$Q \times X \xrightarrow{\lambda} \{Z_i(t), \mathcal{C}_i(t)\}$$

il y a observabilité immédiate si on peut observer

$Z_i(t)$ à l'instant t (observabilité directe)

$Z'_i(t) = \mathcal{F}(\mathcal{C}_i(t), U_k)$ à l'instant t (observabilité indirecte)

observabilité différée : dans le cas contraire, on dira qu'il y a observabilité différée.

Soit l'instant t :

$$Q \times X \xrightarrow{\lambda} \{Z_i(t), \mathcal{C}_i(t)\}$$

il y a observabilité différée si

$Z'_i(t) = \mathcal{F}(\mathcal{C}_i(t), U_k)$ est observable à l'instant $t+k$, $k>1$ (observabilité indirecte).

Remarque : l'observabilité différée est caractéristique des automates de contrôle à observabilité indirecte (par exemple, si U_k est un réseau séquentiel).

1.5 IMPACTS DE L'OBSERVABILITE SUR LA DISTINGUABILITE.

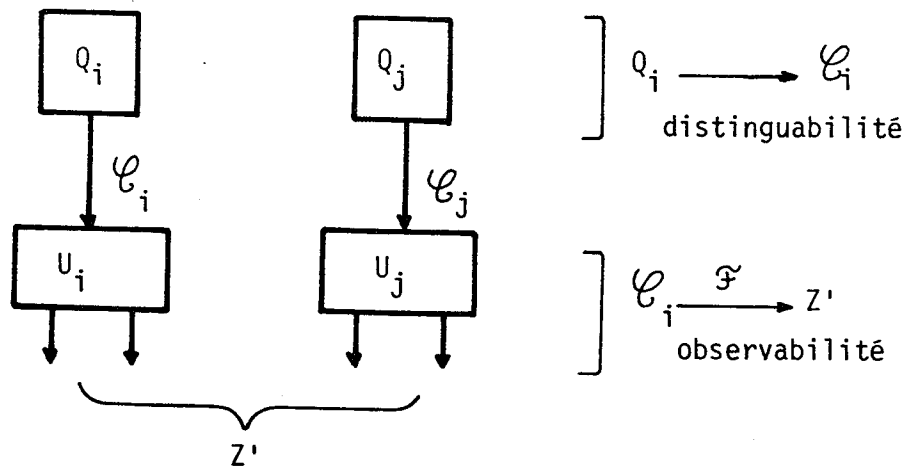
Le critère de distinguabilité et le critère d'observabilité sont fortement dépendants : ayant un ensemble de sorties ou commandes de l'automate caractéristiques d'un état (distinguabilité) on cherche une fonction $\mathcal{F}(\mathcal{C}_i, P, U_k)$ telle que les sorties Z' de la partie opérative soient également caractéristiques de cet état (observabilité).

. dans le cas d'observabilité directe, la fonction \mathcal{F} est la fonction Identité puisque $Z = Z'$

. dans le cas d'observabilité indirecte on cherche $\mathcal{F}(\mathcal{E}_i, P, U_k)$ entre $\{Z'\}$ et $\{\mathcal{E}_i\}$ soit encore entre $\{Z'\}$ et $\{Q_i\}$, caractérisant de façon unique Z' .

On peut considérer 2 structures de base :

1.5.1 Structure de type 1.



Q_i et Q_j sont deux états quelconques de l'automate de contrôle :

$$\begin{aligned} Q_i &\rightarrow \mathcal{E}_i \\ Q_j &\rightarrow \mathcal{E}_j \end{aligned}$$

\mathcal{E}_i est un ensemble de commandes activant l'unité fonctionnelle U_i

\mathcal{E}_j est un autre ensemble activant l'unité fonctionnelle U_j

et $\boxed{\mathcal{E}_i \neq \mathcal{E}_j \Rightarrow U_i \neq U_j}$

L'ensemble des sorties Z' est caractérisé de façon unique par l'ensemble des commandes, soit encore : l'ensemble des sorties observables est caractérisé de façon unique par l'ensemble des états de l'automate.

Remarque : l'ensemble des sorties observables pourra être

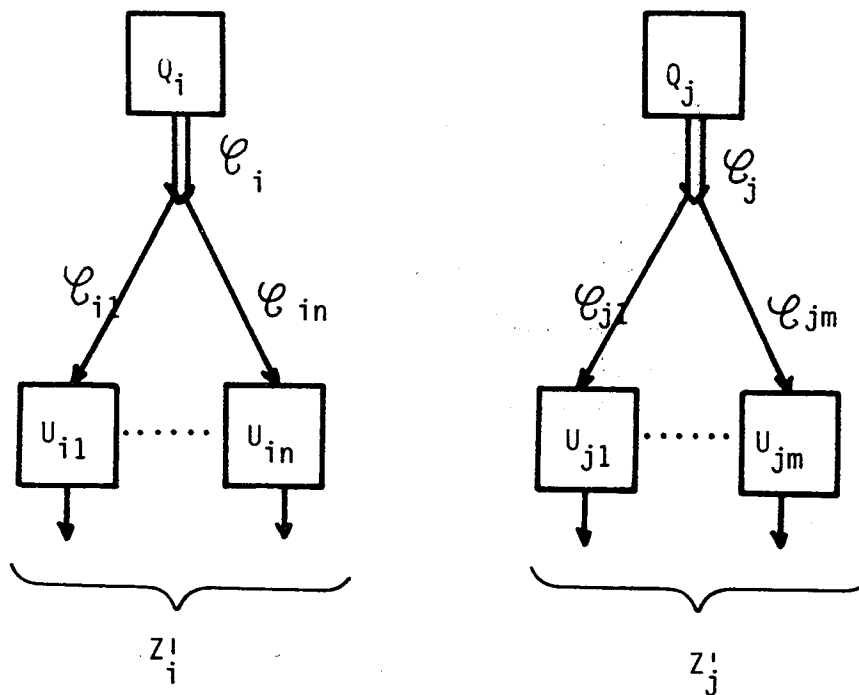
soit les sorties Z'_i de $U_i : Q_i \xrightarrow{\mathcal{R}_c} Z'_i$

soit les sorties Z'_j de $U_j : Q_j \xrightarrow{\mathcal{R}_c} Z'_j$

soit $Z'_i \cup Z'_j = Z'$: $(Q_i, Q_j) \longrightarrow Z'_i \cup Z'_j$

En pratique on ne peut observer qu'une seule unité fonctionnelle. L'identification de l'état Q_i nécessite l'observation de l'unité fonctionnelle U_i (1 observation).

1.5.2 Structure généralisée déduite de la structure de type 1.



Soient Q_i et Q_j deux états distincts de l'automate de contrôle :

$$Q_i \rightarrow \mathcal{C}_i = \{\mathcal{C}_{i1}, \dots, \mathcal{C}_{in}\}$$

$$Q_j \rightarrow \mathcal{C}_j = \{\mathcal{C}_{j1}, \dots, \mathcal{C}_{jm}\}$$

\mathcal{C}_{ik} est un ensemble de commandes activant l'unité fonctionnelle U_{ik}
 \mathcal{C}_{jk} est un ensemble de commandes activant l'unité fonctionnelle U_{jk} ,

et $\{\mathcal{C}_{ik}\} \neq \{\mathcal{C}_{jk}\} \Rightarrow \{U_{ik}\} \neq \{U_{jk}\}$

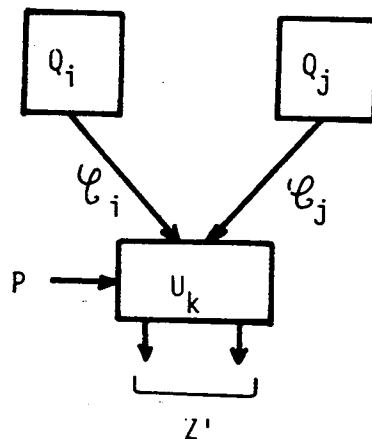
L'ensemble des sorties Z' est caractérisé de façon unique par l'ensemble des commandes, soit encore : l'ensemble des états est identifié de façon unique par l'ensemble des sorties de la partie opérative

$$Q_i \xrightarrow{\mathcal{R}_c} \{Z'_i\}$$

$$Q_j \xrightarrow{\mathcal{R}_c} \{Z'_j\}$$

Dans ce cas, l'identification de l'état Q_i nécessite l'observation des n unités fonctionnelles U_{ik} (n observations)

1.5.3 Structure de type 2.



Remarque : c'est une structure de type 1 où $U_i = U_j$

Soient Q_i et Q_j deux états distincts de l'automate de contrôle.

$$Q_i \rightarrow \mathcal{C}_i$$

$$Q_j \rightarrow \mathcal{C}_j$$

\mathcal{C}_i active une fonction f_i de l'unité fonctionnelle U_k
 \mathcal{C}_j active une fonction f_j de l'unité fonctionnelle U_k

et $\mathcal{C}_i \neq \mathcal{C}_j \implies f_i \neq f_j$

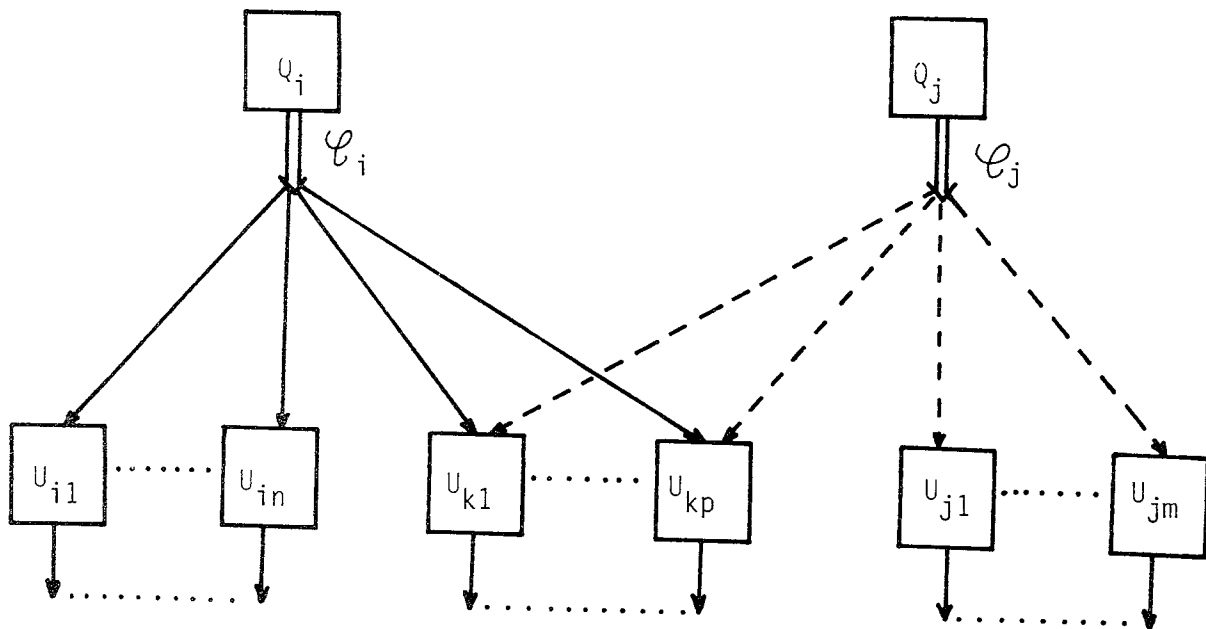
Un choix des paramètres d'entrée P de U_k permet d'établir une relation de caractérisation \mathcal{R}_c entre l'ensemble des commandes et l'ensemble des sorties Z' donc entre l'ensemble des états de l'automate et l'ensemble des sorties observables.

$$\exists P; \quad \begin{aligned} Q_i &\xrightarrow{\mathcal{R}_c} Z'_{ki} \\ Q_j &\xrightarrow{\mathcal{R}_c} Z'_{kj} \end{aligned}$$

L'identification de l'état Q_i nécessite l'observation de l'unique fonctionnelle U_k (1 observation) lors de l'activation de la fonction f_i .

1.5.4 Structure généralisée.

Cette structure est déduite des structures précédentes :



Soient deux états Q_i et Q_j distincts de l'automate de contrôle

$$Q_i \rightarrow \mathcal{C}_i = \{ \mathcal{C}_{i1}^i, \dots, \mathcal{C}_{in}^i, \mathcal{C}_{k1}^i \dots \mathcal{C}_{kp}^i \}$$

$$Q_j \rightarrow \mathcal{C}_j = \{ \mathcal{C}_{j1}^j, \dots, \mathcal{C}_{jm}^j, \mathcal{C}_{k1}^j \dots \mathcal{C}_{kp}^j \}$$

Notation :

. $\mathcal{C}_i^i = \{\mathcal{C}_{i1}^i \dots \mathcal{C}_{in}^i\}$ est l'ensemble des commandes activant les unités fonctionnelles U_{i1}, \dots, U_{in}

. $\mathcal{C}_j^j = \{\mathcal{C}_{j1}^j \dots \mathcal{C}_{jm}^j\}$ est l'ensemble des commandes activant les unités fonctionnelles $U_{j1} \dots U_{jm}$

. $\mathcal{C}_k^i = \{\mathcal{C}_{k1}^i \dots \mathcal{C}_{kp}^i\}$

$\mathcal{C}_k^j = \{\mathcal{C}_{k1}^j \dots \mathcal{C}_{kp}^j\}$

\mathcal{C}_k^i et \mathcal{C}_k^j sont l'ensemble des commandes activant les unités fonctionnelles $U_{k1} \dots U_{kp}$

les commandes \mathcal{C}_k^i activant des fonctions f_k^i

les commandes \mathcal{C}_k^j activant des fonctions f_k^j

et

$$\begin{array}{l} \mathcal{C}_i^i \neq \mathcal{C}_j^j \Rightarrow \{U_i \neq U_j\} \\ \mathcal{C}_k^i \neq \mathcal{C}_k^j \Rightarrow f_k^i \neq f_k^j \end{array}$$

On obtient donc :

$$Q_i \xrightarrow{\mathcal{R}_c} Z_i^i$$

$$Q_j \xrightarrow{\mathcal{R}_c} Z_j^j$$

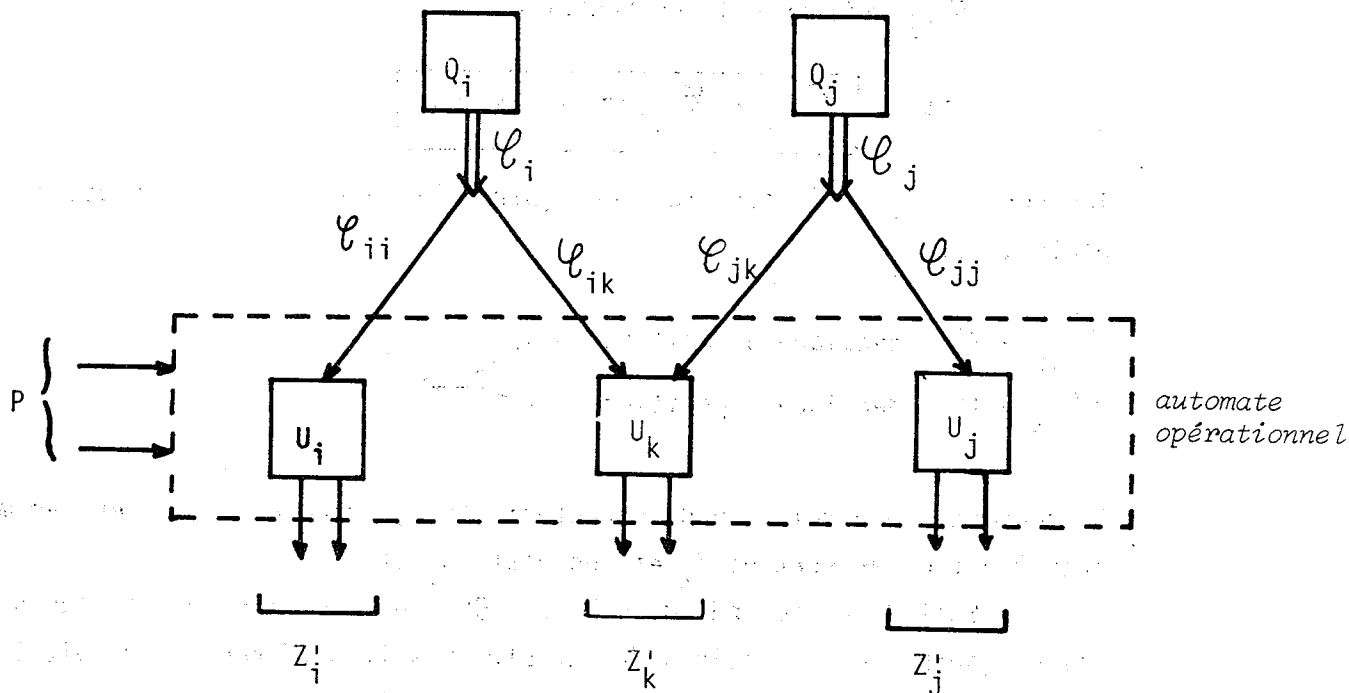
$$\exists P; (Q_i, Q_j) \xrightarrow{\mathcal{R}_c} Z_k^i$$

Dans ce cas l'identification de l'état Q_i nécessite l'observation des n unités fonctionnelles U_i et des p unités fonctionnelles U_k lors de l'activation des fonctions f_k^i (soit $n+p$ observations).

1.5.5 Exemple.

Considérons l'automate de contrôle gérant l'opération de multiplication. Cet algorithme et l'automate de contrôle associé sont explicités aux § 1.3.2 et § 2.3.

La structure considérée est du type défini au § 1.5.4.



Q_i : état caractérisé par la bascule Avance active

Q_j : état caractérisé par la bascule Avance inactive

U_i : Additionneur

U_k : registre N

U_j : registre M

\mathcal{C}_{ii} : commande d'addition

\mathcal{C}_{ik} : commande de décalage gauche 3 positions

\mathcal{C}_{jk} : commande de décalage gauche 2 positions

\mathcal{C}_{jj} : commande de chargement

Distinguabilité :

Les ensembles \mathcal{C}_i et \mathcal{C}_j sont distincts et caractérisent, de façon unique, respectivement Q_i et Q_j

Observabilité :

$$\begin{aligned} \mathcal{E}_{ii} &\rightarrow Z'_i \text{ (sortie de l'additionneur)} \\ \mathcal{E}_{jj} &\rightarrow Z'_j \text{ (sortie du registre M)} \end{aligned}$$

soit $\boxed{\mathcal{E}_{ii} \neq \mathcal{E}_{jj} \Rightarrow U_i \neq U_j}$

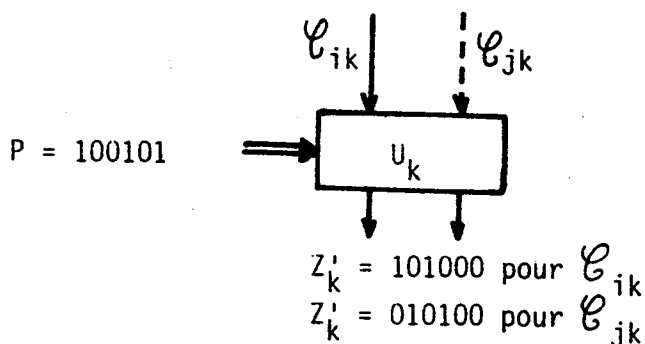
les sorties Z'_i et Z'_j caractérisent (partiellement) de façon unique les états Q_i et Q_j

$$\begin{aligned} \mathcal{E}_{ik} &\rightarrow f_i^k : \text{décalage 3 positions} \\ \mathcal{E}_{jk} &\rightarrow f_j^k : \text{décalage 2 positions} \end{aligned} \quad \begin{array}{c} \text{---} \\ \text{---} \end{array} \rightarrow U_k$$

On recherche une relation de caractérisation, grâce au choix des paramètres P entre les sorties Z'_k et les états Q_i et Q_j .

Soit P un opérande de 6 bits; $\exists P$, tel que le résultat d'un décalage gauche de 3 positions du registre N soit différent du résultat d'un décalage gauche de 2 positions de ce registre.

Par exemple : $P = 100101$



la sortie Z'_k caractérise de façon unique les états Q_i et Q_j

$$\boxed{\exists P; \mathcal{E}_{ik} \neq \mathcal{E}_{jk} \Rightarrow Z'_k(Q_i) \neq Z'_k(Q_j) \mid f_i^k \neq f_j^k}$$

II – ETUDE D'AUTOMATES DE CONTROLE – ETUDE DE TYPE 1.

[53]

Pour des raisons pratiques et des considérations d'efficacité il n'est pas souhaitable d'avoir une méthode de test unique; il est nécessaire d'avoir une famille de méthodes à notre disposition pour les structures les plus classiques.

Nous présenterons dans la suite de ce chapitre des méthodes de test structurelles pour deux types d'automates de contrôle.

Généralement les méthodes de test considèrent soit l'aspect fonctionnel, soit la structure hardware du réseau (D-algorithme). Les méthodes décrites ici prennent en compte ces deux aspects et sont illustrées sur deux exemples pratiques.

Les principales caractéristiques de ces automates sont les suivantes :

- . ces automates de contrôle ne traitent pas directement l'information mais réalisent un ou plusieurs algorithmes. Ce sont des séquenceurs locaux dont les sorties sont des commandes, envoyées à la partie opérative traitant l'information.

L'ensemble constitue un *processeur discret d'information*.

- . ces parties contrôle seront testées à travers la partie opérative : les pannes dans le séquenceur ne pourront être détectées qu'en observant leurs effets aux sorties de la partie opérative. Les sorties de ces automates de contrôle ne sont pas directement observables au niveau de l'automate.

Les méthodes de test proposées ont la caractéristique commune suivante : prendre en compte la signification fonctionnelle de la partie contrôle mais néanmoins garder tous les avantages des méthodes analytiques (localisation et rigueur du test).

$$T = \mathcal{F}(\{f_i(x_{ij})\}, \{x_k\})$$

Les deux types d'automates de contrôle considérés sont les suivants :

- . l'automate de contrôle de type 1 est la partie contrôle typique d'une UAL gérant une seule opération automatique. Un tel automate peut

être modélisé par son tableau d'états et il est possible d'utiliser une méthode d'identification d'automates [34]. L'intérêt d'une telle méthode est évident : moins de restrictions sur le type de fautes, mais il est clair qu'elle n'est pas applicable dans tous les cas puisqu'elle est fonction d'hypothèses précises sur l'automate.

. l'automate de contrôle de type 2 est plus général; c'est un automate entièrement câblé à partir d'un ensemble d'algorithmes, sous forme entièrement décodée (1 bascule par noeud de l'organigramme).

En effet, de telles structures sont souvent le résultat de procédures de conception automatiques qui, à cause de leur inhérente simplicité, apparaissent dans de nombreux systèmes logiques (contrôle de processus industriel, PDP)[4], [16].

2.1 CLASSIFICATION DE L'AUTOMATE.

On considère le bloc de gestion d'une opération automatique dans un grand système logique. La structure hardware décrite est caractéristique d'un automate de contrôle gérant une seule opération automatique et peut être considérée comme la structure élémentaire d'un automate de contrôle.

Cet automate est caractérisé par :

$$\begin{array}{l} Z = \emptyset \\ \{ \mathcal{C}_i \} = \lambda(Q, X) \\ Z' = \mathcal{F}(\mathcal{C}_i, U_i) \end{array}$$

2.1.1 Modélisation de l'automate — Complexité.

Les renseignements fournis sur l'automate se situent aux deux niveaux définis au § 1.2.

. niveau fonctionnel : on connaît la fonction réalisée par cet automate et l'algorithme associé à cette fonction.

. niveau logique : la structure hardware considérée est du type suivant (Fig. III.6); elle consiste en :

- * un compteur qui synchronise et délimite la durée accordée à l'opération, indépendamment du contrôle local (partie contrôle de l'UAL1)
- * une bascule B_i pour l'initialisation et l'arrêt de l'opération
 - . les entrées de B_i sont d'une part des signaux de contrôle (commandes d'exécution, ...) : $x_i \dots x_k$, d'autre part la sortie du compteur
 - . la sortie de B_i active le compteur et un ensemble de bascules $\{B_s\}$
- * un ensemble de bascules $\{B_s\}$ ou bascules d'états (Fig. III.6 : cas d'une seule bascule B_s).
 - . les entrées de B_s sont, d'une part des données (opérandes, ...) $x'_i \dots x'_k$ et d'autre part la sortie de B_i .
 - . la sortie de B_s génère des commandes à travers un circuit combinatoire. Ces commandes $\mathcal{C}_i \dots \mathcal{C}_k$ activent les unités fonctionnelles de la partie opérative.

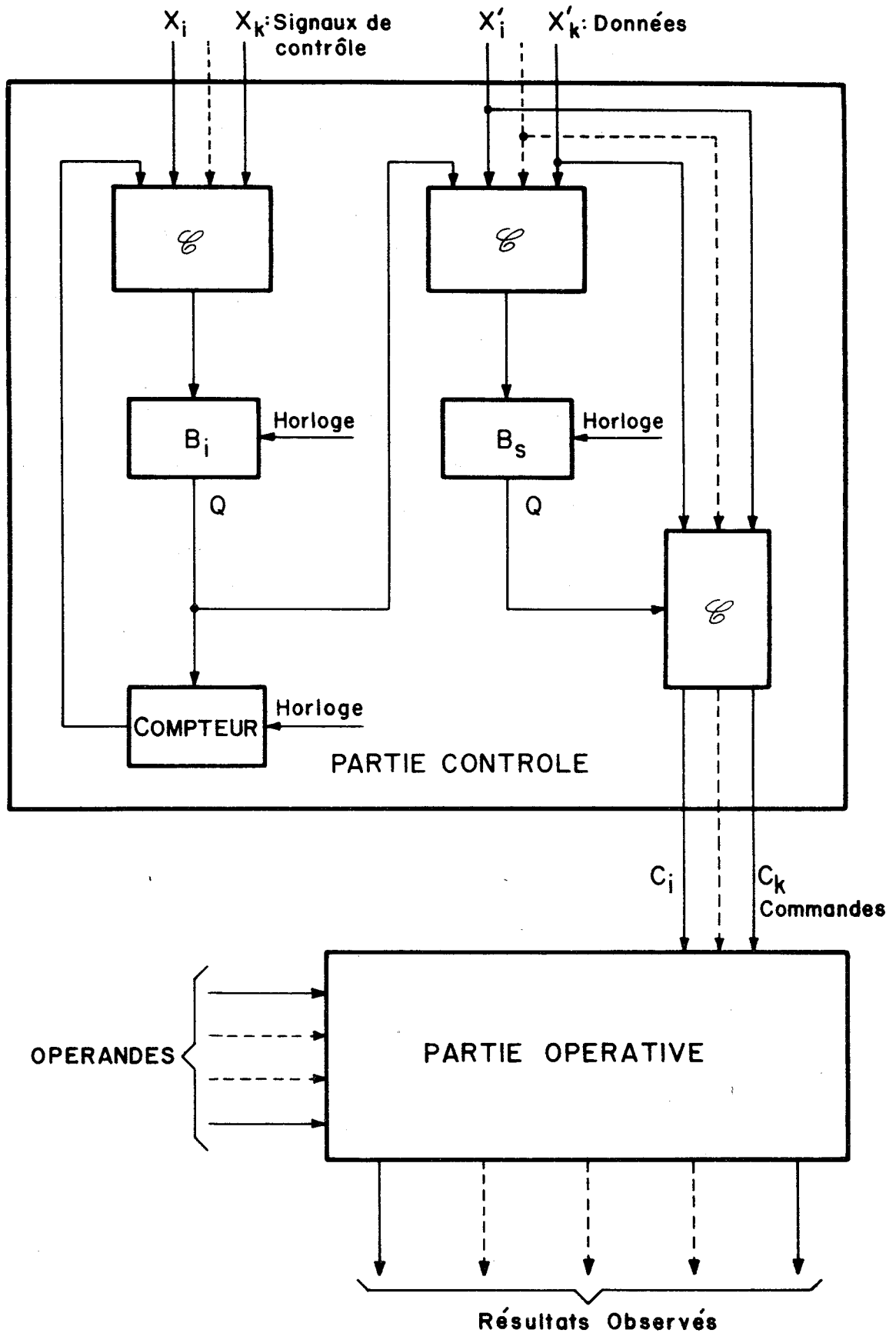


Fig. III.6.

Exemple pratique : l'automate de contrôle est le bloc de gestion de la multiplication dans une UAL (fig. III.7).

2.1.2 Définition du quintuplet $(X, Q, Z, \delta, \lambda)$.

Les renseignements fournis sur l'automate de contrôle permettent de définir $A = \{X, Q, Z, \delta, \lambda\}$ et par conséquent de décrire cet automate par un tableau d'états.

Les entrées, les sorties et les états de l'automate sont décrits comme suit :

- . les entrées sont toutes les combinaisons possibles $x_i \dots x_k$
 $x_i' \dots x_k'$
- . les états sont toutes les combinaisons possibles $(B_i, \{B_s\})$
- . les sorties sont les commandes $\mathcal{C}_i \dots \mathcal{C}_k$

On définit ainsi un tableau d'états associé à l'automate, directement déduit de l'algorithme de l'opération effectuée par l'automate de contrôle.

Réduction : de manière à obtenir une description utilisable en pratique et pour rendre le test plus facile, on cherche un automate réduit en considérant les classes d'équivalence sur les entrées. Ceci est fait en mettant dans une même classe toutes les entrées donnant les mêmes transitions et les mêmes sorties (ceci se traduit par des colonnes identiques dans le tableau d'états) |20|

2.1.3 Contraintes d'entrée.

L'automate est caractérisé par une relation de dépendance faible

- . les entrées $X = \{x_i \dots x_k, x_i' \dots x_k'\}$ ne sont pas indépendantes
- . l'automate a un fonctionnement autonome : $X' = \emptyset$

L'exemple donné au § 1.3. concerne l'automate de contrôle de la Fig. III.7.

2.1.4 Observabilité — distinguabilité.

Distinguabilité : l'automate considéré n'est pas un automate entièrement décodé et seule une analyse précise du tableau d'états permettra de dire si l'automate possède une séquence de distinction.

Observabilité :

. c'est un automate à observabilité indirecte, caractérisé par $\{\mathcal{C}_i; \mathcal{C}_i = \lambda(Q, X)\}$ et $Z = \emptyset$

. c'est un automate à observabilité immédiate ou différée.

2.2 METHODOLOGIE DE TEST. [35].

Le but de la représentation sous forme de tableau d'états est d'appliquer, si cela est possible, une méthode de vérification d'automates (checking experiment), telle qu'elle a été décrite au Chap. I, § 2.2.3.

Rappelons que cette méthode consiste à :

a) vérifier qu'on est dans un état donné en appliquant une séquence de distinction

b) vérifier toutes les transitions du tableau d'états.

. Le pas a) nécessite l'existence d'une séquence de distinction et nécessite de pouvoir définir une relation de caractérisation entre l'ensemble des commandes \mathcal{C}_i et l'ensemble des sorties Z' de la partie opérative.

Rappelons que la machine séquentielle représentée est directement associée à un algorithme et que deux entrées successives X_t et X_{t+1} sont dépendantes.

Par conséquent, pour appliquer une méthode de vérification d'automates à une telle partie contrôle, il est nécessaire que l'automate ait au moins une séquence de distinction de longueur 1.

Cette condition semble très restrictive, mais en pratique, pour une entrée donnée, deux valeurs distinctes des bascules caractérisent deux ensembles de commandes distincts; il s'ensuit que les sorties \mathcal{C}_i sont différentes pour tous les états correspondant à une entrée donnée et chaque entrée est une séquence de distinction.

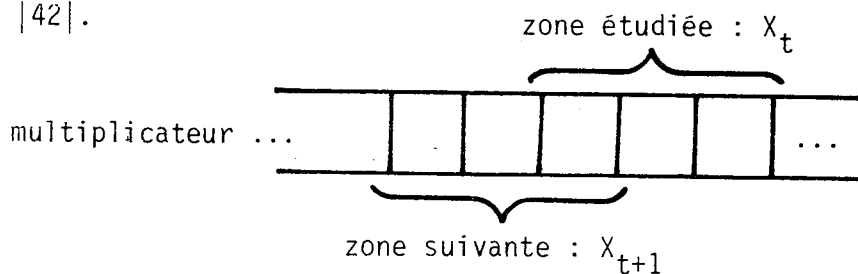
. Le pas b) est possible en considérant la machine réduite, ce qui diminue fortement le nombre de transitions à tester.

. Cette méthode permet de trouver les opérandes de test, généralement trouvés intuitivement, de manière rigoureuse

2.3 RESULTATS PRATIQUES.

2.3.1 Algorithme réalisé (§ 1.3).

L'algorithme réalisé est un algorithme de Booth (multiplication) [42].



2.3.2 Structure hardware.

La structure de ce séquenceur de multiplication est celle indiquée en Fig. III.6. où :

- . B_s est appelée "basculer Avance". Selon la valeur contenue dans la "zone suivante", cette bascule est mise à 1 et un décalage supplémentaire est effectué sur le multiplicande.
- . $\mathcal{C}_i \dots \mathcal{C}_k$ sont d'une part des commandes d'opération (addition soustraction, pas d'opération)
d'autre part des commandes de décalage (1, 2 ou 3 positions).
- . les entrées sont d'une part deux signaux de commande $x_1 \ x_2$
 x_1 : ordre de multiplication
 x_2 : remise à zéro
d'autre part une zone de 5 bits de l'opérande :
 $x_3 \dots x_7$ (zone étudiée + zone suivante).

2.3.3 Classification de l'automate.

C'est un automate de contrôle caractérisé par :

- . $Z = \emptyset, \{\mathcal{C}_i; \mathcal{C}_i = \lambda(B_i, B_s, x_1 \dots x_7)\}$
- . une dépendance faible
- . observabilité indirecte
observabilité immédiate
- . cet automate vérifie une structure de type 2.

$$Q_i \leftrightarrow \{\mathcal{C}_i\}, \quad Q_j \leftrightarrow \{\mathcal{C}_j\}$$

si $\{\mathcal{C}_i\}$ active f_i dans U_k , si $\{\mathcal{C}_j\}$ active f_j dans U_k

alors $\{\mathcal{C}_i\} \neq \{\mathcal{C}_j\} \implies f_i \neq f_j$

2.3.4 Tableau d'états associé.

Cet automate comporte 4 états :

- état a : en cours de multiplication lorsque la bascule Avance est à 1.
- état b : en cours de multiplication lorsque la bascule Avance est à 0.
- état c : état initial
- état d : état final, indiquant qu'il n'y a pas de multiplication en cours ou qu'on est en fin d'opération.

Les entrées de cet automate sont toutes les combinaisons $x_1 \dots x_7$ soit $2^7 = 128$ entrées. L'automate réduit comporte seulement 23 entrées (les représentants des classes d'équivalence).

2.3.5 Résultats pratiques du test.

Dans cet exemple pratique, chaque entrée (représentant d'une classe d'équivalence) est une séquence de distinction. Par conséquent, pour tester le séquenceur de multiplication, il est nécessaire et suffisant de passer par toutes les transitions possibles de l'automate réduit, en prenant en compte la contrainte de dépendance faible : les en-

trées se recouvrent sur 1 bit, ceci étant dû aux exigences spécifiques de l'algorithme de multiplication.

Aussi, on a assemblé les zones de 5 bits de manière à former un opérande complet, soit 20 bits (la machine a un format de 20 bits).

On a obtenu 8 vecteurs de test : les multiplicateurs des 8 multiplications que l'on devra lancer successivement pour tester cet automate.

Les multiplicandes correspondants devront être non nuls pour tester la validité de la commande d'addition-soustraction.

III – ETUDE DE TYPE 2. |49|, |53|.

3.1 CLASSIFICATION DE L'AUTOMATE.

On considère une unité de contrôle directement implémentée à partir d'une famille d'algorithmes sous forme entièrement décodée : une bascule par phase de l'algorithme (codage 1 parmi N).

Il s'agit d'un automate de contrôle gérant une famille d'opérations automatiques et cet automate peut être caractérisé par :

$$\begin{aligned} Z &= \emptyset \\ \{ \mathcal{C}_i \} &= \lambda(Q, X, X') \\ Z' &= \mathcal{F}(\mathcal{C}_i, U_i) \end{aligned}$$

3.1.1 Modélisation de l'automate. Complexité.

Les renseignements fournis sur l'automate se situent aux deux niveaux définis : niveau fonctionnel et niveau logique.

α) Considérations générales.

Le fonctionnement désiré d'un automate de contrôle (il s'agit par exemple d'un organe de gestion de tâches dans un ordinateur ou d'un organe de contrôle d'un processus industriel) est représenté par un organigramme.

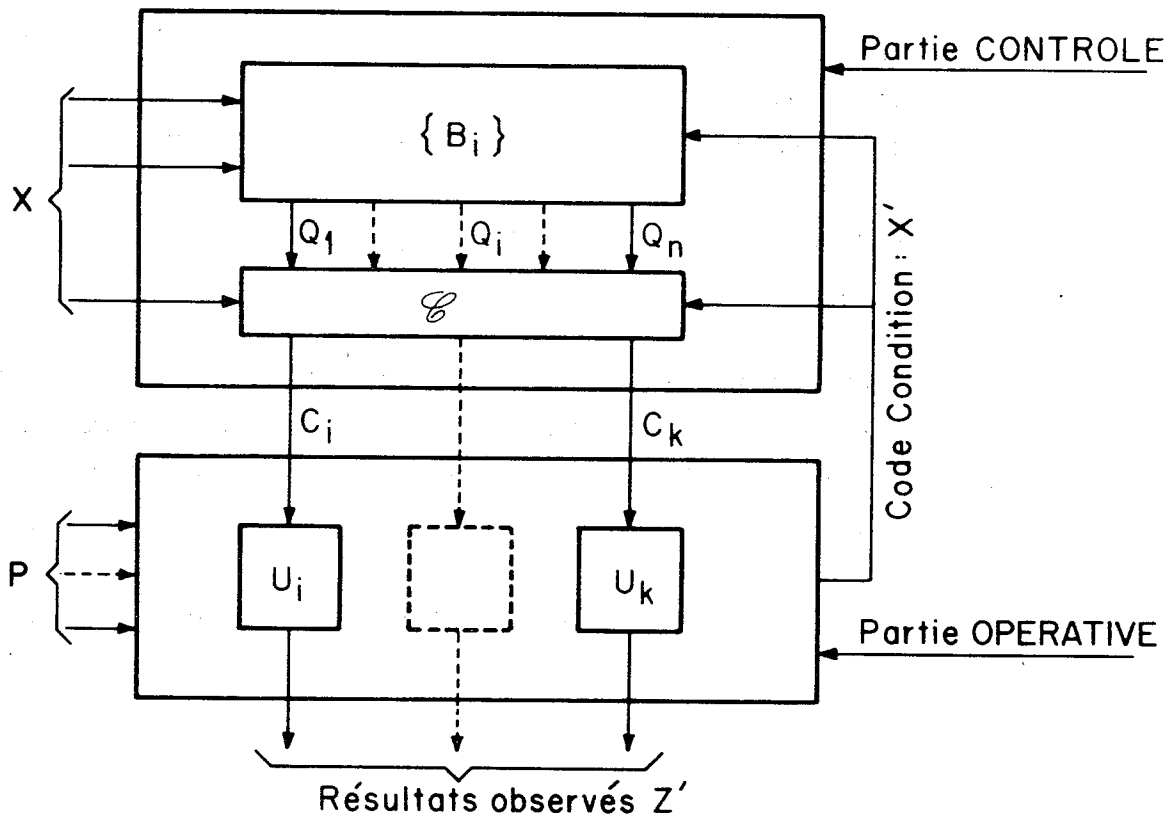
Le noeud d'un tel organigramme est appelé une "phase" et il est caractérisé par un ensemble de commandes $\{\mathcal{C}_i\}$ exécutables simultanément pendant un temps élémentaire (par exemple, le cycle de base d'un ordinateur).

Remarque : Un tel circuit est

- soit synchrone : la durée d'une phase est la borne supérieure du temps d'exécution des commandes associées à une phase.
- soit asynchrone : les unités fonctionnelles U_k commandées émettent des signaux de fin d'opération.

Nous nous placerons ici dans le premier cas.

La structure générale de cet automate est la suivante :



β) Structure de l'automate.

A chaque noeud ou phase θ_i est associée une bascule B_i , active pendant un cycle de base et dont la sortie génère l'ensemble de commandes $\{C_i\}$ qui définissent cette phase; ces commandes $\{C_i\}$ activent des unités fonctionnelles de la partie opérative (par exemple, additionneur, registre, circuit de décalage, ...).

Dans le cas général d'une famille d'algorithmes, ayant un certain nombre de commandes communes, ces commandes sont regroupées en phases de telle sorte que :

- toutes les commandes définissant une même phase sont exécutables simultanément
- le séquençement de ces phases dans chaque algorithme est respecté
- on minimise le nombre de phases (recherche de sous-algorithmes communs).

Remarque : Généralement un seul algorithme est exécuté à un moment donné.

La réalisation, relative à l'ensemble de ces algorithmes, sera donc une imbrication de tous les algorithmes particuliers et représentera le processus global.

Exemple : Considérons 2 algorithmes A1 et A2 activant les commandes $C_1 \dots C_8$:

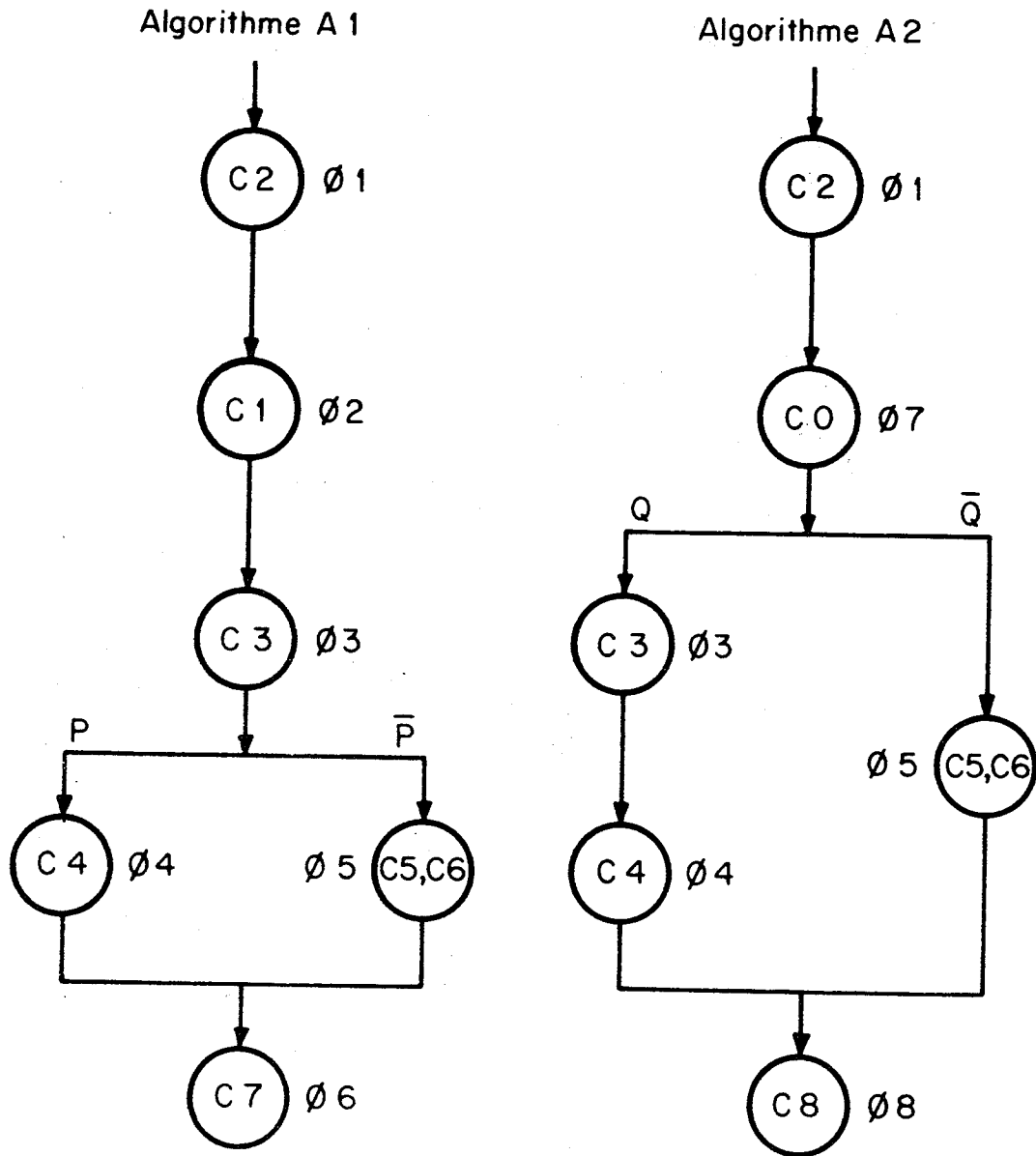
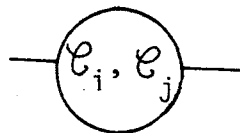
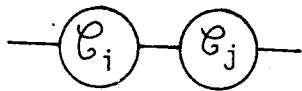


Fig. III. 8.



signifie que les commandes C_i et C_j sont exécutables simultanément.



signifie que les commandes C_i et C_j ne peuvent être exécutées qu'en séquence.

On obtient alors l'automate global, réunion de ces deux algorithmes, comme suit :

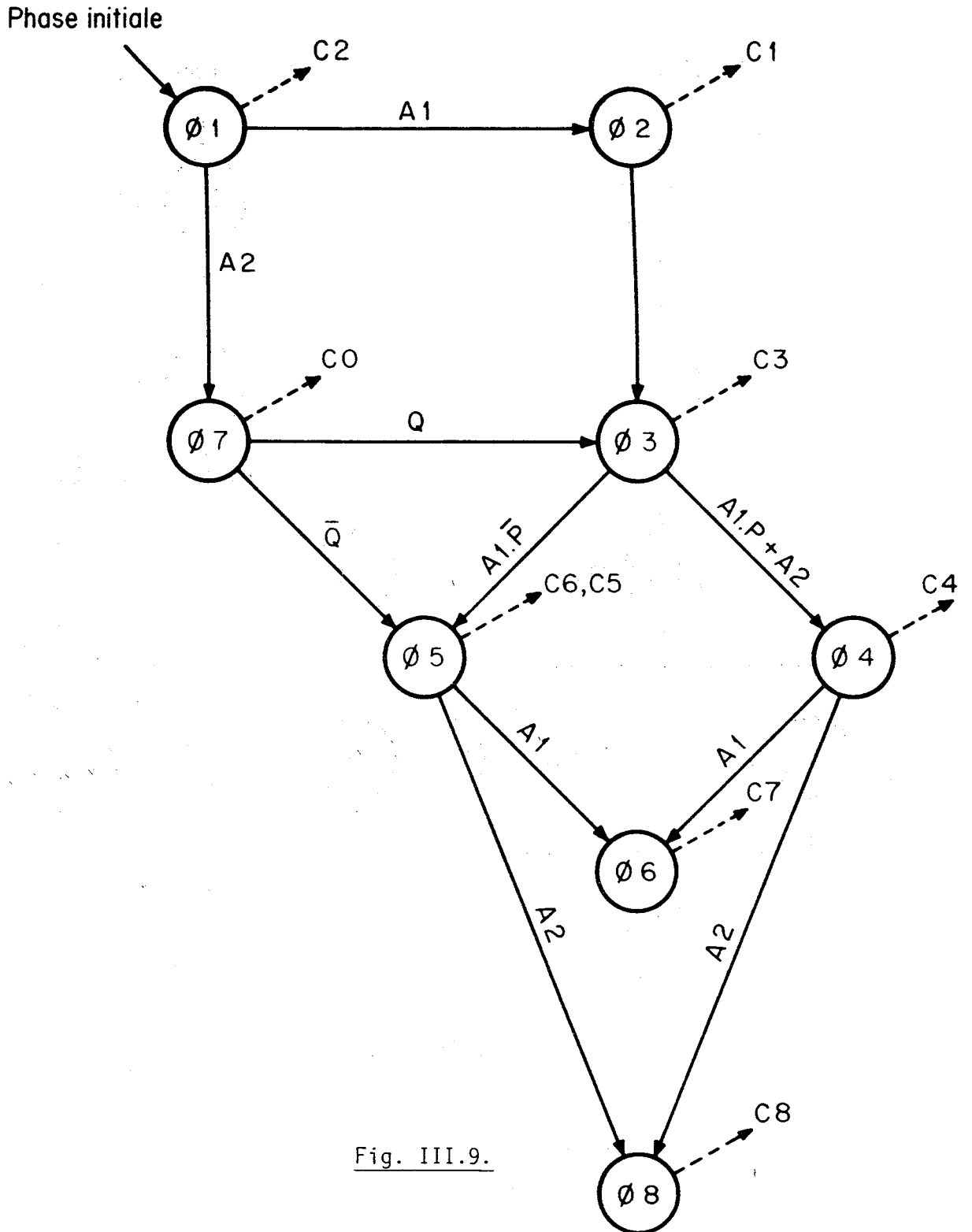


Fig. III.9.

Remarque : le graphe de transition, en général, n'est pas fortement connexe et l'automate n'est pas entièrement déterminé; certaines transitions sont toujours réalisées (inconditionnellement) (par exemple, transition $\text{Ø}2 \rightarrow \text{Ø}3$).

Le circuit câblé directement déduit de l'automate global est le suivant :

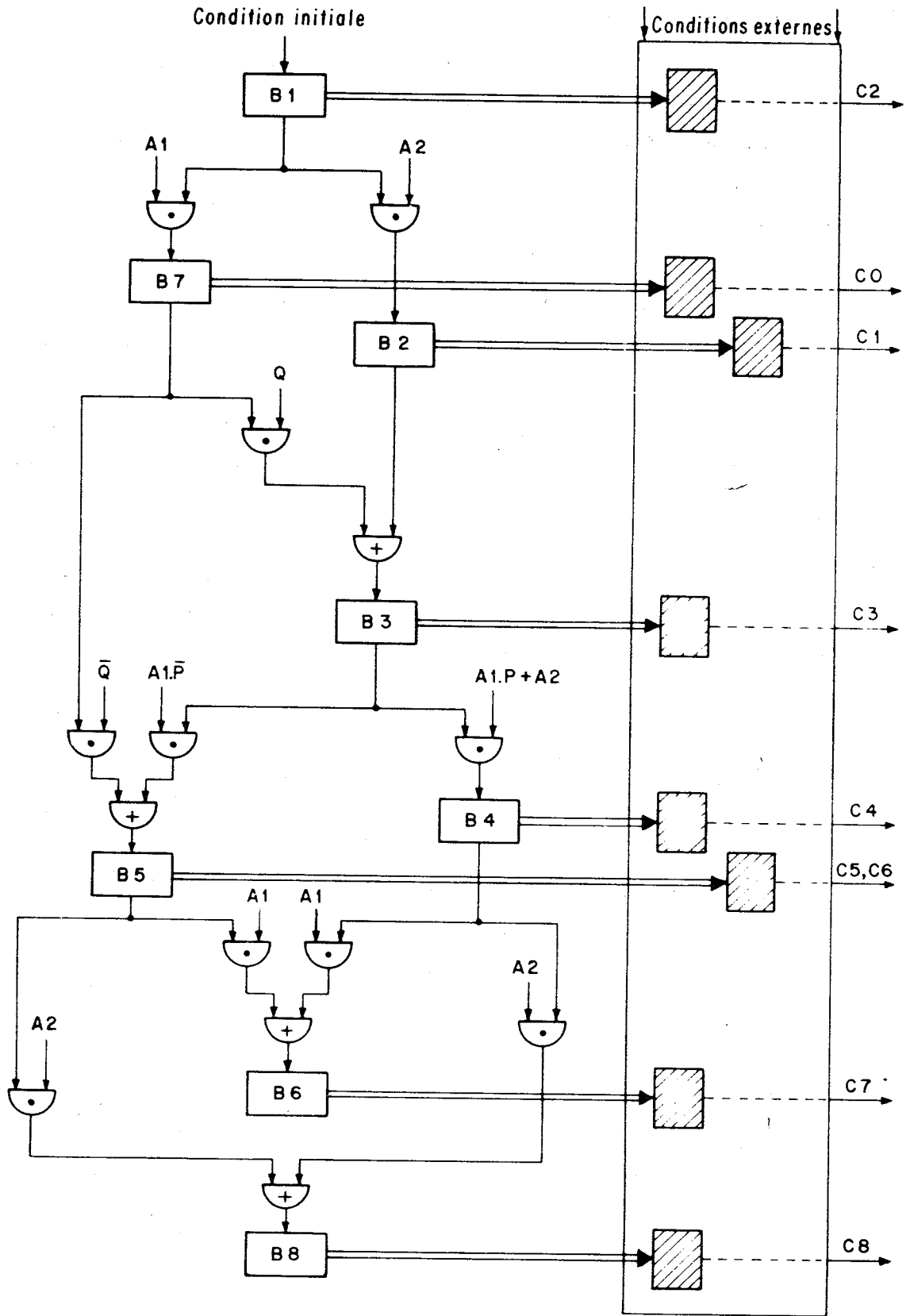


Fig. III.10.

γ) Exemple pratique (Fig. III.11)

L'exemple pratique étudié est la partie contrôle d'une UAL [17] gérant les opérations sur nombres flottants. Cette unité arithmétique et logique fait partie d'un calculateur microprogrammé, spécialisé, de taille moyenne, construit par l'"Institut Français du Pétrole" en collaboration avec l'ENSIMAG.

Cette unité réalise

- d'une part, la multiplication entière
- d'autre part des opérations de décalage et des instructions sur nombres flottants.

Elle gère 10 opérations automatiques (addition - soustraction, multiplication, normalisation, décalages arithmétiques ou logiques, ...) correspondant à 10 algorithmes différents.

Chaque opération automatique exécutée est représentée par un algorithme (Fig. III.12); les différents algorithmes sont assemblés de manière à donner la réalisation de la partie contrôle, comportant environ 150 portes, 17 bascules et 1 compteur (Fig. III.13).

3.1.2 Contraintes d'entrée.

L'automate est caractérisé par une relation de dépendance forte.

- . les paramètres d'entrée initiaux X de l'automate définissent des familles de transitions.
- . $X'(P) \neq \emptyset$: les paramètres d'entrée X définissent une famille de transitions parmi celles déterminées par X (fonctionnement non autonome).

L'exemple donné au § 1.3. concerne l'automate de contrôle de la Fig. III.11.

Les paramètres X sont définis au début d'une opération (type d'algorithme, opérands, ...) et restent tels quels pendant le déroulement de l'algorithme.

3.1.3 Observabilité — Distinguabilité.

Distinguabilité : l'automate considéré est un automate entièrement décodé donc très incomplètement spécifié et à chaque élément de mémorisation correspond un ensemble unique de commandes.

Observabilité :

. c'est un automate à observabilité indirecte, caractérisé par $\{ \mathcal{C}_i; \mathcal{C}_i = \lambda(Q, X, X') \}$ et $Z = \emptyset$

. c'est un automate à observabilité différée ou immédiate.

3.1.4 Impacts sur le choix d'une méthode de test.

La relation de dépendance forte et la contrainte d'observabilité indirecte nous ont conduit à abandonner les méthodes classiques de test :

- . les méthodes de propagation (D-algorithme, différence booléenne, ...) ont été abandonnées vu la complexité et la forte séquentialité de l'automate
- . de même, une méthode d'identification d'automates s'est révélée inapplicable
 - d'une part à cause du nombre élevé d'entrées (60 pour l'exemple considéré)
 - d'autre part parce que la machine ne satisfait pas les hypothèses propres à cette méthode (augmentation du nombre d'états sous l'hypothèse d'une panne, automate très incomplètement spécifié, réseau non fortement connexe, ...).

3.2 METHODOLOGIE DE TEST. [49]

La structure et le contexte particulier d'un automate de ce type nous ont conduit à élaborer une méthode de test spécifique : cette méthode n'est pas une simple vérification d'algorithme mais un test, le plus rigoureux possible de la réalisation hardware correspondante. Le balayage exhaustif des éléments technologiques s'appuiera sur la structure algorithmique de la réalisation.

On définit donc une méthode prenant en compte à la fois l'aspect fonctionnel et la structure hardware du matériel testé, suivant les principes énoncés au Chap. I. § 4.2.

3.2.1 Remarque préliminaire.

La réalisation hardware que nous considérons est caractérisée par la propriété suivante : à un instant donné, une bascule et une seule est activée (sortie égale à 1).

3.2.2 Différents types de pannes.

Il y a 3 sortes de pannes hardware possibles :

- α) les pannes sur les fils de commande
- β) les pannes dans le séquençement des phases
- γ) les pannes sur les éléments de mémorisation qui ont deux manifestations possibles :

- . type 1 : les collages à valeur inactive (sortie collée à 0, ...)
- . type 2 : les collages à valeur active (sortie collée à 1, refus de désexcitation,...).

Les deux derniers types de pannes (β et γ) mentionnés ci-dessus ont un des effets suivants :

- aucune bascule n'est activée
- une bascule autre que celle requise est activée
- deux bascules, au moins, sont activées en même temps.

3.2.3 Pannes sur les éléments de mémorisation.

α) Type 1 :

les pannes de collage à valeur inactive sont détectées :

- en passant au moins une fois par chaque phase en utilisant un ou plusieurs algorithmes. On choisira donc un ou plusieurs paramètres P1 et des paramètres P2 incomplètement spécifiés.

- en observant les résultats intermédiaires : chaque phase sera associée à une unité fonctionnelle U_k , activée par un sous-ensemble des commandes relatives à cette phase; cette unité fonctionnelle est choisie de telle sorte que sa sortie manifeste la panne, de préférence pendant le même cycle de base.

Bien entendu, l'arrêt d'un algorithme (aucun algorithme exécuté, aucune bascule activée) est signalé par la partie opérative.

β) **Type 2 :**

détection des fautes de collage à valeur active: la complexité de la détection des collages à valeur active nous a conduit à élaborer une méthode spécifique.

. Définition préliminaire : on dit qu'une commande \mathcal{C}_j perturbe une commande \mathcal{C}_i , si dans la même unité fonctionnelle U_k , \mathcal{C}_i et \mathcal{C}_j étant simultanément activées, l'opération effectuée est différente de celle activée par \mathcal{C}_i (en observant l'unité fonctionnelle U_k).

Remarque : " \mathcal{C}_j perturbe \mathcal{C}_i " n'implique pas " \mathcal{C}_i perturbe \mathcal{C}_j ".

. Conséquence sur le test des bascules : si \mathcal{C}_i et \mathcal{C}_j sont respectivement des sorties de B_i et B_j et si \mathcal{C}_j perturbe \mathcal{C}_i (\mathcal{C}_i et \mathcal{C}_j affectant la même unité fonctionnelle U_k) alors l'activation de B_j manifeste le collage à 1 de B_i .

On dira que " B_i teste B_j " par observation de l'unité fonctionnelle U_k .

Soit B_i une bascule donnée; on définit la classe $\mathcal{E}\mathcal{L}(B_i, U_k)$ par les règles suivantes :

. $B_i \in \mathcal{E}\mathcal{L}(B_i, U_k)$: B_i est le représentant de $\mathcal{E}\mathcal{L}(B_i, U_k)$

. $B_j \in \mathcal{E}\mathcal{L}(B_i, U_k)$ si et seulement s'il existe une valeur des paramètres d'entrée P_1 telle que " B_i teste B_j " en observant les sorties de l'unité fonctionnelle U_k .

. Construction d'une partition de l'ensemble des bascules

On construit ces classes de manière progressive :

- on choisit un algorithme avec une phase initiale \emptyset_0 (bascule associée B_0); les paramètres d'entrée P1 et P2 sont partiellement déterminés par le choix de l'algorithme et d'un chemin dans cet algorithme; on choisit une unité fonctionnelle U_k et on détermine $\mathcal{C}(B_0, U_k)$;
- on choisit la première bascule suivante B' , non incluse dans $\mathcal{C}(B_0, U_k)$ (soit dans le même chemin, soit dans un autre chemin du même algorithme, soit dans un autre algorithme) et on lui associe de la même manière $\mathcal{C}(B', U'_k)$.

On ne mettra pas dans $\mathcal{C}(B', U'_k)$ une bascule appartenant à $\mathcal{C}(B_0, U_k)$ (déjà testée au premier pas) à l'exception de B_0 (non testée).

- Cette construction est terminée lorsque toutes les bascules ont été considérées.

Remarque : la relation de "perturbation" n'est pas une relation d'équivalence; les classes $\mathcal{C}(B_i, U_k)$ ne sont pas déduites de ces relations mais construites à partir de celles-ci.

La partition n'est donc pas unique et est fonction :

- de l'algorithme initialement choisi
- du premier représentant de chaque classe
- de l'unité fonctionnelle choisie
- de la détermination progressive des paramètres d'entrée.

Exemple : Considérons l'exemple théorique donné au § 3.1.1.β

Supposons que A_1 soit un algorithme d'addition-soustraction et que l'opération choisie soit la soustraction.

C_4 est la commande d'addition, C_5 la commande de soustraction.

Ces commandes sont telles que, lorsque C_4 et C_5 sont simultanément activées, l'opération effectuée est l'addition. Par conséquent, par observation de la sortie de l'additionneur, C_4 perturbe C_5 . Il s'ensuit que " B_5 teste B_4 ".

F_4 et F_5 appartiennent à une même classe et B_5 est le représentant de cette classe : $\mathcal{C}(B_5, \text{Additionneur})$.

Les deux opérandes $Op1$ et $Op2$ nécessaires pour cette opération, sont partiellement déterminés de telle sorte que le résultat de l'addition soit différent du résultat de la soustraction.

Par exemple il suffit de choisir : $Op1 > 0$, $Op2 > 0$ et $Op2 > Op1$; alors $(Op1 + Op2)$ est différent de $(Op1 - Op2)$.

On mettra dans cette classe toutes les autres bascules B_i telles que " B_5 teste B_i " et on déterminera les opérandes de la même manière.

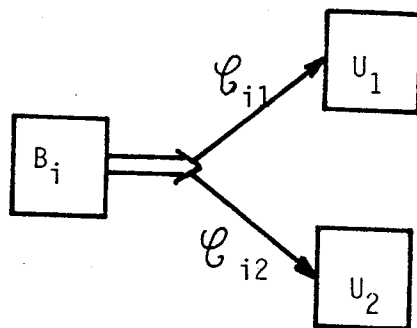
Exemple pratique : dans l'exemple pratique donné au § 3.1.1.γ les 17 bascules ont été partitionnées en 5 classes.

. Conclusion :

Pour tester les collages à un des éléments de mémorisation, on doit passer au moins une fois par le représentant de chaque classe; pour chaque classe, les paramètres d'entrée X et P sont spécifiés (incomplètement ou non) de manière à tester les éléments de cette classe.

3.2.4 Pannes sur les fils de commande.

- le collage à un des fils de commande est testé lors de l'analyse des différents éléments fonctionnels auxquels ils sont associés (ce problème n'est pas considéré ici).
- les collages à zéro : durant le test des collages à valeur inactive des éléments de mémorisation, on note que :



si B_i a été testée à travers l'unité fonctionnelle U_1 le sous-ensemble de commandes \mathcal{C}_{i1} a été testé (collages à 0).

Aussi, durant le test des éléments de mémorisation, on essaie de couvrir le test de tous les fils de commandes, en observant toutes les unités fonctionnelles associées à chaque bascule.

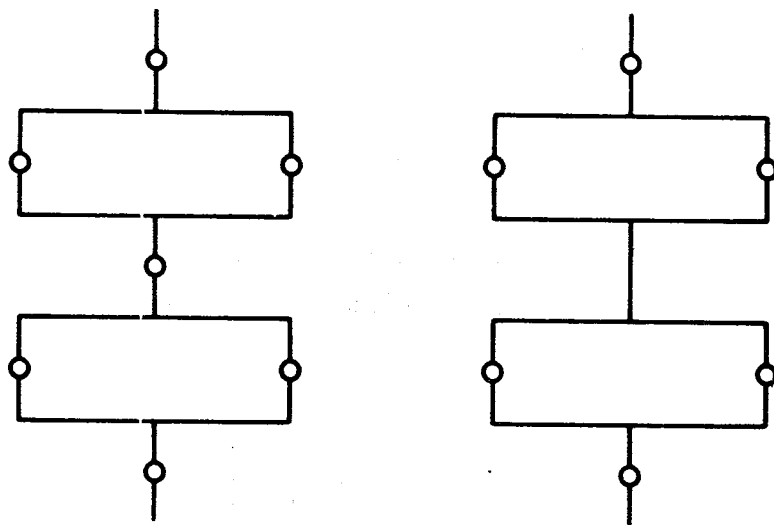
Sinon, les pannes restantes sont considérées dans une seconde étape.

3.2.5 Pannes sur le séquençement des phases.

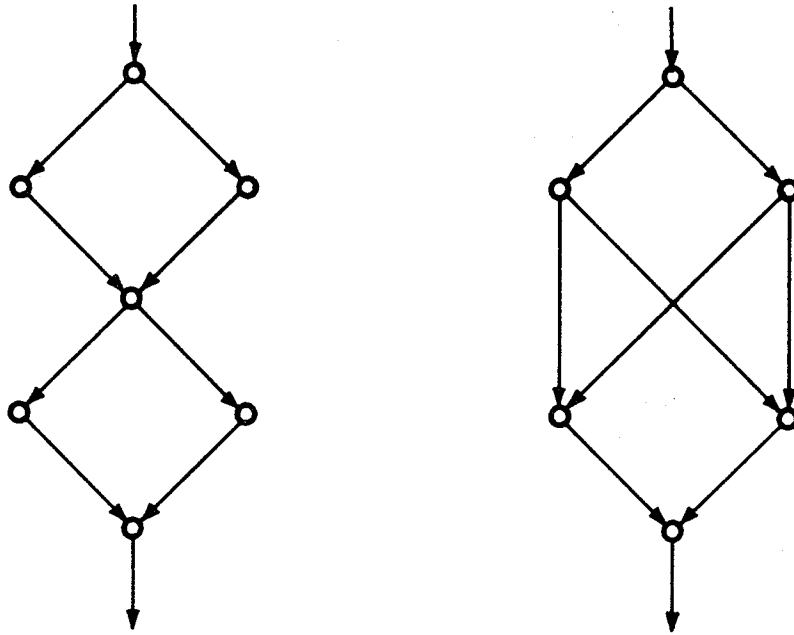
Pour tester le séquençement des phases on doit passer par toutes les transitions possibles de tous les algorithmes, les paramètres d'entrée étant partiellement spécifiés pour cela.

On est donc amené à choisir un ensemble minimal de chemins testant toutes les transitions.

Il apparaît deux types de branchement :



Ces schémas sont équivalents en théorie des graphes aux réseaux suivants :



Dans le premier cas, deux chemins seront nécessaires pour parcourir toutes les transitions, dans le second cas il en faudra 4.

Une fois déterminé cet ensemble minimal on déterminera (partiellement) les opérandes nécessaires pour parcourir les chemins choisis.

3.2.6 Organisation générale du test.

La vérification des éléments de mémorisation et le test des fils de commande requiert le parcours d'un certain nombre de chemins de ces différents algorithmes, en imposant des contraintes sévères sur la valeur des paramètres d'entrée.

Ceci permet de tester partiellement le séquençement des phases. Le test complet de ce séquençement est obtenu en parcourant tous les chemins restants de tous les algorithmes (les contraintes étant dûes uniquement au choix des chemins).

3.3 RESULTATS PRATIQUES.

3.3.1 Algorithmes réalisés.

Cet automate de contrôle réalise 10 algorithmes du même type que celui de la Fig. III.12.

Ces algorithmes caractérisent 10 opérations automatiques sur nombres flottants (cf. § 3.1.1.γ).

3.3.2 Structure hardware.

La structure hardware de l'automate de contrôle est du type de celle de la Fig. III.10.

A chaque phase \emptyset_i de l'algorithme est associée une bascule B_i et un ensemble de commandes \mathcal{C}_i .

Cet automate de contrôle comporte environ 200 portes et 27 bascules.

3.3.3 Classification de l'automate.

C'est un automate de contrôle caractérisé par :

- . $Z = \emptyset$, $\{\mathcal{C}_i; \mathcal{C}_i = \lambda(Q_i, X_i, X')\}$
- . une dépendance forte : X' non vide
- . observabilité indirecte
- observabilité immédiate et différée (fonction des U_i).

3.3.4 Résultats pratiques du test.

-La méthode de test proposé au § 3.2. a été utilisée pour vérifier cet organe de gestion des opérations automatiques sur nombre flottants d'une unité arithmétique et logique.

Dans cet exemple précis on a établi que :

- les pannes dans le séquençement des phases ont été testées en parcourant 30 chemins des différents algorithmes.
- parmi ces 30 chemins, 5 ont été nécessaires pour tester les éléments de mémorisation (5 classes ont été définies).
- durant l'exécution de ces algorithmes, on a aussi détecté toutes les pannes sur les fils de commandes et aucun calcul spécial n'a été nécessaire.

Pour l'ensemble de la partie contrôle (215 portes, 21 bascules, 1 compteur) gérant 10 opérations automatiques, on a eu besoin de :

- 100 microinstructions dont le cycle de base est 70 ns
- 340 vecteurs de test (configurations d'entrée X et P, sorties correctes à comparer avec les résultats observés Z') chargés en mémoire centrale.

Cette méthode de test apparaît efficace pour toute partie contrôle directement implémentée à partir d'une famille d'algorithmes.

IV - CONCLUSION.

Deux méthodes ont été données pour le test de parties contrôle locales, accessibles seulement à travers une partie opérative associée et pour lesquelles on ne suppose aucune facilité de test.

Les conclusions qu'on peut en tirer sont les suivantes :

- pour rendre le test efficace, des méthodes diversifiées doivent être élaborées compte tenu des caractéristiques de la structure hardware.
- des exigences précises pourraient être données au niveau de la conception : existence de séquence de distinction particulière dans le premier cas, visualisation des éléments de mémorisation dans le second cas.

Une extension de ce chapitre pourrait être la comparaison vis-à-vis du test, de la rapidité et du coût des solutions microprogrammées et des solutions câblées pour des automates de contrôle.

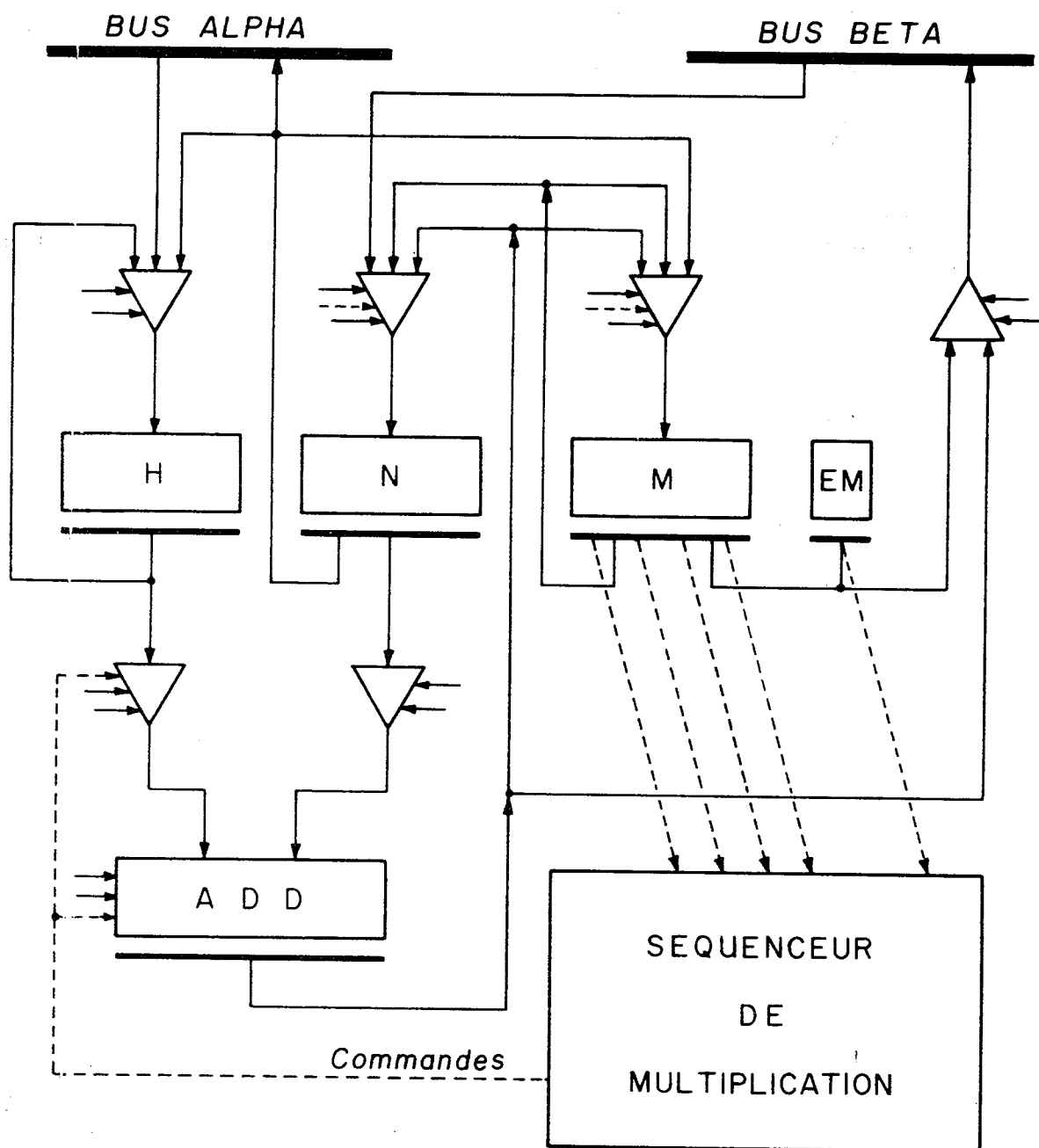


Fig. III.7.

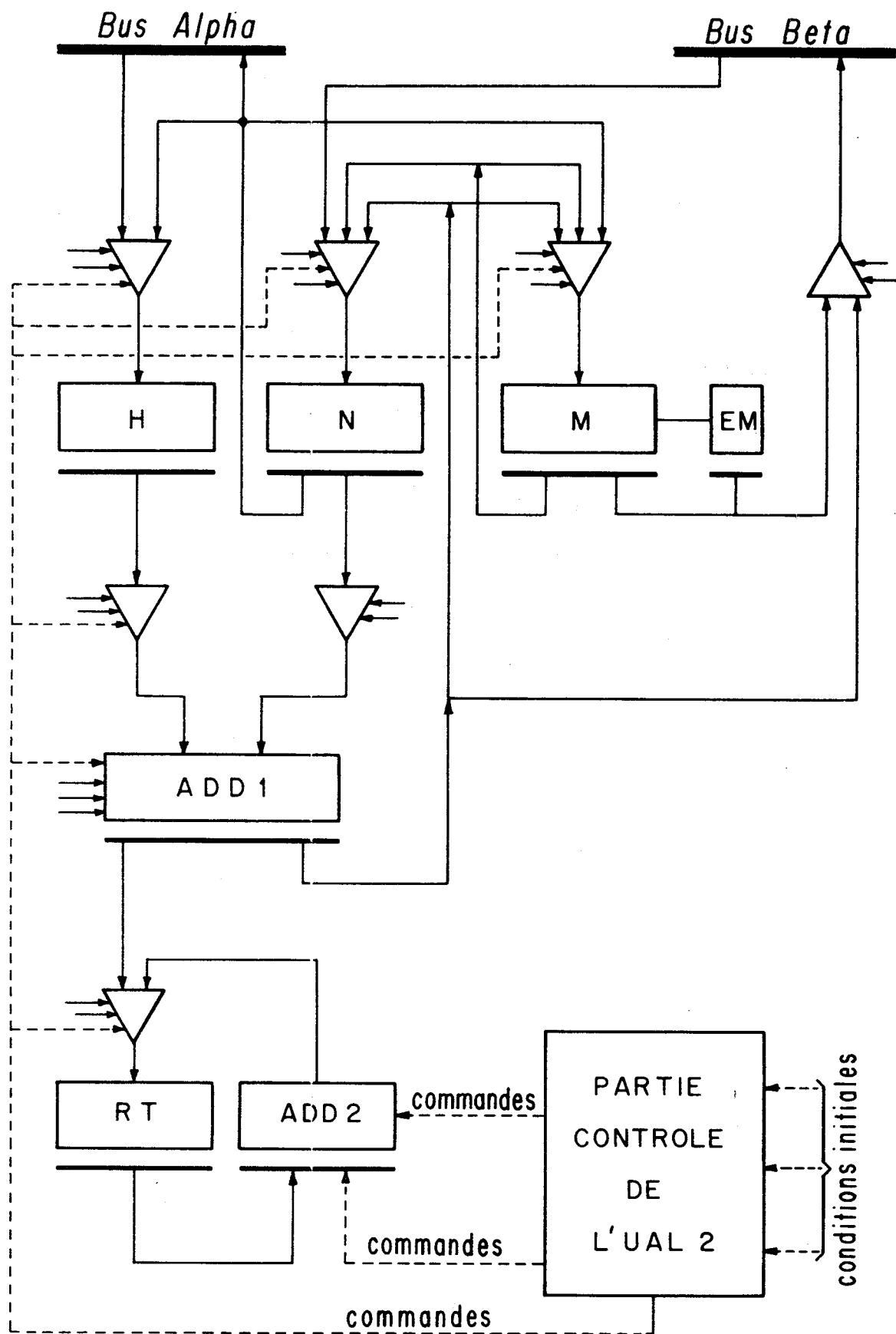


Fig. III.11.

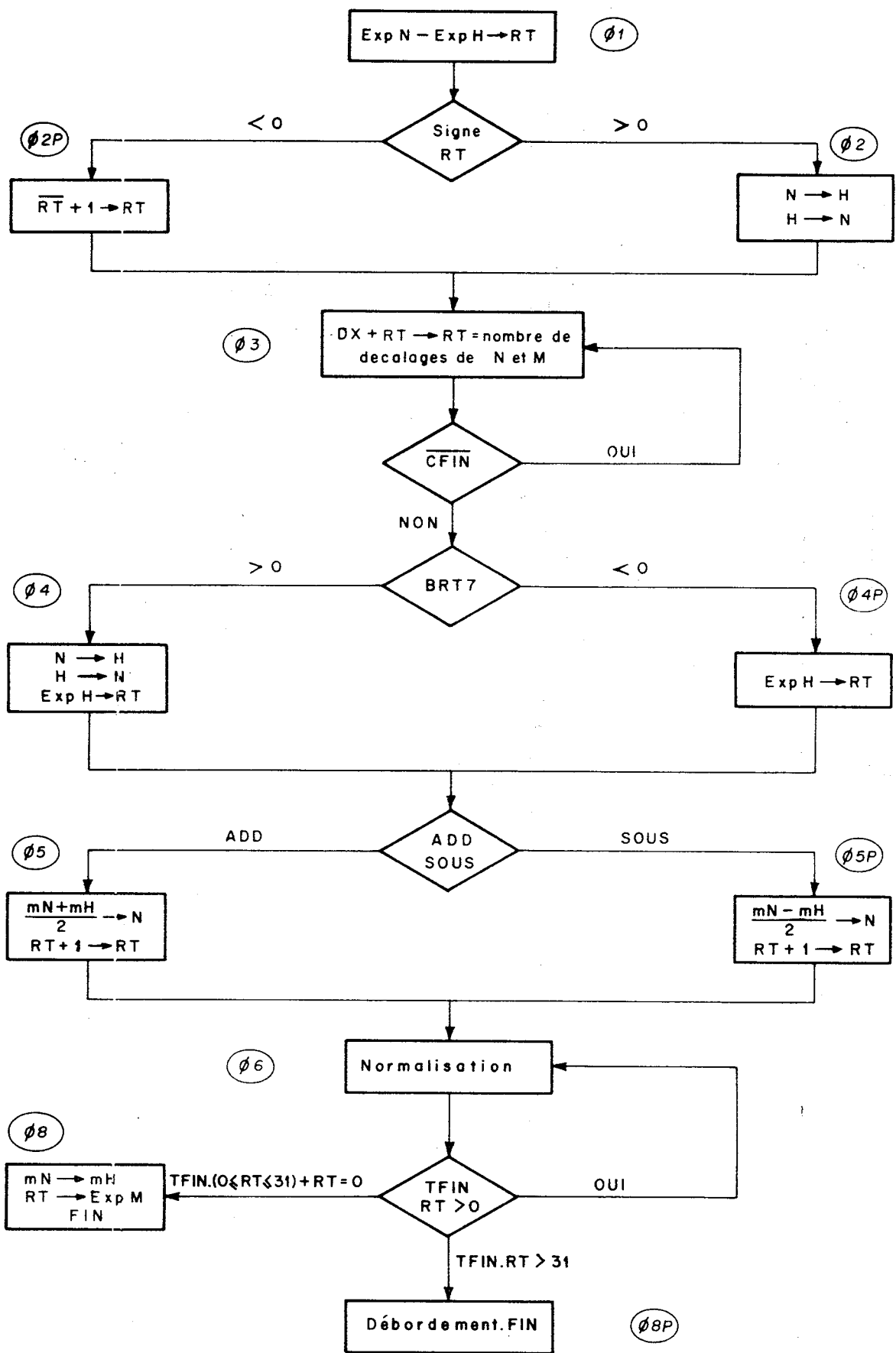


Fig. III.12.

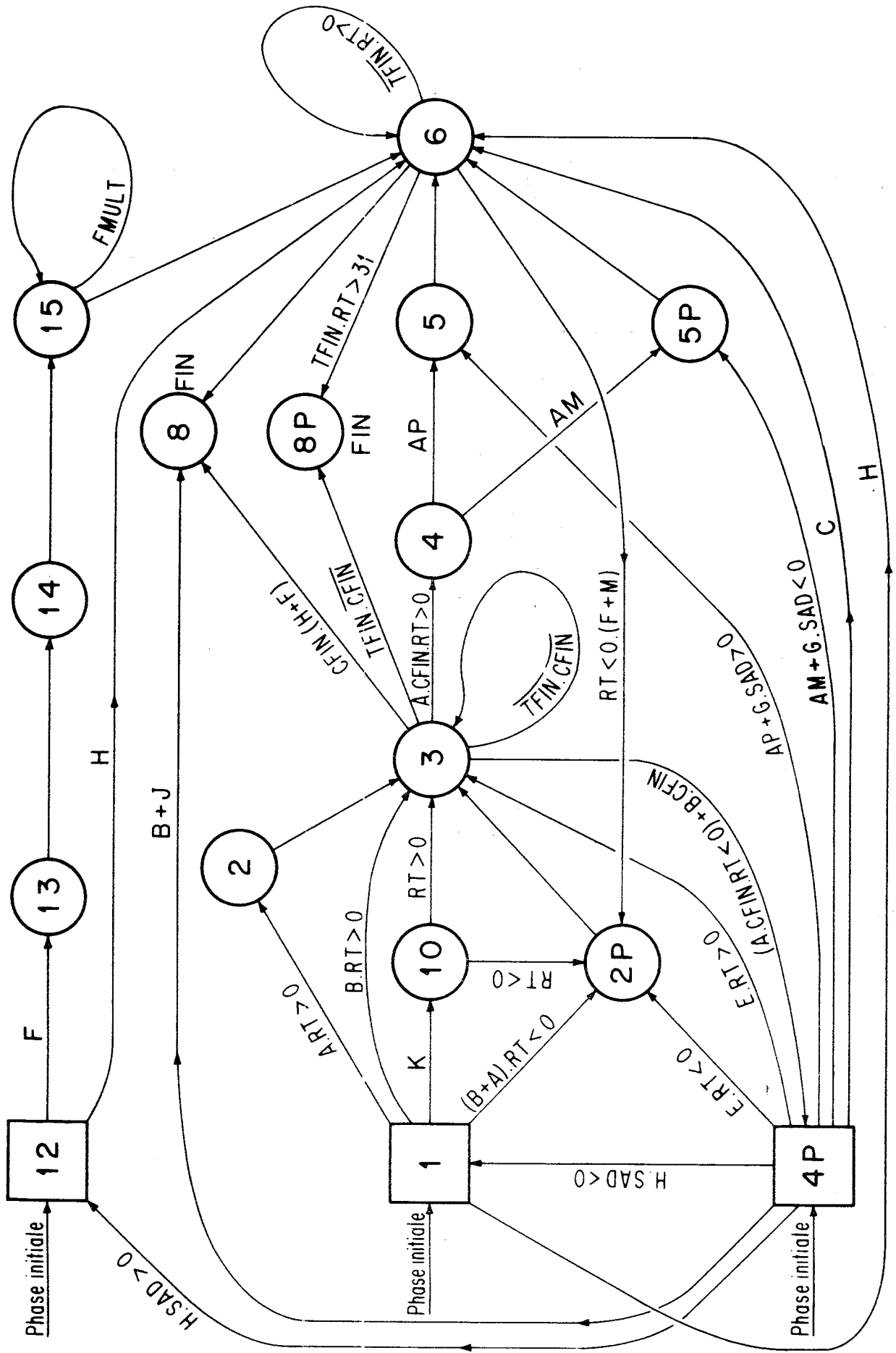


Fig. III.13.

CHAPITRE IV

CONCEPTION D'AUTOMATES DE CONTROLE TOTALEMENT TESTABLES

* * *

INTRODUCTION.

Au cours de la synthèse de systèmes logiques (composants, cartes, ordinateurs, ...) certains modules ont une importance stratégique particulière pour la fiabilité (hardcore, par exemple). On peut soit exiger une fiabilité intrinsèque très importante, soit exiger une testabilité facile et rapide de ces modules.

L'objet de ce chapitre est de préciser cette dernière notion. Les modules considérés seront des automates de contrôle dont l'importance est due à leur rôle de gestion de modules opérationnels de grande complexité.

On a vu dans le chapitre précédent qu'une méthode de test d'un automate de contrôle consiste en l'identification de machines séquentielles connues par leur tableau d'états [35], [57], [58].

Une telle méthode requiert en particulier l'existence d'une séquence de distinction (distinguishing sequence).

Dans le cas où une telle séquence n'existe pas, le problème est alors de savoir si l'on peut concevoir un automate, réalisant les mêmes fonctions, qui posséderait une ou plusieurs séquences de distinction [34].

Dans le cas où l'on a accès aux pouvoirs de décision en cours de synthèse, le problème se pose alors dans les termes suivants : à partir d'un fonctionnement désiré, comment réaliser un automate totalement testable ?

On se placera ici dans le cadre d'automates de contrôle, modélisés sous forme de tableau d'états de petite ou moyenne complexité : on aura donc connaissance du quintuplet $\{X, Q, Y, \delta, \lambda\}$.

Les contraintes d'entrée de cet automate pourront être de l'un quelconque des trois types définis au Chapitre III et ces automates pourront être à observabilité directe ou indirecte.

L'étude présentée ici, définit une méthode efficace indiquant la quantité minimum de hardware à rajouter aux circuits initiaux (sous forme de sorties supplémentaires) en vue de rendre la machine considérée totalement testable [48].

Les résultats proposés sont :

α) un algorithme de résolution permettant de trouver le nombre minimum de sorties supplémentaires à adjoindre à un automate pour le rendre totalement testable.

β) l'optimisation de la séquence de test de l'automate obtenu.

I – SYNTHÈSE ET TESTABILITÉ.

1.1 HYPOTHÈSES PRÉLIMINAIRES.

Les automates considérés sont supposés réduits, fortement connexes et entièrement déterminés.

Ils sont considérés vis-à-vis du test comme des "boîtes noires", ce qui signifie qu'on a uniquement accès aux entrées et aux sorties.

1.2 DÉFINITIONS.

Définition 1 :

On dit qu'un automate est testable lorsqu'il comporte au moins une séquence de distinction ou "distinguishing sequence".

Définition 2 :

Un automate est μ -testable si μ est le plus petit entier tel que toute séquence d'entrée de longueur μ est séquence de distinction. On dira que l'automate est totalement testable (d'ordre μ).

Le problème s'énonce alors comme suit :

α) - étant donné un automate quelconque A, connu par son tableau d'état, trouver le nombre minimal de sorties supplémentaires à rajouter à A pour que A soit totalement testable.

En vue de réalisations pratiques, il convient de minimiser ce nombre et dans un deuxième temps, d'optimiser μ .

β) - comment tester un automate totalement testable ?

La méthode générale d'élaboration d'une séquence de test sera considérablement simplifiée et on notera une amélioration sensible de la longueur du test par rapport aux séquences obtenues par les méthodes connues.

1.3 PRINCIPE GENERAL DE L'ALGORITHME.

1.3.1 Définition des paramètres.

a) N_S : nombre de variables supplémentaires de sortie (variables booléennes) que l'on se propose de rajouter pour rendre l'automate totalement testable.

C'est ce paramètre qu'il importe de minimiser car une sortie supplémentaire se traduit par une augmentation de la circuiterie non négligeable.

b) μ : longueur des séquences de distinction (D.S) considérées.

c) N_μ : nombre de séquences d'entrée de longueur μ , distinctes, qui sont des D.S pour l'automate. Pour un automate ayant x entrées, si $N_\mu = x^\mu$, toutes les séquences d'entrée de longueur μ sont D.S et l'automate est μ -testable.

1.3.2 Exposé général de l'algorithme.

On construit un algorithme déterminant, pour $N_S = 0, 1, \dots$, le nombre N_μ de séquences d'entrée de longueur $\mu = 1, 2, \dots$ qui soient séquences de distinction.

L'observation des tableaux de transition, construits successivement pour les séquences d'entrée de longueur $1 \dots \mu$ donne lieu à des critères : ces critères permettent d'éviter dans certains cas l'incrémementation de $\mu(N_S)$ pour incrémenter directement N_S .

Une fois N_S incrémenté, on cherche un codage de ces sorties supplémentaires tel que toute séquence de longueur μ (μ à déterminer) soit séquence de distinction.

L'algorithme s'arrête lorsque $N_\mu = x^\mu$.

Remarque : il existe des valeurs limites finies de μ et de N_S assurant qu'il n'y a pas de bouclage infini dans l'algorithme.

1.3.3 Organigramme simplifié (Fig. IV.1).

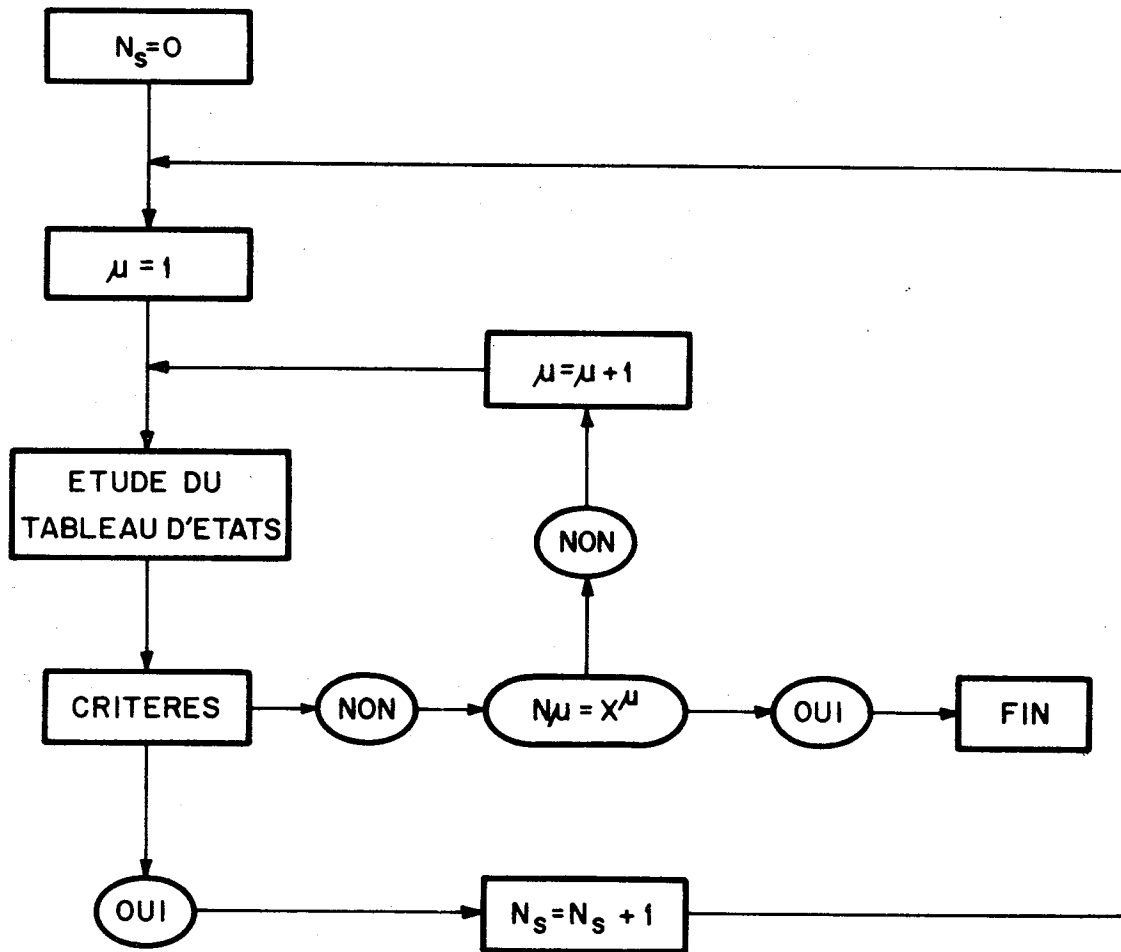


Fig. IV.1.

1.4 PROBLEMES A RESOUDRE.

Cette introduction permet de voir les problèmes qui se posent et que nous nous proposons de résoudre :

α) Existe-t-il des critères qui, au vu du tableau d'états, permettent de déterminer immédiatement s'il est nécessaire d'ajouter des sorties supplémentaires pour rendre l'automate totalement testable ?

Ces critères ont pour but de limiter le nombre d'incrémentations de μ et ils ont déjà partiellement été énoncés, sous une autre forme par Kohavi [34].

β) Existe-t-il une valeur limite finie μ_L de μ , telle que :

- soit $\exists \mu \in [1, \mu_L]$ tel que toute séquence d'entrée de longueur μ soit séquence de distinction (l'automate A est alors μ -testable)
- soit pour une valeur $\mu < \mu_L$ il apparaît au moins un critère, impliquant une sortie supplémentaire.

γ) A-t-on la propriété suivante :

Toute séquence d'entrée de longueur μ comportant comme sous-séquence une séquence de distinction est elle-même une séquence de distinction.

δ) Codage des sorties supplémentaires pour N_S et μ donnés.

II – ETUDE ET RESOLUTION DES PROBLEMES PRELIMINAIRES.

2.1 ETABLISSEMENT DES CRITERES D'ADJONCTION DE SORTIES SUPPLEMENTAIRES.

2.1.1 Critères.

Premier critère :

Pour une séquence d'entrée donnée on cherche la partition "la plus fine possible" en blocs tels que les états d'un même bloc aient :

- . même état successeur
- . même séquence de sortie

Si cette partition n'est pas triviale, il est nécessaire d'ajouter une sortie supplémentaire au moins.

Exemple :

	x_1	x_2	x_k
Q_1	Q_3, y_1	y_k	
Q_2	Q_1, y_1'	y_k'	
Q_3	Q_1, y_1'	y_k'	
Q_4	Q_2, y_1''	y_k''	

Remarque : si un tel bloc comporte k états, le nombre de sorties supplémentaires nécessaire pour dissocier les états de ce bloc sera égal au plus petit entier supérieur à $\log_2 k$ soit $N_S = E(1 + \log_2 k)$.

Deuxième critère :

Pour une séquence d'entrée donnée on cherche la partition "la plus fine possible" en blocs tels que les états d'un même bloc

- . aient même séquence de sortie
- . s'appliquent bijectivement sur eux-mêmes avec la séquence d'entrée considérée

soit $\{\text{états successeurs}\} = \{\text{états du bloc}\}$.

Si cette partition n'est pas triviale, il faut au moins une sortie supplémentaire

Exemple :

	x_1	x_2	$\dots\dots$	x_k
Q_1	Q_3, y_1	$\dots\dots$	y_k	
Q_2	Q_1, y_1	$\dots\dots$	y_k	
Q_3	Q_2, y_1	$\dots\dots$	y_k	
Q_4	Q_5, y_1'	$\dots\dots$	y_k'	
Q_5	Q_6, y_1'	$\dots\dots$	y_k'	
Q_6	Q_4, y_1'	$\dots\dots$	y_k'	
Q_7	Q_1, y_1''	$\dots\dots$	y_k''	

2.1.2 Remarque.

Soit un bloc comportant k états, défini par l'un des deux critères. Si l'on considère toutes les paires d'états de ce bloc, il existe un circuit passant par toutes ces paires d'état (graphe de test).

Les critères incluent les conditions énoncées par Kohavi mais évitent la procédure de recherche de circuits dans un graphe.

2.1.3 Rappels : définition d'un arbre de distinction [35].

Supposons un automate A , donné par son tableau d'états et pouvant être initialement dans l'un quelconque des états $Q_1 \dots Q_n$.

On dira que l'incertitude initiale de A est l'ensemble $(Q_1, Q_2, \dots Q_n)$.

Un vecteur incertitude est un sous-ensemble fini d'états de A comportant n éléments (autant que l'incertitude initiale).

L'arbre successeur est une structure qui montre graphiquement les x_j -successeurs pour chaque entrée x_j possible :

- chaque branche dans l'arbre successeur est associée à un vecteur incertitude
- pour chaque vecteur incertitude il existe autant de branches successeur que d'entrées x_i .
- un vecteur incertitude dont chaque élément contient un seul état est dit "incertitude triviale"
- un vecteur incertitude dont les éléments contiennent des états identiques est dit "incertitude homogène".

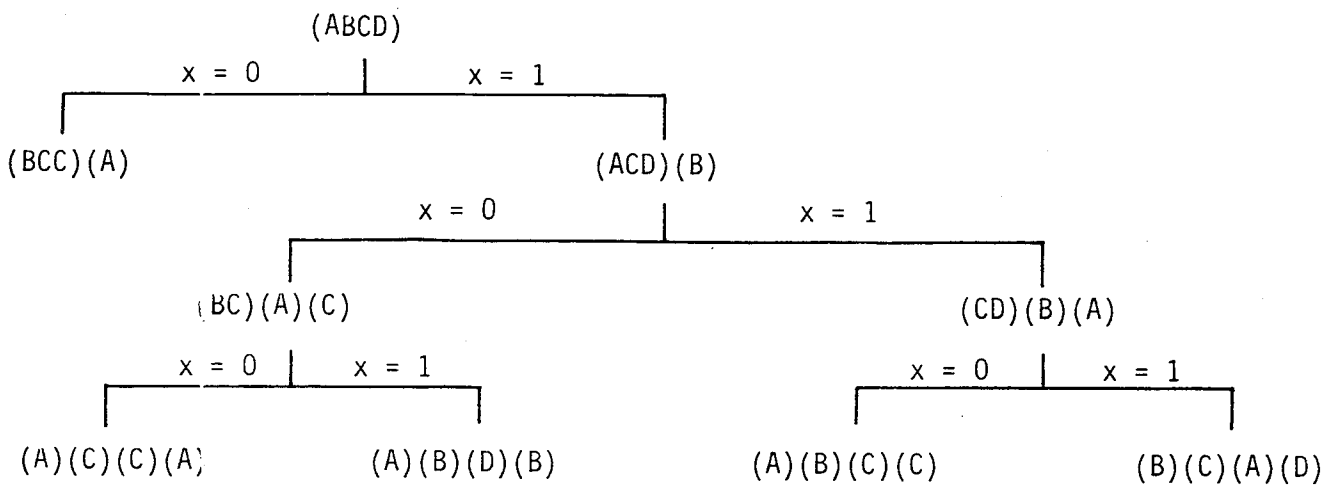
Exemple : Les vecteurs (AA) (C) (B) et (A) (B) (A) (C) sont respectivement homogène et trivial.

L'arbre de distinction est un arbre successeur dont les branches sont terminales dans chacun des cas suivants :

- 1) un vecteur incertitude au niveau k est identique à un vecteur incertitude de niveau $k-1$.
- 2) un vecteur incertitude est homogène.
- 3) un vecteur incertitude est trivial. La séquence d'entrée aboutissant à un tel vecteur est une séquence de distinction.

Exemple : Soit l'automate A et l'arbre de distinction associé

	$x = 0$	$x = 1$
A	C, 0	D, 1
B	C, 0	A, 1
C	A, 1	B, 0
D	B, 0	C, 1



Les séquences d'entrée 100, 101, 110 et 111 sont des séquences de distinction.

2.1.4 Conclusion.

- Lorsque le premier critère n'apparaît pas, cela garantit que les règles 1 ou 2 ne s'appliquent pas
- Lorsque le deuxième critère n'apparaît pas, cela garantit que la règle 1 ne s'applique pas.

2.2 LIMITE SUPERIEURE DE μ

On montre que pour un automate A quelconque il existe une valeur limite finie μ_L telle que, si au pas $\mu = \mu_{L-1}$ on n'a trouvé

- ni critère
- ni A totalement testable

alors

- soit toutes les séquences de sorties sont distinctes pour $\mu = \mu_L$ et A est μ_L -testable
- soit on trouve pour $\mu = \mu_L$ l'un des critères au moins.

Cette limite supérieure est :

$$\mu_L = C_n^2 = \frac{n(n-1)}{2}$$

Démonstration.

En effet, le cas le plus défavorable est celui où, pour une entrée donnée, tous les états de l'automate appartiennent à un même

bloc (dû au deuxième critère).

Ceci équivaut d'après la remarque de § 2.1.2. à un circuit passant par toutes les paires d'états possibles.

Or la longueur maximum d'un tel circuit, pour un automate comportant n états, est :

$$L = C_n^2 = \frac{n(n-1)}{2}$$

Il suffira donc d'avoir $\mu_L = C_n^2$ pour dissocier tous les états de ce bloc [Kohavi].

2.3 PROPRIÉTÉ.

Au cours de l'algorithme énoncé, toute séquence d'entrée considérée comportant comme sous-séquence une D.S (trouvée aux pas précédents) est elle-même séquence de distinction.

Démonstration :

Soit une séquence de distinction de longueur μ :

$$X_1 X_2 \dots X_\mu$$

Étudions les séquences de longueur $(\mu + 1)$ ayant $X_1 X_2 \dots X_\mu$ comme sous-séquence.

Il en existe 4 :

$$\left\{ \begin{array}{l} 0 X_1 X_2 \dots X_\mu \\ 1 X_1 X_2 \dots X_\mu \\ X_1 X_2 \dots X_\mu 1 \\ X_1 X_2 \dots X_\mu 0 \end{array} \right.$$

Si l'on a les tableaux de transition suivants :

	$x = x_1 x_2 \dots x_\mu$	$x = 0$	$x = 1$
A	Q_A, Y_A	A	Q_{1A}, Y_{1A}
B	Q_B, Y_B	B	Q_{1B}, Y_{1B}
\vdots	\vdots	\vdots	\vdots
K	Q_K, Y_K	K	Q_{1K}, Y_{1K}

avec $(Q_A, Q_B, \dots, Q_K) \in \{A, B, \dots, K\}$
 $(Q_{0A}, Q_{0B}, \dots, Q_{0K}) \in \{A, B, \dots, K\}$
 $(Q_{1A}, Q_{1B}, \dots, Q_{1K}) \in \{A, B, \dots, K\}$

et $Y_{0i} \in \{0, 1\}$
 $Y_{1i} \in \{0, 1\}$

Y_i est une suite de symboles binaires de longueur μ , les Y_i étant tous distincts.

a) étude des séquences $X_0 = x_1 x_2 \dots x_\mu$ et $X_1 = x_1 x_2 \dots x_\mu$

Les sorties respectives peuvent s'écrire :

$$Y_i(X_0) = Y_i \& Y_{0i}$$

$$Y_i(X_1) = Y_i \& Y_{1i}$$

or $\forall i, \forall i'; i \neq i' \Rightarrow Y_i \neq Y_{i'}$

Par conséquent, $Y_i(X_0) \neq Y_{i'}(X_0)$

et $Y_i(X_1) \neq Y_{i'}(X_1)$

b) étude des séquences $0X = 0x_1 \dots x_\mu$ et $1X = 1x_1 \dots x_\mu$

On ne pourrait observer des sorties identiques pour deux états Q_A et Q_B que si par l'entrée 0 (respectivement par l'entrée 1) Q_A et Q_B avaient le même état successeur avec la même sortie. Ce serait alors à cet état identique que l'on appliquerait la séquence $X = x_1 \dots x_\mu$.

Or ceci est contraire à l'hypothèse que le premier critère n'est pas apparu à un stade antérieur.

III – CODAGE DES SORTIES SUPPLEMENTAIRES.

3.1 PRECODAGE ET CODAGE.

Pour un pas donné : $N_S = i$, $\mu = j$, de l'algorithme on procède en deux temps :

Etape α :

On cherche si les deux critères énoncés s'appliquent.

- s'ils ne s'appliquent pas on passe à l'étape β .
- s'ils s'appliquent, on détermine l'ensemble des blocs d'états à dissocier, c'est-à-dire les partitions générées par les critères et on ajoute une sortie ($N_S = i+1$).

On ajoute ces blocs aux blocs non encore dissociés préalablement. Soit $\{\mathcal{B}_{ij}(E_k)\}$ l'ensemble de ces blocs (k décrivant l'ensemble des sorties).

On cherche alors un précodage de cette sortie, dissociant les états de ces blocs. S'il n'existe pas de solution dissociant tous les blocs, on dissocie alors le plus de blocs possible, on incrémente μ , on met à jour $\{\mathcal{B}_{ij+1}\}$ et on retourne à l'étape α . Si une solution existe on va en β .

Etape β :

On cherche à achever le codage en distinguant toutes les séquences de sortie non encore comparées précédemment (pour une même séquence d'entrée).

Dans cette étape on sera donc dispensé de comparer les séquences de sortie des états d'un bloc dissocié. S'il n'y a pas de solution on passe au pas suivant (incrémementation de μ : $\mu = j+1$, étape α).

Remarque 1 : on ne considèrera pas les séquences d'entrée éliminées par la propriété du § 2.3.

Remarque 2 : la dissociation du codage en deux étapes : précodage et codage, tend à simplifier le problème du codage en résolvant d'abord les impossibilités immédiatement déterminées par les critères.

3.2 ETAPE α : PRECODAGE.

ou dissociation des états d'un bloc par calcul propositionnel.

Exemple : soit l'automate A :

	x = 0	x = 1
A	B, 0	D, 0
B	A, 0	B, 0
C	D, 1	A, 0
D	D, 1	C, 0
E	A, 0	F, 1
F	C, 1	B, 1

$$N_S = 0$$

$$\mu = 1$$

les deux critères s'appliquent : on détermine l'ensemble des blocs d'états à dissocier (étape α) :

$$\mathcal{B}_{01}(x = 0) = \{(AB), (CD)\}$$

$$\mathcal{B}_{01}(x = 1) = \{(ABCD)\}$$

$$N_S = 1$$

$$\bullet \mu = 1$$

$$\mathcal{B}_{11} = \mathcal{B}_{01}$$

- . blocs dissociables : (AB) (x = 0), (CD) (x = 0).
- . blocs non dissociables (ABCD) (x = 1)
- . précodage partiel (étape α)

	x = 0	x = 1
A	B, 0 s ₀₁	D, 0 s ₁₁
B	A, 0 s ₀₂	B, 0 s ₁₂
C	D, 1 s ₀₃	A, 0 s ₁₃
D	D, 1 s ₀₄	C, 0 s ₁₄
E	A, 0 s ₀₅	F, 1 s ₁₅
F	C, 1 s ₀₆	B, 1 s ₁₆

dissociation de (AB) : $s_{02} = \overline{s_{01}}$

dissociation de (CD) : $s_{04} = \overline{s_{03}}$

• $\underline{\mu = 2}$

$$\mathcal{B}_{12} = \{(ABCD)_{x=1}\}$$

On cherche à dissocier le bloc (ABCD) par la séquence d'entrée $x = 11$ (étape α).

	$x = 00$	$x = 01$	$x = 11$	$x = 10$
A			C, $0s_{11}$ $0s_{14}$	
B			B, $0s_{12}$ $0s_{12}$	
C			D, $0s_{13}$ $0s_{11}$	
D			A, $0s_{14}$ $0s_{13}$	
E
F

Les blocs hachurés correspondent aux états dissociés.

Résolution par calcul propositionnel :

Ecrivons que les sorties doivent être toutes distinctes :

La méthode est la suivante :

Nous comparerons les sorties deux à deux; par exemple, soient deux sorties à comparer :

a b c d

α β γ δ

pour que ces sorties soient distinctes, on écrit l'expression suivante :

$$(a \neq \alpha) \text{ ou } (b \neq \beta) \text{ ou } (c \neq \gamma) \text{ ou } (d \neq \delta)$$

. Si on obtient une expression de la forme

$$(a \neq 0) \text{ ou } (0 \neq 0) \text{ ou } (b \neq \gamma)$$

ou

$$(a \neq 0) \text{ ou } (1 \neq 1) \text{ ou } (b \neq \gamma)$$

cette expression se réduit à $(a \neq 0) \text{ ou } (b \neq \gamma)$

. Si on obtient une expression de la forme

$$(a \neq 0) \text{ ou } (0 \neq 1) \text{ ou } (b \neq \gamma)$$

alors les sorties comparées sont distinctes et n'impliquent pas de contraintes sur la sortie supplémentaire.

Une fois obtenue l'expression simplifiée, on affecte à chaque proposition $(a \neq \alpha)$ une variable booléenne.

$$(a \neq \alpha) = A$$

$$(b \neq \beta) = B$$

$$(c \neq \gamma) = C$$

$$(d \neq \delta) = D$$

Pour l'ensemble des couples de sorties comparées on obtient une équation booléenne de la forme :

$$(A + B + C + D) (\dots\dots\dots) (\dots\dots) = 1$$

après réduction, cette équation donne toutes les solutions possibles dans la mesure où il n'y a pas d'incompatibilité.

Dans l'exemple précédent, l'application de cette méthode donne, après simplification :

$$(s_{12} \neq s_{11}) \text{ ou } (s_{12} \neq s_{14})$$

$$(s_{13} \neq s_{11}) \text{ ou } (s_{11} \neq s_{14})$$

$$(s_{14} \neq s_{11}) \text{ ou } (s_{13} \neq s_{14})$$

$$(s_{13} \neq s_{12}) \text{ ou } (s_{11} \neq s_{12})$$

$$(s_{14} \neq s_{12}) \text{ ou } (s_{13} \neq s_{12})$$

$$(s_{14} \neq s_{13}) \text{ ou } (s_{13} \neq s_{11})$$

on affecte à chaque proposition une variable booléenne.

$$(s_{12} \neq s_{11}) = A \qquad (s_{13} \neq s_{14}) = E$$

$$(s_{12} \neq s_{14}) = B \qquad (s_{13} \neq s_{12}) = F$$

$$(s_{13} \neq s_{11}) = C$$

$$(s_{11} \neq s_{14}) = D$$

l'équation booléenne s'écrit :

$$(A + B) (C + D) (D + E) (F + A) (B + F) (E + C) = 1$$

soit, après réduction :

$$ABDE + ABCD + ABCE + ACDF + ACEF + BDEF + BCDF + BCEF + ADEF = 1$$

Chacun de ces monômes est une solution possible.

Par exemple : ADEF :

$$\left\{ \begin{array}{l} s_{12} \neq s_{11} \\ s_{11} \neq s_{14} \\ s_{13} \neq s_{14} \\ s_{13} \neq s_{12} \end{array} \right.$$

soit

$$\left\{ \begin{array}{l} s_{12} = \overline{s_{11}} \\ s_{14} = \overline{s_{11}} \\ s_{13} = s_{11} \end{array} \right.$$

Le bloc (ABCD) est donc dissocié, avec le codage précédent. Les critères ne s'appliquent plus, on va en étape β .

3.3 ETAPE β : CODAGE COMPLET DES SORTIES SUPPLEMENTAIRES.

La méthode par calcul propositionnel s'applique au cours de l'étape α puis, au cours de l'étape β avec la restriction suivante : deux états ne seront dissociés que s'ils n'appartiennent pas à un même bloc $\mathcal{B}_{ij}(E_k)$.

Nous pouvons reprendre l'exemple précédemment donné en le résolvant entièrement.

Il reste donc à résoudre l'étape β

On a obtenu le tableau de transition suivant :

	x = 00	x = 01	x = 11	x = 10
A	A, $0s_{01}$ $0\overline{s_{01}}$	B, $0s_{01}$ $0\overline{s_{11}}$	C, $0s_{11}$ $0\overline{s_{11}}$	D, $0s_{11}$ $1\overline{s_{03}}$
B	B, $0\overline{s_{01}}$ $0s_{01}$	D, $0\overline{s_{01}}$ $0s_{11}$	B, $0\overline{s_{11}}$ $0\overline{s_{11}}$	A, $0\overline{s_{11}}$ $0\overline{s_{01}}$
C	D, $1s_{03}$ $1\overline{s_{03}}$	C, $1s_{03}$ $0\overline{s_{11}}$	D, $0s_{11}$ $0s_{11}$	B, $0s_{11}$ $0s_{01}$
D	D, $1\overline{s_{03}}$ $1s_{03}$	C, $1\overline{s_{03}}$ $0s_{11}$	A, $0\overline{s_{11}}$ $0s_{11}$	D, $0\overline{s_{11}}$ $1s_{03}$
E	B, $0s_{05}$ $0s_{01}$	D, $0s_{05}$ $0s_{11}$	B, $1s_{15}$ $1s_{16}$	C, $1s_{15}$ $1s_{06}$
F	D, $1s_{06}$ $1s_{03}$	A, $1s_{06}$ $0s_{11}$	B, $1s_{16}$ $0\overline{s_{11}}$	A, $1s_{16}$ $0\overline{s_{01}}$

Comme indiqué en § 3.1., on est dispensé de la comparaison des séquences de sortie des états des blocs dissociés (blocs soulignés sur le tableau). On achève donc le codage (par calcul propositionnel) en distinguant toutes les séquences de sortie non encore comparées.

x = 00	x = 01	x = 11	x = 10
$s_{05} \neq \overline{s_{01}}$	$s_{05} \neq \overline{s_{01}}$		
$s_{06} \neq \overline{s_{03}}$			

$$(s_{05} \neq \overline{s_{01}}) = A$$

$$(s_{06} \neq \overline{s_{03}}) = B$$

L'équation booléenne s'écrit : $(A \cdot B) \cdot A = 1$ soit $AB = 1$

On a donc la solution unique :

$$s_{05} = s_{01}$$

$$s_{06} = s_{03}$$

M est 2-testable (ou totalement testable d'ordre 2) avec 1 sortie supplémentaire

Son tableau d'états s'écrit alors :

	$x = 0$	$x = 1$
A	B, $0s_{01}$	D, $0s_{11}$
B	A, $0\overline{s_{01}}$	B, $0\overline{s_{11}}$
C	D, $1s_{03}$	A, $0s_{11}$
D	D, $1\overline{s_{03}}$	C, $0\overline{s_{11}}$
E	A, $0s_{01}$	F, $1s_{15}$
F	C, $1s_{03}$	B, $1s_{16}$

Les variables booléennes s_{01} , s_{03} , s_{11} , s_{15} et s_{16} peuvent être choisies arbitrairement ou suivant des critères non considérés ici.

IV – ORGANIGRAMME.

4.1 ORGANIGRAMME.

L'organigramme proposé présente les caractéristiques générales du macro-organigramme simplifié du § I.3.3.

La différence essentielle tient dans l'incrémentation de μ (étape β).

Supposons qu'au pas $\mu = k$ on ait trouvé N séquences de distinction DS_1, \dots, DS_N , imposant certaines contraintes sur le choix du pré-codage des sorties supplémentaires (par calcul propositionnel).

Au pas $\mu = k+1$ on étudie uniquement les séquences d'entrée n'admettant pas DS_1, \dots, DS_N comme sous-séquences. Si cette étude introduit des contraintes incompatibles avec celles introduites au pas $\mu = k$ (par le pré-codage) on refait alors une étude complète des x^{k+1} séquences d'entrée ($x =$ nombre d'entrées de l'automate initial considéré)

Remarque :

Le circuit séquentiel considéré, devant être en général facilement testable, le caractère prioritaire de cet algorithme sera l'obtention du nombre minimum de sorties supplémentaires, quel que soit le temps passé à calculer ces sorties. Ceci explique l'exhaustivité de cet algorithme.

L'organigramme détaillé est donné en Fig. IV.2.

4.2 EXEMPLE.

Soit l'automate A comportant 6 états et 2 entrées

	$X_1 = 0$	$X_2 = 1$
A	C, 0	C, 0
B	D, 0	E, 0
C	E, 0	A, 1
D	F, 0	F, 0
E	A, 1	B, 1
F	B, 1	D, 1

Cette machine est fortement connexe et entièrement déterminée.
L'algorithme s'applique comme suit :

$$\boxed{N_S = 0} \quad \mu = 1$$

$$\mathcal{B}_{01} = \{\emptyset\}$$

Aucun critère ne s'applique

X_1 et X_2 ne sont pas des séquences de distinction; on incrémente donc μ .

$$\mu = 2$$

Le tableau d'états considéré est le suivant :

	00	01	10	11
A	E, 00	A, 01	E, 00	A, 01
B	F, 00	F, 00	A, 01	B, 01
C	A, 01	B, 01	C, 10	C, 10
D	B, 01	D, 01	B, 01	D, 01
E	C, 10	C, 10	D, 10	E, 10
F	D, 10	E, 10	F, 10	F, 10

Le deuxième critère s'applique;

On détermine l'ensemble des blocs à dissocier

$$\mathcal{B}_{02}(01) = \{(AD)\}$$

$$\mathcal{B}_{02}(10) = \{(CF)\}$$

$$\mathcal{B}_{02}(11) = \{(ABD), (CEF)\}$$

$$N_S = 1$$

On cherche une solution dissociant les états de tous les blocs (précodage) définis par \mathcal{D}_{02} (étape α)

Précodage :

	$X_1 = 0$	$X_2 = 1$
A	C, $0s_{11}$	C, $0s_{21}$
B	D, $0s_{12}$	E, $0s_{22}$
C	E, $0s_{13}$	A, $1s_{23}$
D	F, $0s_{14}$	F, $0s_{24}$
E	A, $1s_{15}$	B, $1s_{25}$
F	B, $1s_{16}$	D, $1s_{26}$

Dissociation des états des blocs pour les entrées 01, 10, 11 :

	01	10	11
A	A, $0s_{11}$ $1s_{23}$	E, $0s_{21}$ $0s_{13}$	A, $0s_{21}$ $1s_{23}$
B	F, $0s_{12}$ $0s_{24}$	A, $0s_{22}$ $1s_{15}$	B, $0s_{22}$ $1s_{25}$
C	B, $0s_{13}$ $1s_{25}$	C, $1s_{23}$ $0s_{11}$	C, $1s_{23}$ $0s_{21}$
D	D, $0s_{14}$ $1s_{26}$	B, $0s_{24}$ $1s_{16}$	D, $0s_{24}$ $1s_{26}$
E	C, $1s_{15}$ $0s_{21}$	D, $1s_{25}$ $0s_{12}$	E, $1s_{25}$ $0s_{22}$
F	E, $1s_{16}$ $0s_{22}$	F, $1s_{26}$ $0s_{14}$	F, $1s_{26}$ $0s_{24}$

Résolution par calcul propositionnel :

- . bloc (AD) : $s_{11} \neq s_{14}$ ou $s_{23} \neq s_{26}$
- . bloc (CF) : $s_{23} \neq s_{26}$ ou $s_{11} \neq s_{14}$
- . bloc (ABD) : $s_{21} \neq s_{22}$ ou $s_{23} \neq s_{25}$
 $s_{21} \neq s_{24}$ ou $s_{23} \neq s_{26}$
 $s_{22} \neq s_{24}$ ou $s_{25} \neq s_{26}$

- . bloc (CEF) : $s_{23} \neq s_{25}$ ou $s_{21} \neq s_{22}$
 $s_{23} \neq s_{26}$ ou $s_{21} \neq s_{24}$
 $s_{25} \neq s_{26}$ ou $s_{22} \neq s_{24}$

On affecte à chaque proposition une variable booléenne :

$$\begin{aligned}
 (s_{11} \neq s_{14}) = A & & (s_{21} \neq s_{22}) = C & & (s_{21} \neq s_{24}) = E \\
 (s_{23} \neq s_{26}) = B & & (s_{23} \neq s_{25}) = D & & (s_{22} \neq s_{24}) = F \\
 & & & & (s_{25} \neq s_{26}) = G
 \end{aligned}$$

L'équation booléenne s'écrit :

$$(A + B) (C + D) (E + B) (F + G) = 1$$

soit encore

$$(AE + B) (C + D) (F + G) = 1$$

Soit une solution possible : BCF

$$\begin{aligned}
 \text{soit } s_{23} \neq s_{26} \\
 s_{21} \neq s_{22} \\
 s_{22} \neq s_{24}
 \end{aligned}$$

$$\begin{aligned}
 s_{26} &= \overline{s_{23}} \\
 s_{22} &= \overline{s_{21}} \\
 s_{24} &= \overline{s_{22}} = s_{21}
 \end{aligned}$$

Les blocs $(AD)_{01}$, $(CF)_{10}$, $(ABD)_{11}$ et $(CEF)_{11}$ sont dissociés; les critères ne s'appliquent plus et on passe à l'étape β (codage)

Etape β : codage complet des sorties supplémentaires.

On a donc le tableau d'états suivant :

	00	01	10	11
A	E, 0s ₁₁ 0s ₁₃	A, 0s ₁₁ 1s ₂₃	E, 0s ₂₁ 0s ₁₃	A, 0s ₂₁ 1s ₂₃
B	F, 0s ₁₂ 0s ₁₄	F, 0s ₁₂ 0s ₂₁	A, 0 $\overline{s_{21}}$ 1s ₁₅	B, 0 $\overline{s_{21}}$ 1s ₂₅
C	A, 0s ₁₃ 1s ₁₅	B, 0s ₁₃ 1s ₂₅	C, 1s ₂₃ 0s ₁₁	C, 1s ₂₃ 0s ₂₁
D	B, 0s ₁₄ 1s ₁₆	D, 0s ₁₄ 1 $\overline{s_{23}}$	B, 0s ₂₁ 1s ₁₆	D, 0s ₂₁ 1 $\overline{s_{23}}$
E	C, 1s ₁₅ 0s ₁₁	C, 1s ₁₅ 0s ₂₁	D, 1s ₂₅ 0s ₁₂	E, 1s ₂₅ 0s ₂₁
F	D, 1s ₁₆ 0s ₁₂	E, 1s ₁₆ 0 $\overline{s_{21}}$	F, 1 $\overline{s_{23}}$ 0s ₁₄	F, 1 $\overline{s_{23}}$ 0s ₂₁

Il reste donc à écrire que pour chaque entrée, les sorties sont toutes distinctes (pour les états n'appartenant pas aux blocs dissociés)

$$\begin{aligned}
 X = 00 & \quad s_{11} \neq s_{12} \text{ ou } s_{13} \neq s_{14} \text{ (AB)} \\
 & \quad s_{13} \neq s_{14} \text{ ou } s_{15} \neq s_{16} \text{ (CD)} \\
 & \quad s_{15} \neq s_{16} \text{ ou } s_{11} \neq s_{12} \text{ (EF)}
 \end{aligned}$$

$$X = 01 \quad \begin{array}{l} s_{13} \neq s_{11} \text{ ou } s_{25} \neq s_{23} \\ s_{13} \neq s_{14} \text{ ou } s_{25} \neq s_{23} \end{array} \quad (\text{ACD})$$

$$X = 10 \quad \begin{array}{l} s_{25} \neq s_{23} \text{ ou } s_{12} \neq s_{11} \\ s_{25} \neq s_{23} \text{ ou } s_{12} \neq s_{14} \end{array} \quad (\text{CEF})$$

On résoud par calcul propositionnel en affectant une variable booléenne à chaque proposition :

$$\begin{array}{ll} (s_{11} \neq s_{12}) = A & (s_{13} \neq s_{11}) = D \\ (s_{13} \neq s_{14}) = B & (s_{25} \neq s_{23}) = E \\ (s_{15} \neq s_{16}) = C & (s_{12} \neq s_{14}) = F \end{array}$$

L'équation booléenne s'écrit :

$$(A + B) (B + C) (C + A) (D + E) (B + \bar{E}) (E + A) (\bar{E} + F) = 1$$

Soit après réduction :

$$\boxed{ABEF + BCEF + AB\bar{D}\bar{E} + AC\bar{D}\bar{E} + ABDF = 1}$$

Parmi ces 5 solutions possibles on peut en choisir une, par exemple ABDF

soit

$$\begin{array}{l} s_{11} \neq s_{12} \\ s_{13} \neq s_{14} \\ s_{13} \neq s_{11} \\ s_{12} \neq s_{14} \end{array}$$

$$\boxed{\begin{array}{l} s_{12} = \overline{s_{11}} \\ s_{13} = \overline{s_{11}} \\ s_{14} = \overline{s_{13}} = s_{11} \end{array}}$$

Le tableau d'états s'écrit alors :

	X = 0	X = 1
A	C, 0s ₁₁	C, 0s ₂₁
B	D, 0s ₁₁	E, 0s ₂₁
C	E, 0s ₁₁	A, 1s ₂₃
D	F, 0s ₁₁	F, 0s ₂₁
E	A, 1s ₁₅	B, 1s ₂₅
F	B, 1s ₁₆	D, 1s ₂₃

M est 2-testable avec une sortie supplémentaire et les variables de sortie s_{11} , s_{15} , s_{16} , s_{21} , s_{23} et s_{25} peuvent être choisies arbitrairement.

V — ELABORATION DE LA SEQUENCE DE TEST D'UN AUTOMATE μ -TESTABLE.

L'algorithme présenté permet de trouver le nombre minimum de sorties supplémentaires à ajouter à un automate A pour le rendre totale-ment testable.

L'automate A' obtenu admet alors toutes les séquences d'entrée de longueur μ (μ calculé) comme séquences de distinction.

Pour trouver la séquence de test d'un automate μ -testable, il faut et il suffit de passer par toutes les transitions du tableau d'état, à partir d'un état initial donné. Cela revient à parcourir le graphe de transition de l'automate en passant (si cela est possible) une fois et une seule par tous les arcs du graphe et donne une séquence de test optimale (séquence de longueur minimum).

Pour un automate μ -testable ayant X entrées et N états, la longueur minimale de la séquence de test est :

$$L = X.N + \mu$$

(la dernière transition est testée par les μ dernières entrées).

On remarque une amélioration de la séquence de test proprement dite par rapport aux méthodes connues.

Remarque :

On ne considère pas dans cette séquence de test la partie "initialisation" et "vérification du nombre d'états". *On considère la seule vérification des transitions.*

Le problème à ce niveau est donc de parcourir un graphe orienté fortement connexe de façon minimale [14], [37], [40] :

- . soit en passant une fois et une seule par tous les arcs du graphe de transition
- . soit en passant le moins souvent possible par des arcs déjà parcourus.

C'est le problème de recherche d'un chemin eulérien dans un graphe orienté.

Pour un automate à X entrées et N états on recherche donc, s'il existe, un chemin eulérien de longueur X.N.

Condition d'existence d'un chemin eulérien :

Pour qu'un graphe soit eulérien il faut et il suffit que le genre de tous les sommets soit 0 ou que ce genre soit 0 sauf pour deux sommets, l'un étant de genre 1, l'autre de genre -1.

Définition : le genre d'un noeud (ou articulation) est la différence entre le nombre d'entrées et le nombre de sorties de ce noeud.

Remarque : la somme des genres est toujours égale à 0.

Le problème est alors à 2 niveaux :

- . soit il existe un chemin eulérien : donner un algorithme de recherche d'un tel chemin
- . soit il n'existe pas de chemin eulérien : donner un algorithme de recherche du plus court chemin passant par tous les arcs du graphe.

5.1 ALGORITHME DE RECHERCHE D'UN CHEMIN EULERIEN.

On se place ici dans le cas d'existence d'un chemin eulérien et on cherche tous les chemins eulériens du graphe considéré.

5.1.1 Algorithme.

Soit un automate A et son graphe de transition, orienté et fortement connexe (X entrées, N états) :

On utilise une méthode d'énumération lexicographique.

L'algorithme est le suivant :

- ① On numérote chaque arc du graphe
- ② si tous les genres sont 0 on part de l'un quelconque des noeuds du graphe
sinon on part du noeud de genre -1 (l'arrivée se fera sur le noeud de genre +1)

③ On construit un arbre pour chacun des arcs issus du noeud de départ de la manière suivante :

- . la racine de l'arbre est un arc issu du noeud de départ
- . les successeurs d'un arc i seront tous les arcs pouvant être parcourus après cet arc i et qui ne soient pas prédecesseurs dans l'arbre.
- . les feuilles de cet arbre seront obtenues au niveau $X.N$ ($X.N$ étant le nombre d'arcs du graphe) le niveau de départ étant le niveau 1.

④ Les chemins eulériens seront tous les chemins aboutissant aux feuilles de l'arbre, pour chaque arbre construit.

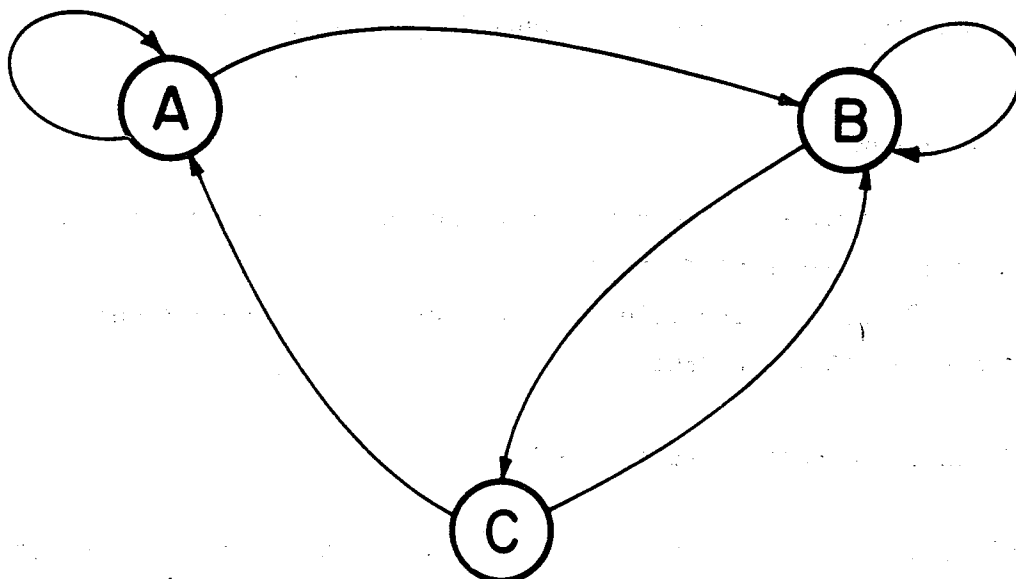
5.1.2 Exemple.

Soit l'automate A ayant

2 entrées : $x = 0, x = 1$

3 états A, B, C.

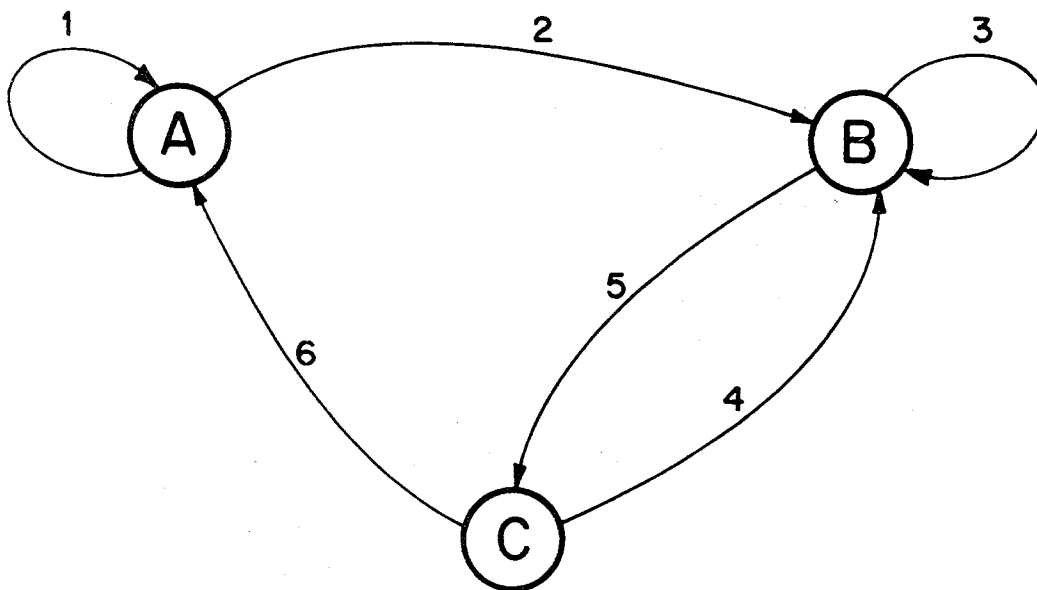
et son graphe de transition.



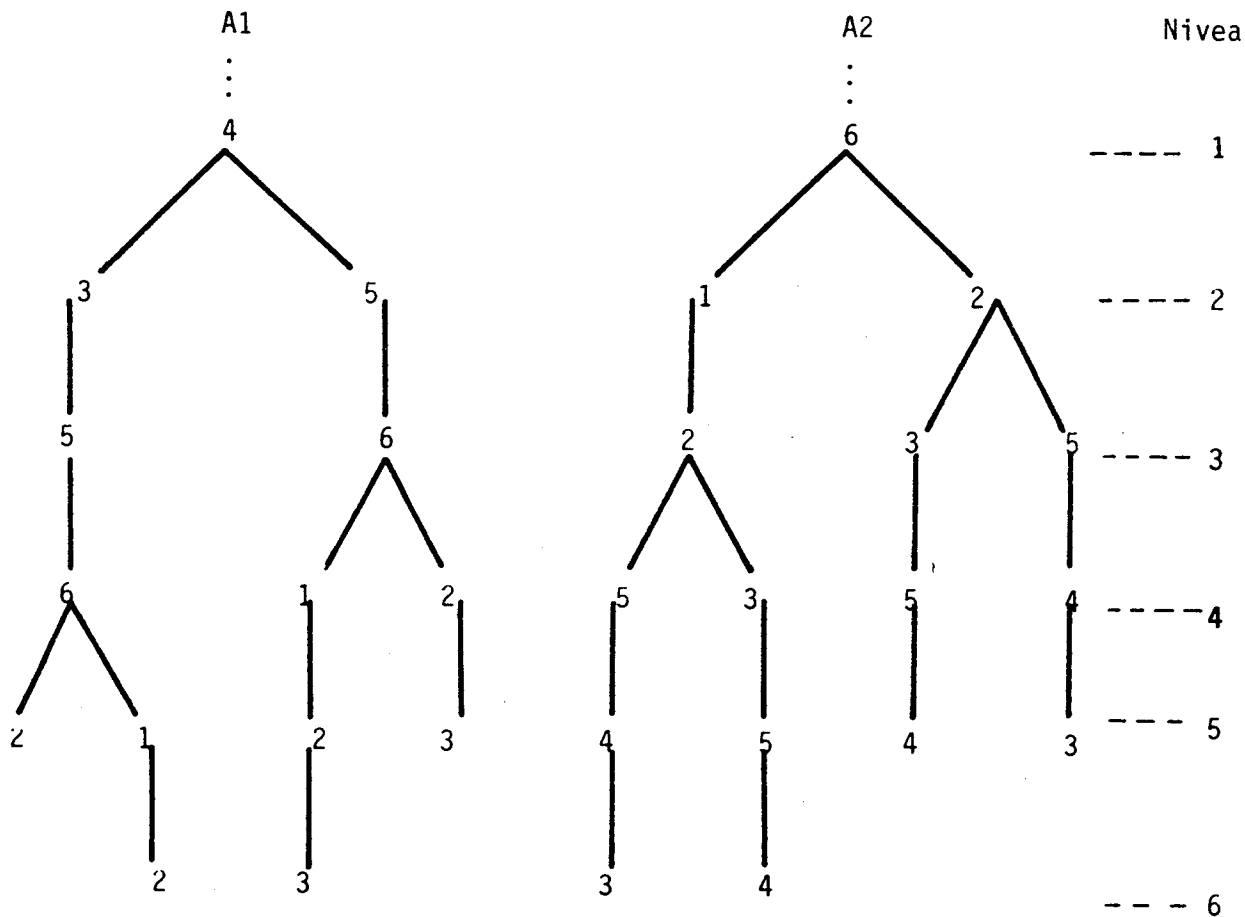
A de genre 0
B de genre +1
C de genre -1

Algorithme :

① On numérote chaque arc du graphe.



- ② Le noeud de départ est le noeud C.
Le noeud d'arrivée sera le noeud B.
- ③ on construit deux arbres A_1 et A_2 de racines 4 et 6 respectivement.
Ces arbres comporteront 6 niveaux (6 étant le nombre de transitions de l'automate A).



L'automate A possède donc 4 chemins eulériens :

4-3-5-6-1-2

4-5-6-1-2-3

6-1-2-5-4-3

6-1-2-3-5-4

5.2 ALGORITHME DE RECHERCHE DU PLUS COURT CHEMIN.

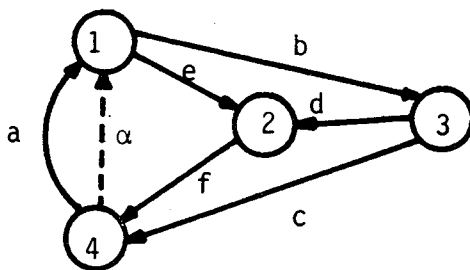
On se place ici dans le cas où il n'existe pas de chemin eulérien

Il s'agit alors de dédoubler certaines connexions existantes pour satisfaire à la condition des genres.

Une fois obtenu ainsi un graphe possédant un chemin eulérien, on applique l'algorithme précédent.

Exemple :

Considérons l'automate donné par son graphe de transition en traits pleins.

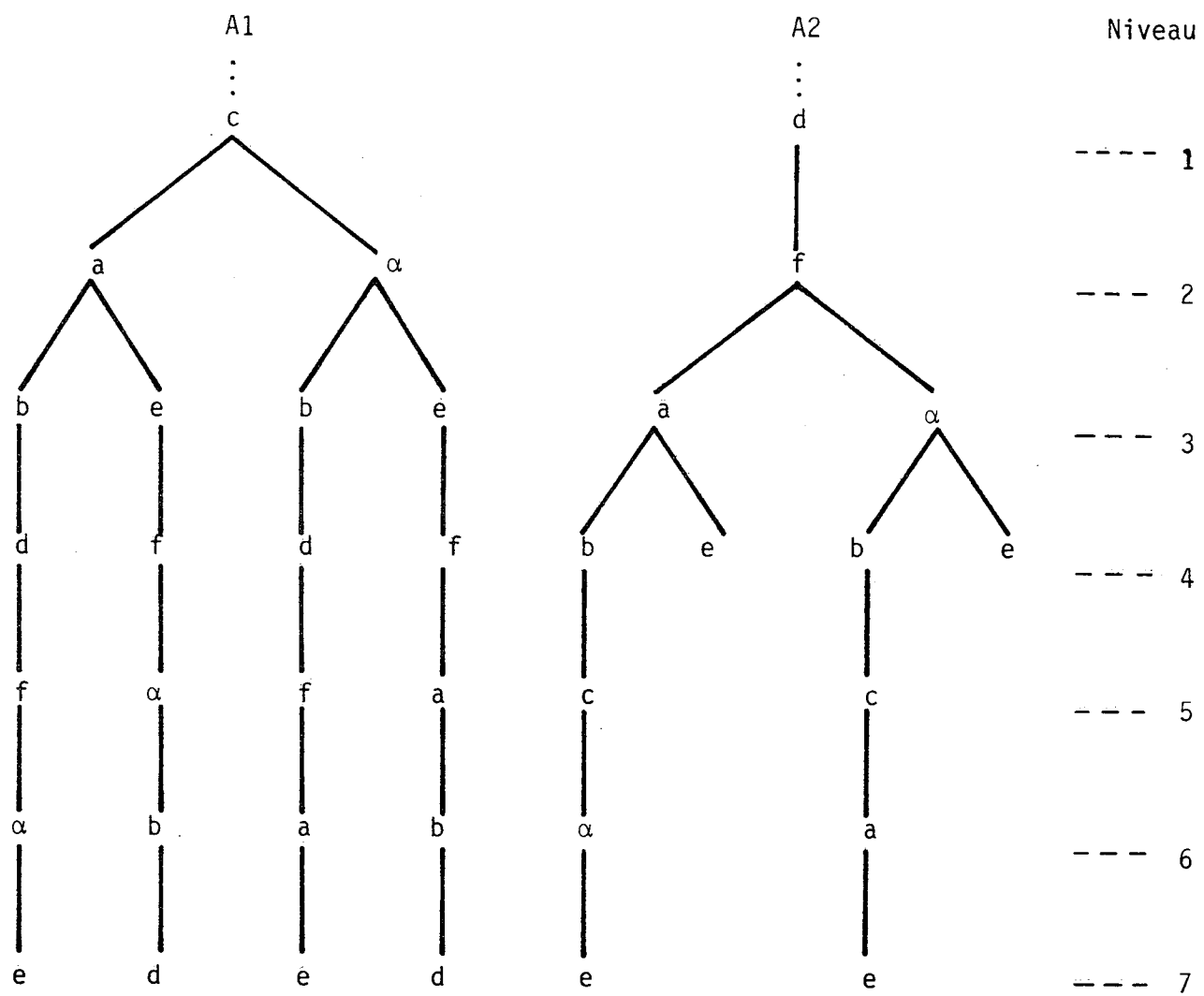


Le sommet	1	est de genre	-1
"	2	" " "	+1
"	3	" " "	-1
"	4	" " "	+1

Rajoutons l'arête en pointillé; les genres deviennent respectivement 0, 1, -1, 0.

Le problème est maintenant possible

- . le noeud de départ est le noeud $\textcircled{3}$
- . on construit deux arbres A_1 et A_1 , de racines c et d respectivement, comme suit :



Le noeud d'arrivée est le noeud (2) .

On obtient ainsi dans le nouveau graphe 6 chemins eulériens :

- c a b d f α e
- c α e f α b d
- c α b d f a e
- c α e f a b d
- d f α b c a e
- d f a b c α e

soit 3 plus courts chemins dans le graphe original :

c a b d f a e

c a e f a b d

d f a b c a e.

CHAPITRE V

ETUDE D'UN EXEMPLE CONCRET : LE GEOPROCESSEUR

* * *

INTRODUCTION.

Le géoprocasseur 46 est un calculateur numérique microprogrammé dont la structure a été conçue par "l'Institut Français du Pétrole" en collaboration avec l'E.N.S.I.M.A.G. Il résulte donc d'une collaboration industrie-université [2], [17].

C'est un calculateur de moyenne importance, destiné spécifiquement à des physiciens, géologues, ... et orienté vers des traitements en temps réel.

Il a été conçu pour effectuer du traitement géophysique dans les agences et les navires de prospection. Ce n'est donc pas un calculateur universel mais un appareil performant dans le domaine du traitement de l'information.

Ses caractéristiques principales sont :

- une capacité mémoire importante
- des organes de calcul rapides
- la rapidité du prélèvement des informations d'entrée
- être relativement puissant
- être particulièrement adapté au traitement du signal : opérations spécialisées (convolution; transformée de Fourier).

I – STRUCTURE HARDWARE DE L'UNITE CENTRALE.

Nous décrivons essentiellement l'unité centrale puisque le micro-programme de test qui a été écrit, est relatif à celle-ci. Cette unité centrale s'organise autour de 2 bus de transfert d'information (ALPHA et BETA) et de 2 bus de contrôle (ATTENTE et TEST). Sur ces bus sont connectés 4 blocs mémoire, 2 unités arithmétiques et logiques, 2 registres (registre d'ETAT et registre INSTRUCTION) et des organes de communication avec l'extérieur.

Cette structure est illustrée par la Fig. V.1.

1.1 LES BLOCS MEMOIRES.

α) **Une mémoire centrale** (M.C) de 4K mots de 21 bits (dont 1 bit de parité), en technologie MOS, extensible par modules de 4K mots jusqu'à 128K.

β) **Une mémoire locale** (M.L) à registres rapides de 16 mots de 20 bits, comprenant entre autres :

- un accumulateur et son extension
- 3 registres de base
- 3 registres d'index
- un compteur ordinal
- une adresse retour micro-programme.

Neuf de ces registres sont adressables par programme, les autres étant réservés au microprogramme.

γ) **Une mémoire opérateur** (M.O) de 1K mots de 20 bits pour l'utilisation spécifique de l'ordinateur.

Elle contient

- l'opérateur dans l'opération de convolution
- les tables de sinus et cosinus en opération transformée de Fourier
- les descripteurs des canaux multiplexés

δ) **Une mémoire de microprogramme** (M. μ) ou mémoire de contrôle qui fournit les commandes aux différents sous-ensembles. Cette mémoire est une mémoire morte de 2K mots de 48 bits; accompagnée de son système d'adressage elle constitue l'organe de gestion de l'unité centrale.

1.2 LES UNITES ARITHMETIQUES ET LOGIQUES.

Elles sont au nombre de 2 et marchent en parallèle.

- α) **L'UAL 1** est une unité arithmétique et logique classique, opérant sur des nombres entiers et comportant une gestion automatique de la multiplication.
- β) **L'UAL 2** permet d'effectuer :
- d'une part, des opérations arithmétiques sur nombres entiers
 - d'autre part, des opérations automatiques sur nombres flottants
addition-soustraction, multiplication, normalisation, ...

1.3 LES REGISTRES.

α) **Un registre instruction (R.I)** de 20 bits sert à l'analyse des instruction
Ce registre comporte un système de décalages circulaires sur 14 bi
de poids faible et est analysé par le microprogramme.

β) **Un registre d'état** de 16 bits, contenant entre autres :

- 2 bits de code condition
- 2 bits désignant le processeur virtuel qui dispose du hardware.

Ce registre est muni d'un système permettant de tester les bits
2 par 2.

1.4 LES BUS.

α) **Deux bus internes de données** , ALPHA et BETA, de 20 bits, constituent l'axe
de l'unité centrale et servent à faire communiquer les différents sous-ensem-
bles entre eux.

β) **Un bus de test (2 bits) et un bus d'attente (1 bit)** informent l'organe
de gestion de l'état de la machine en différents points et lui permet d'en-
chaîner les micro-instructions en conséquence.

25 points de test peuvent être connectés sur le bus de test par choix de la
microinstruction et permettre ainsi des branchements conditionnels à 4 direc-
tions.

3 points peuvent être connectés sur le bus d'attente et causer, si l'opération vérifiée n'est pas terminée, le maintien de la microinstruction en cours

1.5 ORGANES DE LIAISON ENTRE L'UNITE CENTRALE ET L'EXTERIEUR.

α) un registre CANAL de 20 bits où transitent les informations à destination des périphériques. Il peut servir également de registre de travail et permet de se brancher à un sous-microprogramme.

β) un registre NUMERO DE PERIPHERIQUE de 6 bits, permettant d'adresser un périphérique parmi 64.

γ) un bus INFO de 21 bits (dont 1 bit de parité)

δ) un bus NUMERO DE PERIPHERIQUE de 7 bits (dont 1 bit de parité).

ε) **Un panneau de face avant** : ce panneau est destiné, entre autres, à la maintenance et à la mise au point de programmes. Il comporte :

- 2 jeux de clés, l'un servant à afficher des adresses, l'autre à introduire des valeurs
- une rangée de voyants servant à visualiser différentes informations
- des clés individuelles et des commutateurs.

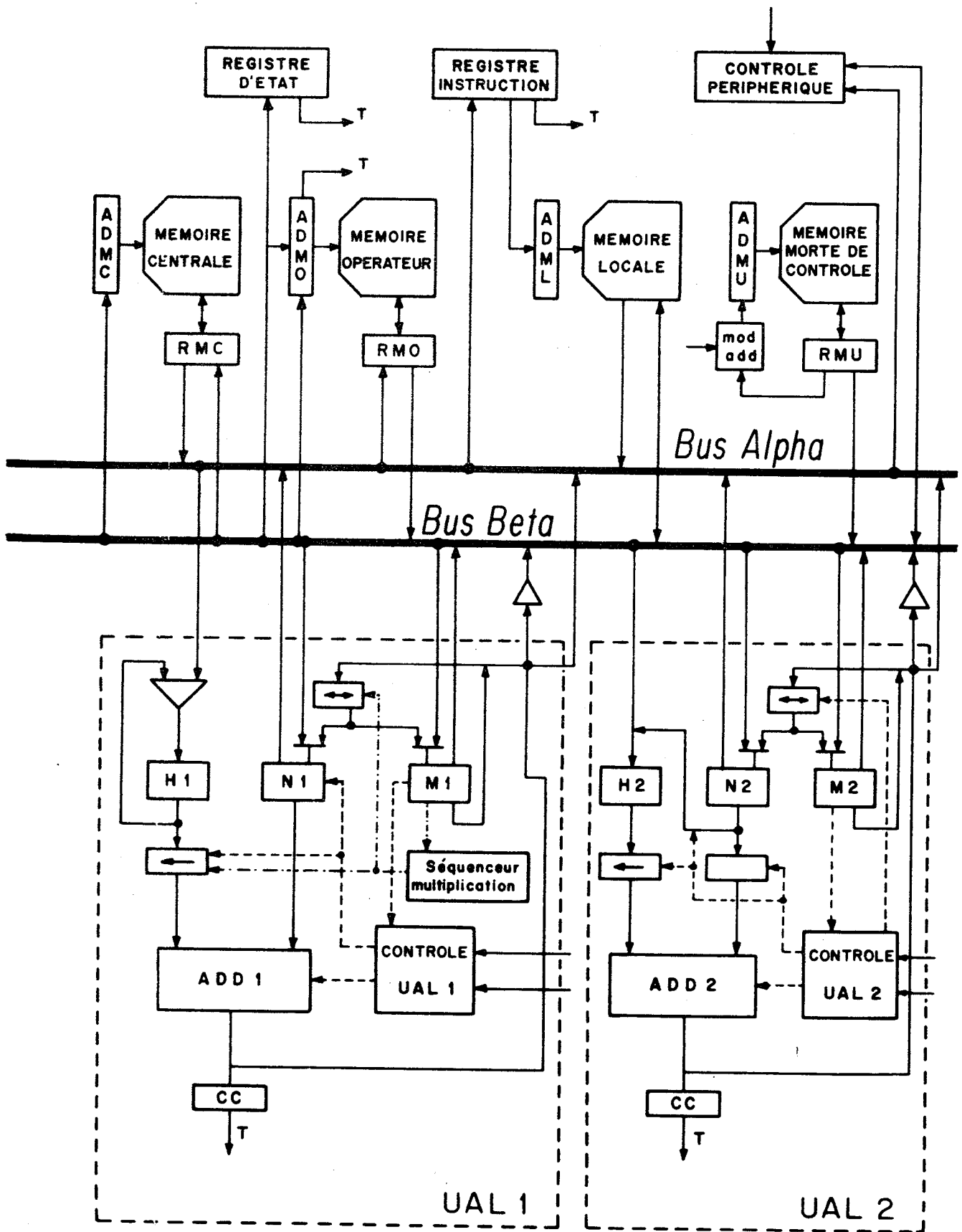


Fig. V.1.

II – CARACTERISTIQUES GENERALES.

2.1 UNITE CENTRALE.

2.1.1 Micro-machine.

La technologie de base utilisée dans la machine est la technologie ECL (emitter coupled logic, série 10000) qui permet d'obtenir un cycle de base de microprogramme de 70 nanosecondes (microcycle) : ce temps correspond au temps de calcul de l'adresse de la microinstruction additionné du temps d'accès ou au temps de decodage des ordres provenant de la microinstruction additionné du temps d'exécution de ces ordres.

2.1.2 Sous-ensembles.

- Mémoire centrale temps d'accès 210 ns
 temps de cycle 280 ns
- Mémoire opérateur temps d'accès 70 ns
 temps de cycle 140 ns
- Mémoire locale : la lecture ou l'écriture s'effectue en un micro-cycle soit 70 ns.
- Mémoire morte de contrôle : elle est organisée en 9 pages de 256 mots de 48 bits.

Parmi ces 9 pages, 7 sont utilisées pour le répertoire d'instructions standard, les 2 autres étant disponibles pour des microprogrammes particuliers (par exemple, microprogrammes de test).

- l'UAL2, très performante, a les temps d'opérations suivants :
 - multiplication fixe 700 ns
 - multiplication flottante 1400 ns
 - addition flottante 700 ns

2.2 FORMAT DES INFORMATIONS – REPRESENTATION DES NOMBRES FLOTTANTS.

L'arithmétique du géoprocasseur est en complément à 2 sur mots de 20 bits.

- Les valeurs entières sont représentées en complément à 2 avec

$$-2^{19} \leq V < 2^{19} - 1$$

- Un nombre flottant s'écrit sur un mot selon le format :



S : signe (1 bit)

M : mantisse (14 bits)

E : exposant (5 bits)

avec $V = M \cdot 2^{E-15}$ où $0 \leq E \leq 31$

Les 5 bits de l'exposant sont significatifs avec la convention suivante :

si $0 \leq E \leq 15$ l'exposant est négatif

si $16 \leq E \leq 31$ l'exposant est positif

$Exp_{GEO} = Exp_{vrai} + 16$

2.3 LES MICRO-INSTRUCTIONS – LEUR STRUCTURE.

Le microprogramme rangé en mémoire morte assure 2 fonctions : le decodage des instructions et le système d'enchaînement des tâches.

La microinstruction a été découpée de manière à pouvoir tirer parti du parallélisme fourni par l'architecture.

Le codage est généralement à un seul niveau à l'exception de quelques fonctions, peu fréquentes, où un codage à 2 niveaux est utilisé.

Une microinstruction a un champ de 48 bits que l'on peut grossièrement partager en une partie opérative et une partie adressage de la mémoire microprogrammée.

2.3.1 La partie opérative.

Elle comprend les ordres de transfert, les ordres d'exécution sur les deux unités arithmétiques et logiques, les commandes sur la mémoire locale et différents ordres sur les registres INSTRUCTION et CANAL.

La partie opérative est divisée en champs qui simplifient au maximum le décodage.

Champ adressage des registres

Ce champ donne les registres en origine et destination pour chaque transfert sur ALPHA et BETA.

Bus ALPHA : 5 origines 3 bits

4 destinations 2 bits

Bus BETA : 7 origines 3 bits
 8 destinations 3 bits

. Champ mémoire locale : la mémoire locale peut être origine ou destination; 2 bits lui sont attribués pour le mode d'adressage.

. Champ opérations arithmétiques et logiques - Sélection d'unités :
 Ce champ se compose de 8 bits : 6 sont réservés aux opérations arithmétiques et logiques, 2 pour la sélection d'unités.

. Champ commandes des mémoires vives : M.C et M.O
 Ce champ de 3 bits sélectionne la mémoire centrale ou la mémoire opérateur et indique la fonction (écriture, lecture, fonction multiple).

. Opérations sur registre INSTRUCTION ou registre CANAL
 Ce champ de 4 bits sélectionne un de ces registres et l'opération à effectuer (décalage,...).

2.3.2 Partie adressage de la mémoire microprogrammée.

. Champ de test : 5 bits.

Les tests se divisent en :

- branchements inconditionnels et marche séquentielle
- tests sur le registre d'état
- tests sur le registre instruction
- tests opérateurs
- tests mémoires
- tests fonctionnement
- tests face avant.

. Champ attente : 2 bits

. Champ adresse mémoire locale : 4 bits

. Champ valeur immédiate : 8 bits

. Champ parité interne : 1 bit.

2.3.3 En résumé.

échange par les 2 bus	11 bits
commandes des opérateurs	8 bits

commandes des mémoires	3 bits
commandes du CANAL	4 bits
commandes de l'adressage micro	7 bits
commande de la mémoire locale	6 bits
adressage de la microinstruction suivante	8 bits
parité	1 bit

Le passage d'une microinstruction à une autre peut se faire de 3 façons :

- . en séquence (incrémentation de ADMU : compteur d'adresse de la mémoire de microprogramme).
- . branchement hors de la page : l'adresse est alors donnée par les 4 bits du champ adresse mémoire locale et les 8 bits du champ valeur immédiate.
- . à la suite d'un test : l'adresse de la microinstruction suivante est le résultat du ou logique entre 2 bits de la valeur immédiate et le résultat du test (branchement court dans tous les cas).

2.4 PARALLELISME.

Pour illustrer le parallélisme dont est capable l'unité centrale, on peut noter qu'il est possible de faire en un cycle microinstruction (soit 70 ns) :

- . l'addition-décalage du contenu de 2 registres avec résultat dans l'un d'eux
- . deux transferts inter-registres par l'intermédiaire des 2 bus
- . un branchement conditionnel portant sur le résultat d'une opération effectuée au cycle précédent.

III – MICROPROGRAMME DE TEST DE L'UNITE CENTRALE.

|13|, |24|, |51|, |52|.

Ce microprogramme de test pourra être utilisé pour un test de maintenance aussi bien que pour un test de fabrication. Il sera élaboré en obéissant au principe du start-small et suivant des contraintes (acheminement, stockage, puissance de calcul), comme défini au Chap II. § 32 et § 5.2.2.

3.1 FACILITES DE TEST – HARDCORE DE PREMIER DEGRE.

Les microprogrammes de test généraux ne sont pas résidents dans la machine : la carte mémoire morte de contrôle, contenant les microprogrammes réels du calculateur, est remplacée par une carte mémoire morte où les microprogrammes de test sont figés.

D'autre part, une carte comparateur, servant à comparer un "résultat juste" mémorisé et un résultat obtenu peut être mise à la place de l'une des 2 unités arithmétiques et logiques.

Pour les tests généraux, on suppose que ces cartes sont en état de bon fonctionnement : c'est le hardcore de 1er degré.

3.2 STRUCTURE GENERALE DES MICROPROGRAMMES DE TEST.

3.2.1 Principe.

La chronologie de test employée pour l'unité centrale du géoprocasseur est celle donnée dans le chapitre II.

Nous donnerons ici un organigramme de test détaillé.

3.2.2 Structure du microprogramme de test. (Fig. V.2.)

a) Test manuel

Ce test correspond au test du hardcore de 2ème degré. Il se fait en pas à pas microinstruction et vérifie le bon fonctionnement des fonctions de chargement et de visualisation de la face Avant.

Il comporte entre autres :

- la visualisation sur face avant
- le test des clés fonction de la face avant
- le test des clés information de la face avant.

β) Test automatique avec recherche des opérandes en mémoire morte

Dans cette phase du test, les opérandes sont élaborés en utilisant le champ valeur immédiate de 2 microinstructions : ce champ étant de 8 bits, avec une extension possible à 12 bits, on peut donc former en 2 microinstructions un opérande de 20 bits.

Ce test est déterminé par les contraintes d'acheminement et de stockage des informations de test.

Ce test comprend :

- la valeur immédiate
- des fonctions particulières de l'UAL1 : ET, OU, incrémentation
- les registres de l'unité centrale (mémoire locale, UAL, ...)
- les échanges bus internes
- liaisons périphériques
- mémoire centrale et opérateur.

Ce test est effectué en vue de charger les opérandes de test en mémoire centrale à partir d'un périphérique.

γ) Test automatique avec recherche des opérandes en mémoire centrale

Ce test est d'abord déterminé par des contraintes de puissance de calcul

- . test UAL1, test UAL2
- . les macroblocs restants sont testés dans l'ordre suivant :
 - registre d'état et registre instruction
 - bascules particulières
 - encodeur-décodeur
 - liaisons périphériques

On obtient donc le schéma de test suivant :

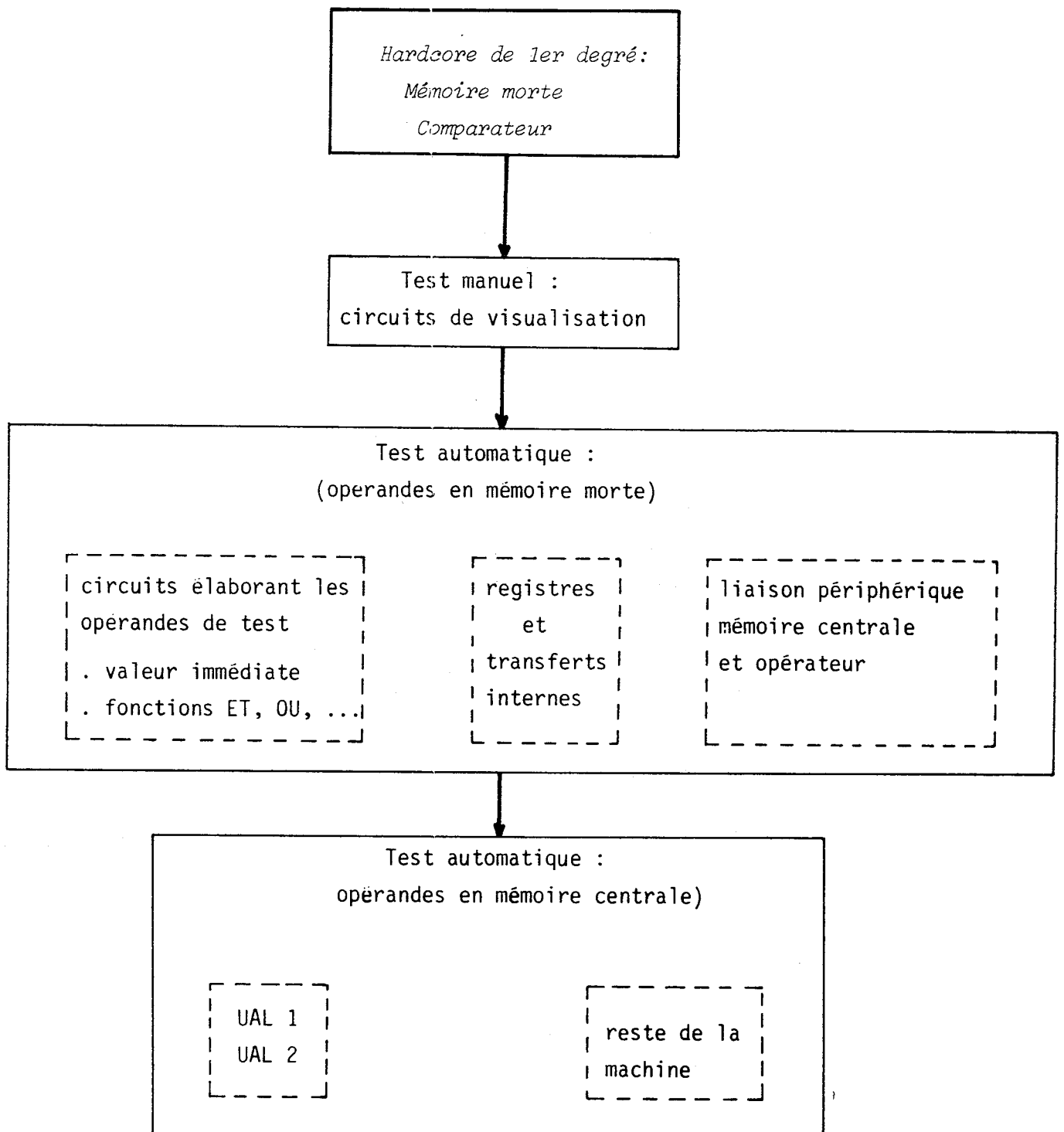


FIG. V.2

3.2.3 Contrôle de validité de bon fonctionnement et utilisation.

Pour le test en pas à pas (test manuel), la validité de bon fonctionnement est laissée à l'initiative de l'opérateur.

Pour le test automatique, la validité de bon fonctionnement est exécutée par plusieurs sous-programmes, suivant l'état des clés de fonction de la face avant.

En cas d'invalidité il y a visualisation de l'erreur en face avant.

D'une manière générale, les différentes parties du microprogramme de test sont composées de modules de microprogrammes de telle sorte que l'opérateur puisse :

- enchaîner automatiquement ces différents modules avec arrêt sur erreur
- commencer l'exécution sur un module quelconque
- boucler sur un module en cas d'erreur
- paramétrer certains modules (cas des modules de test mémoire).

3.3 STRUCTURE DETAILLEE DES MICROPROGRAMMES DE TEST.

3.3.1 Test manuel (phase 1).

a) Vérification des claviers de fonction

On vérifie par visualisation qu'à chaque position du clavier correspond une adresse de microprogramme. Les manipulations s'effectuent dans un ordre déterminé.

B) Vérification des voyants de visualisation de la face avant

L'état de la machine (attente, calcul, ...) est mémorisé dans le registre d'état et les visualisations sont alimentées à partir d'un décodeur portant sur le contenu du registre d'état.

Pour vérifier ces visualisations en pas à pas microinstruction on positionne le registre d'état à l'aide du poids fort valeurs immédiates, les voyants s'allument alors dans un ordre prédéterminé.

γ) Vérification du clavier valeur et de la rangée de voyants

Pour cette partie on passe en dynamique sur une boucle de visualisation (boucle de 3 microinstructions).

On vérifie que chacune des touches allume bien le voyant correspondant.

On vérifie également certains registres intermédiaires, au cours de cette boucle.

δ) Vérification du clavier adresse et de la rangée de voyants

Ce test se fait à travers les registres intermédiaires de γ) et on utilise une boucle de visualisation analogue à la précédente.

ε) Vérification partielle du registre adresse mémoire centrale

On vérifie ce registre (ADMC) en comptage et chargement. En effet la phase 2 des microprogrammes de test sur les registres inclue une boucle de microprogramme où ADCMC est utilisé comme compteur.

La vérification de ADCMC se fait en pas à pas microinstruction et comporte une initialisation de ADCMC à partir d'une valeur immédiate et une boucle d'incréméntation et de visualisation.

La vérification se fait visuellement par l'opérateur.

3.3.2 Test automatique avec opérandes en mémoire morte (phase 2).

La plupart des tests effectués en phase 1 qui ne portaient pas directement sur la face avant, servaient à préparer la phase 2 (test de certains registres, test partiel de ADCMC).

Dans cette phase 2 du test les opérandes sont élaborés à partir du champ valeur immédiate des microinstructions : pour obtenir un vecteur de 20 bits, on utilise une valeur immédiate étendue (VIL) de 12 bits puis une valeur immédiate poids forts de 8 bits. On écrit toutes les valeurs immédiates en séquence dans la mémoire morte de contrôle.

Dans une première phase on transfère ces valeurs immédiates dans les registres N et M, dans une deuxième phase on effectue la recombinaison des poids forts et des poids faibles à l'aide de la fonction décalage.

α) Test valeur immédiate, fonctions ET, OU, DECALAGE ...

Ce test concerne les circuits servant à l'élaboration des opérandes de test.

On vérifie simultanément la génération des valeurs immédiates et le fonctionnement des décalages; mais en cas d'erreur, il faut une observation plus approfondie pour déterminer si l'erreur est due à la valeur immédiate ou au décalage; pour ce faire on peut revenir en pas à pas.

Pour les fonctions ET, OU on fait d'une part l'opération entre la valeur immédiate et elle-même, d'autre part l'opération entre la valeur immédiate et zéro. On obtient les 4 configurations de la table de vérité de ces fonctions logiques.

β) Test registres et bus

. Vérification de l'adressage de la mémoire locale : dans cette phase préliminaire on cherche à s'assurer qu'il n'ya pas d'interaction entre les différents registres de la mémoire locale et que les différents types d'adressage fonctionnent correctement.

On pourra alors, par la suite, considérer qu'on a 16 registres indépendants.

. Test des registres de l'UAL1, de l'UAL2 et de la mémoire locale.

Dans cette partie du test des UAL, l'additionneur est transparent et, selon les commandes, laisse passer le contenu des registres H ou N. Le test des registres H, N et M cherche à faire apparaître des erreurs de câblage et des collages sur les entrées et sorties des registres.

γ) Test partiel des liaisons périphériques

La vérification de la liaison périphérique se fait en considérant un seul mode fonctionnement :

- . périphérique → unité centrale
- . sélection d'un seul périphérique.

δ) Test des mémoires centrale et opérateur

. Mémoire centrale : le test comporte 3 parties :

- Test du registre ADMC en chargement, incrémentation et relecture

- Test du registre protection mémoire : le registre de protection mémoire comporte 20 bits. Chacun de ces bits protège une zone mémoire dont la dimension dépend de la capacité mémoire.

On vérifie d'autre part que le mode système SY et le lever de protection mémoire inhibent le système de protection.

- Vérification de la mémoire : on teste les collages à 0 et à 1 dans la mémoire, et on cherche à faire apparaître également des erreurs de câblage sur le chemin de données. On cherche ensuite à faire apparaître les interactions entre zones adjacentes lors de l'écriture. Au départ les zones sont de dimension 2. On les prend 2 par 2 et on reproduit N/2 fois l'opération nécessaire (cycles de lecture et d'écriture).

On étend ensuite la dimension des zones par puissance de 2 pour arriver à la fin à 2 zones de N/2 mots.

Le problème de la localisation se complique au fur et à mesure que la dimension des zones croît. Il est donc intéressant de tourner sur une boucle d'erreur aussi courte que possible pour pouvoir observer le phénomène à l'oscilloscope mais aussi pour localiser par déduction l'emplacement du défaut.

. Mémoire opérateur

- test du registre adresse mémoire ADMO : ce registre n'est pas accessible en lecture mais, par contre, est muni de 2 codes de test.

- test mémoire : la méthode est identique à celle utilisée pour le test de la mémoire centrale.

3.3.3 Test automatique avec opérandes en mémoire centrale (phase 3).

Dans cette troisième phase du test on utilise le périphérique (testé en phase 2) pour entrer les vecteurs de test dans la mémoire centrale.

Si le format est de 20 bits l'opération est directe; par contre si le format est de 8 bits (dans le cas des cartes) on fait suivre le chargement d'une opération de compactage qui transforme 5 mots de 8 bits en 2 mots de 20 bits (programme IPL).

α) Test de l'UAL1

Un synoptique général de l'UAL1 est donné en fig. 5.3.
L'ordre de test est globalement le suivant :

- . microcommandes particulières (gestion de l'UAL)
- . registres N et M
- . registre H et l'additionneur
- . fonctions de décalage
- . gestion automatique de la multiplication.

β) Test de l'UAL2

Un synoptique général de l'UAL2 est donné en fig. 5.4.
L'ordre de test sera globalement le suivant :

- . microcommandes particulières
- . registres N et M
- . registre H et additionneur
- . gestion automatique de la multiplication
- . partie contrôle gérant les opérations automatiques
- . fonction de décalage
- . registre RT et additionneur associé.

Pour le test des deux unités arithmétiques et logiques, les vecteurs de test sont rangés en mémoire centrale de même que les résultats des opérations entre ces opérands de test. On a accès soit aux résultats partiels soit aux résultats finaux.

Connaissant le vecteur de test ayant manifesté une erreur ainsi que la valeur erronée obtenue, on peut localiser la panne. Les vecteurs sont appliqués dans un ordre tel qu'ils permettent une localisation précise de la panne (technique start-small).

γ) Liaisons périphériques

Ce test se fait à l'aide du coupleur terminal.
La vérification du dialogue avec les périphériques comporte 2 parties :

- vérification du dialogue entre l'unité centrale et les périphériques
- fonctionnement des échanges d'information.

. vérification du dialogue avec les périphériques : la vérification se fait sur le coupleur terminal. Il existe deux types d'échanges :

- échanges élémentaires sur microinterruption
- échanges à l'initiative de l'unité centrale.

contrôle de parité : le coupleur vérifie la parité de toutes les informations qu'il reçoit de l'unité centrale (mot de fonction ou information).

Toute détection d'erreur donne lieu à une microinterruption d'incident.

microinterruption : une microinterruption peut avoir plusieurs significations.

En cas d'incident, l'unité centrale lit sur le bus information la cause de l'incident.

. fonctionnement des échanges réels : ce test se fait avec un périphérique, en dynamique, à la vitesse propre du périphérique.

Les échanges élémentaires ont été vérifiés jusqu'à l'entrée du coupleur. On lance des transferts dont le contenu est connu à l'avance, entre les périphériques et l'unité centrale.

périphérique \longleftrightarrow unité centrale

On vérifie ensuite l'information lue ou émise et s'il y a erreur on refait le transfert de façon cyclique.

δ) Test du reste de la machine

. Registre d'état : le contenu du registre d'état n'est accessible que par test : système permettant de tester les bits 2 par 2.

On utilise une boucle de test en tenant compte du fait que ce registre n'a que 16 bits et comporte une discontinuité de 4 bits en son milieu

. Registre instruction : le test se fait en 2 parties

- vérification des codes de test : les 5 bits du code opération sont transcodés en 6 bits pour former 3 codes de test.
- vérification du chargement, des décalages et de la relecture : les 14 bits de poids faible du registre fonctionnent en décalage circulaire de 2 positions. Ils peuvent être émis ensemble ou de façon partielle sur le bus ALPHA.

. Inversion binaire : on utilise le fait que lorsque l'inversion binaire porte sur un nombre ayant un seul bit égal à 1, elle est équivalente à un décalage.

La vérification se fait donc par comparaison avec des décalages.

. Encodeur-décodeur : on vérifie le câblage de l'encodeur prioritaire-décodeur puis le bon fonctionnement du système de priorité par un test de collage.

. Bascules de fonction : on vérifie certaines bascules particulières.

IV – ETUDE DE DETAIL : LES UNITES ARITHMETIQUES ET LOGIQUES

Nous choisirons comme exemple l'UAL2 du géoprocasseur parce que la plus complexe.

4.1 STRUCTURE DE L'UNITÉ ARITHMETIQUE ET LOGIQUE (Fig. V.5).

L'UAL2 est spécialisée dans les opérations sur nombres flottants. Elle effectue également les opérations d'addition et de multiplication sur les nombres entiers.

Les opérations sur les nombres flottants, la multiplication et les opérations de décalage sont initialisées par microprogramme et s'exécutent automatiquement sur plusieurs cycles.

L'UAL2 est principalement composée de :

- quatre registres H, N, M, RF
- un additionneur ADD1 (ou Σ_1)
- une partie contrôle assurant la gestion automatique des opérations.

4.1.1 Le registre H.

Le registre H, de 20 bits, est constitué de 2 étages de bascules Latch, en cascade, de manière à limiter le nombre de portes nécessaires au fonctionnement de l'ensemble (H, ADD1).

Le principe est donné en Fig. 5.6.

Ce registre reçoit :

- soit le contenu du bus
- soit le contenu de N (opérations automatiques).

La sortie du registre est connectée sur l'entrée de l'additionneur ADD1.

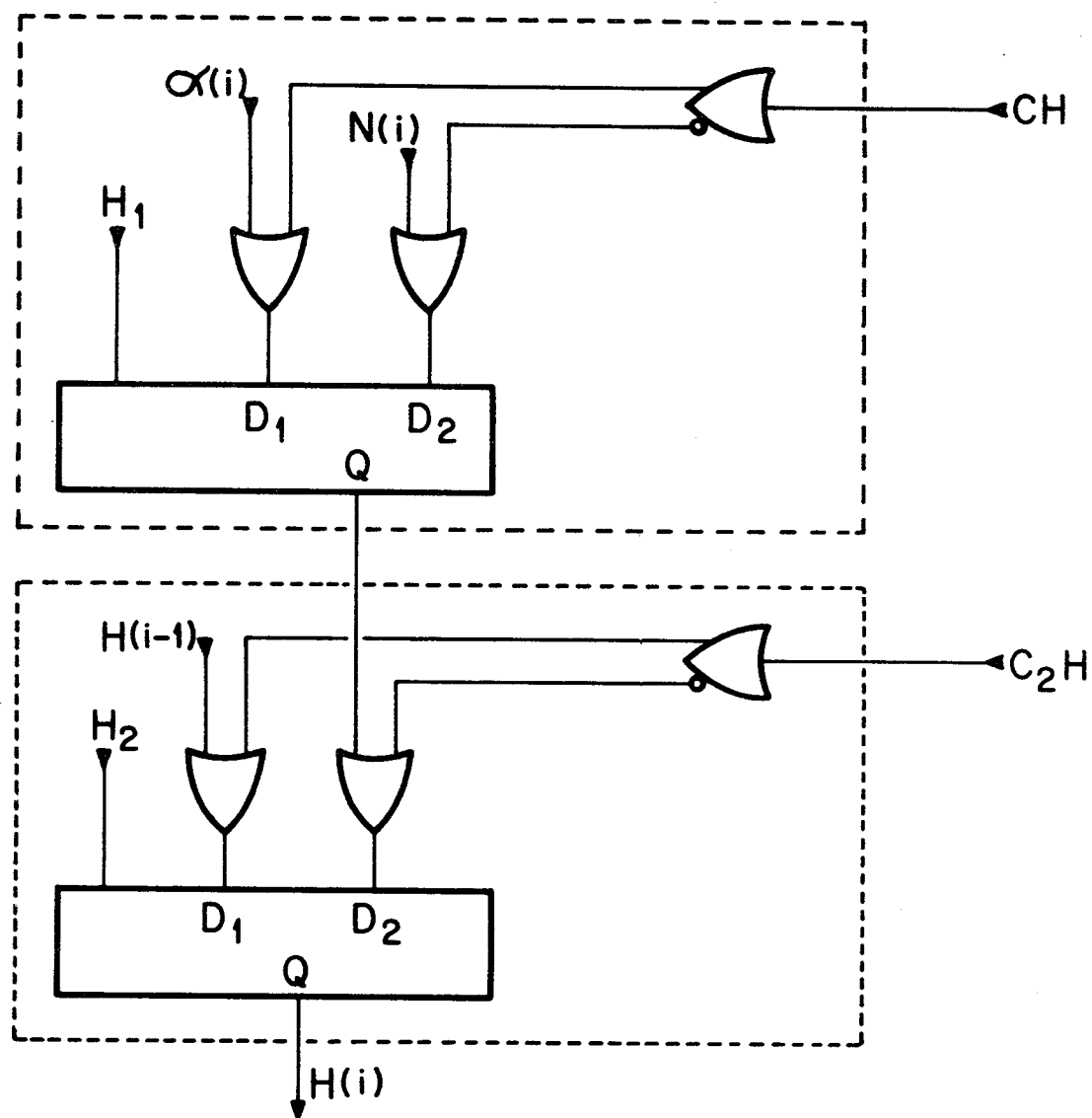


Fig. V.6.

4.1.2 Le registre N.

Le registre N de 20 bits, est constitué de bascules D; l'aiguillage de ses entrées se fait au moyen d'un multiplexeur adressé par 3 bits, ces 3 bits étant le résultat d'un encodage de 8 commandes générées par l'unité de gestion.

Ce registre reçoit :

- soit zéro
- soit le contenu du bus β
- soit la sortie de l'additionneur ADD1 (opérations automatiques)

Sa sortie est connectée par l'intermédiaire de 2 portes

- soit sur le bus α
- soit sur l'entrée du registre H
- soit sur l'entrée de l'additionneur ADD1 (opérations automatiques)

4.1.3 Le registre M.

Le registre M de 20 bits est constitué de 5 quadruples registres à décalage dans les deux sens.

Comme pour le registre N, ses entrées sont multiplexées par des circuits adressés par 3 bits, issus de l'encodage de commandes générées par l'unité de gestion.

Ce registre reçoit :

- soit zéro
- soit le contenu du bus β
- soit sa propre sortie transformée
- soit la sortie de N (opérations automatiques)

Il émet sur le bus β par l'intermédiaire d'une porte.

4.1.4 Le registre RF.

Ce registre reçoit le contenu de 7 bits de poids faible du bus α , pour l'initialisation des opérations spéciales automatiques (gérées par l'unité de contrôle).

4.1.5 L'additionneur ADD1.

L'additionneur ADD1 effectue

- soit une addition
- soit une soustraction
- soit un transfert de N, direct ou inversé, par l'intermédiaire de commandes générées par l'unité de contrôle

La sortie est en communication avec :

- soit le bus β
- soit les registres N et M

- soit l'organe de gestion par portes interposées.

4.1.6 L'unité de contrôle.

L'organe de gestion automatique génère différents signaux nécessaires au fonctionnement de l'UAL2. Ces signaux sont issus :

- soit de la microinstruction
- soit du décodage du registre RF initialisé par microinstruction.

Associés à l'organe de gestion on trouve un registre RT de 7 bits et un additionneur ADD2, propres au traitement des exposants dans les opérations sur nombres flottants.

Le registre RT contient le nombre de décalages pour les opérations de décalage et peut également fonctionner en compteur.

4.2 MICROPROGRAMMATION DE L'UNITE ARITHMETIQUE ET LOGIQUE.

4.2.1 Opérations sur les nombres entiers.

(accès direct par microprogramme)

α) Addition (1 cycle) : en l'absence de toute autre commande, l'addition s'effectue de façon permanente.

β) Multiplication (10 cycles) : la multiplication s'effectue en une séquence de 10 cycles, initialisée par l'organe de gestion.

L'algorithme de multiplication est un algorithme de Booth traitant des chaînes consécutives de 3 bits (cf. Chap. III § 2.3.1).

4.2.2 Opérations sur les nombres flottants et décalages.

(opérations spéciales)

Ces opérations sont initialisées à l'aide du registre RF par la valeur émise sur le bus α : code de l'opération à effectuer (4 bits) + conditions initiales (3 bits).

Il y a un algorithme par opération automatique. Chaque sous-opération dans un algorithme est repérée par un numéro : cette sous-opération (n) est appelée phase n (cf. Chap. III § 3.1.1).

Certaines phases sont elles-mêmes un enchaînement conditionnel de cycles de base (phases de décalage et de normalisation).

4.3 METHODOLOGIE DE TEST ET ORDONNANCEMENT. [50], [51].

Cette méthodologie de test est élaborée suivant les principes énoncés au Chap. II.

L'UAL2 constitue un macro-bloc.

Les différents micro-blocs seront :

- . les registres N, M, H, RT
- . les additionneurs ADD1, ADD2
- . la partie contrôle générale de l'UAL2
- . le bloc de gestion automatique des opérations spéciales.
- . le séquenceur de multiplication.

4.3.1 Analyse du chemin de données.

On considère ici uniquement les micro-blocs de traitement soit H, N, M, RT, ADD1 (ou Σ_1) et ADD2 (ou Σ_2).

α) Relations de commandabilité

- .H est atteint à partir du bus ou de N
- .N est atteint à partir du bus ou de ADD1 ou de (M & ADD1)
- .M est atteint à partir du bus ou de N ou de (RT & N) ou de (M & ADI ou de (M & N)
- .RT est atteint à partir de ADD1 ou de ADD2
- .ADD1 est atteint à partir de H et N
- .ADD2 est atteint à partir de RT.

β) Relations d'observabilité

- .H est observable à travers ADD1
- .N est observable à travers ADD1 ou le bus ou H ou M
- .M est observable à travers le bus ou lui-même ou N
- .RT est observable à travers ADD2
- .ADD1 est observable à travers RT ou le bus ou N ou M
- .ADD2 est observable à travers RT.

4.3.2 Recherche d'un ordonnancement.

On obtient le système d'inéquations suivant :

$$\left\{ \begin{array}{ll}
 \cdot H > \text{bus} + N & \longleftrightarrow H > \text{bus} \\
 H > \text{ADD1} & \\
 \cdot N > \text{bus} + \text{ADD1} + M \cdot \text{ADD1} & \longleftrightarrow N > \text{bus} \\
 N > \text{ADD1} + \text{bus} + H + M & \longleftrightarrow N > \text{bus} \\
 \cdot M > \text{bus} + N + RT \cdot N + M \cdot \text{ADD1} + M \cdot N & \longleftrightarrow M > \text{bus} \\
 M > \text{bus} + N + M & \longleftrightarrow M > \text{bus} \\
 \cdot RT > \text{ADD1} + \text{ADD2} & \\
 RT > \text{ADD2} & \\
 \cdot \text{ADD1} > H \cdot N & \\
 \text{ADD1} > RT + \text{bus} + N + M & \longleftrightarrow \text{ADD1} > \text{bus} \\
 \cdot \text{ADD2} > RT &
 \end{array} \right.$$

L'application des règles de simplification permet d'obtenir le système suivant :

$$\left\{ \begin{array}{l}
 H > \text{ADD1} \\
 N > \text{bus} \\
 M > \text{bus} \\
 RT > \text{ADD2} + \text{ADD1} \cdot \text{ADD2} \\
 \text{ADD1} > H \cdot N \\
 \text{ADD2} > RT
 \end{array} \right.$$

On observe deux blocs d'indiscernabilité : (H, ADD1) et (RT, ADD2).

La chronologie de test pour les micro-blocs de traitement est la suivante :

- 1) N, M
- 2) (H, ADD1)
- 3) (RT, ADD2).

On obtient le schéma de test partiel suivant (Fig. 5.7)

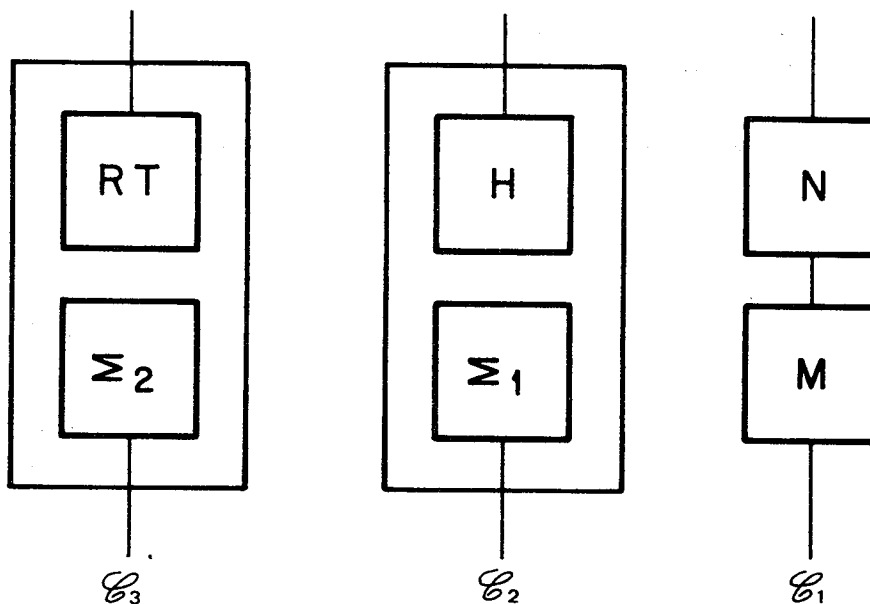


Fig. V.7.

4.3.3 Analyse au niveau commandes.

L'UAL2 comporte 3 micro-blocs de commande :

- . l'unité de gestion de l'UAL2
- . l'automate de contrôle des opérations spéciales
- . le séquenceur de multiplication.

On a vu au Chap. II . § 5.3.2.(ε) qu'il existe deux types de micro-blocs de traitement (gestion totale, gestion partielle) qui définissent l'ordre de test.

. l'unité de gestion est du type a) (gestion totale) et sera donc testée en priorité avant les micro-blocs de traitement qu'elle gère : soit N, M, H et ADD1.

. l'automate de contrôle des opérations spéciales est du type b) (gestion partielle); il sera donc testé après les micro-blocs de traitement pouvant fonctionner indépendamment de lui : soit après H, N, M et ADD1.

. le séquenceur de multiplication est également de type b et sera testé après les micro-blocs de traitement H, N, M et ADD1.

De plus, l'automate de contrôle des opérations spéciales gérant l'opération de multiplication sur nombre flottants et cette opération utilisant la multiplication entière, cet automate sera testé après le séquenceur de multiplication.

Pour l'ensemble de l'UAL2 on obtient donc le schéma de test de la Fig. 5 8.

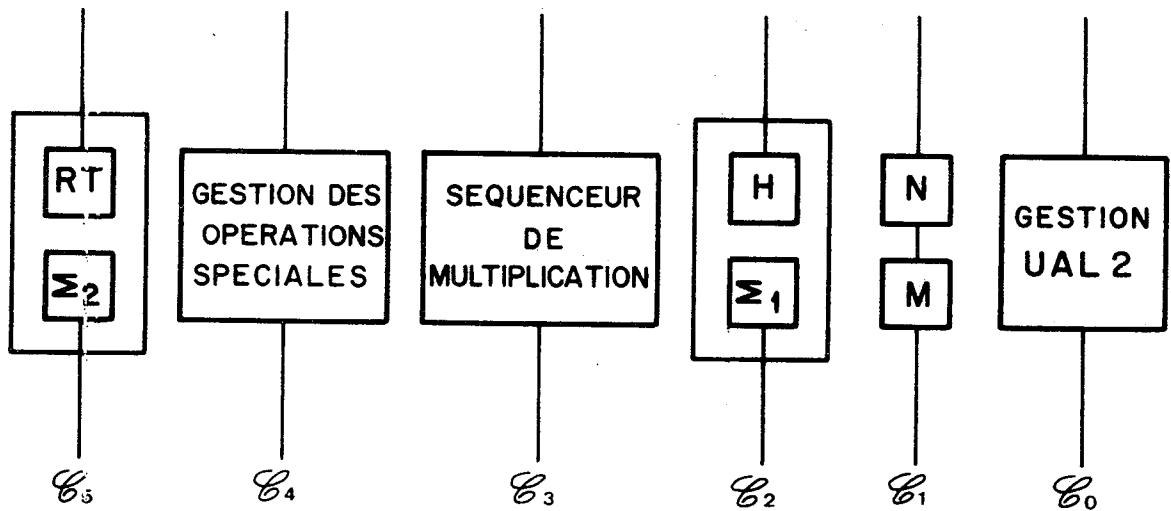




Fig. V.8.

4.4 TRAITEMENT AU NIVEAU DU MICRO-BLOC (Fig. 5.9).


Les méthodes de test au niveau des micro-blocs sont définies suivant les principes donnés dans les trois premiers chapitres.

. Pour les différents registres : H, N, M, RT on choisit des méthodes analytiques pures 

. Pour les additionneurs : ADD1 et ADD2 on choisit une méthode analytique prenant en compte les facteurs de répétitivité et symétrie. 

. Pour le séquenceur de multiplication on choisit une méthode fonctionnelle par vérification d'automates. 

Cet automate de multiplication est un automate de contrôle de type I (cf. Chap. III § II).

. Pour l'unité de gestion des opérations automatiques on choisit une méthode spécifique originale. 

Cette unité de gestion est un automate de contrôle de type II (cf. Chap. III § III).

V – SPECIFICATIONS TECHNIQUES DU MICROPROGRAMME DE TEST

Nous donnerons dans cette section quelques indications sur la place en mémoire morte et le nombre de microinstructions qui ont été nécessaires pour le test global de cette unité centrale.

5.1 PHASE 1 (TEST MANUEL).

ou vérification du **hardcore** de 2ème degré en pas à pas micro instruction.

Place en mémoire morte : 169 mots mémoire.

5.2 PHASE 2 (TEST AUTOMATIQUE AVEC OPERANDES EN MEMOIRE MORTE).

les opérandes de test sont fabriqués à partir du champ valeur immédiate (VI) des microinstructions.

. Valeur immédiate, fonctions ET, OU, DECALAGE

57 microinstructions.

. Test registres

registres mémoire locale 113 mots mémoire
registres UAL1, UAL2, ... 217 mots mémoire

. Test mémoires

protection mémoire	182 microinstructions
mémoire opérateur	109 microinstructions
mémoire centrale	161 microinstructions

5.3 PHASE 3 (TEST AUTOMATIQUE AVEC OPERANDES EN MEMOIRE CENTRALE).

. Test UAL1

- 494 vecteurs de test
- 55 adresses indirectes
- 171 microinstructions.

. Test UAL2

- pour les opérations entières
234 vecteurs de test
75 microinstructions
- pour les opérations flottantes
342 vecteurs de test
100 microinstructions
- et 77 adresses indirectes

. Registre d'état

- 16 adresses indirectes
- 48 vecteurs de test
- 107 microinstructions

. Registre instruction

- 31 adresses indirectes
- 186 vecteurs de test
- 124 microinstructions

. Encodeur - décodeur

- 17 adresses indirectes
- 85 vecteurs de test
- 43 microinstructions

Remarque : les vecteurs de test ou opérandes ainsi que les adresses indirectes sont rangés en mémoire centrale.

L'ensemble du microprogramme de test de l'unité centrale du géoprocasseur occupe donc 1,6 K mots de la mémoire morte de test et environ 1600 vecteurs chargés en mémoire centrale.

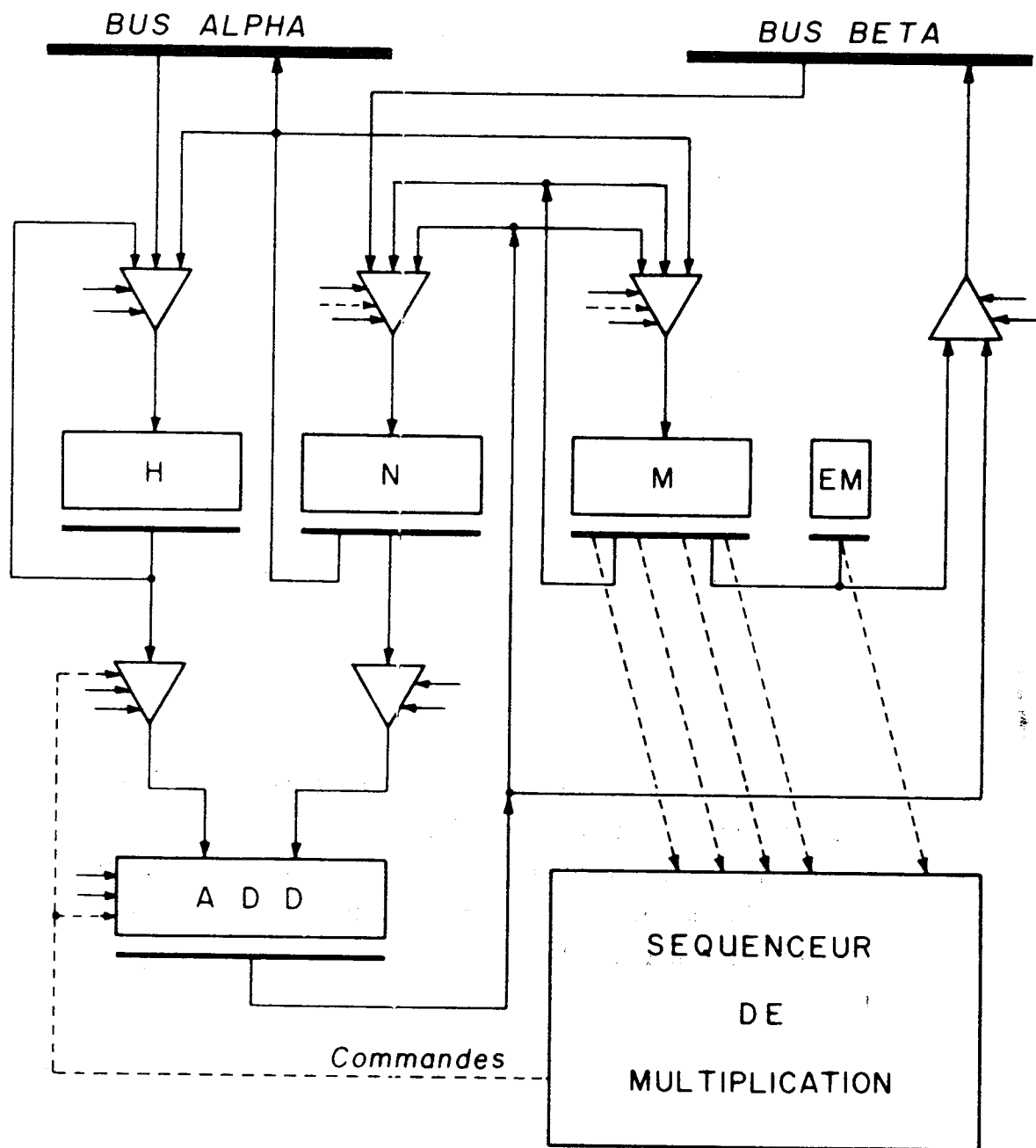


Fig. V.3.

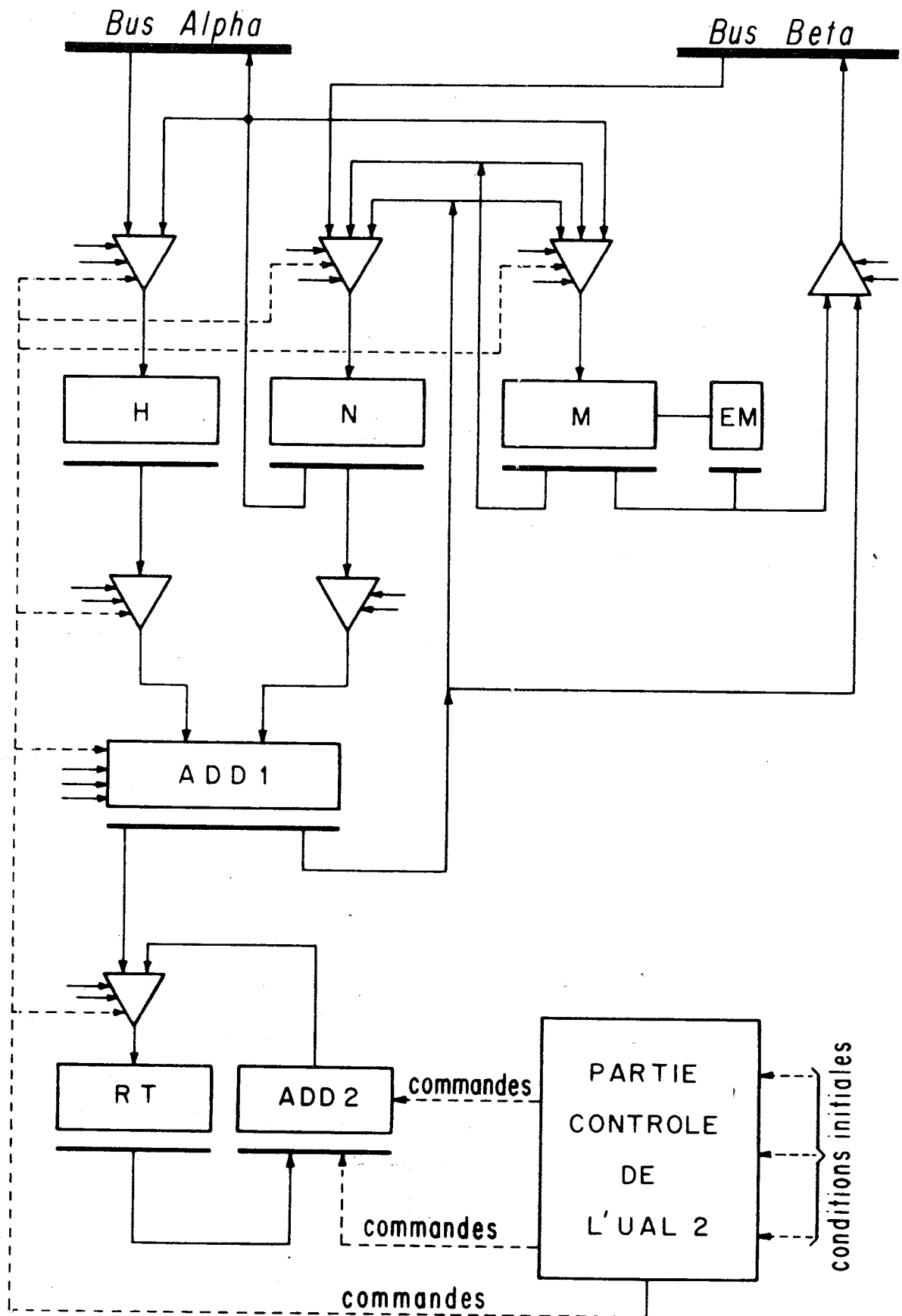


Fig. V.4.

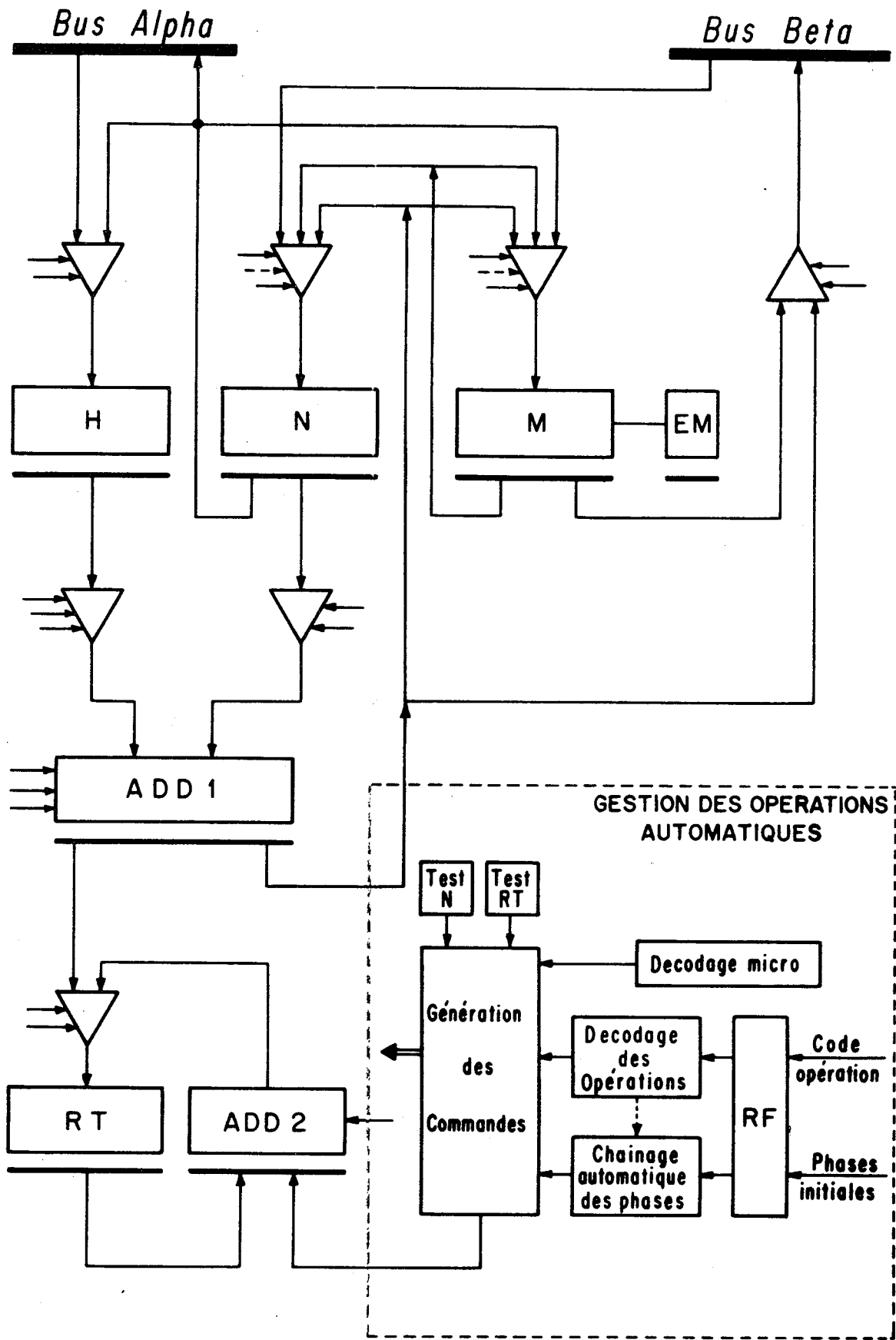


Fig. V.5.

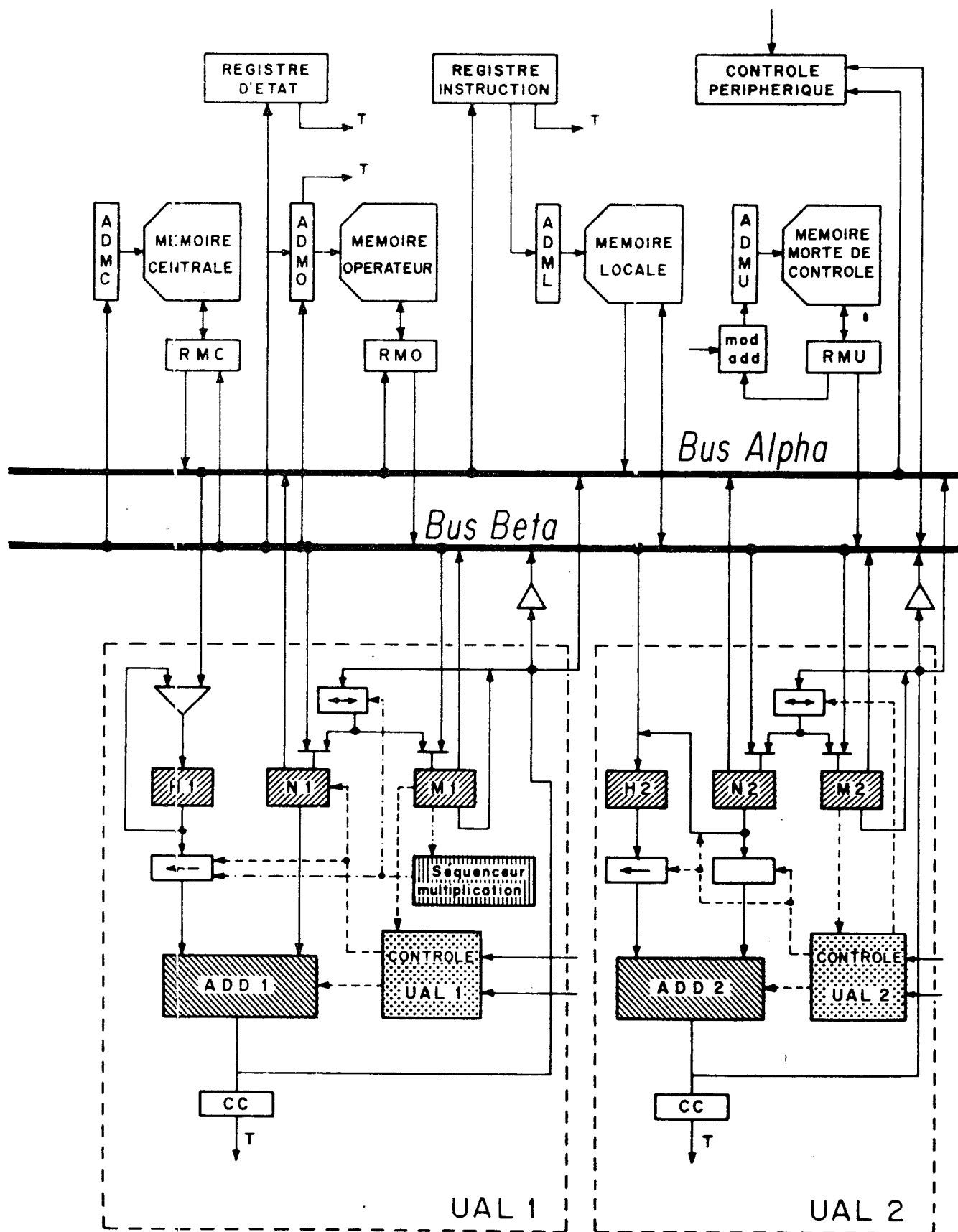


Fig. V.9.

BIBLIOGRAPHIE

* * *

- [1] AGRAWAL.V.D, AGRAWAL.P :
"An automatic test generation for Illiac IV Logic Boards"
IEEE Transactions on Computers, Septembre 1972, pp 1015-1017
- [2] ANCEAU.F, DROUET.Ph, CRETIN.J, BEAUDUCEL.C, COURBOULET.P :
"Geoproscesseur : computer for geophysical research"
Information Processing, Vol 1, 1974, pp 55-59
- [3] BARTOW.N, Mc GUIRE.R :
"System/360 model 85 microdiagnostics"
Spring Joint Computer Conference, 1970, pp 191-197
- [4] BELL.C.G, GRASON.J, NEWELL.A :
"Designing computers and digital systems"
Digital Press, 1972
- [5] BENNETS.R.G., LEWIN.D.W. :
"Fault diagnosis of digital systems-a review"
Computer, Juillet/Août 1971, pp 12-21
- [6] BERGE.C :
"Théorie des graphes et ses applications"
Edition DUNOD, Collection Universitaire de Mathématiques, PARIS 1963
- [7] BREUER.M.A. :
"Hardware fault detection"
Proc. AFIPS, FJCC, 1968, pp 1502-1503
- [8] BREUER.M.A. :
"Generation of fault-detection tests for sequential circuits"
First Annual Symposium on Fault-Tolerant Computing,
Pasadena, USA, 1971

- |9| BREUER.M.A. :
"Design Automation of digital systems"
Chap 7 : *Fault Test Generation* by PREISS.R.J.
Vol 1, Prentice Hall, 1972, pp 335-410
- |10| BURNSTINE.D.C, EPPARD.W.H :
"Maintenance Strategy diagramming technique"
Annual Symposium on reliability, 1966, pp 497-506
- |11| CARTER.W.C. and al. :
"Design of serviceability features for the IBM system 360"
IBM Journal, Avril 1964, pp 115-126
- |12| CHANG.H.Y, MANNING.E, METZE.G :
"Fault Diagnosis of digital systems"
Wiley-Interscience, New York, 1970
- |13| COURTOIS.B, GUERIN.C, ROBACH.Ch, SAUCIER.G, VERDILLON.A :
"Computer Test Program"
Workshop on diagnosis and reliable design of digital systems, Pasadena,
USA, Décembre 1973
- |14| CULLMANN.G :
"Recherche opérationnelle - Théorie et pratique"
Edition Masson et Cie, PARIS, 1970
- |15| DENT.J.J :
"Diagnostic Engineering requirements"
Proc. AFIPS, SJCC, 1968, pp 503-507
- |16| Digital Equipment Corporation :
"PDP 16 : Computer designers handbook", 1971
- |17| DROUET.Ph :
"Présentation générale du geoprocesseur"
Note interne ENSIMAG, Grenoble, Novembre 1973

- |18| DUMITRESCU.D, SAUCIER.G :
"Test de mémoires dynamiques à technologie MOS"
5th Annual Symposium on Fault Tolerant Computing, PARIS, Juin 1975
- |19| FRIEDMAN.A.D, MENON.P.R :
"Fault Detection in digital circuits"
Electrical Engineering series, Englewood Cliffs, N.J. Prentice Hall,
1971
- |20| GAVRILOV.M.A, KUZNETSOV.O.P, KHAZATSKI.V.C :
"Description and analysis of switching circuits with large number of
input variables"
Automation and Remote Control, n° 10, 1969, pp 1643-1650
- |21| GIRARD.E, RAULT.J.C, TULLOUE.R :
"Les méthodes probabilistes de génération de séquences de test :
estimation de l'efficacité et de la longueur des séquences"
Revue technique THOMSON-CSF, Vol 6, N°1, Mars 1974, pp 197-216
- |22| GIRARD.E, RAULT.J.C, TULLOUE.R :
"La simulation des circuits logiques : problèmes et mise en oeuvre"
Revue technique THOMSON-CSF, Vol 6, n°1, pp 171-195
- |23| GLUSHKOV.V.M, LETICHEVSKII.A.A :
"Advances in information systems science", Vol 1
PLENUM Press, New York, edited by Julius.T.Tou, 1969
- |24| GUERIN.C, ROBACH.Ch, SAUCIER.G, VERDILLON.A :
"Programme de maintenance d'ordinateurs"
Journée d'étude IRIA : Prévention des pannes dans les systèmes logiques
Rocquencourt, FRANCE, Avril 1974, pp 62-97
- |25| GUFFIN.R.M :
"Microdiagnostics for the Standard Computer MLP-900 Processor"
IEEE Transactions on Computers, Juillet 1971, pp 803-808

- |26| HACKL.F.J, SHIRK.R.W :
"An integrated approach to automated computer maintenance"
IEEE Conference Record on Switching circuit theory & logical design, 1965, pp 289-300
- |27| HAHN.J.R :
"A maintenance approach for a large computer system"
A talk given at Lough University, Décembre 1971
- |28| HENNIE.F.C :
"Fault Detecting experiments for sequential circuits"
Proc. of the 5th Annual Switching theory & logical design symposium, S-164, pp 95-110, 1964
- |29| HUANG.H.H :
" MSI and LSI impact on digital systems testing"
Proceedings of the 11th Annual design Automation Workshop, 1974, pp 159-165
- |30| JOHNSON.A.M :
"The microdiagnostics for the IBM System 360 model 30"
IEEE Transactions on Computers, Juillet 1971, pp 798-803
- |31| JONES.E.R, MAYS.C.H :
"Automatic test generation methods for large scale integrated logic"
IEEE Journal of solid-state circuits, Vol SC-2, 1967, pp 221-226
- |32| KELLA.J :
"Sequential machine identification"
IEEE Trans on computers, Mars 1971
- |33| KIME.C.R :
"An analysis model for digital system diagnosis"
IEEE Computer group repository, R-70-19
- |34| KOHAVI.Z, LAVALLEE.P :
"Design of sequential machines with fault-detection capabilities"
IEEE Transactions on Computers, Vol EC-16, 1967, pp 473-484

- |35| KOHAVI.Z :
"Switching Automata theory"
Mc Graw-Hill, Computer Science Series, 1970, Chap. 13
- |36| KUBO.H :
"A procedure for generating test sequences to detect sequential circuit failures"
NEC R & D, N° 12, 1968, pp 69-78
- |37| KUNTZMANN.J :
"Théorie des réseaux"
Editions DUNOD, PARIS, 1972
- |38| LEAMAN.R.J, LLOYD.M.H, REPTON.C.S :
"The development and testing of a processor self-test program"
The Computer Journal, Vol 16, Numéro 4, pp 308-314
- |39| OZAWA.Y, MURAKAMI.M, SUZUKI.K :
"Master slice LSI computer aided design system"
11th Design Automation Workshop, Proceedings, Denver, USA, 1974
- |40| PAIR.C, DERNIAME.J.C :
"Etude des problèmes de cheminement dans les graphes finis"
Rapport, Institut universitaire de calcul automatique, NANCY
- |41| PARKER.K, Mc CLUSKEY.E :
"Analysis of logic circuits with faults using input signal probabilities"
Fourth Annual Symposium on Fault-Tolerant Computing, Champaign, USA, Juin 1974
- |42| PETIT-CLERC.A :
"Traité des ordinateurs"
DUNOD, Tome 1, 1970, pp 172-182
- |43| RAMAMOORTHY.C.V :
"A structural theory of machine diagnosis"
Proc. AFIPS, SJCC, 1967, pp 743-756

- |44| RAMAMOORTHY.C.V, CHANG.L.C :
"System modeling and testing procedures for micro-diagnostics"
IEEE Trans. on Computers, Vol C21, n° 11, Novembre 1972, pp 1169-1183
- |45| RAULT.J.C :
"A graph theoretical and probabilistic approach to the fault detection of digital circuits"
First Annual Symposium on Fault Tolerant Computing, Pasadena, USA, Mars 1971, pp 26-29
- |46| RAYNARD.J.C, BRU.X :
"Test du calculateur T1600"
Projet de fin d'études, ENSIMAG, Grenoble, Juin 1974
- |47| RIGAULT.J.P :
"Utilisation de registres à décalages pour le test et diagnostic des systèmes digitaux"
Séminaire ENSIMAG, Grenoble, Mai 1974
- |48| ROBACH.Ch :
"Synthèse d'une partie très testable d'un ordinateur"
Séminaire ENSIMAG, Grenoble, Octobre 1973
- |49| ROBACH.Ch, GUERIN.C, SAUCIER.G :
"Test of a process control unit"
IEEE FTC4 Symposium, Champaign, USA, Juin 1974
- |50| ROBACH.Ch, SAUCIER.G :
"System Test Plan"
CEE Advanced Course on computer system architecture, Serre-Chevalier, FRANCE, Décembre 1974
- |51| ROBACH.Ch, SAUCIER.G :
"Méthodologie de test efficace pour des unités centrales"
5th Symposium on Fault Tolerant Computing, Paris, Juin 1975
- |52| ROBACH.Ch, SAUCIER.G :
"System Test Plan"
1975 Missouri symposium on advanced Automation, Columbia, USA, Avril 1975

- |53| ROBACH.Ch :
"Diversified Test Methods for local control units"
IEEE Transactions on Computers, Mai 1975
- |54| ROTH.J.P :
"Diagnosis of automata failure : a calculus and a method"
IBM Journal R & D, Vol 10, 1968, pp 278-291
- |55| RUSSEL J.D :
"On the diagnosability of digital systems"
International symposium on fault-tolerant computing, 1973, Palo-Alto, Californie
- |56| SALISBURY.A.B, ENSLOW.P.H :
"Diagnostic programming for digital computers- a bibliography"
West Point Military Academy, Avril 1967, AD 813 831
- |57| SAUCIER.G :
"Recherche d'une séquence de test d'une machine séquentielle"
AFCET, RAIRO n° J1, 1972
- |58| de SEITZ.Ch :
"An approach to designing checking experiments based on a dynamic model"
Theory of machines and computations symposium, Haïfa, Israël, Août 1971
- |59| SELLERS.F.F, HSIAO.M.Y, BEARNSON.L.W :
"Analyzing errors with the Boolean Difference"
IEEE Trans. on Computers, Vol C17, 1968, pp 676-683
- |60| SESHU.S :
"On a improved diagnosis program"
*IEEE Trans. on **electronic computers**, Vol EC-14, 1965, pp 69-76*
- |61| SUSSKIND.A.K :
"Diagnostics for logic networks"
IEEE Spectrum, Octobre 1973

- [62] TABARLY.J.H :
"Contribution à la synthèse de tests complets de circuits digitaux par observation-commande, programme DEDALE"
Thèse de spécialité, Université de Toulouse, 1974
- [63] TOTH.A, HOLT.C :
"Automated Database-driven digital testing"
Computer, Janvier 1974, pp 13-19
- [64] UMETANI.Y, MYAMOTO.S, HORIKOSHI.H :
"Some results on the application of FLT generator to a large scale computer"
Third Annual Symposium on Fault Tolerant Computing, Palo Alto, USA, 1973
- [65] UNGER.S.H :
"Asynchronous sequential switching circuits"
Wiley Interscience, 1969
- [66] VERDILLON.A, TURCAT.C :
"Symétrie, récurrence et test"
5th Annual Symposium on Fault Tolerant Computing, Paris, Juin 1975
- [67] VERMA.J.P, SELOVE.D.M, TESSIER.J.N :
"Automatic test generation and test-verification of digital systems"
Proceedings of the 11th Annual design Automation Workshop, 1974,
pp 149-158