Définition du langage CASSANDRE : pour la conception aidée et la simulation des systèmes logiques, leur analyse, description et réalisation

Jean Mermet

N° d'ordre

THESE

présentée à

LA FACULTE DES SCIENCES DE L'UNIVERSITE DE GRENOBLE

pour obtenir

LE GRADE DE DOCTEUR INGENIEUR

par

jean mermet

Ingénieur I. M. A. G.

définition du langage cassandre

(pour la Conception Aidée et la Simulation des Systèmes logiques,

leur ANalyse, Description et REalisation)

Thèse soutenue le 21 Mars 1970 devant la commission d'examen :

Monsieur	J. KUNTZMANN	Président
Messieurs	C. BENZAKEN L. BOLLIET	Examinateurs

LISTE DES PROFESSEURS

Doyen honoraire : Monsieur M. MORET
Doyen : Monsieur E. BONNIER

PROFESSEURS TITULAIRES

MM.	NEEL Louis	Physique Expérimentale
	KRAVTCHENKO Julien	Mécanique Rationnelle
	CHABAUTY Claude	Calcul différentiel et intégral
	BENOIT Jean	Radioélectricité
	CHENE Marcel	Chimie Papetière
	FELICI Noël	Electrostatique
	KUNTZMANN Jean	Mathématiques Appliquées
	BARBIER Reynold	Géologie Appliquée
	SANTON Lucien	Mécanique des Fluides
	OZENDA Paul	Botanique
	FALLOT Maurice	Physique Industrielle
	KOSZUL Jean-Louis	Mathématiques
	GALVANI Octave	Mathématiques
	MOUSSA André	Chimie Nucléaire
	TRAYNARD Philippe	Chimie Générale
	SOUTIF Michel	Physique Générale
	CRAYA Antoine	Hydrodynamique
	REULOS René	Théorie des Champs
	BESSON Jean	Chimie Minérale
	AYANT Yves	Physique Approfondie
	GALLISSOT François	Mathématiques
Melle.	LUTZ Elisabeth	Mathématiques
MM.	BLAMBERT Maurice	Mathématiques
	BOUCHEZ Robert	Physique Nucléaire
	LLIBOUTRY Louis	Géophysique
	MICHEL Robert	Minéralogie et pétrographie
	BONNIER Etienne	Electrochimie et Electrométallurgie
	DESSAUX Georges	Physiologie animale
	PILLET Emile	Physique Industrielle-Electrotechnique
	YOCCOZ Jean	Physique Nucléaire théorique
	DEBELMAS Jacques	Géologie Générale
	GERBER Robert	Mathématiques
	PAUTHENET René	Electrotechnique
	MALGRANGE Bernard	Mathématiques Pures
	VAUQUOIS Bernard	Calcul Electronique
	BARJON Robert	Physique Nucléaire

MM.	BARBIER Jean-Claude	Physique
	SILBER Robert	Mécanique des Fluides
	BUYLE-BODIN Maurice	Electronique
	DREYFUS Bernard	Thermodynamique
	KLEIN Joseph	Mathématiques
	VAILLANT François	Zoologie et Hydrobiologie
	ARNAUD Paul	Chimie
	SENGEL Philippe	Zoologie
	BARNOUD Fernand	Biosynthèse de la Cellulose
	BRISSONNEAU Pierre	Physique
	GAGNAIRE Didier	Chimie Physique
Mme.	KOFLER Lucie	Botanique
MM.	DEGRANGE Charles	Zoologie
	PEBAY-PEROULA Jean-Claude	Physique
	RASSAT André	Chimie Systématique
	DUCROS Pierre	Cristallographie Physique
	DODU Jacques	Mécanique Appliquée I. U. T.
	ANGLES D'AURIAC Paul	Mécanique des Fluides
	LACAZE Albert	Thermodynamique
	GASTINEL Noël	Analyse numérique
	GIRAUD Pierre	Géologie
	PERRET René	Servo-mécanisme
	PAYAN Jean-Jacques	Mathématiques Pures

PROFESSEURS SANS CHAIRE

MM.	GIDON Paul	Géologie
Mme.	BARBIER Marie-Jeanne	Electrochimie
Mme.	SOUTIF Jeanne	Physique
	COHEN Joseph	Electrotechnique
	DEPASSEL R.	Mécanique des Fluides
	GLENAT René	Chimie
	BARRA Jean	Mathématiques Appliquées
	COUMES André	Electronique
	PERRIAUX Jacques	Géologie et Minéralogie
	ROBERT André	Chimie Papetière
	BIARREZ Jean	Mécanique Physique
	BONNET Georges	Electronique
	CAUQUIS Georges	Chimie Générale
	BONNETAIN Lucien	Chimie Minérale
	DEPOMIER Pierre	Physique Nucléaire-Génie Atomique
	HACQUES Gérard	Calcul numérique
	POLOUJADOFF Michel	Electrotechnique
Mme.	KAHANE Josette	Physique
Mme.	BONNIER Jane	Chimie
MM.	VALENTIN Jacques	Physique
	REBECQ Jacques	Biologie
	DEPORTES Charles	Chimie
	SARROT-REYNAULD Jean	Géologie
	BERTRANDIAS Jean-Paul	Mathématiques Appliquées
	AUBERT Guy	Physique

PROFESSEURS ASSOCIES

MM.	RODRIGUES Alexandre	Mathématiques Pures
	MORITA Susumu	Physique Nucléaire
	RADHAKRISHNA	Thermodynamique

MAITRES DE CONFERENCES

MM.	LANCIA Roland	Physique Atomique
Mme.	BOUCHE Liane	Mathématiques
MM.	KAHANE André	Physique Générale
	DOLIQUE Jean Michel	Electronique
	BRIERE Georges	Physique
	DESRE Georges	Chimie
	LAJZEHOWICZ Joseph	Physique
	LAURENT Pierre	Mathématiques Appliquées
Mme.	BERTRANDIAS Françoise	Mathématiques Pures
MM.	LONGEQUEUE Jean-Pierre	Physique
	SOHM Jean-Claude	Electrochimie
	ZADWORNY François	Electronique
	DURAND Francis	Chimie Physique
	CARLIER Georges	Biologie végétale
	PFISTER Jean-Claude	Physique
	CHIBON Pierre	Biologie animale
	IDELMAN Simon	Physiologie animale
	BLOCH Daniel	Electrotechnique I. P.
	MARTIN-BOUYER Michel	Chimie (C. S. U. Chambéry)
	SIBILLE Robert	Construction mécanique (I. U. T.)
	BRUGEL Lucien	Energétique I. U. T.
	BOUVARD Maurice	HYdrologie
	RICHARD Lucien	Botanique
	PELMONT Jean	Physiologie animale
	BOUSSARD Jean-Claude	Mathématiques Appliquées (I. P. G.)
	MOREAU René	Hydraulique I. P. G.
	ARMAND Yves	Chimie I. U. T.
	BOLLIET Louis	Informatique I. U. T.
	KUHN Gérard	Energétique I. U. T.
	PEFFEN René	Chimie I. U. T.
	GERMAIN Jean-Pierre	Mécanique
	JOLY Jean-René	Mathématiques Pures
Melle.	PIERY Yvette	Biologie animale
	BERNARD Alain	Mathématiques Pures
	MOHSEN Tahsin	Biologie (C. S. U. Chambéry)
	CONTE René	Mesures Physiques I. U. T.
	LE JUNTER Noël	Génie Electrique Electronique I. U. T.
	LE ROY Philippe	Génie Mécanique I. U. T.
	ROMIER Guy	Techniques Statistiques quantitatives I. U. T.
	VIALON Pierre	Géologie
	BENZAKEN Claude	Mathématiques Appliquées
	MAYNARD Roger	Physique

MM.	DUSSAUD René	Mathématiques (C. S. U. Chambéry)
	BELORIZKY Elie	Physique (C. S. U. Chambéry)
Mme.	LAJZEROWICZ Jeannine	Physique (C. S. U. Chambéry)
M.	JULLIEN Pierre	Mathématiques Pures
Mme.	RINAUDO Marguerite	Chimie
MM.	BLIMAN Samuel	E. I. E.
	BEGUIN Claude	Chimie Organique
	NEGRE Robert	I. U. T.

MAITRES DE CONFERENCES ASSOCIES

MM.	YAMADA Osamu	Physique du Solide
	NAGAO Makoto	Mathématiques Appliquées
	MAREZIO Massimo	Physique du Solide
	CHEECKE John	Thermodynamique
	BOUDOURIS Georges	Radioélectricité
	ROZMARIN Georges	Chimie Papetière

A Elise et Christiane
mes bons augures.

"Le tourment qu'on feint d'Ixion,
N'approche de ma passion,
Et mieux j'aimerais de Tantale
Endurer la peine fatale
Un an, qu'être un jour amoureux,
Pour languir autant malheureux
Que j'ai fait, depuis que Cassandre
Tient mon coeur et ne veut le rendre"

P. De RONSARD

"Les amours de Cassandre"

Je tiens à exprimer ici toute ma reconnaissance à :

Monsieur Le Professeur J. KUNTZMANN, Directeur du Service de Mathématiques Appliquées de l'Université de Grenoble, qui a dirigé ce travail et qui, par son aide et ses précieux conseils m'a permis de le mener à bien.

Je suis particulièrement sensible à l'honneur qu'il m'a fait en acceptant de présider le Jury.

Je remercie vivement

Monsieur C. BENZAKEN, Maître de Conférence à la Faculté des Sciences de Grenoble,

Monsieur L. BOLLIET, Maître de Conférence à l'Institut Universitaire de Technologie de Grenoble,

qui ont bien voulu accepter de faire partie du Jury.

Je tiens à remercier aussi la D.G.R.S.T., qui par son aide financière a soutenu la présente recherche.

Celle-ci s'est déroulée en collaboration fructueuse avec la CFTH-HB. La part de Monsieur LUSTMAN dans ce travail, est trop importante pour être détaillée, ses publications en font foi [57], [59] qu'il reçoive l'expression de ma reconnaissance. J'associerai à cet hommage MM. SOKOLOFF, LETELLIER ET HARRAND.

Nous avons bénéficié à diverses reprises des travaux de l'équipe de programmation et de l'aide de MM. SIRET, COLMERAUER, GRIFFITHS et PELTIER.

Enfin ce travail participe d'un effort d'équipe dont il serait difficile de dissocier les apports d'une part, de M. ANCEAU, d'autre part de MM. ARCHER, CHATELIN, COMBES, COUTURIER, DESCHIZEAUX, DOUSSY, LIDDELL, PAYAN, PERRON, et de Madame De POLIGNAC. [2] [3] [4] [5] [56] [78].

J'essaierai de signaler les nombreux emprunts que je leur fait dans le présent ouvrage. Qu'ils soient assurés en tout cas de ma gratitude.

Merci à Monsieur RASOLONJATOVO pour les beaux dessins qu'il a effectués, à Mme CURTET pour sa frappe soignée et au service de Tirage qui a assuré la réalisation de ce document.

TABLE DES MATIERES

INTRODUCTION

A - ANALYSE BIBLIOGRAPHIQUE ---------- 1-a
B - CARACTERISTIQUES D'UN LANGAGE DE CONCEPTION ---------- 15-a
B-1- Langage commun ---------- 15-a
B-2- Système software associé au langage ---------- 15-a
B-3- Conception des nouvelles structures de machines---- 16-a
B-4- Domaines d'applications concernés par le langage de conception ---------- 17-a
C - TRAVAUX CONTEMPORAINS OU POSTERIEURS A CASSANDRE -------- 18-a

DEFINITION DU LANGAGE CASSANDRE

CONVENTION D'ECRITURE ---------- 1-b
ELEMENTS TERMINAUX---------- 1-b
CONSTANTES BOOLEENNES---------- 3-b
IDENTIFICATEURS---------- 5-b
VARIABLES---------- 6-b

NOTIONS STRUCTURELLES

A - UNITE---------- 13-b
Connexions et imbrications d'unités---------- 13-b
B - MATERIEL DECRIT---------- 19-b
Declarations et specifications ---------- 22-b
a) declarations des variables de type 1---------- 22-b
b) déclarations des variables de type 2---------- 24-b
c) déclarations de variables de type 3 ---------- 24-b
d) Déclarations des variables de type 4---------- 25-b
e) Variables arithmétiques---------- 25-b

REMARQUES ---------- 26-b
ROLE PARTICULIER DES ENTREES-SORTIES---------- 26-b
PORTEE DES IDENTIFICATEURS ET DES ETIQUETTES---------- 26-b
PROPRIETES PARTICULIERES---------- 27-b
CONCLUSION---------- 29-b
SURVARIABLES, SOUSVARIABLES, VARIABLES GENERALISEES---------- 33-b
DEFINITION DE LA COMPTABILITE DE 2 VARIABLES ---------- 33-b

a) équivalence de variables ---------- 33-b
b) concatenation ---------- 34-b
c) transposition ---------- 35-b
d) variable generalisée ---------- 36-b

NOTIONS FONCTIONNELLES ---------- 39-b
A - NIVEAU BOOLEEN(OPERATEURS EXPRESSIONS)---------- 41-b
a) Operateurs booléens---------- 41-b
b) expression des opérateurs booléens dans les 3 bases suivantes $(\wedge,V,\neg)$, $(V,\neg)$, $(\wedge,\neg)$.---------- 43-b
c) opérateurs non-booléens---------- 44-b
d) opérateurs condition, réduction, incrementation------ 46-b
d_1) l'opérateur condition ---------- 46-b
d_2) l'opérateur de réduction ----------47-b
e) expressions booléennes ---------- 47-b
d_3) l'opérateur incrementation ---------- 49-b
B - NIVEAU ELEMENTAIRE D'OPERATIONS ---------- 55-b
a) Affectation élémentaire ---------- 56-b
b) Connexion de signal. Impulsions conditionnées---------- 58-b
c) connexion d'unité ---------- 62-b
d) Signal-Unité ---------- 67-b
e) Ordre Allera---------- 69-b
f) Ordre Faire---------- 73-b
C - NIVEAU INSTRUCTION---------- 81-b
a) Opération conditionnelle élémentaire---------- 81-b
b) Instruction élémentaire---------- 82-b
c) Instruction conditionnelle---------- 84-b
d) instruction de connexion, instruction d'affectation, état---------- 87-b
e) généralisation de l'instruction conditionnelle------- 89-b
f) connexion d'unité sous condition, signal unité-------- 91-b
D - NIVEAU AUTOMATE ---------- 93-b
a) liste d'états---------- 93-b
b) graphe associé à une unité ---------- 95-b
d) ordre faire, séquences simultanées---------- 98-b
e) forçage d'un état dans une unité ----------101-b
f) Conclusion sur l'unité ----------103-b

NOTIONS FORMELLES ---------- 105-b

A- NOTIONS ARITHMETIQUES MACRO-NOTIONS---------- 107-b

a) Expressions arithmétiques---------- 108-b

b) expression arithmétique de relation, instruction arithmétique conditionnelle ---------- 109-b

c) Instruction pour ---------- 110-b

d) Introduction des notions arithmétiques dans CASSANDRE ---------- 111-b

e) généralisation de $\perp$, opérateur Δ ---------- 113-b

f) Macro constantes. Macro traitement ---------- 115-b

B - DESCRIPTION FORMELLE DE LA SYNTAXE ---------- 118-b

a) opérateur liste ---------- 118-b

b) opérateur condition---------- 122-b

c) description d'une grammaire de CASSANDRE, utilisant l'opérateur liste ---------- 125-b

d) Quelques propriété sémantique de CASSANDRE utilisant l'opérateur condition---------- 132-b

APPLICATION

I - COMPILATION DE CASSANDRE EN EQUATIONS BOOLEENNES - AIDE A A SYNTHSES ET A L'IMPLANTATION ---------- 1-c

A - PROCESSUS DE SYNTHESE ---------- 1-c

B - TRADUCTION EN CASSANDRE BOOLEEN ---------- 3-c

C - AIDE A L'IMPLANTATION ---------- 8-c

II - SIMULATION DES SYSTEMES LOGIQUES ---------- 25-c

A - SIMULATION D'UN LANGAGE DE TRANSFERT DE REGISTRE------ 27-c

DESCRIPTION EN CASSANDRE SIMPLIFIE ---------- 28-c

B - SIMULATION EN FORTRAN DE CASSANDRE 67---------- 32-c

C - SIMULATION SYNCHRONE GENERALE DE CASSANDRE---------- 33-c

D - SIMULATION ASYNCHRONE DE CASSANDRE ---------- 33-c

E - SIMULATION D'UN ENVIRONNEMENT POUR UN SYSTEME DE PROGRAMMES - OPTIMISATION GLOBABLE HARDWARE - SOFTWARE---------- 34-c

F - UTILISATION DU PROGRAMME ECRIT DANS A POUR SIMULER L'ADDITION DE 2 NOMBRES DE 16 BITS ---------- 38-c

III - APPLICATIONS A LA MICROPROGRAMMATION---------- 41-c

A - AIDE A LA CONCEPTION DE MACHINES CONTROLEES PAR DES MICROPROGRAMMES ---------- 42-c

B - DESCRIPTION DE PASCAL ---------- 43-c

C - PASSAGE D'UN BLOC-DIAGRAMME DE CALCULATEUR MICROPROGRAMME A UN RESEAU DE NOEUDS ET D'ETOILES---------- 47-c

D - BLOC DIAGRAMME DU X2030|WIRTH-WEBER| --------- 51-c

E - REMARQUES SUR LA COMPILATION DE MICROPROGRAMMES DECRITS EN CASSANDRE EN MICRO-INSTRUCTION D'UNE MACHINE DONNEE---------- 61-c

IV - AUTRES APPLICATIONS : CONCLUSION ---------- 68-c

A - TEST DE CIRCUIT LOGIQUES DECRIT EN CASSANDRE------- 68-c

B - REALISATION D'AUTOMATISMES SEQUENTIELS ---------- 69-c

C - VISUALISATION DES RESEAUX D'UNITES et des RESULTATS 70-c

CONCLUSION ---------- 75-c

BIBLIOGRAPHIE ----------

TABLE DES FIGURES

Syntaxe de la constante booléenne ---------- 9-b
Syntaxe de l'entier ---------- 9-b
Bloc diagramme d'une unité ----------15-b
Schéma de masques pour circuit M.O.S.T.----------17-b
Syntaxe de la définition d'unité ----------31-b
Syntaxe de la variable généralisée----------37-b
Opérateurs booléens à 2 variables----------43-b
Circuits correspondants à une expression booléenne----------51-b
Syntaxe de l'expression----------53-b
Circuit additionneur----------66-b
Exemple d'unité----------79-b
Contrôle de séquences simultanées----------99-b
Syntaxe de la partie fonctionnelle----------104-b
Syntaxe de l'expression arithmétique----------108-b
Matrice d'unités connectées----------112-b
Tableau des métavariables d'une syntaxe de CASSANDRE----------126-b
Carte syntaxique de CASSANDRE----------139-b
Circuit de contrôle d'un calculateur---------- 11-c
Réalisation du circuit de contrôle---------- 14-c
" " " 15-c
" " " 16-c
" " " 18-c
" " " 19-c
" " " 20-c
" " " 21-c
" " " 22-c
" " " 23-c
" " " 24-c
Module de calcul d'expressions booléennes---------- 36-c
Bloc-diagramme de la machine Pascal---------- 44-c
" du X2030---------- 51-c
" modifié X2030---------- 52-c
Réseau noeuds-étoiles associé au X2030---------- 53-c
Résultat possible de la visualisation d'un réseau d'unités interconnectées---------- 73-c
Schéma réel du même réseau pour sa réalisation intégrée---------- 74-c

INTRODUCTION

Nous essaierons à travers un ensemble de travaux disponibles en 1966, de montrer l'utilité du travail de synthèse qui a été tenté ici. Ceci nous amènera à une liste de caractéristiques, de fonctions et d'objectifs qui définissent ce que nous appellerons un langage de conception des systèmes logiques.

Nous ferons une première confrontation du langage CASSANDRE qui en a résulté avec les quelques travaux déjà cités qui se poursuivent à ce jour et avec certains commencés depuis.

Nous espérons justifier ainsi "a postériori" certains choix prix et certains infléchissements des directions initiales.

A - ANALYSE BIBLIOGRAPHIQUE.

Nous appuierons cette revue sur deux articles qui sont eux-mêmes des analyses bibliographiques, celui de Breuer [11] (Décembre 66) et celui de Teichroew et Lubin [97] (Octobre 66).

Le premier recouvre tous les domaines de l'aide à la conception des ensembles logiques, il signale les travaux les plus intéressants existants dans chacun de ces domaines et suggère l'intérêt de regrouper ceux-ci à l'intérieur d'un même système. Il apparaît en effet à l'étude des quelques 400 références données par Breuer, que chaque phase de la conception logique a été l'objet d'une automatisation plus ou moins poussée. Seule le niveau le plus abstrait de définition d'un calculateur reste du domaine de l'art et les choix de celui de l'intuition.

Sans prétendre remplacer l'expérience et l'imagination de "l'architecte de machine" il apparait qu'un effort doit encore être fait pour rendre plus efficace cette expérience et plus féconde cette imagination, et offrant au concepteur un outil pratique d'aide à la conception

dès la première phase du travail et non pas seulement dans les diverses techniques qui le prolongent et en achèvent la réalisation.

Il apparait d'autre part indispensable d'améliorer le travail d'équipe entre ces différents techniciens et d'améliorer les échanges d'information entre eux en leur apportant un "langage commun".

Le second article s'interesse à la simulation en général et non aux systèmes logiques en particulier. Mais l'expérience de ses auteurs rend très intéressante l'analyse des conditions de création d'un langage orienté vers une application spéciale et celle de la différence de nature fondamentale qui existe entre les phénomènes discrets et les phénomènes continus.

Par ailleurs il nous apparait que la simulation à différents niveaux d'abstraction d'un système logique en cours de projet, reste la fonction la plus importante de l'aide à la conception de ce système et intervient dans les différentes autres fonctions.

Les auteurs notent d'abord l'aide considérable apportée à la programmation d'algorithmes pour une application spéciale, par l'existence d'un langage spécial approprié à cette application et le développement des programmes de traitement (interpréteurs, assembleurs) qui lui sont associés.

Ils distinguent ensuite le langage spécial lui-même, c'est-à-dire sa définition indépendemment de son traitement sur une machine particulière et le "système" associé à ce langage c'est-à-dire les compilateurs, les manuels et les programmes de détection d'erreurs qui sont fournis à l'utilisateur.

Cette distinction est très importante, car pour beaucoup des travaux que nous étudierons seul le premier aspect (définition d'un langage) existe.

Pour le langage CASSANDRE également il faudra distinguer ce premier aspect qui est le sujet de cet ouvrage, des différents programmes de traitement de CASSANDRE qui constituent d'autres travaux.

"Dans la discution sur les langages il est utile de distinguer trois activités différentes :

1) La spécification et la définition du langage :
2) son installation sur calculateur sous forme d'un système de programmes et de la documentation associée.
3) L'utilisation du langage et du système.

Quelquefois les mêmes personnes sont impliquées dans les 3 activités. Il est pratique de parler de trois groupes de gens, les concepteurs du langage, les réalisateurs du système et les utilisateurs".

"L'utilisation d'un langage est déterminée non seulement par ses caractéristiques mais par celles de son système de traitement et celles des machines sur lesquelles il peut fonctionner".

Les auteurs notent ensuite le caractère insoluble mathématiquement des problèmes de simulation. Ces problèmes comportent beaucoup de variables beaucoup de paramètres des fonctions mal définies. La simulation est en fin de compte une technique. Elle utilise des modèles de deux types : les modèles de phénomènes continus et ceux des phénomènes discrets.

Les modèles continus sont représentés par des systèmes d'équations différentielles ou aux différences. Si c'est possible on utilise l'analyse numérique pour les résoudre sinon on les simule. Pour cette simulation les machines analogiques qui sembleraient bien adaptées, sont inutilisables parce que les fonctions sont souvent discontinues, le nombre de variables trop grand, contenant quelquefois des variables aléatoires....

On simule alors les modèles continus par un système d'équations aux différences finies.

Les modèles discrets sont représentés comme un "réseau" de composants (ou sous systèmes) dont chacun exécute des fonctions définies. Les changements "d'états" du système se font à des instants précis dans le temps, son évolution est donc séquentielle.

La première catégorie contient les modèles mathématiques de circuits électroniques. C'est la simulation qui intéresse les fabricants de composants.

La seconde contient les systèmes logiques, les variables qui y apparaissent alors sont booléennes, ou arithmétiques de type indice.

Leur simulation intéresse les fabricants de calculateurs digitaux ou d'automatismes.

Il est curieux de noter que la coupure qui existe entre les natures de ces deux problèmes, se traduit dans l'industrie par une séparation d'activités de plus en plus grande.

La fabrication des calculateurs devient de plus en plus indépendante de celles des circuits qu'elle utilise.

"Les langages de simulation diffèrent des langages de programmation usuels en cela qu'il contiennent des notions qui facilitent les communications entre celui qui défini les problèmes et celui qui les programme. Ils simplifient aussi la programmation des calculs qui sont caractéristiques des problèmes de simulation."

Voilà deux fonctions "langage commun" et "adéquation au problème traité" qui devront apparaître dans tout langage spécialisé.

Notons pour en terminer avec cette étude qu'"un des apports des langages de simulation devrait être une reflexion nouvelle sur la nature de la simulation des phénomènes discrets". Il est souhaité que le langage permette "la simulation d'activités simultanées", "le traitement de structures de listes et même la description des structures d'arbres".

Toutes caractéristiques que nous nous sommes attachés à introduire dans le langage CASSANDRE.

Les langages de simulation les plus intéressants résultent des travaux de Knuth et Mc Neley [SOL) [50] et Nygaard et Dahl (SIMULA) [74].

Un langage algorithmique, pour permettre la simulation de systèmes logiques, devra permettre en plus de la description d'actions simultannées l'accès aux bits.

Il reste peu de langage de programmation vérifiant ces deux conditions, citons ALGOL W, CPL, PLI, BALGOL et A.P.L.

C'est pourquoi de nombreux travaux ont été publiés à ce sujet, et encore ignore-t-on la plus part d'entre eux.

L'un des plus universellement cité est le langage d'Iverson [44], dans son utilisation pour la description de systèmes [30] [29].

Outre ses qualités de concision APL offre certains opérateurs très pratiques (comme la "Réduction" n repris dans d'autres langages et conservés en CASSANDRE.

Par ailleurs la description d'un système en A.P.L., fournit un programme de simulation de ce système, en mode conversationnel sur une machine IBM 360.

Nous avons étudié par ailleurs [96] les avantages et les inconvénients d'APL pour l'aide à la conception des systèmes logiques. Lorsqu'on veut faire correspondre une réalisation en circuits réels à partir d'une description en APL, on doit, comme l'a fait FRIEDMAN dans le système ALERT ne conserver qu'un sous ensemble très restreint d'APL, lequel sous-ensemble est d'ailleurs commun à beaucoup d'autres langages. On perd alors les qualités qu'APL démontrait à un niveau de description plus général, pour en conserver l'inconvénient d'une syntaxe pauvre et d'un formalisme incommode. Il faut par ailleurs lui rajouter des déclarations (|voir ALERT [31] |) ce qui en fait un autre langage.

En France nous pouvons citer deux travaux dont CASSANDRE s'est plus ou moins inspiré. D'abord une note technique de VANCO (Compagnie Bull), sur la simulation d'une machine à lire des caractères.

Le besoin d'une description évoluant en même temps que le modèle simulé aboutit à un langage décrivant un graphe d'"états" et les transitions entre ces "états". L'information élémentaire s'appelle "signal" nous en retiendrons le mot en CASSANDRE.

Certaines notions intéressantes de "séquences"d'états, de signaux et de registres apparaissent dans cette étude, de mêmes que celles de séquences simultanées et de sous automates contrôlés par un automate principal. Des conditions logiques y sont décrites sous forme d'un polynôme booléen.

Mais le langage employé TINBOOL est très simple et limité à des applications de simulation. Il présente l'inconvénient de mélanger les

fonctions de description du modèle et celles de commande de la simulation.

Une autre étude interessante émane de la CAE avec le système "Epicure".

Epicure, programme de simulation, utilise un langage de description, compile celui-ci puis simule le résultat.

Un processus analogue sera suivi dans les différents programmes de simulation de CASSANDRE qui ont été essayés [57] [3].

Le langage a l'avantage de ne pas mélanger la description de la machine simulée, avec les commandes propres à cette simulation.

Certaines notions du langage de description apparaitront aussi en CASSANDRE.

- Une instruction est une suite de micro-opérations

- on défini des variables statiques : bascules, registres sous-registres mémoire.
  et de variables instantanées appelées signal ou ligne.
- ces variables sont déclarées.
- le langage contient des opérateurs arithmétiques et logiques.
- Un transfert d'information dans un registre se note :
  <Registre> = <Expression booléenne>
- Des instructions conditionnelles sont définies.
- Une instruction est repérée par une "étiquette" le transfert d'une instruction dans une autre se fait par un ordre "allera <étiquette>.
- Un ensemble d'opérations pouvant être simultanées est improprement appelé "séquence" dans le système Epicure. L'instruction CASSANDRE

englobera les propriétés de cette "séquence Epicure".

- Un "programme" comprend une partie déclaration suivie d'une partie action.

Dans le système Epicure la simulation comprend 2 phases :

1) initialisation des variables
2) Sortie des résultats (intermédiaires ou finaux).

- Tout un langage interessant de commande pour cette simulation est aussi défini.

A part le langage d'Iverson et les 2 travaux français cités ci-dessus, un certain nombre de travaux étrangers ont servi de point de départ à notre étude au cours de 1966.

Les plus anciens semblaient alors le langage défini par Gorman et Anderson [38] et le langage de transfert de registres utilisé par Schorr [89].

Le langage de Gorman et Anderson a été par la suite utilisé par Proctor [79].

Nous avons déjà étudié [65] ces travaux, nous avions vu que Proctor établissait au cours de la simulation du système logique décrit (dans le langage de Gorman) un tableau des simultanéités, montrant l'état d'utilisation ou de non utilisation de chaque bascule du système à chaque instant.

Ce tableau s'avère très interessant dans l'étude de la micro-programmation d'une machine et nous retiendrons le travail de Proctor pour cette application.

Schorr a montré comment relier une description formelle d'un automate avec les algorithmes connus pour en faire la synthèse.

Nous ferons une approche équivalente en CASSANDRE pour la partie "contrôle" des unités. On considère la description comme une machine séquentielle et on en extrait par un traitement d'analyse le graphe des transitions entre états sur lequel on applique les méthodes de codage classiques.

Notons chez IBM à Yorktown Heights les travaux de J.P. Roth [81], qui seront suivis de ceux de Friedman [31] déjà cités.

Citons encore en 1965, une thèse de Burnett [14] au MIT, un article de Mc Clure et le travail de Zucker (lui aussi d'IBM Yorktown Heights) sur LOCS [106].

Zucker et une équipe d'environ 8 personnes ont mis au point un système permettant de décrire et de simuler sur 7090, à des niveaux variables de complexité, un projet de machine. Ce travail important, mais écrit seulement pour 7090, donne une idée assez précise de ce qui peut être réalisé et des moyens à mettre en oeuvre pour cela.

Locs permettait par exemple de simuler les circuits logiques et les horloges qui les synchronisent, mais également à un niveau beaucoup plus élevé, le code machine du projet étudié.

Le software de la machine en projet pouvait ainsi être testé sur le simulateur et suggérer des corrections dans le projet permettant d'envisager une optimisation hardware - software.

Mais les performances d'une telle simulation n'ont pas été indiquées ce qui enlève de l'intérêt à l'étude. Les caractéristiques du langage employé se retrouveront également chez CHU [22].

Signalons encore un travail de Estrin et Mandell [28], interessant par la démarche suivie, (puisque pour la mise au point des programmes de traitement de CASSANDRE, un metacompilateur (Griffiths-Peltier) a également été utilisé), et un rapport de Wilber [103] à l'université de l'Illinois.

Comme pour LOCS, les travaux de Parnas [75] Metze et Seshu [69] et Allan Giese [35], s'appuient sur un langage de type Algol.

L'accent est mis sur la nécessité d'avoir un formalisme commode, le plus proche possible du langage naturel, pour faciliter au non spécialiste l'emploi de cet outil de conception aidée.

Parnas en définissant le langage "System Function Description-Algol" ou SFD-Algol, veut utiliser le fait qu'Algol est largement connu, facile à apprendre et indépendant d'une machine donnée. Mais tout en cherchant à rester le plus près possible d'Algol il faut tout de même, lui ajouter la possibilité de décrire des actions simultanées et aussi certaines déclarations de variables entrées et sorties des modules du système décrit.

Retenons ces 2 adjonctions et la conclusion de l'article. "Lorsqu'on est certain de l'exactitude des spécifications décrites dans ce langage, on peut les utiliser comme entrée d'un processus complètement automatisé de conception. On a utilisé la description en SFD. Algol pour produire le tableau d'états des systèmes d'états finis. La littérature contient des algorithmes pour produire des circuits logiques à partir de tels tableaux. Le résultat final pourra être que la conception des systèmes à nombre finis d'états consistera seulement à les décrire en SFD. Algol et à tester par simulation l'exactitude de cette description. Le reste du travail de conception sera automatique."

On retrouve cet essai d'automatisation complète dans le "compilateur de calculateur" de Metze et Seshu "Le compilateur de calculateur traduira une description de système, donnée essentiellement dans un langage proche du langage naturel des manuels de programmation, en une description du hardware c'est à dire de protes ET, OU, NI, de bistables et de leurs interconnections". Toutefois la certitude exprimée dans cet article d'arriver par programme à une optimisation plus poussée du hardware semble quelque peu utopique. Par ailleurs le langage utilisé nous semble trop élémentaire et pas assez concis.

C'est un travail dans le même esprit que les précédents que l'on trouve encore avec Hargol (Hardware - Oriented- Algol) d'Allan Giese "Hargol est une extension d'Algol en ce sens qu'il inclu les notions nécessaires pour la description des systèmes Hardware".

Outre celles déjà vues par SFD-Algol on trouve l'extension des opérations définie en Algol sur des variables scalaires à des variables tableaux de dimensions quelconques. Comme dans le langage d'Iverson elles s'appliquent alors composante à composante.

Ces langages d'une façon générale sont intéressants par leur formalisme facile à comprendre et à utiliser. Mais ils semblent tous avoir été conçus par des gens du software ; d'une façon générale nous leur reprocherons de ne pas avoir une signification hardware assez précise.

Ils sont non ambigus au niveau fonctionnel mais très ambigus au niveau circuits logiques. Mais toutes les caractéristiques interessantes que nous avons déjà soulignées vont se retrouver avec quelques autres dans les travaux de CHU [22] et SCHLAEPPI [87] que nous allons voir maintenant.

Le "langage CDL (Computer design langage) est un langage de type ALGOL réunissant en quelque sorte les caractéristiques des travaux de Panas, Giese et du système Epicure et certains opérateurs (comme la Réduction) du langage d'Iverson.

Nous retiendrons sans faire l'analyse de ce travail maintenant assez connu les buts mentionnés par CHU :

1) Définir des standards pour préciser la conception de la logique d'un système digital. Un tel langage standard facilitera la documentation associée au projet aussi bien que la communication entre les concepteurs (aucun programme de traduction n'est nécessaire pour cette fonction).

2) Simulation de la logique et élimination des erreurs. Si on prévoit un simulateur général qui accepte en entrée le langage, le temps mise au point du projet sera beaucoup raccourci.

3) Evaluation du coût et des performances. Il est très souhaitable d'évaluer les caractéristiques de chaque partie du système en projet. La simulation logique sera utilisée pour évaluer les performances en un calcul du coût se fera à partir des équations booléennes.

4) Intégration des diverses phases de la conception. Les calculateurs ont été employé depuis longtemps dans chaque technique particulière de la conception. Avec un langage de conception les opérations de traduction en équations booléennes, simulation, vérification de la logique, modification du projet, estimation du coût, évaluation des performances implantation des cartes, câblâge des panneaux et documentation pourront être automatisées à l'intérieur d'un même système.

5) Conception de machines plus complexes : La conception de système digitaux disons dix fois plus compliquée que les systèmes existants est trop difficile pour être traitée par les méthodes conventionnelles et dans ce cas le projet serait criblé d'erreurs. L'utilisation d'un langage de conception est indispensable dans ce but."

-.- Une description dans le langage de CHU commence par un ensemble de déclarations de registres, sous-registres, mémoires et signaux suivies d'une liste d'instructions étiquetées chaque étiquette représente une condition logique permettant d'exécuter simultanément un certain nombre d'opérations. Des instructions conditionnelles existent et les opérations s'effectuent composante à composante sur les variables non scalaires. L'ensemble des possibilités de ce langage sera retenu en CASSANDRE mais la généralisation sera beaucoup plus poussée dans les instructions en particulier. Par ailleurs, une description en C.D.L. est un tout, segmentable en séquences correspondantes aux différentes opérations mais non pas en modules pouvant être mis au point chacun séparément. Ceci nous semble limiter l'application du langage à certains problèmes assez peu compliqués.

Il est également impossible de décrire commodément des opérations asynchrones.

Pour conclure sur ce travail nous insisterons sur son aspect pédagogique et nous chercherons à conserver cet aspect sur une forme simplifiée de CASSANDRE.

-.- Le langage LOTIS défini par SCHLAEPPI représente un travail plus précis et mieux étudié du point de vue hardware que le langage de CHU, mais son aspect est peut-être moins commode. Une étude très complète

des modes d'opérations synchrones, asynchrones ou d'une combinaison des deux est faite. Une correspondance précise est donnée entre chaque élément de la description et l'élément correspondant de la réalisation. SCHLAEPPI note l'importance d'une double sémantique à l'intérieur d'une même description, l'une exprimant le fonctionnement et l'autre la réalisation physique. Il insiste enfin sur la définition d'une hiérarchie dans la description qui corresponde à la hiérarchie qui existe dans la machine-objet. On peut dans le langage LOTIS tenir compte des retards de chaque élément de circuit et simuler le système décrit même au niveau le plus fin. Une distinctions très nette est faite entre les variables qui ont une certaine permanence comme les registres et la mémoire et les variables qui n'ont qu'une valeur instantanée comme les signaux et les bus. La description des opérations en pas et en séquences est plus générale que dans le langage de CHU. Les appels de procédures sont permis à l'aide des séquences et différents groupes représentent chacun un automate de contrôle regroupant un sous-ensemble de séquences. Il semble toute fois que la coordination des actions entre les différents automates et en particulier la possibilité de forcer de l'un d'eux un état à l'intérieur d'un autre n'ait pas été étudié de façon très précise par l'auteur. Même après discussion avec lui de nombreux problèmes restaient posés à ce sujet. Enfin, quel que soit l'intérêt de cette étude elle reste une approche théorique n'ayant fait l'objet d'aucune réalisation programmée.

B - CARACTERISTIQUES D'UN LANGAGE DE CONCEPTION

Des caractéristiques qui servent se dégage de l'analyse des travaux précédents nous retiendrons les suivants :

B-1 Un langage de conception doit être un langage commun à toutes les personnes concernées par un des aspects de la conception des systèmes logiques. Cette propriété générale implique d'une façon plus précise que le langage soit proche du langage naturel (par ex. de type ALGOL) car il devra être compris et utilisé par des gens qui ont peu de notions de Software. Celà implique aussi que dans chaque technique la sémantique de la description est parfaitement précisée. Une même description aura donc autant de significations que l'on en aura prévu d'applications différents.

Si la fonction de langage commun est vérifiée cela facilitera considérablement l'échange d'informations entre les différentes équipes travaillant sur un même projet en diminuant les erreurs d'interprétation. Toute la documentation relative au projet sera établie dans le même langage et disponible dès la fin de l'étude.

B-2 Un langage de conception pour être utilisé doit être accompagné d'aides programmés accélérant et facilitant le travail dans chacun des domaines de la conception. Un système software sera associé au langage qui intégrera les méthodes actuellement employée de façon séparées. Nous irons beaucoup plus loin que CHU dans les implications de cette deuxième caractéristique. D'abord une description dans le langage par différentes analyses et interprétations devra être rendue syntaxiquement correcte et logiquement cohérente. Ensuite si l'on veut que l'ensemble des aides programmés constitue véritablement un système de conception aidée et non pas un ensemble de programmes mis bout-à-bout : on ne devra pas sortir

du langage. Dans un traitement du langage CASSANDRE la chaîne d'entrée est toujours un texte en CASSANDRE et la chaîne de sortie un autre texte, en CASSANDRE. Par ex : Dans la traduction en équations booléennes l'entrée est en CASSANDRE le plus général, la sortie en CASSANDRE Booléen. Dans une compilation de micro-programme les algorithmes décrits en CASSANDRE pour (description 1) donneraient des micro-programmes décrits en CASSANDRE (description 2) exécutables sur une machine elle-même décrite en CASSANDRE (description 3). Dans l'application à la synthèse des circuits logiques les fonctions décrites en CASSANDRE donneront comme résultats des circuits logiques eux-mêmes décrits en CASSANDRE.

De cette façon la chaîne de sortie d'un traitement quelconque peut toujours servir de chaîne d'entrée dans n'importe quel autre traitement c'est pour celà que les sémantiques différentes du langage doivent chacune recevoir une définition beaucoup plus précise que dans chacun des travaux déjà étudiés pour aucun desquels l'intégration de la conception n'atteint le degré que nous souhaitons.

B-3 Conception des nouvelles structures de machines. Pour remplir au mieux cette fonction déjà souhaitée par CHU le langage de conception et son système de traitement devront être conçus en vérifiant certaines implications précises. D'abord : Tout système décrit devra être segmentable, en partie aussi petites que l'on veut, totalement indépendantes du reste du système. C'est ce qui a entraîné la définition de la notion d'Unité, le contenu d'une unité pouvait être arbitrairement simple ou compliqué. L'interface de l'unité avec tout ce qui l'entoure étant définie par les signaux qui apparaissent dans l'entête d'unité (Notons en passant que dans le travail de Metze la fonction d'interface était déjà remplie entre plusieurs modules décrits par certains registres déclarés interfaces, la rigueur de définition de l'unité impose que l'interface

soit réduite strictement à des variables de type signal.

La deuxième implication est que les algorithmes du système de traitement du langage de conception, constitueront plus souvent des aides que des méthodes automatiques. Le système de traitement devra donc être conversationnel pour permettre l'intervention du concepteur à tous les niveaux, celui-ci libéré des travaux les plus fastidieux et du risque d'erreurs pourra imaginer des structures beaucoup plus complexes que celles qui existent déjà. Le langage de conception devra lui permettre de les décrire de façon concise, c'est pourquoi, nous pensons qu'il faut aller plus loin dans la généralisation des notions qui apparaissent dans les langages déjà étudiés, qui ont été conçus pour la description de systèmes logiques existant déjà.C'est ce qui a été tenté en CASSANDRE, que ce soit au niveau des variables, des expressions, des instructions conditionnelles ou des unités.

B-4 Les objets décrits par un langage de conception des systèmes logiques seront de nature discrète et booléenne. Les traitements feront un usage intensif de listes. Puisque chaque traitement fait passer d'un texte à un autre texte tous deux vérifiant une syntaxe. ces traitements s'apparenteront à des compilations. Les domaines d'applications concernés par le langage de conception seront : la théorie des automates et des machines séquentielles, le calcul booléen, la synthèse des circuits logiques, le découpage en modules, l'implantation en circuits intégrés, circuits imprimés ou panneaux câblés, le test de bon fonctionnement des circuits logiques, la microprogrammation, la simulation à tous les niveaux où les modèles restent des modèles discrets c'est à dire de la simulation de circuits réels en tenant compte des retards de programmation, jusqu'à l'exécution de programmes sur la machine simulée écrits dans le langage de cette machine en projet. Dans chacun de ces domaines le langage pourra pourra servir de base pour l'enseignement de la conception des systèmes logiques.

C - TRAVAUX CONTEMPORAINS OU POSTERIEURS A CASSANDRE

Parmi les travaux déjà analysés dans A) un seul fait l'objet d'une utilisation à l'heure actuelle : c'est le travail de CHU. Un simulateur de CDL a été mis au point par MEZTENYI [68].

De même à l'université de Maryland a été réalisé un traducteur de CDL en équations booléennes. Le travail continue sur ce sujet dans cette université et Le Professeur CHU se sert de CDL pour l'enseignement de la conception des machines. Il est probable par ailleurs que les travaux se servant de description en langage d'IVERSON se poursuivent chez IBM. Bien que FRIEDMAN après avoir mis au point le programme ALERT ait abandonné le sujet. Enfin ALLAN GIESE poursuit à Copenhague l'utilisation de HARGOL dont la nouvelle version contient des possibilités de simulation de modèles asynchrones mais il semble que la simulation soit la seule application envisageable.

PARNAS semble avoir abandonné les problèmes de conception aidée de systèmes logiques et le langage SFD Algol pour l'étude dans toute sa généralité de la simulation des modèles discrets. Parmi les travaux contemporains à CASSANDRE signalons la méthode très intéressante de GERACE [34] qui extrapole les travaux de SCHORR il traduit la description d'une machine sous forme d'un langage de transferts de registres en un tableau d'états de systèmes séquentiels puis applique un algorithme de codage des états et de synthèse de l'automate. Les travaux se poursuivent à PISE dans l'équipe du professeur GERACE.

MANDELL travaille sur un sujet analogue à l'Université de LOS ANGELES (UCLA) et devrait passer une thèse sous peu [28]. On a vu apparaître en septembre 1968 [25] un article de J.R. DULEY et D.L. DIETMEYER sur le langage D.D.L. Il est intéressant de constater

dans cet article des analogies frappantes avec la version du langage CASSANDRE datant de la même époque en particulier pour les instructions généralisées. Comme il est extrêment improbable que les informations aient pu être échangées entre les deux travaux cette étude constitue une excellente confirmation à postériori de l'intérêt de certaines notions introduites en CASSANDRE. Un autre article [26] des mêmes auteurs étudie la traduction d'une description D.D.L. en équations booléennes. Là encore l'analogie avec la méthode employée par Monsieur ARCHER et nous-mêmes [65] est très nette.

Un travail important effectué sur un contrat de l'AIF FORCE dans les laboratoires RCA à Princeton par C.U. SRINIVASAN [94] définit C.D.L. 1 un langage pour la description de calculateurs digitaux. Nous ne pouvons pas dire dans quel état d'avancement se trouve la réalisation du système de traitement associé à ce langage. Nous savons que GORMAN a participé à ce travail durant son passage chez R.C.A.

Parmi les travaux les plus récents citons : S.D.L. langage défini par D.F. GORMAN qui a fait l'objet d'un PhD à l'université de Pennsylvanie en 1968 [39] c'est le langage le plus général que nous ayons rencontré sur ce sujet. Les domaines de conception aidée où il est concerné sont tout à fait analogues à ceux qui ont été définis pour CASSANDRE mais il n'a fait l'objet d'aucun traitement programmé à ce jour.

Ces applications sont néanmoins envisagées par l'auteur. Il est intéressant de noter que celui-ci est l'un de ceux qui ont réalisé les travaux les plus anciens sur ce sujet [38].

Enfin pour terminer signalons une étude commencée fin 1963 par WATERLOT et SARRE [101]. Le langage CRISMASS qu'ils utilisent est une extension de FORTRAN. La simulation de ce langage s'inspire du générateur de simulateur en FORTRAN étudié par LUSTMAN [57] pour le langage CASSANDRE.

DEFINITION DU LANGAGE CASSANDRE

-:-:-:-:-:-:-

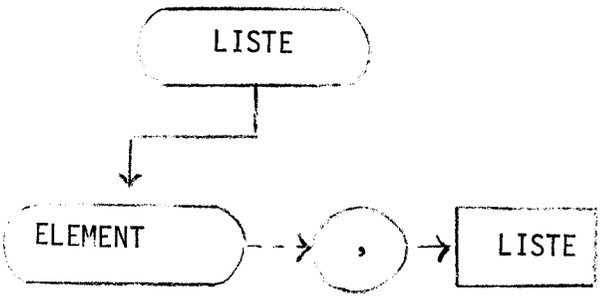
Convention d'Ecriture

Les règles de syntaxe seront mise sous la forme normale de Backus :

<METAVARIABLE> ::= <DEFINITION1>|<DEFINITION2>

Le symbole | sépare les différentes alternatives.

On utilisera la récursivité droite, par exemple pour définir les listes. Chaque liste pourra se réduire à un seul élément.
(d'où le trait pointillé dans le schéma ci-contre).



<LISTE> ::= <ELEMENT>|<ELEMENT>,<LISTE>

Eléments terminaux

Nous pouvons fixer la liste des éléments terminaux comme étant celle ci-dessous.

Il n'est pas question de considérer celle-ci comme fixe et intangible, le choix des symboles répond simplement à certaines traditions maintenant crées, peu importe de changer le vocabulaire au gré des préférences d'un utilisateur particulier, les concepts restent les mêmes. Les programmes de traitement commenceront par un petit éditeur facilement modifiable dans ce but.

<TERMINAL> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|
0|1|2|3|4|5|6|7|8|9|—|/|↑|↑|_|≃|↑|→|τ|V|Λ|=|≠|>|<|"|
<=|≥|≤|V|Λ|&|*|,|;|:|(|)|I|I|→|←|Φ|ε|×|+|-|÷|Rem|≡|
si|allera|pour|égal|pas|à|faire|disque|de|unité|signal|Registre|Impulsion
Etat|Externe|Mémoire|Horloge|Bus|Synchrone|Asynchrone|Synchronisé|Etatinitial

On peut subdiviser ces terminaux de la façon suivante :

<Symbole de base> :: <LETTRE>|<CHIFFRE>|<DELIMITEUR>|<SYMBOLE-SPECIAL>
<DELIMITEUR> ::= <OPERATEUR>|<SEPARATEUR>|<DECLARATEUR>|<SPECIFIEUR>|<OPTION>
<OPERATEUR> ::= <OPERATEUR-MONADIQUE>|<OPERATEUR-DYADIQUE>
<LETTRE> ::= A|B|.....|Z
<CHIFFRE> ::= 0|1|.....|9
<OPERATEUR-MONADIQUE> ::= ¬|/|↑|↑|⊥|∿|↓|↱|τ
<OPERATEUR-DYADIQUE> ::= V|Λ|=|≠|≥|≥|≤|<|Λ|V|&
<SEPARATEUR> ::= |,|;|:|(|)|I|I|→|←|si|fin|début|allera|pour|égal|pas|à|faire|
|"|<=
<DECLARATEUR> ::= Unité|Signal|Registre|Etat|Impulsion
<SPECIFIEUR> ::= Externe|Horloge|Mémoire|Bus|Etat initial
<OPTION> ::= Synchrone|Asynchrone|Synchronise>
<SYMBOLE-SPECIAL> ::= ϕ|ε|<OPERATEUR-ARITHMETIQUE>
<OPERATEUR-ARITHMETIQUE> ::=|+|-|×|÷|Rem

Des commentaires pourront être insérés à n'importe quel endroit d'un texte en CASSANDRE ; il devront être délimités à droite et à gauche par le symbole ".

Constantes booléennes :

Le langage contient une ambiguïté volontaire dans la signification des symboles 1 et 0 qui peuvent être aussi bien les valeurs logiques vrai et faux, les chiffres en base 2 ou les 2 premiers chiffres décimaux. Leur signification est toujours clairement définie par le contexte, l'ambiguïté est donc toujours levée et elle est compensée par plusieurs avantages.

Les constantes booléennes en CASSANDRE seront en général des tableaux de constantes scalaires. On distinguera dans la syntaxe, constantes scalaires, vecteurs ou tableaux.

<CONSTANTE-SCALAIRE> ::= 0|1|ϕ

ϕ est la valeur booléenne indéterminée.

<CONSTANTE-VECTEUR> ::= <CONSTANTE-SCALAIRE>|<CONSTANTE-SCALAIRE> <CONSTANTE-VECTEUR>

C'est une succession de 1, 0 ou de ϕ.

<CONSTANTE-TABLEAU> ::= <CONSTANTE-VECTEUR>|<CONSTANTE-VECTEUR>, <CONSTANTE-TABLEAU>

EXEMPLES :

a) (0 1 1 ϕ ϕ 1 0 0 ϕ 1) est un vecteur de dimension 10

b)
1 0 0 1	1 1 0 0	ϕ ϕ 0 0	0 1 0 0
0 1 1 ϕ	1 1 1	1 0 1 0	0 0 1 0
1 0 1 0	0 0 0 0	0 0 0 1	0 1 0 0

est un tableau de dimensions (4, 3, 4).

Il s'écrira : (1001, 011ϕ 1010, 1100 1ϕ11 0000,
$\phi\phi$00 1010 0001 0100 0010 0100)

Comment l'analyseur retrouve-t-il les dimensions ?

-.- La première est le nombre de digits avant la 1ère virgule soit x_1.

-.- La deuxième le nombre de digits avant la 2ème virgule, soit x_2, divisé par le 1er nombre, soit x_2/x_1.

-.- La troisième le nombre de digits avant la 3ème virgule x_3 divisé.

-.- etc. x_2.(ici : $x_1 = 4$, $x_2 = 12$, $x_3 = 48$).

Ecriture abrégée des constantes en octal ou hexadécimal : (Voir chapitre iV).

SEMANTiQUE

On peut définir des constantes par branchement de fils à la masse ou des tensions disponibles dans le montage. Mais ce ne sont pas les seules contantes. Celles-ci peuvent également apparaître comme "format" dans les équations logiques ; leur action est purement formelle.

Enfin il est correct d'écrire la condition : "Si A = 1" bien que ce soit redondant et qu'il suffise d'écrire "si A", mais la clarté de la description peut en être améliorée (1 exprime ici la valeur vrai).

Elles peuvent également apparaître en condition logique pour tester la valeur d'une variable vectorielle,

EXEMPLE : si Λ /(R = (01101 10)) alors ...

Ceci est plus concis que la forme :

$\neg R(0) \Lambda R(1) \Lambda R(2) \Lambda \neg R(3) \Lambda R(4) \Lambda R(5) \Lambda \neg R(6)$

Nombres :

<ENTIER> ::= <ENTIER-SANS-SIGNE>-<ENTIER-SANS-SIGNE>
<ENTIER-SANS-SIGNE> ::= <CHIFFRE><ENTIER-SANS-SIGNE><CHIFFRE>
<CHIFFRE> ::= 0|1|2|3|4|5|6|7|8|9

SEMANTIQUE :

Un nombre sert à définir les bornes d'une variable non-scalaire. Il indique une indexation dans l'ordre pour, ou devant un opérateur monadique. Une paire de nombres devant l'opérateur ∿ indiquera les directions à permuter.

Donc les nombres que nous avons à considérer seront entiers (autres que les temps, mais ceci ne fera pas partie de la version actuelle du langage, et qui plus est, on doit pouvoir se limiter à des nombres entiers d'unité de durée). Partout où l'on trouve des entiers positifs ou négatifs on pourra trouver aussi une expression arithmétique dont le résultat est un tel entier. Les entiers et les expressions arithmétiques ne donnent aucune indication sur la "nature" des systèmes décrits, ils indiquent simplement des propriétés géométriques du système : structures répétitives, parallélisme.

EXEMPLE : A(-13 : 18) constante vectorielle de dimension 32.

Identificateurs :

<IDENTIFIEUR> ::= <LETTRE>|<IDENTIFIEUR><LETTRE>|<IDENTIFIEUR><CHIFFRE>

Comme dans tous les langages les identificateurs sont formés d'un ensemble de lettres et de chiffres commençant par une lettre. Ils servent à repérer les variables de type Registre, Signal, Impulsion, Unité ou Etat ou les Etiquettes ou les variables arithmétiques.

Variables :

Les variables sont repérées par des identifieurs. Elles peuvent avoir 3 formes syntaxiques :

Signal-unité c'est une variable de type signal que nous verrons plus tard (Chap. IIIB), variable simple et variable-indicée.

<VARIABLE-SIMPLE> ::= <IDENTIFIEUR>
<VARIABLE-INDICEE> ::= <IDENTIFIEUR>(<LISTED'INDICES>).

Nous désignerons ainsi par indice soit un entier, soit une paire d'entiers, soit une expression dont les valeurs sont entières (<expression arithmétique> ou _ <expression booléenne>).

SEMANTIQUE :

Les variables désignent les constituants ou les fonctions élémentaires du système décrit (bascules, registres, connexions ...).

- Toute variable devra être déclarée.
- Une variable simple peut représenter aussi bien un scalaire qu'un tableau.
- Un indice est un entier, donc le résultat de _ (<Expression1>) est un entier, <Expression 1> doit pour cela être réductible à un vecteur A (unidimensionnelle) et _(A) représente le nombre dont l'écriture en binaire est la configuration des digits de A.

EXEMPLES : - Des variables peuvent être déclarées ,

A1, A2(0 : 7), A3(-1 : 4, 0 : 7, 1 : 36).

Cette déclaration définit respectivement A1, A2 et A3 comme un scalaire, un vecteur à 8 composantes et un tenseur de dimension (6,8,36).

Les occurences suivantes sont alors parfaitement correctes :

1) A1 (seule occurence possible).

2) A2 (3) qui désigne le 4ème digit de A2 en partant de la gauche.

3) A2 (E) où E est une expression arithmétique qui peut prendre une valeur entre 0 et 7.

4) A2 (B1) où B1 est par exemple un registre de 3 digits, dans ce cas si B1(0) = 1,B1(2) = 0,B1(3) = 1 la configuration des digits de B1 est (101) et A2(⊥ B1) est égal à A2(5).

5) A3(, ,35) : les zones vides signifient que l'on considère toutes les composantes correspondantes qui apparaissent dans les déclarations : ici A3(,,35)≡A3(-1:4,0:7,35) soit une sous-matrice de dimensions (6,8) du tableau A3.

6) A3 dans ce cas l'écriture est équivalente à celle de la déclaration de A3
A3≡A3(,,)≡A A3(-1:4,0:7,1:36) mais cette convention permet d'alléger considérablement l'écriture.

7) A3(-1:4,1:3,10:20) (ou encore : A3(,1:3,10:20)) ceci est un sous-tableau de dimensions (6,3,11) du tableau A3, dans ce cas aucune nouvelle déclaration du sous-tableau n'a besoin d'être faite, mais le programme d'analyse devra cependant vérifier que les nouvelles paires de bornes définissent bien un sous-tableau de A3 (donc que toutes ses dimensions sont < à celles correspondantes de A3).

8) A3 (E1,,⊥B2(3,0:5)) :

Dans ce cas lorsque E1 vaudra 3 et lorsque la 3ème ligne de la matrice B2 contiendra (011001), l'écriture précédente équivaudra à A3(3,0:7,25) qui est un sous-vecteur de dimension 8 du tableau A3.

9) etc....

SEMANTIQUE ET RECURSIVITE DU SYMBOLE $\perp$:

Nous avons déjà vu l'action de ce symbole sur une variable vectorielle. Plus généralement on peut l'appliquer à toute expression réductible à un vecteur

EXEMPLE : $\perp$(A B(1,,$\perp$C(0 : 3)).

On définit ainsi une sorte "d'adressage indirect" récursif à n'importe quel niveau. C'est pourquoi cet opérateur devra être employé avec précaution.

Il se traduit en Hardware par un décodeur et nous avons décidé qu'il générait une forme canonique (en l'absence d'autre définition de l'utilisateur).

EXEMPLE :

$\perp$B(0 1) $\Longrightarrow$ si B(0).B(1) alors 3 sinon si B(0).$\neg$B(1) alors 2 sinon si $\neg$B(0).B(1) alors 1 sinon si $\neg$B(0).$\neg$B(1) alors 0.

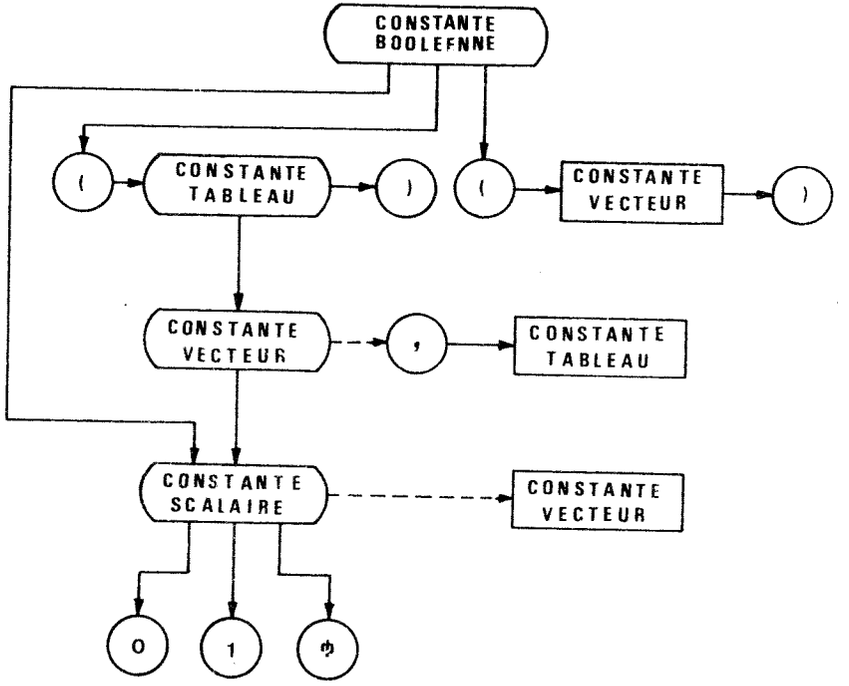
Ceci est assez coûteux et l'imbrication des symboles entraîne une croissance exponentielle du coût.

D'autres décodeurs pourront être définis et employés par le concepteur sous forme d'unité.

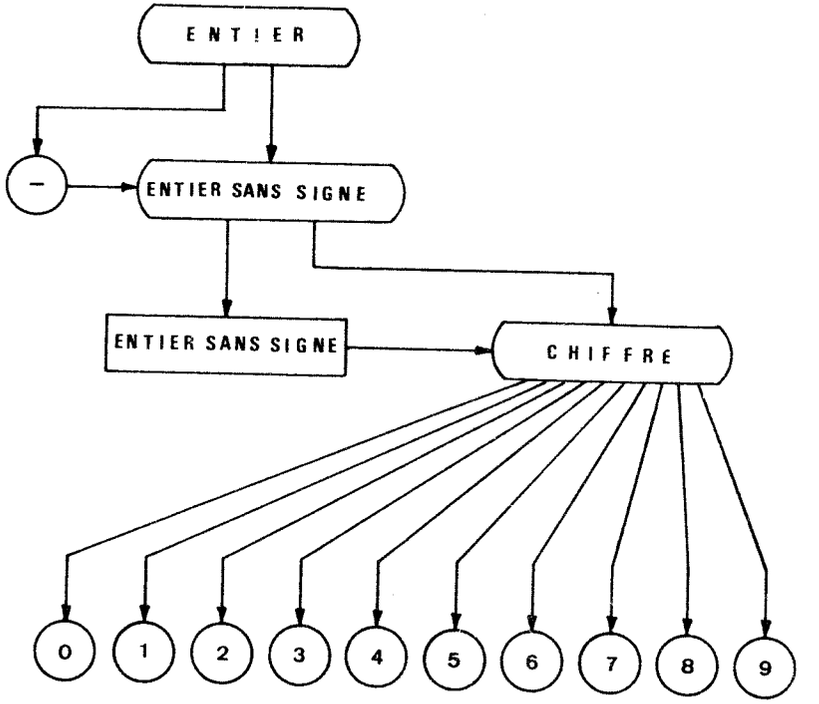
Nous préconisons donc l'emploi de pour générer de façon commode un indice de petite dimension ($\leq$ 8 ou $\leq$ 16).

La carte syntaxique de la variable apparaît au chapitre suivant, à la fin de B.

SYNTAXE DE LA CONSTANTE BOOLEENNE



SYNTAXE DE L'ENTIER



NOTIONS STRUCTURELLES

-:-:-:-:-:-

A - UNITE

Connexions et imbrications d'unités.

Contrairement à certains langages (comme celui de CHU) où une description de système logique se présente comme un tout, une description CASSANDRE est segmentable en autant de morceaux aussi petits ou grands que l'on veut. Ces parties sont appelées Unités et la segmentabilité résulte de 3 impératifs essentiels.

a) La longueur du texte traité doit être bornée pour, d'une part, qu'il puisse être rédigé sans effort par un individu, d'autre part, qu'il puisse être entré en machine et compilé.

Or il est facile et courant de trouver des systèmes qu'il est impossible de décrire d'un seul bloc et dont le volume déborderait toute mémoire de calculateur existant : il faut segmenter la description.

b) Les méthodes usuelles de réalisation d'un système logique imposent que ce système soit simple, si elles sont manuelles, pour rester dans le domaine de compréhension du concepteur, si elles sont automatiques, pour donner des résultats dans un temps raisonnable (les algorithmes employés sont souvent combinatoires et on sait difficilement réaliser, par exemple, des fonctions booléennes de plus de 10 variables).

Ceci impose un découpage du système étudié, en morceaux qui seront eux-mêmes redécoupés, au besoin, jusqu'au moment où chaque partie résultant du découpage sera d'une importance acceptable.

A tout découpage correspondra, en CASSANDRE, une structure d'unités imbriquées.

c) La technologie de réalisation des systèmes logiques, à son tour, impose une partition de celui-ci en sous-ensembles de hiérarchie variable suivant la complexité du matériel qui les réalisera. Ces sous-ensembles peuvent s'appeler "armoires", "tiroirs","plaquettes", "pastilles", "portes-logiques" où "éléments-discrets". Chacun peut être désigné en CASSANDRE par une unité. Leurs interconnexions doivent s'exprimer totalement et simplement, encore qu'on ne résolve en rien, ce faisant, les difficiles problèmes d'implantation posés par les contraintes technologiques qui s'appliquent à ces connexions. Cependant une telle description constitue un bon point de départ pour tout traitement automatique susceptible d'aider à la résolution de ces problèmes.

- L'étude à montré qu'une notion unique, celle d'Unité pouvait vérifier les trois impératifs précédents. Il n'y a pas, en effet, de différence sensible de description entre, par exemple, le bloc-diagramme de l'unité centrale d'un calculateur (figure 1 suivante) et le dessin d'un circuit de transistors MOS(figure 2).

Dans le premier cas apparaît un ensemble de boîtes (Registres ; décodeurs ; unités arithmétiques, etc...) interconnectés par un ensemble de fils ou de câbles de plusieurs fils (8 si on travaille par octets).

Dans le deuxième cas un ensemble de grilles(tachetées) et de diffusions (idem) peuvent être connectées par des métallisations (hachurées).

Dans le 1er cas on ne se préoccupe pas de la façon dont un registre est réalisé (circuit RS,D ou JK), ni des opérations de l'U.A.L, ces boîtes sont seulement accessibles par les bornes d'entrée et de sortie.

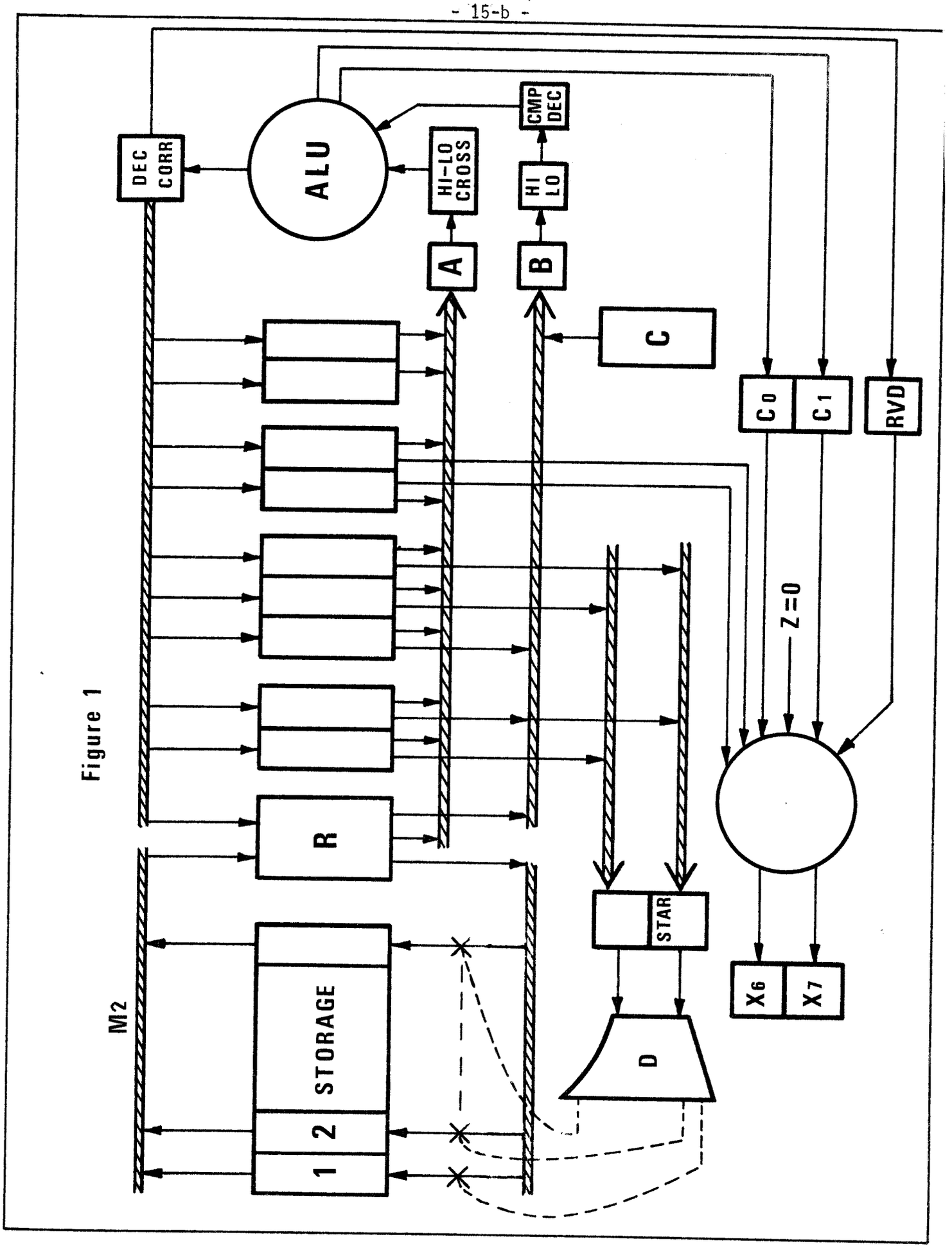

DEC CORR
ALU
HI-LO CROSS
CMP DEC
HI LO
A
B
C
C0
C1
RVD
Z=0
R
STAR
D
X6
X7
M2
STORAGE
1
2

Figure 1

Dans le second cas, on ne vas pas agir dans la surface d'une grille où d'une diffusion ; seules des prises de contact (carrés noirs sur figure 2) permettent aux métallisations d'atteindre ces éléments.

Vue de l'extérieur une unité apparaîtra comme une boîte noire et sera caractérisée entièrement par un nom, le nombre et l'ordre de ses entrées et de ses sorties. La forme la plus concise est celle d'un en-tête de procédure.

EXEMPLE : ADD(8,8 : 8,1) désignera une boîte ADD qui a 2 groupes de 8 entrées, un groupe de 8 sorties, plus une sortie isolée.

Nous verrons après (chapisuivant) comment il est facile de décrire l'interconnexion de telles unités à l'intérieur d'une plus grande qui les contient.

Nous avons vu l'unité de l'extérieur, nous allons voir comment la décrire, maintenant, de l'intérieur.

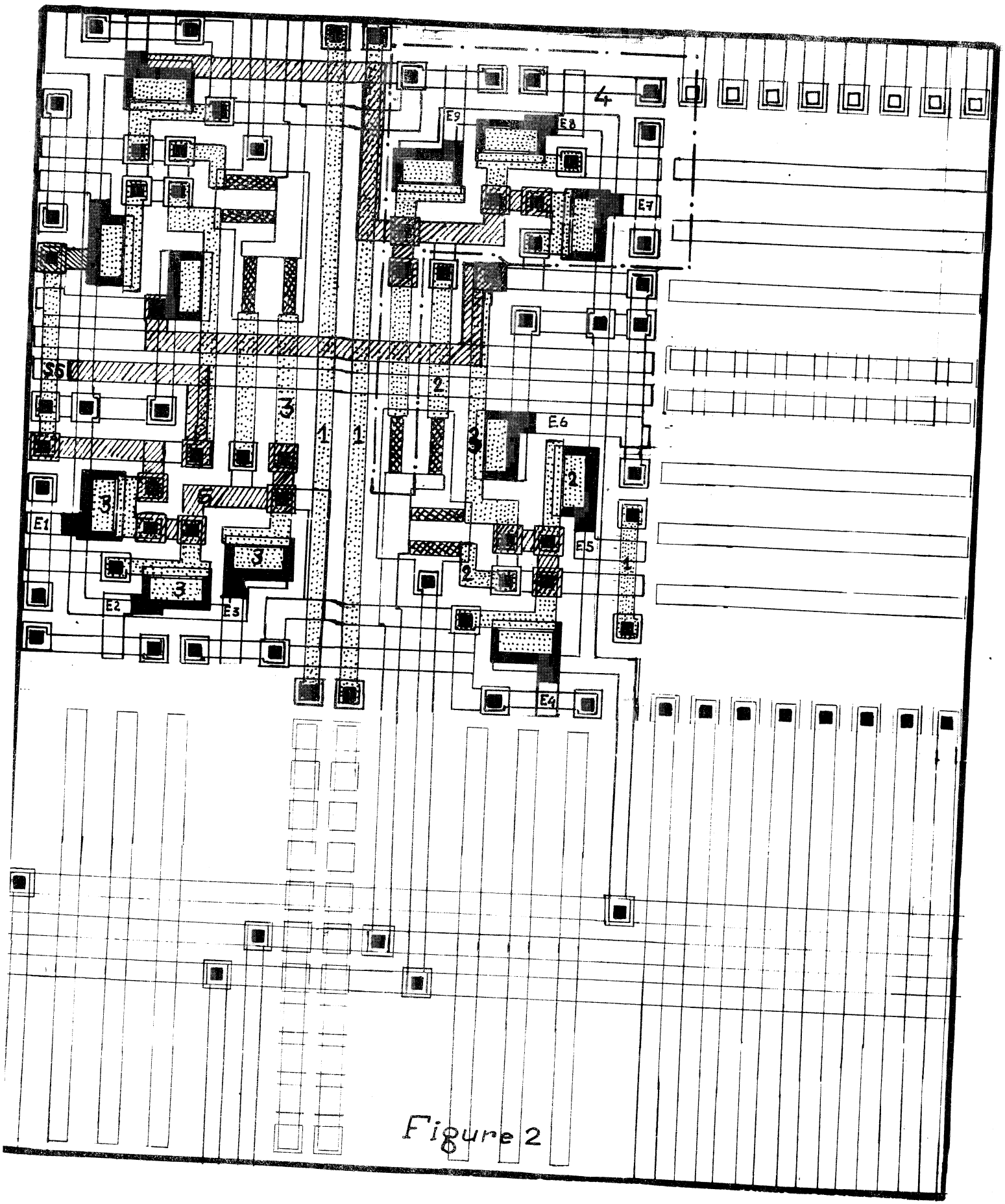
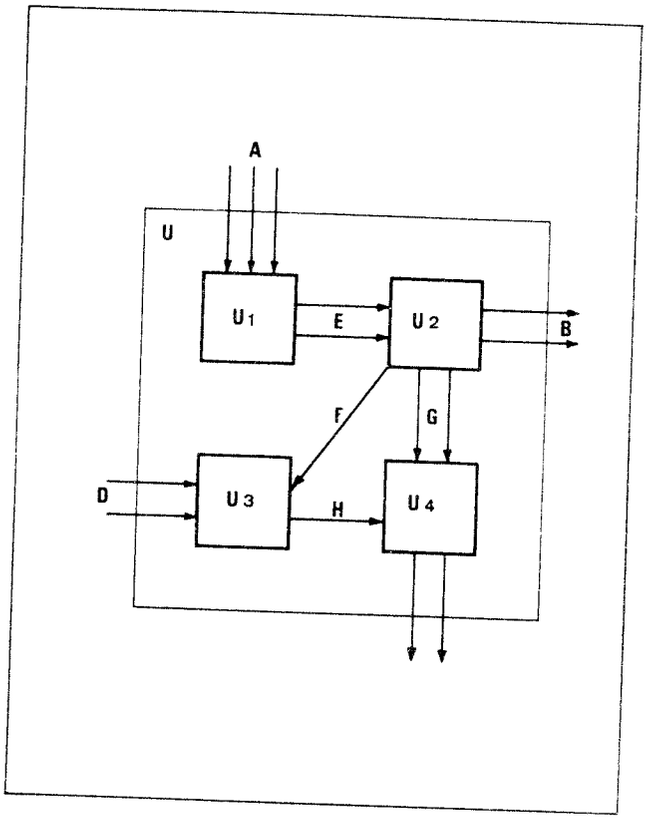


Figure 2

B - MATERIEL DECRIT

Si une unité a été réalisée sa description se bornera à celle d'un réseau d'autres unités interconnectées qui la constituent.

Dans la figure ci-contre l'unité U contient les 4 unités U_1, U_2, U_3, U_4 et les fils A,B,C,D,E,F,G,H qui sont des entrées, des sorties et des connexions.



Tout ce matériel sera décrit par les déclarations. L'unité commence par le symbole Unité suivi de son nom, suivi entre parenthèses de la liste de ses entrées et sorties avec leurs dimensions.

EXEMPLE : Unité U(A(1:3), D(1:2) ; B(1:2), C(1:2)). Les autres fils qui servent aux connexions sont de même nature que les fils d'entrée-sortie et on les déclare : signal E(1:2), F, G(1:2), H ; les unités employées sont des boîtes noires dans U, leur description éventuelle est externe à U ; on les déclare :

Externe U_1(3 ; 2), U_2(2 ; 1, 2), U_3(2 , 1 ; 1), U_4(1 , 2 ; 2) ;
Suivant le formalisme établi au paragraphe précédent.

La connexion de l'unité U_1 dans U s'exprime simplement en écrivant : U_1(A ; E) où l'on remplace les entrées-sorties de U_1 par les signaux de U qui leur sont connectés.

Il résulte de cette convention les 3 propriétés suivantes :

1) Dans tout système connexe d'unités il existe une unité qui contient toutes les autres.

2) Toutes les unités déclarées dans une même autre sont au même niveau. Elles sont au niveau immédiatement inférieur à celui de l'unité qui les contient.

3) La relation d'inclusion d'une unité dans une autre définit un arbre où chaque sommet correspond à une unité. Un programme établi par M. MILESI examine si cette 3ème propriété est bien vérifiée en calculant la fermeture transitive de la matrice des imbrications d'unités établies en examinant les déclarations d'externe. Il est évident que, pour une unité donnée, les faits de contenir une autre unité ou d'y être contenue sont incompatibles.

Mais le but du langage n'est pas de décrire uniquement des systèmes logiques déjà réalisés, par leur réseau, mais aussi des systèmes non encore réalisés, par leurs fonctions. En plus d'un certain nombre d'autres unités, une unité décrite pourra faire appel à des variables de 5 natures :

1) Eléments de mémorisation (Registres, Mémoires, Bascules...)
2) Eléments booléens de connexion (fils, bornes, circuits combinatoires...)
3) Impulsions de synchronisation (horloges, signaux dérivés...)
4) Variables d'état d'automates
5) Variables arithmétiques.

Pour la précision des définitions il est important d'insister sur les deux comparaisons suivantes :

a) Dualité entre les variables des catégories 1 et 2 appelées respectivement Registre et Signal.

La valeur d'un signal est indépendante des variables de type impulsion.

La valeur d'un registre ne change qu'en présence d'une impulsion.

Pour la concision du texte la variable signal qui correspond au fil de sortie d'une variable registre, est repérée par le même nom que ce dernier. L'utilisation qui en est faite suffit à les distinguer.

b) Dualité entre les variables des catégories 2 et 3 appelées respectivement signal et impulsion.

La valeur d'une variable signal est fixe pendant une certaine durée : c'est un niveau.

La valeur d'une variable impulsion sera supposée instantanée et infiniment courte (hypothèse idéale).

Un signal est une fonction booléenne d'un certain nombre de niveaux de sorties de variables de type registre.

DECLARATIONS ET SPECIFICATIONS

Le matériel employé dans une unité doit être déclaré au début de la description de cette unité, juste après l'en-tête.

Nous avons déjà vu comment les unités employées sont déclarées externes.

Les variables de type 1,2,3 et 4 sont également déclarées, celles de type 5, arithmétiques, ne définissent aucun élément constitutif de l'unité, elles sont identifiées par l'emplacement de leur occurence dans le texte et ne nécessitent donc aucune déclaration.

Dans les déclarations, les dimensions sont représentées par des paires de bornes. L'indice relatif à une direction pourra parcourir tout le champ situé entre les 2 bornes, c'est un entier positif ou négatif.

Il faudra vérifier à l'occurence d'une variable qui contient des indices, que ceux-ci se trouvent bien dans les champs permis.

a) Déclaration des variables de type 1 -

Elle est composée du symbole Registre suivi par la liste des identifieurs qui repèrent tous les éléments de mémorisation, avec leurs dimensions et leurs indices.

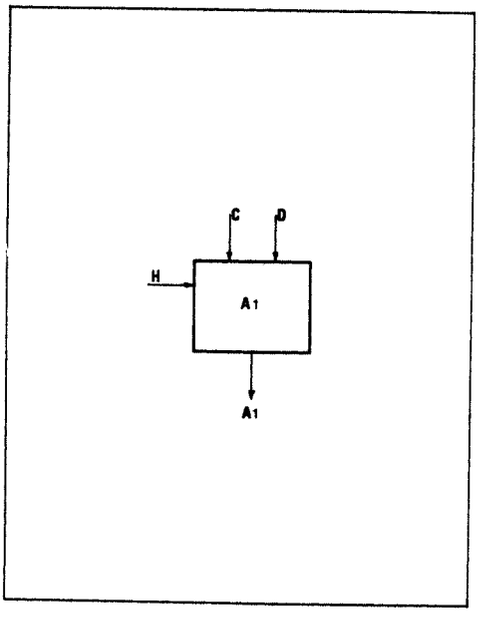
EXEMPLE : Registre A_1,A_2(1:36) , MEM (1:36,0:63,0:63) ;

Ceci établit l'existence d'une bascule A_1, d'un registre A_2 de 36 bits et d'une mémoire MEM à 3 dimensions, ayant 64x64 mots de 36 bits.

Cette déclaration n'indique aucune réalisation technologique de l'élément de mémorisation déclaré qui peut être formé de tores de ferrites aussi bien que de flip-flops RS ou D, eux-mêmes réalisés avec des portes, elles-mêmes réalisées avec des transistors M.O.S.

C'est une boîte noire, comme une unité vue de l'extérieur, et si on décrivait quelque part leur réalisation, la déclaration registre disparaîtrait au profit d'une externe.

Nous admettrons simplement que le module-bascule à 3 entrées : horloge H, donnée D et condition C (voir LIDDELL|56|).



L'horloge est celle qui conditionne les affectations de la donnée à la bascule. La condition peut être inemployée. Si, toute-fois, on connaît la fonction-bascule qui réalisera le registre déclaré, on peut faire suivre le symbole Registre d'un numéro qui spécifie cette fonction dans une classification |9|.

Certaines indications utiles peuvent être rajoutées à une telle déclaration par une spécification dans laquelle dimensions et indices n'ont plus à apparaître.

Ex. Mémoire MEM ;

b) Déclaration des variables de type 2-

Comme ci-dessus elle est constituée par le symbole signal suivi de la liste des identifieurs de signaux avec leurs dimensions.

On peut désigner aussi bien un seul fil qu'un paquet de fils que l'on peut organiser en tableau à nombre arbitraire de dimensions.

EXEMPLE : Signal $C_1(1:36)$, $C_2(-11:6)$;

Certaines spécifications peuvent être utiles.

EXEMPLE : Bus C_2 ;

c) Déclaration des variables de type 3 -

Elle commence par le symbole Impulsion et se trouve pour le reste analogue à la précédente.

EXEMPLE : Impulsion H, D(1:3);

Certaines spécifications peuvent encore apparaître.

EXEMPLE : Horloge H ;

Il est raisonnable en effet de donner un traitement spécial à ces variables qui jouent en synthèse le rôle particulier de fournisseur d'énergie et d'élément de synchronisation.

Il peut arriver que l'horloge soit d'ailleurs une entrée particulière des bascules et n'intervienne pas dans le réseau combinatoire.

d) Déclaration des variables de type 4.

Elle commence par le symbole Etat, suivi d'une liste d'identificateurs auxquels, en général, ne seront associées aucunes dimensions.

En effet, ces variables peuvent être définies avant tout codage des valeurs d'états de l'unité, et comme elles servent à mémoriser ces valeurs, on ne peut pas leur fixer une dimension.

Dans le cours du traitement elles seront manipulées comme les variables de type 1 (éléments de mémorisation).

Il est à noter que la variable interne de l'unité est de ce type, mais elle sera toujours implicite en CASSANDRE et ses affectations seront notées par des ordres allera.

Les valeurs que peuvent prendre les variables d'états et la variable interne de l'unité seront repérées par des étiquettes, c'est à dire des identifieurs non déclarés. Coder les états consiste à remplacer toutes les étiquettes par un monôme booléen faisant intervenir les composantes alors définies des variables d'état.

e) Variables arithmétiques.

Nous verrons leur utilisation dans le chapitre [IV].

RESUME

La définition d'unité permet de définir entièrement un organe logique. Le nom d'unité sert à la repérer ; la liste d'entrée définit les "bornes d'entrée de la boîte noire".

Les identificateurs de la liste d'entrée sont équivalents à des déclarations de SIGNAL ; de même pour la liste de sortie qui définit "les bornes de sortie de la boîte noire".

Les instructions de connexion et d'affectation définissent enfin le fonctionnement interne complet de l'unité (Chapitres suivants).

REMARQUES

Rôle particulier des entrées-sorties

On a vu que la liste d'entrée et la liste de sortie définissent des variables du type SIGNAL. Ces variables pourront donc se trouver dans les instructions. Toutefois, en raison de leur rôle particulier de liens avec l'extérieur, la règle suivante est à respecter :

- une variable appartenant à la liste d'entrée ne doit jamais se trouver à gauche d'un signe d'affectation (ou de liaison).

Des variables de type impulsion peuvent aussi, quelquefois, se trouver en entrée de la définition d'Unité.

A l'inverse des signaux qui s'y trouvent aussi elles seront redéclarées impulsion dans la description de l'unité qui suit.

Un certain nombre d'entrées-sorties sont implicites dans les Unités. Ce sont :

1) Les alimentations
2) Les fils de sortie d'un registre d'état quand l'état d'une unité est mémorisé dans une unité plus grande qui la contient.

Portée des identificateurs et des étiquettes.

La portée de tous les identificateurs déclarés dans la définition d'une unité est valable dans l'unité et ignorée au-delà.

Il en est de même pour les étiquettes ; en effet, la variable interne bien qu'implicite est considérée comme purement locale.

Le cas des identificateurs des listes d'entrée-sortie est un peu particulier ; disons que le nom de ces identificateurs est également ignoré de l'extérieur mais que leur existence est connue en dehors de l'unité.

Propriétés particulières.

- Une unité peut ne pas avoir de liste d'entrée, ou ne pas avoir de liste de sortie. Mais en aucun cas liste d'entrée et liste de sortie ne peuvent être vides simultanément.

- Une unité peut ne comporter aucune déclaration ; ce sera le cas par exemple de la définition d'un composant réduit à un circuit combinatoire.

- Une unité peut aussi ne comporter ni instructions de connexion ni instructions (voir exemples suivants).

EXEMPLE :

Nous venons de voir le début de la description des Unités en CASSANDRE, nous avons également décrit de façon sommaire comment ces unités peuvent être interconnectées, nous pouvons, sans plus attendre, illustrer ceci sur un exemple inspiré de la figure 2 (page 1:b).

La définition de l'Unité a pour but ambitieux de délimiter un morceau d'un système logique de complexité arbitraire. Les chapitres suivants donneront peut-être une idée de la complexité qui peut être maitrisée par la description ; ici nous allons utiliser les unités les plus simples qu'il soit possible d'imaginer.

L'élément vide exclu, l'élément qui est minimal est le bout de fil conducteur orienté (ou le morceau de diffusion par exemple).


```
Unité FIL (ENTREE ; SORTIE) ;
        SORTIE := ENTREE
```

(Boîte de type 1, sur la figure 2)

A peine plus compliquée serait une GATE M.O.S. formée d'un transistor de charge ou de commande et de sa diffusion de sortie associé (boîte de type 2 sur la figure 2).

```
Unité MOS CHARGE (ALIM ; OUT) ;
Unité MOS COMMANDE (E1 ; S1) ;
```

} exemples d'unités n'ayant ni déclaration, ni connexion, ni instruction.

Une gate formée d'un MOST de commande et de son MOST de charge associé forme alors un inverseur (boîte de type 3, figure 2).

```
Unité INVERSEUR 1 (E ; S) ;
        S := ¬ E ;
```

Si, comme c'est souvent le cas, figure 2, on peut prendre la sortie de l'inverseur en plusieurs point :

```
Unité INVERSEUR 2 (E ; S (1:2)) ;
      S(1) := ¬E ;
      S(2) :=  S(1) ;
```

Une porte N1 à 3 entrées (boîte 4, figure 2) sera maintenant décrite ;

```
Unité N 13 (ALIM, ENT (1:3) ; SOR (1:2)) ;
Externe MOSCOMMANDE (1 ; 1) , MOSCHARGE (1 ; 1) ;
Signal SIG ;
       SOR(1) := ¬ ENT(1) Λ ¬ ENT(2) Λ ¬ ENT(3) ;
       SOR(2) :=  SOR(1) ;
       SIG := SOR(2) ;
  MOS CHARGE (ALIM ; SIG) ;
Pour I égal 1 pas
     1 à 3 faire MOS COMMANDE [I] (ENT(I) ; SIG) ;
```

Le signal SIG définit la 1ère métallisation (N° 5)

SOR(1) , SOR(2) et SIG se trouvant sur une même équipotentielle qui vaut la fonction N1 des entrées ENT (1:3).

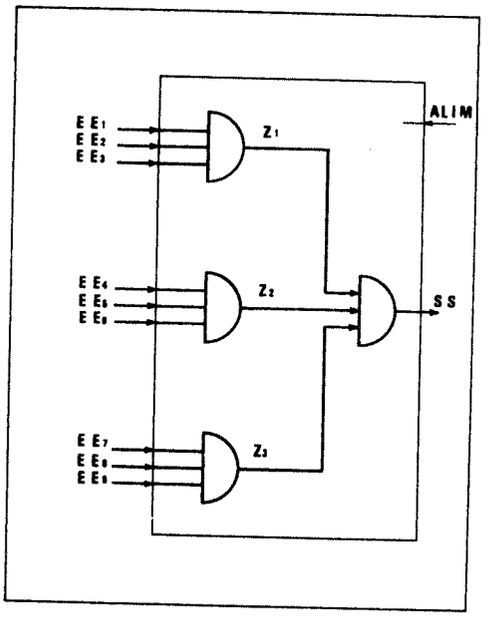
On peut alors décrire n'importe quel circuit sur la figure 2, chaque nouveau signal définira une métallisation supplémentaire.

Par exemple le schéma sera décrit en CASSANDRE,

Unité CIRCUIT (EE(1:9) ; SS ; EN-TETE

Signal Z_1,Z_2,Z_3,ALIM ; DECLARATION
Externe N13 (1,3;2) ; SPECIFICATION

N13 1 (ALIM,EE(1:3) ; Z_1&Z_1); PARTIE
N13 2 (ALIM,EE(4:6) ; Z_2&Z_2); FONCTION-
N13 4 (ALIM,EE(7,9) ; Z_3&Z_3); NELLE
N13 4 (ALIM,Z_1&Z_2&Z_3; SS&SS);



CONCLUSION

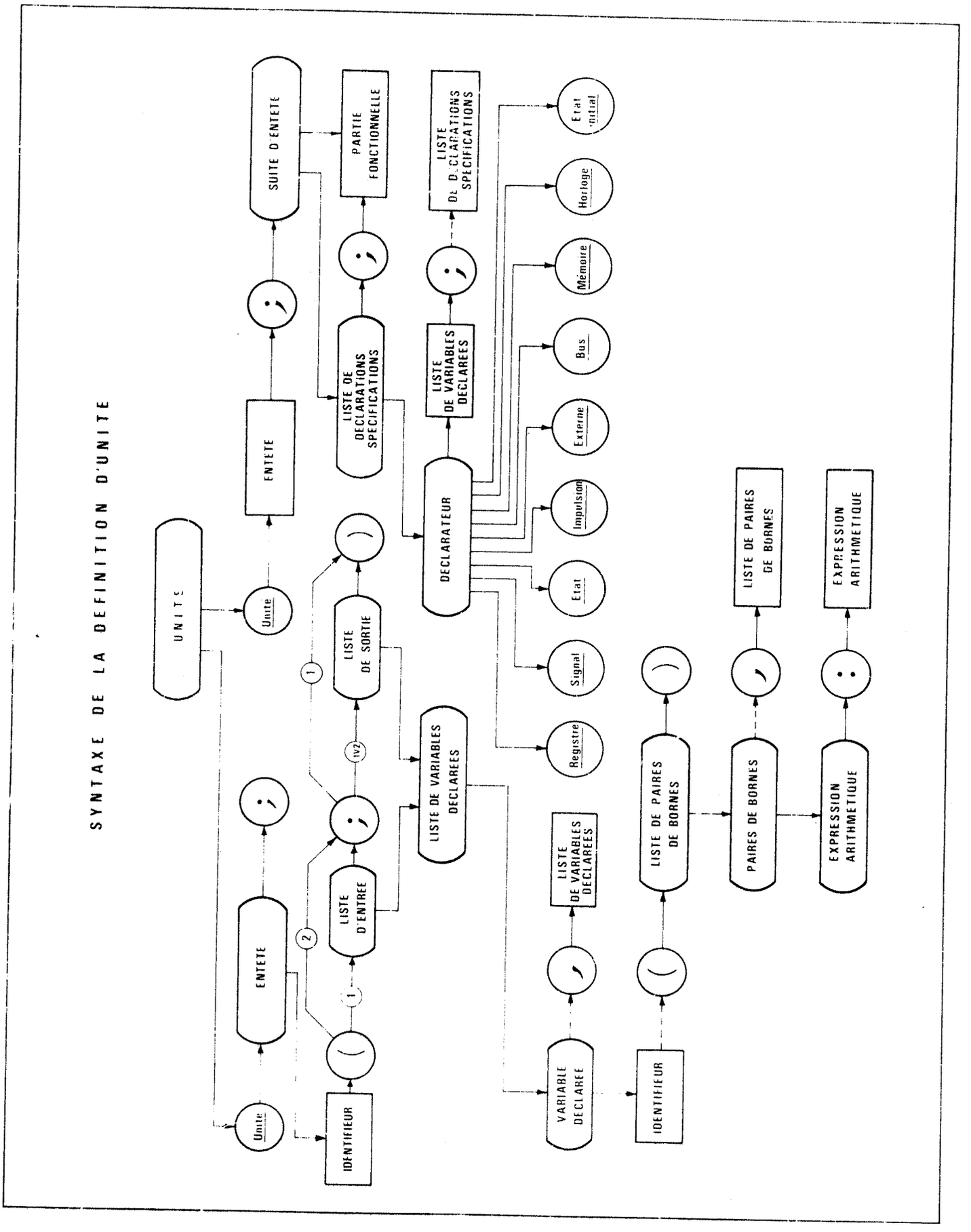
1) Nous pouvons voir sur la figure 2 une réalisation réelle de l'unité CIRCUIT ci-dessus où les métallisations créées par les déclarations de signaux EE(1:9) SS, Z_1, Z_2, Z_3 et ALIM sont dessinées hachurées.

S'il est impossible d'établir une correspondance précise entre la description ci-dessus et le dessin de la figure 2, néanmoins la définition en CASSANDRE du circuit semble une des plus concises et des plus adaptées pour entrer le problème en machine et le fournir en donnée à des programmes de dessin de masque ou d'implantation.

2) Nous avons déjà pu voir sur les exemples précédents, quoique très simples, qu'une description d'unité comporte un ENTETE, suivi d'une liste de déclarations et spécifications suivie d'une partie fonctionnelle ; c'est-à-dire partie fonctionnelle que nous allons consacrer les chapitres suivants. Nous en avons terminé avec la partie statique et descriptive de réseau constituée par la juxtaposition et l'imbrication des unités.

3) Une description en CASSANDRE est une liste de définition d'unités. L'une d'entre elles contient toutes les autres.

SYNTAXE DE LA DEFINITION D'UNITE



SURVARIABLES, SOUSVARIABLES, VARIABLES GENERALISEES

Nous venons de voir que des variables déclarées définissent le matériel employé, en particulier les registres propres à chaque unité.

Nous allons maintenant grâce à l'ordre d'équivalence donner la possibilité de désigner par plusieurs identificateurs la même variable ou partie de variable. Nous allons aussi permettre de la restructurer avec les opérateurs de concaténation et de transposition.

DEFINITION DE LA COMPTABILITE DE 2 VARIABLES.

Soient 2 variables A et B de type tableau. Lorsqu'on les rencontre en un point quelconque de la description en CASSANDRE leurs dimensions ne sont pas forcément indiquées. Il faut se reporter à leurs déclarations.

Supposons que l'on trouve en déclaration :

$$A(a_1 : b_1 , a_2 : b_2 , ... , a_i : b_i)$$

$$\text{et } B(c_1 : d_1 , c_2 : d_2 , ... , c_j : d_j)$$

A et B seront compatibles si l'on a :

$$(i=j) \wedge ((b_1-a_1) = (d_1-c_1)) \wedge ((b_2-a_2)) = (d_2-c_2) \wedge ..... \wedge ((b_i-a_i) = (d_j-c_j))$$

- La compatibilité entre 2 variables peut s'étendre évidemment à la compatibilité entre expressions.

a) <u>Equivalence de variables</u>.

Deux variables déclarées de <u>même type</u> peuvent être désignées comme <u>équivalentes</u> si elles se trouvent à la suite des déclarations repliées par le symbole ≡.

<u>EXEMPLE 1</u> : Registre A(1 : 36), ADDR (0 : 17) ;
A(19 : 36) ≡ ADDR ;

Le bilan du matériel effectivement employé devra être fait modulo les ordres d'équivalence.

Il est clair dans le cas précédent, qu'il y a non pas 2 mais un seul registre A, dont il est commode de baptiser les 18 derniers bits en en faisant un sous-registre ADDER.

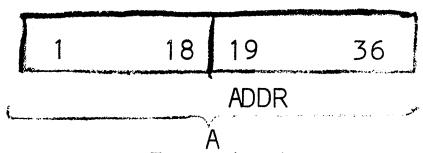
Notons qu'appliqué à des signaux, l'ordre ≡ a le même effet que := (l'ordre de connexion, voir(58b) puisque la définition d'un signal nouveau n'augmente pas le matériel employé.

EXEMPLE 2 : Signal B(0:11) , C(1:32 ;

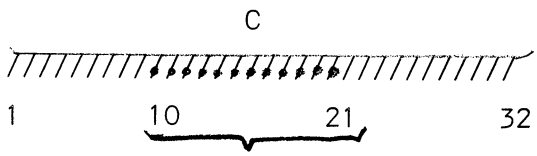
B≡C (10:21) ou B := C (10:21)

Dans le premier cas, il y a un seul signal ;

Dans le deuxième cas, deux connectés par des fils que l'on peut supposer arbitrairement courts, donc confondus.



Exemple 1



Exemple 2

b) Concaténation :

Cet opérateur & , permet d'accoler suivant la première direction, deux variables ayant le même nombre de dimensions et des dimensions compatibles dans toutes les directions autres que la première.

EXEMPLE 3 A(2 : 7, 0 : 7, 1 : 36)

B(1 : 10, 1 : 8, -10 : 25)

A & B est alors défini, c'est un tableau de dimensions (18, 8, 36), obtenu en juxtaposant A et B parallèlement à la première direction.

REMARQUE : Un cas particulier de la concaténation précédente, est celle de deux vecteurs ou d'un scalaire et d'un vecteur :

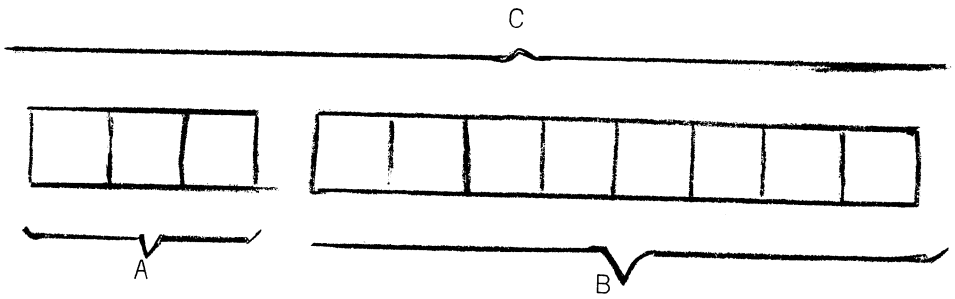
EXEMPLE 4 :

A(0 : 2) & B(1 : 8)≡C(0 : 10), A(1) & B(0 : 6)≡C(0 : 7)

La concaténation, employée dans une équivalence, permet de définir des survariables.

Dans l'exemple 4 si l'on a une déclaration, Registre A (0 : 2), B(1 : 8), C(0 : 10) ;

Le surregistre C, désigne l'ensemble des registre A et B mis bout à bout :



c) Transposition :

L'opération de transposition ∿, permet de restructurer les variables ayant au moins deux directions, en permuttant deux directions de cette variable.

Ceci sera tout à fait clair sur un exemple.

EXEMPLE 5 :

B(0 : 4, 1 : 3, 0 : 7, 1 : 36)

2 : 4 ∿ B ≡ B (0 : 4, 1 : 36, 0 : 7, 1 : 3)

On permutte les directions indiquées par les 2 entiers qui précèdent .

Quand il y a seulement deux directions, on peut omettre la paire d'entiers.

La transposition permet d'appliquer la concaténation à toutes les directions d'une variable.

EXEMPLE 6 :

B(0 : 4, 1 : 3, 0 : 7) , A(1 : 6, 1 : 3, 1 : 5) ;

C(0 : 15, 0 : 2, 0 : 4)

C ≡ A & 1 : 3 ∿ B

Il suffit de ramener en première place la direction choisie. Les deux opérations précédentes permettent d'élargir la notion de variable.

d) Variable généralisée :

Nous appellerons ainsi le résultat d'un nombre quelconque de transpositions et de concaténations, sur un nombre quelconque de variables. La transposition aura une priorité supérieure à celle de la concaténation.

EXEMPLE 7 : Registre A(1 : 3, 1 : 8) , B(1 : 9, 0 : 3, 1 : 15) ,
C(0 : 7, 1 : 16) , D(1 : 32, 0 : 2) ;

∿ A & B(2 : 9, 0 : 2, 7) & ∿ C(, 1 : 3) & D

définit une nouvelle variable de dimensions

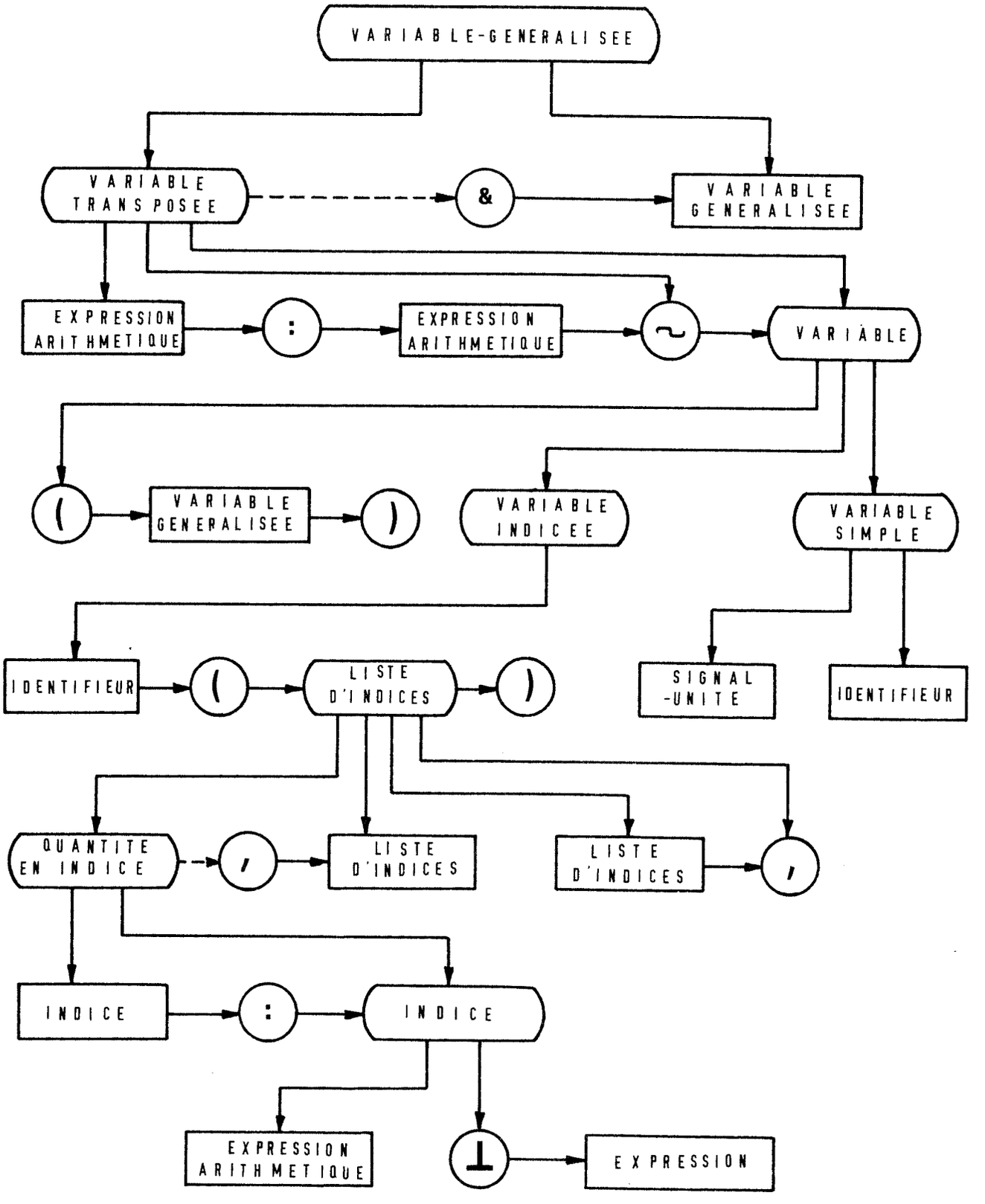
8 + 8 + 16 + 32 = 64 et 3.

On pourra désormais la désigner plus simplement en lui donnant un nom :

EXEMPLE : Registre E (1 : 64, 0 : 2) ;

E ≡ ∿ A & B(2 : 9, 0 : 2, 7) & ∿ C(1 : 3) & D ;

SYNTAXE DE LA VARIABLE-GENERALISEE



NOTIONS FONCTIONNELLES

A - NIVEAU BOOLEEN (Opérateurs, Expressions).

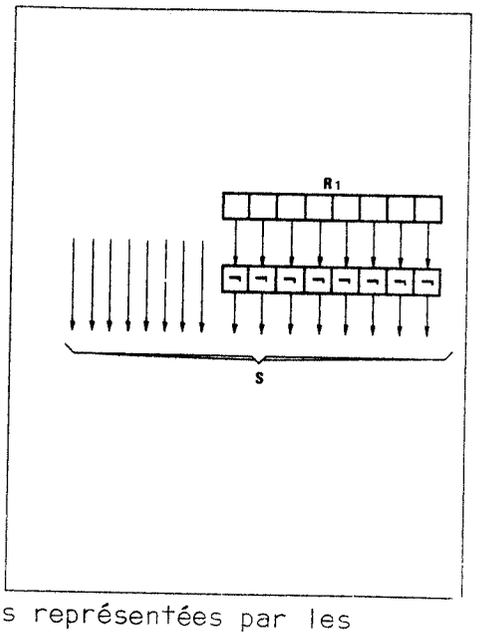
a) Opérateurs booléens :

Toutes les fonctions booléennes de deux variables sont représentées en CASSANDRE par un symbole de base.

0 et 1 représentent donc aussi les fonctions booléennes de 0 variables.

Les identifieurs déclarés et l'opérateur non ( ) définissent toutes les fonctions d'une variable.

EXEMPLE : Registre R1 (1 : 8) ;
Signal S (1 : 16) ;
S (9 : 16) := ¬R1 ;
Ceci défini le schéma :



L'opérateur ¬, habituellement défini sur des quantités booléennes scalaires, est en CASSANDRE, comme dans le langage d'Iverson, appliqué à chaque composante d'un tableau booléen.

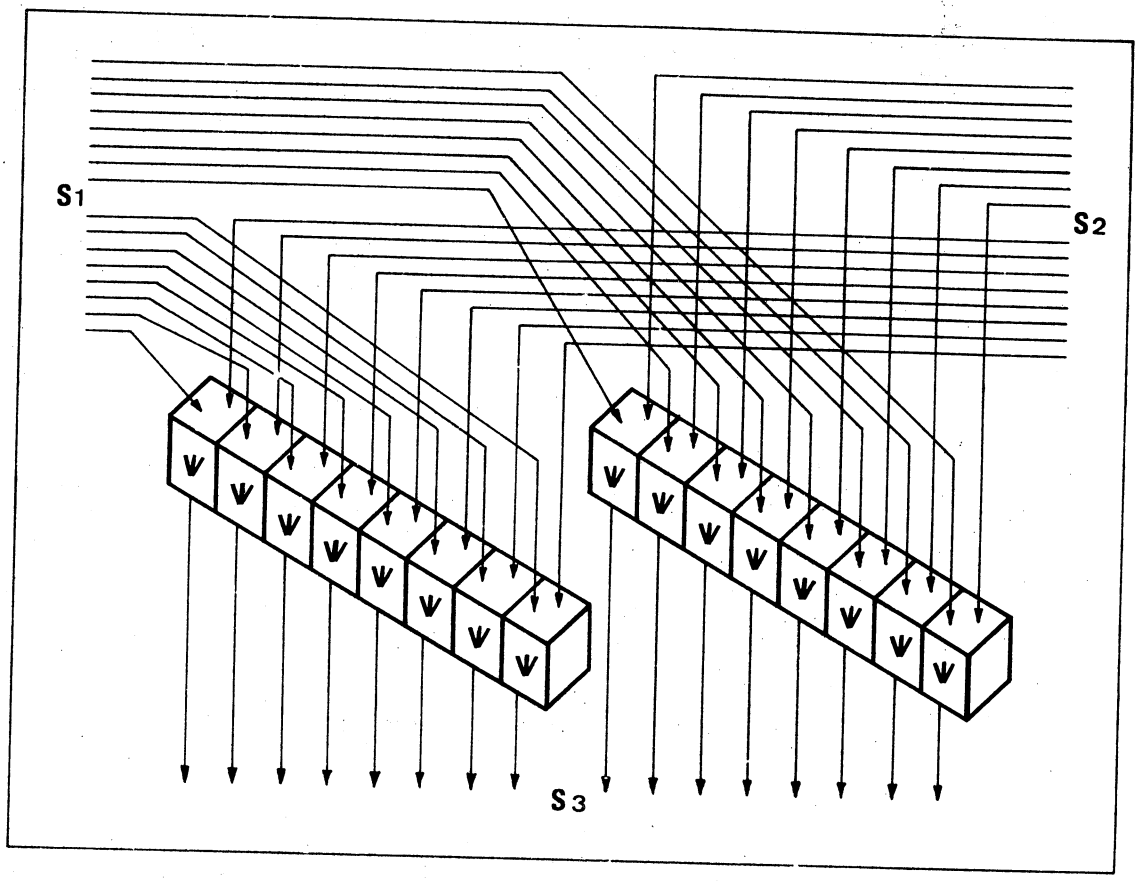
Les fonctions de deux variables sont toutes représentées par les six opérateurs :

$=$, $\neq$, $>$, $<$, $\geq$, $\leq$

et par les opérateurs et $\wedge$, ou $\wedge$, non et $\not\wedge$ et non ou (ou ni $\vee$).

EXEMPLE : Signal S1(1 : 2, 0 : 35), S2 (1 : 2, 1 : 2, 1 : 8), S3 (0 : 6, 1 : 18) ;
S3(1 : 2, 1 : 8) := S1 (,1 : 8) $\vee$ S2 ;

Ceci correspond au schéma suivant :



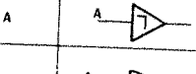
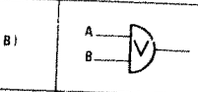
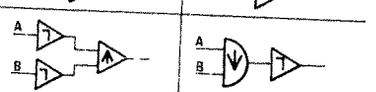
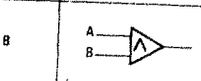
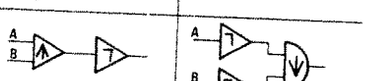
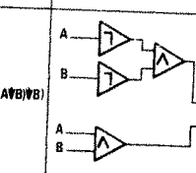
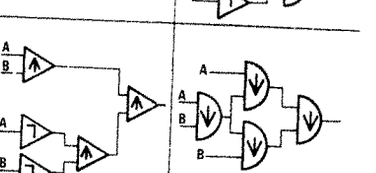
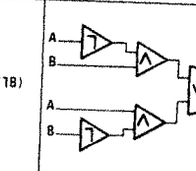
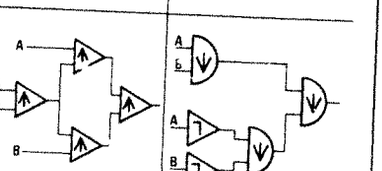
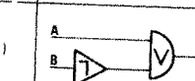
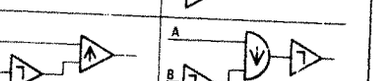
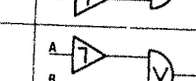
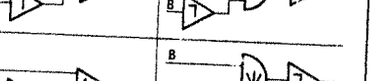
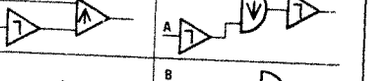
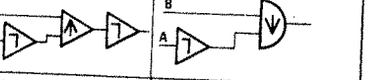
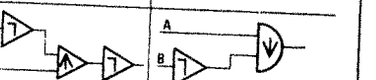
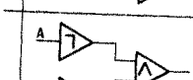
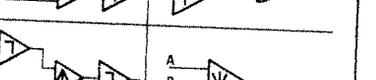
Pour les opérateurs dyadiques booléens l'opération qu'ils représentent s'applique composante à composante entre des tableaux de dimension quelconque.

Ceci entraîne l'obligation de vérifier pour chaque opérateur que les deux variables auxquelles il s'applique sont bien compatibles ; (Voir définition de la compatibilité au chapitre précédent).

Dans l'exemple ci-dessus, on voit que S1 et S3 ont été chacun réduits pour être compatibles entre eux et avec S2.

b) Expression des opérateurs booléens dans les 3 bases suivantes : ($\wedge$, $\vee$, $\neg$) ; ($\neg$, $\uparrow$) ; ($\neg$, $\downarrow$) .

Nous les numéroterons, I, II et III.

Operateur	Base I	Base II	Base III	Schema logique I	Schema logique II	Schema logique III
$\neg A$	$\neg A$	$\neg A$	$\neg A$			
$A \vee B$	$A \vee B$	$\neg A \uparrow \neg B$	$\neg(A \downarrow B)$			
$A \wedge B$	$A \wedge B$	$\neg(A \uparrow B)$	$\neg A \downarrow \neg B$			
$A = B$	$(\neg A \wedge \neg B) \vee (A \wedge B)$	$(A \uparrow B) \uparrow (\neg A \uparrow \neg B)$	$((A \downarrow B) \downarrow A) \downarrow ((A \downarrow B) \downarrow B)$			
$A \neq B$	$(\neg A \wedge B) \vee (A \wedge \neg B)$	$((A \uparrow B) \uparrow A) \uparrow ((A \uparrow B) \uparrow B)$	$(A \downarrow B) \downarrow (\neg A \downarrow \neg B)$			
$A \geq B$	$A \vee \neg B$	$\neg A \uparrow B$	$\neg(A \downarrow \neg B)$			
$A \leq B$	$\neg A \vee B$	$A \uparrow \neg B$	$\neg(\neg A \downarrow B)$			
$A > B$	$A \wedge \neg B$	$\neg(A \uparrow \neg B)$	$\neg A \downarrow B$			
$A < B$	$\neg A \wedge B$	$\neg(\neg A \uparrow B)$	$A \downarrow \neg B$			
$A \downarrow B$	$\neg A \wedge \neg B$	$\neg(\neg A \uparrow \neg B)$	$A \downarrow B$			
$A \uparrow B$	$\neg A \vee \neg B$	$A \uparrow B$	$\neg(\neg A \downarrow \neg B)$			

Notons que fournir un circuit réalisable en opérateurs Ni ou Nand à partir d'une description en CASSANDRE consiste simplement à exprimer les opérateurs du langage dans les bases ii ou iii. Un tel circuit ne sera certainement pas optimal, il faudrait ajouter à un processus de synthèse dont ce serait le dernier travail, des procédures de réduction et de simplification.

c) Opérateurs non-booléens :

Nous avons déjà vu au précédent chapitre les opérateurs & et ∿ qui permettent de définir des variables généralisées, nous trouvons également dans le langage les opérateurs de décalage, de rotation, de retard, de dérivation, et d'intégration.

c_1) L'opération de décalage est définie sur des vecteurs.

EXEMPLE 4 ↓ B (1 : 8) signifie le décalage à gauche de quatre positions de B. Les positions libérées à droite sont remplacées par des 0.

- 3 ↑ B signifie le décalage de trois positions à droite.

  4 ↓ B ≡ 5(5)&B(5)&B(7)&B(8)&0&0&0&0

- 3 ↓ B ≡ 0&0&0&B(1)&B(2)&B(3)&B(4)&B(5)

L'opérateur de rotation ↑ opère comme celui de décalage, mais les positions libérées à gauche ou à droite sont remplacées par les positions précédemment perdues.

  4 ↑ B ≡ B(5)&B(6)&B(7)&B(8)&B(1)&B(2)&B(3)&B(4)

- 3 ↑ B ≡ B(6)&B(7)&B(8)&B(1)&B(2)&B(3)&B(4)&B(5)

Ces deux opérations comme la réduction peuvent s'étendre à chaque dimension d'un tableau de dimensions quelconques.

Comme on peut le voir ces opérateurs non pas un rôle logique mais formel. ils n'introduisent pas de matériel supplémentaire, mais seulement des permuttations entre fils.

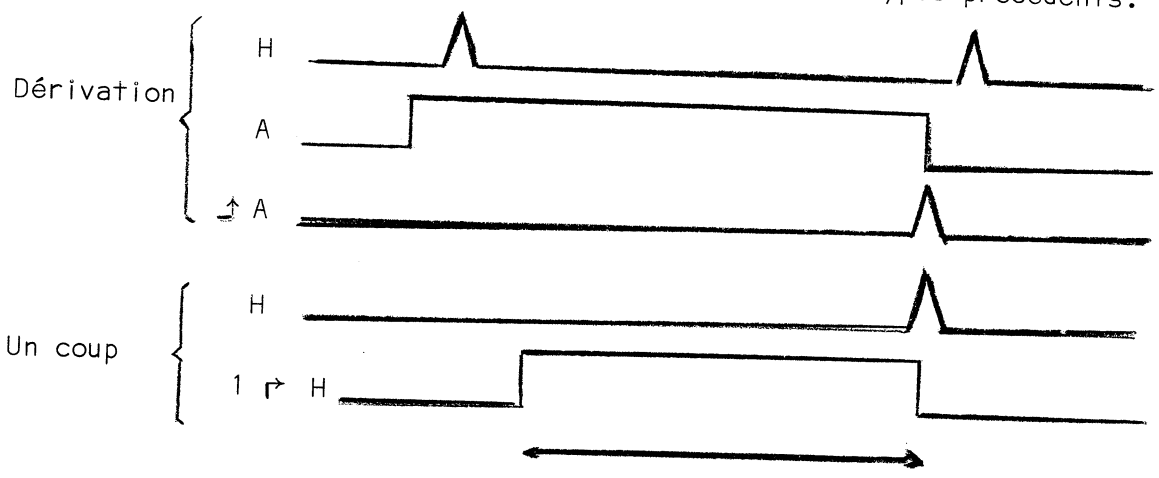
c_2) Les opérateurs ↑, ↱ et τ, définis avec Messieurs ANCEAU et LUSTMAN permettent de donner une description schématisée mais aussi approchée que l'on veut de la synchronisation des signaux.

L'opérateur de dérivation ↑ sur front de montée permet de passer d'un niveau de type signal à une impulsion.

La variable résultant de cette opération doit avoir été déclarée impulsion.

On veut dériver sur front de descente en complémentant le signal. L'opérateur "un coup" ↱ fait l'opération inverse, il doit s'appliquer à une expression de type impulsion et il génère un niveau donc de type signal. Il peut être précédé d'un entier qui indique la durée du niveau.

L'opérateur τ de retard associé à une durée exprimée par un nombre entier permet de retarder des variables des deux types précédents.



d) Opérateur condition, réduction, incrémentation.

d_1) L'opérateur condition.

L'opérateur condition est un opérateur booléen particulier pour lequel la règle de compatibilité précédemment énoncée ne s'applique pas.

EXEMPLE : Si A alors B (1 : 3, 1 : 4)

Si A est une variable booléenne scalaire l'expression précédente définit une variable E(1 : 3, 1 : 4) telle que par exemple :

E(2,3) = A $\wedge$ B (2,3)

REMARQUES :

1) On retrouve au niveau de chaque composante la signification de condition logique.

2) Appliqué à des tableaux quelconques l'opérateur condition permet de faire l'intersection de toutes les composantes du tableau avec une même variable booléenne scalaire.

3) Nous verrons dans le chapitre IV comment généraliser cette opération.

4) On peut appliquer n'importe qu'elle opération booléenne, entre un scalaire A et un tableau B de la façon suivante :

Soit B (1 : 3, 1 : 4) et soit $\varepsilon 1$ (1 : 3, 1 : 4) le tableau de mêmes dimensions que B dont toutes les composantes sont égales à 1.
Si A alors $\varepsilon 1$(1 : 3, 1 : 4) produit un tableau de même dimensions que B dont toutes les composantes sont égales à A, l'opérateur = peut alors s'appliquer composante à composante : (si A alors $\varepsilon 1$(1 : 3, 1 : 4)) = B

d_2) L'opération de réduction / est définie par Iverson pour des vecteurs.

Etant donné un opérateur dyadique booléen ⊙ et une variable vectorielle A(1 : 4, nous définissons la réduction par :

$$\odot / A \equiv A(1) \odot (A(2) \odot (A(3) \odot A(4)))$$

Le résultat de la réduction d'un vecteur par rapport à l'opérateur ⊙ est donc un scalaire.

Nous pouvons généraliser cette opération aux tableaus de dimension quelconque en l'appliquant à leur première direction.

EXEMPLE :

$$\odot / B(1 : 3,\ 1 : 7) \equiv C\ (1 : 3)$$

$$C(1) \equiv B(1,1)\odot(B(1,2)\odot(B(1,3)\odot(B(1,4)\odot B(1,5)\odot(B(1,6)\odot(B(1,7)))))).$$

$$C(2) \equiv B(2,1)\odot(B(2,2)\odot(B(2,3)\odot(B(2,4)\odot(B(2,5)\odot(B(2,6)\odot B(2,7)))))).$$

$$C(3) \equiv B(3,1)\odot(B(3,2)\odot(B(3,3)\odot(B(3,4)\odot(B(3,5)\odot(B(3,6)\odot B(3,7)))))).$$

L'opération de transposition permet également d'appliquer la réduction à n'importe qu'elle direction ramenée en première position.

EXEMPLE :

$$\odot / \sim B(1:3\ ,\ 1:7) \equiv D(1:7)$$

$$D(1) \equiv B(1,1) \odot (B(2,1) \odot B(3,1))$$

$$D(2) \equiv B(1,2) \odot (B(2,2) \odot B(3,2))$$

$$D(3) \equiv B(1,3) \odot (B(2,3) \odot B(3,3))$$

$$D(4) \equiv B(1,4) \odot (B(2,4) \odot B(3,4))$$

$$D(5) \equiv B(1,5) \odot (B(2,5) \odot B(3,5))$$

$$D(6) \equiv B(1,6) \odot (B(2,6) \odot B(3,6))$$

$$D(7) \equiv B(1,7) \odot (B(2,7) \odot B(3,7))$$

REMARQUES :

1) Le résultat de la réduction d'un tableau à p dimensions est un tableau à p-1 dimensions.

2) La vérification de compatibilité des variables dans les opérations booléennes doit évidement tenir compte des réductions de dimensions apportées par d'éventuelles opérations de réduction.

3) En toute rigueur la réduction ne fait pas disparaître une dimension mais la réduit à la longueur 1.

EXEMPLE :

La réduction d'une matrice est une matrice ligne ou colonne :

⍟ / B(1 : 3, 1 : 4) C (1 : 3, 0 : 0)
⍟ / B(1 : 3, 1 : 4) D (0 : 0, 1 : 4)

Cette remarque est indispensable pour permettre de définir par concaténation toutes les nouvelles variables que l'on veut.

EXEMPLE :

(4) E (1 : 3, 1 : 4) = (∧/∿ B) & (∨/∿B) & (=/B)

Or en revenant à l'exemple précédent :

∧/∿B, ∨/∿B et =/B sont de la forme D(0 : 0, 1 : 4)
donc (4) est bien correct.

On peut par regroupement de lignes ou de colonnes créer une matrice, un tableau,... . Une composante de tableau est un tableau de même nature dont toutes les directions sont dégénérées.

d_3 L'opérateur incrémentation sera note + ou - selon qu'il y a incrément ou décrément.

Il a pour but d'ajouter ou de soustraire une constante fixe au contenu d'un registre et se traduira en hardware par une réalisation particulière en général, bien moins couteuse qu'un additionneur classique.

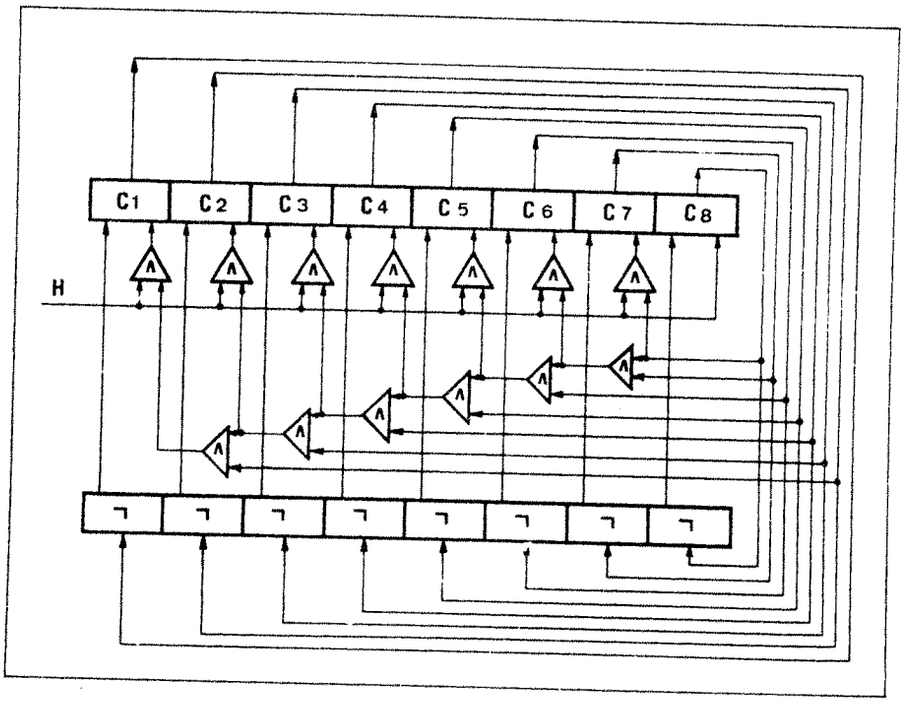
L'exemple d'application le plus courant de cet opérateur est l'incrémentation de 1 d'un registre.

Par exemple un compteur C.

<H> C ←+ 1 ;

EXEMPLE DE REALISATION :

Registre C (1 : 8)



e) Expressions booléennes :

Nous avons vu au chapitre précédent la description sous l'aspect d'un réseau, d'un circuit combinatoire. L'expression booléenne représente elle la définition fonctionnelle en CASSANDRE des cricuits combinatoires. Un grand nombre de réalisations possibles de chaque expression existent avec des modules technologiques et des circuits logiques différents, définir le passage à l'une de ces réalisations c'est définir un algorithme de synthèse. C'est un tel algorithme lié à la syntaxe de CASSANDRE qui est utilisé par Monsieur LIDDEL | |.

Les expression sont formées par une combinaison des variables généralisées et des opérateurs définis au début de ce chapitre. Une expression pour être valide doit vérifier les règles de compatibilité entre variables déjà énoncées et les règles de priorité définies par la syntaxe pour les opérateurs (Voir carte syntaxique |53-b|).

Chaque expression se traduira par une partie de réseau logique dont les entrées seront les variables apparaissant dans cette expression, et la sortie est la valeur de l'expression.

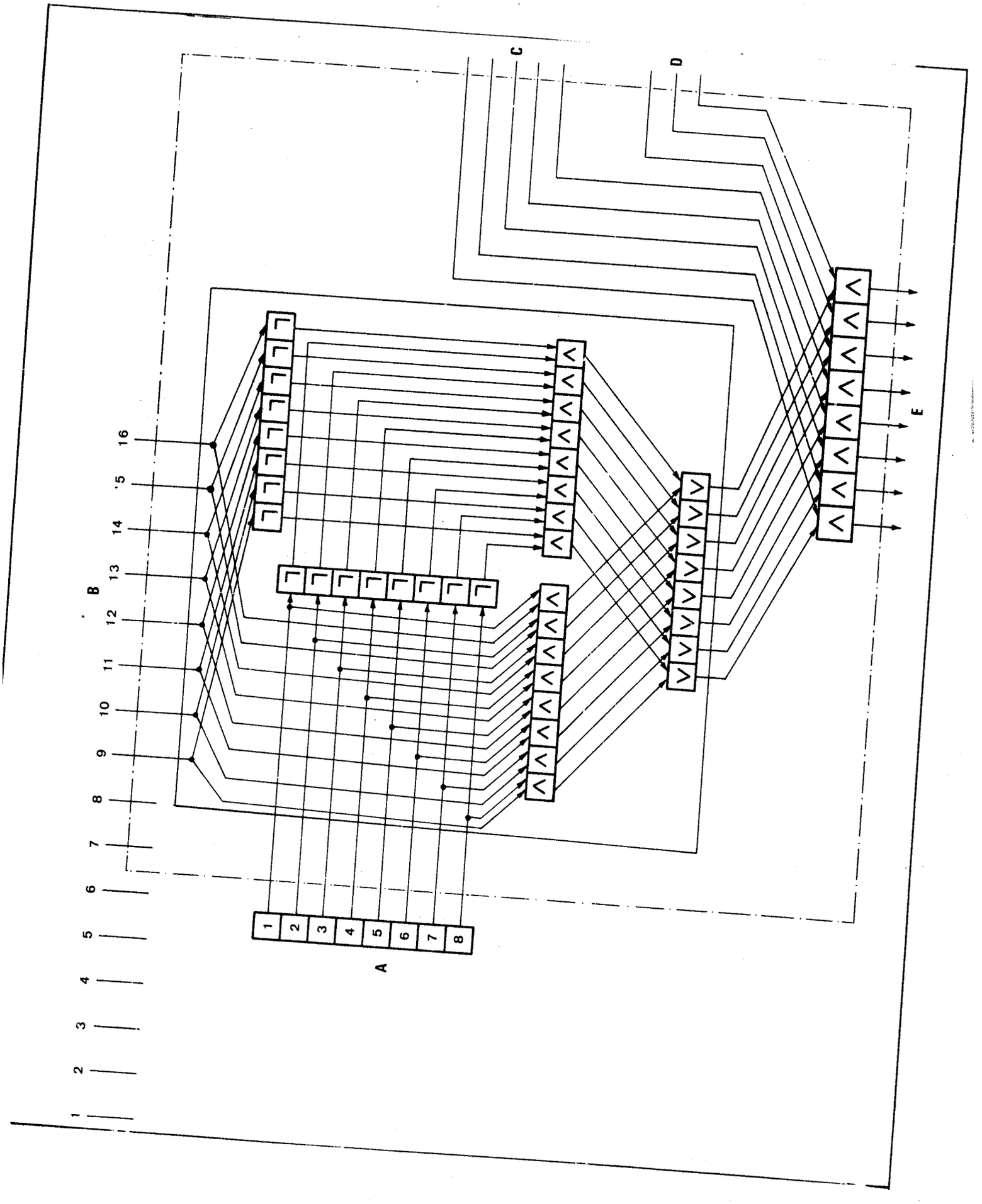
Une expression à formellement le sens de la fabrication d'un signal. Le temps d'atablissement de tous les signaux déclarés ou formels doit être, sauf spécification contraire, supposé inférieur à la période de l'horloge hypothèses courant dans les systèmes synchrones.

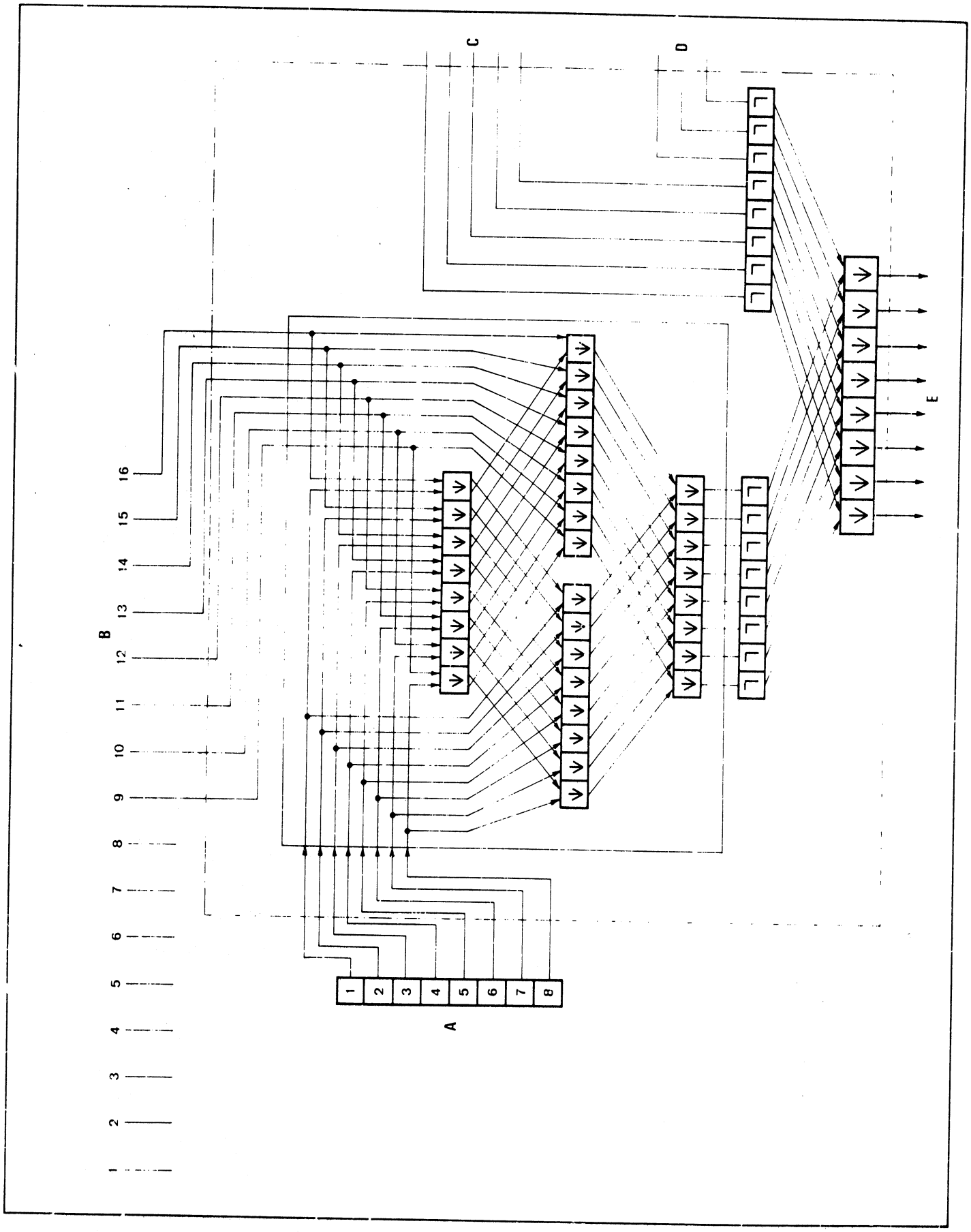
Comme pour les variables généralisées on introduit aussi les opérateurs & et $\sim$, mais au niveau de l'expression simple.

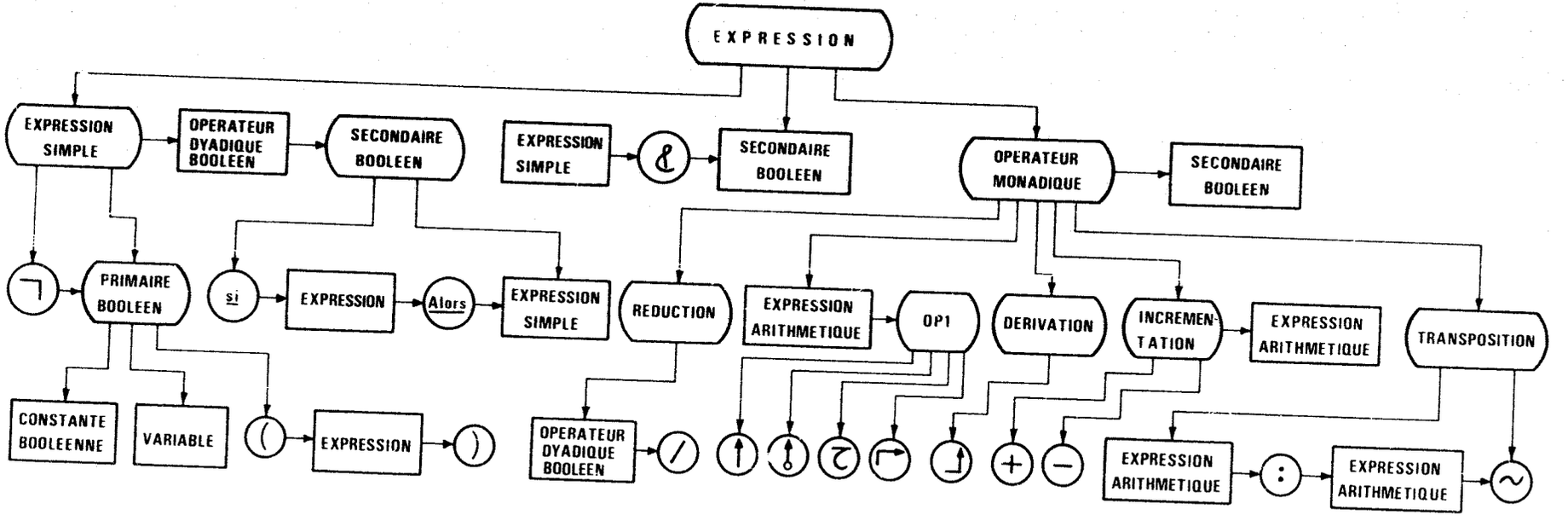
Ces opérateurs appliqués au niveau de l'expression seront en général séparés par des parenthèses, des mêmes opérateurs appliqués au niveau des variables.

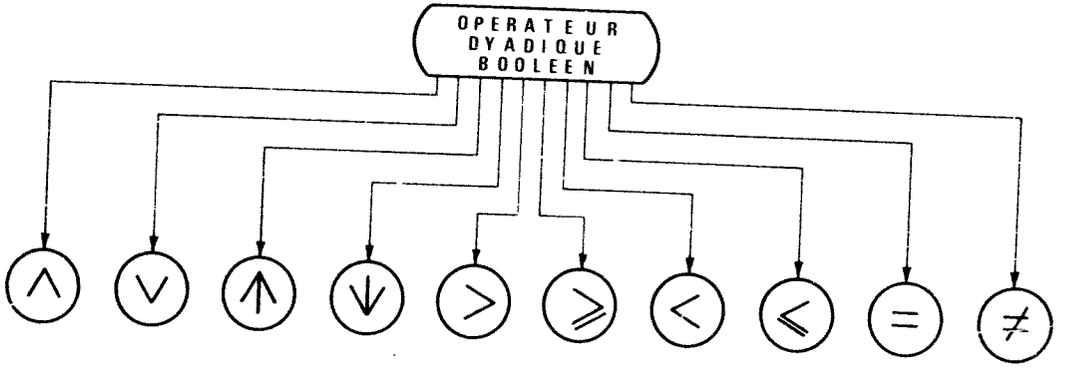
EXEMPLE : Registre A(1 : 8) ;
Signal B(1 : 16), C(0 : 4), D(1 : 3), E(1 : 8) ;
E := (A = B (9 : 16) Λ (C&D) ;

EXEMPLES DE CIRCUITS CORRESPONDANTS.




A
B
C
D
E
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16


EXPRESSION
EXPRESSION SIMPLE
OPERATEUR DYADIQUE BOOLEEN
SECONDAIRE BOOLEEN
EXPRESSION SIMPLE
&
SECONDAIRE BOOLEEN
OPERATEUR MONADIQUE
SECONDAIRE BOOLEEN
¬
PRIMAIRE BOOLEEN
si
EXPRESSION
Alors
EXPRESSION SIMPLE
REDUCTION
EXPRESSION ARITHMETIQUE
OP1
DERIVATION
INCREMENTATION
EXPRESSION ARITHMETIQUE
TRANSPOSITION
CONSTANTE BOOLEENNE
VARIABLE
(
EXPRESSION
)
OPERATEUR DYADIQUE BOOLEEN
/
+
−
EXPRESSION ARITHMETIQUE
:
EXPRESSION ARITHMETIQUE
~


OPERATEUR DYADIQUE BOOLEEN
∧
∨
↑
↓
>
≥
<
≤
=
≠

B - NIVEAU ELEMENTAIRE D'OPERATIONS

On peut distinguer en CASSANDRE Un certain nombre d'opérations élémentaires, équivalence de variables, connexion de signal, connexion d'unité, ordre allera, chargement de registre ou de variable d'état, ordre faire, génération d'impulsion.

Eliminons tout de suite l'équivalence de variable définie au chapitre précédent, qui est une opération purement formelle, et dont la syntaxe est très simple :

<EQUIVALENCE> ::= <VARIABLE> ≡ <VARIABLE-GENERALISEE>

Les sept autres opérations se ramènent en fait à deux sortes d'opérations, les opérations de mémorisation qui doivent être synchronisées par une impulsion et les opérations de connexion qui permettent de définir les liaisons entre réseaux combinatoires, de définir des fonctions booléennes multiples et des fonctions de fonctions. Ordre allera, chargement de registre, chargement de variable d'état appartiennent au premier type, connexion de signal, connexion d'unité et ordre faire appartiennent à l'autre type.

a) Affectation élémentaire :

La syntaxe en est simple :

<AFFECTATION> ::= <VARIABLE-GENERALISEE>← <EXPRESSION>

Si la variable qui se trouve à gauche du symbole ← d'affectation est de type état, le résultat de l'expression doit être une valeur d'état, et toutes ces variables ont des dimensions non précisées comme nous l'avons dit au chapitre précédent. Ceci ne veut pas dire que, dans l'expression, des variables de type Registre ou Signal ne peuvent pas apparaître (par exemple dans le calcul de conditionslogiques).

Il est impossible de vérifier syntaxiquement la compatibilité de l'opération et l'homogénéité du résultat comme état.

Ce genre de vérification délicate peut être envisagée dans le programme qui suit l'analyseur syntaxique.

EXEMPLE :

Etat Z1, Z2 ; Registre A ; Impulsion H ;

Supposons que ETA 1, ETA 2, et ETA 3 soient trois étiquettes :

<H> Z ← si A alors ((si Z = ETA 1 alors ETA 2) V
(si Z ≠ ETA 1 alors ETA 1)) V

si ¬A alors ETA 3 ;

Nous voyons avec Z = ETA 1 qu'une expression d'état peut parfaitement avoir une valeur de condition logique et nous renvoyons pour cette notion à ANCEAU [3].

Si la variable à gauche du symbole ← est de type Registre, le résultat de l'expression doit alors avoir des dimensions compatibles avec celle du Registre. La vérification de compatibilité déjà spécifiée pour les expressions doit s'étendre aux deux membres de l'affectation.

EXEMPLE : Registre A (1 : 8, 1 : 36), B (1 : 4, 1 : 8, 1 : 36) ; Etat Z ;
Signal C (1 : 8, 1 : 72) ; Impulsion H ;

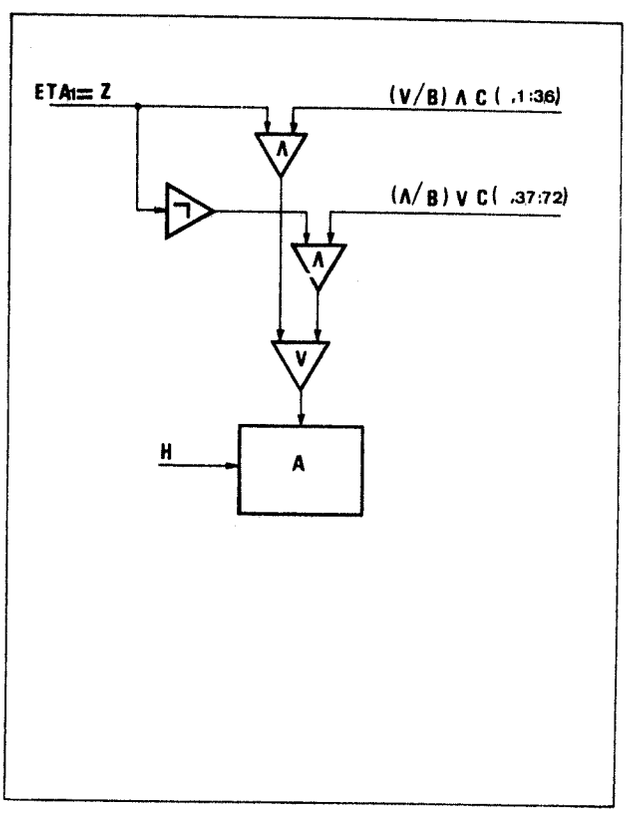
Utilisons encore l'étiquette ETA 1.

<H> A ← si ETA 1 = Z alors ((V/B)∧C(, 1 : 36))∨
si ETA 1 ≠ Z alors ((/B)∨C(, 37 : 72)) ;

On peut vérifier la compatibilité dans l'exemple ci-dessus.

L'impulsion qui effectue les affectations des deux types précédents est placée devant l'opération entre crochets pointus < >.

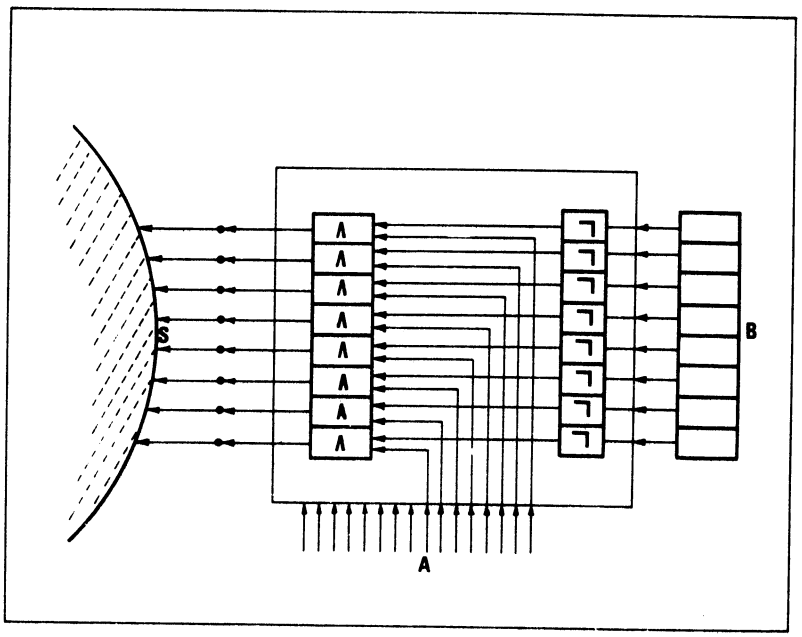
Exemple de réalisation



b) Connexion de signal, impulsions conditionnées

La connexion d'une expression E à un signal S correspond à l'action physique de souder sur les fils appelés S les sorties du circuit qui réalise E. Elle se note S := E

EXEMPLE : Signal S(1 : 8), A(1 : 16) ;
Registre B(1 : 8) ;
S := A(9 : 16)Λ¬B ;



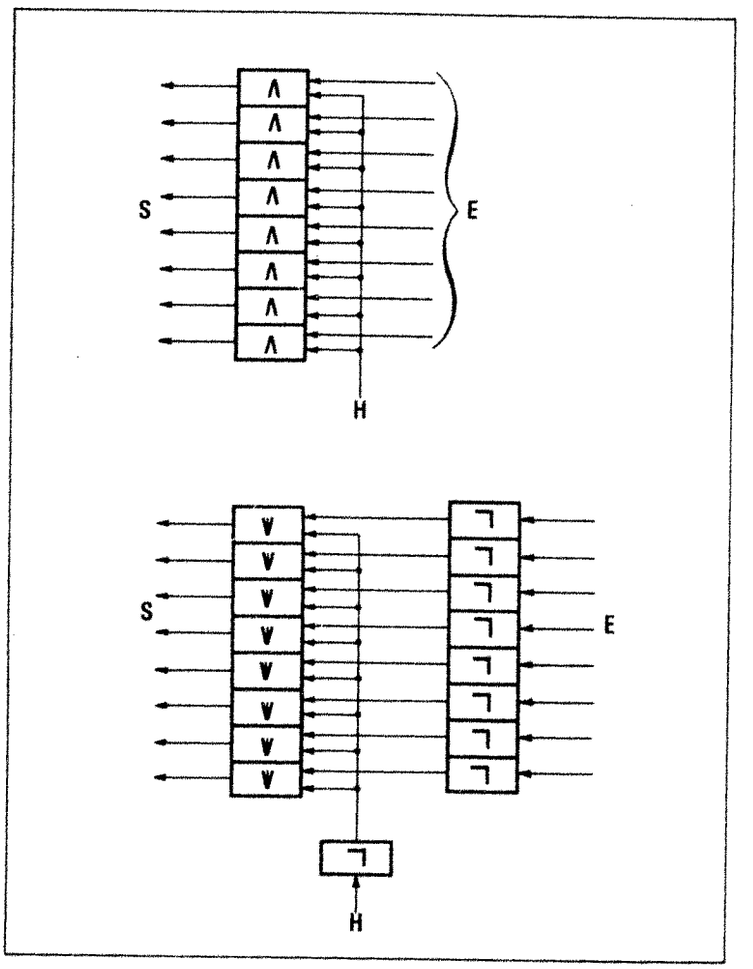
Cette opération n'a pas un sens temporel comme l'affectation de registre, l'échange d'information au niveau de la connexion est indépendant des impulsions, alors qu'au niveau des entrées d'un élément de mémorisation les deux sont liés.

Si une impulsion se produisait au niveau d'une connexion S, elle se propagerait dans le réseau combinatoire dont S est peut être une entrée et S serait alors un ensemble d'impulsions en phase.

Ceci peut toutefois être admis si on déclare S comme Impulsion et non comme signal.

Dans ce cas une connexion conditionné par une horloge : <H> S := E ; aura pour effet de générer autant d'impulsionsque S a de fils, en phase avec H et dont l'existence est conditionnée par la valeur des sorties de E ?

EXEMPLE : Impulsions S (1 : 8), H ;
En base I



ou en base III

De même, s'il se trouve dans E un opérateur de dérivation, le résultat de E sera de type impulsion et la connexion, non conditionnelle par une horloge produira encore un résultat devant être connecté à une variable S déclarée impulsion.

Dans la description en CASSANDRE on trouvera donc les deux types d'opérations précédemment définies, certaines conditionnées par des horloges d'autres pas.

On peut reconnaître chaque opération par le type (Registre, Signal et impulsion) de la variable située à gauche du symbole d'affectation. S'il y a des chargements de Registre, au moins une horloge sera présente entre crochets.

EXEMPLE : <H> O1, O2, O3, ... On ;

Si O1 est de la forme Registre ← <expression>
H conditionnera l'opération O1 alors de type affectation.

Si O2 est de la forme Signal := <expression>

H ne devra avoir aucun effet sur O2 de type connexion.

Si O3 est de la forme impulsion := <expression> et si l'expression contient des dérivations produisant en fin de compte des impulsions, H ne devra plus influencer O3.

Si Oi est de la forme impulsion := <expression> mais que le résultat de l'expression soit de type Signal, alors l'horloge H conditionnera la connexion pour produire des impulsions en phase avec H.

Monsieur ANCEAU qui a beaucoup contribué à clarifier ces notions a suggéré que les connexions de signal et les affectations de registre soient séparées et représentées par deux symboles différents (:= et ←). Aucune connexion ne serait placée sous la portée d'une impulsion.

Le mode de fonctionnement du matériel décrit sera synchrone, synchronisé ou asynchrone suivant que les impulsions sont des horloges indépendantes, qu'il y a des impulsions conditionnées et en phase avec une horloge ou que des impulsions quelconques soient crées à l'aide des opérateurs. ↑, → et τ.

Nous irons plus loin en notant <== les créations d'impulsions, (impulsion <==<expression>), on aura alors trois types d'opérations l'affectation (←), la connexion (:=), la génération d'impulsion (<==).

Les règles simples suivantes définissent les modes de fonctionnement.

1 En mode synchrone l'opération<= n'existe pas.

2 En mode synchronisé l'opération<= est toujours conditionnée par une horloge (on crée des impulsions en phase).

3 En mode asynchrone l'opération <= est soit sous la portée d'une horloge, soit en dehors (impulsions quelconques).

4 Dans les trois modes l'opération ← est toujours conditionnée par une horloge, l'opération := jamais.

c) Connexion d'unité :

La connexion d'unité résume toute une série de connexions de signaux qui sont les entrées et sorties de cette unité.

Toute unité utilisée dans une autre doit être déclarée en Externe par son nom et la liste des dimensions de ses entrées et sorties. (il n'est pas utile de mettre la liste des noms de ces entrées-sorties puisqu'ils sont internes à l'unité connectée. De l'extérieur seul leur format est utile).

Donnons un exemple pour illustrer la déclaration d'externe.

Soit U1 une unité dont l'entête est :

Unité U1 (E1(1 : 8), E2(1 : 4, 1 : 36), E3 ; S(-1 : 13)) ;

A l'intérieur d'une unité U2 ; U1 peut-être utilisée, elle sera alors déclarée :

Externe U1(8,(4,36), 1 ; 15) ;

Si une entrée ou une sortie est un tableau a plus d'une dimension, il suffit de mettre à nouveau une parenthèse autour de la liste de ses dimensions, c'est le cas ici pour la deuxième entrée.

La connexion se fait à l'aide de signaux (ou d'expressions) de l'unité principale.

Ceux-ci apparaissent dans l'ordre de connexion entre parenthèses, à la place de l'entrée ou de la sortie à laquelle ils sont liés.

EXEMPLE : Signal A(1:8), B(1:8), S(1:8) ;
Externe G(8, 8 ; 8) ;
G(AVB, AΛB ; S) ;

REMARQUES :

1) S'il y a des registres en paramètres d'une connexion d'unité ce ne peut être qu'en entrée. En effet la sortie d'un registre est un signal, mais sont entrée doit être affectée sous-top d'horloge. Or,

2) Une connexion d'unité n'est jamais conditionnée par une impulsion.

3) S'il y a des expressions en paramètre d'une connexion d'unité ce ne peut être qu'en entrée (la sortie de l'expression est connectée à l'entrée correspondante).

4) Si l'on ne veut pas connecter une entrée ou une sortie, il suffit de la laisser en blanc dans la connexion d'unité.

5) La connexion de la même unité peut se faire en plusieurs étapes, avec plusieurs occurences du nom de l'unité suivi de paramètres différents. Si la même entrée de l'unité est connectée plusieurs fois ce doit être sous des conditions logiques disjointes deux à deux.

6) Si plusieurs exemplaires d'une même unité doivent être connectés dans une autre, on peut rajouter après le nom un numéro pour les distinguer (ou une expression arithmétique dont le résultat constituera ce numéro).

7) Les sorties dans un appel d'unité sont toutes des variables généralisées de type signal.

8) La compatibilité de dimension entre les signaux (ou expressions) et les entrées-sorties respectives qu'ils remplacent doit être rigoureusement vérifiée. (Grâce à la déclaration d'externe).

9) Une sortie d'une unité ne peut-être relié qu'à une entrée d'une ou plusieurs autres unités de même niveau.

Par contre une entrée d'unité ne sera liée qu'à une seule sortie d'une autre unité de même niveau (pour recevoir plusieurs sorties directement il faudrait un opérateur de concentration comme un ou logique (V)).

10) Cependant, les entrées et les sorties d'une unité peuvent être reliées respectivement aux entrées et sorties de l'unité qui les contient.

Illustrons ce qui précède par la description d'un réseau logique simple que forme une case d'additionneur à l'aide des unités suivantes :

Unité N12(A2,B3 ; C2) ;
C2 := ¬A2 Λ ¬B2 ;
Unité N13(A3,B3,D3 ; C3) ;
C3 := ¬A3Λ¬B3Λ¬D3 ;
Unité N14 (A4,B4,D4,E4 ; C4) ;
C4 := ¬A4Λ ¬B4Λ¬D4Λ ¬E4 ;

Appelons maintenant ADDER l'additionneur à une case construite de la manière suivante à partir des composants NI précédents.

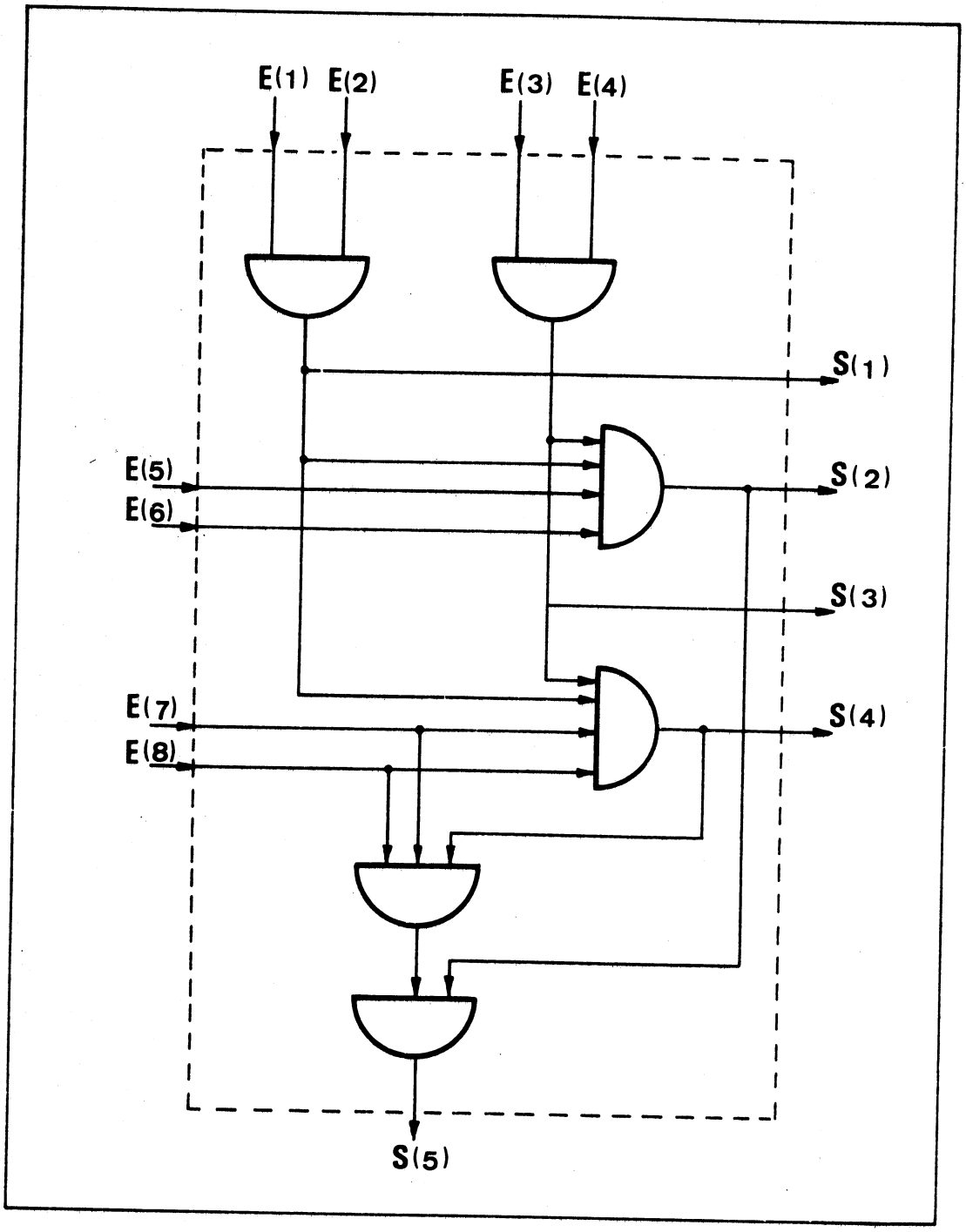
```
Unité ADDER (E(1:8) ; S(1:5) ;
Externe NI2(1,1 ; 1) , NI3(1,1,1;1,),
        NI4(1,1,1,1;1) ;

Signal J ;
```

Connexion d'unité en trois exemplaires

```
{ NI2 [1]  (E(1),E(2) ; S(1)) ;
{ NI2 [2]  (E(3),E(4) ; S(3)) ;
{ NI2 [3]  (J ,  S(2) ; S(5)) ;
```

Connexion d'unité en deux exemplaires

```
  NI3  (E(7),E(8),S(4) ; J) ;
{ NI4 [1] (S(3), S(1), E(5), E(6) ; S(2)) ;
{ NI4 [2] (S(3), S(1), E(7), E(8) ; S(4)) ;
```


E(1)
E(2)
E(3)
E(4)
S(1)
E(5)
E(6)
S(2)
S(3)
E(7)
E(8)
S(4)
S(5)

d) Signal-unité :

On appellera signal-unité une connexion d'unité dans laquelle l'une des entrées ou sorties est remplacée par *.

L'occurence d'un signal-unité est équivalente à celle du signal d'entrée ou de sortie remplacé par l'étoile. Le signal-unité est donc une variable de type signal, on peut l'employer dans une expression, en particulier on peut le mettre en entrée ou sortie d'une autre connexion d'unité.

Si l'étoile * remplace une entrée le signal-unité pourra se placer à gauche d'un :=. Si elle remplace une sortie il pourra faire partie de l'expression à droite du :=.

Cette notion est utile pour exprimer à la fois la connexion des unités employées et les remplacer par un circuit booléen si elles représentent elles-mêmes des fonctions booléennes et si on veut les faire disparaître.

Ce qui précède apparaîtra mieux sur un exemple. Reprenons l'additionneur précédent, on peut maintenant le décrire :

```
Unité ADDER (E(1 : 8) ; S(1 : 5) ;
Externe NI2 (1, 1 ; 1), NI3 (1, 1, 1 ; 1), NI4 (1, 1, 1, 1 ; 1) ;
S(1) := NI2 [1] (E(1), E(2) ; *) ;
S(2) := NI4 [1] (E(5), E(6), S(1), S(3) ; *) ;
S(3) := NI2 [2] (E(3), E(4) ; *) ;
S(4) := NI4 [2] (E(7), E(8) ; S(1), S(3) ; *) ;
S(5) := NI2 [3] (S(2), NI3 (E(7), E(8), S(4) ; *) ; *) ;
```

Nous verrons au chapitre suivant que l'utilisation d'un signal-unité dans une affectation de registre conditionnelle peut alléger sensiblement la description.

Supposons que l'on veuille calculer les équations booléennes de :

S(1), S(2), S(3), S(4), S(5),

il faut remplacer les unités NI1, NI2, NI3, par des expressions.

Si le signal-unité représente la sortie d'un réseau combinatoire et d'une unité U, il existe dans la description de l'unité U dont il est issu une relation du type :

(1) S := <expressions>

où S est justement la sortie remplacée par une *.

Il suffit de remplacer l'occurence du signal-unité par l'équation (1) où l'on remplace les variables locales à l'unité U, par celles qui apparaissent dans le signal-unité.

Quand on a fait la même chose pour tous les signaux-unités de la description il suffit d'appliquer un algorithme simple d'élimination, (voir LIDDELL [56]), pour avoir les équations souhaitées.

Sur l'exemple précédent, on obtiendrait :

S1 := $\neg$E(1) $\wedge$ $\neg$E(2) ;
S2 := $\neg$E(5) $\wedge$ $\neg$E(6) $\wedge$ $\neg$S(1) $\wedge$ $\neg$S(3) ;
S3 := $\neg$E(3) $\wedge$ $\neg$E(4) ;
S4 := $\neg$E(7) $\wedge$ $\neg$E(8) $\wedge$ $\neg$S(1) $\wedge$ $\neg$S(3) ;
S5 := $\neg$S(2) $\wedge$ (E(7) $\vee$ E(8) $\vee$ S(4)) ;

e) Ordre Allera :

Nous avons vu que les valeurs des états possibles d'une unité sont représentées par des étiquettes.

La variable interne de l'unité considérée comme machine séquentielle est de type état au sens précédemment défini, mais elle n'est pas déclarée car on supposera implicitement qu'une telle variable (exemple : Registre d'états) existe dans chaque unité.

Les étiquettes repèrent les valeurs que prendra cette variable indépendamment de tout codage.

Il résulte de la définition des variables d'état, qu'une variable déclarée comme telle est élément de mémorisation (analogue à un Registre) dont on ignore la dimension (puisque les états ne sont pas codés).

Il est évident, après la définition d'un codage, qu'une telle variable pourrait alors se déclarer Registre.

L'ordre allera signifiera toujours l'affectation d'une valeur d'état à la variable interne de l'unité qui contient cet allera.

Cet ordre est donc synchronisé par une horloge, comme une affectation de Registre.

e1) allera <ETIQUETTE>, signifie l'affectation, de l'état repéré par l'étiquette à la variable interne.

e2) allera <VARIABLE-D'ETAT>, signifie transfert dans la variable interne de la valeur mémorisée dans la variable d'état,

- Dans un processus de simulation du fonctionnement :

l'allera prend alors le même sens que l'ordre

allera ALGOL dans le cas e_1),

et qu'un branchement indirect dans le cas e_2). (Analogue avec l'ordre TRA et TRA * en MAP.

Lorsque l'état suivant de la machine décrite est celui qui correspond à l'instruction immédiatement en-dessous immédiatement en-dessous dans la description, il convenait de définir une forme condensée de l'instruction Allera. Le symbole → est traduit comme Allera <Etiquette qui suit immédiatement dans la description> .

On rejoint ainsi les facilités d'écriture propres aux langages de programmation, dans lesquels Allera est implicite quand il s'agit d'exécuter l'instruction immédiatement après dans l'écriture du programme.

EXEMPLE : Les écritures suivantes sont équivalentes :

```
ETA 1 {} A ← B, Allera ETA 2 ;          ETA 1 {} A ← B, → ;
ETA 2 {} C ← D, Allera ETA 3 ;    ⟺    ETA 2 {} C ← D, → ;
ETA 3 {} T ← U, Allera ETA 44;          ETA 3 {} T ← U, → ;
- - - - - - - - - - - - - - -          - - - - - - - - - -
ETA n {} AB ← EFG, Allera ZOZO ;        ETA n {} ABC ← EFG, Allera ZOZ
```

e3) Etat non spécifié :

L'ordre Allera, ou → défini l'état suivant de la machine.

Une autre possibilité est offerte au concepteur : lorsqu'une instruction comporte l'ordre Allera ∅, celà signifie que l'état suivant est provisoirement non défini. Cette possibilité peut être intéressante, cet état sera choisi au moment de la synthèse comme étant celui qui simplifie au maximum le réseau de contrôle.

On peut également penser qu'après avoir fait un certain travail la machine continued'évoluer, sans que l'on ait besoin de préciser cette évolution, on en reprendra le contrôle en cas de besoin en forçant l'état initial de l'extérieur.

e4) Règle des Allera :

Chaque étiquette correspondant à une valeur de la variable interne, rappelons par conséquent qu'on ne peut pas se transférer simultanément à deux états différents, (sauf dans le cas voulu d'un automate non déterministe.

EXEMPLE : -.- ETA 3 : A ← B, Allera ETA 2, C ← D, → ;

Cette instruction est fausse.

-.- ETA 3 : A ← B, C ← D, si E alors allera ETA 2 sinon →;

Celle-ci par contre est correcte.

-.- ETA 3 ; A ← B, allera ETA 2, si C alors → ;

est également faux puisque lorsque C = 1 on devrait faire 2 allera différents.

-.- ETA 3 : A ←B, si D alors allera ETA 2, si C alors → ;

Pourra être correcte si D ∧ C = 0, quand on aura 1(D∨C) = 1, l'état suivant sera non spécifié.

e5) Syntaxe l'Allera ;

<ALLERA> ::= allera <EXPRESSION-D'ETAT>.

Pour la syntaxe de l'expression d'état, voir syntaxe CASSANDRE-360

Nous dirons simplement ici, que l'expression d'état a pour résultat une valeur d'état ou une variable d'état, et nous donnerons quelques exemples suffisants pour définir la notion.

- allera si B alors ETA 1 sinon ETA 2.

B est une condition logique ETA 1 et ETA 2 des expressions d'états.

C'est un classique branchement conditionnel.

- allera ETA 1 v ETA 2 v .... v ETA K.

Nous exprimerons simplement ainsi une version restreinte de l'allera indéterminé, au moment du codage on choisira une valeur d'état parmi le sous ensemble ETA 1, ETA 2, ... ETA K.

- allera ETA 1 Λ ETA 2 Λ...Λ ETA K

Nous nous servirons simplement de cette forme pour décrire le graphe d'un automate non déterministe et la transition simultanée à ETA 1, ETA 2, ... et ETA K.

f) Ordre faire :

Lorsqu'une fonction apparaît plusieurs fois dans la description d'une machine, ou a la possibilité de ne la décrire (et aussi de n'en faire la synthèse) qu'une fois, en définissant un signal.

Lorsqu'un ensemble de microopération simultanées, ou même lorsqu'une succession de microinstructions se reproduisent à différents moments de la description et appartiennent en commun à plusieurs microprogrammes de la machine il faut définir un moyen, formellement, d'en éviter l'écriture multiple, matériellement, d'alléger la réalisation.

Nous distinguerons pour la clarté de la définition de faire les deux cas suivants :

f1) faire s'applique à un ensemble d'actions simultanées ne contenant pas d'allera.

EXEMPLE : SEQU : <H> I_1, I_2, ... I_p ;

ETA 36 : <H> J_1, J_2, ..., Faire SEQU, ... , J_q ;
ETA 68 : <H> K_1, K_2, ..., Faire SEQU, ... , K_q ;

L'ordre a pour résultat d'"injecter" à l'emplacement de Faire la suite I_1, I_2, ... I_p ;

Du point de vue fonctionnement ceci est équivalent à :ù

ETA 36 : <H> J_1, J_2 , ... , I_1 , I_2 , ... I_p , ... , J_q ;
ETA 68 : <H> K_1, K_2 , ... , I_1 , I_2 , ... I_p,, ... , K_q ;

Mais dans la réalisation, comme dans la description, les circuits relatifs à SEQ n'apparaitront qu'une seule fois.

. SEQU est une étiquette, qui repère donc une valeur d'état de la machine mais dans le cas présent les instructions I_1, I_2 , ... I_p, ne contiennent pas d'ordre allera et par exemple dans ETA 36 et ETA 68, le passage à l'état suivant ne dépend pas de l'ordre Faire.

Dans cette utilisation, qui a uniquement pour objectif une économie d'écriture et de réalisation, l'ordre Faire ressemble à une macroinstruction de substitution.

f2) Faire s'applique à un ensemble d'actions simultannées contenant un ou des allera.

(faire initialise une séquence d'instructions).

Supposons pour clarifier un exemple qu'une séquence d'opérations SEK_1 , SEK_2 , ... , SEKℓ, intervienne en commun dans plusieurs microprogrammes de la machine dans l'ordre où nous les écrivons (cet ordre est juste destiné à clarifier l'exemple, il n'a aucune raison d'être dans la pratique).

SEK_1 : <H> I_{11}, I_{12} , ... , I_{1p} ,→ ;
SEK_2 : <H> I_{21} , ... , I_{2q} ,→ ;

SEKℓ : <H> I_{e1}, ... I_{ek}, allera Z ;

ETA 36 : <H> J_1, ... , Faire SEK_1, ... J_1 ;
ETA 67 : <H> K_1, ... , Faire SEK_1, ... K_i ;

L'occurence de Faire SEK_1, a pour SEK_1 le même effet que dans a) pour SEQU, mais comme SEK_1 contient allera SEK_2, ETA 36 et ETA 67 ne devront pas contenir d'ordre allera sous les mêmes conditions que l'ordre faire.

L'état suivant de ETA 36 et ETA 67 sera SEK 2.

Nous avons mis en évidence dans SEKℓ, un ordre allera Z, supposons qu'il existe entête de l'unité une déclaration Etat Z ;

On peut alors introduire dans ETA 36 un ordre : Z ← ETA37,

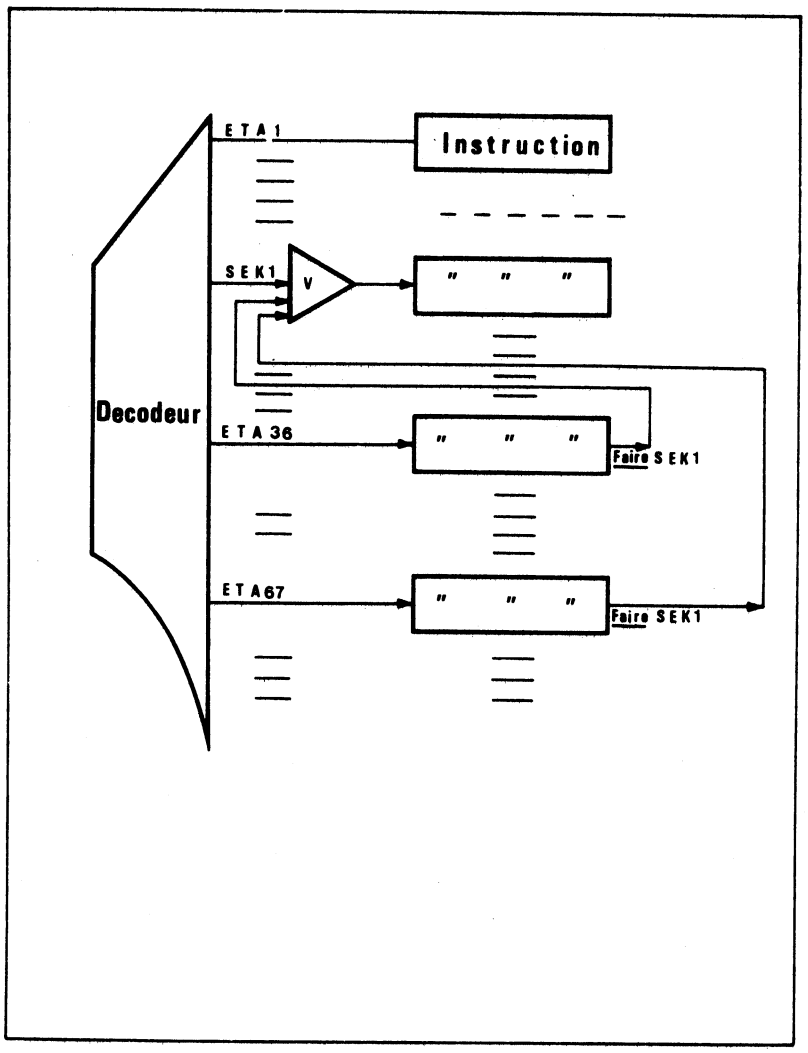
et dans ETA 67 un ordre : Z ← ETA68

. Après l'éxécution des ordres faire, quand la machine se trouvera dans l'état SEKℓ, le passage à l'état suivant se fera par allera Z en ETA 37 ou ETA 68 suivant que Z a alors pour valeur l'une ou l'autre de ces deux valeurs d'état.

On voit alors comment la notion de variables d'état permet ici commodément de définir une notion de "sous-microprogrammes" et sans doute d'alléger sensiblement écriture et réalisation.

On peut aussi noter que l'ordre allera SEK1 dans les deux cas aurait produit le même effet, associé à un ordre Z ← ETA 37 ou Z ← ETA 68. Dans cette utilisation faire a l'avantage de bien repérer une suite privilégiée d'opérations et économise un cycle d'horloge par rapport à l'allera.

f3) Interprétation hardware de faire : | 4 |



Nous supposerons que l'unité possède un décodeur d'état, qui produit autant de signaux que la machine a d'états.

Un seul signal est activé à la fois et il conditionne les opérations qui sont, dans la description, dans la portée de l'état correspondant.

L'ordre faire peut rendre ainsi actifs plusieurs signaux de sortie du décodeur d'état à un instant donné.

L'utilisateur doit évidemment prendre toutes précautions pour que toutes ces opérations en principe destinées à s'éxécuter par groupes séparés soient bien compatibles et puissent s'éxécuter simultanément.

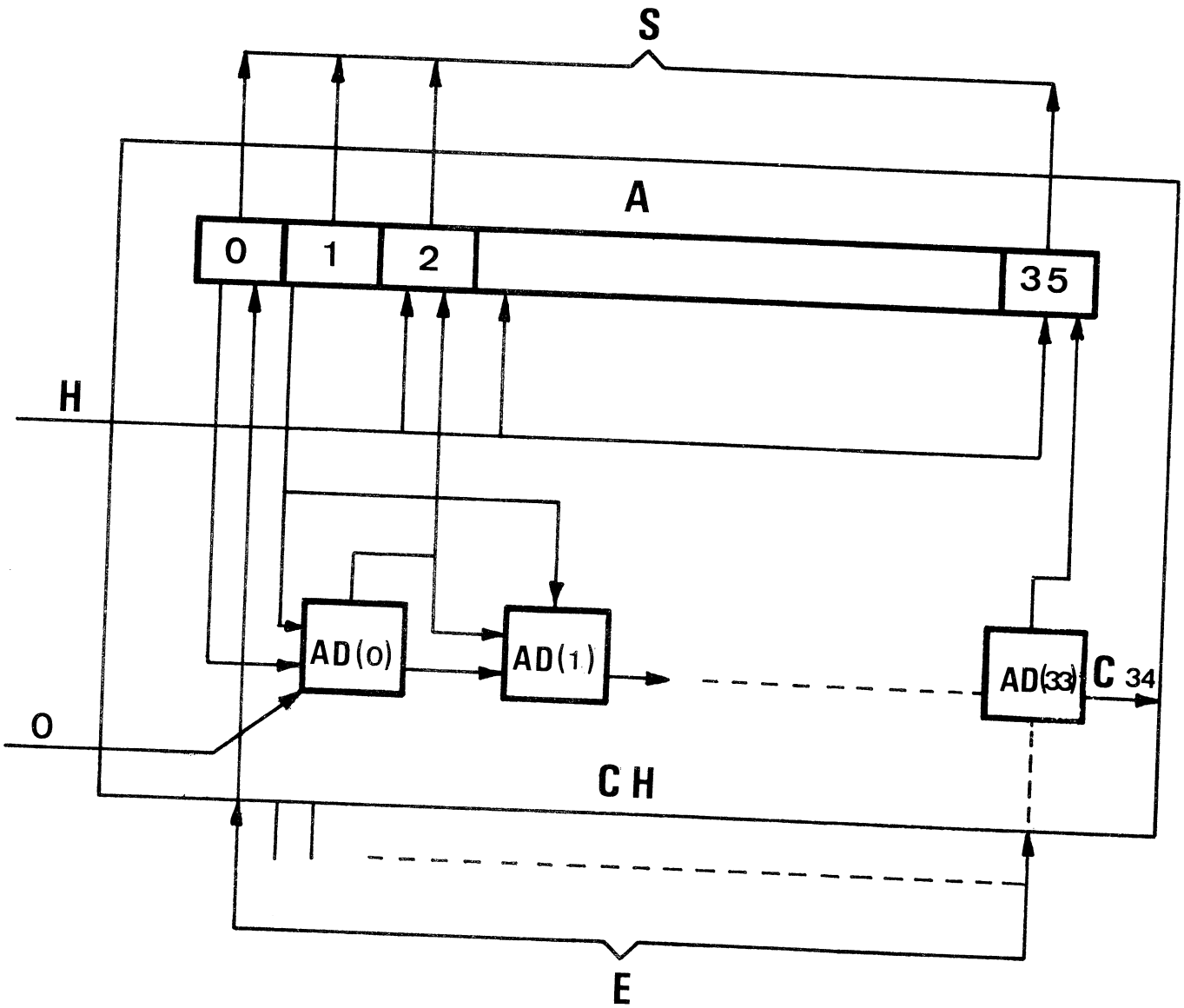
Remarquons enfin que faire n'est vraiment indispensable que dans le cas a) précédent. Dans ce cas l'étiquette (SEQU dans l'exemple) repère un état de la machine dans lequel elle ne se trouvera jamais (puisqu'elle n'en sortirait pas vu que SEQU ne contient aucun allera).

Cet "état fictif" ne sera pas distingué des autres états, pour simplifier la compilation de la description, mais on pourrait se dispenser de le coder. On rejoint ainsi la définition de l'ordre faire et de séquence que l'on trouve dans [65] et [59].

Nous verrons dans l'application à la Microprogrammation des applications de l'ordre faire.

Donnons un exemple pour illustrer quelques notions de ce chapitre : (Exemple emprunté à | |).

```
Unité CH (H, E(1 : 36) ; S(0 : 35)) ;                          } Entête
Registre A(0 : 35) ;                                           }
Signal C(0 : 36 , B(2 : 35) ;                                  } Déclarations
Externe AD(1, 1, 1 ; 1, 1) ;                                   } spécifications
Impulsion H ;                                                  }
                C(0) := 0 ;                                    } Connexions
                S    := A ;                                    } indépendantes
                  A  := E ;                                    }
ETA 1 : <H> pour I égal à 33 faire                             } Automate
A (I+2)←AD |I| (A(I,A(I+1),C(I) ; *, C(I+1)) ,
"EXEMPLE DE SIGNAL-UNITE"
allera si C(34) alors ETA 2 sinon ETA 1 ;
" EXEMPLE D'ALLERA <EXPRESSION-D'ETAT>"
ETA 2 : pour J égal 2 à 35 faire
AD [J] ((A(35-J), A(34-J), C(J-2) ; B(J), C(J-1)) ;
"EXEMPLE DE CONNEXION D'UNITE"
<H> A ← B&C(34) & 1, allera ETA 1 ;
```


S
A
0
1
2
35
H
AD(0)
AD(1)
AD(33)
C 34
O
CH
E

C - NIVEAU INSTRUCTION

a) Opération conditionnelle élémentaire :

Il est indispensable pour exprimer le fonctionnement d'un organe un tant soit peu compliqué, de pouvoir conditionner certaines actions par les contenus des registres de cet organe à cet instant.

Un moyen commode consiste à adopter le formalisme des instructions conditionnelle utilisées en programmation.

Nous avons défini au paragraphe précédent les opérations élémentaires de CASSANDRE, affectation de registre, d'état, génération d'impulsion, connexion de signal d'unité, ordre faire. Nous allons maintenant définir la notion de condition logique.

Puisque les variables en CASSANDRE sont booléennes chaque composante de l'une d'elles peut en soit constituer une condition logique.

Plus généralement, on pourra dans une Unité considérer comme condition logique toute expression formée à l'aide des variables déclarées dans cette unité et dont le résultat est homogène à un scalaire :

Syntaxiquement on aura :

<CONDITION LOGIQUE> ::= <EXPRESSION>

Ce n'est pas au niveau de la syntaxe que l'on pourra vérifier que l'expression est bien réductible à un scalaire.

EXEMPLE : Registre A(1 : 8) ;

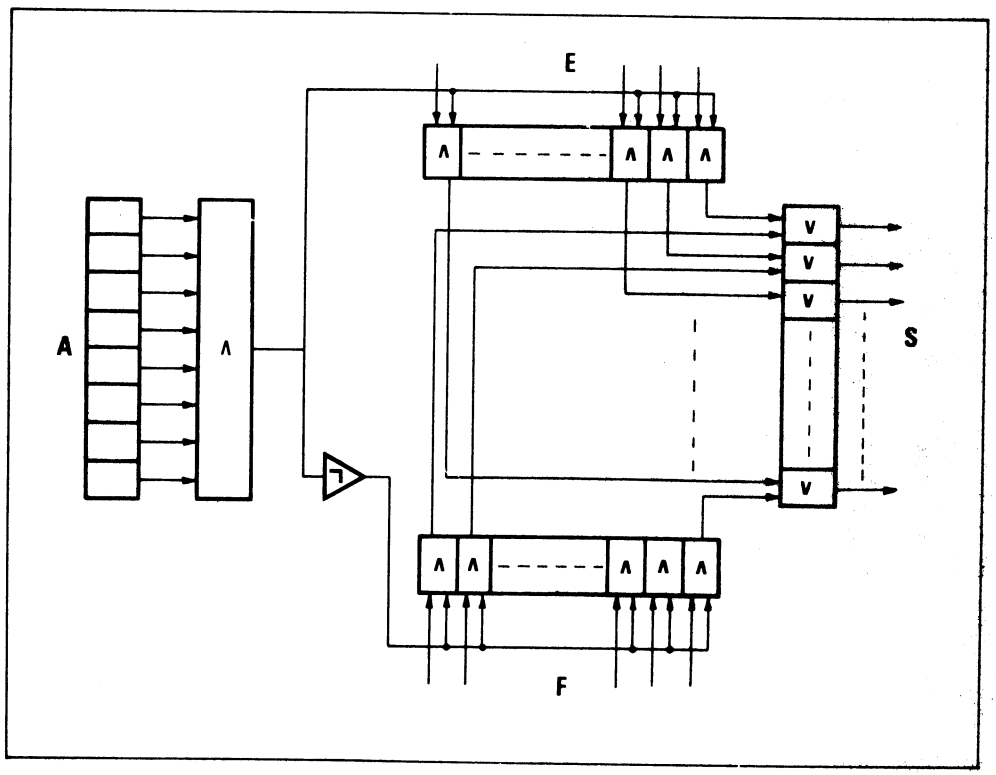
Λ/A = A(1) Λ A(2) Λ ...Λ A(8) ;

peut constituer la condition logique "tous les bits de A sont égaux à 1".

Dans ce cas la "phrase" on éxécute l'opération O1 si tous les bits de A sont égaux à 1 sinon on exécute O2" s'écrit en CASSANDRE :

Si Λ/A alors O1 sinon O2 ;

Avec par exemple O1 ≡ S := E et O_2 ≡ S := F



Un schéma de ce type représenterait l'opération conditionnelle élémentaire.

b) Instruction élémentaire :

Dans tous sytème logique, on ne peut pas en général fixer un ordre aux actions qui s'éxécutent, comme dans un programme. A un instant donné et sous un ensemble donné de conditions logiques plusieurs opérations élémentaires peuvent avoir lieu. On exprime ceci dans la description en faisant une liste de ces opérations, séparées par un symbole qui peut être une virgule.

On appellera alors instruction élémentaire :

$O_1, O_2, O_3, \ldots, O_n$.

EXEMPLE : R := E Λ F, si A alors S := E sinon S := F, faire X ;

Les trois opérations listées ci-dessus, s'éxécuteront en parallèle.

Il n'y a pas d'ordre à priori des actions, l'état interne, c'est-à-dire la valeur de tous ses éléments de mémorisation, fixe seul à un instant les actions en cours : ce sont celles conditionnées par des expressions logiques vraies dans l'état interne considéré. Si l'unité a un registre d'état son contenu est décodé, et le résultat de ce décodage est lui-même une condition logique.

Il faut une vue globale du système à chaque instant, il faut donc un examen complet de la description CASSANDRE à chaque pas du processus, c'est l'une des principales difficultés de la simulation.

- Nous allons faire une forte restriction sur l'instruction élémentaire : il y a deux catégories d'opérations élémentaires : celles qui dépendent d'une impulsion de synchronisation et les autres.

Nous noterons A_1, A_2 ..., A_i (A comme affectation) les premières affectations de registre, d'état et allera, nous noterons B_1, B_2, ... B_j (B comme branchement) les autres connexions de signal, d'unité. L'ordre faire et l'opération <= pourront suivant les cas se noter A ou B.

"Une instruction élémentaire est une liste d'opérations du type A ou une liste d'opérations du type B".

c) Instruction conditionnelle :

Dans un système logique une condition agit en général sur plusieurs actions simultanées et non pas sur une seule.

Ceci peut se décrire par l'instruction conditionnelle dont la syntaxe est :

- <INSTRUCTION CONDITIONNELLE> ::= si <CONDITION LOGIQUE>
  alors <INSTRUCTION INCONDITIONNELLE>|
si <CONDITION LOGIQUE> alors <INSTRUCTION INCONDITIONNELLE> sinon <INSTRUCTION INCONDITIONNELLE>

- <CONDITION LOGIQUE> ::= <EXPRESSION>

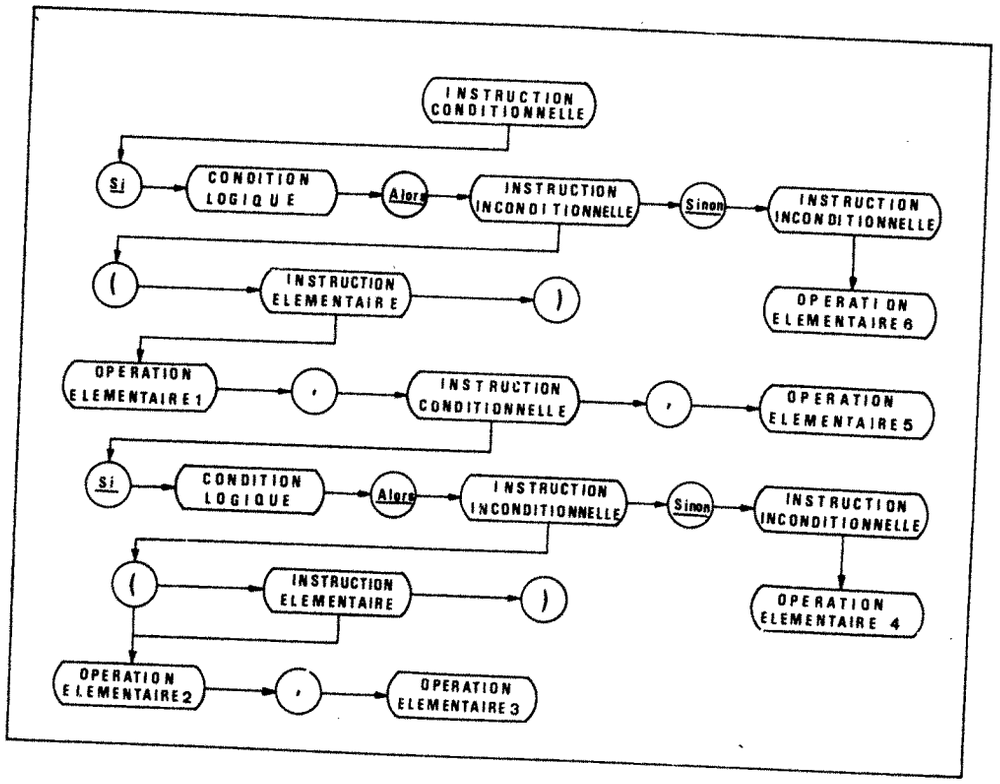
Comme pour les opérations élémentaires, nous rangerons les instructions conditionnelles dans la catégorie A ou B suivant qu'elles portent sur des opérations de la catégorie A ou B et nous généralisons la définition de l'instruction élémentaire en permettant à la liste qui la constitue de contenir des instructions conditionnelles.

Nous définirons alors <INSTRUCTION INCONDITIONNELLE> comme soit une opération élémentaire, soit une instruction élémentaire entre parenthèses.

Cette dernière proposition établit une récursivité croisée entre <INSTRUCTION ELEMENTAIRE> et <INSTRUCTION CONDITIONNELLE>. Chacune pouvant contenir l'autre à des niveaux différents.

EXEMPLE Si A = B alors (R ← G, si E alors (H ← G ∧ B, allera ETA 6)
sinon allera ETA 8, F ← Z ≠ Y) sinon allera ETA 1 ;

L'analyse syntaxique de la proposition précédente, qui décrit en CASSANDRE l'exécution sous diverses conditions (A = B, E, ⌐E, ⌐(A = B)) des actions R ← G, H ← G ∧ B, allera ETA 6, allera ETA 6, F ← Z ≠ Y, allera ETA 1, que nous numéroterons de 1 à 6, donne la structure :



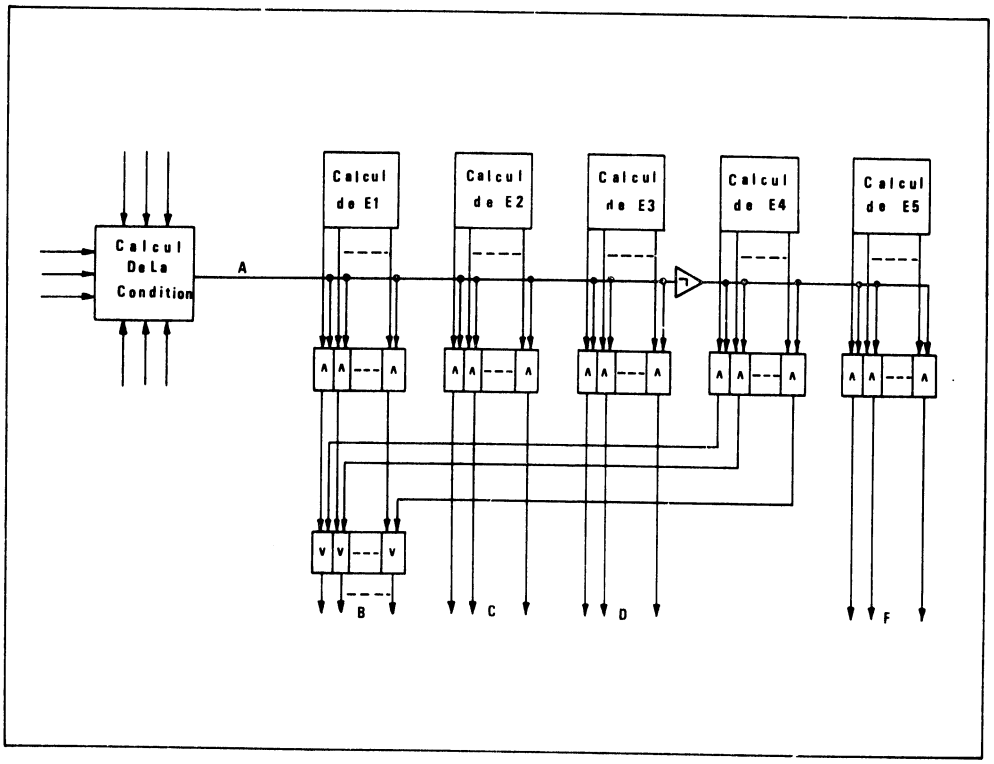
Il n'y a aucune limitation sur le nombre d'éléments de la liste qui constitue une instruction élémentaire.

Il n'y a aucune limitation sur le nombre de niveau d'imbrications dans les instructions conditionnelles.

Ces définitions permettent donc de décrire la structure logique la plus complèxe que l'on veut, elles définissent une hiérarchie naturelle des conditions.

Chaque condition logique ayant si l'on veut une portée maximum, la réalisation qui découlera de cette description aura déjà une certaine optimisation.

EXEMPLE : Si A alors (B := E_1, C := E_2, D := E_3)
sinon (B := E_4, F := E_5) ;



La condition A, qui peut être complèxe n'est calculée qu'une fois, on distribue ensuite le résultat.

Si l'on était parti par exemple des équations booléennes de B,C,D et F, il se peut que l'on ait alors calculé A plusieurs fois.

d) Instruction de connexion, instruction d'affectation, etat :

D'après les définitions précédentes nous voyons qu'au niveau le plus élevé se trouve soit une instruction élémentaire, soit une instruction conditionnelle.

Nous poserons :

<INSTRUCTION DE CONNECTION> ::= <INSTRUCTION ELEMENTAIRE DE type B>|
<INSTRUCTION CONDITIONNELLE de type B>

C'est seulement après la reconnaissance des opérations élémentaires que l'on saura s'il s'agit bien du type B donc de connexion.

Nous appellerons maintenant instruction d'affectation un ensemble d'opérations conditionnées par une impulsion et nous poserons :

<INSTRUCTION-D'AFFECTATION> ::=
<<IMPULSION>><INSTRUCTION-ELEMENTAIRE-de type A>|
<<IMPULSION>><INSTRUCTION-CONDITIONNELLE-de type A>

EXEMPLE : G := E, si A alors F := B sinon F := C, K := L ;
<H> si A alors $R_1 \leftarrow E \wedge F$ sinon $R_1 \leftarrow L$, allera ETA 36 ;

Nous avons vu qu'un état interne d'une unité est repéré par une ETIQUETTE.

Dans cet état le système peut éxécuter des opérations de type affectation (sous une ou différentes impulsions). Un certain nombre de connexions peuvent être également conditionnées par cet état.

Donc par définition :
<ETAT>:= <ETIQUETTE> : <LISTE D'INSTRUCTIONS>

<LISTE D'INSTRUCTIONS> est un ensemble d'INSTRUCTION D'AFFECTATION et d'INSTRUCTION DE CONNEXION séparées par un symbole qui peut être ";".

EXEMPLE : (tiré de | |

```
        Registre A |0 : 1|, B|0 : 1|, C|-1 : 1| ;
        Signal U1, V1, W1, S1, S2 ;
        Impulsion H ;

        S1 := U1 ≠ V1 ≠ W1 ;
        S2 := (U1 Λ V1)V(U1 Λ W1)V(V1 Λ W1);
ETA 1 : U1 := A |1|, V1 := B |1|, W1 := 0 ;
 <H>  C |1| ← S1, C |-1| ← S2, ALLERA ETA 2 ;
ETA 2 : U1 := A |0|, V1 := B |0|, W1 := C |-1| ;
 <H>  C |0| ← S1, C |-1| ← S2, ALLERA SUITE ;
```

REMARQUES :

Toutes les opérations d'une instruction pouvant être simultanées, il est très important de vérifier qu'elles sont bien toutes compatibles deux à deux. D'où la règle :

> Dans une instruction, deux affectations ou connexions différentes de la même variable doivent s'effectuer sous des conditions disjointes.
>
> Il en est de même pour 2 ordres allera

EXEMPLE : Si A alors (B ← C, E ← F), si G alors U ← T sinon (V ← S, B ← Q);
Il faut pour que ceci ait un sens que l'on ait :

$$G \Lambda A \equiv 0.$$

e) Généralisation de l'instruction-conditionnelle :

Une condition logique est une variable booléenne scalaire, mais tout un ensemble de conditions logiques peuvent être résumées dans un tableau booléen. Chaque composante du tableau conditionnera alors un ensemble d'actions.

Tableaux-d'instructions-élémentaires.

La syntaxe d'un tel tableau sera la même que celle d'une constante booléenne, mais chaque composante du tableau sera au lieu de 1, , ou 0, une liste d'opérations entre parenthèses constituant une instruction élémentaire. Une seule opération ou la chaîne vide peut aussi se trouver entre parenthèses pour constituer une composante du tableau.

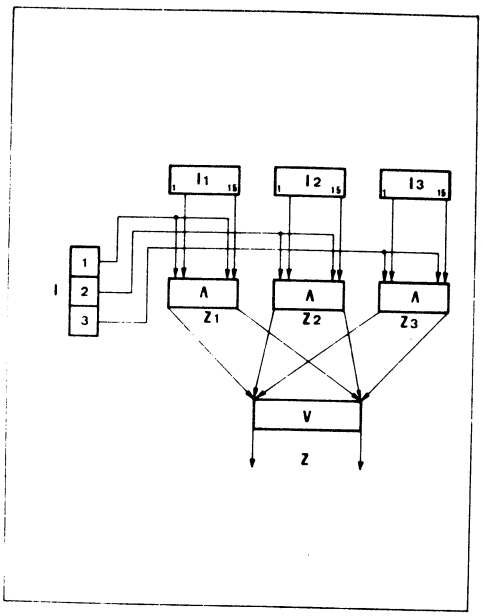
On a alors :

<INSTRUCTION-CONDITIONNELLE> ::= si <EXPRESSION> alors
<TABLEAU-D'INSTRUCTIONS-ELEMENTAIRES>| si <EXPRESSION> alors
<TABLEAU-D'INSTRUCTIONS-ELEMENTAIRES> sinon <TABLEAU-D'INSTRUCTIONS-ELEMENTAIRES>
<TABLEAU-D'INSTRUCTION-ELEMENTAIRE> ::= (<INSTRUCTION-ELEMENTAIRES>)
| (<LISTE-DE-VECTEUR-D'INSTRUCTION>)|<VECTEUR-D'INSTRUCTION>
<LISTE-DE-VECTEUR-D'INSTRUCTION>;<VECTEUR-D'INSTRUCTION>
<VECTEUR-D'INSTRUCTION> ::= (<INSTRUCTION-SCALAIRE>)
|(<INSTRUCTION-SCALAIRE>)<VECTEUR D'INSTRUCTION>
<INSTRUCTION-SCALAIRE> ::= <CHAINE-VIDE>|<INSTRUCTION-ELEMENTAIRE>

Après cette définition il convient de vérifier une autre forme de compatibilité : les tableaux correspondants aux conditions logiques et aux instructions situées soit après alors soit après sinon doivent être compatibles.

Pour celà, il faudra compléter les tableaux d'instructions par des chaînes vides entre parenthèses s'il n'y a pas d'instructions relatives à une condition donnée.

EXEMPLE : Soient I(1 : 3, I_1(1 : 15), I_2 (1 : 15), I_3 (1 : 15), 4 registres.



Le schéma ci-dessus peut se décrire :

si I alors (Z_1 := I_1) (Z_2 := I_2) (Z_3 := I_3),

$Z := Z_1 \vee Z_2 \vee Z_3$.

Le résultat Z est égal à l'union logique du contenu des registres sélectés par les bits de I.

Dans le 7044 IBM, si I_1, I_2, I_3 sont des registres d'index et I le champ d'indexage d'une instruction, l'instruction conditionnelle généralisée ci-dessus décrit l'adressage indirect.

- Si l'on avait une forme :

si I alors (Z_1 := I_1) ( ) (Z_3 := I_3) sinon ( ) (Z_2 := I_2) ( ) ;

Cela signifierait que Z_2 est sélecté par la présente d'un 0 dans I(2) au lieu de la présence d'un 1.

f) Connexion d'Unité sous condition signal-unité :

Lorsqu'une connexion d'unité apparaît à l'intérieur d'une instruction conditionnelle toutes les entrées et toutes les sorties présentes dans l'unité sont connectées sous condition. Si l'unité est purement combinatoire il peut y avoir redondance et gaspillage.

EXEMPLE : Unité U(A,B,C ; S1, S2) ;

S_1 := A ≠ B ≠ C ;

S_2 :=(A ∧ B)∨(B ∧ C)∨(A ∧ C);

---

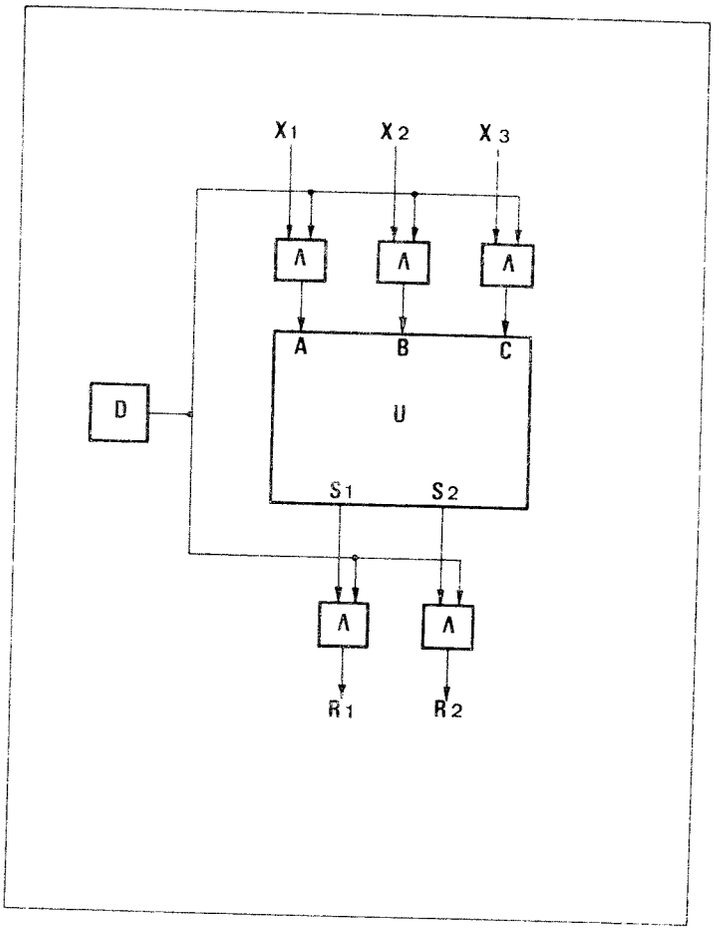
si D alors U (X_1, X_2, X_3 ; R_1, R_2) ;

donnerait un résultat analogue au schéma ci-dessous

Il vaut mieux écrire alors : U (X_1, X_2, X_3 ; ,),

si D alors U (,, ; R_1, R_2) ;

Les zones non connectées sont simplement laissées en blanc.



Illustrons l'utilisation du signal-unité par un exemple :

EXEMPLE :

<H> si D alors Y ← Z U (X_1, X_2, X_3 ; *, R_2) ;

Ceci est strictement équivalent à :

<H> si D alors (U(X_1, X_2, X_3 ; R_1, R_2), Y ← Z ∧ R_1);

Ce qui est absurde car jamais une connexion d'unité ne doit se trouver sous la portée d'une impulsion.

On pourrait alors écrire :

si D alors U (X_1, X_2, X_3 ; R_1, R_2) ;
<H> si D alors Y ← Z ∧ R_1 ;

Mais la condition D peut être compliquée, au lieu d'une seule condition D on peut se trouver à l'intérieur d'un grand nombre de si imbriqués, réécrire celà une deuxième fois alourdi inutilement le texte.

On peut alors s'en tirer en écrivant :

si D alors R_1 := Z ∧ U (X_1, X_2, X_3 ; *, R_2) ;

<H> Y ← R_1 ;

On voit que l'utilisation judicieuse de signaux intermédiaires et du signal unité permet d'alléger sensiblement l'écriture et la réalisation.

D - NIVEAU AUTOMATE

Nous avons vu en partie II A la définition d'une unité sous l'aspect structurel et statique.

Le système étudié est un ensemble d'unité interconnectées, imbriquées et juxtaposées.

Nous avons maintenant tous les éléments pour définir l'unité sous l'aspect fonctionnel et séquentiel. Chaque unité est un automate séquentiel ayant une partie contrôle et une variable interne qui mémorise l'état dans lequel l'automate se trouve. Dans certaines réalisations cette variable interne peut être celle d'une unité plus grande qui la contient, la partie contrôle est alors extérieure à l unité, seuls y entrent les fils d'activation de chaque état, qui sont des sorties du décodeur d'état de l'unité principale, (dans ce cas nous l'avons déjà dit, ces entrées sont implicites, puisqu'à priori on ne connaît pas leur nombre).

a) Liste d'Etats

Nous avons défini un Etat à la fin du chapitre précédent. Il est constitué d'une étiquette, qui repère une valeur de la variable interne avant codage, suivie d'une liste d'instructions. Ces instructions sont commandées par une série de signaux de contrôle, envoyés par l'automate de contrôle de l'unité lorsque la valeur de la variable interne est celle repérée par l'étiquette.

La partie fonctionnelle d'une unité est constituée par une liste d'états qui commencent tous par une étiquette.

EXEMPLE :

```
Unité ADDITIONNEUR (H,M(1 : n) ; R(1 : n+1)) ;
Registre D, OF, L ADD, N (1 : n) , A(1 : n) ;
ETA 1 :  <H> N ← M, D ← 0, OF ← 0, → ;
ETA 2 :  <H> A ← N ≠ A, OF & N ← 1 ↑ N Λ A, D ← V/N Λ A, → ;
ETA 3 :  <H> si D = 0 alors (    L ← OF, ADD ← 0, → )
             sinon (D ← 0, allera ETA 2) ;
ETA 4 :  <H> si ADD = 0 alors allera ETA 4 sinon (L ← 0, allera ETA 1);
```

Dans cette unité il y a une liste de quatre états, l'automate de contrôle d'ADDITIONNEUR aura donc lui aussi quatre états repérés par les étiquettes ETA 1, ETA 2, ETA 3, ETA 4.

Les transitions entre états sont marquées par des ordres allera souvent conditionnels.

Il est toujours possible de calculer la condition totale qui s'applique à chaque allera, il suffit de cumuler les conditions logiques de toutes les instructions-conditionnelles qui le contienne.

Nous désignerons désormais par des lettres grecques ces conditions (α, β, ω).

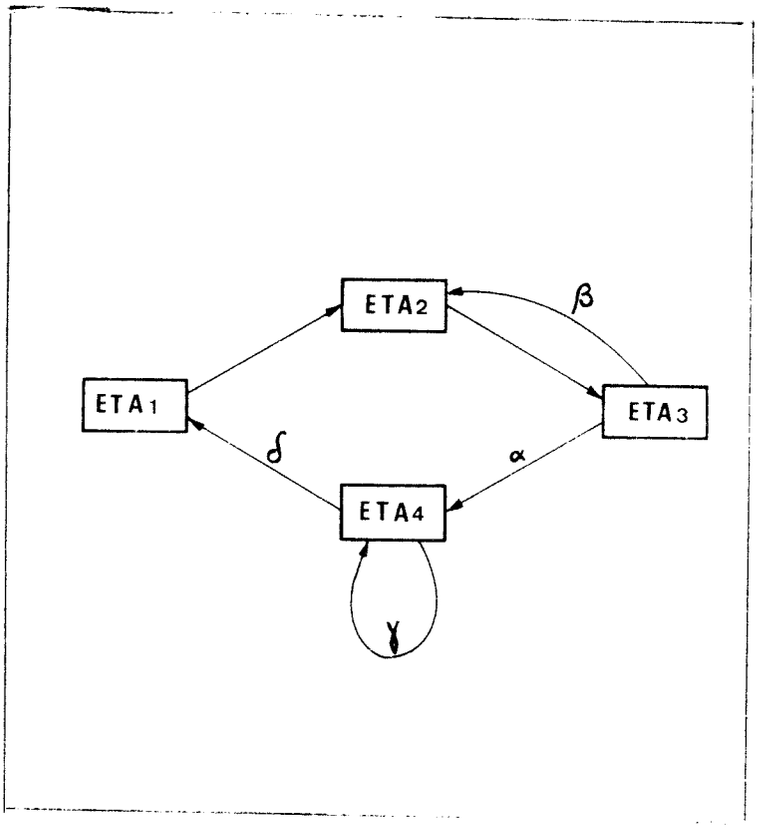
b) Graphe associé à une unité :

Il est immédiat de faire correspondre à chaque unité un graphe dont les noeuds sont constitués par toutes les étiquettes de l'unité et les branches par tous les ordres allera avec leur condition associée.

EXEMPLE PRECEDENT

(D = 0) ≡ α (ADD = 0) ≡ γ

(D ≠ 0) ≡ β (ADD ≠ 0) ≡ δ



Pour le codage et la réalisation de l'automate de contrôle ainsi défini voir Monsieur GERACE [34].

c) Sous automate :

En CASSANDRE, une étiquette contrôle une liste d'instruction nous l'avons vu. Elle peut aussi contrôler une liste d'état.

EXEMPLE :

ETA 1 : G_1 : C_1 : B_{11}, B_{12}, ..., B_{1i1} ;
<H 1> A_{11}, A_{12}, ..., A_{1i2} ;
C_2 : B_{21}, B_{22}, ..., B_{2i3} ;
<H 2> A_{21}, A_{22}, ..., B_{2i4} ;
G_2 : D_1 : ---------- ;
<H 1> ---------- ;
D_2 : ---------- ;
<H 1> ----------;
ETA 2 : K_1 : E_1 : ---------- ;
<H 2> ---------- ;
E_2 : ---------- ;
<H 2> ---------- ;
K_2 : F_1 : ---------- ;
<H 1> ---------- ;
F_2 : B_{81}, B_{82}, ..., B_{8i8} ;
<H 1> A_{81}, A_{82}, ..., A_{8i9} ;
<H 2> A_{91}, A_{92}, ..., A_{9i10} ;

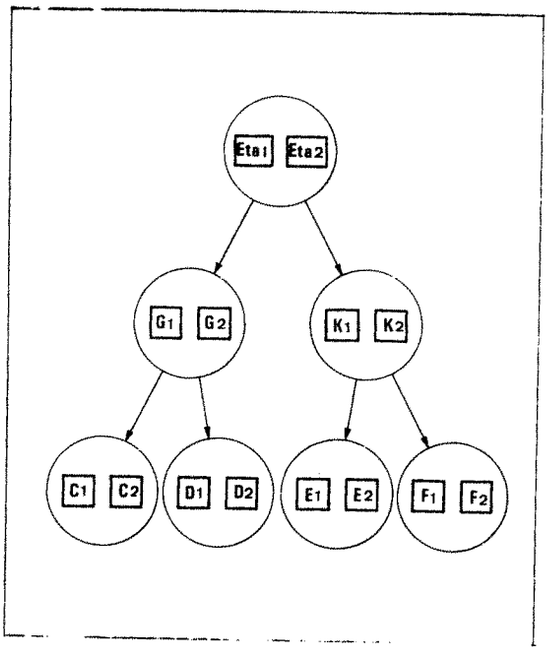
cet exemple est purement formel, mais possible.

On voit que F_2 par exemple contrôle trois instructions l'une de type B deux autres de type A avec deux horloges H_1 et H_2.

K_2 contrôle deux états, F_1 et F_2 de même, ETA 2 contrôle K_1 et K_2.

Chaque étiquette qui contrôle une liste d'états définis un sous-automate de l'automate de contrôle de l'unité.

On peut très simplement définir une structure hiérarchique d'automates qui réalisent le contrôle de l'unité.



La portée de chaque étiquette dans la description est clairement définie, elle s'étend jusqu'à l'étiquette suivante de même niveau.

La notion de sous-automate a déjà été étudiée par Monsieur DESCHIZEAUX. Si elle a été conservée sous la forme que nous venons de voir, nous n'avons cependant pas gardé la contrainte qu'il fixait de toujours entrer dans un bloc par le même point.

Les transitions entre états se font par des ordres allera <ETATCOMPOSE> au lieu d'une étiquette simple après allera on peut en trouver plusieurs reliées par des :.

EXEMPLE PRECEDENT : Dans D_2 on peut trouver allera ETA 2 : K_1 : E_2 ;

Dans le cas où l'on reste dans un même bloc une seule étiquette suffit.

EXEMPLE : Dans D_2 on peut trouver allera D_1 ;

Avec cette définition toutes les transitions entre deux états quelconques sont permises.

d) Ordre faire, séquences simultanées ;

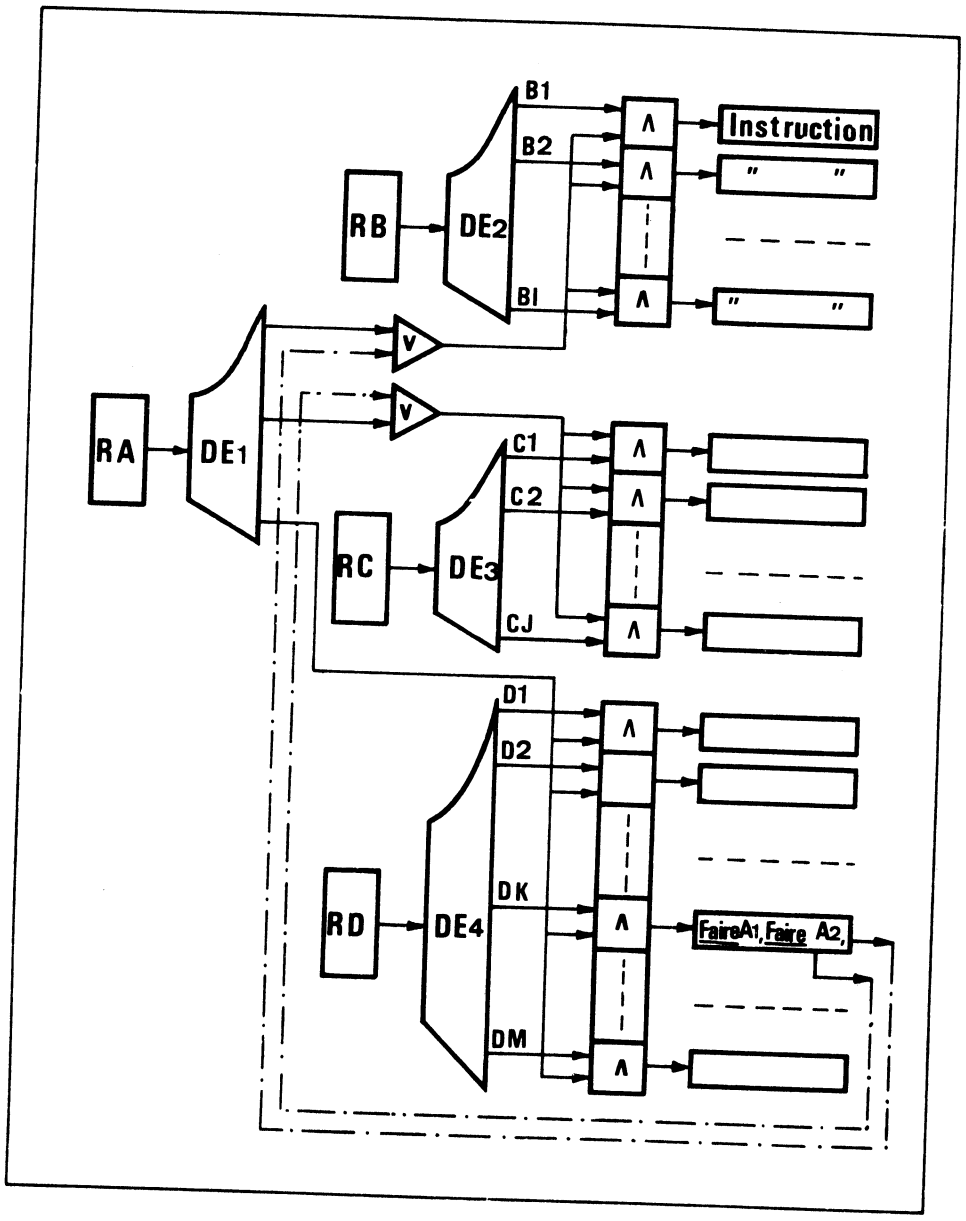
L'ordre faire associé à la notion de sous-automate permet de décrire le fonctionnement simultané de deux sous-automates à l'intérieur d'une même unité. Nous allons simplement montrer ceci sur un exemple pour que ce soit compréhensible.

```
Unité ZOZO (                    ) ;
Déclaration {                   } ;
A1 : B1 : <liste d'instructions>  ;
     B2 :     "        "          ;
     ---------------------------- ;
     Bi :     "        "          ;
A2 : C1 :     "        "          ;
     C2 :     "        "          ;
     ---------------------------- ;
     CJ :                        ;
A3 : D1 :                        ;
     D2 :                        ;
     ---------------------------- ;
     DK :                        ;
     ---------------------------- ;
     DM :                        ;
```

L'unité ZOZO a un automate principal de trois états A1, A2, A3 et trois sous-automates ayant respectivement I, J et M états.

On peut schématiser la structure de la partie contrôle avec les éléments suivants :

Quatre regitres RA, RB, RC et RD qui servent à coder les états A,B,C et D et leurs décodeurs associés DE1, DE2, DE3, DE4.



Si l'on admet que jamais deux instructions parmi B1, ... Bj, C1, CJ, D1 ... Dm ne s'éxécutent en même temps, on peut alors coder les états B les états C et les états D des trois sous-automates avec les mêmes variables.

Le schéma est le même que ci-dessus sauf que les trois registres RB, RC et RD sont remplacés par un seul et les trois décodeurs DE2, DE3, DE4, par un seul aussi.

Dans l'hypothèse que nous avons choisie l'un des trois sous-automates peut changer d'état, sans que les deux autres varient.

Supposons que l'automate B soit dans l'état BL et qu'une séquence d'instructions internes à cet automate existe à partir de BL. On passera de chaque état BL1 au suivant BL2 par un ordre allera BL2 qui ne modifie que le contenu de RB.

Faisons la même supposition pour l'automate C à partir d'un état CN.

Si maintenant l'unité ZOZO se trouve dans l'état A3 : DK et que l'on trouve dans cet état une instruction comme :

si X alors allera Z sinon (faire A_1, faire A_2) ;

Tant que la condition X n'est pas vérifiée ou reste dans le même état et les contenus de RA et RD sont inchangés. Pendant le même temps les signaux faire A_1 et faire A_2 sont activés et pour B et C tout se passe comme si A était dans l'état A_1 et A_2 respectivement ; (définition déjà vue de faire).

Comme B et C étaient dans l'état BL et CN respectivement les deux séquences qui commençaient en ces points se déroulent simultanément jusqu'à ce que l'un de ces évènements arrivent :

1) Les séquences s'arrêtent sur un état dans leur sous-automate, si à ce moment là la condition X n'a pas changé tout est arrêté,

2) La condition X change et alors les signaux faire A1 et faire A2 n'étant plus activés les séquences s'arrêtent, et le sous-automate D reprend en Z,

3) L'une des séquences contient un allera AQ : Y et le contenu de RA n'étant plus A3, les séquences s'arrêtent et un autre sous automate commandé par AQ prend la relève.

- On voit que les définitions précédentes laissent un immense champ de possibilités de fonctionnement d'automates correlés à exploiter.

e) Forçage d'un etat dans une unité

Il est indispensable de préciser les moyens que l'on a de charger de l'extérieur d'une unité dans le registre d'état de cette unité une valeur.

Pour celà, il faut distinguer deux sortes d'unités :

e1) L'unité ne contient pas d'étiquette

Ce peut être, parce qu'elle est purement combinatoire et parce que ses sorties sont des fonctions booléennes instantannées des entrées.

L'unité est alors un ensemble de signaux.

Ce peut être également parce que les états ont été codés et la partie contrôle réalisée.

L'unité contient alors un ou plusieurs éléments de mémorisation pour les valeurs des états et des décodeurs associés. Tous les ordres allera et faire ont disparu et toutes les étiquettes. Le résultat est un circuit séquentiel décrit par des instructions conditionnelles. Les signaux de sortie des décodeurs sont des conditions comme les autres.

Dans ce cas, si certaines entrées de l'unité sont connectés aux éléments de mémorisation de ses états, leur format est alors bien spécifié, ce sont des entrées normales de l'entête, et le forçage d'un état se fait par une simple connexion de l'unité dans une autre (l'état forcé étant calculé comme un signal dans cette autre).

e2) L'unité contient encore des étiquettes des allera et des faire.

Supposons que l'on veuille forcer U1 dans l'état ETA FORCE.

ETA FORCE est une étiquette qui appartient à U1, elle n'a aucun sens à l'extérieur.

On peut alors résoudre le problème en faisant commencer la description de U1 par :

(1) <H> si A alors allera ETA FORCE ;

On place alors le signal-condition logique A en entrée de U1, le forçage est ramené à une connexion de U1 dans une autre unité, au cours de laquelle on attribue à l'entrée A une valeur convenable.

Il faudra prévoir dans la définition de U1 tous les états susceptibles d'être forcés et leur attribuer à chacun une instruction comme (1) et une entrée-condition logique dans l'entête de U1.

f) Conclusions sur l'unité

On peut finalement justifier à postériori les niveaux de description en CASSANDRE précédemment définis.

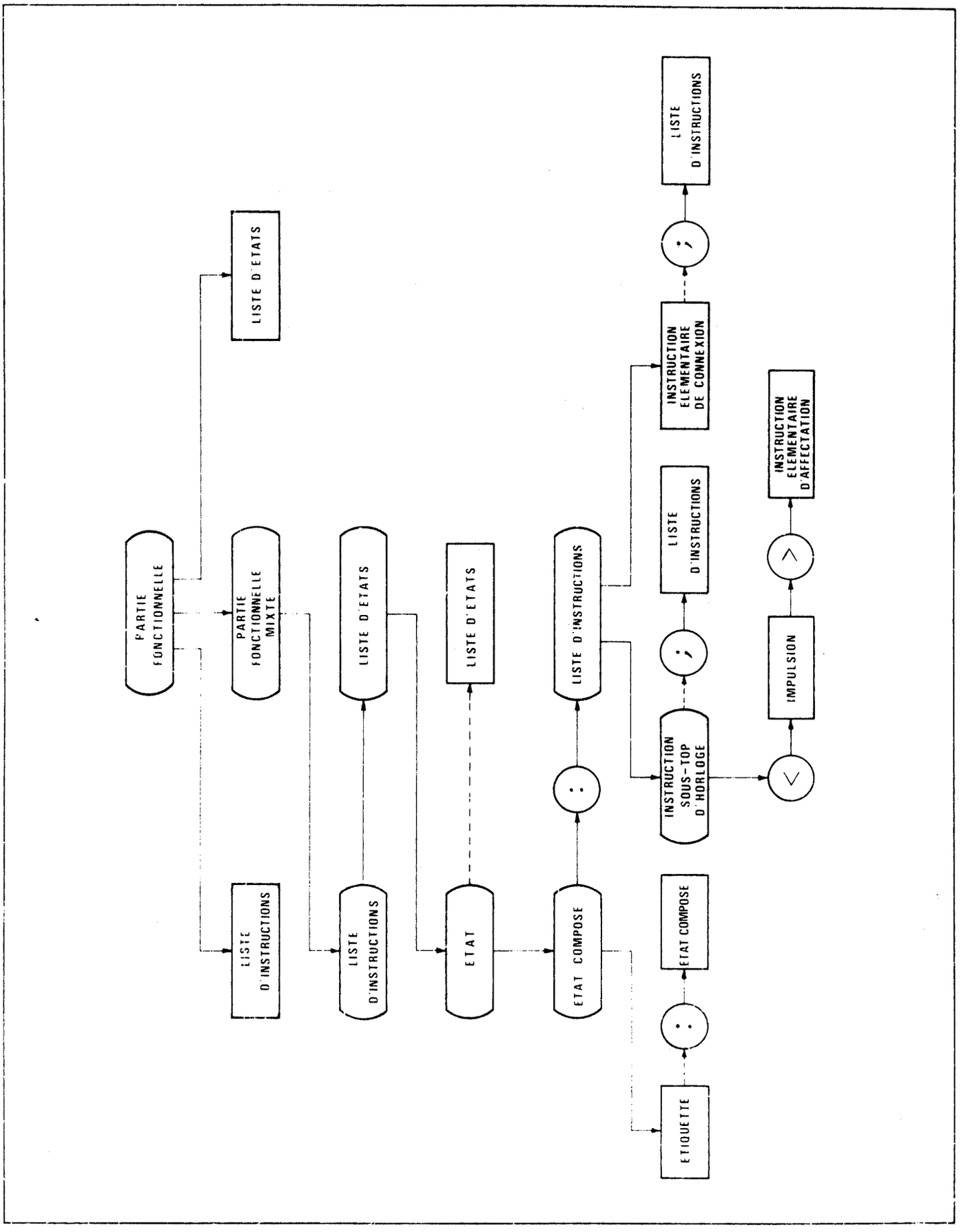
1) Le niveau réseau de boîtes interconnectées est présent dans toutes les descriptions. Il est décrit par les déclarations et les connexions.

2) Le niveau automate se superpose au précédent dans les descriptions les plus abstraites.

3) Le niveau micro instructions s'obtiend après synthèse des automates de contrôle, disparition des faire, des allera, des variables d'état qui deviennent registres, des étiquettes.

4) Le niveau booléen s'obtiend après disparition de toutes les conditions logiques et leur réduction à des expressions.

5) Le niveau circuit logique s'obtiend après synthèse de toutes les expressions à l'aide de modules standard "boîtes noires". Il rejoint le niveau 1 et se réduit à la description d'un réseau.


PARTIE FONCTIONNELLE
PARTIE FONCTIONNELLE MIXTE
LISTE D'ETATS
LISTE D'INSTRUCTIONS
LISTE D'ETATS
LISTE D'INSTRUCTIONS
ETAT
LISTE D'ETATS
ETAT COMPOSE
LISTE D'INSTRUCTIONS
:
INSTRUCTION ELEMENTAIRE DE CONNEXION
;
LISTE D'INSTRUCTIONS
LISTE D'INSTRUCTIONS
;
INSTRUCTION SOUS-TOP D'HORLOGE
ETIQUETTE
:
ETAT COMPOSE
<
IMPULSION
>
INSTRUCTION ELEMENTAIRE D'AFFECTATION

NOTIONS FORMELLES

-:-:-:-:-

A - NOTIONS ARITHMETIQUES MACRO-NOTIONS

La boucle pour en CASSANDRE, les expressions arithmétiques et l'instruction si arithmétique sont dérivées des notions correspondantes d'ALGOL nous n'insisterons donc pas sur ces définitions.

Toutes les notions exposées dans ce chapitre ont un rôle purement formel et ne correspondent à aucune fonction du système logique décrit. (le symbole + par exemple ne signifiera jamais un additionneur, un additionneur correspondrait lui à une unité). Ce sont des facilités d'écriture et des extensions de langage qui augmentent sa puissance et sa concision. Que ce soit notions arithmétiques ou macro-notions toutes peuvent être traitées dans une phase préliminaire par un macrogénérateur qui ne laisse subsister que les notions définies dans les chapitres précédents et qui constituent le langage de base.

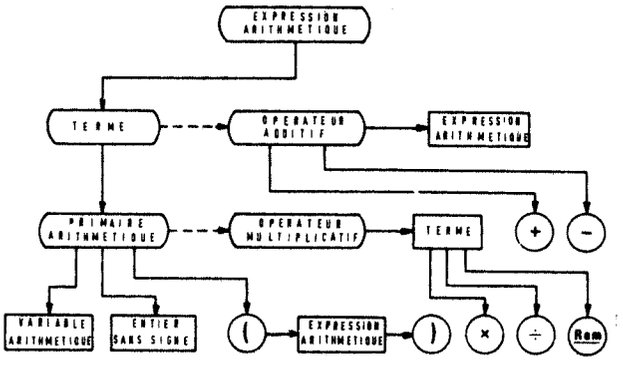
Nous n'aurons peut être pas intérêt à procéder ainsi et au contraire, nous essayerons le plus longtemps possible dans tout traitement de conserver toutes les facilités et la concision de ces notions.
(Voir Monsieur ANCEAU simulation | |, Monsieur LIDDELL sur-découpage | |).

a) Expressions arithmétiques :

En CASSANDRE, les expressions arithmétiques correspondent aux expressions arithmétiques simples d'ALGOL, elles ne portent que sur des quantités entières.

Elles sont obtenue par combinaison de variables arithmétiques, (donc non déclarées en CASSANDRE) et des opérateurs +, -, x, ÷, Rem, avec les règles de priorité d'ALGOL c'est-à-dire deux niveau plus (+), moins (-) et multiplié (x), divisé (quotient entier ÷, Reste modulo la division entière (Rem)).

SYNTAXE DE L'EXPRESSION ARITHMETIQUE



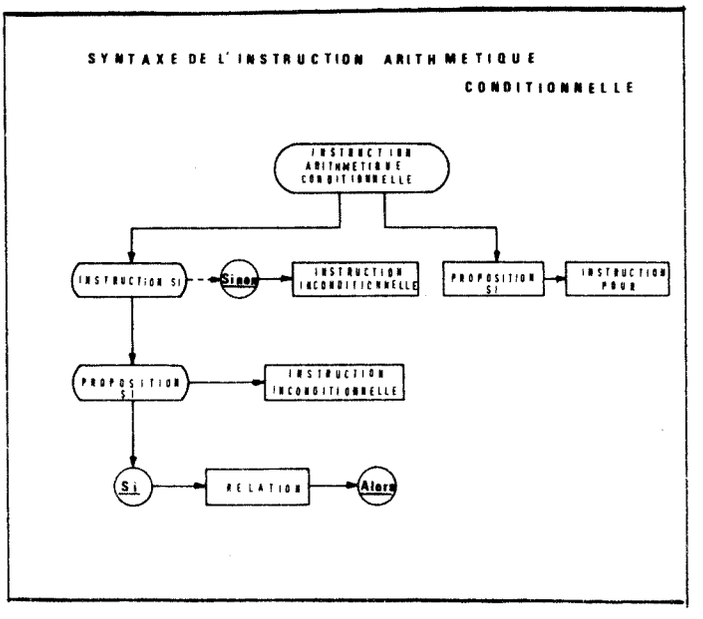
Dans les chapitres précédents, partout où dans la syntaxe apparaît entier on peut remplacer entier par expression-arithmétique qui, en CASSANDRE aura toujours pour résultat un entier.

b) Expressions arithmétique de relation, Instruction Arithmétique conditionnelle.

Les opérateurs de relation sont $<|\leq|=|\geq|>|\neq$ la relation est formée par : <RELATION> ::= <EXPRESSION-ARITHMETIQUE><OPERATEUR-RELATION> <EXPRESSION-ARITHMETIQUE>

La valeur d'une relation est vrai ou faux.

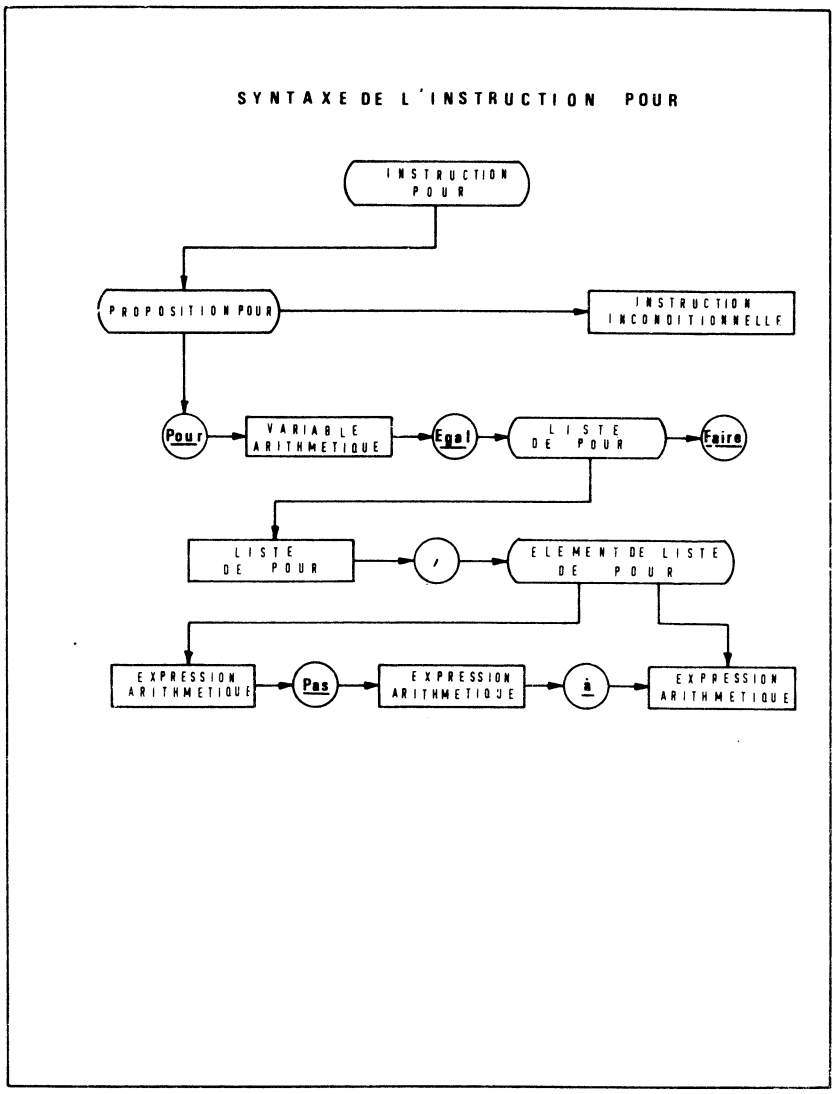
L'instruction arithmétique conditionnelle vérifiera alors après simplification de la syntaxe ALGOL : la carte suivante.



Dans la carte précédente le seul élément de CASSANDRE de base est <INSTRUCTION-INCONDITIONNELLE> dont nous avons vu que c'est soit <OPERATION-ELEMENTAIRE> , soit (<INSTRUCTION-ELEMENTAIRE>).

c) Instruction Pour :

Là encore, nous retrouvons une forme simplifiée d'ALGOL. La carte syntaxique n'a pas besoin d'explication.



La forme tant que ALGOL ne serait d'aucune utilité en CASSANDRE

d) Introduction des notions arithmétiques dans CASSANDRE :

Les notions précédentes permettent de décrire des structures répétitives, ou des ensembles d'opérations, semblables mais combinées de façon différente, avec le minimum de texte.

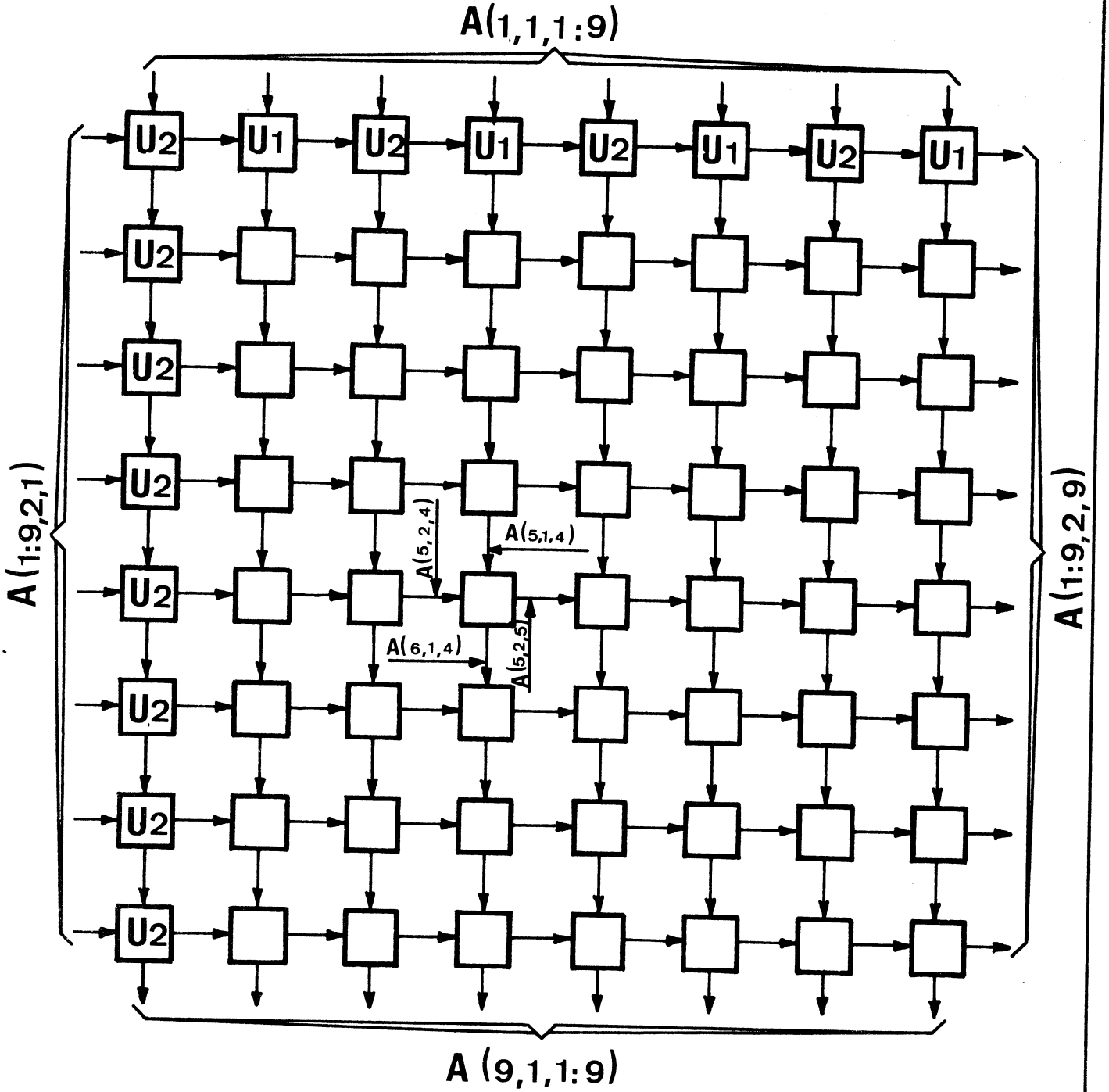
Elles définissent une sorte de "jeu de construction" du texte et n'ont pas de signification propre en hardware.

On introduit dans la syntaxe instruction POUR, et instruction arithmétique conditionnelle au même niveau que l'instruction conditionnelle CASSANDRE.

EXEMPLE :

```
Externe U1 (1,1 ; 1,1), U2 (1,1 ; 1,1) ;
Signal A(1 ; 9, 1 : 2, 1 : 9) ;
Pour I égal 1 pas 1 à 8 faire pour J = 1 pas 1 a 8 faire
(si (2 Rem ((I-1)x 8 + J ) = 0)alors
U1 (A(I,1,J), A(I,2,J); A(I+1,1,J), A(I,2,J+1)) sinon
U2 (A(I,1,J), A(I,2,J); A(I+1,1,J), A(I,2,J+1))) ;
```

On crée une matrice 8 x 8 dont les noeuds sont alternativement U1 et U2 puisque la condition 2Rem ((I-1) x 8+J) est alternativement vrai ou fausse.


A(1,1,1:9)
A(1:9,2,1)
A(1:9,2,9)
A(9,1,1:9)
U2
U1
A(5,2,4)
A(5,1,4)
A(6,1,4)
A(5,2,5)

e) Généralisation de $\lfloor$, Opérateur Δ.

e_1) Nous avons vu que $\lfloor$ signifie un décodeur qui à tout nombre binaire fait correspondre un signal permettant de sélecter une sous-variable d'une variable déclarée, dont l'indice est égale à l'"adresse" constituée par le nombre binaire décodé.

Le décodeur réel qui sera utilisé dans le hardware de la machine dépend de la signification donnée au mot binaire décodé :

Ce peut être un nombre octal, décimal ou hexadécimal dans ces cas le symbole $\lfloor$ seul convient, mais ce peut être aussi bien un nombre décimal codé binaire, ou un code de Hamming ... et il faudra faire précéder $\lfloor$ d'un autre symbole qui donne cette indication, car le décodeur résultant sera différent.

EXEMPLE :

```
Registre R(1 : 16), MEM(0 : 65 535, 1 : 16), AD(1 : 20) ;
<H> R ← MEM (DcB ⌊ AD, ) ;
```

e_2) Opérateur Δ : Nous noterons ainsi un opérateur purement formel qui a pour effet d'écrire un nombre binaire selon un code spécifié devant Δ à partir d'un nombre écrit en décimal et traduit simplement en binaire on ne mettra rien devant Δ, sinon on mettra le nombre correspondant à la base de numération utilisée, ou un symbole indiquant le code souhaité.

EXEMPLE :

1) Δ 77 produira (100 1101 qui est égal en binaire au nombre qui s'écrit 77 en décimal ;

2) 8 Δ 77 produira (111 111) qui est égal en binaire au nombre qui s'écrit en octal 77.

3) 16 Δ 77 produira (0 111 0 111) nombre binaire correspondant à 77 en hexadécimal ;

4) 16 Δ 1 E 39 produira (0001 1111 0 00111001)

5) Dcb Δ 1938 produira (0001 1001 0011 1000)

L'opérateur Δ apparaît comme l'inverse de l'opérateur ⊥, mais, alors qu'un décodeur correspond à ce dernier, il ne correspond rien à Δ.

C'est uniquement une facilité d'écriture pour écrire sous une forme plus claire et plus concise des constantes booléennes vectorielles.

c) Transcodage ; La description d'un transcodeur se fera par l'usage des deux opérateurs précédents.

EXEMPLE :

Ecrire dans B en binaire le nombre qui se trouve dans A en décimal codé binaire :

<H> B ← Δ(Dcb ⊥ A) ;

Dans cet usage on peut considérer que Δ n'introduit toujours aucun matériel, mais modifie le sens de ⊥.

f) Macro constantes, Macro traitement :

f_1) ε 0(a,b,c,...,i) est une macro constante qui est égale à une constante de type tableau et de dimensions a,b,c,...,i, dont toutes les composantes sont égales à 0.

ε 1(a,b,c,...i) et ε ϖ(a,b,c...i)

ont la même définition que ε 0 mais toutes leurs composantes sont égales respectivement à 1 et à ϖ .

EXEMPLE :

ε 0 (3,4) ≡ (0000,00000000)

Nous avons déjà vu chapitre |III A| un exemple d'utilisation de ces constantes.

f_2) L'instruction pour a une portée limitée puisqu'elle peut en CASSANDRE s'appliquer seulement à une instruction-inconditionnelle.

S'il s'avère utile de recopier, en faisant évoluer une variable arithmétique à chaque pas, une partie quelconque de texte. On pourra alors définir une macro instruction de recopie dont la forme pourrait être ;

§ Pour <VARIABLE-ARITHMETIQUE> égal <EXPRESSION-ARITHMETIQUE> pas <EXPRESSION-ARITHMETIQUE> à <EXPRESSION-ARITHMETIQUE>

Recopier |<TEXTE>|.

<TEXTE> représente un texte en CASSANDRE séparé au début et à la fin par |et| insérés entre deux symboles de base.

Bien entendu, ce genre de traitement devra se faire avant tout autre, car le résultat ne sera pas forcément du CASSANDRE syntaxiquement exact.

L'analyse syntaxique suivra donc.

Il s'agit bien d'un macro traitement.

EXEMPLE :

```
§ Pour I égal 3 pas 1 à 7 Recopier |
  ETA I : A (I) := B(I) ;
          <H> C (I) ← D (I), → ;
```

produira :

```
ETA 3 : A(3) := B(3) ;
        <H> C(3) ← D(3), → ;
ETA 4 : A(4) := B(4) ;
        <H> C(4) ← D(4), → ;
ETA 5 : A(5) := B(5) ;
        <H> C(5) ← D(5), → ;
ETA 6 : A(6) := B(6) ;
        <H> C(6) ← D(6), → ;
ETA 7 : A(7) := B (7) ;
        <H> C(7) ← D(7), → ;
```

f_3) Expressions régulières, opérateur Dès que :

On peut en CASSANDRE écrire des expressions régulières à l'aide des variables, des opérateurs logiques et des opérateurs & (concaténation) et * (itération).

Monsieur PAYAN [78] à introduit un nouvel opérateur Dès-que qui permet de dérinir des expressions régulières plus commodément.

On peut ainsi décrire en CASSANDRE une classe beaucoup plus grande d'automates.

Supposons en effet, qu'un automate soit connu par quelques états principaux, dans lesquels il effectue des actions connues, les autres états sont ignorés, on peut juste définir les transitions entre les états principaux par des séquences d'entrées-sorties décrites sous forme d'expressions définies par Monsieur PAYAN.

A partir de cette description, un algorithme [78] permet de trouver un automate qui rempli les fonctions souhaitées, et qui est alors décrit en CASSANDRE de base, car on connait entièrement son graphe.

Ce traitement préliminaire s'apparente bien à un macro traitement.

Les actions qui se produisent après une séquence d'entrées-sortie seront décrites comme dans une instruction-conditionnelle :

si <EXPRESSION-PAYAN> alors <INSTRUCTION-INCONDITIONNELLE>.

ou si <EXPRESSION-PAYAN> alors <INSTRUCTION-INCONDITIONNELLE> sinon <INSTRUCTION-INCONDITIONNELLE>.

B - DESCRIPTION FORMELLE DE LA SYNTAXE

a) Opérateur liste

- Soit Q un ensemble fini ou dénombrable :

$$\{q_1, q_2, \ldots q_n \ldots\}$$

- q_i peut être par exemple une fonction booléenne, une instruction arithmétique, une affectation de registre, une connexion de signal, un entier un scalaire booléen {0,1, }... .

- Soit S un ensemble fini d'éléments que nous appellerons séparateurs $\{s_1 s_2, \ldots s_n\}$

- s_j peut être par exemple : |,;,(,),I,I,{,},←,+,.,-,V,Λ,V,Λ..

- Soit $L(Q,S)$ un ensemble d'éléments L_i^j que nous appellerons "espace liste" sur Q et S.

L est défini par les règles suivantes :

1) $\forall \mid \in S$, ${}_kL_n^1 \in L$, $\exists \{q_1, q_2, \ldots q_n\} \in Q$

tels que ${}_kL_n^1 = q_1 | q_2 | q_3 \ldots | q_n$

2) $\forall, \in S$ et ${}_kL_p^j \in L \mid \{L_{n1}^{j1}, L_{n2}^{j2}, \ldots L_{np}^{jp}\} \in L$

tels que ${}_kL_p^j = L_{n1}^{j1}, L_{n1}^{j1}, L_{n2}^{j2}, L_{n3}^{j3}, \ldots, L_{np}^{jp}$

et $\quad Max \{j_1, j_2, j_3, \ldots, j_p\} = j-1$

REMARQUES

1) Les indices i et j de ${}_kL_i^j$ n'ont pas pour but de numéroter l'élément $L \in \mathcal{L}$, mais représentent :

j : le nombre de séparateurs distincts utilisés

i : Le nombre d'éléments de $\mathcal{L}_i$ quelconque, cela n'a de sens que pour $\mathcal{Q}$ fini.

2) D'après les règles de récurrence qui définissent $\mathcal{L}$ une liste de liste est une liste.

3) Il est évident que ${}_kL_i^o \in \mathcal{Q}$.

4) Nous appellerons vecteur à n composantes tout élément ${}_kL_n^1$

EXEMPLE :

$\mathcal{Q} = \{0,1,\varphi\}$ $\mathcal{S} = \{.\}$

1.0.φ.0.0.1.1.φ.0.0. est un vecteur de type L_{10}^1, il est booléen.

5) Nous appellerons tenseur à n dimensions tout élément ${}_kL_n^2$.

EXEMPLE :

$\mathcal{Q} = \{0,1,\varphi\}, \mathcal{S} = \{. \ ,\}$

1.0.φ.0.0.1,1.1.10.φ.0.0.1.1,1.1.0,φ.φ.φ.1.1.0

est de la forme : ${}_kL_4^2 = k_1 \ L_6^1, \ k_2L_q^1, k_3L_3^1, k_4L_6^1$

6) Un tableau à n dimensions $p_1, p_2, \ldots, p_n$ est un cas particulier de tenseur à n dimensions :

${}_kL_n^2 = {}_{k^1}L_{p1}^1, \ {}_{k^2}L_{p1x(p2-1)}^1, \ {}_{k3}L_{p1xp2x(p3-1)}^1, \ldots, \ L_{p1xp2x\ldots x(pn-1)}^1$

EXEMPLE :

1.0.1.φ,1.1.1.1.0.0.0.0.1.1.1.1,

1.0.1.0.φ.0.φ.0.1.φ.1.φ.φ.φ.φ.φ.

est un tableau à 3 dimensions

p1 = 4, p2 = 4, p3 = 2

7) Dans les 2 derniers cas le blanc est souvent utilisé comme 1er séparateur, ceci ne doit pas faire perdre de vue la nécessité d'un tel séparateur.

EXEMPLE :

010 100 111 001 sont les composantes d'un vecteur ${}_kL_4^1$ = 010.100.111.001 ou peuvent être les sous-listes d'un tenseur :

ℓL_4^2 = 010,100,111,001

8) Nous appellerons <u>couple</u> tout élément ${}_kL_2^j$

EXEMPLE :

a) Ω = {entiers relatifs} , δ = { : }

${}_kL_2^1$ = -13 : 18

est un couple d'indices délimitant une zone de 32 positions.

b) Ω = {ensemble des signaux d'une <u>unité</u> U}, S = {, ;}

${}_kL_{p1}^1$ = $E_1, E_2, \ldots, E_{p1}$

ℓL_{p1}^1 = $S_1, S_2, \ldots, S_{p2}$

L_2^2 = ${}_kL_{p1}^1$; L_{p2}^1 = $E_1, E_2, \ldots, E_{p1}$; $S_1, S_2, \ldots, S_{p2}$

est un couple de liste d'entrées-sorties d'une unité connectée à l'intérieur de U.

9) L'opérateur réduction défini au chapitre III Ad2 est une liste $\odot/A(1:n) \equiv {}_{A}L^{1}_{n}$ avec $Q \equiv \{A(1),A(2),\ldots,A(n)\}$

$S \equiv \{\odot\}$

En effet : ${}_{A}L^{1}_{n} = A(1) \odot A(2) \odot \ldots \odot A(n)$

- <u>Isomorphie de 2 éléments de *L*</u>

Elle est défini par les formules de récurrence :

1) Deux éléments $k_1L^{o}_{1}$ et $k_2L^{o}_{1}$ sont isomorphes.

2) Deux éléments $k_1L^{p}_{n} = L^{p1}_{t1}|L^{p2}_{t2}|\ldots|L^{pn}_{tn}$ et $k_2L^{q}_{m} = L^{q1}_{u1}|L^{q2}_{u2}|\ldots|L^{qm}_{um}$ sont isomorphes si $m = n$ et $p = q$ et si $L^{p1}_{t1}, L^{p2}_{t2},\ldots L^{pn}_{tn}$ sont isomorphes respectivement à $L^{q1}_{u1}, L^{q2}_{u2},\ldots,L^{qm}_{um}$.

- <u>Elément blanc</u> :

Nous désignerons ainsi une liste λ^{i}_{o} utilisant i séparateurs mais aucun élément de a.

<u>Conséquences</u> :

a) 2 éléments isomorphes utilisent les mêmes séparateurs.

b) 2 éléments isomorphes ont le même nombre de composantes.

c) 2 éléments isomorphes ont toutes leurs sous-listes isomorphes 2 à 2.

b) Opérateur Condition :

-.- Soit un ensemble de fonctions à 2 valeurs. (que l'on peut appeler {0,1} ou {-1,1}...)

Les fonctions booléennes appartiennent à F.

Appelons 1-f ou $\neg f$ le complément de tout élément $f \in F$. On a évidemment $\neg f \in F$.

-.- Soit $\circledcirc$ un opérateur dyadique défini sur un ensemble Q et qui à tous éléments $f_i \in F$ et $q_i \in Q$ fait correspondre : $q_j = f_i \circledcirc q_i \in Q$.

. Nous imposerons à $\circledcirc$ d'être distributif par rapport à tous les éléments d'un ensemble S.

A tout élément $L_n^i = L_{p1}^{u1} | L_{p2}^{u2} | \ldots | L_{pn}^{un} \in L(Q,S)$

$f \circledcirc L_n^i$ fait correspondre : $L_n^j = f \circledcirc L_{p1}^{u1} | f \circledcirc L_{p2}^{u2} | \ldots | f \circledcirc L_{pn}^{un}$

-.- Soit $\oplus$ un opérateur dyadique défini sur un ensemble Q et tel que $\forall\, q_i, q_j \in Q, \exists\, q_k \in Q, q_k = q_i \oplus q_j$

Nous étendrons cet opérateur $\oplus$ aux éléments de L de la façon suivante.

soient $k_1 L_n^p, k_2 L_n^p \in L$ isomorphes $\exists\, k_3 L_n^p$

$k_3 L_n^p = k_1 L_n^p \oplus k_2 L_n^p$ si tous les éléments $k_3 L_1^o$ sont obtenus par une opération $k_1 L_1^o \oplus k_2 L_1^o$ $(L_1^o, k_2 L_1^o, k_3 L_1^o \in Q)$.

De la définition de l'isomorphie sur L il découle que ceci est toujours possible.

DEFINITION

Soient 3 ensembles $\mathcal{F}$, $\mathcal{S}$, $\mathcal{Q}$, définis comme précédemment. Soient 2 opérateurs $\odot$ et $\oplus$ vérifiant les propriétés précédentes.

Soient ${}_iL_p^n$, ${}_jL_p^n$ et ${}_kL_p^n$ 3 éléments de $L(\mathcal{S},\mathcal{Q})$ isomorphes et $f \in \mathcal{F}$.

Nous appellerons opérateur-condition $C(\oplus,\odot,\mathcal{F},\mathcal{S},\mathcal{Q})$ la fonction, qui à f, ${}_iL_p^n$ et ${}_jL_p^n$ fait correspondre

$$ {}_kL_p^n = C_{\oplus}^{\odot}(f,{}_iL_p^n,{}_jL_p^n) = f \odot {}_iL_p^n \oplus f \odot {}_jL_p^n $$

EXEMPLE 1 :

Soit X une fonction booléenne quelconque et $\delta(X)$ la fonction qui vaut 1 lorsque X est vrai et -1 lorsque X est faux.

soit $\odot$ tel que $a \odot b = e^{axb}$ (a et b réels)
soit $\oplus \equiv +$.
soit $\mathcal{Q} \equiv \{x\}$, variable réelle.
$\mathcal{S} \equiv$ {ensemble vide}.

1) $C_{+}^{\odot}(\delta(X),x,x) = e^{x}+e^{-x} = 2\,\mathrm{ch}x$

C'est une fonction qui vaut toujours la même valeur pour x fixe et X quelconque.

2) $C_{-}^{\odot}\ (\delta(X),x,x) = e^{x}-e^{-x} = 2\ sh\ x$

ou $e^{-x}-e^{x} = -2\ shx$

Cette fonction vaut 2 sh x si X est vrai

-2 sh x dans le cas contraire

On voit que l'on peut définir une fonction réelle conditionnelle (positive si la condition est vrai, négative dans le cas contraire (par exemple).

EXEMPLE 2 : Opérateur condition chap $\left[\text{III A } d_1\right]$

Soit A une expression booléenne de dimensions

$a_1 : b_1,\ a_2 : b_2,\ \ldots,\ a_n : b_n$.

Les composantes de A forment un tableau que nous pourrons alors noter $A\ L_n^2$.

avec $\mathcal{Q} \equiv \{A(a_1,a_2,\ldots,a_n),\ A(a_1+1,a_2,\ldots,a_n),\ldots,A(b_1,b_2,\ldots,b_n)\}$

Soit C une constante booléenne scalaire.

Posons $\odot = \Lambda$ et $\oplus$ = NOP (non-opération).

$$C_{NOP}^{\Lambda}\ (C,AL^2{}_n,AL_n^2)$$

correspond à la définition en CASSANDRE de l'opérateur condition, dont on voit que c'est un cas très particulier de la nouvelle définition ci-dessus.

c) Description d'une grammaire de CASSANDRE, utilisant l'opérateur liste.

Nous présenterons la syntaxe sous forme d'un tableau dont la première colonne contiendra un nom pour chaque métavariable. La deuxième colonne est affectée à l'ensemble Ω, la troisième contient le numéro de la métavariable, la quatrième les éléments de l'ensemble S éventuellement, et la dernière la définition syntaxique de la métavariable.

Les métavariables sont notées M_i et les listes ${}_kL^i_j$, i étant le nombre de séparateurs utilisés, j le nombre d'éléments dans la liste et k le numéro de la métavariable ainsi définie.

EXEMPLE :

M17 désigne une liste de paire d'entiers et c'est une liste $17L^2_n$ utilisant les 2 séparateurs : et, et ayant n quelconque éléments.

PROPRIETE

Toute liste ${}_kL^m_n$ peut être réduite à un seul élément, le dernier séparateur s_n est alors inutilisé.

METAVARIABLE	$\mathcal{Q}$	N°	$\mathcal{S}$	DEFINITION
LETTRE	{A,B,C,....,Z}	1		M1 $\in \mathcal{Q}$
CHIFFRE	{0,1,2,3,4,5,6,7,8,9}	2		M2 $\in \mathcal{Q}$
EXPRESSION-PAYAN		3		Voir [ ]
IDENTIFIEUR	{M1} $\cup$ {M2}	4	blanc	M4=M2 $_4L^1_n$
DECLARATEUR	{Unité, Signal, Registre, Etat, Impulsion, Externe, Horloge, mémoire Bus, Etainitiai}	5		M5 $\in \mathcal{Q}$
ENTIER-SANS-SIGNE	{M2}	6	blanc	M6=$_6L^1_n$
ENTIER-AVEC-SIGNE		7		M7=-M6
VARIABLE ARITHMETIQUE		8		M8=M4
CONSTANTE-SCALAIRE	{0,1,φ}	9		M9 $\in \mathcal{Q}$
CONSTANTE-VECTEUR	" "	10	blanc	M10=$_{10}L^1_n$
CONSTANTE-TABLEAU	" "	11	,	M11=$_{11}L^2_n$
VARIABLE SIMPLE		12		M12=M4
PAIRE-D'ENTIERS	{M7} $\cup$ {M6}	13	:	M13=$_{13}L^1$
LISTE-D'ENTIERS	{M6}	14	,	M14=$_{14}L^1_n$
LISTE-DE-LISTE-d'ENTIERS	{M6} {(M14)}	15	,	M15=$_{15}L^2_n$
PAIRE-DE-BORNES	{M106} $\cup$ {M6} $\cup$ {M7}	16	: ,	M16=$_{16}L^1_n$
LISTE-DE-PAIRES-D'ENTIERS	{M16} $\cup$ {M7}	17		M17=$_{17}i^2_n$
INDICES	{M16} $\cup$ {M6} $\cup$ {M7} $\cup$ {M106} $\cup$ {M103}	18		M18 $\in \mathcal{Q}$
LISTE-D'INDICES	{M18} $\cup$ {Chaîne-vide}	19	,	M19=10L^1_n
LISTE-DE-DIMENSIONS	{M15} $\cup$ {M14} $\cup$ {M17}	20		M10 $\in \mathcal{Q}$
VARIABLE INDICEE		21		M21=M12(M19)
VARIABLE DECLAREE		22		M22=M12(M20)

LISTE-DE-VARIABLES-DECLAREES	{M12} ⊔ {M22}	23	,	M23=23L_n^1
ENTETE		24		M24=Unite M23;M23);
DECLARATION		25		M25=M5 M23
LISTE-DE-DECLARATIONS SPECIFICATIONS	{M25}	26	;	M26=26L_n^1
VARIABLE	{M12} ∪ {M21} ∪ {(M30)} *	27		M27 ∈ Q
TRANSPOSITION		28		M28=M16∿
TRANSPOSITION MATRICIELLE		29		M29 = ∿
VARIABLE-GENERALISEE	{M27} ∪ {M28 M27} ∪ {M29 M2 }	30	&	M30 = $_{30}L_n^1$
EQUIVALENCE		31		M31=M27≡M30
AFFECTATION ELEMENTAIRE		32		M32=M30⇐M103
CONNEXION ELEMENTAIRE		33		M33=M30:=M103
GENERATION D'IMPULSION		34		M34=M30⇐=M103
LISTE-D'ENTREES CONNECTEES	{M30} ∪ {M103} ∪ {chaîne vide}	35	,	M35=35L_n^1
LISTE-DE-SORTIES CONNECTEES	{M30} ∪ {chaîne vide}	36	,	M36=$_{36}L_n^1$
CONNEXION D'UNITE		37		M37=M4(M35;M36)
CONNEXION D'UNITE-INDICEE		38		M38=M4|M106| M37
ETIQUETTE		40		M40 = M4
ETATCOMPOSE	{M40}	41	:	M41=$_{41}L_n^1$
UNION-D'ETAT	{M41}	42	V	M42=42L_n^1
IMPULSION		43		M43=M4

INTERSECTION -D'ETAT	{M41}	44		$M44 = {}_{44}L^{1}_{n}$
ETAT-CONDI-TIONNEL		45		M45=si M103 alors M41 sinon M41
EXPRESSION-D'ETAT	{M40}∪{M41}∪{M42}∪{M44}∪{M45}	46		M46 ∈ Q
ALLERA		47		M47=Allera M46
ALLERA-SIMPLI-FIE		48		→
FAIRE		49		M49=faire M46
MEMBRE-D'INS-TRUCTION-D'AFFECTATION	{M32}∪{M34}∪{M47}∪{M49}∪{M55}∪{M56} {M63}∪{M64}∪{M70}∪{M72}∪{M77}∪{M86} * {M88}∪{M90}∪{M48}	50		M50 ∈ Q
INSTRUCTION ELEMENTAIRE D'AFFECTATION	{M50}	51	,	$M51 = {}_{51}L^{1}_{n}$
MEMBRE d'INS-TRUCTION DE CONNEXION	{M49}∪{M33}∪{M34}∪{M37}∪{M38}∪{M58} {M59}∪{M68}∪{M69}∪{M71}∪{M73}∪{M78} * {M87}∪{M89}∪{M91}	52		M52 ∈ Q
INSTRUCTION-ELEMENTAIRE-DE-CONNEXION	{M52}	53	,	$M53 = {}_{53}L^{1}_{n}$
INSTRUCTION INCONDITION-NELLE D'AF-FECTATION	{M32}∪{M34}∪{M47}∪{M48}∪{M49}∪{(M51)}	54		M54 ∈ Q
DEMI-INSTRUC-TION D'AFFEC-TATION CONDI-TIONNELLE		55		M55=si M103 alors M54
INSTRUCTION-CONDITIONNELLE D'AFFECTATION		56		M56=M55 sinon M54
INSTRUCTION-INCONDITION-NELLE-DE-CON-NEXION	{M33}∪{M34}∪{M37}∪{M38}∪{M49}∪{(M53)}	57		M57 ∈ Q
DEMI-INSTRUC-TION DE CON-NEXION CONDI-TIONNELLE		58		M58=si M103 alors M57

INSTRUCTION CONDITIONNELLE DE CONNEXION		59		M55=M58 sinon M57
VECTEUR-D'AFFECTATIONS	{M51} ∪ {chaîne vide}	60	)(	$M60={}_{60}L^{1}_{n}$
LISTE-DE-VECTEURS-D'AFFECTATION	{(M60)}	61	,	$M61={}_{61}L^{1}_{n}$
TABLEAU-D'INSTRUCTIONS-D'AFFECTATION	{(M61)} ∪ {(M60)}	62		M62 ∈ Q
DEMI-INSTRUCTION GENERALISEE d'AFFECTATION		63		M63=si M103 alors M62
INSTRUCTION-GENERALISEE-D'AFFECTATION		64		M64=M63 sinon M62
VECTEUR-DE-CONNEXIONS	{M53} ∪ {chaîne vide}	65	)(	$M65={}_{65}L^{1}_{n}$
LISTE-DE-VECTEURS-DE-CONNEXION	{(M65)}	66	,	$M66={}_{66}L^{1}_{n}$
TABLEAU-D'INSTRUCTIONS DE-CONNEXION	{(M66)} ∪ {(M65)}	67		M67 ∈ Q
DEMI-INSTRUCTION GENERALISEE DE CONNEXION		68		M68=si M103 alors M67
INSTRUCTION-GENERALISEE DE-CONNEXION		69		M69=M68 sinon M67
DEMI-CONDITION PAYAN-D'AFFECTATION		70		M70=si M3 alors M54
DEMI-CONDITION PAYAN-DE-CONNEXION		71		M71=si M3 alors M57

CONDITION-PAYAN-DE-CONNEXION		73		M73=M71 sinon M57
ELEMENT-DE-LISTE-DE-POUR	{M106}	74	pas a	$M74={}_{74}L^{1}_{3}$
LISTE-DE-POUR	{M106}	75		$M75={}_{75}L^{2}_{n}$
PROPOSITION-POUR		76		M76=Pour M8 égal M75 faire
INSTRUCTION-POUR d'AFFECTATION		77		M77=M76M54
INSTRUCTION-POUR-DE-CONNEXION		78		M78=M76M57
INSTRUCTION SOUS-TOP-D'HORLOGE		79		M79=<M43>M51
LISTE D'INSTRUCTIONS	{M79} ∪ {M53}	80	;	$M80=80L^{1}_{n}$
ETAT		81		M81=M41:M80
LISTE-d'ETATS	{M81}	82	blanc	$M82=82L^{1}_{n}$
OPERATEUR-DE-RELATION	{<, ≤, >, ≥, =, ≠}	83		M83 ∈ Q
RELATION	{M106}	84	M83	$M84=L^{1}_{2}$
PROPOSITION SI		85		M85=si M84 alors
DEMI-INSTRUCTION SI-D'AFFECTATION		86		M86=M85M54
DEMI-INSTRUCTION-SI-DE-CONNEXION		87		M87=M85M57
INSTRUCTION-POUR-CONDITIONNELLE d'AFFECTATION		88		M88=M85M77

INSTRUCTION-POUR CONDITIONNELLE DE CONNEXION		89		M89=M85M78
INSTRUCTION-SI D'AFFECTATION		90		M90=M86 sinon M54
INSTRUCTION-SI DE CONNEXION		91		M91=M86 sinon M57
PARTIE-FONCTIONNAIRE MIXTE		92		M92=M80M82
PARTIE-FONCTIONNELLE	{M92} ∪ {M80} ∪ {M82}	93		M93 ∈ 𝒬
SUITE D'ENTETE		94		M94=M26M93
DEFINITION-D'UNITE	{M24} ∪ {M24 M94} ∪ {M24 M93}	95		M95 ∈ 𝒬
OPERATEUR-ADDITIF	{+ , -}	96		M96 ∈ 𝒬
OPERATEUR-MULTIPLICATIF	{× , ÷ , Rem}	97		M97 ∈ 𝒬
EXPRESSION SIMPLE	{M100} ∪ {¬ M100}	98		M98 ∈ 𝒬
OPERATEUR MONADIQUE	{M28} ∪ {M29} ∪ {M106 M106} ∪ {M107}	99		M99 ∈ 𝒬
PRIMAIRE BOOLEEN	{M10} ∪ {M11} ∪ {M27} ∪ {M103} ∪ {M9}	100		M100 ∈ 𝒬
OPERATEUR DYADIQUE	{M109} ∪ {M83} ∪ {&}	101		M101 ∈ 𝒬
SECONDAIRE BOOLEEN	{M98} ∪ {si M103 alors M98}	102		M102 ∈ 𝒬
EXPRESSION	{M96 M106} ∪ {M98} ∪ {M102} ∪ {M99 M102}	103		M103 ∈ 𝒬
EXPRESSION ARITHMETIQUE SIMPLE	{M8} ∪ {M6} ∪ {(M106)}	104		M104 ∈ 𝒬
TERME ARITHMETIQUE	{M 104}	105	M97	$M105=105L_n^1$
EXPRESSION ARITHMETIQUE	{M105}	106	M96	$M106=106L_n^1$
REDUCTION		107		M107=M101
OP1	{↑, ↑, τ, ↦}	108		M108 ∈ 𝒬
OPERATEURBOOLEEN	{∨, ∧, ⊻, ⊼}	109		M109 ∈ 𝒬
DESCRIPTION LOGIQUE	{M95}	110	blanc	$M110=L_n^1$

d) Quelques propriétés sémantiques de CASSANDRE utilisant l'opérateur condition

Reprenons le tableau précédent.

1) Les 10 premières métavariables sont très simples. Dans les 3 listes qui apparaissent le séparateur utilisé est le blanc qu'il ne faut pas confondre avec la chaîne-vide. Dans la pratique si ce blanc se caractérise par un espace il peut avoir une longueur quelconque (nombre quelconque de colonnes vierges sur une carte perforée).

L'analyse syntaxique, au niveau de l'identifieur par exemple, est réalisée par un éditeur.

2) On a ${}_{11}L_n^2 = {}_{10}L^1_{p_1}, {}_{10}L^1_{p_1}X(p_{2-1}), \cdots, {}_{10}L^1_{p_1}Xp_2X\ldots(p_n-1)$.

propriété déjà notée des tableaux.

3) M13 est un couple (forme L_2^1)

4) M15 utilise le même séparateur que M14 mais toutes les listes M14 sont entre parenthèses. Le fait d'être entre parenthèses ou non distingue les 2 sortes de virgule.

5) La définition de M16 qui contient M106, n'est pourtant pas récursive car toute la définition de M106 pourrait se placer avant celle de M16 sans se servir de M16.

6) Par contre la définition de M18 est récursive car elle contient M 103, dont la définition contient M18.

7) M30 la variable généralisée se présente comme une liste de variables simples ou transposées, concatenées (c'est-à-dire séparées par &).

8) Dans M35 et M36, il est possible de vérifier syntaxiquement qu'une unité connecté ne contient pas d'expression (M103) en paramètre de sortie. Il n'est pas possible, par contre, de vérifier que ces paramètres sont des variables (M30) de type signal.

9) Dans la définition de M102 on retrouve l'opérateur condition

si iM103 alors iM98.

Nous avons vu qu'on peut le noter : $C^{\Lambda}_{NOP}\ ({}_{i}M103,\ {}_{i}M98)$.

Le résultat est l'intersection de chaque composante ${}^{k}_{i}M98$ de ${}_{i}M98$ avec ${}_{i}M103$.

Nous noterons de même : $C^{\Lambda}_{V}({}_{i}MO13,\ {}_{i}M98,\ {}_{j}M98)$ l'opérateur condition donnant un tableau dont la composante générale de rang k est :

$$({}_{i}M103\ \Lambda\ {}^{k}_{i}M98)\ V\ (\neg\ {}_{i}M103\ \Lambda\ {}^{k}_{j}M98)$$

Soit A une variable scalaire et O l'opérateur tel que :

$A \odot (M30 \leftarrow {}_{i}M103) = (M_{30} \leftarrow C^{\Lambda}_{NOP}(A, {}_{i}M103))$.

$A \odot (M_{3} \Leftarrow {}_{j}M103) = (M30 \Leftarrow C^{\Lambda}_{NOP}(A, {}_{j}M103))$.

A ⊙ (Allera ${}_{i}M40$) = (Allera si A alors ${}_{i}M40$)

A ⊙ (Faire ${}_{j}M40$) = (Faire si A alors ${}_{j}M40$)

A ⊙ (si ${}_{j}M103$ alors M54) = si (A Λ ${}_{j}M103$) alors M54

A ⊙ (si ${}_{j}M103$ alors M54 sinon M54) = si (A Λ ${}_{i}M103$) alors M54 sinon M54

(car ${}_{j}M103$ et ${}_{i}M103$ sont dans ce cas des scalaires.)

A ⊙ (si $_{j}$M103 alors M62) = si C^{Λ}_{NOP}(A, $_{j}$M103) alors M62

A ⊙ (si $_{i}$M103 alors M62 sinon M62) = si C^{Λ}_{NOP}(A, $_{i}$M103) alors M62 sinon M62

(Car $_{j}$M103 et $_{i}$M103 sont des tableaux dans ce cas).

On peut alors définir $C^{\odot}_{NOP}$(A,B)

B $\in \mathcal{Q}$ = {M32,M34,M47,M49,M55,M56,M63,M64}

Posons A ⊙ (Pour M8 égal M75 faire M54) =
Pour M8 égal M75 faire $C^{\odot}_{NOP}$ (A,M54)

De même - A ⊙ (si M84 alors M54) = si M84 alors $C^{\odot}_{NOP}$(A,M54)

- A ⊙ (si M84 alors $_{i}$M54 sinon $_{j}$M54 = si M84 alors $C^{\odot}_{NOP}$(A, $_{i}$M54)
sinon $C^{\odot}_{NOP}$(A, $_{j}$M54)

- A ⊙ (si M84 alors pour M8 égal M75 faire M54) si M84 alors pour M8
égal M75 faire $C^{\odot}_{NOP}$(A,M54)

Après avoir défini ces opérations

M55 = si M103 alors M54 peut s'écrire $C^{\odot}_{NOP}$(M103,M54)

On voit dans le tableau précédent que :

M54 $\in$ $\mathcal{Q}$ = {M32,M34,M47,M49,(M51)}

De même M56 = si M103 alors $_{i}$M54 sinon $_{j}$M54

peut s'écrire $C^{\odot}_{,}$ (M103, $_{i}$M54, $_{j}$M54)

où la "," indique seulement que :

$C^{\odot}_{,}$ (M103, $_{i}$M54, $_{j}$M54) = $C^{\odot}_{NOP}$(M103, $_{i}$M54), $C^{\odot}_{NOP}$(M103, $_{j}$M54)

11) Donnons à l'opérateur ⊕ la même définition pour M33,M34,M49,M58 M59,M68,M69,M71,M73,M78,M87,M89 et M91 que ⊛ pour M32,M34,M97,M49,M55,M56,M63, M64,M70,M72,M77,M86,M88,M90.

Ajoutons simplement que :

$A \oplus M4(M35 ; M36) = M4(C^{\oplus}_{NOP}(A,M35) ; C^{\oplus}_{NOP}(A,M36))$.

et $A \oplus M4\ |M106|(M35 ; M36) = M4\ |M106|\ (C^{\oplus}_{NOP}(A,M35) ; C^{\oplus}_{NOP}(A,M36))$.

Comme précédemment :

M58 = si M103 alors M57 peut s'écrire

$C^{\oplus}_{NOP}$ (M103, M57) et

M59 = si M103 alors $_{i}$M57 sinon $_{j}$M57

$C^{\oplus}_{;}$ (M103, $_{i}$M57, $_{j}$M57).

12) $M60 = {}_{1}M51)({}_{2}M51)({}_{3}M51)(.....)({}_{n}M51)$

On voit alors que (M60) est une liste de parenthèses contenant une M51 ou rien

$({}_{1}M51)({}_{2}M51)\ (\ )\ (\ )\ (....)\ ({}_{n}M51)$

13) M61 est un tableau donc de la forme :

$(M61) = (({}_{1}M51)_{1}\ ({}_{1}M51)_{2}\ (\ )_{3}\ (.....)\ ({}_{1}M51)_{p1}$,

$({}_{2}M51)_{1}\ ({}_{2}M51)_{2}\ (.....)({}_{2}M51)_{p_1 \times p_2 - 1}$,

---

$({}_{n}M51)_{1}\ ({}_{n}M51)_{2}\ (.....)\ ({}_{n}M51)_{p_1 \times p_2 \times \ldots \times (p_n - 1)})$

14) M63 = si ${}_{i}$M103 alors ${}_{i}$M62, M64 = si ${}_{i}$M103 alors ${}_{i}$M62 sinon ${}_{j}$M62. Comme ici ${}_{i}$M103 est un tableau de conditions booléennes et ${}_{i}$M62 et ${}_{j}$M62 sont des tableaux d'instructions d'affectation, M63 et M64 sont des généralisations de M55 et M56.

On pourrait les décrire d'une façon tout aussi concise en généralisant à son tour l'opérateur condition :

Soit f un tableau de fonctions à 2 valeurs et ${}_{i}L_{n}^{k}$, ${}_{j}L_{n}^{k}$ deux listes ayant par rapport à leurs 2 derniers séparateurs (d'indices k et k-1) une structure de tableaux isomorphes au tableau f.

C_{Θ}^{O} (f, ${}_{i}L_{n}^{k}$, ${}_{j}L_{n}^{k}$) est un tableau d'opérateurs condition, de même structure que f et ayant pour chaque composante d'indice I la valeur

$$C_{O}^{O}(f_{I}, {}_{I_{i}}L_{n}^{k}, {}_{I_{j}}L_{n}^{k})$$

15) Mêmes remarques pour M68 et M69.

16) Les conditions-Payan pourraient aussi s'exprimer à l'aide d'opérateurs C. Renvoyons à | |.

17) M74 peut se noter aussi bien :
M106 pas M106 à M106 que :
M106 à M106 pas M106.

18) Soit $\mathcal{Q}$ l'ensemble défini pour M51. Soit $\uparrow$ l'opérateur qui sert à la synchronisation des opérations de $\mathcal{Q}$ (ce peut être simplement Λ).

Soit ${}_{i}$M51 une liste d'éléments de $\mathcal{Q}$, soit H une impulsion d'horloge, on appellera instruction sous top d'horloge :

$$C_{NOP}^{\uparrow}(H, {}_{i}M51)$$

La réalisation de l'opérateur $\uparrow$ dépend de la technologie employée.

19) Soit $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$ avec

$\mathcal{Q}_1$ = {ensemble des instructions sous top d'horloge}

et $\mathcal{Q}_2$ = {M53}

Soit $\|$ l'opérateur égal à $\odot$ pour les éléments de $\mathcal{Q}_1$ et égal à $\oplus$ pour ceux de $\mathcal{Q}_2$.

soit ; un séparateur et ${}_i$M80 une liste d'éléments de $\mathcal{Q}$. Soit E une étiquette.

On appellera Etat : $C^{\|}_{NOP}$ (E, ${}_i$M80).

20) Les états d'une liste étant séparés par des blancs, en pratique un état contient tous les éléments compris entre 2 états-composés successifs ou entre un état-composé et le signe Unité qui suit.

21) Soit M84 une condition booléenne arithmétique, soit q un élément de l'ensemble $\mathcal{Q}$ = {M54,M57,M77,M78} et $\dagger$ l'opérateur tel que :

M84 $\dagger$ q = q si M84 est vrai et M84 $\dagger$ q = {chaîne vide} si M84 est faux on a alors :

M86 = $C^{\dagger}_{NOP}$(M84,M54) (noté si M84 alors M54)

M87 = $C^{\#\dagger}_{NOP}$(M84,M57) (noté si M84 alors M57)

M98 = $C^{\#}_{NOP}$(M84,M77) (si M84 alors Pour ....)

M89 = $C^{\#}_{NOP}$(M84,M78) (" ")

M90 = $C^{\#}_{;}$(M84, ${}_i$M54, ${}_j$M54) (si M84 alors ${}_i$M54 sinon ${}_j$M54)

M91 = $C^{\#}_{;}$(M84, ${}_i$M57, ${}_j$M57) (si M84 alors ${}_i$M57 sinon ${}_j$M57)

22) Une description logique est une liste d'unités séparées par des blancs. Une unité contient tous les éléments situés entre deux symboles Unité successifs.

APPLICATIONS

I - COMPILATION DE CASSANDRE EN EQUATION BOOLEENNES - AIDE A LA SYNTHESE ET A L'IMPLANTATION

Choisissons une base d'opérateurs booléens. La base la plus usuelle est la base I ($\neg, V, \Lambda$). Conservons l'opérateur condition indispensable pour décrire les opérations entre une variable scalaire et une variable tableau. Conservons les notions formelles (arithmétique et macro notions) qui ont un rôle purement descriptif. Laissons inchangées l'entête d'unité et les déclarations - spécifications.

Nous appellerons CASSANDRE-BOOLEEN le langage que l'on peut définir avec les notions ci-dessus et les opérations d'affectation et de connexion.

A - PROCESSUS DE SYNTHESE

a) Plusieurs personnes ont traité la synthèse d'une description de système logique comme celle d'un système séquentiel. Le profresseur Gerace par exemple transforme d'abord la description en tableau d'état puis traite le tableau d'état.

En CASSANDRE nous avons vu que l'on trouve imbriqués, la description d'opérations logiques se traduisant par des réseaux combinatoires joignant entre elles les différentes bascules, et la description d'un automate de contrôle défini par les états d'une unité et les ordres allera et faire.

Nous traiterons comme une machine séquentielle l'automate de contrôle, puis le reste du processus consistera à faire la synthèse de fonctions combinatoires.

Cette dissociation en 2 traitements consécutifs est possible si l'on admet que les variables internes seront mémorisées dans des registres. Nous renvoyons pour la première partie du traitement à Gerace [34], Schcorr [89] et Payan [78]

En CASSANDRE cette partie consistera à traiter une unité (ayant plus d'un état) à la fois.

Les états seront codés et l'automate sera réduit (cela veut dire qu'après traitement l'unité n'aura peut être pas le même nombre d'états qu'elle avait au début. Ce qui est évident dans le traitement défini par Payan).

Dans ce traitement les ordres aller seront remplacés par des affectations (←), les ordres faire par des connexions (:=) et les états disparaitrait au profit d'une instruction conditionnelle.

Le résultat de ce premier traitement sera une unité décrite en CASSANDRE, sans l'aide des 3 notions qui ont disparu.

Le deuxième traitement consiste à passer de cette description à une autre description équivalente en CASSANDRE-BOOLEEN.

Cette phase a été réalisée sur l'IBM 7044 avec la collaboration de Monsieur Y. Archer [65]. Nous n'en donnerons ici qu'une description plus succinte à l'aide des notions définies au chapitre précédent.

Si la technologie permet la réalisation à l'aide des opérateurs de la base choisie (exemple $\neg$, $\wedge$, $\vee$) on ajoutera seulement une procédure de minimisation des fonctions booléennes tenant compte si c'est nécessaire des contraintes de fan-in et fan-out (de telles procédures ont déjà fait l'objet de nombreux travaux). S'il faut changer de base (parce que l'on utilise des opérateurs Nor, Nand ou Majorité), on appliquera au préalable à chaque équation booléenne les procédures de synthèse dans ces technologies particulières qui existent nombreuses à ce jour (Deschizeaux, Saillard, Lustman [58], Macheras [60],...) Ces dernières procédures ne fonctionnent en général que pour des fonctions booléennes assez simples (moins de 10 variables), il est donc raisonnable dans le traitement de CASSANDRE de ne les appliquer qu'après une partition de la description (et du système) en

morceaux assez petits.

Il n'est donc pas question d'une minimisation globale d'un système logique mais d'optimisations locales. D'ailleurs ce problème de minimisation semble bien moins important que ceux d'implantation, le processus de réalisation s'employera de préférence à favoriser ces derniers.

B - TRADUCTION EN CASSANDRE-BOOLEEN

Le point de départ est le langage complet moins les états, les ordres allera, et les ordresfaire.

Les règles sémantiques définies au chapitre précédent permettent de décrire les procédures nécessaires à cette traduction.

b1) Les instructions d'affectation ou de connexion conditionnelles généralisées sont transformées en liste d'instructions d'affectation conditionnelle ou de connexion conditionnelle normales, séparées par des virgules.

La procédure b1 est récursive puisqu'une instruction conditionnelle généralisée peut en contenir une autre.

EXEMPLE si A(1:3) alors (L1)(L2)(L3) sinon (L4) ( ) (L5)

donnera : si A(1) alors L1 sinon L4, si A(2) alors L2,
si A(3) alors L3 sinon L5,

b2) Les conditions-logiques qui sont constituées par une expression booléenne contenant plus d'une variable, sont réalisées par des ordres de connexion placés en tête de l'unité. Un signal est créé pour les repérer. De cette façon on ne les construira qu'une fois et on utilisera le signal crée à chaque occurence.

EXEMPLE : si Λ/AVB alors (I_1, si V/A≠B alors I_2 sinon I_3, I_4,...) sinon ... ;

donnera : S_1 := Λ/AVB ;

S_2 := V/A≠B ;

---

si S_1 alors (I_1, si S_2 alors I_2 sinon I_3, I_4,...) sinon ... ;

---

b_3) Les conditions disparaissent en appliquant les règles |...| du chapitre précédent :

EXEMPLE : si S_1 alors (A ← B, si S_2 alors
C D sinon E ← F, G ← M) sinon .... ;

donnera :

si S_1 alors (A ← B, C ← si S_2 alors D,
E ← si S2 alors F, G ← H) sinon .... ;

puis A ← si S_1 alors B, C ← si (S_1 Λ S_2) alors D,
E ← si (S_2 Λ S_1) alors B, G ← si S_1 alors M,

Ces règles se traduisent toutes par des procédures récursives.

b_4) Les opérateurs booléens qui ne sont pas ceux de la base que l'on conserve sont traduits dans cette base suivant les règles de la table p(43-b).

Quand les traitements b_1,b_2,b_3,b_4 ont été effectués, le langage obtenu est du CASSANDRE-BOOLEEN mais il reste à regrouper les équations.

Si les règles de compatibilité des opérations ont été respectées, 2 opérations affectant la même variable doivent s'opérer sous des conditions disjointes. Il reste à fixer un opérateur pour regrouper les seconds membres

des équations. Prenons, car c'est courant, le ou logique (V), ce n'est pas obligatoire.

b_5) Regroupement des équations.

Toutes les connexions de signaux sont regroupées en tête. Puis toutes les affectations de registre classées par catégories suivant l'horloge qui synchronise les transferts.

EXEMPLE : Unité <ENTETE> ;
<Liste de déclarations-spécifications> ;
<Liste de connexions> ;
<Horloge1> <Liste d'affectations>;
<Horloge2> <Liste d'affectations>;

<Horlogep> <Liste d'affectations> ;

b_6) Regroupement des seconds membres d'équations.

On fait le ou-logique (par exemple) des expressions qui affectent une même variable dans un même groupe.

EXEMPLE : $A := E_1$, $B := E_2$, $A := E_3$,
$A := E_4$, $C := E_5$, ...,
donnera : $A := (E_1)\ V\ (E_3)\ V\ (E_4)$, $B := E_2$, $C := E_5$,...

Comme on a pris soin de créer des signaux intermédiaires chaque fois que la même expression risquait de s'effectuer 2 fois, on peut sans crainte effectuer ensuite les réductions des seconds membres (si ces réductions existent !)

Pour ces opérations complémentaires nous renvoyons à Mme De Polignac [65].

- Notons simplement ceci : si pour une connexion de signal on connait toutes les conditions logiques sous lesquelles ce signal est connecté et si l'on dispose aisément du complément de ces conditions logiques on peut introduire dans les seconds membres des variables Φ booléennes qui peuvent susciter des réductions.

EXEMPLE : S := si A alors B, ...,
S := si C alors D, ...,
S := si E alors F, ...,

donnera : S := (si A alors B) V (si C alors D) V (si E alors F) V
(si ¬ (A V C V E) alors φ) ;

Cette règle ne s'applique pas aux affectations de registre.

Illustrons A et B sur un exemple :

```
Unité  ADD (AAD, AA(1:16), MM(1:16) ; S(1:17), ADT) ;
Registre A(1:16), M(1:16), N(1:16), AD, OF, L ;
Impulsion H ;
            ADT := AD ;
            S   := L & M ;
<H>   AD ← AAD ;
ETA0 : <H> OF ← 0, A ← AA, M ← MM ;
ETA1 : <H> N ← M, → ;
ETA2 : <H> A ← A ≠ M, OF&N ← (N Λ A) & 0, → ;
ETA3 : <H> si V/(N Λ A) alors allera ETA2 sinon
           (M ← A, L ← OF, AD ← 0, → ) ;
ETA4 : <H> si AD alors (L ← 0, faire ETA0,
           allera ETA1) sinon allera ETA4 ;
```

Après 1er traitement on obtiendrait :

Unité ADD (AAD,AA(1;16),MM(1:16) ; S(1:17), ADT) ;
Registre A(1:16); M(1:16), N(1:16), AD,OF,L,R(1:2) ;
Impulsion H ;

ADT := AD ;
S := L&M ;

<H> AD ← AAD,
si R(1) Λ R(2) alors (N ← M, R ← (10)),
si R(1) Λ ¬R(2) alors (A ← A ≠ M, OF & N ← (N Λ A) & 0, R ← (01)),
si ¬R(1) Λ R(2) alors (si V/(N Λ A) alors R ← (10)
sinon (M ← A, L ← OF, AD ← 0, R ← (00),
si ¬R(1) Λ ¬R(2) alors (si AD alors (L ← 0, OF ← 0, A ← AA, M ← MM,
R ← (11)) sinon R ← (00)) ;

On a introduit un registre d'état R, réduit les états à 4 et codés ceux-ci.

Après le 2ème traitement on obtiendrait :

Unité ADD(AAD,AA(1:16),MM(1:16); S(1:17),ADT) ;
Registre A(1:16),M(1:16),M(1:16),AD,OF,L,R(1:2) ;
Signal Z_1,Z_2,Z_3,Z_4,Z_5,Z_6,Z_7,Z_8,Z_9 ;
Impulsion H ;

ADT := AD ;
S := L&M ;
Z_1 := V/(N Λ A) ;
Z_2 := R(1)Λ¬R(2) ;
Z_3 := R(1) Λ R(2) ;
Z_4 := R(1)Λ R(2) ;
Z_5 := R(1)Λ ¬R(2) ;
Z_6 := Z_4 Λ ¬ Z_1 ;
Z_7 := Z_4 Λ Z_1 ;
Z_8 := Z_5 Λ AD ;
Z_9 := Z_5Λ¬ AD ;

<H> N ← (si Z_2 alors M) V (si Z_3 alors (N(2:16) A(2:16)) & 0).

OF ← (si Z_3 alors N(1) Λ A(1)) V (si Z_8 alors 0),
A ← (si Z_3 alors (¬ A ΛM) V (¬ M ΛA)) V (si Z_8 alors AA),
M ← (si Z_8 alors A) V (si Z_8 alors MM),
L ← (si z_6 alors OF) V (si Z_8 alors 0),
AD ← (si Z_6 alors 0) V AAD.
R ← (si Z_2 alors (10)) V (si Z_3 alors (01))
V (si Z_7 alors (10)) V (si Z_6 alors (00) V (si Z_8 alors (11))
V (si Z_9 alors (00) ;

A titre d'exemple simplifions l'équation de R

$R_o(1)$ ← R(1) Λ ¬R(2) - [illegible]
$R_1(1)$ ← ¬R(1) Λ ((R(2) Λ Z1) V (¬ R(2) Λ AD)).
$R_o(2)$ ← R(2).
$R_1(2)$ ← R(2) Λ (R(1) V AD).

On voit que cette simplification est extrémement souhaitable.

On voit également que les équations se dédoublent car on peut charger aussi bien 1 que 0.

C - AIDE A L'IMPLANTATION

Il existe dans la description en CASSANDRE d'un système logique une modularité due au découpage en unités, mais également due à la manipulation de variables de type tableau pouvant exprimer des structures très répétitives, due aussi à l'usage de l'arithmétique et due enfin à la mise en facteur des conditions logiques.

Lorsqu'on traduit une telle description en équations booléennes, ce qui est presque toujours indispensable pour la réalisation effective, on peut craindre de perdre cette modularité et de plus on fera certainement disparaître le découpage à priori du système, que le concepteur a fait en

y mettant toute sorte d'indications utiles pour l'implantation.

En effet, à titre d'exemple les équations des entrées de différentes bascules d'un registre risquent d'être assez différentes lorsque l'on a rajouté ce que chacune a de particulier à ce que toutes ont en commun, et après un processus de réalisation partant des équations booléennes, toutes ces bascules peuvent se trouver éparpillées, ce qui est fâcheux puisqu'elles forment un registre et qui risque d'être catastrophique pour le dépannage et la maintenance.

Ce qui vient d'être dit pour un registre sera tout aussi vrai pour les signaux qui constituent en général des regroupements intermédiaires précieux.

Faire la synthèse d'un système logique c'est passer de sa description en CASSANDRE la plus abstraite possible, à une description fonctionnellement équivalente, qui ne contienne plus que des connexions d'unités, lesquelles unités existent toutes faites dans la technologie en vente sur le marché.

On peut envisager d'automatiser ce traitement en utilisant les définitions sémantiques associées à tous les éléments du langage et précédemment décrites.

C'est pourquoi Monsieur Liddell [56] réalise un tel processus, en s'appuyant sur une analyse syntaxique du texte initial en CASSANDRE.

Sans perdre aucune des données initiales il superpose au réseau d'unités interconnectées des réseaux de plus en plus fins d'unités imbriquées dans les précédentes, jusqu'à obtenir des unités correspondant à des opérateurs élémentaires.

Aucune information d'ordre topologique n'est perdue puisque tous les intermédiaires sont mémorisés et s'ajoutent au découpage initial.

Nous renvoyons à [56] et [5] pour la définition de ce processus de SURDECOUPAGE (Partitionning), ajoutons qu'il sera tout à fait général, quand l'opération inverse correspondant à regrouper plusieurs unités pour en faire une seule sera permise. Nous appellerons RESTRUCTURATION cette dernière opération.

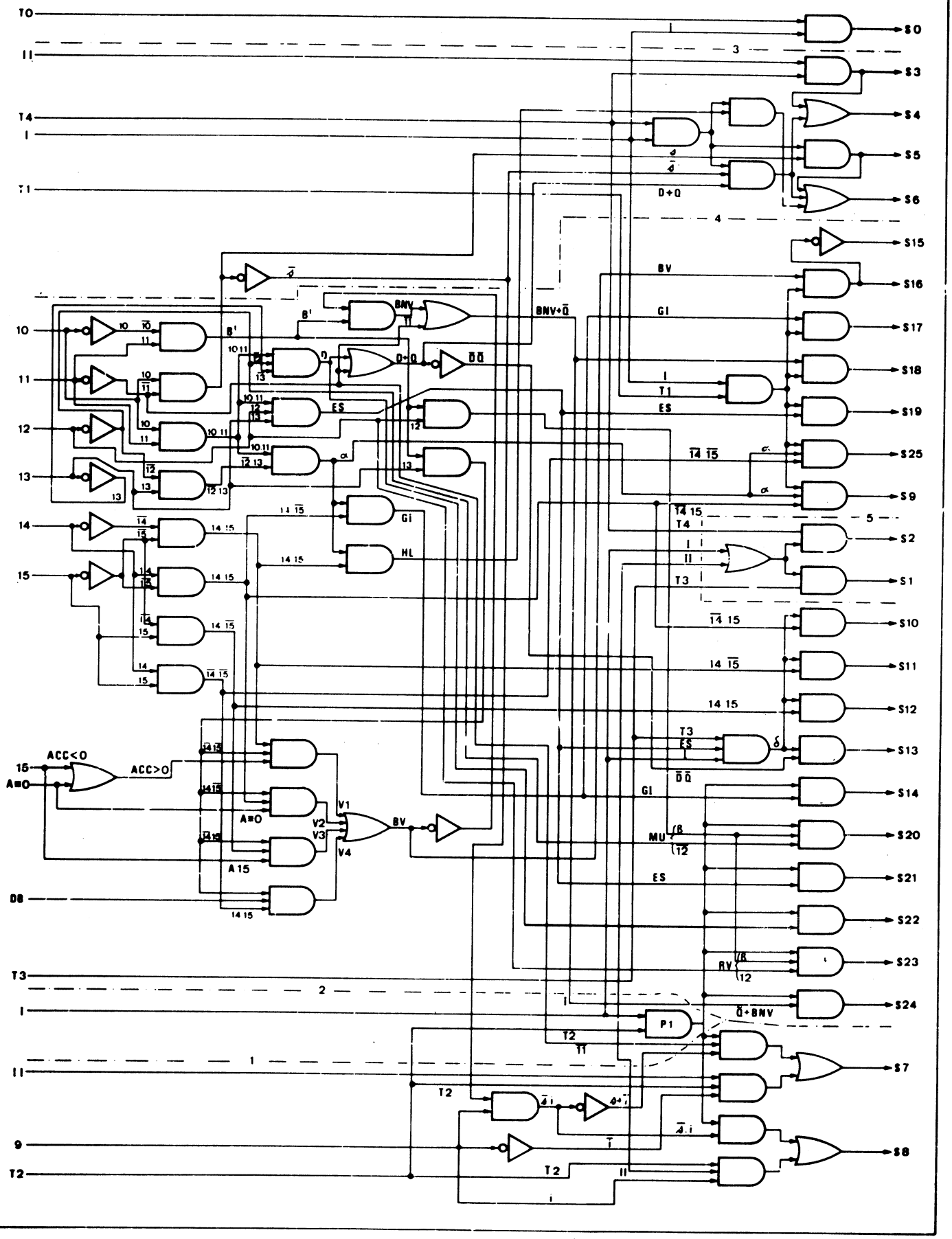
Les opérations de SURDECOUPAGE et de RESTRUCTURATION sont internes dans CASSANDRE.

Le texte source est en CASSANDRE le texte produit aussi. Elles s'opèrent en conversation homme-machine à l'aide de ce langage. L' ntil développé par M. Liddell semble très adapté à l'introduction des contraintes d'implantation tout au long du processus de synthèse ce qui est indispensable pour réaliser un ensemble logique important dans les technologies modernes (utilisation des boitiers M.S.I. ou mise sur une pastille de L.S.I.). Deux procédures semblent particulièrement utiles à ce niveau, nous les appellerons COUT et CONNEXION.

La procédure COUT aurait pour effet de donner une approximation dans une technologie donnée du nombre d'entrées d'opérateurs élémentaires qui interviennent dans la réalisation de cette unité. (le nombre d'entrées d'opérateurs donne une mesure plus exacte que le nombre d'opérateurs eux-mêmes).

Cette procédure utiliserait le COUT des unités contenues dans l'unité considérée, elle serait <u>récursive.</u>

La procédure CONNEXION, calculerait le nombre de liaisons d'une unité avec l'extérieur, elle opérerait donc un calcul très simple sur l'ENTETE de l'unité. Nous pouvons illustrer ces quelques reflexions sur un exemple fourni par Electronique Marcel Dassault : Circuit de contrôle à réaliser en L.S.I avec une technologie M.O.S.T. (voir figure suivante).


T0
I1
T4
I
T1
10
11
12
13
14
15
ACC<0
ACC>0
A=0
DB
T3
I
I1
T2
9
T2
S0
S3
S4
S5
S6
S15
S16
S17
S18
S19
S25
S9
S2
S1
S10
S11
S12
S13
S14
S20
S21
S22
S23
S24
S7
S8
BV
GI
ES
MU
RV
P1
D+Q
BNV+Q
A15
V1
V2
V3
V4
HL

Supposons qu'après avoir appliqué au circuit de contrôle considéré le traitement de surdécoupage, on obtienne au niveau le plus bas le circuit représenté figure 1.

Appliquons à cette unité la fonction COUT.
Le résultat serait 119 entrées de l'opérateur Λ et
21 entrées de l'opérateur V ou
12 entrées de l'opérateur $\neg$ non.

Nous allons chercher à réaliser le circuit à l'aide d'une cellule de base adaptable, en technologie MOS et décrite en figure 2 chapitre IA. Cette cellule peut réaliser 8 entrées de 0 ou 6 entrées de si l'on ne dispose que des variables directes et jusqu'à 12 entrées de dans le cas où on dispose aussi des variables complémentées ou encore jusqu'à 11 entrées de .

Elle peut également réaliser les circuits combinant ces opérateurs et vérifiant $2n1+(n2+1)+n3 \leq 12$ si n1, n2 et n3 sont respectivement le nombre d'entrées des opérateurs et Λ , ou V, $\neg$ non dans le cas où l'on ne dispose que des variables directes

et $n1+(n2+1)+n3 \leq 12$ lorsqu'on a également les variables complémentées.

Un calcul simple montre que le COUT du circuit en fonction de cette cellule est :

$$14 \leq k \leq 24$$

Nous allons essayer de le réaliser en 16 cellules en utilisant au mieux les 19 variables qui existent sous forme directe et complémentée On voit sur un tel schéma l'énorme complexité des connexions. Il semble peu réaliste d'envisager un traitement automatique du partitionnement en cellules ayant entre elles le minimum de connexions.

Un traitement conversationnel pourrait donner une solution satisfaisante et rapide.

On procède par simplification progressive du problème en appliquant la RESTRUCTURATION aux unités créées.

A la vue de figure 1, 5 coupures s'imposent rapidement (traits numérotés).

Les parties A et B séparées par 3 et 5 ne sont pas liées au reste du circuit. La partie C séparée par 1 ou 2 n'a que 3 liaisons avec le reste seule la fonction COUT permet de dire si l'on coupe suivant 1 ou 2 .

Avec la porte P1 : $23 \leq \text{COUT (C)} \leq 38$

sous la porte P1 : $21 \leq \text{COUT (C)} \leq 34$

Si l'on admet que l'on dispose des entrées I et $\bar{I}$, II et $\overline{II}$ (ce qui est vrai puisque ce sont des sorties de bascule), on peut couper suivant 2 et on aura COUT (C) = 24 ce qui autorise la réalisation de C en 2 cellules.

On obtiend ainsi les figures 2,3 et 4 correspondant à des circuits réalisables en 5 cellules.

Le circuit résiduel peut être redessiné, il correspond à la figure 5.

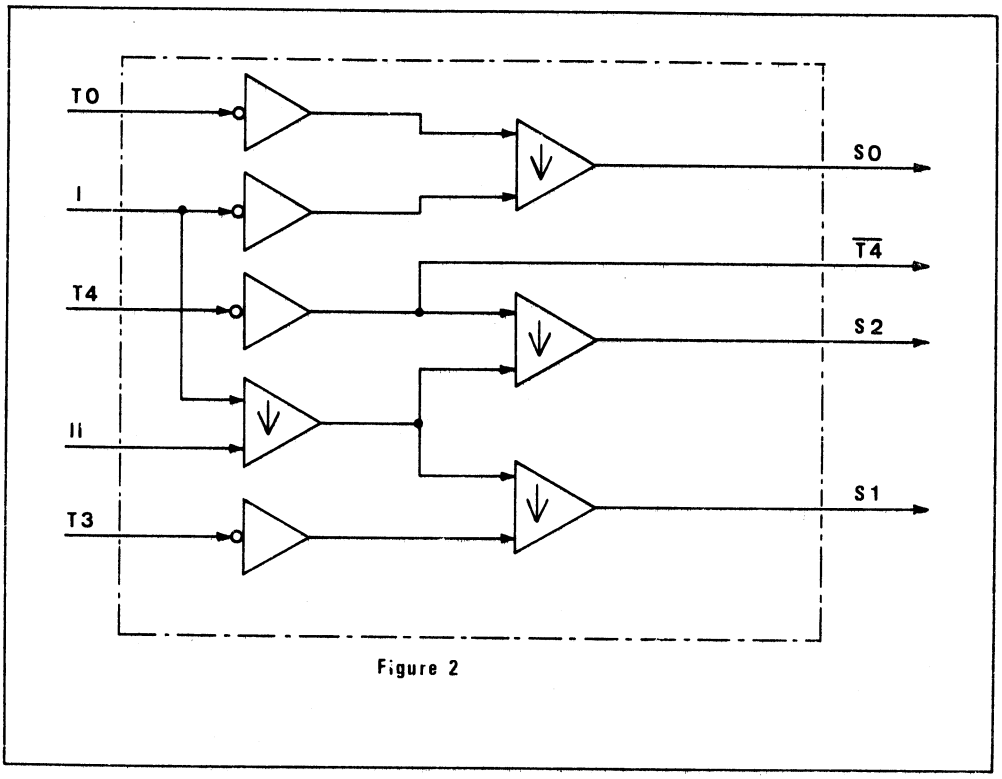


Figure 2

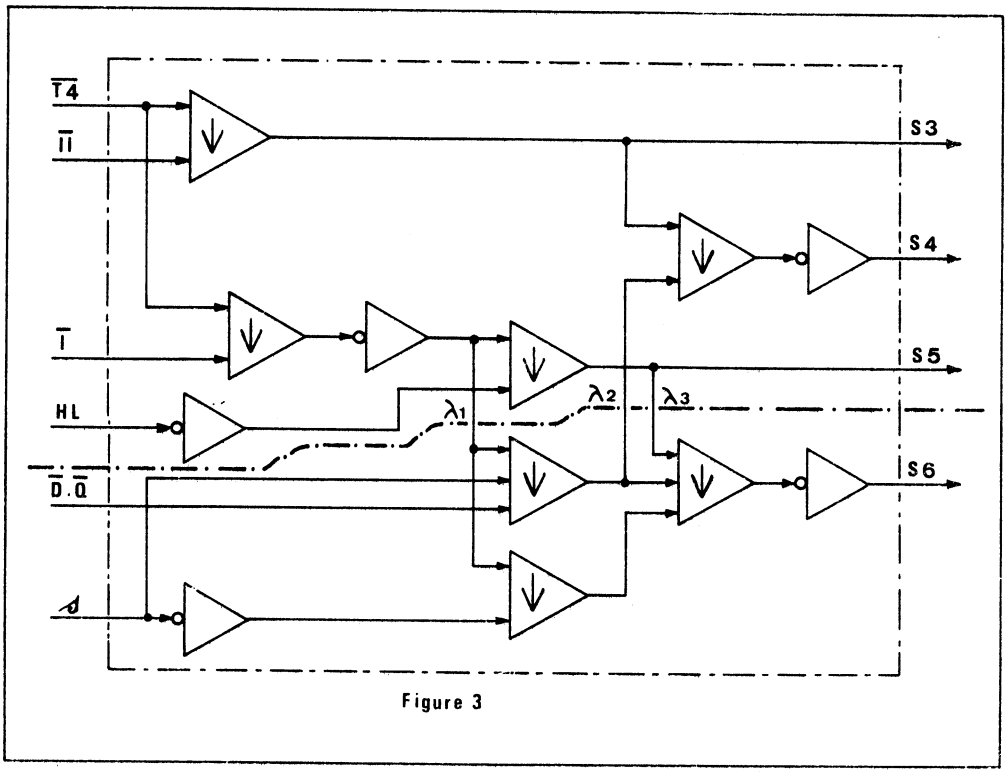


Figure 3

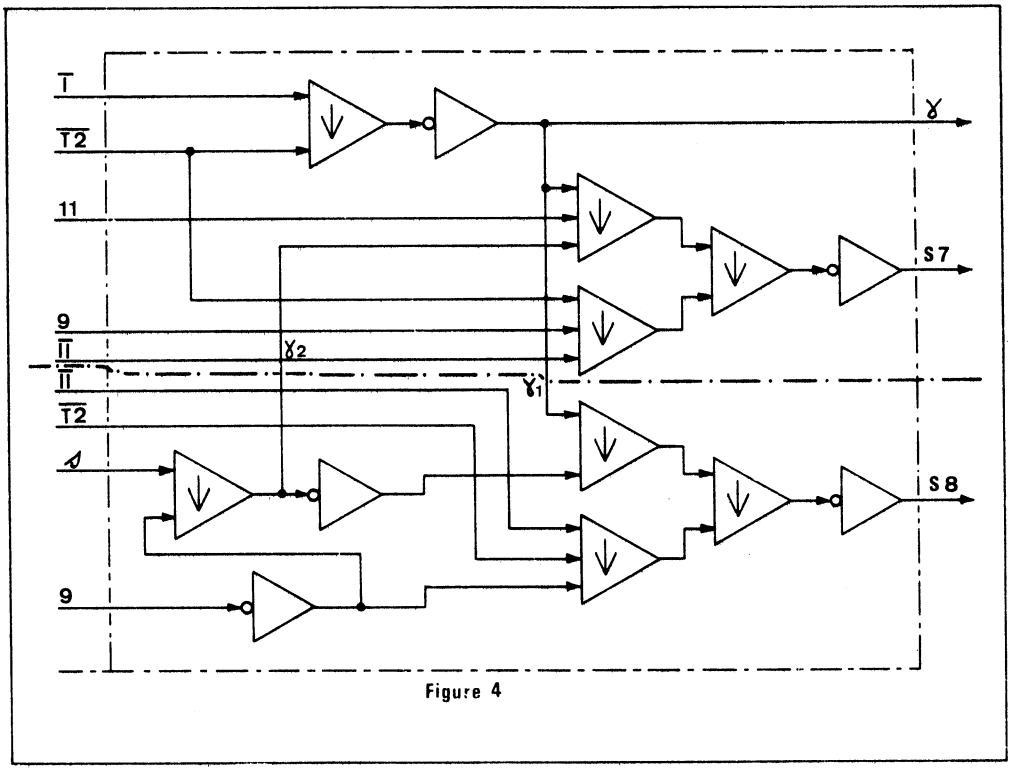
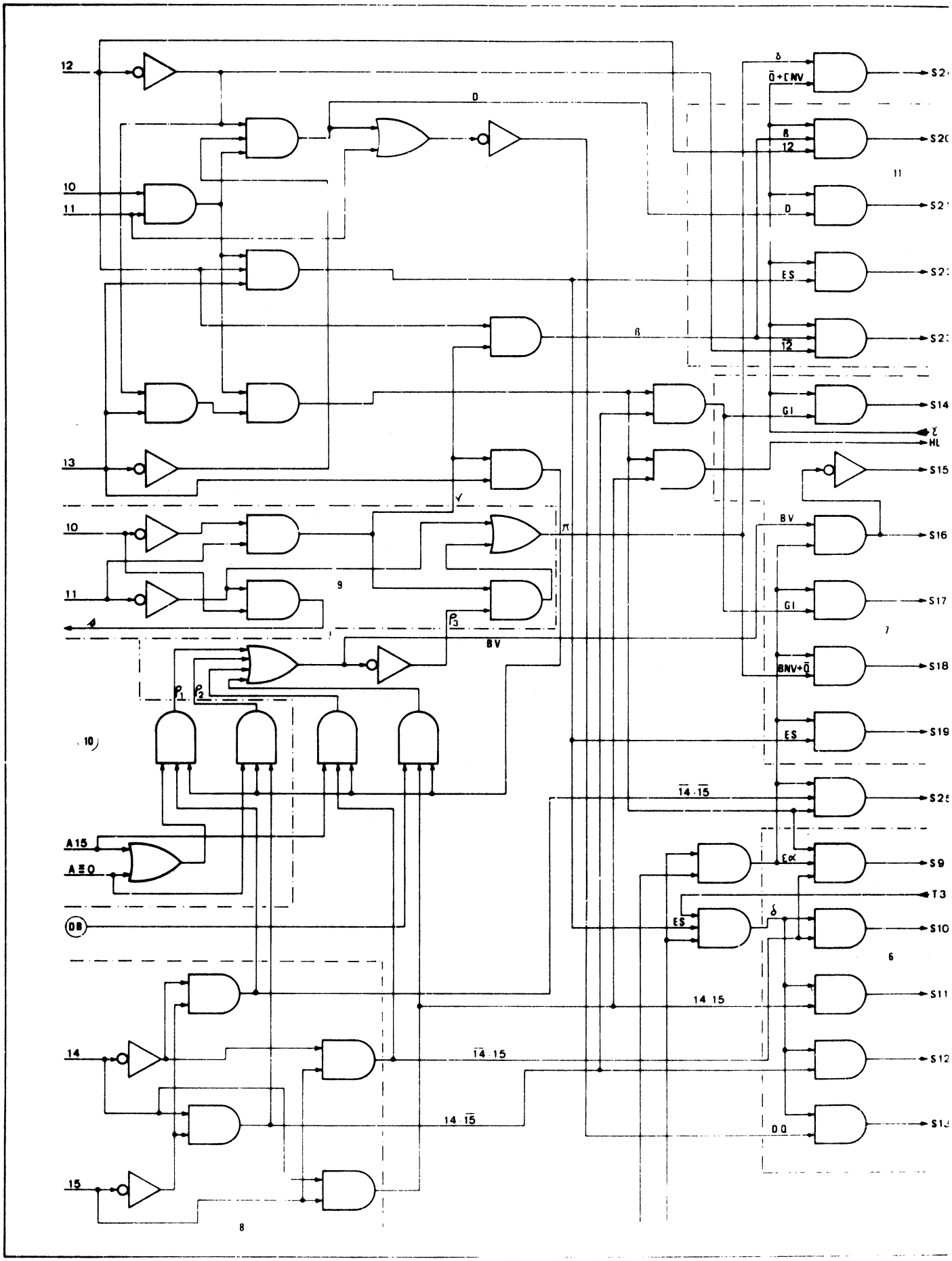


Figure 4


12
10
11
13
10
11
A15
A≡0
DB
14
15
D
ß
BV
14 15
14 15
14 15
δ
ES
Eα
DQ
S9
T3
S10
S11
S12
S14
S15
S16
S17
S18
S19
S25
HL
6
7
8
9
10
11

Sur la figure 5, on peut faire apparaître assez facilement les mroceaux 6,7,8,9,10,11 auxquels correspondent les figures 6,7,8,9,10,11. Chaque morceau se trouve cette fois réalisable en 1 cellule.

Le circuit résiduel est de nouveau RESTRUCTURE, on peut en donner le schéma figure 12.

On peut alors terminer le travail en formant les morceaux 13,14,15,16,17 chacun réalisé par une cellule. On a bien donné une réalisation en 16 cellules et on étudie pour l'implantation le graphe de connexion de ces cellules entre-elles. Cette étude fait apparaître 2 sous-graphes planaires, on peut enfin minimiser les croisements dans les liaisons entre les 2 sous-graphes.

On obtiend les schémas p 75-C et 77-C.

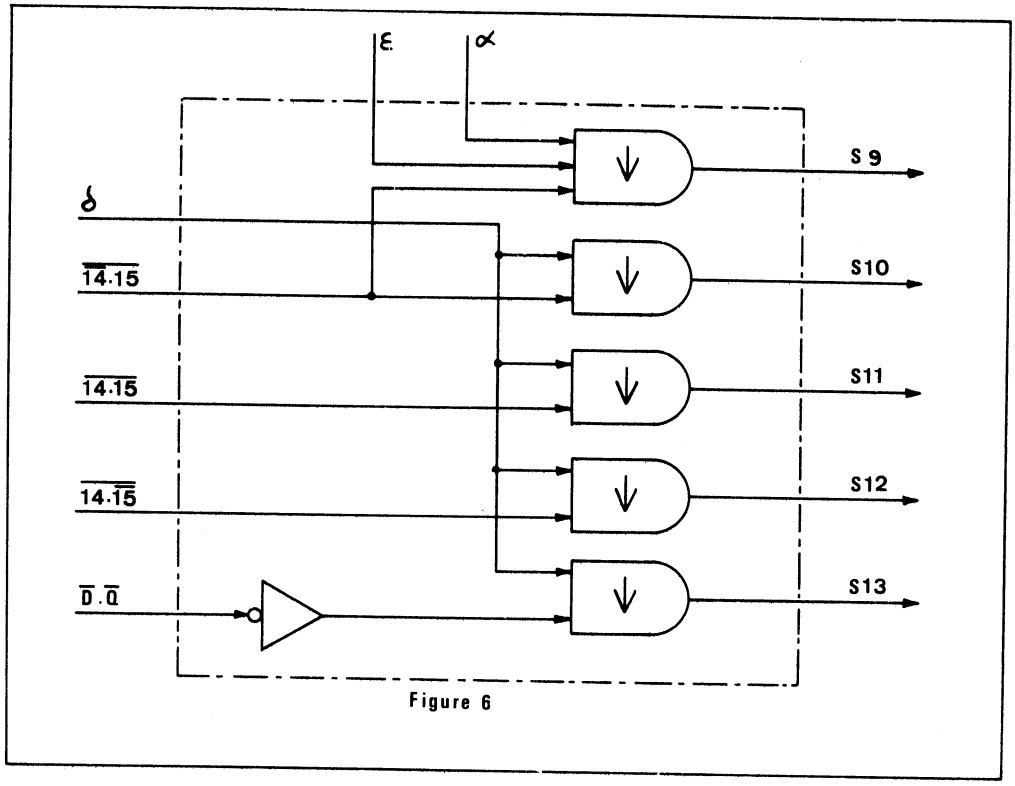


Figure 6

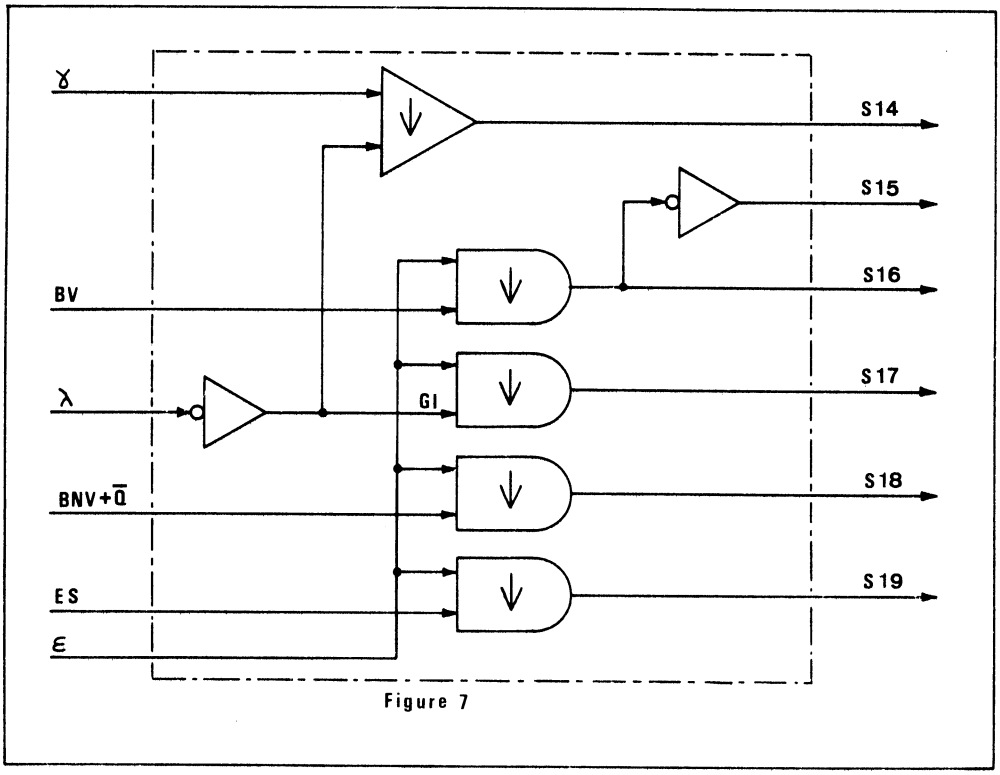


Figure 7

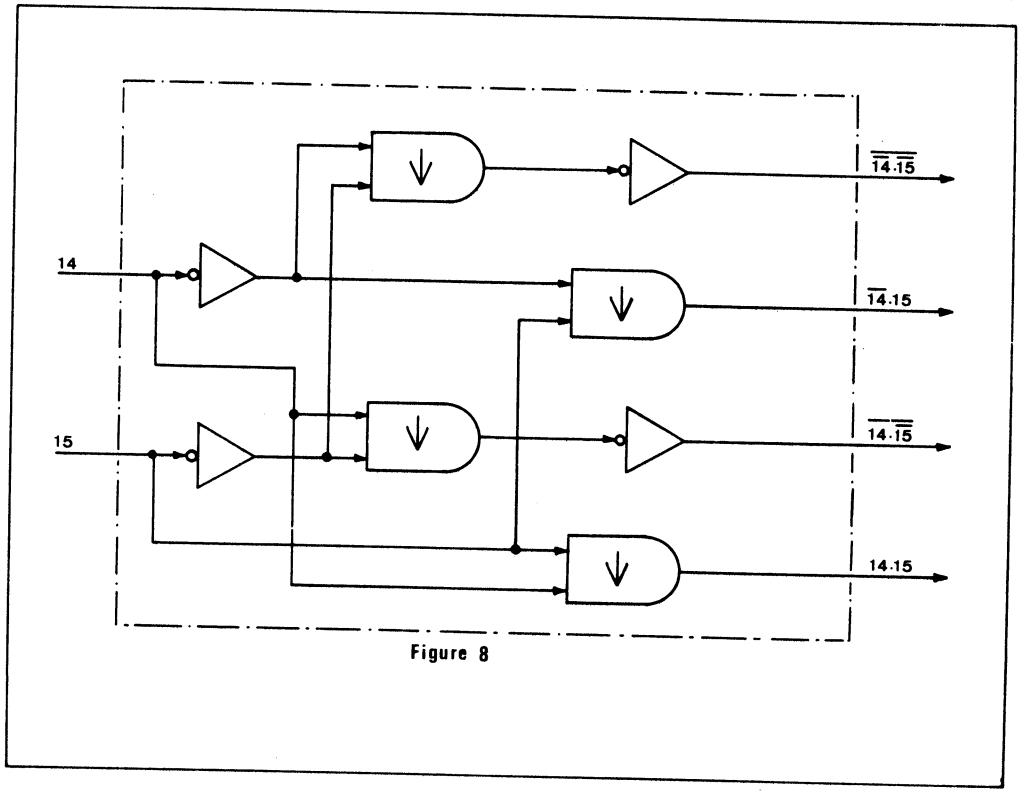


Figure 8

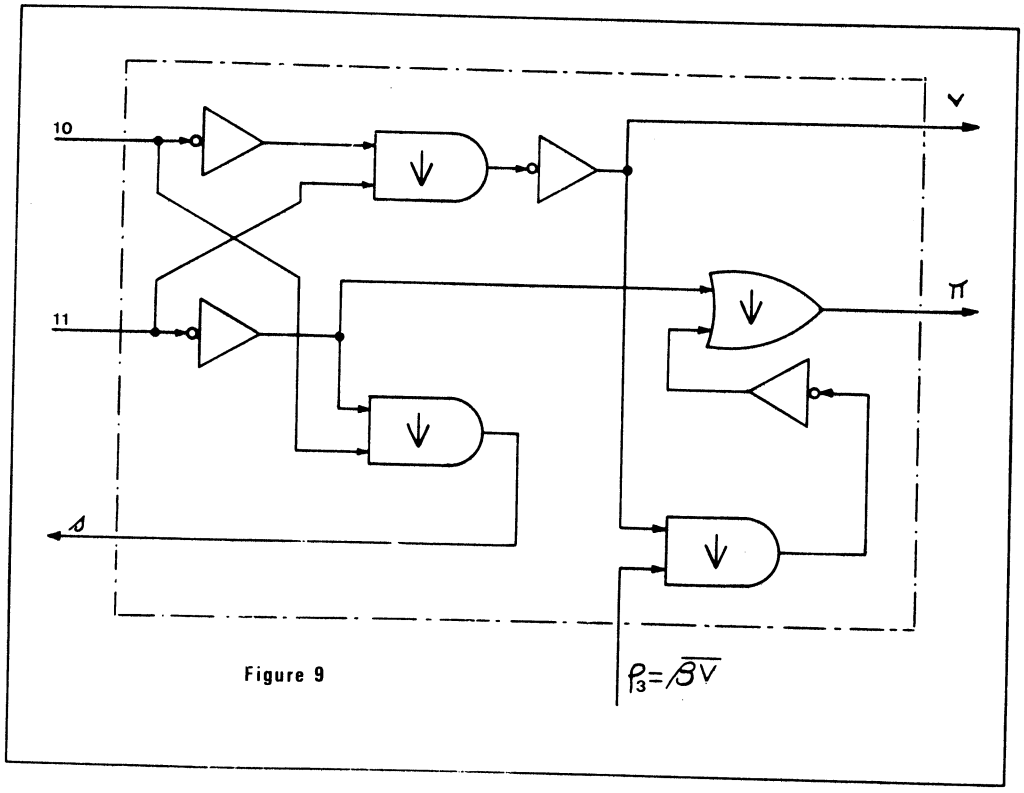


Figure 9

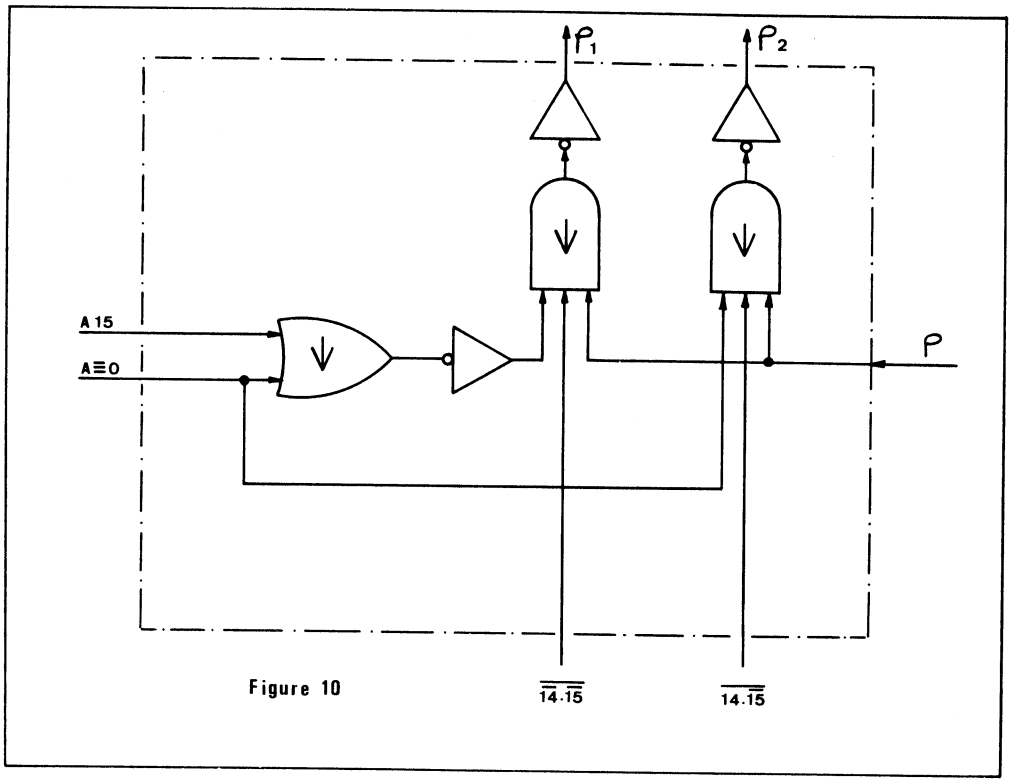


Figure 10

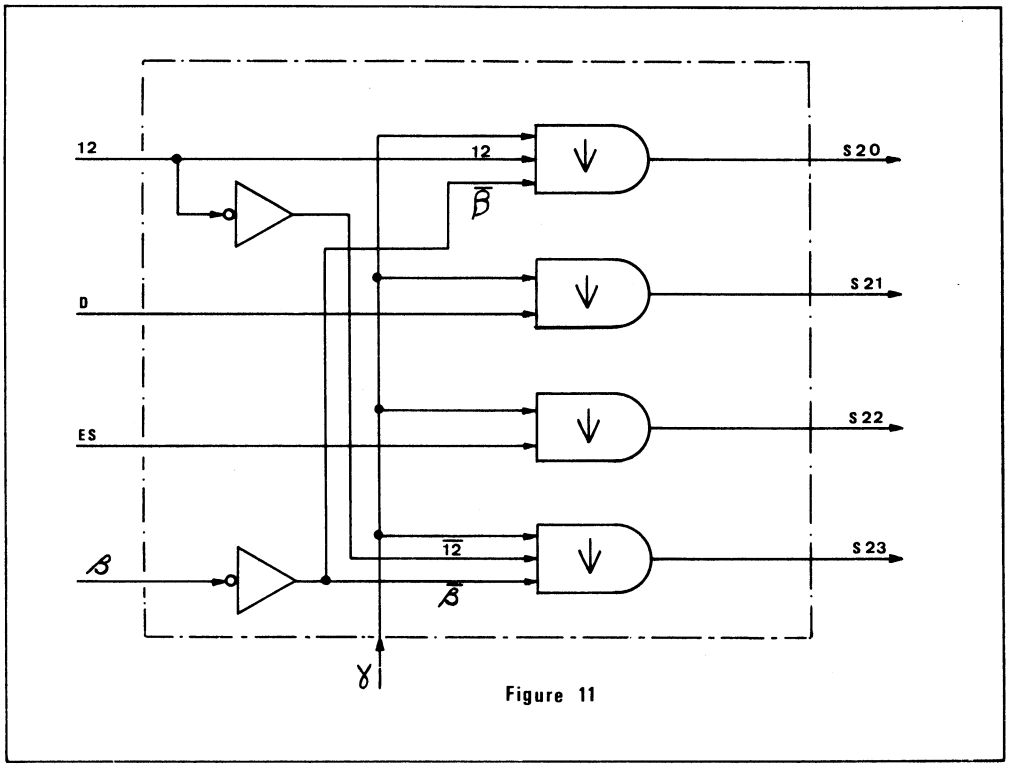


Figure 11

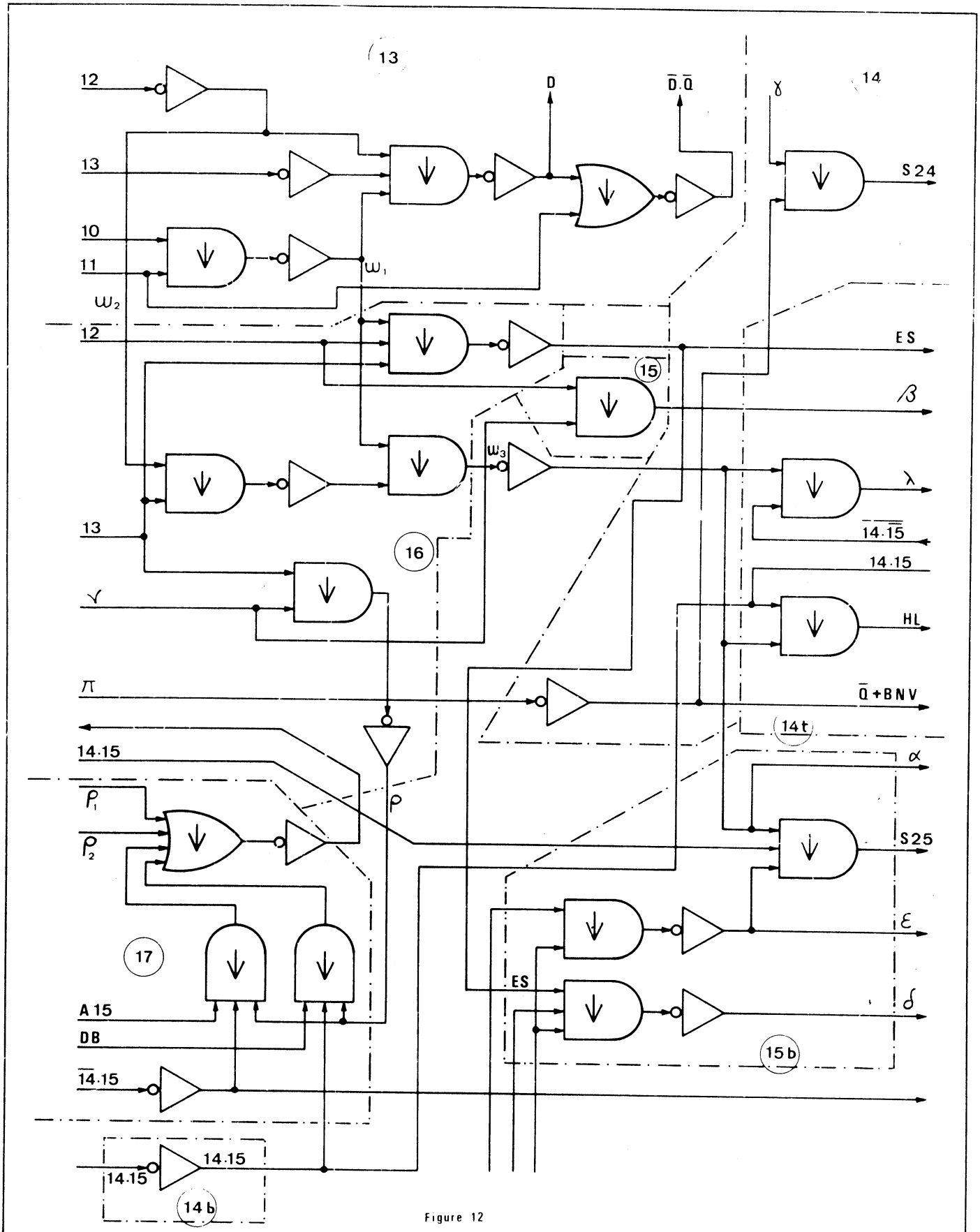

13
14
12
D
D.Q̄
γ
13
S24
10
11
ω1
ω2
12
ES
15
β
ω3
λ
13
14.15
16
14.15
HL
Y
π
Q̄+BNV
14t
14.15
α
ρ1
ρ2
ρ
S25
ε
17
ES
A15
DB
δ
14.15
15b
14.15
14.15
14b

Figure 12

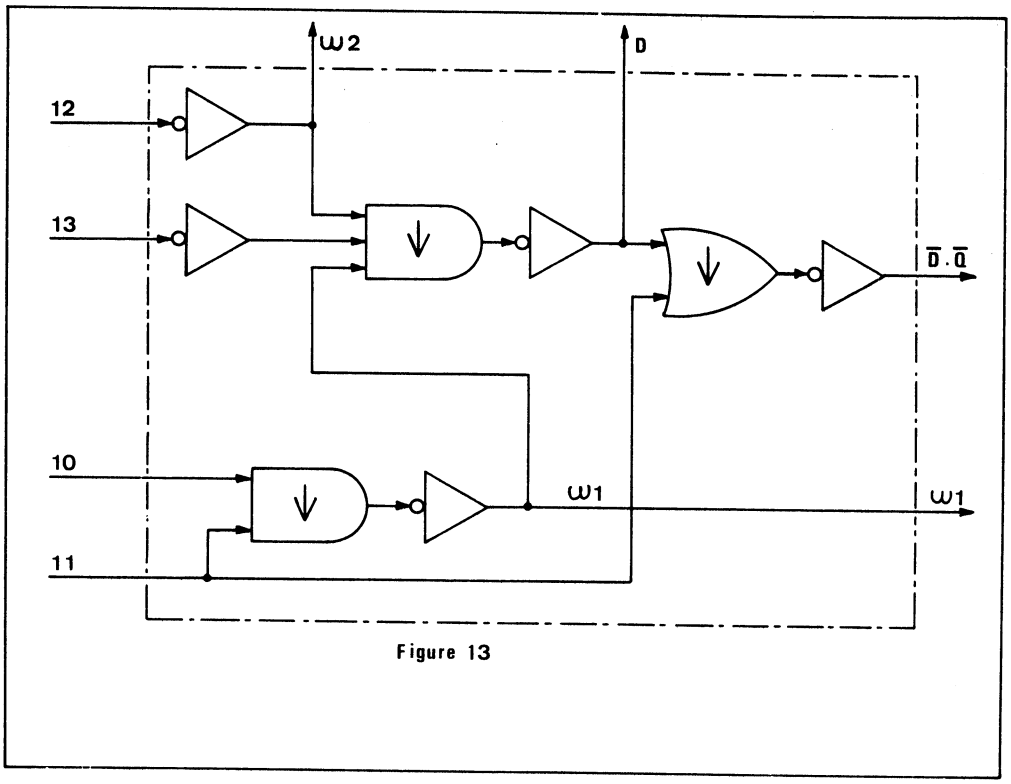


Figure 13

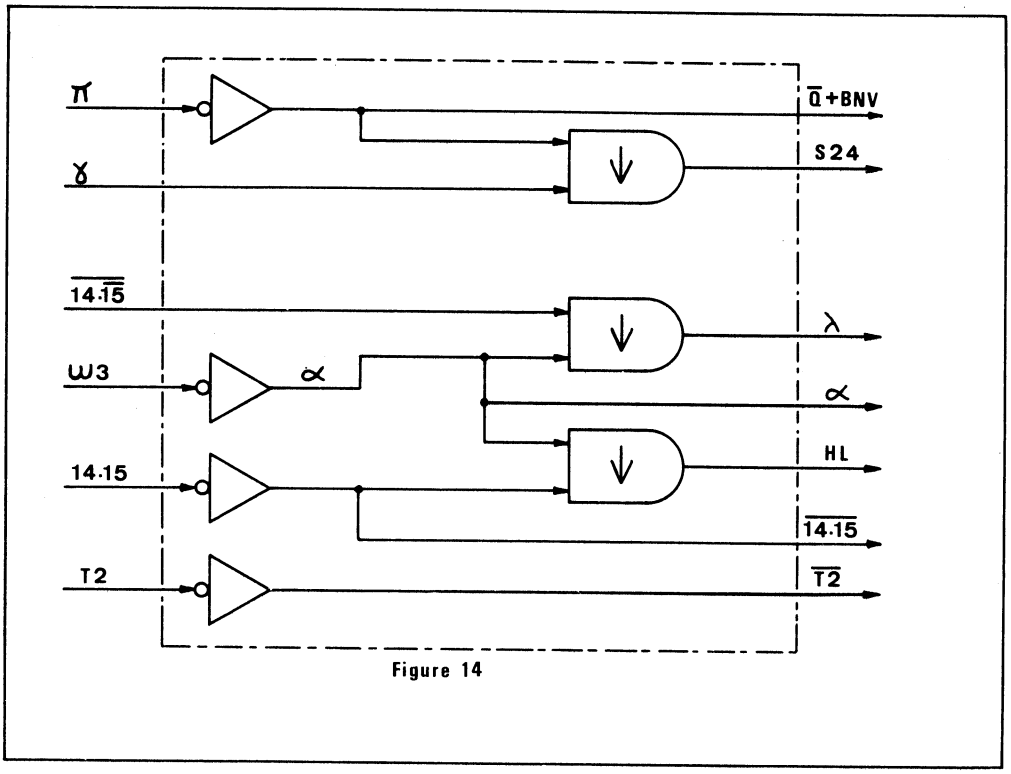


Figure 14

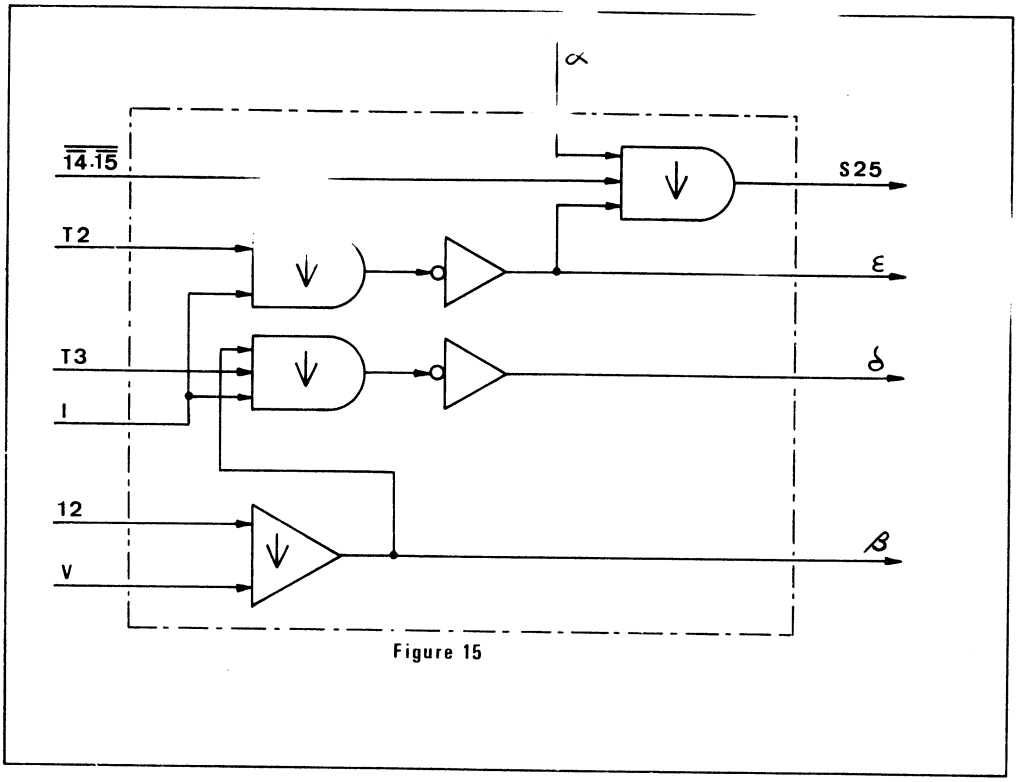


Figure 15

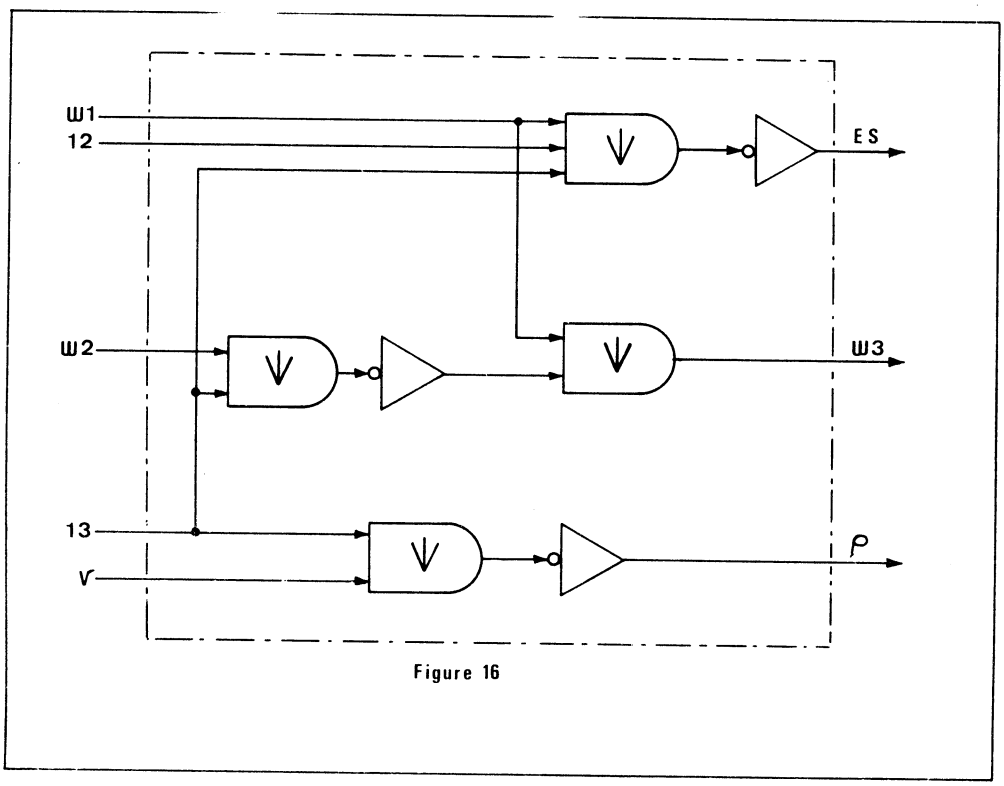


Figure 16

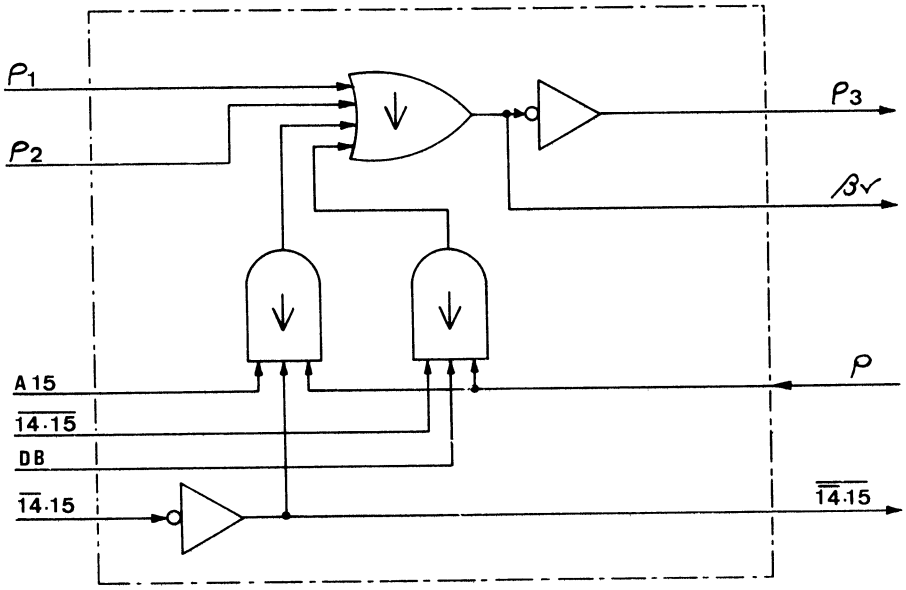


Figure 17

II-SIMULATION DES SYSTEMES LOGIQUES

Les bases de CASSANDRE ont été posées en vue de simuler le plus facilement et le plus fidèlement les ensembles logiques puisqu'il était au départ "langage pour la conception et la simulation des systèmes logiques".

L'étude préliminaire des travaux existants en collaboration avec Messieurs LETELLIER et LUSTMAN [57] a permis de dégager les caractéristiques fondamentales du langage en cours de définition.

Nous ne reviendrons pas sur les travaux étudiés en |a|, revus et complétés dans la première partie de cet ouvrage, citons simplement : "La simulation devra permettre de dépister les erreurs logiques à tous les niveaux et de vérifier que le système décrit respecte bien les clauses du Cahier des Charges.

Le langage devra, pour tenir compte de cette nécessité, être proche d'un langage de simulation, pour obtenir par une phase de traduction facile, un programme simulant le système décrit".

Plusieurs remarques sont à reprendre :

1) Les langages de programmation décrivent l'exécution séquentielle d'algorithmes sur des calculateurs qui exécutent une seule instruction à la fois. Ils sont donc par nature mal adaptés à la description d'un grand nombre d'opérations simultanées, s'exécutant sans ordre séquentiel privilégié et sous des conditions logiques complexes. En particulier Algol 60 ou FORTRAN, ne contiennent aucune notion de parallélisme, toutefois CPL [95] et B-Algol étendu [102] semblent mieux adaptés au problème.

2) D. TEICHROEW et J.F. LUBIN [97] font ressortir de leur analyse des langages généraux de simulation (SIMULA, SOL, SIMSCRIPT, CLP, CSL, GASP, GPSS ...) que l'alternative de créer son propre langage de simulation ou d'en choisir un existant subsiste, car l'implémentation de ces derniers est difficile, peut son traduisibles en ALGOL ou en FORTRAN et cette traduction, lorsqu'elle est possible, impose un étage supplémentaire de programmes par rapport à une interpretation directe.

3) La traduction du langage de simulation en un langage de programmation évolué, impose la proximité des deux langages, donc un langage de conception très simple, un peut comme le langage pédagogique de Y. CHU [22].

Elle donnera un simulateur facile à écrire mais limité et peu performant.

Dans le cas d'un langage de conception évolué et complexe, et si l'on dédire une certaine efficacité à l'exécution, une interprétation du langage s'impose en passant par l'intermédiaire d'un langage interprétable particulier à l'application envisagée.

Nous ne décrirons pas les principes des générateurs de simulateurs, ni de la simulation elle-même qui font l'objet des travaux d'autres personnes, nous illustrerons seulement les grandes options de CASSANDRE en tant que langage pour cette application, en partant d'un sous-langage très rudimentaire décrivant des machines simples (analogue au langage de SCHORR [88] ou de GERACE [34]).

A - SIMULATION D'UN LANGAGE DE TRANSFERT DE REGISTRE

1) Supprimons de CASSANDRE toutes les variables logiques de type signal, état, signal-unité, impulsion, unité.

Nous déclarerons BOOLEEN les variables restantes qui sont donc toutes de type Registre.

2) Supprimons aussi l'ordre faire et simplifions l'instruction conditionnelle en imposant à la condition logique d'être toujours un scalaire.

Le langage très simple ainsi obtenu est un sous ensemble de CASSANDRE parfaitement défini.

- C'est un langage de transfert de registres qui permet de décrire plusieurs unités indépendantes et chacunes synchronisées par une horloge qui détermine tous les chargements de bascules.

- L'arithmétique entière de CASSANDRE, qui est dérivée d'ALGOL peut être conservée.

- Nous imposerons seulement à l'utilisateur de déclarer ETATINITIAL dans chaque unité décrite, l'état dans lequel celle-ci reste lorsqu'on ne l'utilise pas et de faire en sorte de toujours se ramener à cet état à la fin de toutes les opérations de l'unité considérée.

S'il ne prend pas cette précaution, la simulation que nous allons décrire, ne pouvant fonctionner, il devra corriger sa description.

Considérons un additionneur défini par MERCER, pris comme exemple et modifié par SCHORR [88].

C'est un additionneur binaire parallèle de 16 bits possédant 3 registres propres de 16 bits M, N et A. et 4 bascules L, OF, AD et D.

Au début de l'utilisation les 2 opérandes sont dans M et A, la bascule AD est mise à 1.

A la fin de l'opération la bascule AD est remise à 0, le résultat se trouve dans M et le report éventuel dans L.

DESCRIPTION EN CASSANDRE SIMPLIFIE

```
Unité ADD(A, M, AD) ;
Booléen M(1:16), N(1:16), A(1:16), AD, D, OF, L ;
Etatinitial ETA4 ;

ETA1 : N ← M, D ← 0, OF ← 0, allera ETA2 ;
ETA2 : A ← A ≠ N, OF&N ← (N Λ A) & 0,
       D ← V/(N Λ A), allera ETA3 ;
ETA3 : si D alors (D ← 0, allera ETA2)
       sinon (M ← A, L ← OF, AD ← 0, allera ETA4) ;
ETA4 : si AD alors (L ← 0, allera ETA1)
       sinon allera ETA4 ;
```

Nous allons étudier le passage de cette version simplifiée de CASSANDRE à un programme en ALGOL 60 simulant l'additionneur décrit.

a) Les états deviennent des étiquettes, les allera qui correspondaient au chargement d'un registre d'état deviennent des allera ALGOL.

b) Les opérateurs n'existant pas en ALGOL sont traduits simplement.

c) Les opérandes de type tableau sur lesquels les opérations sont définies composante à composante sont traités par une boucle de pour ALGOL à l'aide des indices apparaissant dans les déclarations.

d) L'utilisateur met en paramètre après ADD, ceux des registres qui ne sont pas particuliers à ADD, mais aussi utilisés à l'extérieur.

e) La principale difficulté vient du parallélisme qu'il faut remplacer par une exécution séquentielle. La solution est classique, nous l'avons déjà employée pour passer de CPL où le parallélisme existe au langage MAP [67], elle consiste à dédoubler les variables et à les remettre à jour après chaque modification. Pour chaque booléen déclaré nous créerons un nouvel identificateur de même dimension en faisant précéder l'ancien de la lettre X.

Sur l'exemple précédent un premier passage de transformation donne :

```
          Procédure ADD (AD, M,A)
Booléen AD ;
Booléen tableau M, A ;
     etatinitial ETA4
     début
Booléen D, OF, L, XD, XOF, XAD, XL ;
Booléen tableau : N, XM, XN, XA (1 : 16) ;
     Etat1 : N := XM ;
             D := 0 ;
             XN := N ;
             XD := D ;
             allera Etat 2 ;
```

```
Etat2 : A :=  XA ∧ XN ∨ XA  ∧ XN ;
        OF & N := (XN ∧ XA) &O ;
        D := V/(XN ∧ XA) ;
        XA := A ;
        XOF := OF ;
        XN := N ;
        XD := D ;
        allera ETAT3 ;
Etat3 : si D alors début
           D := O ;
           XD := D ;
           allera ETAT2 ; fin sinon
         début M := XA ;
               L := XOF ;
               AD := O ;
               XM := M ;
               XL := L ;
               XAD := AD ;
               allera ETA4 ;
         fin ;
Etat4 : si AD alors début
           L := O ;
           XL := L ;
         allera ETA1 ; fin
        sinon allera Etat4 ;
```

Il convient maintenant d'initialiser et de terminer la procédure, on se sert de la déclaration Etatinitial.

La procédure commencera par une séquence d'initialisation des variables déclarées dans l'entête, terminée par allera ETAT4.

On sait qu'à la fin on doit revenir à Etat4 et rester dans cet état, il y a donc sous Etat4 un ordre allera Etat4 qui permet de reboucler l'état sur lui-même. On le transforme en Allera FINADD, pour sortir de la procédure.

D'une façon générale, au cours du premier passage on dresse la liste des états qui deviennent des étiquettes et on repère l'ordre allera ETATINITIAL placé sous l'état ETATINITIAL qui a été déclaré par l'utilisateur Etatinitial.

Au second passage on crée une étiquette FINUNITE si UNITE est le nom de l'unité étudiée et on remplace l'ordre allera repérée par allera FINUNITE.

Après un deuxième passage la description précédente deviendra

```
Procédure ADD(AD,M,A) ;
Booléen AD ; Booléen tableau M, A ;
début
Booléen D, OF, L, XD, XDF, XOF, XL, XAD ;
Booléen tableau N, XM, XN, XA (1 : 16) ;
Entier I ; XAD := AD ; Pour I := 1 pas 1 Jusqua 16 faire debut XA |I| := A |I|
          XM |I| := M |I| fin ; Allera ETAT4 ;
ETAT1 : Pour I := 1 pas 1 jusqu'à 16 faire
        N |I| := XM |I| ;
        D := 0 ; Pour I := 1 pas 1 Jusqu'à 16 faire
        XN |I| :=  N |I| ;
        XD := D ; allera ETAT2 ;
ETAT2 : Pour I := 1 pas 1 jusqu'à 16 faire
       A |I| :=   XA |I| Λ XN |I| V XA |I| Λ  XN |I| ;
       OF := XN |1|Λ XA |1| ; N |16| := 0 ;
       Pour I := 1 pas 1 jusqu'à 15 faire
       N |I| := XN |I+1|   XA |I+1| ;
       D := XN |1| Λ XA |1| V XN |2|  Λ XA |2| V...V XN |16| Λ XA |16| ;
                    XOF := OF ; XD := D ;
```

```
        Pour I := 1 pas 1 jusqu'à 16 faire début
        XA |I| := A |I| ; XN |I| := N |I| fin ;
        allera ETAT3 ;
ETAT3 : si D alors début
           D := 0 ; XD := D ; allera ETAT2 ; fin
        sinon début
        Pour I := 1 pas 1 Jusqu'à 16 faire
        M |I| := XA |I| ;
        L := XOF ; AD := 0 ; XL ;: L ; XAD := AD ;
        Pour I := 1 pas 1 Jusqu'à 16 faire
        XM |I| := M |I| ; allera ETAT4 ; fin ;
ETAT4 : si AD alors début
        L := 0 ; XL := L ; allera ETAT1 ; fin ;
        sinon allera FINADD ;
FINADD ;
        fin ;
```

On peut voir sur le programme ALGOL produit les qualités de concision de la description initiale pourtant réalisée avec des possibilités très réduites de CASSANDRE.

Nous donnerons ci-après le détail d'une addition de 2 chiffres binaires de 16 bits réalisée avec le programme de simulation ci-dessus.

B - SIMULATION EN FORTRAN DE CASSANDRE 1967.

Si l'on rajoute au langage de transfert précédent les variables de type signal on obtient un langage beaucoup plus riche [57] mais on complique beaucoup le problème de la simulation.

En effet il n'est pas possible de déterminerun ordre d'exécution séquentielle des affectations comme dans le cas où il n'y a que des variables de type registre.

Il convient bien entendu de dédoubler encore les variables, mais les valeurs des variables après un top d'horloge seront obtenues à l'issue d'un certain nombre de boucles de calcul à l'intérieur du même état de la machine simulée.

Monsieur LUSTMAN à la CFTH-HB a étudié ce problème et réalisé un programme de traduction en FORTRAN, d'une description CASSANDRE contenant des signaux [57].

C - SIMULATION SYNCHRONE GENERALE DE CASSANDRE

A la suite des travaux de Monsieur LUSTMANN, Monsieur ANCEAU et avec lui Monsieur DOUSSY réalisent, dans le cadre d'un contrat pour le compte de la CFTH-HB, un système de simulation synchrone de la version la plus récente de CASSANDRE (enrichie par les apports en particulier de Monsieur ANCEAU lui-même [4]).

Ce traitement compile CASSANDRE en un langage intermédiaire, interprété ensuite sur 360-67 [3].

D - SIMULATION ASYNCHRONE DE CASSANDRE

Si l'on introduit les opérateurs de retard dérivation et un coup on peut décrire les machines asynchrones. Leur simulation s'obtient par généralisation de la simulation précédente, mais les performances sont bien inférieures.

On est en effet amené, à peu près, à redécouper chaque temps d'horloge en autant de parties que les retards et les horloges dérivées en définissent. L'information à mémoriser est également beaucoup plus importante. Monsieur COUTURIER étudie ce problème.

E - SIMULATION D'UN ENVIRONNEMENT POUR UN SYSTEME DE PROGRAMMES - OPTIMISATION GLOBALE HARDWARE-SOFTWARE.

Une des caractéristiques des simulations précédentes est un rapport de durée dans l'exécution des opérations de la machine réelle et de son programme de simulation situé entre 10^4 et 10^7.

Cet allongement du temps rend impossible le test par cette simulation du software de la machine projettée, et par là même une optimisation globale, du matériel étudié aboutissant au meilleure comprómis software-hardware.

Pour arriver à ce but il faudrait réduire le temps de calcul des nouvelles valeurs des registres du système logique en fonction des anciennes, à moins de 50 fois ce qu'elles seront sur la machine une fois construite. Ceci n'est pas possible à partir d'une description en CASSANDRE de cette machine.

On a coutume pour traiter ce problème de définir un "langage machine" puis de programmer sur machine existante un interpréteur du langage machine du système en projet. Cette solution, jusqu'à présent sans alternative, présente à notre avis le double inconvénient d'établir à priori une séparation software-hardware, ou software-firmware(pour les machines à mémoire morte) arbitraire et sûrement très éloignée d'un optimum, et de rendre brutales et indirectes les retouches du hardware que la simulation des programmes peut amener à faire.

Seule l'existence d'une"réalisation virtuelle, du hardware" contrôlée dans les moindres détails durant l'exécution des programmes peut permettre un véritable modelage du système logique étudié.

Puisque cette "réalisation virtuelle du hardware" est trop peu performante tant qu'elle reste un programme de simulation dérivé d'une description en CASSANDRE, on peut envisager d'en donner un "modèle physique" sous forme d'un périphérique spécialisé, capable de calculer très rapidement

les fonctions logiques égales aux entrées de toutes les bascules du système simulé à un instant donné.

Prenons un exemple pour être plus clair.

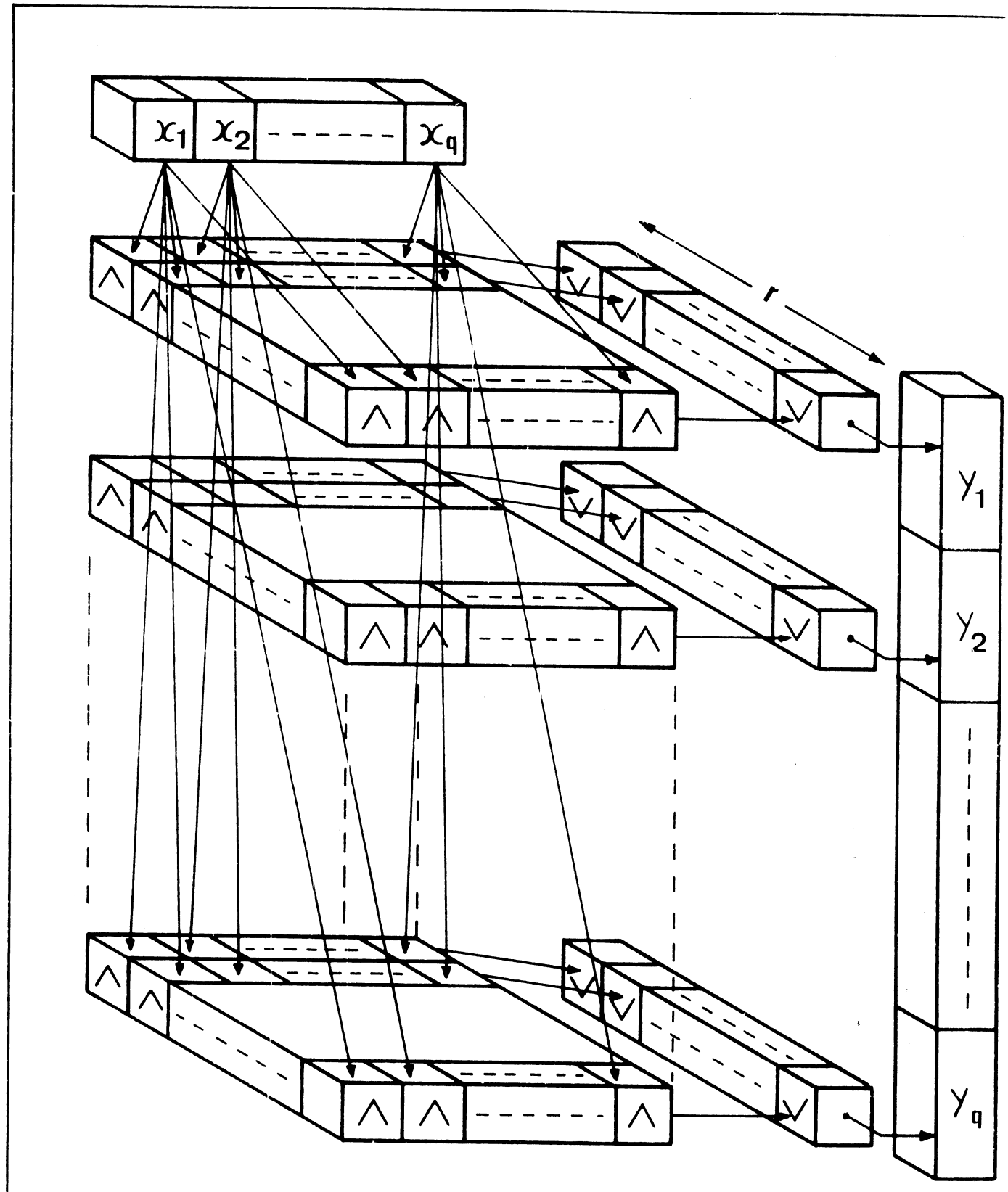
Supposons que le système étudié comporte 100 bascules (en comptant les différents registres et les bascules d'état). Une méthode de travail peut consister à transformer la description CASSANDRE initiale, par les systèmes de Monsieur LIDDELL et Madame DE POLIGNAC en CASSANDRE simplifié (Voir A).

On doit alors simuler 100 fonctions (les entrées des bascules), de 100 variables (la valeur des bascules dans le temps d'horloge précédent).

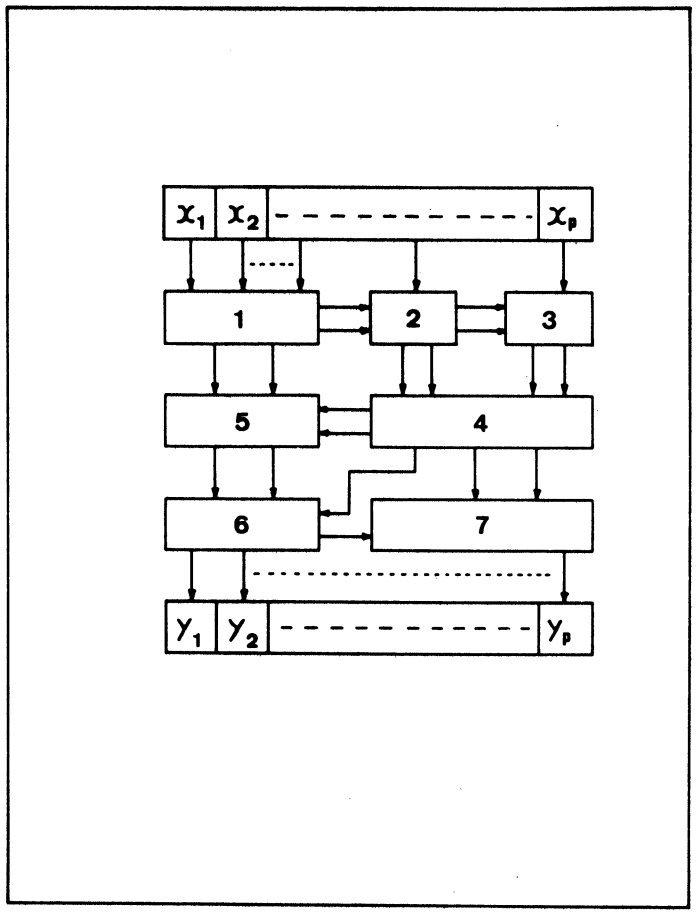
Supposons, dans la description en CASSANDRE simplifié des équations, que les seconds membres soient en sommes de monômes et que chaque fonction possède 20 monômes en moyenne.

Si la somme de 2 monômes prend p microsecondes chaque pas de la simulation durera au minimum 2000×p microsecondes, on voit que l'on dépassera le facteur 10^4 mentionné plus haut comme multiplication du temps. Appelons EB un périphérique capable d'évaluer en 1 temps d'horloge, en parallèle q fonctions (assez simples) de q variables. Un tel circuit est facile à imaginer, travaillant par exemple sur des bases premières de chaque fonction EB serait composé de q × q × r (* *) bistables, conditionnant autant de portes $\wedge$ à 2 entrées et de r× q portes $\vee$ à 2 entrées.

(* *) (r est le nombre maximum de monômes dans les formes réduites des fonctions à calculer).



E B



L'utilisation d'un opérateur tel qu'EB, demande la résolution préalable de plusieurs problèmes.

a - Il est impensable de mettre dans EB toutes les fonctions de toutes les variables de la machine simulée, soit toute sa logique. Est-il possible de partitionner cette logique en un certain nombre de morceaux dont chacun ne dépend que des précédents et dont le nombre d'entrées et le nombre sorties soient toujours inférieurs au nombre q qui intervient dans la définition de EB ?

b - Est-il possible de disposer d'une très grande mémoire de masse, à bon marché, dont l'écriture peut être très lente, mais à lecture très rapide ?

- Si l'on sait répondre affirmativement aux deux questions précédentes, on peut envisager d'écrire dans la mémoire de masse toutes les pages de logique calculées. Puis on les inscrits chacune successivement en parallèle dans EB, en même temps qu'on remplace $x_1, x_2, \ldots x_q$ par $y_1, y_2, \ldots y_q$ et on calcule les nouveaux $y_1, y_2 \ldots y_q$. Si t est le temps en microsecondes de lecture d'un mot de la mémoire dans EB et si s est le nombre de pages nécessaires, chaque pas de la simulation prendra alors à peu près 2 x t x s microsecondes. Il faudra alors étudier le prix de revient d'un tel périphérique pour voir si la construction est envisageable et le connecter à la machine sur laquelle la simulation est effectuée.

Toutes ces études sont à faire.

F - UTILISATION DU PROGRAMME ECRIT DANS A POUR SIMULER L'ADDITION DE 2 NOMBRES DE 16 BITS

écriture de résultats.

```
IEF236I ALLOC. FOR MERMET    GO          SIMUL
IEF237I PGM=*.DD CN 191
IEF237I SYSUT1   CN 192
'TRUE'   'TRUE'   'FALSE'  'TRUE'   'FALSE'  'FALSE'  'TRUE'   'FALSE'  'TRUE'
'FALSE'  'TRUE'   'FALSE'  'TRUE'   'FALSE'  'TRUE'   'FALSE'  'FALSE'  'FALSE'
'FALSE'  'FALSE'  'TRUE'   'FALSE'  'TRUE'   'TRUE'   'FALSE'  'TRUE'   'FALSE'
'FALSE'  'TRUE'   'FALSE'  'FALSE'  'FALSE'  'FALSE'  'FALSE'  'FALSE'  'FALSE'
'TRUE'   'TRUE'   'FALSE'  'FALSE'  'FALSE'  'TRUE'   'FALSE'  'TRUE'   'FALSE'
'TRUE'   'FALSE'  'TRUE'   'FALSE'  'TRUE'   'FALSE'  'FALSE'  'FALSE'  'FALSE'
'FALSE'  'TRUE'   'TRUE'   'FALSE'  'FALSE'  'TRUE'   'FALSE'  'FALSE'  'FALSE'
'TRUE'   'FALSE'  'FALSE'  'FALSE'  'FALSE'  'FALSE'  'FALSE'  'FALSE'  'TRUE'
'TRUE'   'FALSE'  'FALSE'  'FALSE'  'TRUE'   'FALSE'  'TRUE'   'FALSE'  'TRUE'
'FALSE'  'TRUE'   'FALSE'  'TRUE'   'FALSE'  'FALSE'  'FALSE'  'FALSE'  'TRUE'
'FALSE'  'FALSE'  'FALSE'  'FALSE'  'TRUE'   'FALSE'  'FALSE'  'FALSE'  'TRUE'
'FALSE'  'FALSE'  'FALSE'  'FALSE'  'FALSE'  'FALSE'  'FALSE'  'FALSE'  'TRUE'
'FALSE'  'FALSE'  'FALSE'  'TRUE'   'FALSE'  'TRUE'   'FALSE'  'TRUE'   'FALSE'
'TRUE'   'FALSE'  'TRUE'   'FALSE'  'FALSE'  'FALSE'  'TRUE'   'FALSE'  'FALSE'
'FALSE'  'FALSE'  'FALSE'  'TRUE'   'FALSE'  'FALSE'  'FALSE'  'TRUE'   'FALSE'
'TRUE'   'TRUE'   'TRUE'   'FALSE'  'TRUE'   'FALSE'  'FALSE'  'TRUE'   'FALSE'
'TRUE'   'FALSE'  'FALSE'
```

```
00003           'PROCEDURE' ADD(AD,M,A) ;
00004           'BOOLEAN' AD ; 'BOOLEAN' 'ARRAY' M,A ;
00006           'BEGIN'
00006           'BOOLEAN' D,OF,L,XD,XOF,XL,XAD ;
00007           'BOOLEAN' 'ARRAY' N,XM,XN,XA(/1:16/) ;
00008     'INTEGER' I;        XAD:=AD; 'FOR' I:=1 'STEP' 1 'UNTIL' 16
00010     'DO' 'BEGIN' XA(/I/):= A(/I/);XM(/I/):=M(/I/) 'END';'GOTO'ETAT4;
00013           ETAT1: 'FOR' I:=1 'STEP' 1 'UNTIL' 16 'DO' N(/I/):=XM(/I/) ;
00014  D:='FALSE'    ; 'FOR' I:=1 'STEP' 1 'UNTIL' 16 'DO'
00015           XN(/I/):=N(/I/) ;
00016           XD:=D ; 'GOTO' ETAT2 ;
00018     ETAT2: 'FOR' I:=1 'STEP' 1 'UNTIL' 16 'DO'
00018           A(/I/):=XA(/I/) ε ¬ XN(/I/) |¬ XA(/I/) ε XN(/I/) ;
00019           OF:=XN(/1/) ε XA(/1/) ; N(/16/):='FALSE' ;
00021           'FOR' I:=1 'STEP' 1 'UNTIL' 15 'DO'
00021           N(/I/):=XN(/I+1/) ε XA(/I+1/) ;
00022           D:=XN(/1/) ε XA(/1/) | XN(/2/) ε XA(/2/)
00022           | XN(/3/) ε XA(/3/) | XN(/4/) ε XA(/2/)
00022           | XN(/5/) ε XA(/5/) | XN(/6/) ε XA(/6/)
00022           | XN(/7/) ε XA(/7/) | XN(/8/) ε XA(/8/)
00022           | XN(/9/) ε XA(/9/) | XN(/10/) ε XA(/10/)
00022           | XN(/11/) ε XA(/11/) | XN(/12/) ε XA(/12/)
00022           | XN(/13/) ε XA(/13/) | XN(/14/) ε XA(/14/)
00022           | XN(/15/) ε XA(/15/) | XN(/16/) ε XA(/16/) ;
00023           XOF:=OF ; XD:=D ;
00025           'FOR' I:=1 'STEP' 1 'UNTIL' 16 'DO'
00025           'BEGIN' XA(/I/):=A(/I/) ;
00026           XN(/I/):=N(/I/)   'END' ;
00027           'GOTO' ETAT3 ;
00028     ETAT3;
00028    'BEGIN' OUTBOULEAN(1,OF); OUTBOOLEAN(1,D); 'FOR'I:=1'STEP'1
00030    'UNTIL'16'DO' 'BEGIN' OUTBOOLEAN(1,A(/I/)); OUTBOOLEAN(1,N(/I/));
00032    'END' ; OUTBOOLEAN(1,I); 'END' ;
00035               'IF' D   'THEN' 'BEGIN'
00035  D:='FALSE'    ; XD:=D ; 'GOTO' ETAT2 ; 'END'
00038           'ELSE' 'BEGIN'
00038           'FOR' I:=1 'STEP' 1 'UNTIL' 16 'DO'
00038           M(/I/):=XA(/I/) ;
00039  L:=XOF; AD:='FALSE'     ; XL:=L ; XAD:=AD ;
00043           'FOR' I:=1 'STEP' 1 'UNTIL' 16 'DO'
00043           XM(/I/):=M(/I/) ; 'GOTO' ETAT4 ; 'END' ;
00046        ETAT4: 'IF' AD   'THEN' 'BEGIN
00046     L:='FALSE' ; XL:=L;'GOTO'ETAT1;
00049           'END'
00049           'ELSE' 'GOTO' FINADD ;
00050        FINADD: 'END' ;
```

Ordre d'écriture injectée dans la procédure

	OF	D	A								N							
			1,2	3,4	5,6	7,8	9,10	11,12	13,14	15,16	1,2	3,4	5,6	7,8	9,10	11,12	13,14
Valeur initiale	0	0	11	00	00	00	00	01	10	00	11	11	11	10	01	11	11
1er Pas	1	1	00	11	11	10	01	10	01	00	10	00	00	00	00	11	00
2ème Pas	0	1	10	11	11	10	01	01	01	00	00	00	00	00	01	00	00
3ème Pas	0	1	10	11	11	10	00	01	01	00	00	00	00	00	10	00	00
Résultat	0	0	10	11	11	10	10	01	01	00	00	00	00	00	00	00	00
Répétition du Résultat			10	11	11	10	10	01	01	00							

Le tableau ci-dessus est obtenu en ordonnant les résultats de l'écriture précédente, en remplaçant 'true' par 1 et 'false' par 0 et en abandonnant la bascule L non significative.

On voit très clairement l'opération en 4 pas et la propagation de la retenue à chaque pas, le résultat est juste à la retenue près.

La bascule de retenue OF passe bien à 1 au 1er pas, mais ensuite revient à 0 ce qui est absurde. Ceci fait en effet apparaître une erreur dans la description (page 4) où l'ordre OF&N ← (N Λ A)&0 doit être remplacé par :

OF ← (N(1) Λ A(1)) V OF, N ← (N(2/16) Λ A(2:16))&0

Cette dernière remarque montre que la simulation est absolument indispensable même pour les problèmes les mieux connus.

III - APPLICATION A LA MICROPROGRAMMATION

On peut désigner par microopérations, les opérations élémentaires d'une machine donnée (transfert de registres, branchements de réseaux combinatoires, décalages...). Ces opérations sont représentées par les opérateurs de base de CASSANDRE. On peut alors appeler microinstruction tout ensemble de microopérations, conditionnées ou pas, compatibles entre elles et pouvant s'exécuter simultanément dans l'un des états d'une machine. (Pour la compatibilité, le parallélisme et les simultanéités, voir le chapitre III).

L'instruction CASSANDRE est donc une généralisation de la notion désormais classique de microinstruction.

De cette façon la suite d'instructions relatives à ce qui a été appelé "partie-fonctionnelle" d'une unité constitue une généralisation de la notion de microprogrammes.

Ce microprogramme plus général ne sera exécutable sur une machine donnée, que si on l'oblige à satisfaire des contraintes strictes et si l'on fixe certaines restrictions sur le langage.

Mais, qui peut le plus peut le moins, la description de microprogrammes est toujours possible, et comparé aux langages de programmation CASSANDRE apparaît pour cet usage comme un véritable langage de microprogrammation.

C'est un double champ d'applications qu'il faut alors envisager, qui correspond d'ailleurs aux deux approches actuelles de la microprogrammation par les logiciens et les spécialistes du système.

La première voie s'écarte peu de la conception du "harware" d'une machine, elle consiste à rajouter une mémoire à lecture non destructive, pour y inscrire une liste de mots de contrôle. Cette pratique rendra plus économique et beaucoup plus simple l'automate de contrôle de la machine en projet.

La deuxième voie rejoint les problèmes de langage et de software puisqu'il s'agit de la compilation d'un langage évolué (CASSANDRE) en un ensemble de microprogrammes exécutables sur une machine existant déjà. Plus exactement il s'agit d'un générateur de compilateur puisqu'il doit pouvoir s'appliquer à toute machine, décrite elle-même en CASSANDRE et vérifiant certaines propriétés.

A - AIDE A LA CONCEPTION DE MACHINES CONTROLEES PAR DES MICROPROGRAMMES

De même que l'on étudie un système d'aide à la synthèse et l'implantation des circuits logiques de la machine en projet (Chapitre I précédent) on peut essayer de constituer un ensemble de programmes et procédures, utilisé en conversation homme-machine et permettant, à partir d'une description du projet en CASSANDRE, d'en donner une réalisation dont la partie contrôle est dirigée par des microinstructions inscrites dans une mémoire convenable.

Un tel processus de conception aidée aurait 4 phases principales.

a) Détermination à partir de la description des modules correspondants aux fonctions booléennes utiles.

b) Optimisation par le concepteur au vu des résultats de la phase précédente et éventuellement rebouclage sur celle-ci.

c) Détermination du format du micromot.

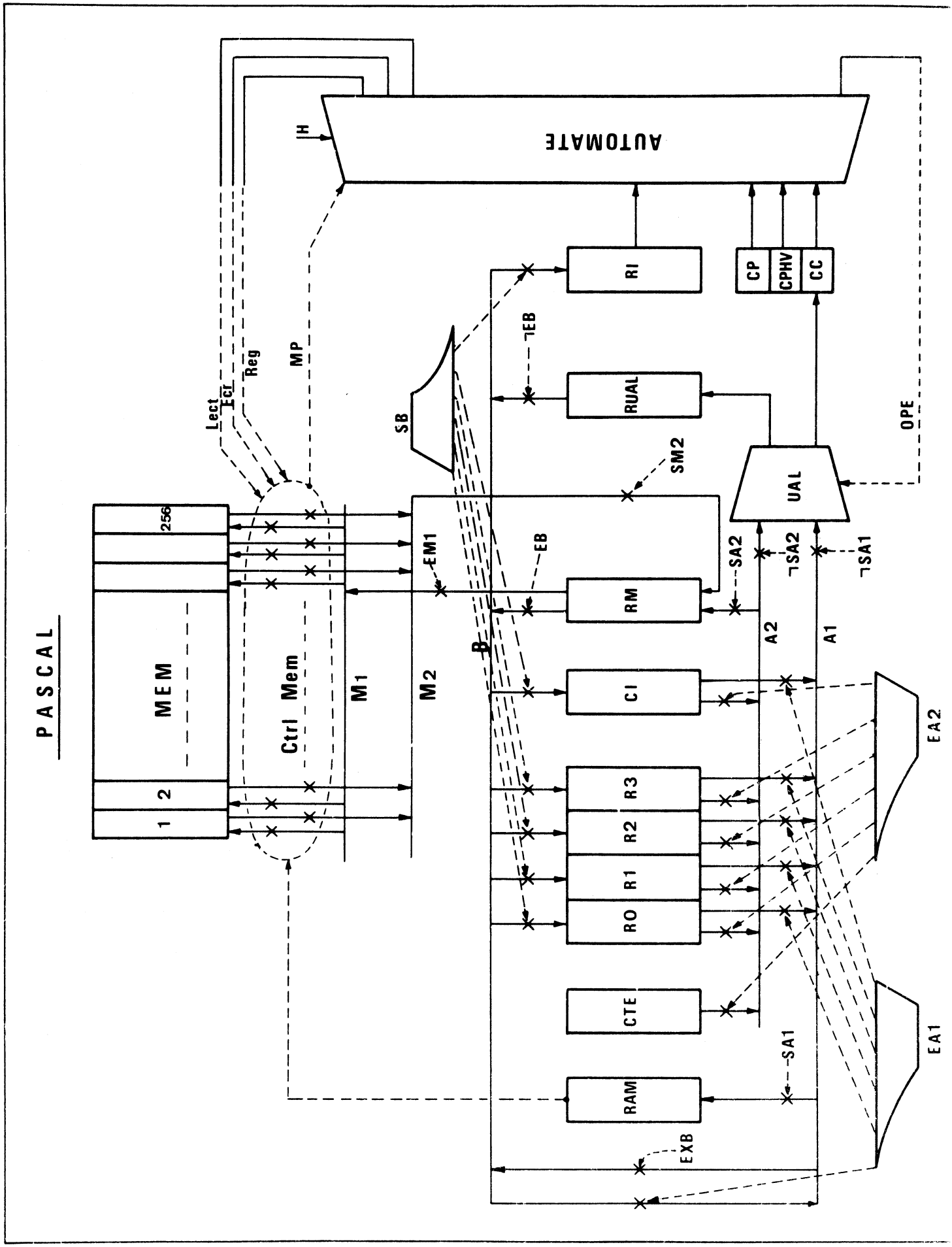
d) Codage des microprogrammes et synthèse des modules fonctionnels utilisés.

Nous n'aborderons pas plus une telle étude envisagée par Monsieur ANCEAU avec la collaboration de Madame de POLIGNAC et de Monsieur CHATELIN.

Nous donnerons simplement un exemple de description en CASSANDRE d'une telle machine pouvant servir de point de départ à un tel processus.

B - DESCRIPTION DE PASCAL

```
Unité PASCAL(H,      ;      );
Registre R0(0:0,1:8),R1(0:0,1:8),R2(0:0,1:8),R3(0:0,1:8),
CI(0:0,0:8) ,RI(0:0,1:8),CTE(0:0,1:8),CPHI,CP,CC(0:2),RM(1:8),
RAM(1:8),RUAL(1:8,MEM(1:8,1:256);
Externe UAL
  AUTOMATE
Signal B(0:0,1:8),ERA1(1:6,1:8),ERA2(1:6,1:8),SRB(1:6,1:8),
A1(1:8),A2(1:8),M1(1:8),M2(1:8),OPE(1:4),EA1(1:3),EA2(1:3),
SB(1:3),SM1(1:8),EM2(1:8),E1UAL(1:8),E2UAL(1:8),S1UAL(1:8),
S2UAL(1:3),MP,SA1,SA2,EB,SM2,REG,LECT,ECR,EM1,EXB;
Impulsion H;
Mémoire MEM;Horloge H;Bus A1,A2,M1,M2,B;
UAL(E1UAL,O,P,E,E2UAL;S1UAL,S2UAL);
AUTOMATE(CP,CPHI,CC,RI,MP;LECT,REG,ECR,...)
Si LECT alors EM2 := RAM;
Si ECR V REG alors SM1 := RAM;
M2 := MEM(, ⊥ EM2);
SM2 := LECT;
EM1 := REG V ECR;
Si EM1 alors M1 := RM;
```


PASCAL
AUTOMATE
H
MP
Reg
Ecr
Lect
MEM
256
2
1
Ctrl Mem
M1
M2
SB
B
EM1
EB
¬EB
RI
RUAL
RM
CI
R3
R2
R1
RO
CTE
RAM
EXB
SM2
SA2
¬SA2
¬SA1
SA1
A2
A1
UAL
CP
CPHV
CC
OPE
EA2
EA1

ERA_1 := B&R_0&R_1&R_2&R_3&CI ;

ERA_2 := CTE&R_0&R_1&R_2&R_3&CI;

SI EB alors B := RM sinon B := RUAL;

Si EXB alors B := A_1;

Si ¬ SA_1 alors E_1UAL := A_1;

A_1 := ERA_1(⊥ EA_1,) ;

A_2 := ERA_2(⊥ EA_2,) ;

SRB(⊥ SB,) := B;

Si ¬ SA_2 alors E_2UAL := A_2;

Si ¬ SA_2 alors E_2UAL := A_2;

<H> CC ← S_2UAL,RUAL ← S_1UAL,

MEM(, ⊥ SM_1) ← M_1,R_0&R_1&R_2&R_3&CI&RI ← SRB,

si SM_2 alors RM ← M_2 sinon si (SA_1 ∧ ¬ SM_2) alors RM ← A_2,

si SA_1 alors RAM ← A_1;

FETCH : CP_1 : SA_1 := 1;EA_1 := (110);

<H> si MP alors → ;

CP_2 : LECT := 1;

<H> CI ← +1, si MP alors → ;

CP_3 : REG := 1,EB := 1,SB := (110);

<H> si MP alors allera EXEC : CP_1 ;

EXEC : CP_1 : EA_1 := RI(4:5) ; EA_2 := RI(6:7);

SA_1 := 0;SA_2 := 0;OPE := RI(0:3);

<H> → ;

CP_2 : EB := 0;SB := RI(4:5);

<H> CPHI ← 1, allera FETCH : CP_1;

REMARQUES SUR LA DESCRIPTION PRECEDENTE

1) Les déclarations et les connexions de la page 43 et 45 sont strictement équivalentes au schéma de la page 44. Des signaux intermédiaires ont été créés entre les bus et les registres.

2) Deux bus M_1 et M_2 ont été associés à la mémoire, il faut noter que cette représentation simplifie le schéma en enlevant à la mémoire son rôle à part. Ceci n'est malheureusement jamais fait sur les schémas de ce type et en particulier on trace une liaison RAM-MEM qui est absurde puisque jamais l'adresse contenue dans RAM ne rentre ou ne sort de la mémoire, elle est simplement décodée dans un bloc logique qui contrôle les transferts entre MEM et ses bus M_1 et M_2.

3) Il faut accentuer la remarque précédente en distinguant les deux catégories entièrement différentes de fils qui sont en général confondues sur le schéma. Ceux qui servent effectivement aux transferts de l'information (pleins) et ceux qui sont des fils de contrôle et qui ne transportent que des commandes.(Pointillés). Seuls le graphe des premiers est à considérer dans le problème qui nous intéresse.

4) On pourrait simplifier la description en créant une unité de plus CONTROLE MEMOIRE, pointillés) qui aurait en entrée la sortie de RAM, les fils LECT, ECR, REG, les sorties de MEM et les sorties de M_1 et en sortie les entrées de MEM, de M_2 et le signal MP. Ce bloc est partiellement décrit pages

5) La seule propriété qui pourrait distinguer la Mémoire des registres (indépendamment de son temps de lecture-écriture) réside dans le fait qu'on peut écrire un seul mot à la fois. Mais c'est du à l'adressage qui active un seul fil de sortie à la fois du décodeur d'adresse.

Tout ensemble de registres adressés par un même décodeur aura la même propriété. C'est pourquoi en CASSANDRE nous avons toujours déclaré la mémoire comme registre. La spécification mémoire ne sera utile que pour quelques applications précises.

6) La page 45 décrit les 2 phases d'une instruction à 2 adresses. On voit qu'après la description détaillée des pages 43et45 et les choix faits, la description de ces instructions est une suite de microinstructions presques codées, il reste peu à faire pour déterminer les micromots.

C - PASSAGE D'UN BLOC-DIAGRAMME DE CALCULATEUR MICROPROGRAMME A UN RESEAU DE NOEUDS ET D'ETOILES.

a) Connexion de signaux.

1-a) Un signal est connexe à lui-même.

2-a) Deux signaux sont connexes s'ils sont respectivement en partie gauche et en partie droite d'une connexion (:=).

3-a) S'ils sont connexes à un même $3^{ème}$.

b) Relation d'équivalence.

Deux signaux sont dans une même classe

1-b) S'ils ne sont pas des bus.

2-b) S'ils sont connexes.

Les 3 propriétés de la relation d'équivalence sont évidemment vérifiées réflexivité (1-a), symétrie (2-a), transitivité (3-a).

PROPRIETE

- Les classes ainsi formées seront appelées noeuds-booléens (ce sont en général des fonctions booléennes multiples de plusieurs variables).

- Si l'on décide de mettre dans une même catégorie que l'on peut appeler "étoiles" les registres et les bus de la machine, on obtiend un réseau noeuds-étoiles orienté d'articulations.

- Montrons-le :

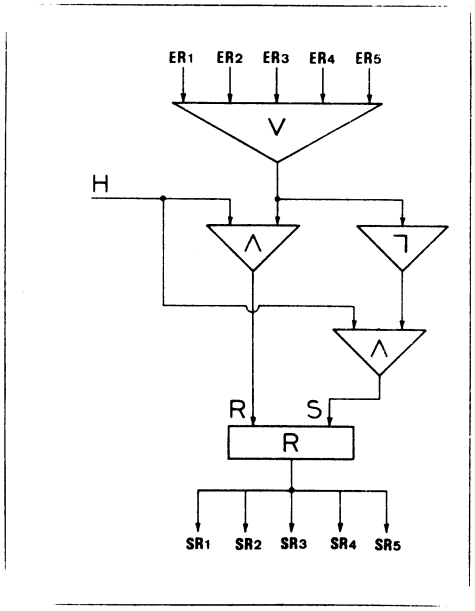
Les variables de type registre ne jouent un rôle différent des signaux que lorsqu'elles se trouvent en partie gauche d'une affectation.

En partie droite (dans une expression booléenne) toutes les variables sont des niveaux et peuvent être considérées comme des signaux, même lorsque l'identificateur qui les représente a été déclaré registre.

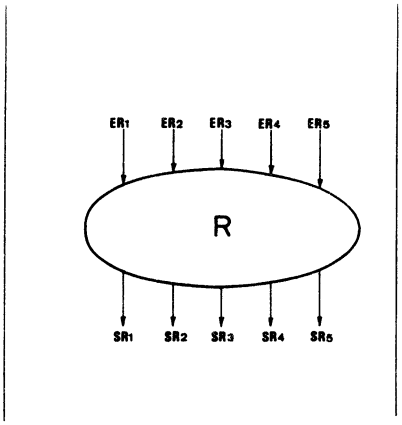
Pour ne pas mettre dans une même classe d'équivalence toutes les fonctions qui ont en commun au moins une variable nous allons créer un signal nouveau correspondant à la sortie d'un registre, à chaque occurence de ce registre en partie droite d'une affectation. Un registre aura donc autant de sorties que d'occurences en partie droite dans la description CASSANDRE.

- Nous faisons de même pour l'entrée d'un registre en créant un nouveau signal connecté à l'entrée du registre chaque fois que son nom apparaît en partie gauche d'un chargement de registres.

- Ceci pourrait se traduire en hardware par le schéma suivant :



En réalité seule la vue macroscopique comme "étoile" nous intéressera, où toutes les entrées-sorties sont de type signal (pour appliquer l'équivalence précédente).



Pour rendre légitimes toutes les opérations nous énonçons la règle :

"A un instant donné peuvent être activées soit une seule entrée d'une étoile de type registre soit toutes les sorties de cette étoile".

- Cette règle reste vraie pour les étoiles de type "bus".

- Les articulations de type "noeud" ont aussi des entrées et des sorties car les relations := définissent une orientation du réseau.

- les entrées d'un "noeud" ne se trouvent qu'en partie droite d'un :=, les sorties qu'en partie gauche.

CONSEQUENCES DES DEFINITIONS CI-DESSUS

a) 2 noeuds ne sont jamais adjacents.

b) 2 registres ne sont jamais adjacents.

c) Entre 2 bus ou 1 bus et 1 registre il y a toujours au moins 1 porte donc 1 signal donc 1 noeud.

d) Le réseau d'articulations ainsi défini est un réseau noeuds-étoiles.

e) Les noeuds correspondent aux plus petites fonctions booléennes que l'on peut considérer comme indépendantes dans le problème traité. On pourra donc se contenter de les optimiser séparément.

f) Les noeuds contiennent toutes les fonctions que l'on peut utiliser à un instant donné dans l'exécution des microinstructions.

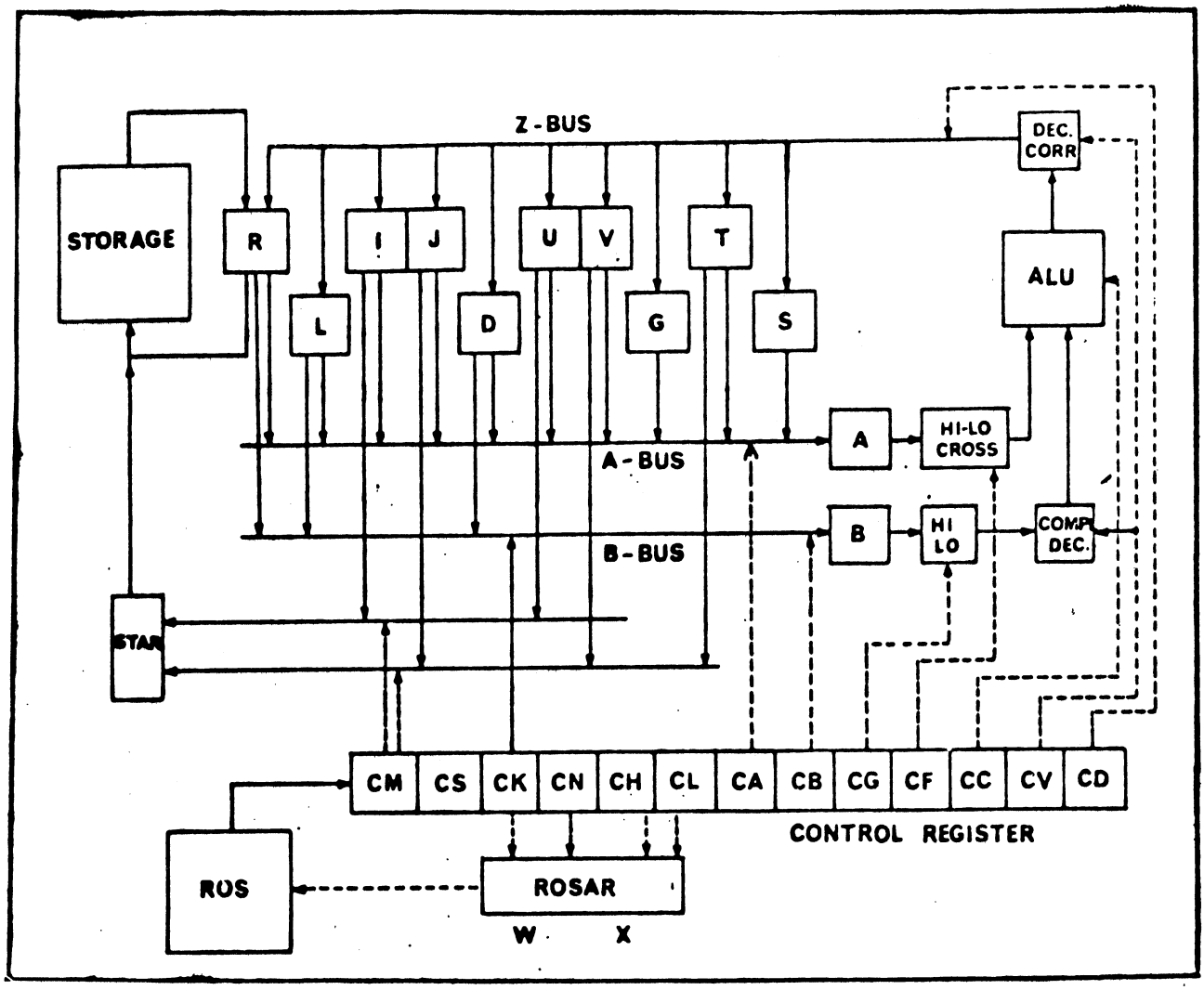
g) Les étoiles et leur règle d'utilisation fixent la compatibilité entre ces fonctions et les contraintes de parallélisme et de simultanéité.

D - BLOC-DIAGRAMME DU X2030 |WIRTH-WEBER| (102).

Unité X2030 ( );

Registre STOR(0:7,0:65535),ROS(1:47,0:255),
STAR(1:16),ROSAR(1:8),CM(0:2),CS(0:3),CG(0:1),CK(0:7),CN(0:3),
CH(0:3),CL(0:3),CA(0:3),CB(0:1),CF(0:2)CC(0:2),CV(0:1),CD(0:3),
R(0:0,1:8),L(0:0,1:8),D(0:0,1:8),I(0:0,1:8),C(0:1),RVD(0:0),SEGO(0:0,1:1),
J(0:0,1:8),U(0:0,1:8),V(0:0,1:8),T(0:0,1:8),G(0:1,1:8),S(0:1,1:8);

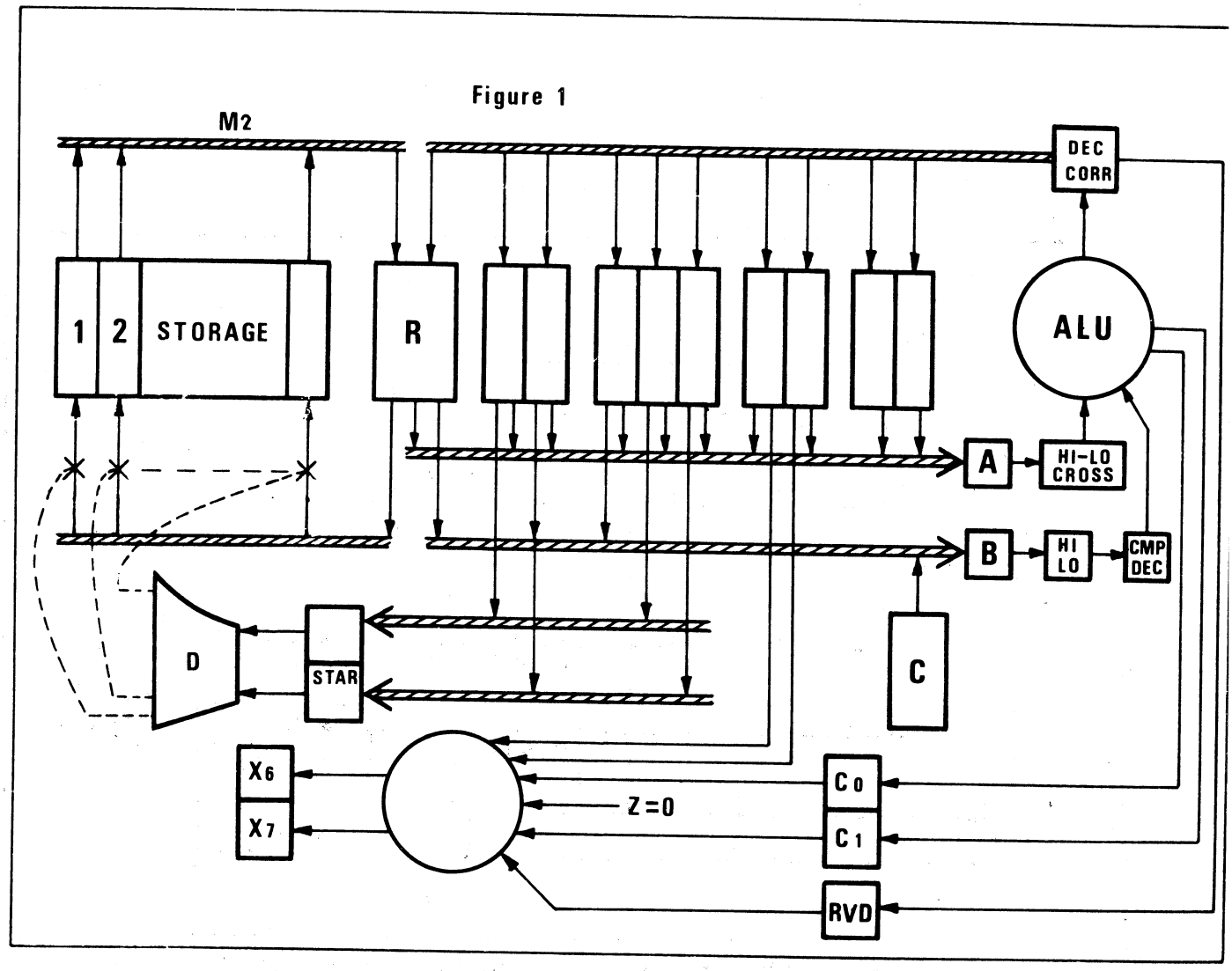
Externe A(1:8,1:8),B(1:8,1:8),
HI-LO-CROSS((0:2,(0:7);(0:7)),HI-LO((0:1),(0:7);(0:7)),
COMP-DEC((0:0),(0:1),(0:7);(0:7)),DECCORR((0:1),(0:7);(0:7)),
ALU((0:7),(0:7),(0:0),(0:2);(0:7));
Signal A(0:7),B(0:7),Z(0:7),RC(1:47),X_6,X_7,X(6:15,0:8),X_1(0:3,0:8),
X_2(6:15,0:8),X_3(1:5,1:16),Q(0:8),W(0:8),M_1(0:8),XX_1(0:1,1:16),XX_2(0:15,0:7)
XX_3(0:15,0:7),XX4(0:9,0:7)),M_2(0:8);
Mémoire STOR,ROS;Bus A,B,Z,C,D,M_1,M_2;Horloge H;



Le schéma précédent est celui de l'article bien connu de Wirth auquel nous empruntons l'exemple.

- Notons qu'ici les fils de transfert d'information sont en plein, les commandes en pointillé, la distinction que nous pensons souhaitable est ainsi faite, mais le registre d'adresse mémoire est toujours relié à STORAGE par un trait plein et la mémoire elle même relié à R sans intermédiaire.

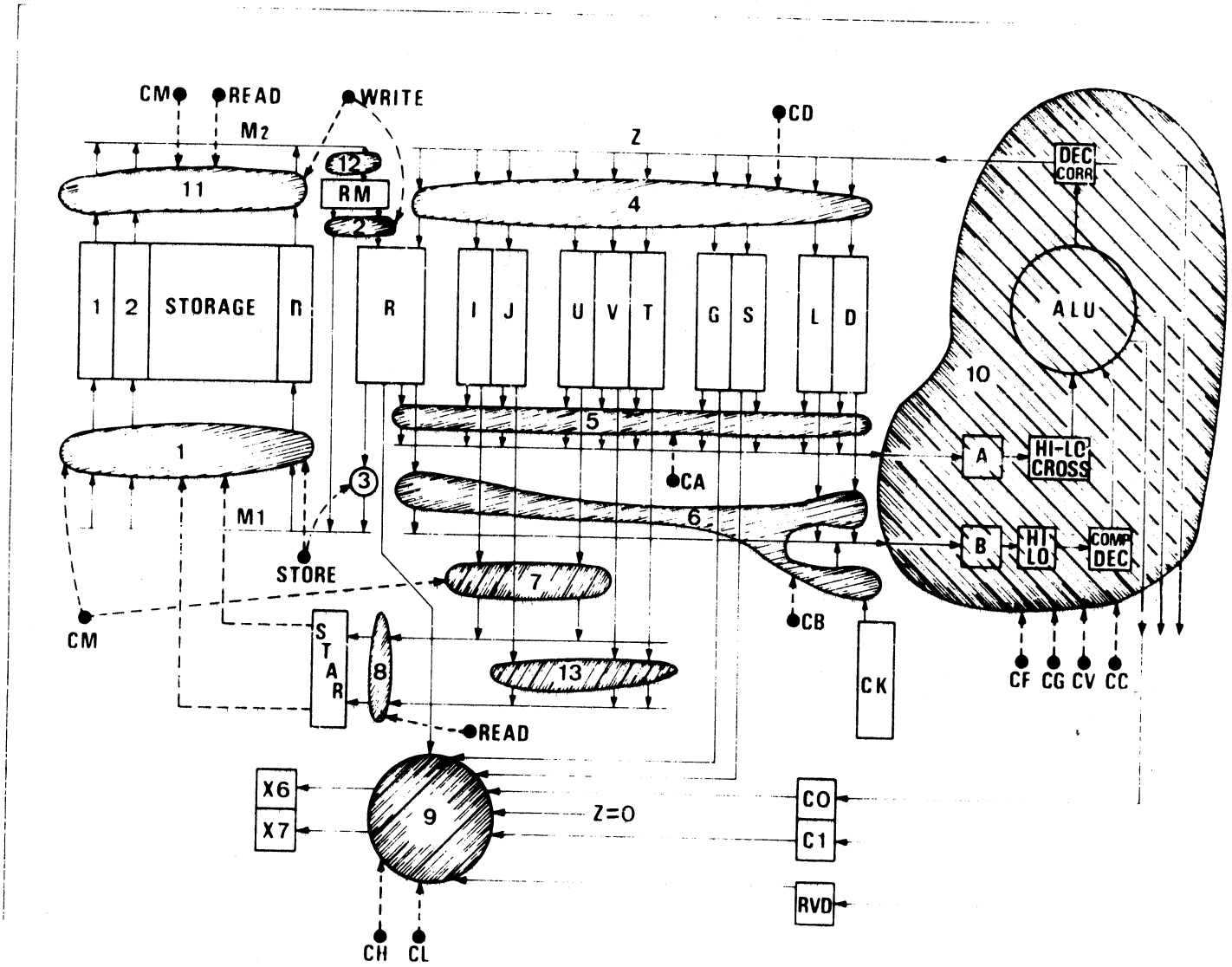
Figure 1



- Nous ne décrirons pas tout le tableau, maintenant que le principe est illustré avec CH,CL,CM,CA et CD.

Chaque symbole ⊥ dénote la présence d'un décodeur dont 1 seul fil de sortie à la fois est à 1. Cela signifie dans ce cas particulier que l'on pourra lire ou écrire un seul registre à la fois (comme pour les mots de mémoire). Pour l'écriture ce n'est évidemment pas obligatoire en principe.

- L'application de la relation d'équivalence précédemment définie à cette description, génère le réseau de noeuds et d'étoiles suivants :



Sur le schéma modifié deux registres appartenant à la partie contrôle du X2030 sont en réalité relié par des traits pleins à la partie opératoire, ce sont CK et X(6:7), nous les gardons sur le schéma, mais le reste du contrôle ne nous intéresse que par les commandes (pointillés) qu'il génère.

Nous rajoutons un registre de sortie de mémoire RM(0:8).

Ce schéma permet entièrement l'écriture des tables de codage (page 17 article Wirth).

XX_1(0,):=0&1&R(,0)&0&1&0&C(,0)&S(,1)&S(,0)&S(,2)&S(,4)&S(,6)&G(,0)&G(,2)
&G(,4)&G(,6)

XX_1(1,):=0&1&R(,1)&1&0&ZEGO(,1)&C(,1)&RVD&S(,1)&S(3)&S(,5)&S(,7)&G(,1)
&G(,3)&G(,5)&G(,7)

<H>X(6:7)←XX_1(0:1,←(CH&CL));

XX_2(,0):=¬ε(16);XX_2(,1):=I&J;XX_2(,2):=U&V;XX_2(,3):=¬ε(8)&T

XX_2(,4):= ε(8)&V;XX_2(,5):= ε(8)&T:XX_2(,6):=STAR;XX_2(,7):=STAR:

Q&W:=XX_2(,⊥CM);

<H>si STORE alors MEM(,⊥STAR)←M_1 sinon si WRITE alors (R←RM.

MEM(, STAR)←M_1)sinon si READ alors STAR←Q W, RM←M_2;

WRITE := (CM=11);

READ := (CM=001) V (CM=010) V (CM=011) V (CM=1000) V (CM=101);

STORE := (CM=111);

si READ alors M_2 := MEM(,⊥STAR) sinon

si WRITE alors M_1 := RM sinon si STORE alors M_1 := R;

XX_3:= (1,8)&φ(1,8)&φ(1,8)&φ(1,8)&φ(1,8)&φ(1,8)
S&R&D&L&G&T&V&U&J&I;

A :=XX_3(⊥(A,);

<H>si∧/(CD≠(0000)) alors S&R&D&L&G&T&V&U&J&I ← XX_4;

XX_4(⊥CD,) := M_2

etc...

On peut associer une matrice à un tel réseau en mettant en ligne des étoiles et en colonne des noeuds 8

Cette matrice peut être commodément scindée en 4 sous matrices en distinguant.

1) Registres et mots mémoire

2) Bus

3) Noeuds fonctionnels

4) Noeuds de simple transfert (portes et logique de commande).

	1	2	3	4	5	6	7	8	11	12	9	10
Mem 1	1								-1			
Mem 2	1								-1			
Mem 3	1								-1			
Mem 4	1								-1			
R		1	-1	1	-1	-1					-1	
I				1	-1		-1					
J				1	-1		-1					
U				1	-1		-1					
V				1	-1		-1					
T				1	-1		-1					
G				1	-1						-1	
S				1	-1						-1	
L				1	-1	-1						
D				1	-1	-1						
STAR								1				
RM		-1								1		
X											1	
C-RVD											-1	1
CK												
A					1							-1
B						1						-1
C							1	-1				
D							1	-1				
Z				-1								1
M1	-1	1	1									
M2									1	-1		

Pour obtenir une telle matrice il a fallu couper plusieurs noeuds pour éviter qu'ils aient en commun avec une étoile une entrée et une sortie. Ainsi le noeud 1, a produit (1,11,12). On repère alors la sortie d'une étoile par un 1. Toutes les autres composantes sont mises à 0.

Un chargement de registre par exemple G ← U+R suppose alors le départ de 2 registres U et R, le transfert par plusieurs noeuds, le passage par des bus, l'opération + dans un noeud fonctionnel et le retour à un registre G.

Ceci se traduit par des "chemins" possibles dans la matrice précédente en allant d'un 1 à un -1 et réciproquement.

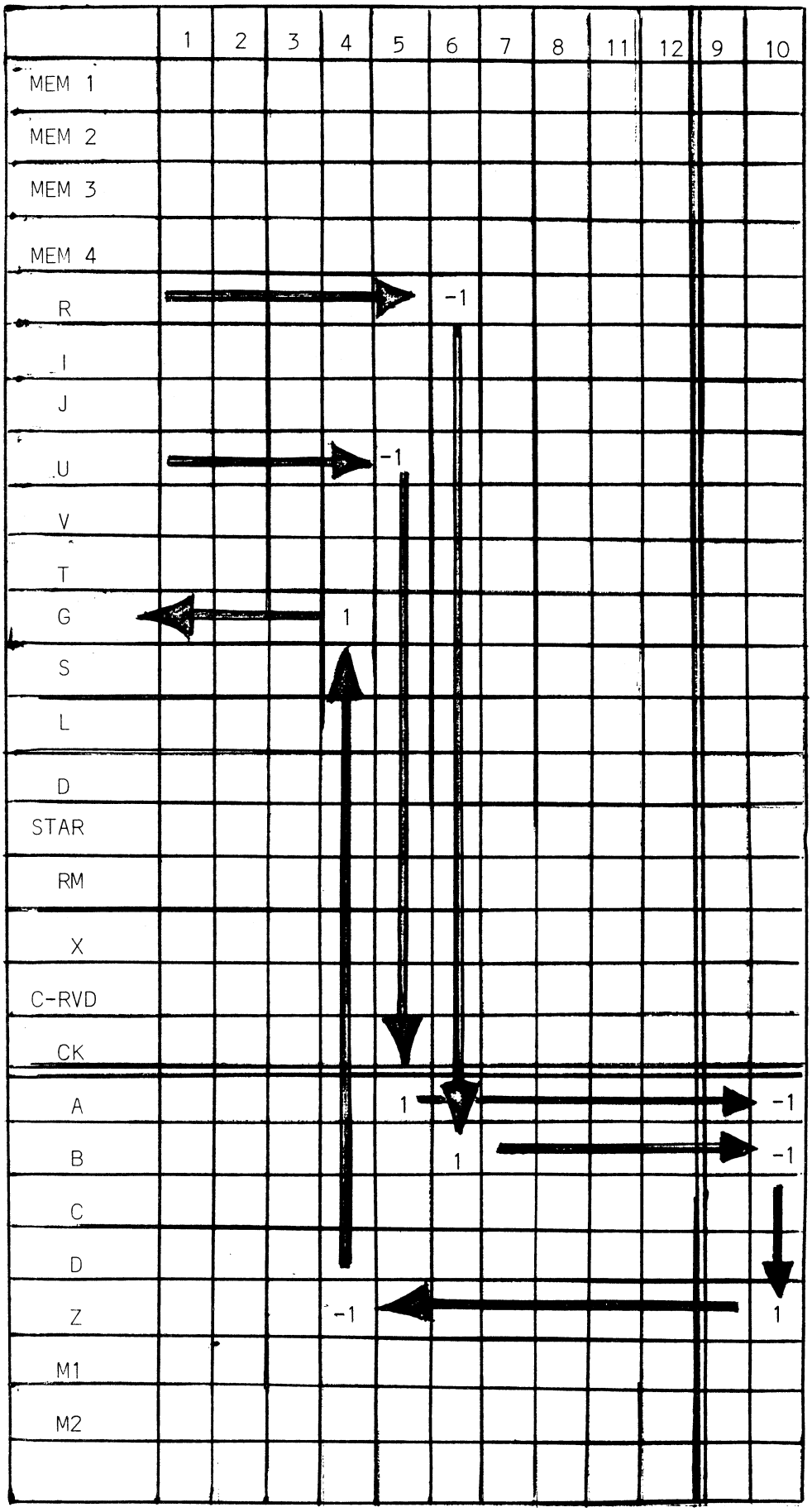
Les règles énoncées dans les chapitres précédents ont les conséquences suivantes.

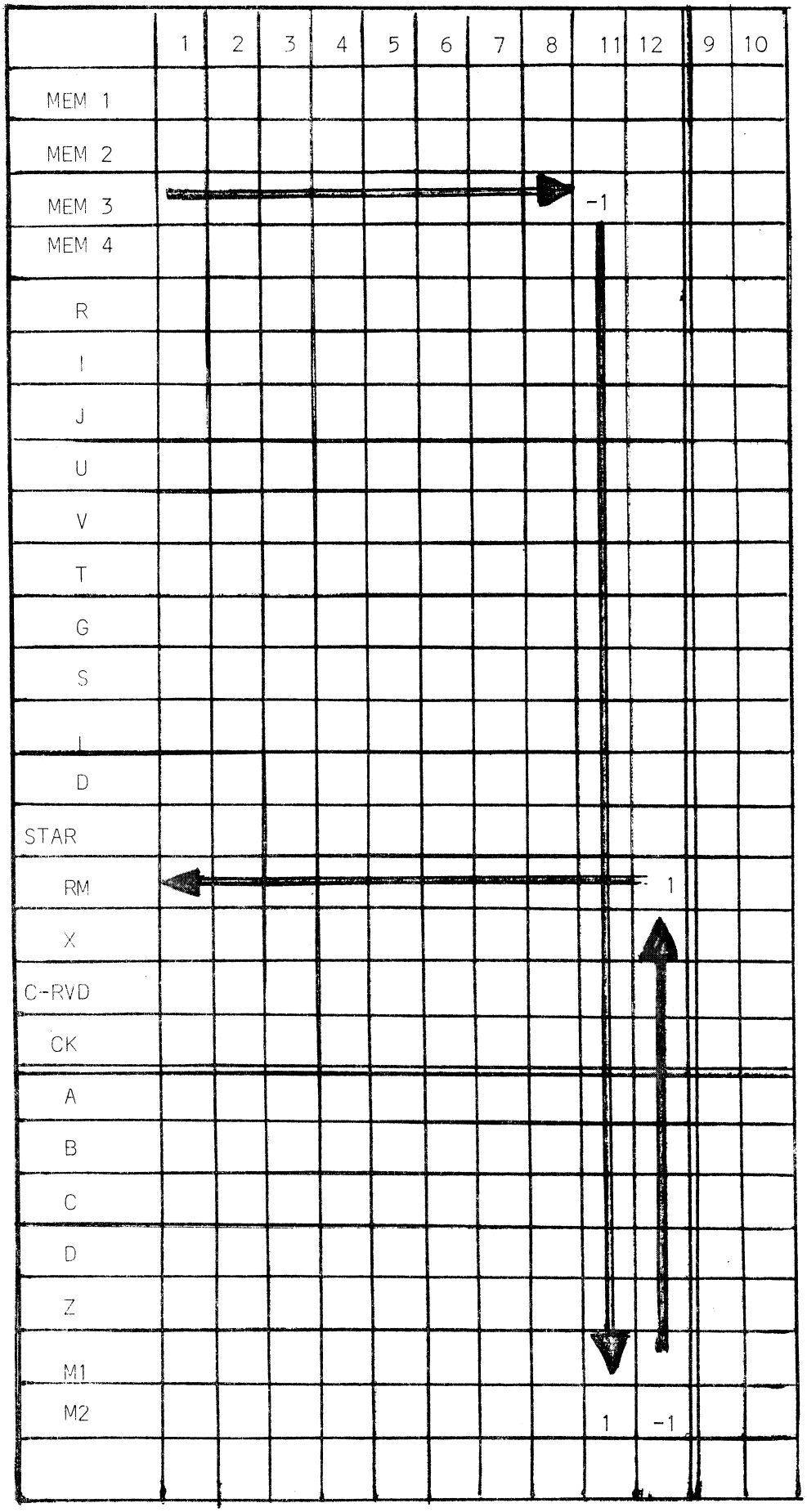
1 - La matrice représentative des chemins doit être "compatible" avec la matrice principale (leur "ou logique" doit être égal à la matrice principale).

2 - Dans la colonne d'un noeud de transfert le nombre des 1 et des -1 doit être le même.

3 - Dans les lignes, il ne peut y avoir qu'un seul 1.

La matrice suivante donne un exemple de réalisation de l'opération G ← R+U.


1
2
3
4
5
6
7
8
11
12
9
10
MEM 1
MEM 2
MEM 3
MEM 4
R
I
J
U
V
T
G
S
L
D
STAR
RM
X
C-RVD
CK
A
B
C
D
Z
M1
M2
-1
-1
1
1
-1
-1
1
-1
1


1
2
3
4
5
6
7
8
11
12
9
10
MEM 1
MEM 2
MEM 3
MEM 4
R
I
J
U
V
T
G
S
L
D
STAR
RM
X
C-RVD
CK
A
B
C
D
Z
M1
M2
-1
1
1
-1

Si maintenant on veut représenter LECTURE(MEM(,3)) on obtient les chemins de la figure qui précède.

Les deux opérations peuvent s'éxécuter en même temps on en déduit la définition suivante.

"Deux microopérations sont compatibles si l'intersection de leur matrice associée est nulle et leur union logique compatible avec la matrice principale".

E - REMARQUES SUR LA COMPILATION DE MICROPROGRAMMES DECRITS EN CASSANDRE EN MICROINSTRUCTIONS D'UNE MACHINE DONNEE.

a) La version de CASSANDRE utilisée pour décrire des microprogrammes sera certainement une version restreinte pour deux raisons. D'abord les notions nécessaires à la description des algorithmes à microprogramme sont moins nombreuses que celles qu'il faut pour définir entièrement un système logique.

Dans le cas où nous nous plaçons, nous travaillons sur une machine dont la structure est fixée, tous les registres sont connus et on ne peut pas rajouter une seule porte logique au hardware. Par exemple les opérateurs $\lceil^{\rightarrow}$, $\lfloor^{\uparrow}$ et τ sont inutiles puisque les microprogrammes ont un déroulement synchrone. De même les notions arithmétiques qui n'ont qu'un rôle descriptif en CASSANDRE devront disparaître au cours d'un prétraitement.

Ensuite parmi les notions restantes du langage il faudra imposer les restrictions d'utilisation pour que les microprogrammes aient un sens : par exemple le symbole $\perp$ sera strictement limité à l'adressage des mémoires. Ses seules occurrences possibles seront donc du type MEM(, $\perp$A) (où A est un registre susceptible de contenir une adresse).

b) La description en CASSANDRE sera constituée d'une unité.

La partie indépendante du contrôle de cette unité sera la description de la machine sur laquelle les microprogrammes seront exécutés (exemple description précédente du X2030).

En effet puisque l'on décrit une machine déjà réalisée, toutes les v variables de contrôle sont codées et seules les déclarations et des connexions ou affectations de registres non contrôlées par un état apparaîtront dans cette partie.

La partie de la description contrôlée par des variables d'état, sera la description des microprogrammes eux-mêmes. Cette liste d'instructions CASSANDRE devra être traduite en micromots inscriptibles dans la mémoire morte de la machine décrite en-dessus, en respectant les champs et le codage choisi pour cette machine (et définis par tables de codage).

c) Nous supposerons que toutes les machines traitées possèdent au moins des opérations $\neg$, $\wedge$ et V cablées.

Les autres opérations cablées de la machine (ex : ou $\neq$ ou + arithm) pourront être déclarées comme opérateur par l'utilisateur et il s'en servira dans la description de ses microprogrammes.

Ainsi on peut imaginer une première phase du traitement comme une analyse de la description définie ci-dessus au cours de laquelle :

1) On développe et fait disparaître les boucles pour, les instructions si arithm.

2) On traduit tous les opérateurs non déclarés en $\neg$, V et $\wedge$.

3) On crée en mémoire les tables de codage définies par la description de tous les décodeurs de la machine (voir exemple du X2030 p...).

4) On fait disparaître les opérations (autre que le seul allera) sous conditions booléennes (instructions si booléennes), en créant autant d'états que de conditions différentes peuvent se présenter.

EXEMPLE :

HETA1 : si A alors (I_1,,si B alors I_2 sinon I_3) sinon I_4, ;

HETA2 : ;

sera transformée en :

```
HETA1 : si A alors allera E1 sinon allera E2 ;
  E1 : I1, →;
  E11 : si B alors allera → sinon allera E4 ;
  E3 : I2, allera HETA2 ;
  E4 : I3, allera HETA2;;
  E2 : I4, → ;
HETA2 :                                    ;
```

Dans cet exemple on voit que chaque condition s'exprime alors sur des branchements et nécessite 1 microinstructions.

Il y a 4 microinstructions générées pour les opérations.

Ce développement correspond à une machine où l'on peut conditionner un branchement par l'état d'une bascule.

Dans le cas du X2030 où il y a 2 bascules de branchement conditionnel le développement serait différent.

Exemple :

```
HETA1 : si A/B alors allera E1
        sinon si A∧¬B alors allera E2
        sinon allera E3;
  E1 : I1,I2, allera HETA2;
  E2 : I1,I3, allera HETA2;
  E3 : I4 → ;
HETA2 :                    ;
```

La transformation dépend de la machine considérée, on imposera donc de spécifier le nombre de bascules utilisables pour les branchements conditionnels. Cette spécification branche servira pour la transformation des conditions.

5) Les ordres faire SEQ, provoqueront eux-aussi un allera.

```
Exemple : SEQ :          ;
                         ;
                 , allera SUITE;
          HETA1 : Faire SEQ,   ;
          HETA2 :         ;
```

sera transformé

```
          SEQ :                ;
                               ;
                , allera SUITE;
          HETA1 : SUITE   HETA2, allera SEQ;
          HETA2 :              ;
```

Après cette transformation le texte ne comportera plus que des branchements (:=) des affectations (←) conditionnés seulement par des états et des allera conditionnés par 1,2 ou n bascules suivant la machine.

Comme pour la simulation la difficulté vient alors des signaux. Ils ont une valeur instantanée. On ne peut pas les calculer une fois pour toutes et les stocker dans les registres de la machine.

6) On doit donc mémoriser formellement les branchements de signaux et en refaire le calcul dans chaque instruction où ils sont utilisés (mais le branchement est chaque fois conditionné par un état différent).

On peut également pour les signaux assez compliqués les compiler en un sous-microprogramme appelé par un ordre faire à chaque occurence du signal. L'optimisation du microprogramme produit, suivant la solution adoptée, reste à étudier.

Mais pour les signaux, comme pour les registres le second membre est une expression pas forcément "compatible" avec la machine donnée :

Exemple : dans le X2030 :

R (U G) V I;

ne peut pas s'éxécuter en un temps d'horloge comme c'est fait en CASSANDRE.

Il faut définir une séquence comme :

L← U;
D← G L;
R← D V I;

avec utilisation de 2 registres intermédiaires L et D.
Comment rendre "compatible" l'instruction CASSANDRE, à ce stade de la traduction, avec la machine pour laquelle elle est destinée.

d) Dans la phase précédente l'analyse du texte peut générer la matrice principale de la machine traitée, les tables de codages (déjà connues avant et exprimées en CASSANDRE ex : page 54) et la liste des opérations élémentaires réalisées par chaque noeud fonctionnel (par n° (liste qui doit correspondre aux déclarations de type opérateur).

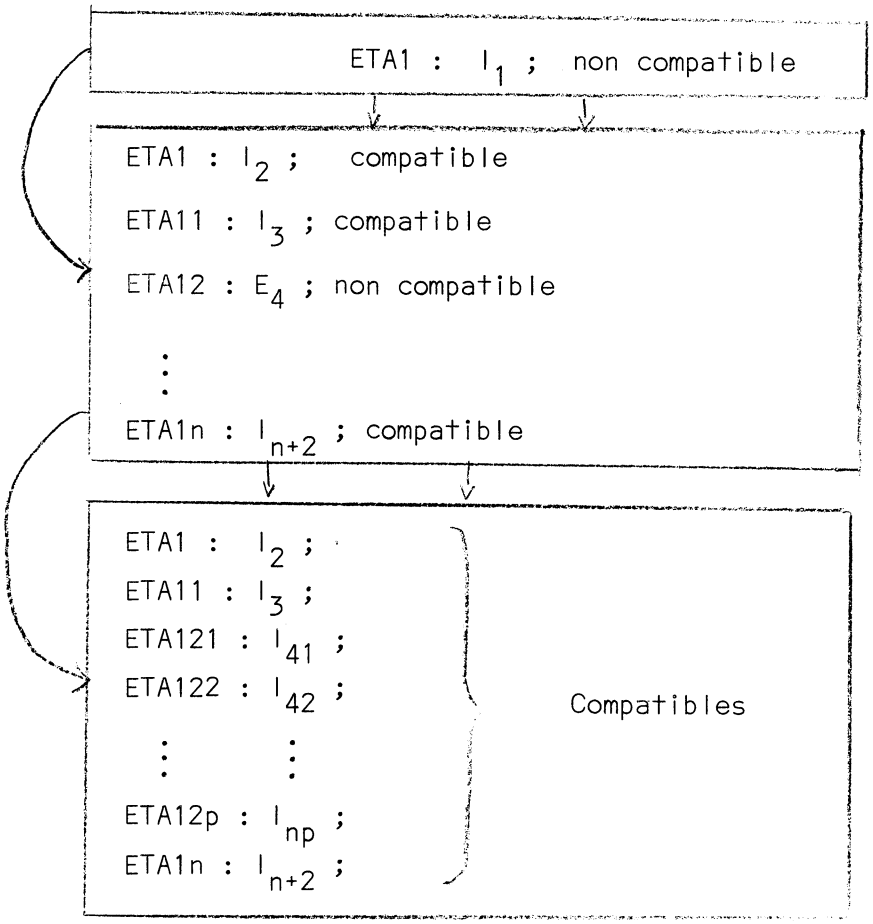
Si on analyse l'instruction CASSANDRE à compiler, (c'est-à-dire les expressions booléennes à calculer et à charger dans des registres) on obtiendra en général un ensemble d'opérations non compatibles avec la

matrice principale. Pour les rendre compatibles il faudra, en gros, faire exécuter séquentiellement ce qu'on ne peut faire simultanément.

Cela imposera l'utilisation de registres intermédiaires et un ordre non arbitraire des opérations. L'ordre peut être déterminé en attribuant une précédence à chaque opération élémentaire de façon à ménager au maximum les simultanéités.

En faisant l'union logique des matrices associées à toutes les opérations, pouvant encore s'éxécuter simultanément après cette transformation, on essayera d'obtenir un résultat compatible avec la matrice principale, sinon on devra itérer le processus pour cette nouvelle instruction.

Schéma :



Bien entendu l'étude d'un tel algorithme utilisant les "chemins" dans une matrice de connexions reste à achever.

Monsieur ANCEAU en propose un autre ayant pour support une analyse syntaxique.

e) Lorsqu'on a une séquence de microinstructions compatibles le passage au micromot se fait par un simple transcodage utilisant les tables de codage en mémoire.

Il reste à coder les états en en faisant des adresses et en codant les allera conditionnels (le résultat des conditions aura du être placé dans les bascules convenables). Ce rôle devra être confié à une sorte d'assembleur universel.

Le chargement du microprogramme en mémoire non destructive se fera par un chargeur qui transforme ces adresses relatives en adresses absolues.

Notons que les travaux mentionnés, à peine commencés, laissent place à une recherche importante et une programmation considérable. La plus grosse difficulté demeure la nécessité de produire des microprogrammes assez performants et économiques, les deux exigences n'étant pas pour une fois totalement incompatibles, puisque plus un microprogramme est court plus il peut être rapide et moins il occupe de mémoire.

[2] **F. ANCEAU Compilation de microprogrammes - Séminaire.**

IV - AUTRES APPLICATIONS : CONCLUSION

A - TEST DE CIRCUIT LOGIQUES DECRIT EN CASSANDRE

Le test des circuits logiques suppose l'existence de 3 outils.

a) Un langage pour les décrire et fournir la description au calculateur (nous avons vu en particulier dans I que CASSANDRE assure cette fonction).

b) Des algorithmes pour simuler le fonctionnement normal de ces circuits (voir système de simulation CASSANDRE [3]).

c) Des algorithmes pour déterminer des séquences de données assez courtes que l'on puisse envoyer sur les entrées du circuit afin que les valeurs successives des sorties permettent de repérer un élément en panne s'il y en un. Les travaux les plus interessants en France dans ce domaine ont été effectués au LAAS à Toulouse et au CNET à Lannion. Pour une telle application le langage CASSANDRE et son système de simulation fournit un outil équivalent à ceux qui existent déjà, nous pensons générer les séquences de test par la méthode des différences booléennes ou la méthode développée au CNET. Mais ces tests s'appliquent à des circuits ne dépassant pas l'importance d'une carte logique ou d'une pastille de circuits L.S.I.

Pour des ensembles logiques de l'importance d'un calculateur par exemple il semble que les programmes existant s'avèrent impuissants. Le problème qui se pose consiste à détecter une panne, puis à la localiser dans une carte puis à appliquer à cette carte la méthode précédemment définie. C'est dans ce comaine que CASSANDRE semble fournir un apport interessant en permettant la simulation à différents niveaux d'abstraction.

On peut envisager de simuler le fonctionnement d'un système beaucoup plus important qu'une carte logique et de déterminer les données de simulation qui permettent de détecter le mauvais fonctionnement d'un sous-ensemble et de localiser ce sous-ensemble.

Une telle étude pratiquée en même temps que la conception du système peut amener à modifier son découpage pour faciliter la localisation des pannes, on voit alors l'intérêt d'un système d'aide à la conception faisant interagir toutes les phases de conception d'un ensemble logique, comme le système CASSANDRE.

D'autres problèmes se posent au sujet des tests qui devraient être abordés dans ce contexte :

Ce sont : - Prise en compte des aléas possibles du circuit testé.
- Test de circuits séquentiels où l'on n'a pas accès à toutes les entrées-sorties des bascules. (Il semblent cependant que certains résultats apparaissent aujourd'hui dans ce domaine).

B - REALISATION D'AUTOMATISMES SEQUENTIELS

Nous avons vu au chapitre IIID de la partie précédente comment il était possible d'extraire de la description en CASSANDRE d'une unité le graphe de l'automate qui en assure le contrôle. Cet automate a pour états ceux de l'unité et pour sorties les signaux de commande des différentes actions de l'unité.

Un automate industriel ou un contrôleur de processus se présente de façon identique. Ces sorties sont alors les signaux de commande du processus.

Il peut se définir en CASSANDRE, on peut même le décrire à l'aide des EXPRESSIONS-PAYAN ce qui donne des possibilités nouvelles et sa synthèse se fera à partir de son graphe. Nous renvoyons pour cela à [78].

C - VISUALISATION DES RESEAUX D'UNITE ET DES RESULTATS :

Une Unité en CASSANDRE est conservée dans un fichier en mémoire sous forme codée, vérifiée et précompilée. C'est cette forme qui sert d'entrée à tous les traitements que l'on peut faire subir à une description. Un programme de SORTIE DE TEXTE [56] permet de réécrire un texte en CASSANDRE lisible et mis en page à l'imprimante ou à la machine à écrire du télétype, on peut sortir ce texte sur une plaque offset et la faire imprimer. On voit donc qu'à tout moment il est possible d'établir automatiquement la documentation relative à toutes les unités étudiées. La mise à jour de cette documentation ne pose aucun problème.

Il est très souhaitable de pouvoir sortir également sur une console de visualisation ou sur une table traçante le dessin du réseau correspondant à une unité.

Nous allons indiquer brièvement comment ceci pourrait être fait assez simplement.

a) L'unité visualisée occupe toute la page ou tout l'écran.

b) On ne dessinera donc que le réseau des unités contenues dans l'unité visualisée sans dessiner l'intérieur de celles-ci.

c) Si à son tour on visualise l'une de ces unités l'effet sera celui d'un zoom rapprochant, cadré sur cette unité, puisque son intérieur occupera alors tout l'écran.

d) Pour appliquer le processus précédent il suffit de considérer la liste des unités déclarées externe, d'associer à chacune le nombre de ses duplications et d'extraire la liste des signaux qui sont connectées à leurs entrées-sorties. Cette information peut s'extraire facilement de la description en CASSANDRE d'une unité.

e) On peut associer un graphe à ces unités et leurs interconnexions. Un algorithme de placement faisant intervenir des forces d'attractions et de répulsion peut effectuer un placement sur un quadrillage préétabli.

Les unités fortement connectées à l'extérieur seront disposées à la périphérie.

f) On remplace alors chaque unité par un rectangle de largeur standard et de longueur proportionnelle à $Max(p_1,p_2)$ si p_1 est le nombre d'entrées et p_2 le nombre de sorties de l'unité. Chaque unité est disposée dans celle de ses 4 dispositions possibles qui facilite le mieux ses connexions (ces positions se déduisent par rotation de 90°).

On écarte 2 lignes du quadrillage initial si 2 rectangles empiètent l'un sur l'autre.

g) On trace toutes les connexions des unités entre elles, en appliquant un algorithme de tracé dans les seules directions horizontale et verticale, sans se préoccuper des croisements de fils. Si une impossibilité de tracé apparait par manque de place on écarte encore les lignes ou les colonnes correspondantes du quadrillage initial.

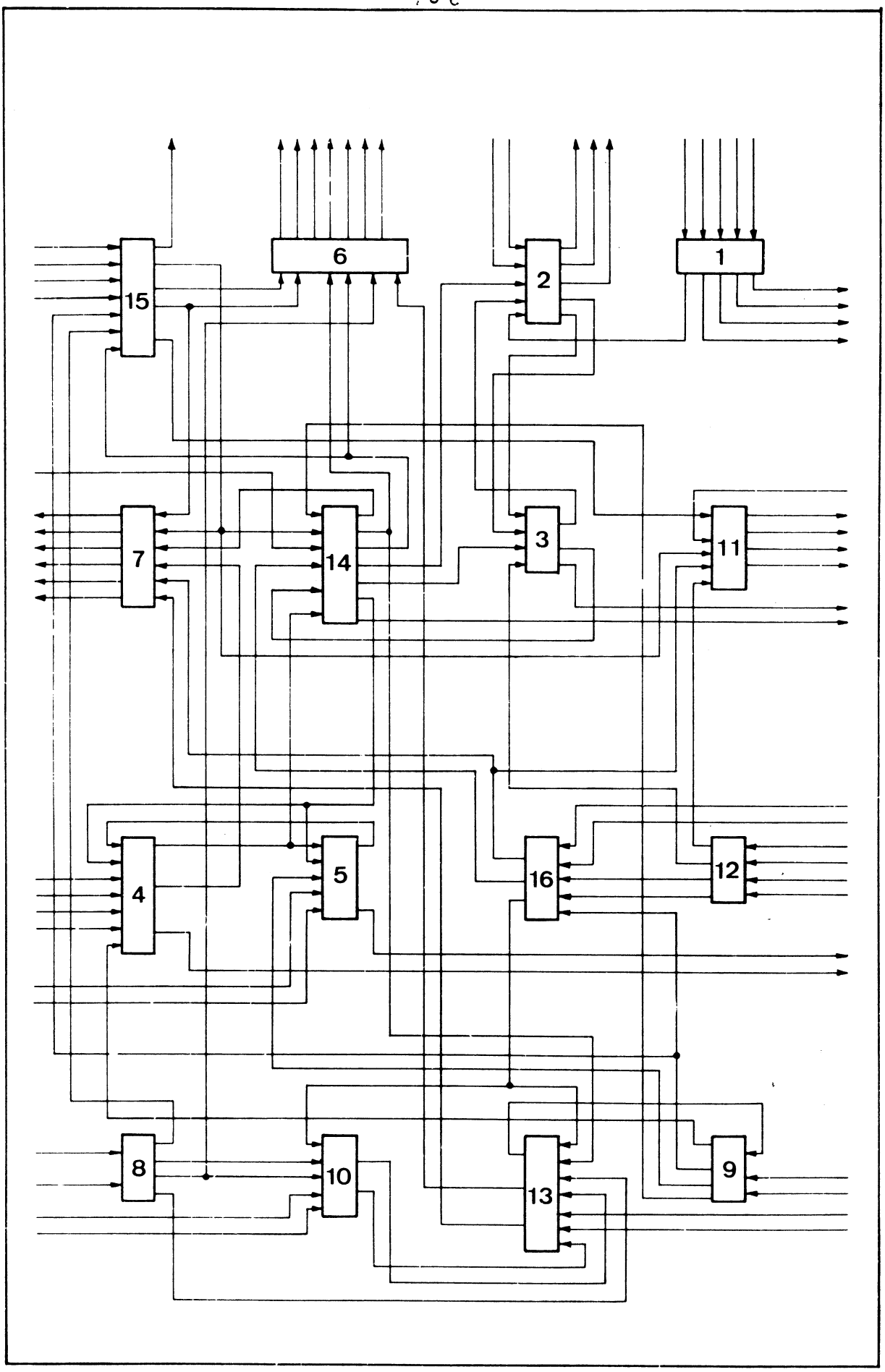
h) Si en cours de processus on sort des limites imposées par l'écran ou la feuille on envoit un message à l'utiiisateur pour qu'il restructure l'unité en unités plus petites avant de reboucier pour celles-ci sur le même processus.

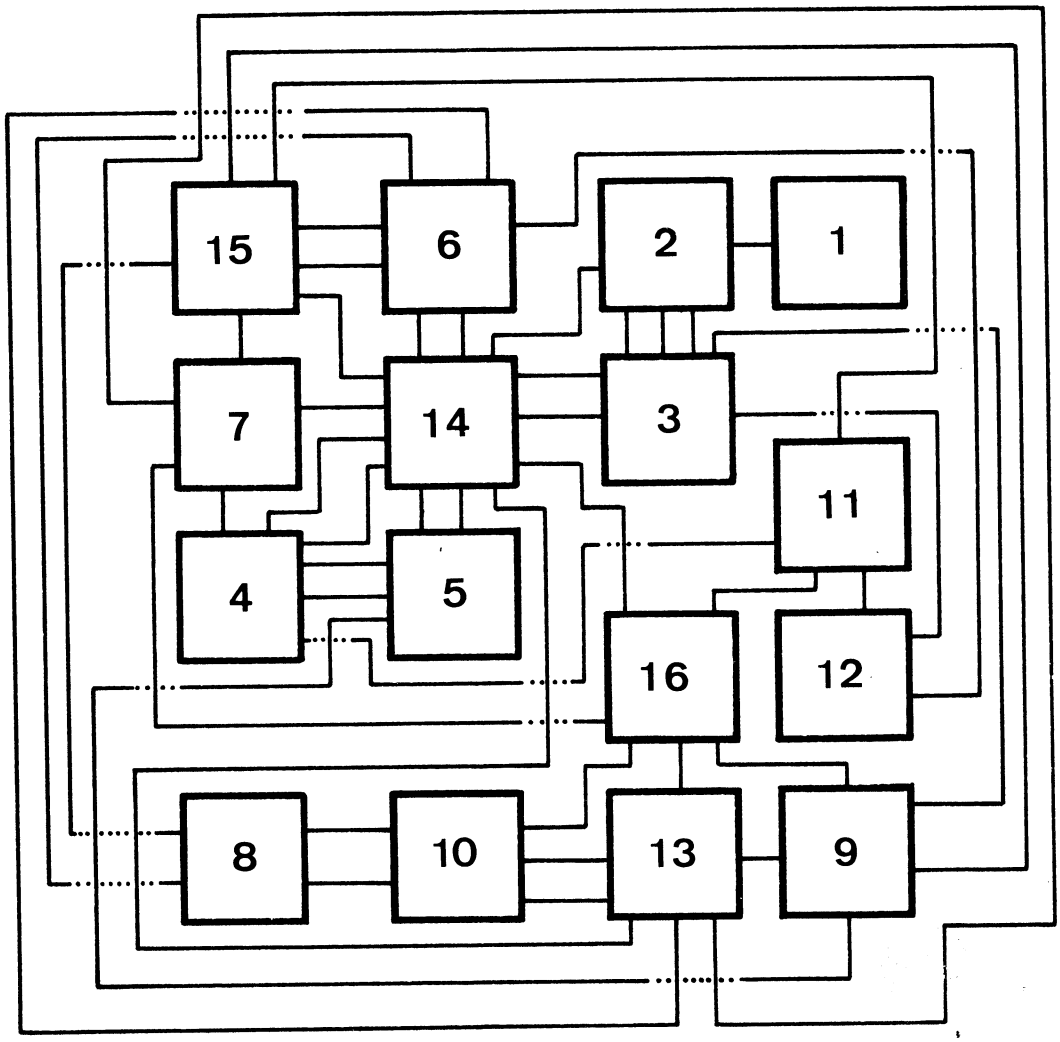
Ce traitement aura pour effet d'associer à toute unité qui en contient d'autres, le réseau de celles-ci dans un premier tracé, non optimisé et très imparfait, mais pourra faire ensuite l'objet d'une optimisation en conversation homme-machine... mais ceci sort de notre propos.

Un tel traitement aurait pu donner comme résultat le schéma de la page | 73 | à partir du réseau découpé en cellules dans un paragraphe ci-dessus.

On pourra ainsi constituer une bibliothèque de schémas optimisés qui sera jointe à la bibliothèque d'unités. Ce 2ème fichier décrira un schéma réel par des propriétés géométriques, alors que le fichier d'unités ne donne qu'un graphe abstrait de connexions d'unités. Leur tracé établira automatiquement à la demande le schéma de cablage du système étudié assurant ainsi totalement la fonction de documentation et d'archivage automatique.

A titre d'exemple le schéma page | 73 | pourrait être le résultat d'un premier tracé. Après son optimisation effectué normalement en mode conversationnel il pourrait donner le circuit directement implantable de la page | 74 |.


6
2
1
15
7
14
3
11
4
5
16
12
8
10
13
9


15
6
2
1
7
14
3
11
4
5
16
12
8
10
13
9

CONCLUSION

Le travail qui vient d'être présenté constitue un essai de synthèse des langages pour la conception aidée des systèmes logiques et de l'état de l'art en ce domaine.

Nous avons sur plusieurs points maintenant dépassé le stade de la recherche pour entrer dans celui des réalisations et quelquefois de l'exploitation. A ce niveau une collaboration étroite avec l'industrie est indispensable, c'est pourquoi, après la CFTH'B qui a été notre premier partenaire, nous avons commencé le travail en commun avec l'équipe recherche de Télémécanique et l'équipe T.I.T.N. sur l'étude de petits calculateurs et d'automatismes.

La collaboration, timidement amorcée avec la C.I.I. d'une part et la SPERAC d'autre part, devrait se développer rapidement et nous avons l'intention de contacter tous les concepteurs de systèmes logiques, sans exclusivité.

Mais il nous faut souligner que sur la plupart des applications, la dernière partie de cet ouvrage constitue plus un programme de travail pour les années à venir qu'un exposé de réalisations. La recherche y conserve une matière importante.

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

[1] G.M. AMDAHL, G.A. BLAAUW and F.P. BROOKS, *in, "Architecture of the IBM system/360, "IBM J., vol 8, pp. 87-101, 1964.*

[2] ANCEAU F. *"La compilation de microprogramme "Séminaire de logique mars 69.*

[3] ANCEAU F.,*"Le système de simulation CASSANDRE - CFTH" rapport de fin de contrat IMAG - CFTH-HB Mars 1970*

[4] F. ANCEAU, P. LIDDELL, J. MERMET, C. PAYAN, *"CASSANDRE : Conception assistée des systèmes digitaux" Onde électrique. Numéro Spécial, Janvier 1969.*

[5] F. ANCEAU, P. LIDDELL, J. MERMET, C. PAYAN, *"A language to Describe Digital Systems, Application to logic Design" December 18-20 - 1969.*

[6] P. BARLOW, *"Simulation as a design tool, "in Proc. of Design Automation Seminar, May 1964, sponsored by MESA Scientific Corp., Inglewood. Calf., 90303, pp. 164-192.*

[7] J.V. BLANKENBAKER, *"Logically microprogrammed computers-examples of small computers. Information processing, Proc. International Conference on Information Processing, UNESCO, 1969. Butterworths London, 1960.*

[8] L. BOLLIET, *"L'écriture des compilateurs," R.F.T.I. chiffres col. 9 n° 1, 1966.*

[9] A. BOUCHET, C. PAYAN, *"Synthèse des automates en fonctions croissantes. Immersion d'ordonné, "Séminaire de Logique Grenoble, Juin 1968.*

[10] G. BOULAYE, *"Logique et organes des calculatrices", DUNOD-UNIVERSITE 1970*

[11] M.A. BREUER, *"Général Survey of design automation of digital computers, "Proc of IEEE vol 54, n° 12, décembre 1966.*

[12] M.A. BREUER, *"Techniques for the simulation of computer logic, "Commun. ACM, pp. 443-446, July 1964.*

[13] J.A. BRZOZOWSKI, *"Dérivation of regular expressions," J. Assoc. Com. March. 11, 481-494, 1964.*

[14] C.J. BURNETT, *"A design Language for digital systems, "Masters Thesis MIT Août 1965.*

[15] S.H. CALDWELL, *"Switching circuits and logical design,"* J. Wiley and Sons, Inc. N.Y. 1958.

[16] C.E.A., *"Système Epicure". Document interne.*

[17] G. CECCHINI, G.B. GERACE, *"L'organizzazione Logica dell"Unita Di Controllo della C.E.P., "Centro studi calculatrici elettroniche del CNR presso l'Universita di Pisa.*

[18] R. M. Mc CLURE, *"A programming language for simulating digital systems,"* J.A.C.M., vol. 12, pp. 14-22 Jannuary 1965.

[19] Y. CHU, *"Digital computer design fundamentals, "Mc Graw Hill. N.Y. 1962.*

[20] Y. CHU, *"Macrologic design of digital computer, "University of Maryland Computer Science Center, Collège Park, Maryland.*

[21] Y. CHU, *"Buiding blocks for large-scale integration of logic circuits, "University of Maryland, College Park, Maryland.*

[22] Y. CHU, *"An Algol-like computer design language,"* Communications of the ACM, vol. 8, n° 10, october 1965.

[23] Y. CHU, *"Introduction to computer organization and programming,"* PRENTICE HALL 1970.

[24] COMPAGNIE DES MACHINES BULL, *"Direction des études, Note "Mémoire logique de microcommande pour calculateur moyennes performances.*

[25] J.R. DULEY, D.L. DIETMEYER, *"A digital system Design Language DDL"* IEEE trans-comput Vol 17, n° 9 septembre 1969.

[26] J.R. DULEY, D.L. DIETMEYER, *"Translation of a DDL digital system spécification to boolean équations IEEE, Vol C18, n° 4 Avril 1969.*

[27] E.W. DYKSTRA, *"The structure of the T.H.E. multiprogramming system,"* ACM Syp. On Operating, System Principles, 1967.

[28] G. ESTRIN, an R. MANDELL, *"Metacompiler as a design automation tool"* 1966 Proc. Share Design Automation Conference.

[29] A.D. FALKOFF, *"Formal description of processes, the first step in design automation", Thomas J. Watson Research Center 6/5/65. Note technique.*

[30] A.D. FALKOFF, K.E. IVERSON and E.H. SUSSENGUTH, *"Formal description of system/360, "IBM Sys. J., vol 3, pp. 198-262, 1964.*

[31] T.D. FRIEDMAN, *"Alert : A program to produce logic design from preliminary machine descriptions," RC 1578, march 24, 1966, IBM Research.*

[32] GAVRILOV, *"Langage LIAPAS permettant de traiter les algorithmes de synthèse des structures logiques,"Editeur NAOUKA, Moscou 1966.*

[33] G.B. GERACE, G. GESTRI, *"State assignments for reducing the number of delay elements in sequential machines," Colloque de Trieste, Juillet 1966.*

[34] G.B. GERACE, G. GESTRI, *"A method for designing a digital system as a sequential network system," Université de Pise, Juin 67.*

[35] A. GIESE, *"Hargol - A hardware oriented algol language," Internal report n° VA5, Août 1966, A/S Regnecentralen Copenhague, Dannemark.*

[36] A. GIESE, *"Formal définition of Hargol" Regnecentralen Falhoneralle 1 Copenhague (rapport interne) Fevrier 1969.*

[37] H.T. GLANTZ, *"A note on microprogramming". Journal ACM 3, p. 77, 1956.*

[38] D.F. GORMAN, J.P. ANDERSON, *"A logic design translator," 1962 Proc. FJCC pp. 86-96.*

[39] D.F. GORMAN, *"A system descriptive language and its uses" Ph.D. Université de Pensylvanie Avril 1968.*

[40] A. GRASSELLI, *"The design of a program modifiable microprogrammed control units", IRE Trans on Electronic computer Juin 1962.*

[41] J. GREEN, *"Microprogramming, emulators and programming languages. Comm. ACM 9,p.230,1966.*

[42] W.A. HANNING and T.L. MAYES, *"Impact of automation on digital computers design, "1960 Proc. EJCC, pp. 211-232.*

[43] L. HELLERMAN, *"A catalog of three variable OR-invert and AND-invert logical circuits," IEEE Trans. on Electronic Computers, vol. EC-12. pp. 198-223 June 1963.*

[44] K.E. IVERSON, *"A programming language, "J. Wiley and Sons. Inc. N.Y. LONDON 1962.*

[45] K.E. IVERSON, *"Elementary functions," Science Research Associates, INC 259 East Eric Street, Chicago 60611 - 1966.*

[46] K.E. IVERSON, *"APL/360 User"s Manual," International Business Machines Corporation, July 1968.*

[47] E.A. JOSEPH, *"Impact fo large-scale integration on aerospace computers", IEEE Trans. on Electronic Computers, vol. EC-16, n° 5, october 1967.*

[48] T.W. KAMPE, *"The design of a general-purpose microprogram controled computer with elementary structure. IRE Trans., EC-9, p. 208, 1960.*

[49] W.H. KAUTZ, *"A cellular threshold array," IEEE Trans. on Electronic Computer, vol. EC-16 n° 5, october 1967.*

[50] D.E. KNUTH, J.L. Mc NELEY, *"SOL-A symbolic language for general-prupose systems simulation," IEEE Trans. on Electronic Computers, vol. EC-13, pp. 401-408, August 1964.*

[51] J. KUNTZMANN, *"Algèbre de Boole," DUNOD*

[52] E.L. LAWLER, *"An approach to multilevel Boolean minimization," J. ACM, Vol. 11, pp. 283-296, July 1964.*

[53] V.G. LAZAREV, *"Matrix method of minimization of microprogram systems," Avril 1964.*

[54] O. LECARME, J. MERMET, *"Programmation 7040-44. Cours polycopies Math. Appli.*

[55] C.Y. LEE, *"An algorithm for path connections and its applications," IRE Trans. On Electronic Computers, vol. EC-10, pp. 346-365, September 1961.*

[56] P. LIDDELL, *"Découpage syntaxique de systèmes logiques décrits en CASSANDRE" Thèse 3e cycle Mars 1970*

[57] F. LUSTMAN, *"Langages exprimant le fonctionnement d'un système " Rapport DGRST - octobre 1967.*

[58] F. LUSTMAN, *"Réalisation de fonctions booléennes à l'aide de l'opérateur Majorité, "Thèse Docteur-Ingénieur, Grenoble mars 1966.*

[59] F. LUSTMAN, J. MERMET, *"CASSANDRE - Un langage de description de machines digitales," Revue Bleue de l'AFIRO N° 15, 1969.*

[60] MACHERAS, *"Synthèse d'une fonction booléenne par l'opérateur U," Thèse de 3ème cycle, Université de GRENOBLE, Juin 1966.*

[61] MAITRA, *"Cascaded switching networks of two input flexible calls," IRE Trans. EC-4, 136-143, 1962.*

[62] S. MAJERSKI and MICHEAL WIWEGER, *"Wor-gate binary adder with carry completion detection," IEEE Trans. on Electronic Computers, fevrier 1966.*

[63] R.J. MERCER, *"Microprogramming," Journal ACM 4, p. 157 (1957).*

[64] J. MERMET, *"Description formelle d'un langage orienté vers le hardware," Colloque sur les calculateurs embarqués sur fusées et satellites, 3.6. décembre 1968 CNES Paris.*

[65] J. MERMET , *"Le langage CASSANDRE", Rapport DGRST, Contrat 66 00 069 31 mars 1968.*

[66] J. MERMET, *"Synthèse de fonctions booléennes par des opérateurs en ligne. Bull-Inst. Polytechnique de JASSY, Tome XIII 1967, Fasc. 1-2.*

[67] J. MERMET, G. AURIAUX, *"Etude d'un macrogénérateur. Projet de fin d'étude, 1966.*

[68] C.K. MESZTENYI, *"Translator and simulator for computer design and simulation," Program (CDSP) Version I, University of Maryland, Computer Science Center, College Park, Maryland.*

[69] G. METZE, S. SESHU, *"A proposal for a computer compiler," 1966 Proc. SJCC, pp. 253-264.*

[70] G. METZE, S. SESHU, *"Computer compiler part I-Preliminary report," Coordinated Science Lab., University of Illinois, Urbana. Dept. R-364, August 1965.*

[71] R.C. MINNICK, R.A. SHORT, *"Cellular linera input logic," Final Report Contrat AF 19 (628)-498, project 4641, Task 464101, 1964.*

[72] R.C. MINNICK, R.A. SHORT, J. GOLDBERG, H.S. STONE, M.W. GREEN, M. YOELI, W.H., KAUTZ, *"Cellular arrays for logic and storage," Final report, contrat AF 19-628)-4233, Project 4641, Tasl N° 464101, avril 66.*

[73] NASLIN, *"Circuits logiques et automatisme à séquence," DUNOD 1965.*

[74] K. NYGAARD and O.J. DAHL, "*SIMULA, an Algol, Based-Simulation language,*" *Communication of the ACM vol. 9, n° 9, sept. 1966.*

[75] D.L. PARNAS, "*A language for describing the functions of synchronous systems,*" *Communication of the ACM, vol. 9, n° 2 février 1966.*

[76] D.L. PARNAS, "*On simulating Networks of parallel prosesses in wvhich simultaneous events may occur*". *Comm. of. ACM. Vol 12- n° 9 sept 1969.*

[77] D.L. PARNAS, and J.A. DARRINGER, "*SODAS and methodology for system design,*" *Fall joint computer conference, 1967.*

[78] C. PAYAN, "*Description et synthèse des automates- Notion "dès que" Colloque international Systèmes logiques, conception et Applications Bruxelles septembre 69.*

[79] R. PROCTOR, "*A logic design translator experiment demonstrating relationship of langage to systems and logic design,*" *IEEE Trans. on Electronic Computers, col. EC-13, pp. 422-430, August 1964.*

[80] J.P. ROTH, "*Systematic design of automata, "1965 Proc. FJCC, pp. 1093-1099.*

[81] J.P. ROTH, "*Hardware implementation of microprograms,*" *IBM Rapport interne, RC 1456, août 1965.*

[82] J.P. ROTH, W.G. BOURICIUS, P.R. SCHNEIDER, "*Programmed algorithms to computer tests to detect and distinguish between failures in logic circuits,*" *IEEE Trans. on Electronic Computers, vol EC-16, n° 5 October 1967.*

[83] R.A. RUTMAN, "*Logik a syntax-directed compiler for computer BIT-time simulation,*" *M.S. thesis, University of California, Los Angeles, August 1964.*

[84] J.C. SAILLARD, "*Etude des formes lexicographiques des fonctions booléennes simples,*" *Représentation à l'aide de l'opérateur U, Thèse 3ème Cycle, 23 Décembre 1968.*

[85] M. SARRET, "*Problèmes d'implantation, étude de la planéarité pour les réseaux électroniques,*" *Contrat DRME, 6634215/6544.*

[86] G. SAUCIER, "*Codage des états internes de systèmes séquentiels asynchrones,*" *Thèse 3ème Cycle, Université de GRENOBLE.*

[87] H.P. SCHLAEPPI, *"A formal language for describing machine logic, timing and sequencing (LOTIS),"* IEEE Trans. on Electronic Computers, vol. EC-13, pp. 439-448, August 1964.

[88] H. SCHORR, *"A program for the analyses of digital systems,"* Digital Systems Lab. Techn. Rept. 32, Princeton University, Princeton, N.J., 60 pp., Décembre 1962.

[89] H. SCHORR, *"Computer-aided digital system design and analyses using a register transfert language,"* IEEE Trans. On Electronic Computers, Vol. EC-13, pp. 730-737, December 1964.

[90] B. SEMPE, *"Conception et réalisation d'un calculateur industriel,"* Faculté de GRENOBLE, Thèse de Docteur Ingénieur.

[91] S. SESHU, *"On an improved diagnostic program,"* IEEE Trans. on Electronic Computers (Short Notes), Vol. EC-14, pp. 76-79, February 1965.

[92] D.L. SMITH, *"Models and data structures for digital logic simulation,"* MIT, Projet MAC, août 1966.

[93] M. SOKOLOFF, *"Tentatives d'emploi du produit tensoriel généralisé en langage programme Iversion pour la compilation automatique des schémas logique,"* Colloque d'Algèbre de Boole - GRENOBLE 1965.

[94] C.V. SRINIVASAN, *"An introduction to CDLI - A computer description language,"* RCA Lab. N.Y. Contrat n° AF 319(628)4789, proj. 5682 Task n° 563202, septembre 1967 Report n° 2 octobre 1967.

[95] C. STRACHEY, *"A general Purpose Macrogenerator,"* Computer Journal vol. 8 N° 3 octobre 1965.

[96] *Table ronde sur les langages pour la conception aidée des systèmes logiques. Laboratoire de Mathématiques Appliquées, St Martin d'Hères. 9-10 Octobre 1967.*

[97] D. TEICHROEW, J.F. LUBIN, *"Computer simulation - Discussion of the technique and comparaison languages,"* Communications of the ACM vol. 9 n° 10. Octobre 1966.

[98] P. TISON, *"Etude des systèmes logiques,"* Rapport de stage aux Etats Unis, 16/11/1966.

[99] VAN CO, *"Description et simulation d'une machine digitale," Document interne, Compagnie Bull.*

[100] J. VINCENT-CARREFOUR, *"Utilisation d'un calculateur pour la définition des ensembles logiques," L'onde Electrique, janvier 1968.*

[101] B. WATERLOT et M. SARRE, *" Crismass: un outil pour la conception, la réalisation ou l'implantation et la simulation de machines séquentielles synchrones". Note Sema Sperac.*

[102] H. WEBER, *"A microprogrammed implementation of EULER on IBM/360/30 Comm. ACM 10, p. 549, 1967.*

[103] J.A. WILBER, *"A language for describing digital computer," Report n° 197, Dept. of Com. Science, University of Illinois, Feb. 15, 1966.*

[104] M.V. WILKES, *"The best way to design an automatic calculation machine," Manchester University Computer Inaugural Conference, p. 16 1951.*

[105] M. WINER, *"Codage de microinstructions, "note n° 66/0324, SETI 1966.*

[106] M.S. ZÜCKER, *"LOCS : an EDP machine logic and control simulator," 1965 IEEE Conv. Rec., pt. 3, pp. 28-51.*

[107] F.W. ZURCHER, B. RANDELL, *"Iterative multi-level modelling - A methodelogy for computer system disign," RC 1938, novembre 10, 1967, IBM Research.*

VU

Grenoble, le

Le Président de la Thèse

VU

Grenoble, le

Le Doyen de la Faculté des Sciences

Vu, et permis d'imprimer,

Le Recteur de l'Académie de GRENOBLE