Etude méthodologique de la conception assistée par ordinateur des systèmes logiques : CASSANDRE

Jean Mermet

THESE

Présentée à

l'Université Scientifique et Médicale de Grenoble

pour obtenir

le grade de Docteur es-sciences mathématiques par

Jean MERMET

Etude méthodologique de la conception assistée par ordinateur des systèmes logiques : CASSANDRE

Soutenue le 10 avril 1973 devant la commission d'examen

MM. J. KUNTZMANN Président

P.J. LAURENT
Mme G. SAUCIER
MM. M. GRIFFITHS
F. GENUYS
} Examinateurs

UNIVERSITE SCIENTIFIQUE

ET MEDICALE DE GRENOBLE

LISTE DES PROFESSEURS

Président : Monsieur Michel SOUTIF
Vice-Président : Monsieur Gabriel CAU

PROFESSEURS TITULAIRES

MM.	ANGLES D'AURIAC Paul	Mécanique des fluides
	ARNAUD Georges	Clinique des maladies infectieuses
	ARNAUD Paul	Chimie
	AYANT Yves	Physique approfondie
Mme	BARBIER Marie-Jeanne	Electrochimie
MM.	BARBIER Jean-Claude	Physique expérimentale
	BARBIER Reynold	Géologie appliquée
	BARJON Robert	Physique nucléaire
	BARNOUD Fernand	Biosynthèse de la cellulose
	BARRA Jean-René	Statistiques
	BARRIE Joseph	Clinique chirurgicale
	BENOIT Jean	Radioélectricité
	BESSON Jean	Electrochimie
	BEZES Henri	Chirurgie générale
	BLAMBERT Maurice	Mathématiques Pures
	BOLLIET Louis	Informatique (IUT B)
	BONNET Georges	Electrotechnique
	BONNET Jean-Louis	Clinique ophtalmologique
	BONNET-EYMARD Joseph	Pathologie médicale
	BONNIER Etienne	Electrochimie Electrométallurgie
	BOUCHERLE André	Chimie et Toxicologie
	BOUCHEZ Robert	Physique nucléaire
	BRAVARD Yves	Géographie
	BRISSONNEAU Pierre	Physique du Solide
	BUYLE-BODIN Maurice	Electronique
	CABANAC Jean	Pathologie chirurgicale
	CABANEL Guy	Clinique rhumatologique et hydrologie
	CALAS François	Anatomie
	CARRAZ Gilbert	Biologie animale et pharmacodynamie
	CAU Gabriel	Médecine légale et Toxicologie
	CAUQUIS Georges	Chimie organique
	CHABAUTY Claude	Mathématiques Pures
	CHARACHON Robert	Oto-Rhino-Laryngologie
	CHATEAU Robert	Thérapeutique
	CHENE Marcel	Chimie papetière
	COEUR André	Pharmacie chimique
	CONTAMIN Robert	Clinique gynécologique
	COUDERC Pierre	Anatomie Pathologique
	CRAYA Antoine	Mécanique
Mme	DEBELMAS Anne-Marie	Matière médicale
MM.	DEBELMAS Jacques	Géologie générale
	DEGRANGE Charles	Zoologie
	DESSAUX Georges	Physiologie animale
	DODU Jacques	Mécanique appliquée
	DREYFUS Bernard	Thermodynamique
	DUCROS Pierre	Cristallographie
	DUGOIS Pierre	Clinique de Dermatologie et Syphiligraphie
	FAU René	Clinique neuro-psychiatrique
	FELICI Noël	Electrostatique
	GAGNAIRE Didier	Chimie physique
	GALLISSOT François	Mathématiques Pures
	GALVANI Octave	Mathématiques Pures

MM.	GASTINEL Noël	Analyse numérique
	GERBER Robert	Mathématiques Pures
	GIRAUD Pierre	Géologie
	KLEIN Joseph	Mathématiques Pures
Mme	KOFLER Lucie	Botanique et Physiologie végétale
MM.	KOSZUL Jean-Louis	Mathématiques Pures
	KRAVTCHENKO Julien	Mécanique
	KUNTZMANN Jean	Mathématiques Appliquées
	LACAZE Albert	Thermodynamique
	LACHARME Jean	Biologie végétale
	LATREILLE René	Chirurgie générale
	LATURAZE Jean	Biochimie pharmaceutique
	LAURENT Pierre	Mathématiques Appliquées
	LEDRU Jean	Clinique médicale B
	LLIBOUTRY Louis	Géophysique
	LOUP Jean	Géographie
Mle	LUTZ Elisabeth	Mathématiques Pures
	MALGRANGE Bernard	Mathématiques Pures
	MALINAS Yves	Clinique obstétricale
	MARTIN-NOEL Pierre	Seméiologie médicale
	MASSEPORT Jean	Géographie
	MAZARE Yves	Clinique médicale A
	MICHEL Robert	Minéralogie et Pétrographie
	MOURIQUAND Claude	Histologie
	MOUSSA André	Chimie nucléaire
	NEEL Louis	Physique du Solide
	OZENDA Paul	Botanique
	PAUTHENET René	Electrotechnique
	PAYAN Jean-Jacques	Mathématiques Pures
	PEBAY-PEYROULA Jean-Claude	Physique
	PERRET René	Servomécanismes
	PILLET Emile	Physique industrielle
	RASSAT André	Chimie systématique
	RENARD Michel	Thermodynamique
	REULOS René	Physique industrielle
	RINALDI Renaud	Physique
	ROGET Jean	Clinique de pédiatrie et de puériculture
	SANTON Lucien	Mécanique
	SEIGNEURIN Raymond	Microbiologie et Hygiène
	SENGEL Philippe	Zoologie
	SILBERT Robert	Mécanique des fluides
	SOUTIF Michel	Physique générale
	TANCHE Maurice	Physiologie
	TRAYNARD Philippe	Chimie générale
	VAILLAND François	Zoologie
	VAUQUOIS Bernard	Calcul électronique
Mme	VERAIN Alice	Pharmacie galénique
M.	VERAIN André	Physique
Mme	VEYRET Germaine	Géographie
MM.	VEYRET Paul	Géographie
	VIGNAIS Pierre	Biochimie médicale
	YOCCOZ Jean	Physique nucléaire théorique

PROFESSEURS ASSOCIES

MM.	BULLEMER Bernhard	Physique
	RADHAKRISHNA Pidatala	Thermodynamique

PROFESSEURS SANS CHAIRE

MM.	AUBERT Guy	Physique
	BEAUDOING André	Pédiatrie
	BERTRANDIAS Jean-Paul	Mathématiques Appliquées
	BIARES Jean-Pierre	Mécanique
	BONNETAIN Lucien	Chimie minérale
Mme	BONNIER Jane	Chimie générale
MM.	CARLIER Georges	Biologie végétale
	COHEN Joseph	Electrotechnique
	COUMES André	Radioélectricité
	DEPASSEL Roger	Mécanique des Fluides
	DEPORTES Charles	Chimie minérale
	DESRE Pierre	Métallurgie
	DOLIQUE Jean-Michel	Physique des Plasmas
	GAUTHIER Yves	Sciences biologiques
	GEINDRE Michel	Electroradiologie
	GIDON Paul	Géologie et Minéralogie
	GLENAT René	Chimie organique
	HACQUES Gérard	Calcul numérique
	JANIN Bernard	Géographie
Mme	KAHANE Josette	Physique
MM.	MULLER Jean-Michel	Thérapeutique
	PERRIAUX Jean-Jacques	Géologie et minéralogie
	POULOUJADOFF Michel	Electrotechnique
	REBECQ Jacques	Biologie (CUS)
	REVOL Michel	Urologie
	REYMOND Jean-Charles	Chirurgie générale
	ROBERT André	Chimie papetière
	SARRAZIN Roger	Anatomie et chirurgie
	SARROT-REYNAULD Jean	Géologie
	SIBILLE Robert	Construction Mécanique
	SIROT Louis	Chirurgie générale
Mme	SOUTIF Jeanne	Physique générale
M.	VALENTIN Jacques	Physique nucléaire

MAITRES DE CONFERENCES ET MAITRES DE CONFERENCES AGREGES

Mle	AGNIUS-DELORD Claudine	Physique pharmaceutique
	ALARY Josette	Chimie analytique
MM.	AMBLARD Pierre	Dermatologie
	AMBROISE-THOMAS Pierre	Parasitologie
	ARMAND Yves	Chimie
	BEGUIN Claude	Chimie organique
	BELORIZKY Elie	Physique
	BENZAKEN Claude	Mathématiques Appliquées
Mme	BERTRANDIAS Françoise	Mathématiques Pures
MM.	BLIMAN Samuel	Electronique (EIE)
	BLOCH Daniel	Electrotechnique
Mme	BOUCHE Liane	Mathématiques (CUS)
MM.	BOUCHET Yves	Anatomie
	BOUSSARD Jean-Claude	Mathématiques Appliquées
	BOUVARD Maurice	Mécanique des Fluides
	BRIERE Georges	Physique expérimentale
	BRODEAU François	Mathématiques (IUT B)
	BRUGEL Lucien	Energétique
	BUISSON Roger	Physique
	BUTEL Jean	Orthopédie
	CHAMBAZ Edmond	Biochimie médicale
	CHAMPETIER Jean	Anatomie et organogénèse

MM. CHIAVERINA Jean	Biologie appliquée (EFP)
CHIBON Pierre	Biologie animale
COHEN-ADDAD Jean-Pierre	Spectrométrie physique
COLOMB Maurice	Biochimie médicale
CONTE René	Physique
CROUZET Guy	Radiologie
DURAND Francis	Métallurgie
DUSSAUD René	Mathématiques (CUS)
Mme ETERRADOSSI Jacqueline	Physiologie
MM. FAURE Jacques	Médecine légale
GAVEND Michel	Pharmacologie
GENSAC Pierre	Botanique
GERMAIN Jean-Pierre	Mécanique
GIDON Maurice	Géologie
GRIFFITHS Michaël	Mathématiques Appliquées
GROULADE Joseph	Biochimie médicale
HOLLARD Daniel	Hématologie
HUGONOT Robert	Hygiène et Médecine préventive
IDELMAN Simon	Physiologie animale
IVANES Marcel	Electricité
JALBERT Pierre	Histologie
JOLY Jean-René	Mathématiques Pures
JOUBERT Jean-Claude	Physique du Solide
JULLIEN Pierre	Mathématiques Pures
KAHANE André	Physique générale
KUHN Gérard	Physique
Mme LAJZEROWICZ Jeannine	Physique
MM. LAJZEROWICZ Joseph	Physique
LANCIA Roland	Physique atomique
LE JUNTER Noël	Electronique
LEROY Philippe	Mathématiques
LOISEAUX Jean-Marie	Physique Nucléaire
LONGEQUEUE Jean-Pierre	Physique Nucléaire
LUU DUC Cuong	Chimie Organique
MACHE Régis	Physiologie végétale
MAGNIN Robert	Hygiène et Médecine préventive
MARECHAL Jean	Mécanique
MARTIN-BOUYER Michel	Chimie (CUS)
MAYNARD Roger	Physique du Solide
MICOUD Max	Maladies infectieuses
MOREAU René	Hydraulique (INP)
NEGRE Robert	Mécanique
PARAMELLE Bernard	Pneumologie
PECCOUD François	Analyse (IUT B)
PEFFEN René	Métallurgie
PELMONT Jean	Physiologie animale
PERRET Jean	Neurologie
PERRIN Louis	Pathologie expérimentale
PFISTER Jean-Claude	Physique du Solide
PHELIP Xavier	Rhumatologie
Mlle PIERY Yvette	Biologie animale
MM. RACHAIL Michel	Médecine interne
RACINET Claude	Gynécologie et obstétrique
RICHARD Lucien	Botanique
Mme RINAUDO Marguerite	Chimie macromoléculaire
MM. ROMIER Guy	Mathématiques (IUT B)
ROUGEMONT (DE) Jacques	Neuro-Chirurgie
STIEGLITZ Paul	Anesthésiologie

MM. STOEBNER Pierre	Anatomie pathologique
VAN CUTSEM Bernard	Mathématiques Appliquées
VEILLON Gérard	Mathématiques Appliquées (INP)
VIALON Pierre	Géologie
VOOG Robert	Médecine interne
VROUSSOS Constantin	Radiologie
ZADWORNY François	Electronique

MAITRES DE CONFERENCES ASSOCIES

MM. BOUDOURIS Georges	Radioélectricité
CHEEKE John	Thermodynamique
GOLDSCHMIDT Hubert	Mathématiques
YACOUD Mahmoud	Médecine légale

CHARGES DE FONCTIONS DE MATIRES DE CONFERENCES

Mme BERIEL Hélène	Physiologie
Mme RENAUDET Jacqueline	Microbiologie

Fait le 8 MARS 1972.

Je tiens à exprimer ici toute ma reconnaissance à :

Monsieur le Professeur J. KUNTZMANN, Directeur de l'ENSIMAG, qui a couvert ce travail de son autorité bienveillante et 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 le Professeur P.J. LAURENT qui m'a fait découvrir quelques élégantes propriétés de l'optimisation convexe.

Ma gratitude ira également à

Madame G. SAUCIER, Maître de Conférences à l'ENSIMAG,

Monsieur M. GRIFFITHS, Maître de Conférences à l'Institut de Programmation, et à

Monsieur F. GENUYS, Président de l'AFCET, Directeur à IBM France, qui ont bien voulu accepter de faire partie du Jury.

Je tiens à remercier aussi le C.R.I. qui par son aide financière a soutenu la présente recherche.

Celle-ci s'est déroulée en collaboration fructueuse avec les Sociétés T.I.T.N., Télémécanique Electrique, Philips Data Systems, et Institut Français du Pétrole, pour ne pas mentionner à nouveau la CFTH-HB et la contribution de Monsieur F. LUSTMAN.

Enfin ce travail est essentiellement le résultat de l'effort d'une équipe dans laquelle nous remercierons collectivement : Madame De POLIGNAC, et MM. ANCEAU, ARCHER, BOGO, CHATELIN, COMBES, COUTURIER, DAVID, DESCHIZEAUX, DOUSSY, FANTINO, GUYOT, HANZAKOWSKI, LIDDELL, MENARD, PAYAN, PERRON.

Je voudrais signaler l'apport particulièrement important de Messieurs DAVID et FANTINO au présent ouvrage. Il est précieux d'avoir de tels collaborateurs.

Enfin je remercie Monsieur RASOLONJATOVO, pour la patience et le goût qu'il a manifestés dans la réalisation des nombreux dessins, Madame DUFFOURD qui a produit un texte soigné, et le service de tirage qui a assuré la délicate réalisation de ce document.

TABLE DES MATIERES

INTRODUCTION

Le domaine dans lequel nous nous plaçons tout au long de cet ouvrage est celui de la conception des machines digitales (par exemple les ordinateurs) et plus largement de ce que nous appelerons les "systèmes logiques". Un système logique peut être défini entièrement à l'aide de l'algèbre de Boole et de la théorie des automates finis, mais ce n'est peut être pas la façon la plus concise de le décrire (ni la plus "naturelle").

Si on réalise physiquement le système logique décrit, on aboutit à une machine digitale. Mais dans de nombreux cas, ceux par exemple où l'on peut en rester à une étude de réseau (au sens de KUNTZMANN (1)), la conception ne débouche pas forcément sur une réalisation physique ; c'est en ce sens que la notion de système logique est plus générale que celle de machines digitales.

Définir une méthodologie pour la conception assistée par calculateur dans ce domaine (comme dans tout autre) demande que l'on mène trois études préalables :

- Etude de la nature des problèmes rencontrés
- Etude des méthodes des concepteurs de systèmes logiques
- Inventaire des outils programmés existants et réellement utiles.

Le résultat de ces études a été résumé dans (2) et dans plusieurs articles. Rappelons simplement quelques notions de base.

Les problèmes rencontrés dans la conception des systèmes logiques sont de nature logique, analogique ou topologique.

Les problèmes topologiques sont ceux que l'on rencontre dans une phase proche de la réalisation, lorsqu'on doit placer les unes par rapport aux autres, les différentes parties du matériel servant

à construire la machine en projet et interconnecter ces "boites" en respectant les contraintes géométriques ou de planarité qui accompagnent les choix technologiques.

Il est difficile de définir à ce niveau un outil programmé suffisamment général pour traiter tous les problèmes d'implantation, car un faible changement technologique peut modifier radicalement les contraintes géométriques et topologiques.
Mais la question est cruciale dans l'industrie.

C'est pourquoi, sur ces questions avant toutes autres, de très nombreux programmes (exemple (3)) ont été réalisés depuis presque vingt ans, limités chacun à une technologie assez précisément définie, et par là même souvent vite périmés.

Chaque grand constructeur de calculateur à bâti à grands frais sa propre bibliothèque de tels programmes, répondant au mieux à ses besoins internes. Les caractéristiques détaillées de ces systèmes sont en général non communiquées. Nous pensons qu'au niveau des méthodes employées pour résoudre les problèmes d'implantation il est illusoire, et d'ailleurs inutile, d'espérer obtenir une automatisation complète des travaux.

Il s'agit donc de mettre au point des systèmes conversationnels où le concepteur garde le contrôle des opérations, en dialogue avec l'ordinateur.

Le système devra faire appel à différents programmes de placement, de découpage en modules et de tracé de connexions, suivant les choix du concepteur. Il devra donner à celui-ci toute latitude d'intervenir pour faire une correction ou complèter le travail (par exemple modifier le découpage, le placement ou le tracé). Ceci implique entre autre, trois possibilités encore bien mal satisfaites dans les systèmes existant à ce jour :

- Rentrer les données du traitement dans le minimum de volume, problème pratique considérable, qui impose en fait la génération semi-automatique de ces données à partir d'un langage évolué (qui est alors le langage d'entrée).

- Prendre en compte, dans la structure de données du système, toute modification ou adjonction voulue par les concepteurs.

- Vérifier la cohérence des données introduites et le bon fonctionnement du matériel décrit, à tout moment de l'évolution du projet.

Ces trois objectifs nous ont guidés dès le début de notre travail. On verra dans la partie C les implications qu'ils ont eues sur notre réalisation. Sans proposer d'algorithme pour résoudre les problèmes topologiques, nous définissons une méthode pour les aborder.

Les problèmes analogiques se rencontrent dès qu'il s'agit d'étudier les caractéristiques et le fonctionnement des circuits électroniques qui réalisent les composants des machines digitales actuelles (4). L'interface avec les problèmes logiques et topologiques est réduit respectivement à la donnée de la durée des différents signaux (ou des retards auxquels ils sont soumis, voir CASSANDRE asynchrone), et la donnée des caractéristiques géométriques des circuits élémentaires.

C'est dans le domaine logique que nous avons développé notre étude. On a coutume de diviser ce domaine en quatre niveaux :

- Celui des réseaux de portes logiques
- Celui des sous ensembles logiques et microprogrammes
- Celui des programmes
- Celui du système

Le langage et le système CASSANDRE que nous avons développés couvrent les deux premiers niveaux mais ne peuvent s'étendre aux deux autres, nous allons voir pourquoi.

Les objectifs d'un système d'aide à la conception sont essentiellement :

- La génération de l'information détaillée relative au projet
- La mise à jour permanente de celui-ci
- La vérification du bon fonctionnement du système aux différents niveaux de description et la compatibilité de ces niveaux entre eux.
- La préparation des données de la fabrication (listes de cablage, séquences de test, schémas logiques...).

Nous avons expliqué dans (2) que ceci entrainait à notre avis la définition d'un langage évolué de description des systèmes logiques. Tout le système d'aide à la conception s'ordonne alors autour du langage, qui peut aussi, autre avantage, servir de standard et de moyen de communication entre des équipes différentes.

Les vérifications de bon fonctionnement s'effectuent en compilant une description dans ce langage en un programme qui simule le système décrit.

Suivant le niveau auquel on se place, on simulera donc respectivement la propagation des signaux dans un réseau d'opérateurs logiques, des transferts entre registres et des microinstructions, l'exécution de programmes sur la machine décrite ou les fonctions du système de celle-ci.

Or il est assez facile de montrer que la durée d'une simulation au premier niveau est à peu près 100 000 fois plus grande que la durée simulée et au deuxième niveau 1000 000 de fois plus grande.

S'il est concevable de simuler des microprogrammes à raison d'une microinstruction par seconde, il est par contre absurde de vouloir simuler des programmes à raison d'une instruction par minute.

C'est pourquoi CASSANDRE ne couvre que les deux premiers niveaux logiques de description, d'autres outils étant nécessaires aux niveaux supérieurs.

Les méthodes employées par les ingénieurs pour les problèmes logiques sont celles de l'algèbre de Boole et de l'algèbre séquentielle. Nous avons cherché à voir en partie C comment intégrer ces méthodes dans un système d'aide à la conception.

De nombreuses manipulations sans bases scientifiques nous ont semblées se ramener à un traitement de réseau, et nous avons développé en partie D quelques idées à ce sujet.

Ensuite nous avons cherché à montrer comment les outils puissants développés pour la compilation des langages de programmation pouvaient se transposer dans le domaine du hardware ou des microprogrammes ; c'est sans doute la partie la plus originale de notre étude.

Tout ceci est illustré à l'aide d'une machine assez complexe décrite en partie B et qui a servi de sujet d'expérimentation

Enfin, bien que de nombreux travaux aient été publiés récemment sur le sujet que nous avons étudié, selon RALPH, J. PREISS "la Recherche Relative à un Système Conversationnel d'aide à la conception à la fois simple et efficace, reste à faire". (5)

Nous espérons que notre étude aura contribuée à rendre caduque son point de vue.

BIBLIOGRAPHIE

1 - J. KUNTZMANN, "Théorie des réseaux",DUNOD, Paris 1972.

2 - J. MERMET, "Définition du Langage CASSANDRE", Thèse de Docteur Ingénieur, Université de Grenoble, Mars 1970.

3 - D. LITAIZE, "Tracé automatique de Plaquettes de circuits imprimés en Technologie Multicouche Haute Densité", rapport final contrat C.R.I. n° 70015, Octobre 1972.

4 - C. LE FAOU, "Problèmes d'analyse numérique rencontrés dans le programme IMAG2", Séminaire d'Analyse Numérique, Université de Grenoble, Mathématiques Appliquées, Février 1972.

5 - M.A. BREUER,"Design Automation of Digital Systems" Prentice Hall 1972.

A - DEFINITION DU LANGAGE CASSANDRE

I - ELEMENTS DE BASE DU LANGAGE

a) Avertissement

Le langage que nous décrivons dans cette partie a fait l'objet de plus de trois ans d'utilisation. Comme on pourra le constater, la version actuelle est seulement une extension de celle décrite dans | 1 |, ce qui prouve qu'elle a bien résisté à l'épreuve de l'expérience pratique et ce qui nous permettra d'alléger l'exposé en excluant toute description formelle de la syntaxe pour laquelle on pourra se reporter à | 1 |. Les principales différences seront les suivantes : on changera les symboles ⊥ en ⫽ et ¬ en -. On parlera des expressions d'état et d'horloge décrites par Anceau dans | 2 |. On insistera sur la sémantique et l'usage des spécifications Mémoire, Bus, Etat initial qui restaient des points non approfondis dans | 1 |. On supprimera l'opérateur noté précédemment Δ(le symbole Δ restant), en demandant à l'utilisateur de décrire lui-même sous forme d'unité tout décodeur autre que le banal décodeur d'adresse noté ⫽. Enfin on tirera la leçon d'une erreur instructive des étudiants au cours des travaux pratiques de CASSANDRE : ils ont utilisé plusieurs fois des instructions ou des expressions conditionnelles commençant par 'Si' ⫽<expression> 'alors' ... qui n'existent pas dans le langage mais se trouvent très pratiques dans les machines microprogrammées. On les introduira dans le langage

b) Eléments terminaux

Ils sont actuellement au nombre de 100 et leur liste s'établit comme suit :

<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|—|/|↑|↕|∉|≃|Δ|τ|V|Λ|=|≠|>|<|"|
<=|≥|≤|V|Λ|&|*|,|;|:|(|)|[|]|→|←|Φ|ε|×|+|-|÷|Rem|≡|dq|h|o|d|
si|allera|pour|à|faire|de|unité|signal|Registre|impulsion|
début|Etat|Externe|Mémoire|Horloge|Bus|Synchrone|Asynchrone|
Synchronisé|Etatinitial|fin

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> ::= —|/↑|↕|∉|≃|Δ|τ|*|dq

<OPERATEUR-DYADIQUE> ::= V|Λ|=|≠|≥|>|≤|<|∧|∨|&

<SEPARATEUR> ::= |,|;|:|(|)| | |→|←|si|fin|début|allera|pour|de|à|faire|
|"|<=|≡

<DECLARATEUR> ::= Unité|Signal|Registre|Etat|Impulsion

<SPECIFIEUR> ::= Externe|Horloge|Mémoire|Bus|Etat initial

<OPTION> ::= Synchrone|Asynchrone|Synchronise

< SYMBOLE -SPECIAL > ::= Φ|ε|+|-|×|÷|Rem|h|o|d

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

c) Constantes booléennes

Les symboles 0 et 1 sont employés soit comme entiers arithmétiques, soit à la place des valeurs logiques faux et vrai, pour suivre une terminologie usuelle. Nulle part il n'y a ambiguité car les valeurs arithmétiques et les expressions logiques ne peuvent être confondues dans une description CASSANDRE.

On appelle constante booléenne soit 0,1 et Ø (valeur indéterminée) soit un vecteur dont les composantes prennent l'une de ces trois valeurs, soit même un tableau ayant un nombre quelconque de dimensions.

Exemples :

1) (0IØ0011ØØ) vecteur à 9 composantes

2)

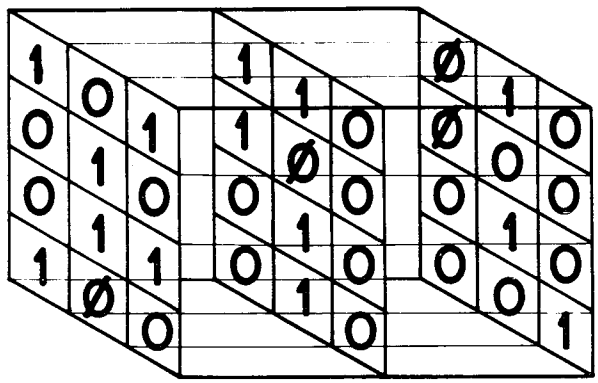


Tableau à 3 dimensions (3,4,3) qui s'écrit :

(101, 0100111Ø0, 1101Ø0010010Ø10Ø00010001)

On "restructure" la constante en retrouvant ses dimensions de la façon suivante :

α) Soit x_1 le nombre de composantes avant la première virgule, x_2 le nombre avant la 2ème virgule, x_3 avant la 3ème....

β) La première dimension est égale à x_1, la deuxième est égale à x_2/x_1, la troisième x_3/x_2 etc...

3) Ecriture abrégée des constantes booléennes

α) $\varepsilon(x_1,x_2,\ldots,x_n)$ est un tableau à n dimensions respectivement égales à $x_1,x_2,\ldots,x_n$ dont toutes les composantes sont égales à 0.

- $\varepsilon(x_1,x_2,\ldots,x_n)$ est le même tableau complémenté, donc toutes ses composantes sont égales à 1.

Exemple :

$\varepsilon(3,4,3)$ = (000,000000000,000000000000000000000000)

β) h(NOMBREXA) où NOMBREXA est un nombre écrit en base hexadécimale, est une constante égale à l'écriture en binaire de ce nombre.

Exemple :

h(77,FFFF) = (01110111,1111111111111111)

γ) O(NOMBROCTAL) a le même effet pour un nombre, NOMBROCTAL, écrit en base 8.

Exemple :

O(77,333333) = (111111,011011011011011011)

δ) d(NOMBREDEC) a le même effet pour un nombre, NOMBREDEC, écrit en décimal

Exemple :

d(77) = 1001101

Ces abréviations sont une économie d'écriture appréciable lorsqu'en particulier on simule des microinstructions.

SEMANTIQUE

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

On en verra des exemples unité COMPTEUR ( ) lignes (8 & 10) de la description du Z0001, page |B 13|.

Elles peuvent constituer le contenu immuable de registres utilisés dans certaines opérations ou d'une mémoire morte. Exemples dans le codage des champs de la microinstruction de Z0001, partie B, III, b.
Elles peuvent également apparaître en condition logique pour tester la valeur d'une variable vectorielle.

Exemple :

si $\Lambda/(R = (01101\ 10))$ alors ...

ceci est plus concis que la forme :

$-R(0)\Lambda R(1)\Lambda R(2)\Lambda -R(3)\Lambda R(4)\Lambda R(5)\Lambda -R(6)$ (pour la signification des opérateurs voir partie (III)

d) Nombres entiers, notions arithmétiques

Dans le langage les entiers positifs ou négatifs sont utilisés comme indices (Boucles pour), comme paires de bornes (variables de type tableau), devant certains opérateurs formels (nombre de positions de rotation permutation de dimensions) et comme indication de durée (nombre d'intervalles de temps) devant l'opérateur de retard τ. Partout où l'on peut trouver des entiers on permet de mettre également une expression arithmétique dont la valeur sera toujours un entier en CASSANDRE.

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.

1) Expressions arithmétiques

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

Elles sont obtenues par combinaison de variables arithmétiques, et des opérateurs +, -, ×, ÷, Rem, avec les règles de priorité d'ALGOL c'est-à-dire deux niveaux plus (+) ou moins (-) et multiplié (×) ou divisé (quotient entier ÷, Reste modulo la division entière (Rem)).

Exemple :

Si C(I) alors OP(I :3÷I)

tiré de l'unité DECG dans la description du Z0001, page |B17|

montre un entier dans un identifieur, en indice et deux expressions arithmétiques en paire de bornes.

2) Expressions arithmétiques de relation, instruction sia

On peut comparer deux expressions arithmétiques à l'aide des opérateurs de relation =, ≠, >, ≥, <, ≤ . Le résultat de cette comparaison est la valeur logique vrai ou faux. On peut l'utiliser comme condition dans une instruction conditionnelle arithmétique. De telles instructions servent à noter dans une structure répétitive décrite par des boucles pour les particularités géométriques du système qui apparaissent pour certaines valeurs des indices ou certaines relations entre ces indices.

(Voir exemple suivant page (A-7) et Z0001 partie C page (101)

3) Instruction pour, déclaration de variable arithmétique

Les variables arithmétiques, repérées par des identificateurs, (voir e), sont déclarées chacune à l'aide d'une instruction du type :

pour <variable-arithmétique>=<borne inférieure>à
<borne supérieure> début <texte> fin

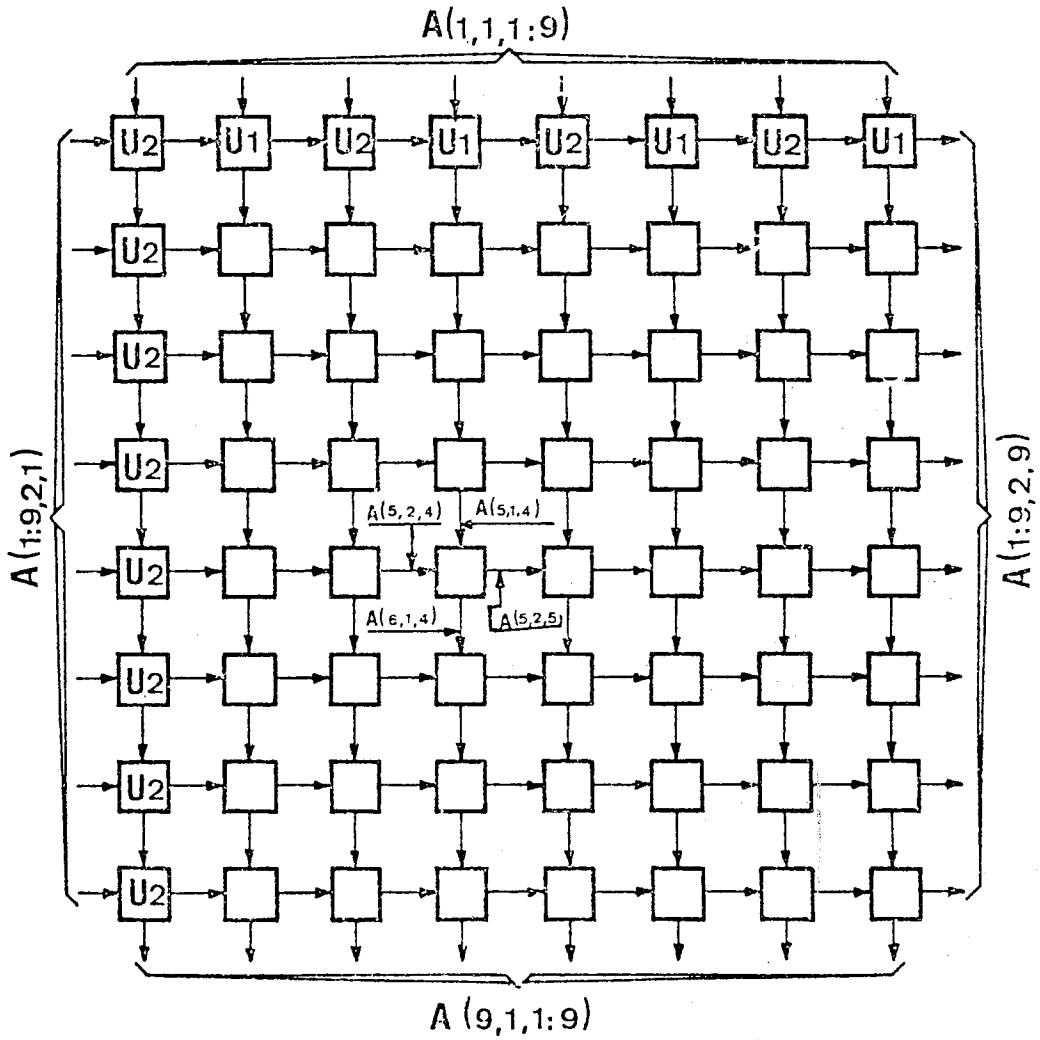
Cette variable prend toutes les valeurs entières situées entre sa borne inférieure et sa borne supérieure (correspond à "pas 1" en Algol 60), sa validité est limitée au texte situé entre début et fin. Une autre variable arithmétique pourra sans risque de confusion être repérée par le même identificateur si elle apparaît hors de cet intervalle.

Les indices et les notions arithmétiques permettent de décrire des structures symétriques ou répétitives avec un minimum de texte. Elles définissent une sorte de "jeu de construction" du texte et n'ont pas de signification propre en hardware.

Exemple :
.....
(pour la signification des déclarations et connexions d'unité voir dans la suite respectivement II b5, II b2, III b5).

```
Externe U1(1,1;1,1), U2(1,1;1,1);
Signal  A(1:9,1:2,1:9);
Pour    I=1 à 8 début
        Pour J=1 à 8 début
              sia(2 Rem (J+(I-1)×8)=0)
        début U1(A(I,1,J),A(I,2,J); A(I+1,1,J),A(I,2,J+1))
        fin sinon début U2(A(I,1,J),A(I,2,J) ; A(I+1,1,J),A(I,2,J+1))
              fin
```

On créé une matrice 8 × 8 dont les noeuds sont alternativement U1 et U2 car la condition constituée par la relation |2Rem (J+(I-1)×8)=0| vaut alternativement vrai ou faux.



* - Autre exemple : voir unité DECG page B17.

e) Identificateurs et variables logiques

Les identificateurs commencent par une lettre et sont formés d'une suite de lettres ou chiffres.

Outre les variables arithmétiques déjà vues ils servent à nommer les variables logiques de type registre, signal, impulsion, état, unité ou étiquette.

Ces variables peuvent prendre 3 formes :

signal-unité que nous verrons par la suite partie III b5 µ.

variable simple et variable indicée. Dans ce dernier cas les indices ou paires de bornes suivent entre parenthèses le nom de la variable.

Nous désignerons 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. (voir partie II).

- 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).

- En effet l'opérateur $ s'applique à un vecteur ou à une expression booléenne réductible à un vecteur (tableau à une dimension). Il interprète ce vecteur comme un nombre et équivaut à l'indice dont le codage en binaire serait ce nombre. Son emploi est récursif, exemple : $ A_3(E1,$ B2(3,0:5))

est une écriture permise équivalente à /$ A3(3,0:7,25) qui contient un sous vecteur de A3. (voir exemple 6 suivant).

La correspondance hardware de cet opérateur est un circuit de décodage d'adresse ayant 2^n sorties pour n entrées. Son coût croît donc exponentiellement avec n, il convient d'utiliser /$ avec précautions et seulement lorsque le décodeur qu'il représente est indispensable. Nous verrons celà à propos de l'adressage des mémoires (page A18, A19 et A 20 partie II b)

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 tableau de dimensions (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.

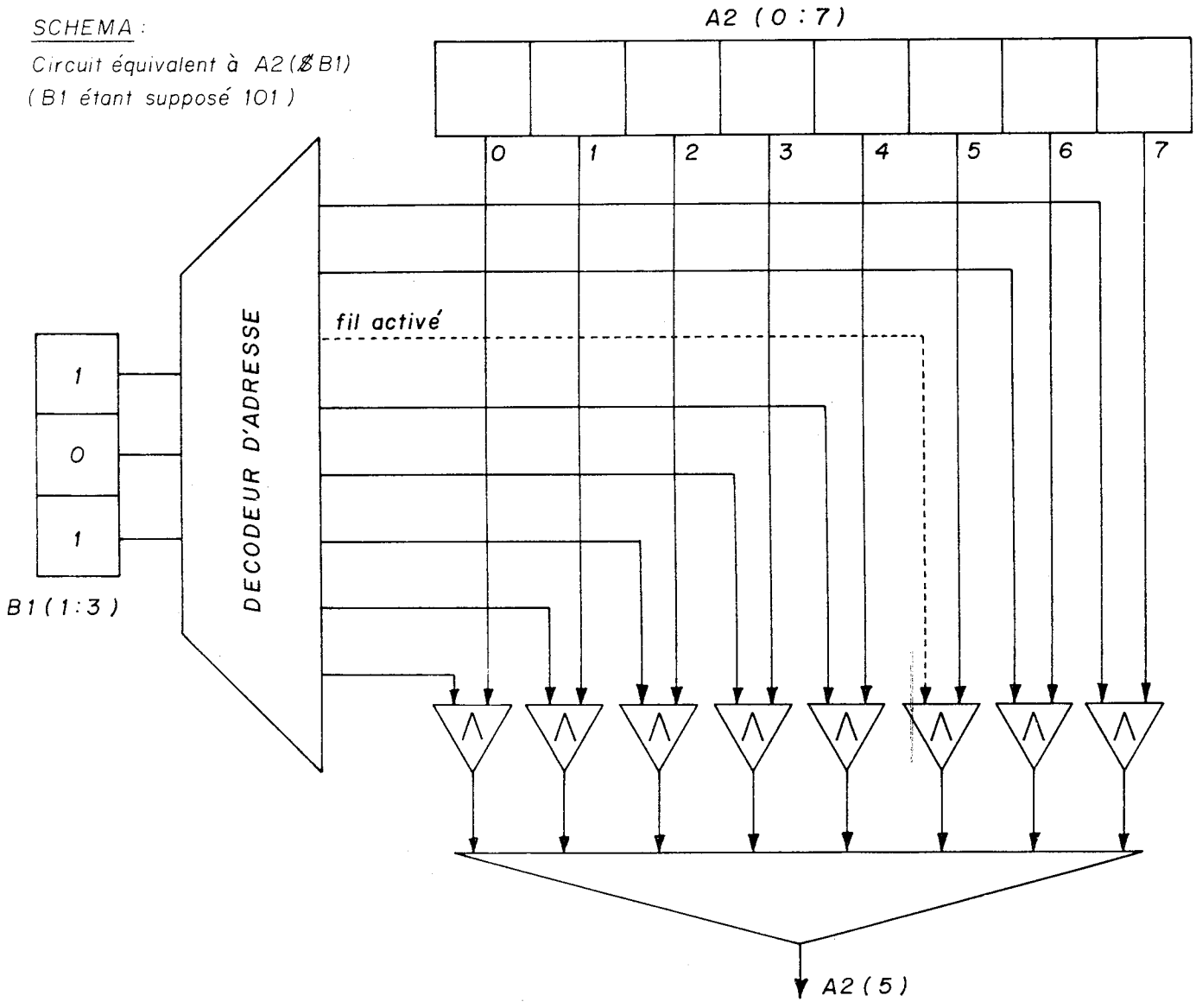
3) A2(E) où E est une expression arithmétique qui peut prendre une valeur entre 0 et 7.

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

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

6) 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. Dans les zones vides on prend en compte toutes les composantes définies par la déclaration (ici 0:7).



II - NOTIONS STRUCTURELLES

La description d'un système digital d'une certaine importance doit être segmentable pour plusieurs raisons simples.

D'abord l'importance du système que le concepteur peut étudier et décrire d'un seul coup est limitée en volume. Ensuite l'importance des fonctions qu'un éventuel programme (souvent à base de combinatoire dans ces problèmes) peut traiter, est limitée en nombre de variables si l'on veut rester dans des temps acceptables. Enfin la nature même de ce que l'on décrit révèle un découpage en réseaux de boites interconnectées eux-mêmes contenus dans des boites, qui forment à leur tour un nouveau réseau etc...

En effet en regardant un système digital existant on verra des portes logiques interconnectées formant des réseaux logiques. Ceux-ci sont alors chacun contenus dans des boitiers eux-mêmes interconnectés sur des "cartes logiques". Ces "cartes logiques", à leur tour, interconnectées sur des panneaux de cablage, donnent des "tiroirs" réunis au sein d'"armoires" qui sont entre elles reliées par des cables etc... jusqu'aux réseaux vraiment très gros, d'ensembles informatiques reliés par télétransmissions.

Nous avons voulu qu'une description en CASSANDRE vérifie aussi bien les contraintes de segmentation que les particularités de structure des systèmes décrits. Donc, contrairement à certains langages (comme celui de CHU) où une description de système logique se présente comme un tout, une description CASSANDRE est divisée en autant de morceaux aussi petits ou grands que l'on veut. Ces parties sont appelées Unités, la plus simple peut se réduire à un bout de fil (une entrée et une sortie), la plus complexe n'a pas de limite théorique de complexité.

- Chaque unité peut contenir un réseau d'unités interconnectées.

- Il existe une unité qui contient toutes les autres.

- Une description en CASSANDRE peut être considérée comme un arbre dont chaque noeud est un réseau de boites.

Il y a une conséquence fondamentale de cette propriété :

- Chaque unité peut à son tour être considérée comme la racine d'un sous-arbre, donc, avec l'ensemble des unités qu'elle contient, comme une description CASSANDRE complète et indépendante.

Ceci permet de limiter un traitement donné à l'ensemble logique minimum qu'il faut considérer et permet aussi dans un ensemble complexe de faire un effet de zoom sur une partie de l'ensemble en prenant cette partie comme nouveau tout.

L'unité, module CASSANDRE de base est un texte qui contient principalement deux descriptions de natures différentes :

α) Partie structurelle comprenant
- Entête d'unité avec ses entrées sorties
- Déclaration propres à l'unité
- Connexions permanentes et inconditionnelles.

β) Partie fonctionnelle :
C'est la description des fonctions logiques que cette unité est censée effectuer, description qui n'implique aucune réalisation particulière.

Réaliser une unité, c'est donc remplacer cette deuxième partie fonctionnelle par un réseau, donc une liste de connexions permanentes et inconditionnelles,donc une partie structurelle : nous verrons ceci partie C.

Dans ce chapitre nous ne traitons que des notions structurelles.

a) Entête d'unité

L'unité en CASSANDRE correspond à la notion hardware de "boite noire", c'est à dire de module dont l'interface avec l'extérieur se réduit à un certain nombre de "bornes servant de support aux connexions possibles. Vues de l'extérieur ces bornes sont repérées par des noms sans signification par rapport au contenu de la boite mais on peut les identifier par leur position pour connaître leurs fonctions. Ces fonctions seront décrites à l'intérieur, les bornes auront cette fois un nom qui apparaîtra dans la définition de ces fonctions. L'interface de l'unité sera désigné de l'extérieur par une déclaration d'"externe" que nous verrons dans le paragraphe suivant et à l'intérieur de l'unité elle-même par son entête c'est à dire son nom, suivi d'une liste d'entrées et d'une liste de sorties.Nous en donnerons deux exemples tirés de la description du Z0001 où on pourra en trouver d'autres si l'on veut.

'Unité' Z0001 (H,I,ST(1:16) ;

liste d'entrées

SSPES (1:16), CECH (1:3), CM (1:3)) ;

liste de sorties

'Unité' LMEM (I2,I3,D(0:3), AD12 (1:3), AIC,SS,COM12; O(0:3), Q(0:3)TCC(1:4));

Les entrées et sorties sont des variables en CASSANDRE et peuvent donc être des tableaux. Le nombre des connexions d'une unité avec l'extérieur doit donc se calculer en multipliant les dimensions d'une variable entre elles et en ajoutant ce résultat à ceux des autres variables.

L'entête de l'unité décrit son interface avec l'extérieur. Mais la description de sa structure interne et de ses fonctions fait appel à un certain nombre de variables qui sont associées au matériel employé et qu'il faudra énumérer exhaustivement en une liste de déclarations situées après l'entête. La description d'une unité 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) Autres unités.

Nous avons déjà vu dans I) que l'on rencontrait également des variables arithmétiques dont nous ne parlerons plus.

Pour mieux distinguer ces variables nous allons les opposer deux à deux.

1) Variables de type 1 et 4

Elles ont toutes deux la nature d'élément de mémorisation.
Mais nous avons d'une part une variable structurée à un certain nombre de dimensions (matrice, vecteur, tableau tridimensionnel...) ayant un nombre fixe d'éléments de mémorisation et que l'on appellera 'Registre' en précisant ses dimensions, d'autre part une variable qui sert à repérer indépendament de tout codage un état d'automate fini. Puisque ce dernier n'est pas codé et puisque deux codages différents risqueraient de comporter des nombres différents de bits on ne peut pas préciser ses dimensions. Le problème "d'assignement des états" revient donc à transformer une variable déclarée état en variable déclarée registre.

2) Variables de type 2 et 3

Elles sont toutes deux relatives à des signaux logiques comme il y en a dans tout circuit. Mais les premières déclarées 'signal' sont des niveaux logiques qui restent établis pendant tout intervalle élémentaire de temps. Les autres déclarées "impulsion" seront supposées infiniment courtes (ce qui n'est pas rigoureusement le cas dans la réalité). C'est leur apparition qui rythme les événements dans le système logique décrit. Elles fixent l'intervalle de temps élémentaire. Signaux et impulsions comme les registres sont des variables structurées dont il faut déclarer les dimensions.

3) Variables de type 1 et 2

Les variables de type registre peuvent garder dans nos hypothèses indéfiniment la valeur qu'elles mémorisent. Elles ne peuvent changer de valeur que sous l'action d'une impulsion. Les variables de type signal sont à un moment donné des fonctions de registres et d'autres signaux. Ce sont donc des fonctions de fonctions que l'on pourrait réduire à des fonctions de variables de type registre en éliminant de proche en proche les signaux qu'elles contiennent.

Réciproquement si un système donné ne contient qu'un nombre fixe et invariant de variables de type registre rien ne s'oppose à ce qu'il contienne un nombre arbitraire de signaux (qui sont des fonctions de ces variables).

b) Déclarations

Exemple

```
'UNITE' Z0001(H,I,RAZ,E1(1:16);CM(1:3),CECH(1:3),ST(1:16));
  'SIGNAL' TE(1:8),E1(1:4),E2(1:4),S(1:4),AD1(1:4),D1(1:4),AD2(1:4),D2
  (1:4),SR1(1:16),SR2(1:4),DP1(1:4),DP2(1:4),CONS(1:4),COMMANDE(1:15),
  TEST(0:9),AIG,CDEC,SS,CSP,CE1(1:2),CE2(1:2),CR(1:2),CVAL(1:4),AD12(1
  :3),ADD(1:2),CONS2(1:2),CC(1:2),CONTR(1:43),SSPES(1:16);
  'HORLOGE' H,I;
  'IMPULSION' I1,I2,I3;
  'EXTERNE' VAL3(3,8),
            EMEM('HORLOGE','HORLOGE',4,3,,,;4,4,4),
            CONTROLE('HORLOGE','HORLOGE','HORLOGE',4,10,,8;43,3,3),
            VAL('HORLOGE','HORLOGE',4,4,4,,;4,,),
            DEC('HORLOGE',,8,4,4,2,4;4,4),
            SPES('HORLOGE','HORLOGE',16,16,4,,,;16),
            RI('HORLOGE','HORLOGE',2,,2,3,2,4,16;16,4,4),
            HORLOGE('HORLOGE','HORLOGE';'HORLOGE','HORLOGE','HORLOGE');
              VAL3(1)(AIGCDE1;);
              VAL3(2)(AIGCDE2;);
              HORLOGE(H,I;I1,I2,I3);
```

1) Variables de type 1

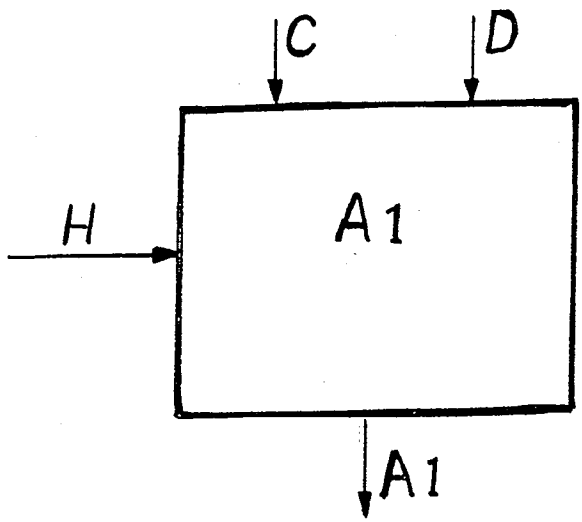
Le symbole 'Registre' est suivi de la liste des variables du type accompagnées de leurs dimensions (qui sont données par une liste de paires de bornes).

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 logiques, elles-mêmes réalisées avec des transistors M.O.S.

C'est une boîte noire, comme une unité vue de l'extérieur et on pourrait supprimer la déclaration registre au profit d'une externe si l'on décrivait comme unité la réalisation de chaque case du registre.

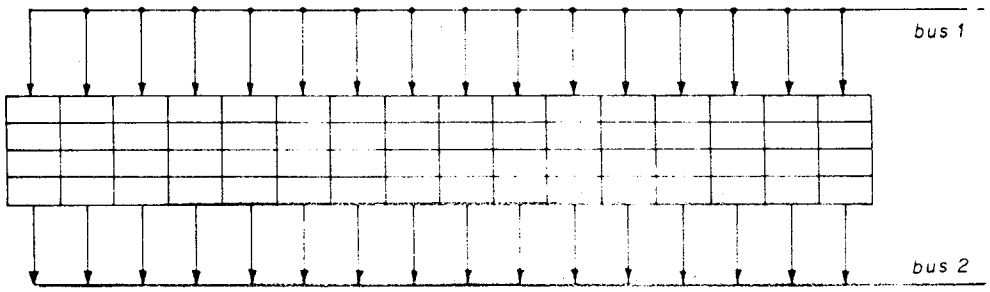
Nous admettrons simplement que le module correspondant à une bascule a 3 entrées : impulsion H, donnée D et condition C (voir LIDDELL | 3 |)

L'impulsion est celle qui conditionne les changements de valeur de la bascule. La condition peut inhiber l'horloge, elle peut aussi être inemployée. (On la remplace alors par 1)



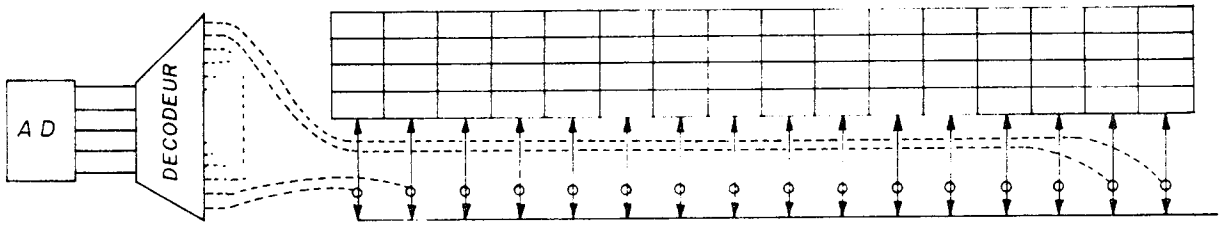
Cette déclaration de variables de type 1 peut dans certains cas être précisée par une spécification mémoire, dont nous allons expliquer l'emploi.

Le symbole mémoire est suivi d'une liste de variables déjà déclarées registre, ayant au moins la dimension d'un vecteur.
Considérons à titre d'exemple la variable déclarée M(0:3,0:15) .



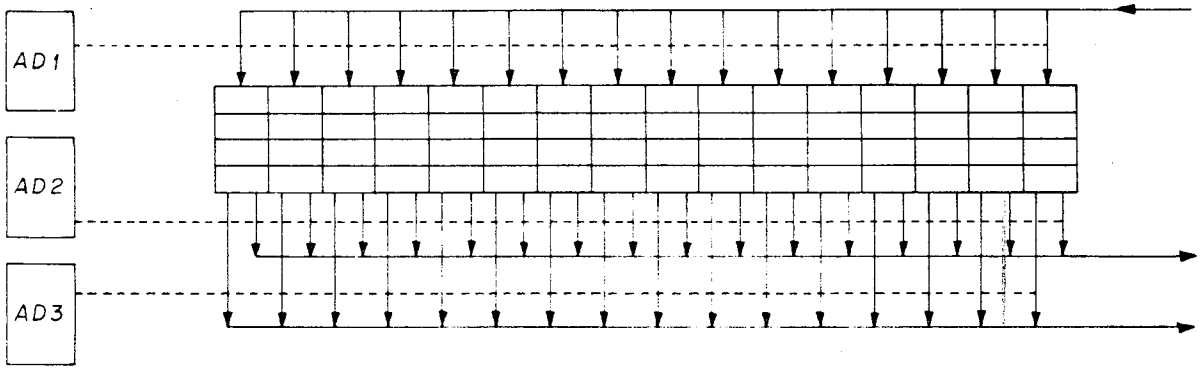
On peut si on le veut accéder en lecture ou en écriture à chaque bit de cette variable mais cela créeera un nombre considérable de connexions. On peut décider pour simplifier les opérations de ne manipuler que des morceaux de 4 bits que l'on appellera des mots, il y en a un ensemble de 16. On peut de plus décider de n'accéder qu'à un seul mot à la fois (schéma ci-dessus). Pour cela 16 signaux de contrôle seront nécessaires pour désigner quel mot est considéré dans l'opération, mais comme un seul de ces mots sera pris à la fois on peut coder ces signaux de contrôle sur 4 bits qui deviennent ainsi l'adresse du mot. Soit AD(0:3) cette adresse dans notre exemple, un mot sera désigné par M(0:3,$ AD) (voir paragraphe précédent pour $).

On peut aller plus loin en décidant de ne pas lire un mot en même temps qu'on l'écrit ou qu'on en écrit un autre, dans ce cas le seul registre d'adresse déjà vu AD suffira pour toutes les opérations et le schéma devient :



La notion d'adressage introduite par l'opérateur $ ne suffit pas à définir une mémoire. Il faut ajouter que la lecture et l'écriture à un instant donné sont incompatibles et encore qu'on ne lit ou écrit qu'une seule case à la fois.

C'est cette organisation, avec toutes les restrictions que nous avons énumérées que nous associerons à la spécification mémoire. Bien entendu il existe d'autres organisations qui sont quelquefois appelées mémoire, par exemple :



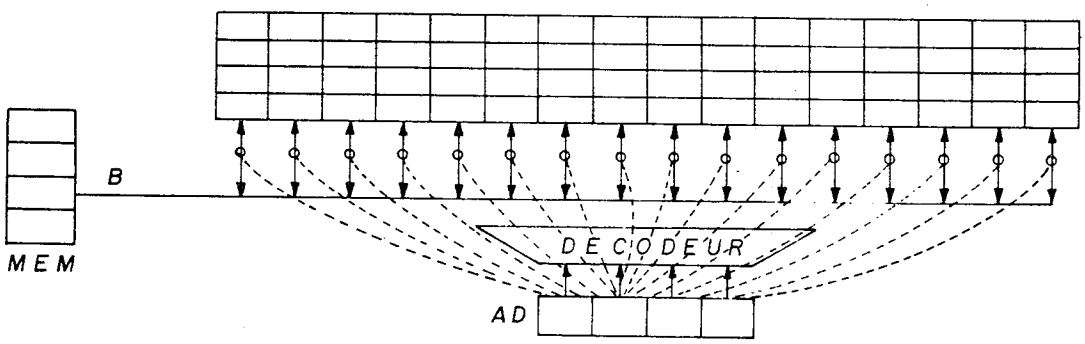
Le même registre associé à 3 registres d'adresse AD1, AD2, AD3 en permettant simultanément de lire deux mots et d'en écrire un troisième, que l'on désigne par M(,$AD1), M(,$AD2), M(,$AD3). Cette structure existe dans le commerce, mais beaucoup d'autres sont imaginables. Nous ne les appellerons pas mémoire par une décision un peu arbitraire, mais qui englobe quand même la quasi-totalité de ce que l'on a appelé "mémoire" jusqu'à ce jour.

La spécification mémoire est donc un quintuplet comportant le registre d'adresse, le registre d'entrée-sortie s'ils existent, le bus unique si cela est utile, et les temps de lecture et d'écriture exprimés en intervalles de l'impulsion qui synchronise les écritures de la variable spécifiée.

Exemple
.....

Mémoire M(AD; * MEM,B,2,3) ;

Registre AD(0:3),MEM(0:3),M(0:3,1:16);



L'adresse peut être obtenue par concaténation de plusieurs registres d'adresses, la spécification ne sera plus un quintuplet car une liste de registres remplace alors le registre d'adresse.

Exemple : base B, déplacement D :

Mémoire MM(B,D;BUF,BUS,3,3) ;

Registre B(1:4),D(1:6),BUF(1:8) ;

dimensions devra contenir au moins un symbole $, de plus elle devra être compatible avec son registre d'entrée-sorties.

Exemple
.....

M(,$AD) se réduit à un vecteur de 4 bits comme le registre MEM

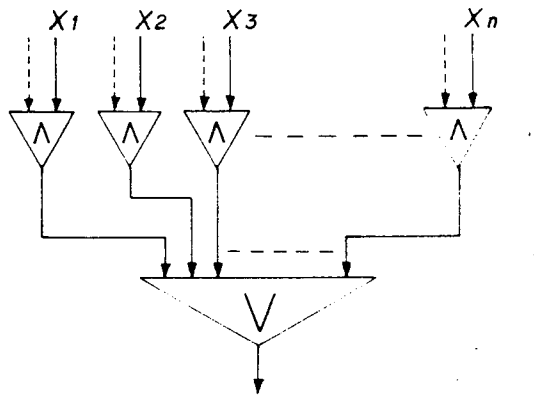
MM(,$B,$D) est un octet comme BUF, c'est l'un des 16x64 octets de MM, adressés en poids forts par B et en poids faibles par D.

2) Variables de type 2

Le symbole signal est suivi de la liste des variables du type avec leurs dimensions (voir exemple ci-dessus). Nous avons vu la signification des signaux leur correspondance hardware est un ensemble de fils ou de bornes. Certaines variables déclarées 'signal' peuvent être spécifiées 'bus', nous avons déjà vu des bus dans le

paragraphe précédent, nous allons préciser leur sens en CASSANDRE.

Soit un circuit ET/OU à n entrées et une sortie (schéma ci-dessous)



α) Porte

Considérons l'un des opérateurs ET à 2 entrées, soit x l'une de ces entrées, l'autre étant une condition logique. (Il est presque toujours possible dans les systèmes que nous considérons de distinguer entre variables "données" et variables "de contrôle"). La condition logique peut être par exemple, exprimée en français, "l'adresse contenue dans AD vaut 12". Si la condition logique est vraie la sortie vaut x, si elle est fausse la sortie vaut 0.
(on pourrait raisonner aussi avec un opérateur NI à 2 entrées).

"Une porte est un opérateur ET à deux entrées dont l'une des entrées s'appelle commande et l'autre donnée. Si la condition logique commande est vraie la sortie est égale à l'entrée donnée, la porte est "ouverte", sinon la porte est "fermée", "la sortie vaut 0."

β) Fonction 1 parmi n :

Supposons dans le schéma ci-dessus que les opérateurs ET soient des portes et que leurs conditions s'expriment par exemple : "l'adresse contenue dans AD vaut 0","l'adresse contenue dans AD vaut 1,... etc "l'adresse contenue dans AD vaut 15". Ces conditions forment des commandes disjointes, jamais deux d'entre elles ne sont vraies en même temps. La sortie de l'opérateur OU est toujours égale à la donnée dont la commande est activée ou à 0 s'il n'y en a pas. La fonction réalisée n'est donc jamais la fonction booléenne OU, mais la fonction "1 parmi n". Ce circuit permet d'aiguiller l'une des entrées et une seule vers la sortie que nous appellerons et pourrons spécifier 'bus'. (on pourrait aussi l'appeler multiplexeur).
La spécification 'bus' peut comporter le nom de la variable de commande si elle est repérée explicitement.

Exemple
....

'Signal' B(0:7),COM(1:16),AD (0:3) ;

on peut avoir soit : 'bus' B(COM);

soit encore: 'bus' B(# AD);

3) Variables de type 3

Elles sont déclarées comme les précédentes mais avec le symbole 'impulsion'.

Dans le cas d'impulsions venant de l'extérieur de l'unité considérée on les appellera horloge. La spécification 'horloge' repère des impulsions fournies à l'unité par le monde extérieur.

4) Variables de type 4

Les variables de type état se déclarent comme les précédentes mais avec le symbole 'état' et sans la première dimension (pour des raisons de non-codage déjà vues).

Un état distingué de l'unité peut être spécifié 'état initial' (en lui associant éventuellement un signal). Cela signifie que sous l'action d'une commande de "remise à zéro générale" l'ensemble des registres de l'unité est remis à 0 et la variable interne est positionnée sur la valeur spécifiée.

Exemple
....

'Etatinitial' ETAO(RAZ);

Remarque :
La variable interne de toute unité séquentielle n'est pas déclarée, elle porte implicitement le même nom que cette unité. De même les étiquettes qui repèrent des valeurs de cette variable interne ne sont pas non plus déclarées (encore que l'une d'elles puisse être spécifiée 'étatinitial').
La déclaration 'état' ne concerne donc que des éléments de mémorisation susceptibles de stocker des valeurs de la variable interne. Alors en désignant ces variables d'état dans une expression d'état c'est leur contenu (qui peut varier) que l'on désignera.

Un ensemble de variables d'état peut se regrouper en tableaux dont on ignore simplement la première dimension.

Exemple :
....

Voir III d 3

```
unité RSPES
'Etat' STATE(,1:4,1:4);
```

5) Variables_de_type_5

Toute unité utilisée dans une autre doit être déclarée 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. (voir également exemple Z0001 ci-dessus).

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.

c) Remarques

- La définition d'unité permet de décrire entièrement un organe logique et son interface avec l'extérieur.

- Le matériel utilisé dans cette unité est listé dans toute une série de déclarations, complétées par des spécifications.

- Les variables utilisées dans l'interface de l'unité avec l'extérieur sont de type signal. (Elles ne sontpas redéclarées signal dans l'unité, ce serait redondant). Certaines d'entre-elles peuvent être de type impulsion, (dans ce cas elles sont spécifiées 'horloge' dans l'unité).

- Un certain nombre d'entrée-sorties sont implicites dans l'entête d'unité ce sont

  α) les alimentations

  β) les sorties du registre d'état interne de l'unité (lui-même implicite), pour les tester

  γ) éventuellement des fils d'entrée à ce registre si l'on veut forcer un état de l'extérieur.

- Un certain nombre de variables apparaissent dans la description sans être déclarées, parce qu'elles sont repérées sans équivoque par leur emplacement. Ce sont les variables-arithmétiques (boucles 'pour'), les valeurs d'état 'étiquettes' et les signaux de l'interface (entête d'unité).

- La portée de tous les identificateurs utilisés est limitée à l'unité. On peut sans inconvénient les réutiliser pour d'autres variables dans d'autres unités.

- Une description d'unité peut ne pas comporter d'entrée ou pas de sortie ou pas de déclarations.

d) Survariables, sous-variables, variables généralisees

Nous allons 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.

Définition de la compatibilité 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 :

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

et

$B(c_1:d_1, c_2:d_2, \ldots, c_j:d_j)$

A et B seront compatibles si l'on a :

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

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

1) Equivalence d'élément de mémorisation

Deux variables déclarées registre ou état peuvent être désignées comme équivalentes si elles se trouvent à la suite des déclarations reliées par le symbole ≡.

Exemple 1

```
Registre A(1:36),ADDR(0:17);
         A(19:36) ≡ ADDR;
```

ou encore page ( ) III d3dans Unité RSPES : X5 ≡ X1, X4 ≡ X0;

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. De même pour les registres d'état X5 et X1, X4 et X0.

2) 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 2

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. On trouvera d'autres exemples de son emploi dans l'unité UAL de Z0001.

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

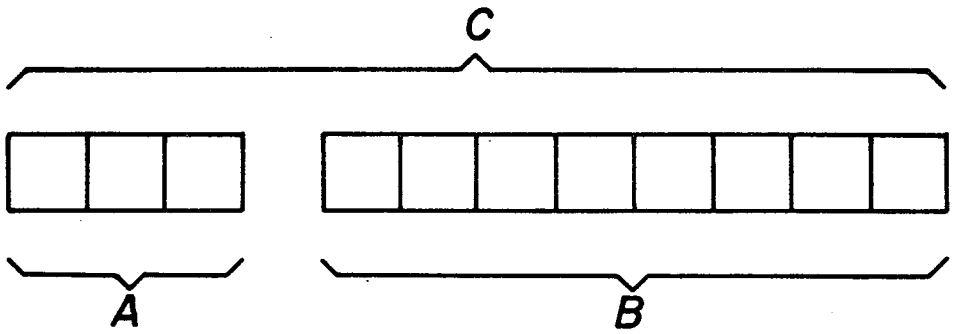
Exemple 3

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 registres A et B mis bout à bout :



3) 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 4

B1(0:4, 1:3, 0:7, 1:36)

2:4 ∿ B1 ≡ B2(0:4, 1:36, 0:7, 1:3)

On permute 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 5

B(0:4, 1:3, 0:7), A(1:8, 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.

4) 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 6

```
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
```

défini une nouvelle variable de dimensions

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

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

Exemple 7

```
Registre E(1:64, 0:2) ;
E ≡ ∿  A & B(2:9, 0:2, 7) & ∿ C(1:3,)& D;
```

III - NOTIONS FONCTIONNELLES

a) Opérateurs et expressions

Nous pouvons diviser les opérateurs employés en CASSANDRE en 3 catégories, ceux qui sont répétitifs et booléens, ceux qui sont booléens mais non répétitifs, ceux qui sont non booléens et introduisent le temps.

1) Opérateurs booléens répétitifs :

Les opérandes sur lesquels agissent ces opérateurs sont des variables en CASSANDRE, donc des tableaux de dimension quelconque. Les opérandes d'un même opérateur doivent être compatibles (voir paragraphe précédent) , dans ce cas l'opération booléenne s'applique bit à bit sur les composantes correspondantes.

Exemple : dans 'Unité' UAL (description de 20001) on trouve :

$L \leftarrow OP_1 \neq OP_2$; cette opération est possible car les déclarations indiquent : $L\ (0:3)$, $OP_1\ (0:3)$, $OP_2\ (0:3)$

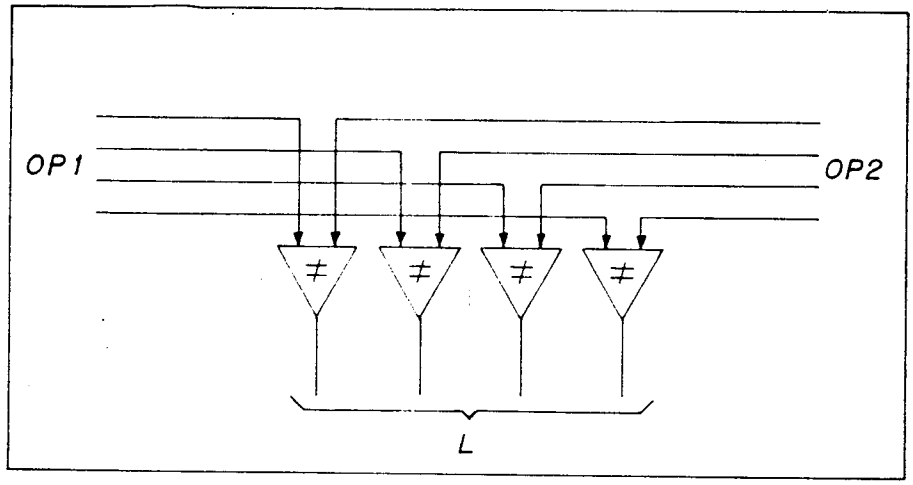
L'opération est équivalente à

$L(0) := OP_1\ (0) \neq OP_2\ (0)$;

$L(1) := OP_1\ (1) \neq OP_2\ (1)$;

$L(2) := OP_1\ (2) \neq OP_2\ (2)$;

$L(3) := OP_1\ (3) \neq OP_2\ (3)$;



Toutes les fonctions booléennes de deux variables sont représentées par un opérateur CASSANDRE.

Négation notée —

OU logique notée V

ET logique notée Λ

NI noté ⩛

NAND noté ⩚

et les opérateurs de relation. $>, \geqslant, <, \leqslant, =, \neq$

Tous ces opérateurs peuvent se représenter en fonction de sous-ensembles d'entre-eux que l'on appellera une base. Citons 3 bases couramment utilisées

$(\Lambda, V, -)$ $(\barwedge, -)$, $(\barvee, -)$.

Nous avons donné dans [1] les représentations dans ces bases.

2) Opérateurs booléens non répétitifs :

α) Opérateur condition

La notation 'si' <condition logique> 'alors', <expression 1> 'sinon' <expression 2> , exprime l'intersection de la variable à une seule composante <condition logique> avec chacune des composantes de <expression 1> et l'intersection de — <condition logique> avec toutes les composantes de <expression 2>

Exemple : dans 'unité' SPES (de Z0001) on trouve :

INSP : = 'si' CSP 'alors' ST 'sinon' SR ; qui montre l'emploi de l'opérateur 'si' que nous appellons "opérateur condition".

Cet opérateur permet aussi de "distribuer" n'importe quel opérateur booléen entre une variable à un bit et toutes composantes d'une autre variable.

Exemple : signal A , B (1 : 3, 1: 4), C (1 : 3, 1 : 4);

C : = ('si' A 'alors' — ε(3,4))≠ B;

On a pour chaque composante de C , C(i,j) := A ≠ B(i,j);

L'opérateur condition sera généralisé dans les formes suivantes :

α1) 'si' ∉ <condition> 'alors' (< expression 1>) (< expression 2>) ..... (< expression n >);

Si <condition> est une expression à p composantes on a : $n = 2^p$,

la forme précédente équivaut à :

'si' (<condition> = $\overbrace{00\ldots0}^{p}$) 'alors' (<expression 1>)

'si' (<condition> = $\overbrace{00\ldots01}^{p}$) 'alors' (<expression 2>)

.....

'si' (<condition> = $\overbrace{11\ldots1}^{p}$) 'alors' (<expression n>)

(On peut bien sûr trouver une <condition> à n composantes à la place de ∉ <condition>).

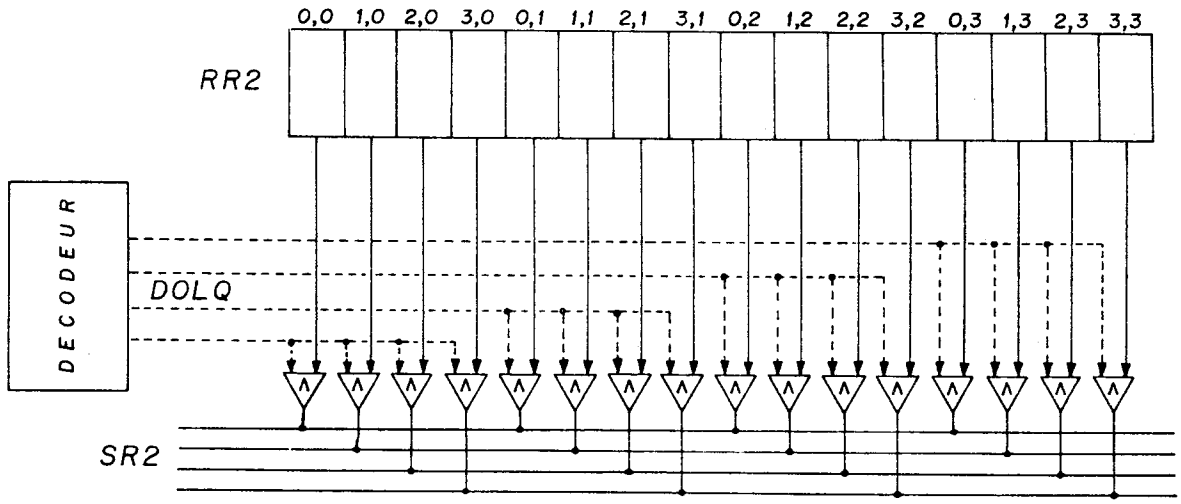
On voit en se reportant à notre définition de 'bus' (page A22)que :

B : = 'si' ∉ C 'alors' (<exp 1>) (< exp2>)....(<exp n>) défini B comme la spécification 'bus' B (∉ C).

On voit aussi que les sorties de exp 1, exp2...etc sont branchées sur ce bus.

Exemple : dans 'unité' R de Z0001 on trouve : 'Bus' SR2 (DOLQ);

SR_2 := 'si' DOLQ 'alors' (RR2(,0)) (RR2(, 1)) (RR2(,2)) (RR2 (,3))



Nous n'avons pas mis d'opérateur OU car on a déjà vu qu'il s'agit en réalité de la fonction 1 parmi n et dans ce cas souvent, la technologie permet de la réaliser par branchement direct.

α2) 'si' ∉ <condition> 'alors' <expression (1:n,...)> ;

Dans ce cas la première dimension de l'expression qui suit 'alors' doit être $n = 2^P$. Comme dans le cas de l'opérateur réduction (paragraphe suivant) cette dimension disparaît dans le résultat qui est équivalent à :

'si' (<condition >= $\overbrace{00...0}^{P}$)'alors' <Expression (1,..,...)> ;

'si' (<condition >= 00...1)'alors' <Expression (2,...)> ;

.....

'si' (<condition> = 11...1) 'alors' <Expression (n,...)> ;

Exemple :dans l'unité TESTT de Z0001 on trouve :

<I> TESST ← 'si' ∉ COND 'alors'&T(1) & E &T(2:7) &-T(7) ;
&T (8:16) &- T(10:16) ;

ou TESST est de dimension 1 et COND de dimension 5.

β) Réduction

La notation / associée à un opérateur booléen binaire est la "réduction" définie par Iverson sur les vecteurs. Nous l'appliquerons à des variables de dimension quelconque dans la première direction. Après l'action de l'opérateur réduction, la première direction sera dégénérée.

Exemple : Dans 'unité' COMPTEUR de ZOOOl on trouve :

'Registre' CT (1:4); Signal TC ;

TC := /∧CT

Ce qui équivaut à: TC := CT(1)∧ CT(2) ∧CT(3) ∧CT(4);

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

Exemple : /⍉ ∿ B(1: 4 , 1: 3)≡ D(1:4)

D(1) ≡ B(1,1) ⊙ (B(2,1) ⊙ B(3,1))

D(2) ≡ B(1,2) ⊙ (B(2,2) ⊙ B(3,2))

D(3) ≡ B(1,3) ⊙ (B(2,3) ⊙ B(3,3))

D(4) ≡ B(1,4) ⊙ (B(2,4) ⊙ B(3,4))

A la différence du précédent, l'opérateur réduction change la structure de la variable sur laquelle il porte. La première direction est réduite à une composante. Bien que les variables A(1:1, i:j, k:1,..., m:n) et B(i:j, k:1,...,m:n) soient isomorphes, nous ne les considérerons pas comme équivalentes, A a n+1 directions si B en a n et en utilisant la concaténation A va générer des variables à n+1 dimensions alors que pour B ce n'est pas possible.

Nous dirons qu'une direction est dégénérée si sa dimension est 1. Si on opère la réduction de toutes les directions d'une variable on obtiendra un tableau à n directions et une seule composante (cas extrême). A l'exception de la concaténation et de la transposition nous supposerons que les opérations s'appliquent comme si les directions dégénérées n'existaient pas (de même que les vérifications de comptabilité). On peut aussi supprimer les directions dégénérées en redéfinissant une variable : Exemple : Registre D(1:4), B(1:4,1:3); Signal A(1:3,1:4,1:5,1:6), C(1:4,1:5; D ≡/ ⍉ ∿ B; C:= / + A(,,,1);

γ) L'opérateur incrémentation sera noté + 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 moins coûteuse 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 AD

<H> 'si' CU & CD 'alors' (AD← +1) (AD← -1);

serait équivalent au compteur CT (dans COMPTEUR de ZOOO1)

δ) L'opération troncature est définie sur des vecteurs.

Exemple : $4\downarrow B(1:8)$ signifie le décalage à gauche de quatre positions de B. Les positions débordant à gauche sont perdues. Comme l'opérateur de réduction la troncature modifie donc la première dimension, mais ne la rend pas dégénérée sauf cas extrême.

- $3 \downarrow B$ signifie le décalage de trois positions à droite.

$4 \uparrow B \quad B(5) \,\&B(6)\&B(7)\&B(8)$

$-3 \uparrow B \quad B(1)\&B(2)\&B(3)\&B(4)\&B(5)$

On peut associer cette opération avec la concaténation si l'on veut rétablir la dimension modifiée : (exemple : 'unité'UAL dans ZOOO1 :

RS4 = $1 \updownarrow 0 \,\&(oP_1 \wedge oP_2) \vee 0 \,\&oP_1 \wedge RS4 \,\& 0 \vee 0 \,\&oP_2 \wedge RS4 \,\& 0$;

ε) L'opérateur de rotation $\updownarrow$ opère comme celui de troncature mais les positions libérées à gauche ou à droite sont remplacées par les positions précédemment perdues.

$4\downarrow B \equiv B(5)\&B(6)\&B(7)\&B(8)\&B(1)\&B(2)\&B(3)\&B(4)$

$-3\uparrow B \equiv B(6)\&B(7)\&B(8)\&B(1)\&B(2)\&B(3)\&B(4)\&B(5)$

Ces deux opérations comme la réduction peuvent s'étendre à chaque dimension d'un tableau de dimensions quelconques. (Toujours à l'aide de l'opérateur $\wr$).

3) Expressions :

α) Expressions logiques

A l'aide des opérateurs précédents on forme en CASSANDRE des expressions booléennes. Comme les variables employées sont des tableaux et comme certains opérateurs peuvent modifier leur structure, il importe de vérifier soigneusement la cohérence des expressions et leur validité.
Un programme assez complexe est chargé de cette fonction. On peut associer un certain matériel à chacun des opérateurs que nous avons définis. En particulier ceux que nous avons désignés pour former des bases ($\wedge$, V, —), ($\forall$, —) et ($\barwedge$, —) se trouvent la plupart du temps réalisés technologiquement.

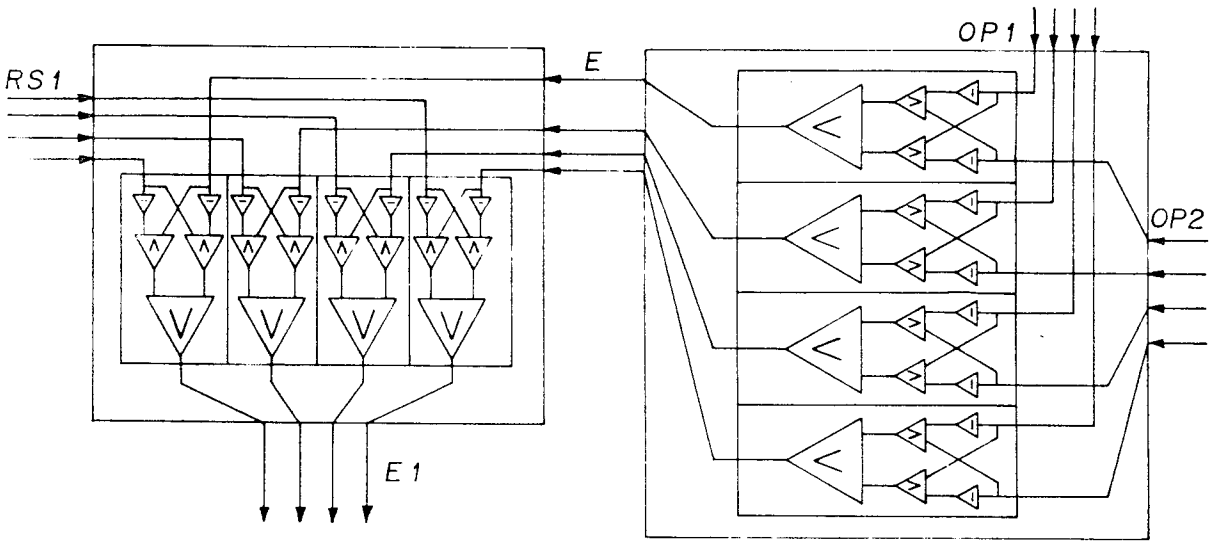
Comme tous les autres opérateurs définis peuvent s'exprimer sur ces bases (concaténation troncature rotation et transposition &, ↕, ↑, ∿ sont purement formels et ne"coûtent" aucun matériel)il en résulte qu'à toute expression on peut associer un circuit logique.

Toute expression CASSANDRE représente un circuit combinatoire et la valeur de la fonction est la sortie de ce circuit.

Exemple : 'unité' UAL (dans Z0001)

on trouve E1 = E ≠ RS1 (1:4) ; E := $OP_1 \neq OP_2$;

On peut leur associer le circuit :



Les opérateurs que nous avons défini s'appliquent tous à des variables de type signal ou registre.

La sortie d'une expressions ainsi formée est toujours un signal.

β) Expressions d'impulsions

Mais nous avons vu qu'il y avait des variables impulsions nous pouvons les utiliser dans des expressions, avec certaines précautions, car tous les opérateurs définis ne conservent pas un sens dans ce cas.

Les opérations permises sont :

β1) V entre deux variables impulsions, le résultat est une impulsion (qui contient les deux précédentes). Si les deux impulsions sont simultanées l'opération est permise mais peut se traduire dans la réalisation physique par deux impulsions légèrement décalées, cause de pannes pour le circuit. (Impossibilité de détecter les aléas éventuels d'un circuit décrit en CASSANDRE).

β2) Connexion d'un signal ou condition d'une impulsion (voir paragraphe suivant).

β3) Λ entre une impulsion et un signal, le résultat est une impulsion "filtrée" par le signal.

β4) Opérateur condition sur une impulsion (la condition étant évidemment un signal ou un registre) le résultat est encore une impulsion "filtrée" par la condition et les opérateurs "formels" $\frac{1}{0}$, ↑ , &, ∿ .

Les expressions formées en respectant ces règles ont toujours pour résultat une impulsion, nous les appellerons expressions d'impulsions. (voir à titre d'exemple l'unité HORLOGE de ZOOOl page (38) partie B.

γ) Expression d'état :

De même nous avons noté l'existence de variables 'état' (et aussi 'd'étiquettes'). Nous pourrons appliquer à ces variables les opérateurs suivants :

γ1) = entre deux variables d'état. Le résultat est un signal ayant valeur de condition logique.

γ2) ≠ entre deux variables d'état. Même chose que précédemment.

γ3) Opérateur condition sur une variable d'état (la condition étant évidemment un signal ou un registre) le résultat est un état.

γ4) Etat forcé : On appelle ainsi un état associé à un nom d'unité par le symbole de base 'de', c'est encore une variable d'état,ex: ETA6 'de' ZOZO.

γ5) Etat composé :On appelle ainsi une expression formée d'un nombre quelconque d'identificateurs d'état (ou d'étiquettes) séparés deux à deux par ":". Exemple : 'Unité' RSPES Paragraphe d3,X1 : Y2 : ECR page A53.

Bien entendu l'emploi des opérateurs formels n'aurait aucun sens puisqu'on ne connait pas la dimension des variables 'état'.

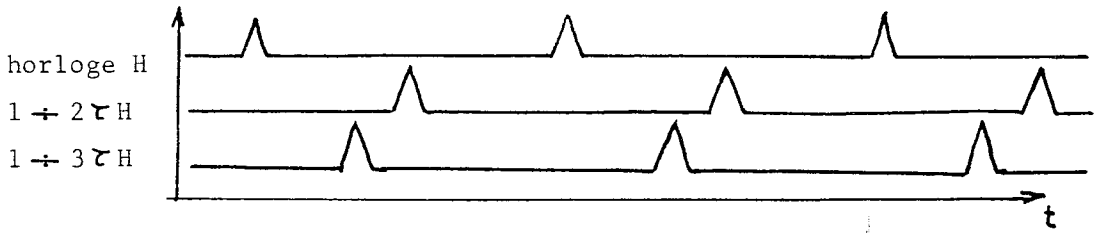
Les expressions formées ainsi ont tantôt une valeur de condition, logique, tantôt une valeur d'état nous les appellerons expressions d'état.

γ6) Si l'on utilise des tableaux de variables d'état, il faut bien sur utiliser une seule **composante** de ces tableaux pour que ce soit une variable d'état,(c'est à dire sans dimensions).

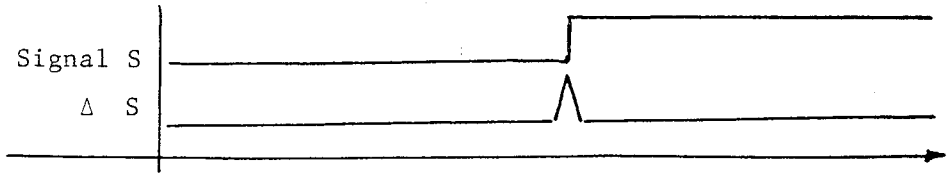
4) Opérateurs asynchrones

Ce sont les opérateurs de retard τ et de dérivation Δ . Ils n'éxistent que dans une description spécifiée en mode asynchrone.

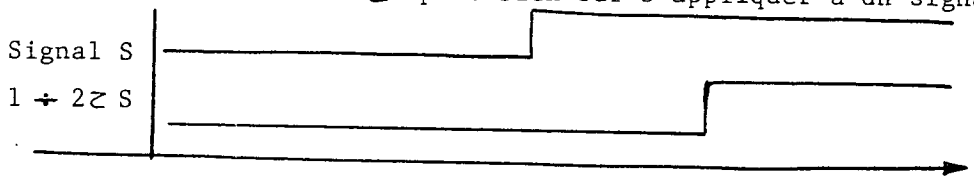
L'opérateur de retard peut agir sur des impulsions pour les retarder d'une certaine fraction de l'intervalle de temps élémentaire. On peut ainsi à partir d'une horloge de base former autant d'impulsions que l'on veut qui subdivisent l'intervalle de base.



L'opérateur de dérivation agit sur des signaux et fabrique une impulsion à l'aide de front de montée du signal. Cet opérateur permet ainsi de décrire le fonctionnement détaillé de certaines bascules qui ne ressortissent pas de la description CASSANDRE synchrone car leur chargement est commandé par un signal et non une impulsion.



L'opérateur de retard $\succeq$ peut bien sur s'appliquer à un signal



L'opérateur $\succeq$ suffit pour introduire en CASSANDRE la notion de durée (qui n'est pas une notion logique).

L'opérateur Δ permet de passer des variables signal aux impulsions.

Le passage d'une impulsion à un signal d'une certaine durée ne pose aucun problème.

b) Opérations élémentaires :

Il existe en Cassandre 7 opérations de base : chargement de registre, affectation d'état, ordre 'allera', connexion de signal, connexion d'unité, ordre 'faire', génération d'impulsion. Les 3 premières reviennent à charger des éléments de mémorisation, les 4 autres à définir des liaisons entre réseaux combinatoires (expressions) ou réseaux d'unité. Ces dernières opérations définissent des variables intermédiaires, qui sont dans le cas général des fonctions de fonctions (et de type soit impulsion, soit signal).

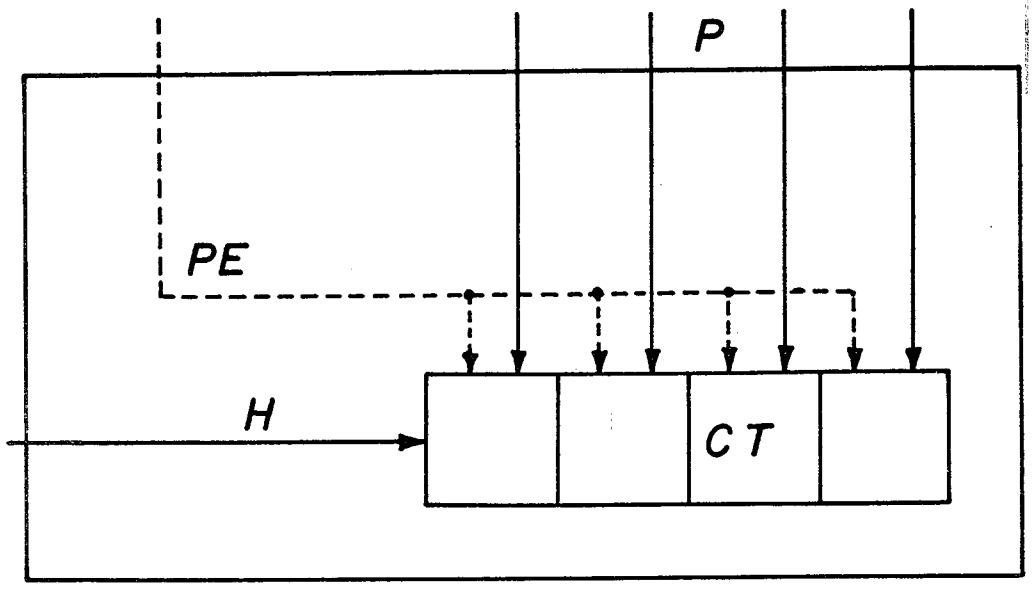
1) Chargement de registre :

L'opération se note R ← <expression1>. R doit être une variable déclarée registre au préalable et doit être compatible avec <expression1>.

Cette opération charge bit à bit dans l'élément de mémorisation R les bits correspondants de l'expression sous l'action d'une impulsion écrite entre crochets et en général sous l'action aussi d'une condition de chargement.

Exemple : Dans 'Unité' COMPTEUR de ZOOO1 on trouve :

< H > 'si' PE 'alors' (CT ← P) ;



2) Affectation d'état :

Elle se note comme la précédente E ← <expression 2> puisqu'il s'agit d'états, E doit avoir été déclarée comme telle et expression 2 doit être une expression d'état.

Exemple : Etat Z 1, Z 2 ; Impulsion H ;

<H> Z1 ← 'si' (Z2 = ETA1) 'alors' ETA2 'sinon' ETA3 ;

Ou l'on suppose que ETA1, ETA2, ETA3 sont trois étiquettes :

- Z2 = ETA1 est une expression d'état ayant une valeur de condition logique.
- 'si' <condition> 'alors' ETA2 'sinon' ETA3 est un opérateur condition portant sur des états, donc une expression d'état ayant valeur d'état.

3) Ordre 'Allera' :

Chaque unité considérée comme machine séquentielle possède une variable interne qui est une variable de type état implicite. Les étiquettes repèrent les valeurs de cette variable indépendamment de tout codage.
Les transitions de cette variable interne d'un état à un autre sont repérées par des ordres : 'allera' < expression d'état > . Ces ordres sont synchronisés par une impulsion comme les deux opérations précédentes.

α) 'Allera' < étiquette >

Exemple : < H > 'si' C 'alors' 'allera' E1 'sinon' 'allera' E2 sous le top d'impulsion H et suivant la valeur de la conditionC, les valeurs E1 et E2 sont attribuées à la variable interne. Cette opération équivaut au chargement d'un registre(variable interne) à partir d'une table de constantes (étiquettes). Quand l'étiquette mentionnée est la première qui apparaisse dans la suite de la description on peut écrire l'ordre précédent sous la forme condensée → .

β) 'Allera' <variable d'état>

Exemple : 'état' ZOZO ;

< H >'allera' 'si' C 'alors' E1 'sinon' ZOZO.

L'élément de mémorisation ZOZO peut stocker des valeurs d'état. Suivant la valeur de la condition C, la variable interne se voit alors attribuer soit la valeur constante repérée par E1 comme précédemment soit celle qui a été stockée au préalable dans ZOZO (qui se **trouve** donc être un paramètre variable).

γ) 'Allera' <Etat-Composé >.

Nous verrons au paragraphe suivant que l'on peut sur les états définir une structure de blocs, un état est alors repéré par une hiérarchie d'étiquettes suivant le nombre de blocs dans lesquels il se trouve imbriqué.

Exemple : <H> 'allera' ETA1 : HETA1 : E1 ;

Cet ordre équivaut à assigner non pas une mais trois variables internes, chacune propre à un niveau de bloc donné, l'état de l'unité dans laquelle on se trouve est en somme la concaténation de ces trois registres d'état.

δ) 'Allera' ∅ ;

Cet ordre ne spécifie pas la valeur de l'état suivant, on choisira donc au moment de la synthèse de l'automate de contrôle de l'unité,l'état qui simplifie le plus cette synthèse. Il se peut aussi que l'unité continue d'évoluer après avoir réalisé les fonctions qu'on en attend. On reprendra son contrôle soit en la faisant revenir à 'état initial' soit en forçant la variable interne de l'extérieur à un état donné.

ε) 'Allera' <état forcé >;

Exemple : 'Externe' ZOZO (,,,;,,,) ;

<H> 'allera' ETA1 : HETA1 : E1 'de' ZOZO ;

Cet ordre exceptionnellement ne spécifie pas un état de l'unité en cours, mais celui d'une unité qu'elle contient. Il a pour résultat de forcer dans la variable interne de cette unité contenue (ici ZOZO) l'état où l'on veut la placer (ici ETA1 : HETA1 : E1) Cet ordre implique la création pour l'unité forcée des entrées convenables pour cette opération, et pour l'im-

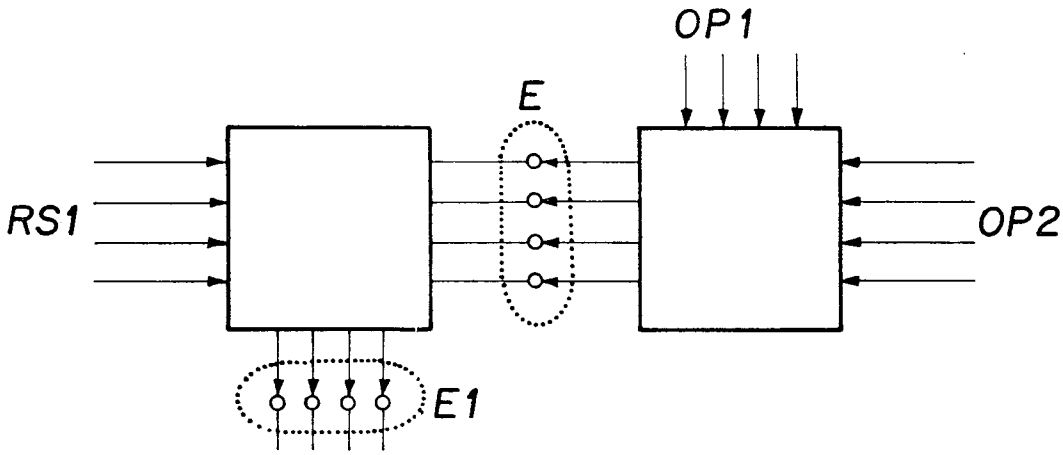
pulsion (ici H) qui la conditionne.

4) Connexion de signal ;

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. Reprenons un exemple tiré se Z0001 déjà vu.

Exemple : deux connexions E1 := E ≠ RS1 (1:4), E := OP1 ≠ OP2 ;

(en introduisant la fonction intermédiaire E)



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.

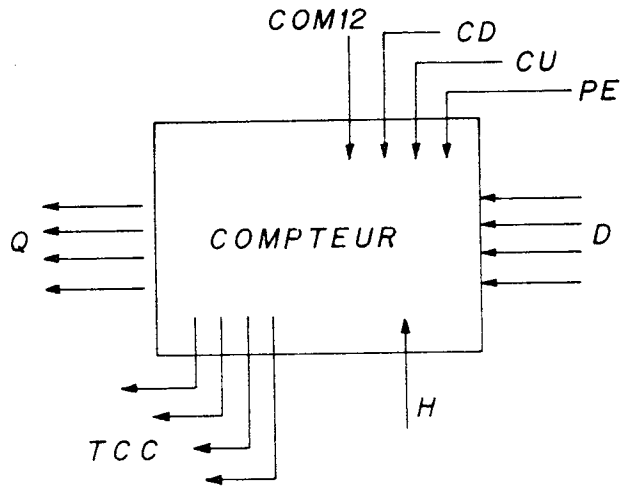
5) Connexion d'unité ;

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

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 : Dans l'unité LMEM de Z0001 on trouve la déclaration 'Externe' COMPTEUR ('horloge', 4,1,1,1,1;4,1) ; et la connexion : COMPTEUR (H,D,PE, CU,CD,CO M12; Q,TC)(1:4) :



α) 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 son entrée doit être affectée sous l'action d'une impulsion Or,

β) Une connexion d'unité, pas plus qu'une connexion de signal n'est jamais conditionnée par une impulsion.

γ) 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).

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

ε) 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

ι) 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. (voir à ce sujet les boucles 'pour' paragraphe 1)

χ) Les sorties dans une connexion d'unité sont toutes des variables généralisées de type signal.

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

μ) Signal-unité :

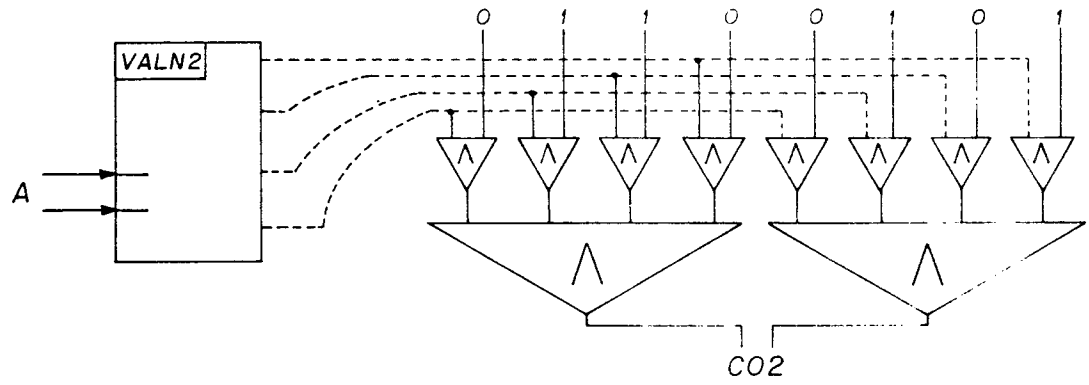
On appellera signal-unité une connexion d'unité dans laquelle l'une des entrées ou sorties est remplacée par *.

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 en 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 :=.

Exemple : Dans l'unité DEC de Z0001 page (B18) on trouve

'si' VALN2 (A; *) 'alors' (CO2 = 00) (CO2 = 11) (CO2 = 10)(CO2 =01);



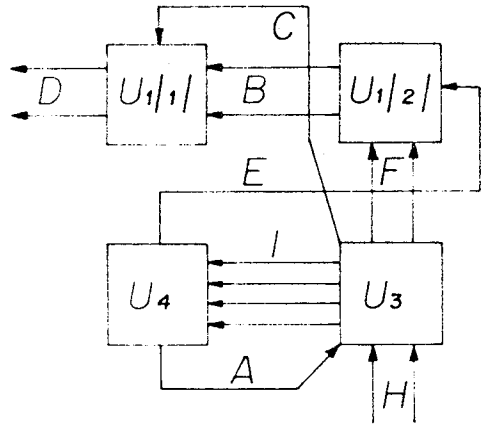
On économise la variable de type signal qui aurait servi à connecter les sorties de VALN2.

γ) Réseau de boites

La connexion d'unité,définit sur son contour, une permutation fixe des connexions d'entrée-sortie, bien que du point de vue fonctionnel cette permutation n'ait pas de sens. Mais les ordres de connexion d'unité font de CASSANDRE un langage de description de réseaux de boites, et il peut être utile d'utiliser plusieurs unités du même modèle (logique) mais avec une permutation différente de leurs entrées-sorties. On spécifie ces changements par rapport au modèle décrit dans la déclaration 'externe' en associant à la connexion d'unité une permutation de nombres entre parenthèses.

Exemple :

Externe : U1 |1| (2,1;2);
U1 |2| (2,1;2);
U3 (1,2;2,1,4)
U4 (4;1,1);



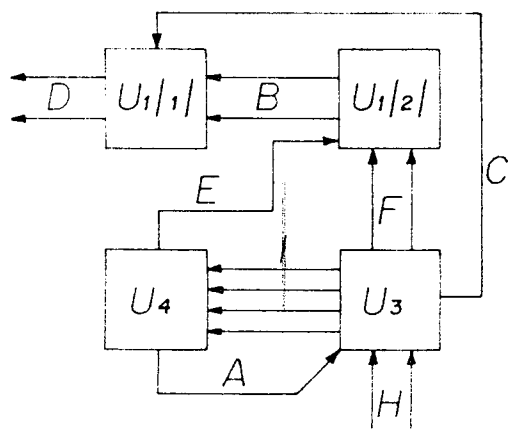
Signal : A,B (1:2) C,D (1:2),E
F(0:1), H(0:1) I(1:4);

Les connexions :

U1 |1| (B,C;D);
U1 |2| (F,E;B);
U3 (A,H;F,C,I);
U4 (I;E,A);

décrivent le réseau ci-contre (une réalisation possible).
Certaines propriétés topologiques (comme la planarité des connexions) peuvent dépendre de la permutation des entrées-sorties :

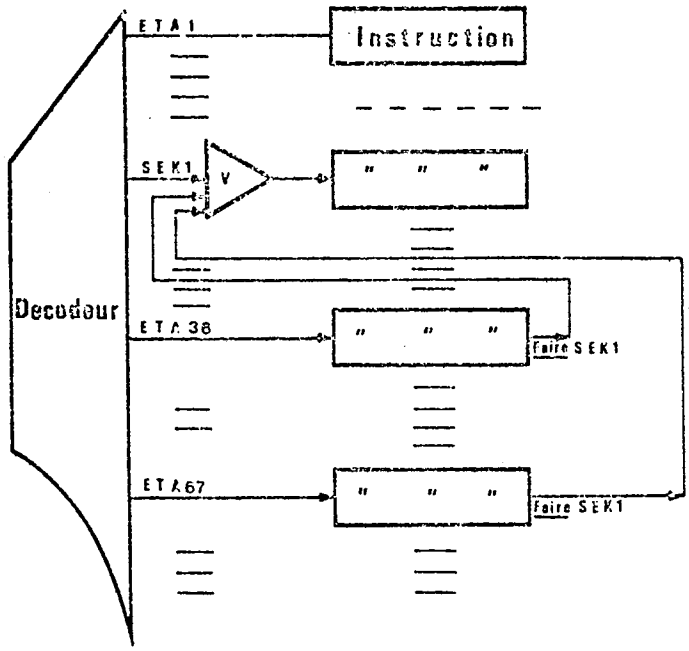
Exemple : U1 |1| (B,C;D);
U1 |2| (F,E;B)(2,1,3);
U3 (A,H;F,C,I)(1,2,4,4,5);
U4 (I;E,A)



décrira le nouveau réseau ci-contre identique en CASSANDRE mais cette fois planaire.

6) Ordre 'faire':

L'ordre 'faire' <expression d'état> s'applique comme allera à des variables ou expressions de type état, mais il n'est pas sous condition d'une impulsion car il ne change pas la valeur de la variable interne. Il crée simplement un signal qui rend exécutables simultanément un ensemble d'actions primitivement incompatibles et exécutables dans deux états différents de la machine. Pour comprendre son fonctionnement détaillé il faut avoir vu les instructions et séquences d'intructions CASSANDRE, nous le verrons donc au paragraphe suivant :



Nous pouvons supposer, sans faire de restriction, que les valeurs des états de l'unité sont des sorties d'un circuit général de décodage.

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.

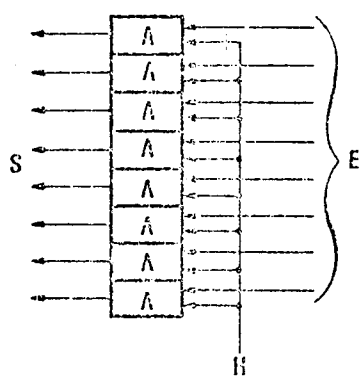
L'utilisateur doit évidemment prendre toutes précautions pour que toutes ces opérations en principe destinées à s'éxécuter par groupes séparés soient bien compatibles et puissent s'éxécuter simultanément.

7) Génération d'impulsions :

Il y a plusieurs façons de fabriquer de nouvelles impulsions

a) Connexions sous condition d'impulsion. Dans ce cas l'opération <H> S<= E, par exemple, aura pour effet de générer autant d'impulsions que 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 ;

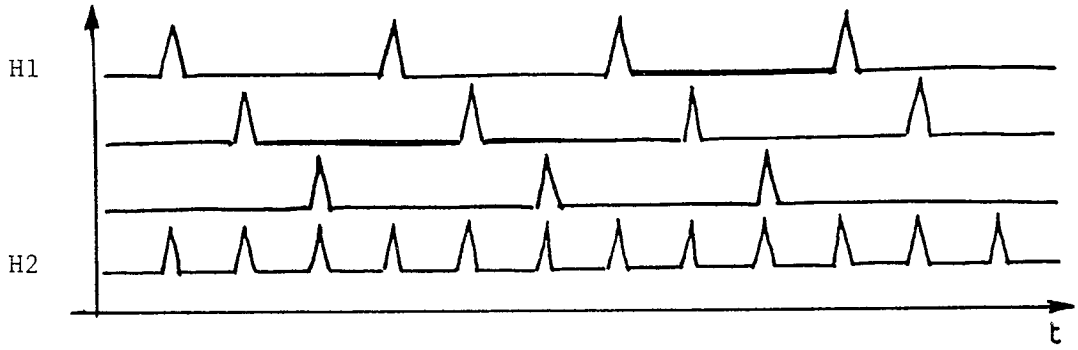


β) Retard d'une impulsion :

Nous avons vu dans la partie α) qu'une impulsion retardée est une autre impulsion de même fréquence.

Exemple : Impulsion H1, H2;

H2 ⇐(1÷3⊂ H1)V(2÷3H1)



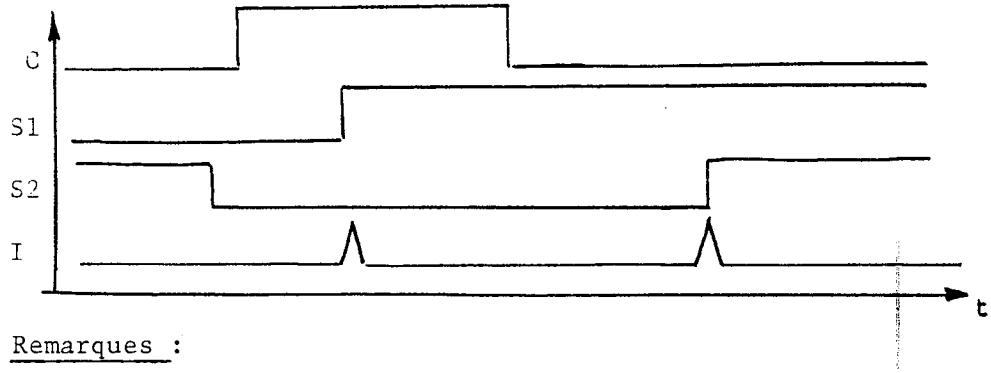
H2 a une fréquence triple de H1.

γ) Dérivation d'un signal :

Nous avons également vu cette opération en α)

Exemple : Signal S1, S2 ; Impulsion I ;

I⇐ Δ('si' C 'alors' S1 'sinon' S2) ;



8) Remarques :

α) Aucune des opérations que nous venons de voir ne peut s'appliquer deux fois à la même variable si ce n'est sous des conditions logiques disjointes (sauf l'ordre 'faire' pour lequel ce cas ne peut pas se produire).

Exemples : ‹H› R ⟵ E1, R⟵ E2; Chargement multiple.

mais, ‹H› R ⟵'si' C 'alors' E1 'sinon' E2; est correct

mais, ‹H› 'allera' ETA1,...; 'allera' ETA2; séquencement non déterministe.

‹H› 'allera' 'si' C 'alors' ETA1 'sinon' ETA2; est correct.

mais, ‹H› ETA⟵ HETA1 V HETA2; Expression d'état non permise.

‹H› ETA⟵ 'si' C 'alors' HETA1 'sinon' HETA2; est correct

mais, S := E1,...., S := E2; Connexion non définie.

'SI' C 'alors' (S := E1) 'sinon' (S := E2); est correct.

VALN2 (A;*); VALN2 (B;*) est faux car l'entrée de VALN2 reçoit à la fois A et B. Par contre : VALN2 (si C 'alors' A 'sinon' B; *);est correct.

β) L'opération ← est toujours conditionnée par une horloge, l'opération := jamais .

γ) Les modes de fonctionnement d'une unité sont définis par les règles : en mode synchrone l'opération ⇐ n'existe pas, en mode synchronisé cette opération est toujours conditionnée par une impulsion et crée des impulsions en phase, en mode asynchrone l'opération ⇐ peut être ou non conditionnée par une impulsion.

C) Instructions :

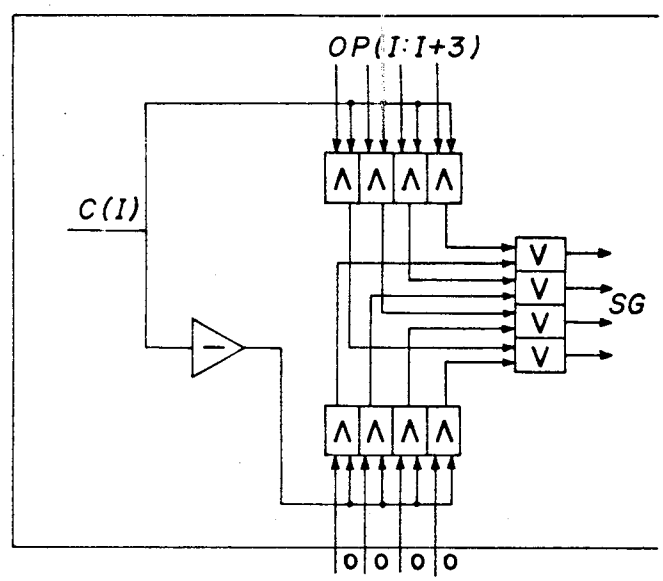
1) Opération conditionnelle :

Nous avons vu à propos des 'bus' que l'on pouvait distinguer entre variables "données" et variable de "contrôle". Plus généralement pour décrire le fonctionnement d'un organe logique, il est indispensable de pouvoir "contrôler" des actions élémentaires par une fonction de certains registres de cet organe, considérée comme "condition logique". Comme en CASSANDRE, les variables sont booléennes, chaque composante d'une variable pourra au besoin être considérée comme condition logique.

Exemple : dans l'unité DECG de Z0001 on trouve :

'si' C (I) 'alors' SG := OP (I:I+3) 'sinon' SG := 0000;

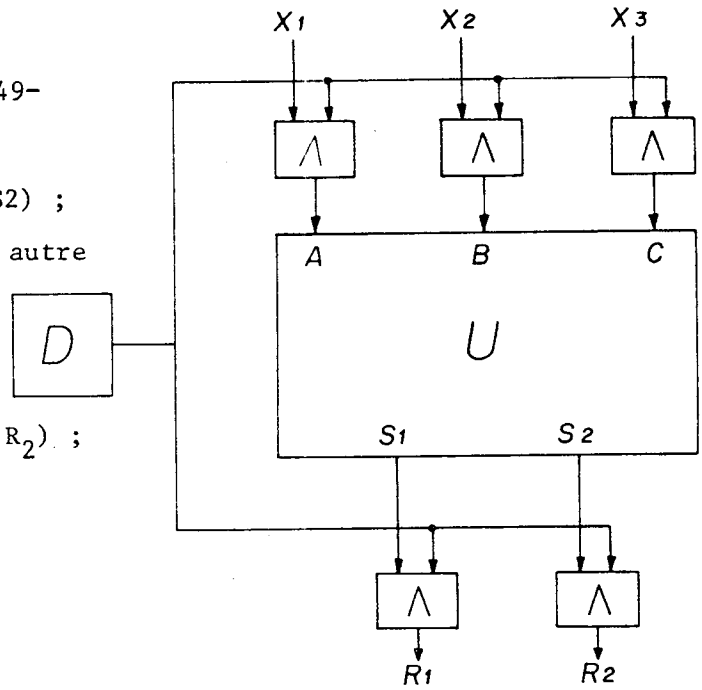
Réalisation de l'opération conditionnelle



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.

Exemple : Unité U(A,B,C ; S1, S2) ;
Cette unité peut être dans une autre utilisée sous la forme :



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

Il en est de même pour les variables 'signal-unité'. L'action de la même condition sur les entrées et les sorties peut être superflue si l'unité contient par exemple des circuits combinatoires (il suffit de conditionner les sorties). Il se peut aussi que chaque entrée-sortie soit validée sous des conditions différentes. Pour ces raisons il importe de ne faire apparaître dans la connexion d'unité sous condition, que les connexions utiles, c'est le cas dans l'exemple suivant.

Exemple : 'unité' Z0001 on trouve successivement :

```
LMEM [1] (M,I,S,AD12,AIG,SS,COMMANDE (12) ;,,TEST (6)); LMEM [2] (H,I,S,AD12,
AIG,SS,COMMANDE (12);,,TEST (7)); 'si' VALN3 [1] (,*) 'alors'
(E1 := SR2)(E1 :=LMEM [1] (;;;,;,*,)(E1 :=LMEM [2] (;;;;,,;,*,)(E1:=LMEM [1]
(;;;,;*,;))(E1 :=SR2)(E1 :=LMEM [2] (;;;;,;,*,))(E1 :=LMEM [1] (;;;,;,*,))
(E1 :=LMEM [2] (;;;;,;*,));
```

Pour les unités LMEM [1] et LMEM [2] les entrées sont inconditionnelles et deux des sorties (voir page A51) validées par une condition différente.

2) Instruction élémentaire :

Dans un système 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.

On appellera alors instruction élémentaire :

O_1, O_2, O_3, ..., O_n.

Mais nous avons vu qu'il y a deux catégories d'opérations élémentaires,celles de type A (affectation de registre, 'allera', affectation d'état), celles de type B (branchement de signal, connexion d'unité, ordre 'faire' . Les générations d'impulsions (⇐) peuvent appartenir soit à

l'une soit à l'autre.

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

Exemples : Dans l'unité CODDE de Z0001 :

On trouve : CINS (5) := -P; CINS (7) := P

C'est une instruction élémentaire de type B.

3) Instruction conditionnelle :

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

L'instruction conditionnelle s'obtient en faisant porter la condition non sur une opération élémentaire mais sur une instruction élémentaire.

Toujours dans l'unité CODDE on a :

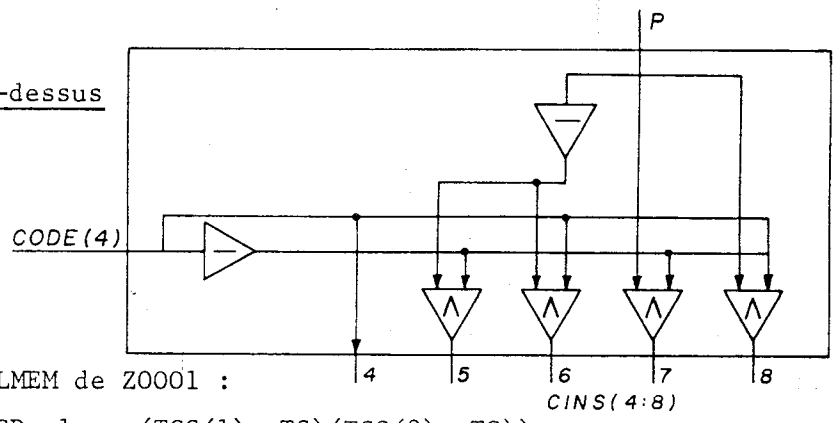
'si' CODE (4) 'alors' (CINS (4) := 1; CINS (6) := -P; CINS (8) := P) 'sinon' CINS (5) := -P; CINS (7) := P;

où nous retrouverons l'instruction élémentaire du paragraphe précédent. Mais l'instruction élémentaire CASSANDRE peut à son tour contenir des instructions conditionnelles dans sa liste. Ceci établit entre instruction conditionnelle et instruction élémentaire une récursivité croisée pouvant agir sur un nombre quelconque de niveaux d'imbrication. (voir exemple en [1]).

Ces définitions permettent donc de décrire la structure logique la plus complexe 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.

Schéma de l'exemple ci-dessus



CODE (4)

Autre exemple tiré de LMEM de Z0001 :

si AIG alors (si CU & CD alors (TCC(1):=TC)(TCC(2):=TC))

sinon (si CU & CD alors (TCC(2):=TC)(TCC(4):=TC));

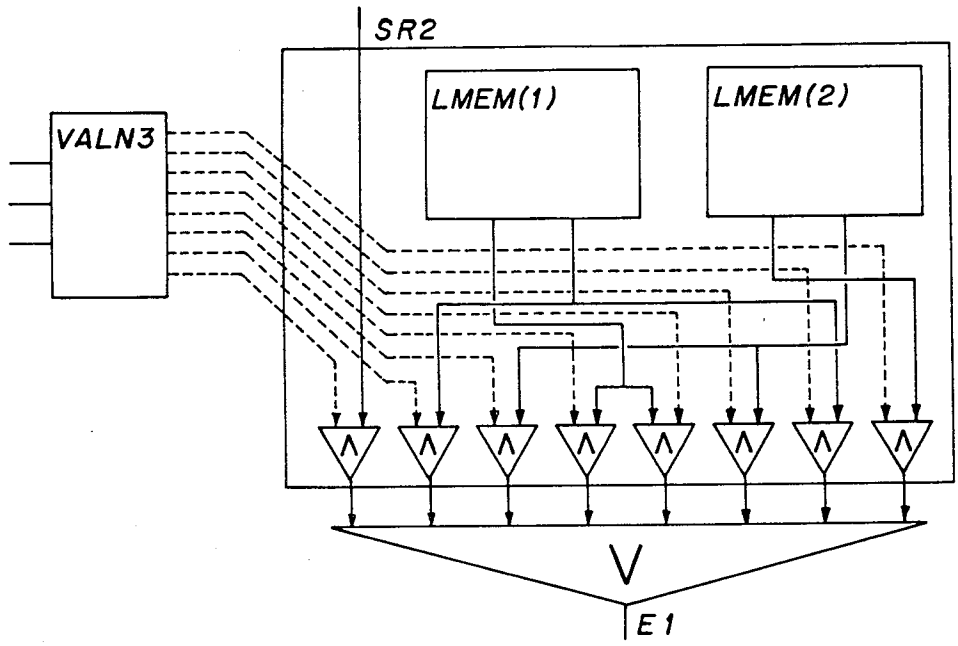
Le premier niveau conditionné par AIG en contient un second conditionné par CU & CD (voir paragraphe ci-dessous).

4) Généralisation de l'instruction conditionnelle :

Une condition logique est une variable booléenne scolaire; mais tout un ensemble de conditions logiques peut être regroupé en un tableau (c'est à dire une variable ou une expression en CASSANDRE). Chaque composante du tableau conditionnera une "instruction élémentaire".

Exemple : reprenons l'exemple précédent tiré de Z0001

'si' VALN3 [1] (;*) 'alors' (E1 := SR2)(E1 := LMEM [1] (,,,,;;*,))
(E1 := LMEM [2] (,,,,,;*,))(E1 := LMEM [1] (,,,,,;,*))(E1 := SR2)
(E1 := LMEM [2] (,,,,,;*,;))(E1 := LMEM [1] (,,,,,;*,))(E1 := LMEM [2](,,,,,;,*))



Cette instruction conditionnelle généralisée prend comme condition la sortie de l'unité VALN3 [1], les opérations sont des connexions sur E1 des sorties de LM [1] et LM[2] (2 exemplaires d'unités du même modèle), présentées en forme de signal-unité.

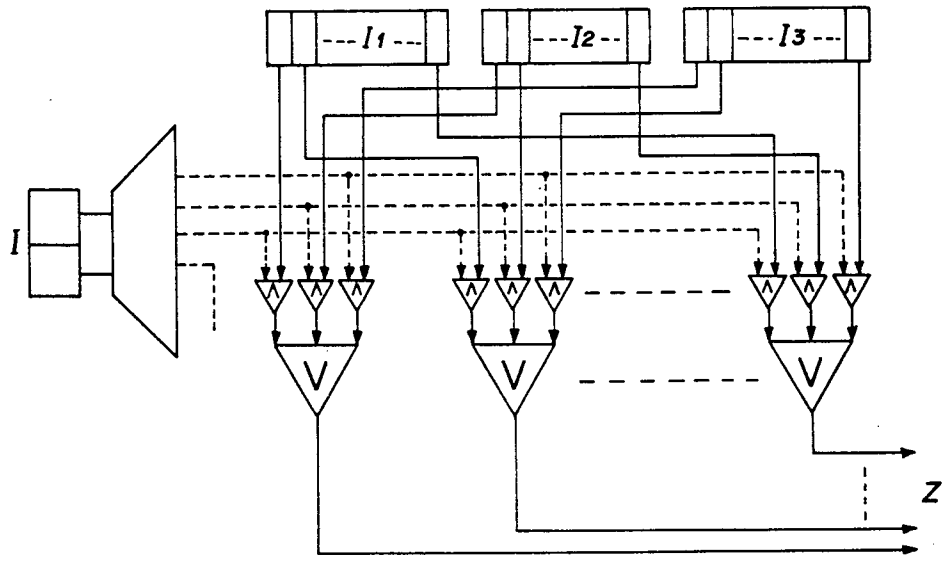
La condition peut être aussi une variable de type vecteur précédée de $.

Exemple : dans l'unité DECD de Z0001 on trouve :

'Si' S C01 'alors' (SG := OP1)(SG := 0&OP1 (1:3) (SG := 00&OP1 (1:2)
(SG := 000&P(1);

Autre exemple : Reprenons l'exemple de (1) page 90b

'Registre' I (1:2), I1 (1:15), I2 (1:15), I3 (1:15); signal Z (1:15);
'si' $ I 'alors' (Z := I1) (Z := I2) (Z := I3) ( ) ;



Il importe dans cette instruction conditionnelle généralisée que le tableau de condition ait autant de composantes qu'il y a d'instructions élémentaires entre parenthèses après le 'alors' ou le 'sinon'. Pour cela, si une composante de la condition ne commande aucune action il faut lui associer une paire de parenthèses vides.

d) Automate d'une unité :

L'unité CASSANDRE est aussi simple ou complexe que l'on veut. Ce peut être un simple fil, ce peut être un ensemble de circuits combinatoires, c'est à dire sans éléments de mémorisation. Dans ces cas, les sorties sont des fonctions booléennes des entrées, l'unité, en tant que système séquentiel n'a qu'un seul état et y reste. Dès qu'un élément de mémorisation (registre, ou état) apparaît, c'est un système séquentiel à plusieurs états. Suivant une démarche devenue classique (voir GERACE [6]), nous subdiviserons l'unité en automate de contrôle et automate opératoire.

En CASSANDRE cette distinction est particulièrement simple à obtenir en effet :

α) Tout élément de mémorisation déclaré registre fait partie de l'automate opératoire.

β) Tout élément de mémorisation de type état (ceux déclarés état et la variable interne dont les valeurs sont les étiquettes) fait partie de l'automate de contrôle.

Ceci posé, ces deux automates sont interconnectés d'une façon extrêmement précise :

γ) Toute condition portant sur un ordre 'faire', ordre 'allera' ou une affectation d'état est une entrée de l'automate de contrôle.(et une sortie de l'automate opératoire).

δ) Toute occurence dans la description d'une variable d'état ou d'une étiquette défini une sortie de l'automate de contrôle.(et une entrée de l'automate opératoire, qui a en plus toutes les entrées et sorties de l'unité considérée).

1) Etat :

Nous avons vu que la valeur d'un état d'une unité est repérée par une étiquette, suivie du symbole ":". Cette étiquette peut être multiple dans le cas d'un état composé.

Dans un état le système peut exécuter un nombre arbitraire d'opérations élémentaires et d'instructions telles que nous venons de les définir dans la partie C).

```
Exemple : 'Signal' E(1:4,1:4), S(1:4,1:4),
          FORCE, PLUSY , PLUSGY  , MOINSGY  ;
          'Impulsion' H; Registre SPES (0:4,0:4,0:4);
          Etat STATE (,1:4,1:4), ADRESSE;
< H >STATE (,I,J) ← 'si' FORCE  'alors' ADRESSE  'sinon'
         ('si' PLUSY 'alors' X |I+1|  : Y|J| 'sinon'
         ('si' MOINSY 'alors' X |I-1| : Y|J| 'sinon'
         ('si' PLUSGY 'alors' X |I| : Y|J+1| 'sinon'
         ('si' MOINSGY 'alors' X |I|: Y|J-1| 'sinon' X |I| : Y |I| ;
 X |J| : Y |J| : ECR: <H >SPES  (,I,J) ← E (,J), 'allera' STATE(,I,J) ;
                 LEC: S (,J) := SPES (,I,J); < H >'allera'  STATE (,I,J);
```

L'exemple montre une variable d'état STATE (,I,J) qui sert à stocker une expression d'état complexe afin que l'on puisse se contenter de l'écrire une seule fois.

L'exemple possède 2 états l'un contrôlé par une étiquette multiple X (I): Y (J): ECR, l'autre par une étiquette simple LEC.

Dans le premier on écrit E (,J) dans le registre SPES, dans l'autre on lit SPES dans S (,J).

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

> Dans un état, deux affectations ou connexions différentes de la même variable doivent s'effectuer sous des conditions disjointes, de même que 2 ordres allera.

2) Séquences :

Dans chaque état l'unité exécute entre autre le passage à l'état suivant grâce à des ordres 'allera'.

Reprenons l'exemple précédent en le développant à l'aide de boucles 'pour'.

Exemple : les déclarations sont les mêmes. Puis on pose :

X1 ≡ X5 ; X0 ≡ X4 ; Y1 ≡ Y5 ; Y0 ≡ Y4 ;

'Pour' I = 1 'a' 4 'début' 'Pour' J = 1 'a' 4 'début'

X |I| : Y |J| : ECR : <H> SPES (,I,J) ← E (,J) , 'allera' STATE (,I,J) ;

LEC : S (,J) := SPES (,I,J) ; <H> 'allera' STATE (,I,J) ;

'fin' ; 'fin';

Les deux boucles 'pour' imbriquées génerent une séquence de 32 états (que nous expliciterons au paragraphe suivant). Suivant l'état des conditions logiques on passe de l'état X |I| : Y |J| à l'état X |I+1| : Y |J| ou X |I| : Y |J+1| ou X |I-1| : Y |J| ou X |J| : Y |J-1| ou à l'état X |I| : Y |J| lui même.

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 contiennent. Nous pouvons désigner par des lettres grecques ces conditions cumulées. Il est immédiat de dessiner le graphe d'états de l'automate de contrôle d'une unité. Il a pour noeuds les valeurs des états (étiquettes) et pour branches les ordres allera avec leurs conditions associées.

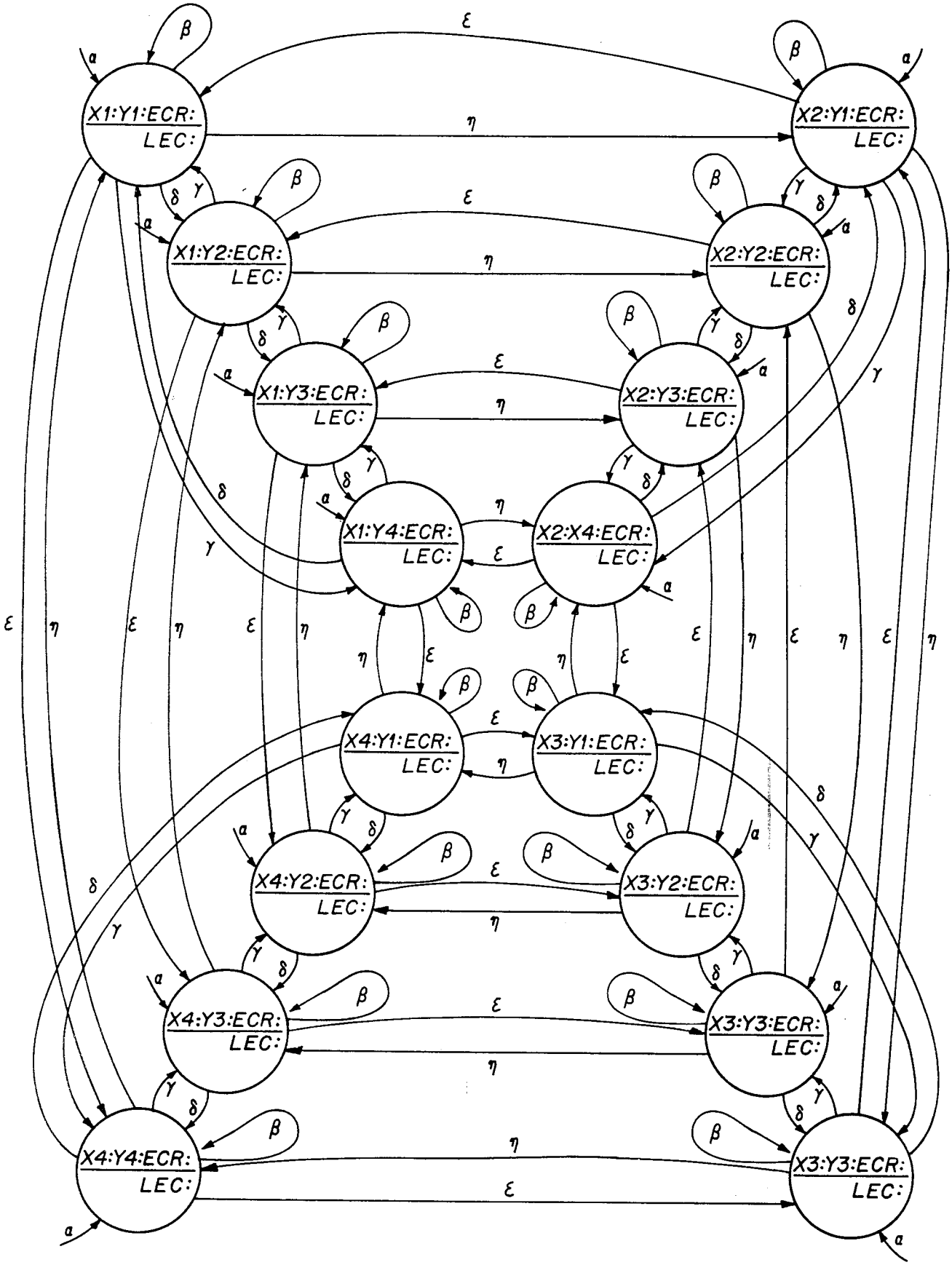
Dans l'exemple que nous avons pris, les conditions cumulées sont :

$\alpha \equiv$ FORCE forçage d'un état à partir de l'extérieur

$\beta \equiv$ - PLUSY ∧ - MOINSY ∧ - PLUSGY ∧ - MOINSGY ∧ -FORCE on reste dans le même état,

$\gamma \equiv$ - PLUSY ∧ - MOINSY ∧ - PLUSGY ∧ MOINSGY ∧ - FORCE, $\delta \equiv$ - PLUSY ∧ - MOINSY ∧ PLUSGY ∧ - FORCE

$\varepsilon \equiv$ - PLUSY ∧ MOINSY ∧ - FORCE, $\eta \equiv$ PLUSY ∧ - FORCE; le graphe de l'exemple est alors :


X1:Y1:ECR:
LEC:
X2:Y1:ECR:
LEC:
X1:Y2:ECR:
LEC:
X2:Y2:ECR:
LEC:
X1:Y3:ECR:
LEC:
X2:Y3:ECR:
LEC:
X1:Y4:ECR:
LEC:
X2:X4:ECR:
LEC:
X4:Y1:ECR:
LEC:
X3:Y1:ECR:
LEC:
X4:Y2:ECR:
LEC:
X3:Y2:ECR:
LEC:
X4:Y3:ECR:
LEC:
X3:Y3:ECR:
LEC:
X4:Y4:ECR:
LEC:
X3:Y3:ECR:
LEC:
α
β
γ
δ
ε
η

3) Sous-automate :

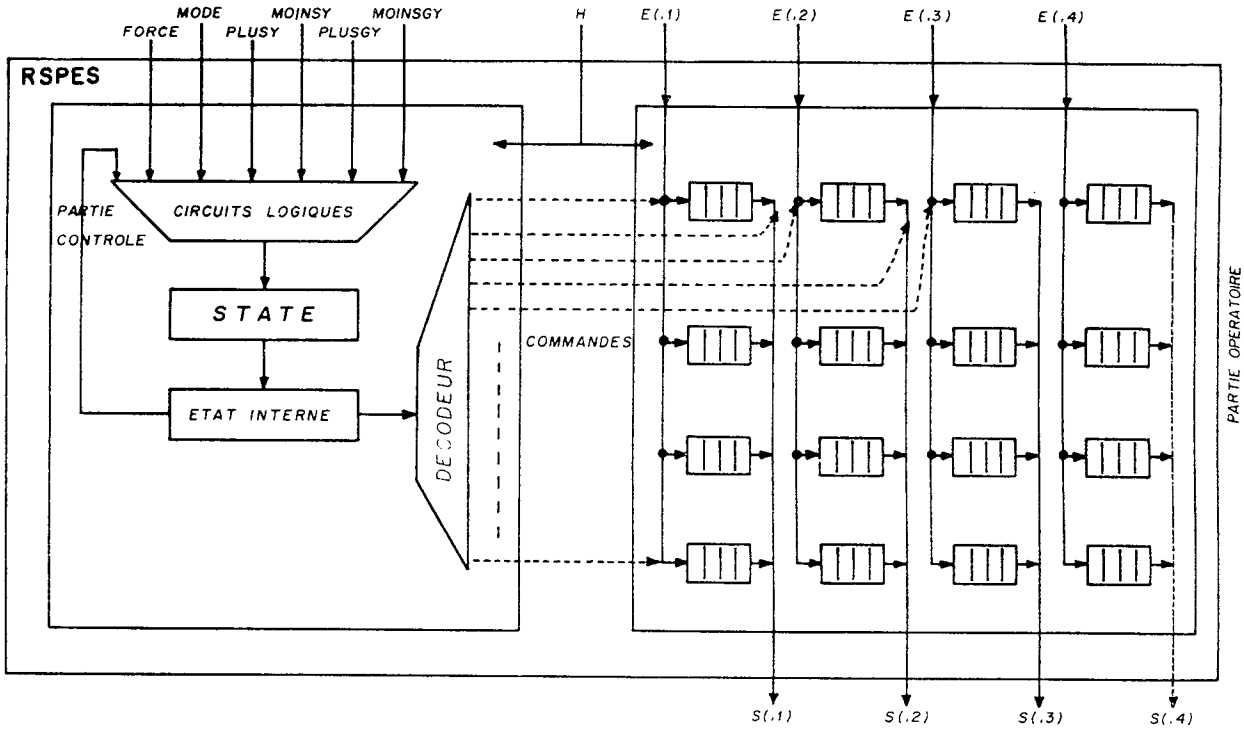
Nous avons vu que dans un état une étiquette contrôle une liste d'instructions. Dans le cas d'une étiquette composée elle contrôle une liste d'états, c'est à dire un automate. Développons l'exemple déjà vu précédemment :

```
Exemple : 'Unité' RSPES (H,MODE,FORCE,PLUSY,MOINSY,PLUSGY,MOINSGY,E(1:4,1:4);
          S(1:4,1:4) ) ;

          'Horloge' H; 'Registre' SPES(1:4,1:4,1:4) ;

          'Etat' STATE (,1:4,1:4), ETAT, ADRESSE ;
          X1 ≡ X5; XO ≡ X4; Y1 ≡ Y5; YO ≡ Y4;
'Pour' I = 1 'a' 4 'début' 'Pour' J = 1 'a' 4 'début'
<H> STATE (,I,J) ← 'si' FORCE 'alors'  ADRESSE 'sinon'
                   ('si' PLUSY 'alors' X |I+1| : Y |J| 'sinon'
                   ('si' MOINSY 'alors' X |I-1| : Y |J| 'sinon'
                   ('si' PLUSGY 'alors' X |I| : Y |J+1| 'sinon'
                   ('si' MOINSGY 'alors' X |I| : Y |J-1| 'sinon'
                   X |I| : Y |J|)))) : ETAT; 'fin'; 'fin';

<H> 'si' MODE 'alors' ETAT ← LEC 'sinon' ETAT ← ECR;

X1 : Y1 : ECR:<H>SPES(,1,1) ← E(,1), 'allera' STATE(,1,1);

          LEC: S(,1) := SPES(,1,1); <H> 'allera' STATE(,1,1);
     Y2 : ECR:<H>SPES(,1,2) ← E(,2), 'allera' STATE(,1,2);
          LEC: S(,2) := SPES(,1,2); <H> 'allera' STATE(,1,2);
     Y3 : ECR:<H>SPES(,1,3) ← E(,3), 'allera' STATE(,1,3);
          LEC: S(,3) := SPES(,1,3); <H> 'allera' STATE(,1,3);
     Y4 : ECR:<H>SPES(,1,4) ← E(,4), 'allera' STATE(,1,4);
          LEC: S(,4) := SPES(,1,4); <H> 'allera STATE(,1,4);

X2 : Y1 : ECR:<H>SPES(,2,1) ← E(,1), 'allera' STATE(,2,1);
          LEC: S(,1) := SPES(,2,1); <H> 'allera' STATE(,2,1);
     Y2 : ECR:<H>SPES(,2,2) ← E(,2), 'allera' STATE(,2,2);
          LEC: S(,2) := SPES(,2,2); <H> 'allera' STATE(,2,2);
     Y3 : ECR:<H>SPES(,2,3) ← E(,3), 'allera' STATE(,2,3);
          LEC: S(,3) := SPES(,2,3); <H> 'allera' STATE(,2,3);
     Y4 : ECR:<H>SPES(,2,4) ← E(,4)  'allera' STATE(,2,4);
          LEC: S(,4) := SPES(,2,4); <H> 'allera' STATE(,2,4);

X3 : Y1 : ECR:<H>SPES(,3,1) ← E(,1)  'allera' STATE(,3,1);
          LEC: S(,1) := SPES(,3,1); <H> 'allera' STATE(,3,1);
```

```
     Y2 : ECR:<H>SPES(,3,2) ← E(,2) 'allera' STATE(,3,2);
          LEC: S(,2) := SPES(,3,2);<H>'allera' STATE(,3,2);
     Y3 : ECR:<H>SPES(,3,3) ← E(,3) 'allera' STATE(,3,3);
          LEC: S(,3) := SPES(,3,3);<H>'allera' STATE(,3,3);
     Y4 : ECR:<H>SPES(,3,4) ← E(,4) 'allera' STATE(,3,4);
          LEC: S(,4) := SPES(,3,4);<H>'allera' STATE(,3,4);

X4 : Y1 : ECR:<H>SPES(,4,1) ← E(,1) 'allera' STATE(,4,1);
          LEC: S(,1) := SPES(,4,1; <H>'allera' STATE(,4,1);
     Y2 : ECR:<H>SPES(,4,2) ← E(,2) 'allera' STATE(,4,2);
          LEC: S(,2) := SPES(,4,2);<H>'allera' STATE(,4,2);
     Y3 : ECR:<H>SPES(,4,3) ← E(,3) 'allera' STATE(,4,3);
          LEC: S(,3) := SPES(,4,3);<H>'allera' STATE(,4,3);
     Y4 : ECR:<H>SPES(,4,4) ← E(,4) 'allera' STATE(,4,4);
          LEC: S(,4) := SPES(,4,4);<H>'allera' STATE(,4,4);
```

Cette unité a des étiquettes composées qui définissent non pas un automate de contrôle mais une hiérarchie d'automates. Le premier à 4 états X1,X2,X3,X4, dont chacun valide un autre automate ayant 4 états Y1,Y2,Y3,Y4, dont chacun à son tour valide un autre automate à 2 états ECR, LEC. On peut ainsi définir une structure quelconque d'automates qui évoluent indépendamment. Les transitions entre états se font par des ordres allera <Etat composé>(il est clair ici que STATE ne contient que des états composés à 3 étiquettes X(1) : Y(J) :{LEC/ECR. On peut forcer l'état STATE depuis l'extérieur, on peut aussi en fonction des diverses commandes MODE, FORCE... faire évoluer l'état interne de RSPES grâce aux ordres allera.

RSPES représente un ensemble de 16 registres de 4 bits accessibles en entrée par E(,1) E(,2) E(,3) E(,4) et en sortie par S(,1) S(,2) S(,3) S(,4). L'état X permet de les adresser par paquets de 4, puis l'état Y de sélecter le bon enfin le 3e état indique si c'est en lecture ou en écriture.
De plus, l'adresse X peut augmenter de 1 (signal PLUSY, on saute alors 4 registres) ou diminuer de 1, de façon cyclique puisque Xo≡X4 et X1≡X5. De même pour l'adresse Y quand X ne varie pas.



Comme nous l'avons déjà dit, l'unité CASSANDRE peut se décomposer en partie contrôle et partie opératoire. Ici la partie opératoire ne compte que le registre SPES et ses connexions en entrée et en sortie. La partie contrôle contient le registre d'état STATE (qui double le registre d'état interne, qui existe dans toute unité, puisque tous les ordres 'allera'sont suivis de STATE: Ils forment alors tous deux une sorte de registre maître-esclave), les circuits logiques qui déterminent le contenu de STATE à partir des entrées et de l'état interne et le décodeur qui fabrique toutes les commandes pour la partie opératoire à partir de l'état interne.

4) Ordre 'faire':

Nous avons vu, dans la définition des opérations élémentaires, que l'ordre faire avait pour effet de faire exécuter simultanément deux états distincts,celui dans lequel se trouve l'ordre faire et celui désigné par l'ordre faire (voir par b, 6). Nous allons voir 3 emplois de cet ordre.

α) Macro-substitution :

Faire s'applique à un état MICRO qui ne contient pas d'ordre allera.

Exemple : MICRO :<H> I_1, I_2, ... I_p ;

ETA 36 :<H> J_1, J_2, ... Faire MICRO, ..., J_c;

ETA 68 :<H> K_1, K_2, ... Faire MICRO, ... , K_p;

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 :

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_p;

Dans la réalisation les circuits relatifs aux actions I_1, I_2... I_p ne sont réalisés qu'une fois au lieu de plusieurs, il y a économie de matériel comme d'écriture. Le passage en séquence s'opère à l'aide des allera contenus dans ETA 36 et ETA 68, puisqu'il n'y en a pas parmi I_1, I_2, ... I_p.

β) Sous-microprogramme :

Si l'ordre faire désigne un état qui contient au moins un ordre allera cet état est donc le premier d'une séquence, cela revient à effectuer un branchement à un même sous-microprogramme à partir de plusieurs états.

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

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

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

Dans ETA 36 et ETA 67 on valide l'état SEK_1 et la suite d'états qui se succèdent à partir de là. Si Z est une variable d'état, on peut s'en servir pour y stocker une "adresse de retour". Si on remplace faire SEK_1 par exemple par :

'si' C 'alors' (<H> Z ← ETA 48; faire SEK_1) 'sinon' (H allera ETA 47);

On voit que l'on peut se dérouter à partir d'un point quelconque et sous certaines conditions, puis revenir au point que l'on désire si l'on a prévu ce retour :

ETA 36 'si' C SEK 1
SEK 2

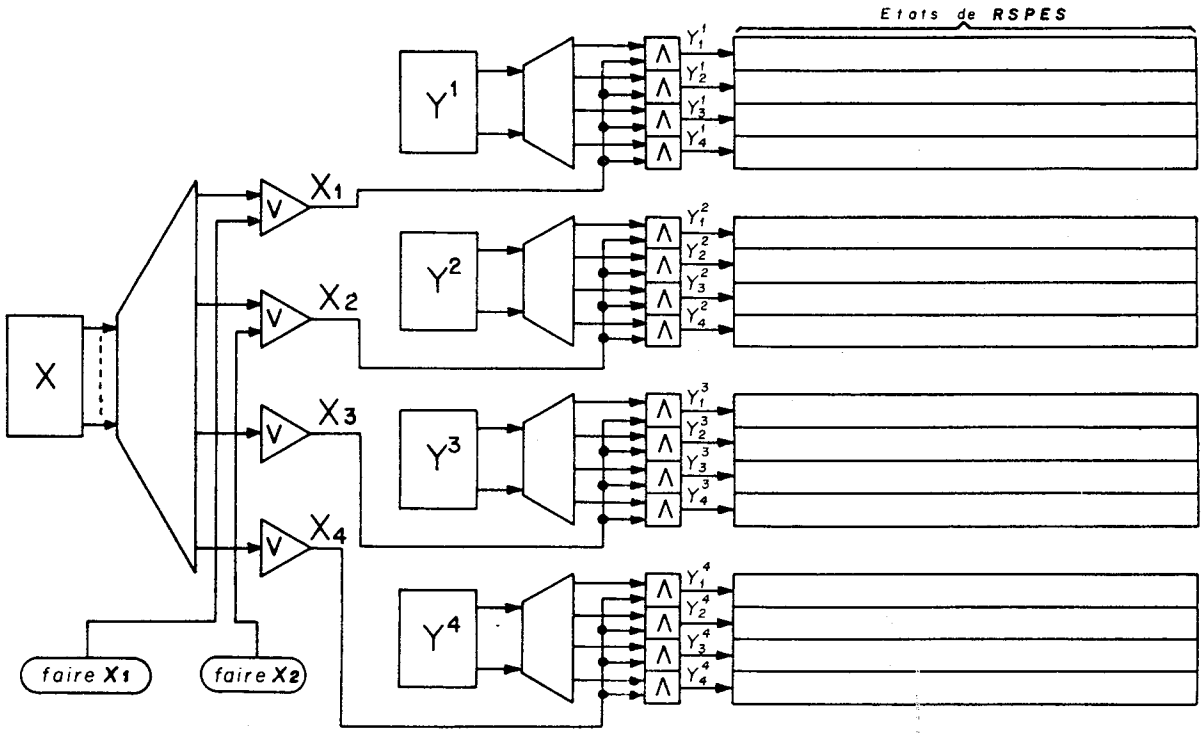
'si' -C

ETA 47 SEK 1, , allera Z
ETA 48

On étend ainsi au hardware la notion de sous-programme.

γ) 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. Reprenons l'exemple de l'unité RSPES vue précédemment. L'automate principal de la partie contrôle à 4 états X1, X2, X3, X4. Chacun de ces états contrôle un sous-automate à 4 états Y1, Y2, Y3, Y4. Supposons tout d'abord que, bien qu'ayant les mêmes noms d'états les sous-automates controlés par X1 et par X2 soient différents :



Dans ce cas les états Y1, Y2, Y3, Y4 sont codés avec des variables différentes suivant le sous-automate auquel elles appartiennent (ex : Y_1^1, Y_1^2, Y_1^3, Y_1^4, Y_2^1, Y_2^2,....).

Supposons le 1er sous-automate dans l'état Y_2^1 et le second dans l'état Y_1^2 et supposons encore que l'on rencontre faire X1, faire X2. D'après la définition de faire tout se passe comme si l'automate principal était à la fois dans les états X1 et X2. Les deux sous-automates sont donc validés et le premier va exécuter (supposons les dans le mode LEC) :

(1) S(,2) := SPES(,1,2); <H> 'allera' X1 : Y_3^1 : LEC;

la second va faire (2) S(,1) := SPES(,2,1); <H> 'allera' X2 : Y_5^2 : LEC;

Considérons les ordres allera . Dans le premier tout se passe comme si X était dans l'état X1 et ETA dans l'état LEC, l'ordre 'allera' X1:Y_3^1: LEC va donc se réduire à un ordre 'allera' Y_3^1 appliqué à la variable interne Y^1. Dans le second tout se passe comme si X était dans l'état X2 et ETA encore dans LEC, l'ordre 'allera' se réduit à 'allera' Y_2^2 appliqué à Y^2.

Donc les deux instructions (1) et (2) sont compatibles et exécutables simultanément. Al'instant suivant le premier sous-automate va devoir exécuter :

S(,3) := SPES(,1,3); <H> 'allera' X1: Y_4^1: LEC;

et le second : S(,2) := SPES(,2,2); <H> 'allera' X2: Y_3^2: LEC;

Ces deux instructions sont compatibles et vont s'exécuter si les ordres faire X1 et faire X2 sont toujours valides.

On peut ainsi lire en parallèle deux mots de 4 bits dans le registre SPES et les avoir sur des sorties S différentes.

Cet exemple montre aussi la possibilité par les ordres faire de valider plusieurs sous-automates simultanément et de leur faire exécuter des séquences d'instructions en parallèle. Les possibilités de fonctionnement ouvertes ainsi par l'utilisation des ordres 'allera' et faire sont innombrables.

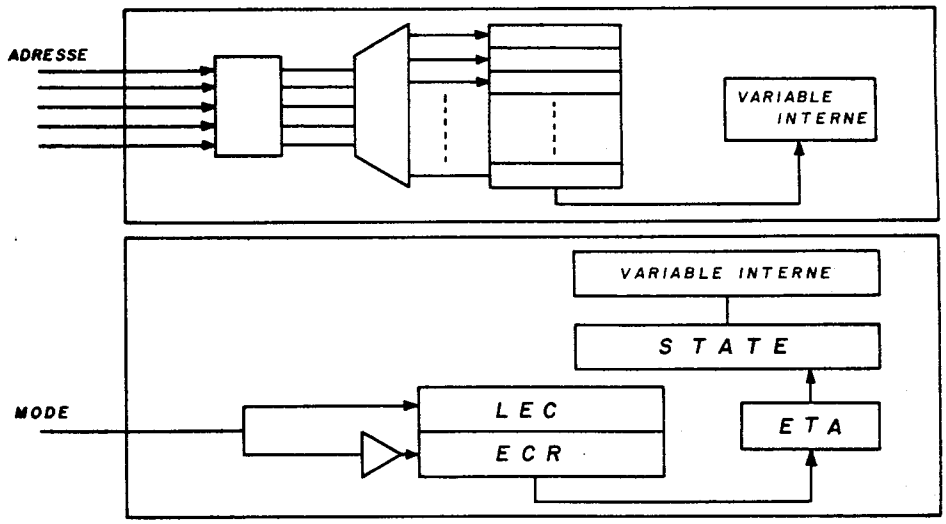
δ) Forcage d'un état dans une unité :

Nous avons vu que les variables employées dans une unité n'avaient de signification qu'à l'intérieur de celle-ci et que seules des "bornes" d'entrée-sortie étaient accessibles de l'extérieur. Si l'on veut de l'extérieur forcer la variable interne à une certaine valeur,deux moyens différents le permettent en CASSANDRE.

On peut premièrement prévoir tous les états de l'unité qui seront forcés de l'extérieur. On les range dans l'unité, à l'intérieur d'un ensemble de variables d'état qui constituent alors une sorte de mémoire morte. L'adresse d'un état dans cette mémoire sera le signal nécessaire et suffisant pour l'extraire et le charger dans le registre de la variable interne. On respecte ainsi la définition de l'unité déjà donnée et de plus on diminue le nombre des entrées de celle-ci.

Exemple : Reprenons l'unité RSPES; on y trouve une variable ETAT susceptible de prendre les 2 valeurs d'état LEC et ECR (peut être chacune codées sur de nombreux bits). Un seul fil MODE suffit pour déterminer de l'extérieur laquelle des 2 valeurs sera stockée dans ETAT :

<H> 'si' MODE 'alors' ETAT ← LEC 'sinon' ETAT ← ECR ;



Mais on peut désirer plus de souplesse et dans ce cas il faut connaître à l'extérieur la valeur des états intéressants de l'unité (c'est à dire l'étiquette ou le nom de la variable d'état) on les met donc dans la déclaration d'externe.

Exemple : Soit l'unité U contenant RSPES , 'Unité' U (liste d'entrées sorties); 'Externe' RSPES (ADRESSE ,X1, X2, X3, X4, Y1, Y2, Y3, Y4) ('horloge', 1,1,1, 1,1,1,(4,4); (4,4));

Impulsion H; Registre R 1(1:2), R2(1:2), R3, PARALEL;

(1)<H> 'si' FORCE 'alors' ADRESSE ← 'si' # R1 'alors' (X1)(X2)(X3)(X4) :
'si' # R2 'alors' (Y1)(Y2)(Y3)(Y4) ;

(2)<H> 'si' /∧ R1 &R2 &R3 'alors' 'allera' X1:Y1 'de' RSPES;

(3) 'si' PARALEL 'alors' (faire Y1 de RSPES;faire Y2 de RSPES; faire Y3 de RSPES; faire Y4 de RSPES);

La première instruction établit que R1 et R2 sont les registres d'adresse (contenus dans U) qui permettent de sélecter l'un des 16 mots de 4 bits de SPES (contenus dans RSPES). Suivant son contenu en effet, c'est l'un des 16 états de RSPES qui est stocké dans ADRESSE dont le signal FORCE cause le transfert dans STATE.

La deuxième instruction exprime que l'on force l'état X1:Y1 dans la variable interne de RSPES dès que R1 et R2 valent chacun 3 si R3 vaut 1.

On obtiendrait le même résultat en ajoutant un signal d'entrée de RSPES, RAZ et la spécification Etat initial X1:Y1(RAZ); On connecterait l'entrée RAZ dans U avec la sortie de /\ R1 & R2 & R3.

La troisième instruction montre que sous le signal PARALEL et dans un état Xi quelconque de RSPES on exécute simultanément les 4 états Y1, Y2, Y3, Y4 du sous-automate c'est à dire qu'on lit ou écrit en parallèle un mot de 16 bits de SPES. C'est possible car cela utilise 16 entrées ou sorties différentes de RSPES. Cette unité est donc une mémoire ou l'on peut faire varier de 4, 8, 12 à 16 bits la longueur des mots. Ceci se fait simplement grâce à des ordres faire.

5) Description d'automates :

Les automates que nous venons de voir sont ceux de la partie contrôle d'une unité. On peut bien sur décrire en CASSANDRE un automate séquentiel fini tel qu'on se le donne dans la théorie des machines séquentielles.

α) Automate définit par son graphe d'états et ses entrées-sorties.

On a une liste d'états H1; H2,...Hn, une liste de transitions entre ces états sous certaines conditions α_1, α_2...., α_p, une liste d'entrée E1, E2, ...Eq, une liste de sorties S1, S2, ...Sr et un ensemble de fonctions booléennes F1, F2, ..., Fs des entrées et des états. L'automate est une unité CASSANDRE :

'Unité' AUTOMATE (E1, E2,..., Eq; S1, S2, ...Sr);

'Signal' F1, F2, ...Fs;

H1: S1 := F1, S2 := F2,..., si α_1 alors allera H 1;

H2:

Hn

β) Automate décrit par des expressions régulières :

Une expression régulière est constituée de variables booléennes reliées par les opérateurs &(concaténation) V(Union) et * (itération). Les expressions régulières permettent de décrire tout langage reconnaissable par un automate d'états finis. Mais ces expressions sont quelquefois difficiles à manipuler c'est pourquoi, le cahier des charges qui défini un automate séquentiel utilise un langage proche du langage parlé faisant appel à des locutions comme : "si", "lorsque", "dès que", "tant que", "jusqu'à ce que", "chaque fois"...

Payan a défini formellement un opérateur qu'il a appelé "dès que" (noté en CASSANDRE 'dq').

Il a montré (6) que les locutions précédemment citées peuvent se traduire directement à l'aide des opérateurs & , V, * et dq, qui ont été mis dans CASSANDRE.

En général le graphe de l'automate est partiellement connu, l'évolution entre les états connus est décrite par des expressions formées à l'aide des 4 opérateurs précédents. Payan a défini l'algorithme de passage de cette forme, au graphe complètement défini qui nous ramène au cas ℰ1)

Exemple : Donné par PAYAN dans (6)

'Unité' SERRURE (X,Y; S(1:2));
Impulsion : H;
OUVERTE : 'si' XVY 'alors' (<H> 'allera' FERMEE; S(1):=1);
FERMEE : 'dq' X & Y & X(<H> 'allera' OUVERTE; S(2):=1);

Après traitement on obtiendrait :

'Unité' SERRURE (X,Y; S(1:2);
'Impulsion' H;
OUVERTE : 'si' XVY 'alors' (<H> 'allera' FERMEE 1; S(1):=1);
FERMEE 1 :<H> 'si' X 'alors' 'allera' FERMEE 2 'sinon'('si' Y 'alors' 'allera' FERMEE 1);
FERMEE 2 :<H> 'si' Y 'alors' 'allera' FERMEE 3 'sinon'('si' X 'alors' 'allera' FERMEE 2);
FERMEE 3 : 'si' X 'alors' (<H> 'allera' OUVERTE; S(2) := 1) 'sinon' ('si' Y 'alors' 'allera' FERMEE 1);

Ces notions sont une base suffisante pour une extension de CASSANDRE en direction d'un langage de rédaction de cahier des charges de systèmes logiques, sous-langage d'une langue parlée.

BIBLIOGRAPHIE

1 - J. MERMET "Définition du Langage CASSANDRE"
Thèse de Docteur Ingénieur Université de GRENOBLE, Mars 1970.

2 - F. ANCEAU, COUTURIER, J. DOUSSY, F. PERRON
"Compilation et Simulation du Langage CASSANDRE".
Chapitre II, rapport final du contrat C.R.I. "Industrialisation de CASSANDRE".

3 - P. LIDDELL,"Partition Syntaxique de Systèmes Logiques décrits en CASSANDRE"
Thèse de 3e cycle, Université de GRENOBLE, Mars 1970.

4 - F. LUSTMAN, J. MERMET "CASSANDRE, Un Langage de Description de Machines Digitales"
Revue bleue de l'A.F.I.R.O. n° 15 1969.

5 - G.B. GERACE, G. GESTRI "A Method for designing a digital system as a sequential network system" University de Pise, juin 1967.

6 - C. PAYAN "Langage de description de systèmes séquentiels"
Chapitre V, rapport final du contrat C.R.I. "Industrialisation de CASSANDRE".

B - DESCRIPTION DU CALCULATEUR Z0001

I) Eléments du Z0001

a) Avertissement :

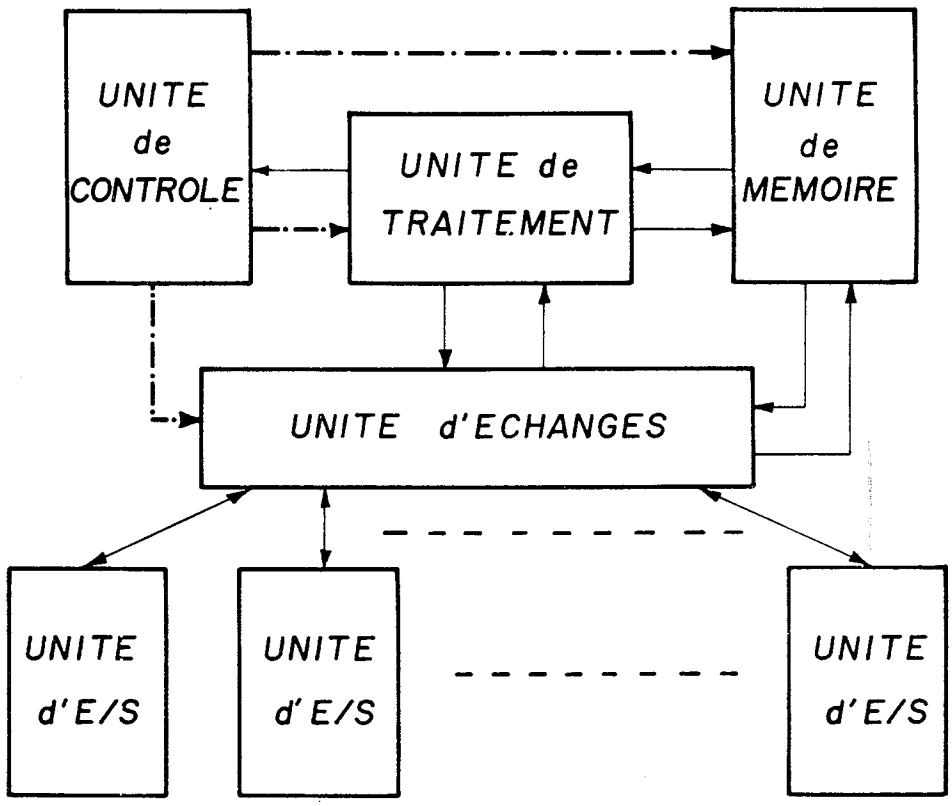
Ce calculateur a été conçu en collaboration avec C. MALKA, comme contre-projet, sans objectif de réalisation, du calculateur T1000 TELEMECANIQUE. Le but poursuivi était d'abord l'utilisation dans une architecture de minicalculateur des circuits M.S.I., nouveaux en 1970, pour essayer d'arriver à un optimum du moment quant au compromis coût-performances, ensuite la familiarisation d'une équipe avec la micro-programmation telle qu'elle est pratiquée dans les machines dites "à mots longs", avec les remises en cause que cela peut entraîner. Le Z1000 ainsi conçu a été détourné, après avoir rempli sa fonction, vers un usage pédagogique. C'est ainsi que sous le nom de Z0001 il a servi de support à l'examen de microprogrammation en C4 de maîtrise d'informatique, de sujet de travaux dirigés de CASSANDRE en 3e année de l'E.N.S.I.M.A.G. et qu'il a fourni à P. DARONDEAU un "projet d'étude d'un minicalculateur microprogrammé entièrement compatible avec le 360, réalisable avec des moyens minimaux"(1). Des élèves-ingénieurs de l'Ecole d'Electronique ont mis au point pour lui, une mémoire de contrôle de 256 X 32 bits inscriptible à partir d'un ruban perforé (2). Des élèves de l'ENSIMAG ont étudié autour d'un système de plusieurs modules Z0001 "une hiérarchie de mémoire utilisant une technologie à domaines magnétiques" (3).

Il n'est pas improbable que ce calculateur fasse l'objet d'une réalisation prototype dans les laboratoires de l'ENSIMAG au titre des travaux pratiques de "physique des machines", vu le faible matériel électronique qu'il met en jeu.

L'utilisation que nous ferons de ce modèle du Z0001 est tout d'abord une illustration des concepts du langage CASSANDRE (la partie A y puise la plupart de ses exemples). Il servira ensuite de "cobaye" aux traitements décrits dans les parties C et D.

b) Remarques sur l'organisation générale du Z0001

Comme la plupart des ordinateurs vu d'un angle très macroscopique, le Z0001 répond au schéma ci-dessous :



L'unité de mémoire contient la majeure partie des éléments de mémorisation rapidement accessibles dans le cours du fonctionnement du calculateur.

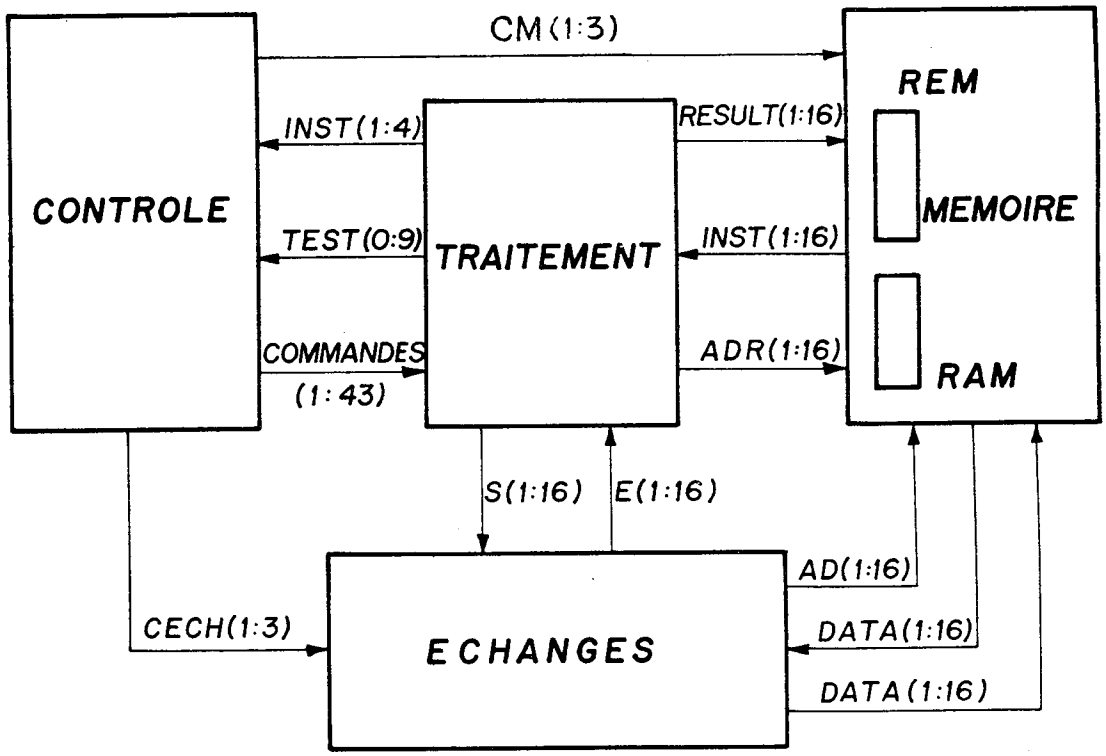
Cette mémoire est destinée à stocker des programmes qui sont envoyés dans l'unité de traitement instruction par instruction et d'autres informations qui sont appellées données lorsqu'elles sont transférées vers l'unité de traitement et résultats en sens inverse. Elle peut aussi envoyer des informations à l'Unité d'échanges, elles seront alors considérées comme données, elle peut en recevoir des données ou des adresses.

Tous ces transferts sont controlés par des signaux de l'unité contrôle. L'unité d'échanges va canaliser les flux d'information en provenance des unités périphériques, (qui transitent par des unités d'entrées-sorties) vers l'unité de traitement ou l'unité de mémoire et en sens contraire.

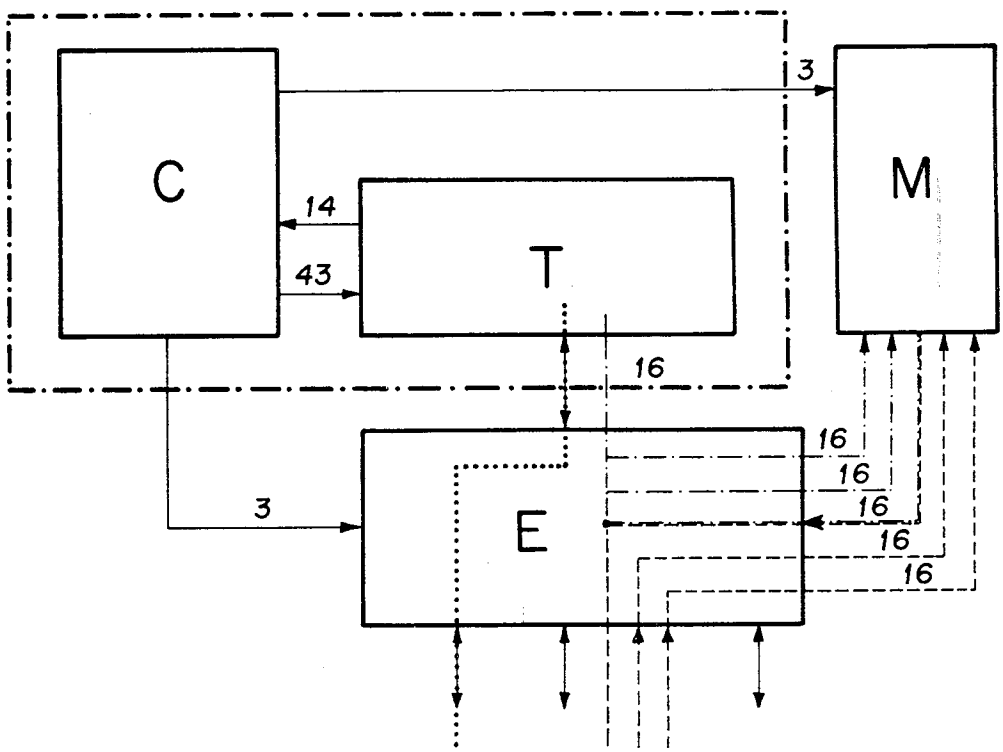
Nous ne décrirons ici, que les signaux d'interface entre ces trois unités d'entrées-sorties, dont les réalisations ne prêtent guère à variations. Le lecteur qui voudrait un complément d'information de ce côté pourrait utilement consulter la brochure "Calculateur T2000, systèmes d'entrées-sorties" dont nous avons essayé de respecter les spécifications.

L'unité de contrôle envoie aussi, ses signaux, à l'unité de traitement conditionnant à l'intérieur de celle-ci les opérations (c'est-à-dire des transferts élémentaires d'information) d'une façon que nous expliquerons dans le détail. Elle reçoit de celle-ci des signaux d'état qui lui servent pour faire des tests logiques,et des codes instructions (en provenance de l'unité de mémoire) qu'elle interprète par ses microprogrammes.

Il est instructif, dès à présent, d'avoir une idée approchée du nombre d'interconnexions entre les unités. Nous supposerons la mémoire organisée en mots de 16 bits transférés en parallèle.



L'Unité Traitement a cinq liaisons de 16 bits, mais on peut les réduire à une seule bidirectionnelle, en diminuant légèrement le parallélisme et en passant par l'unité d'échanges.



Même après réduction maximum, le bloc de traitement a encore environ 75 liaisons avec l'extérieur, par contre un bloc contenant à la fois les unités de contrôle et de traitement pourrait en avoir moins de 30.

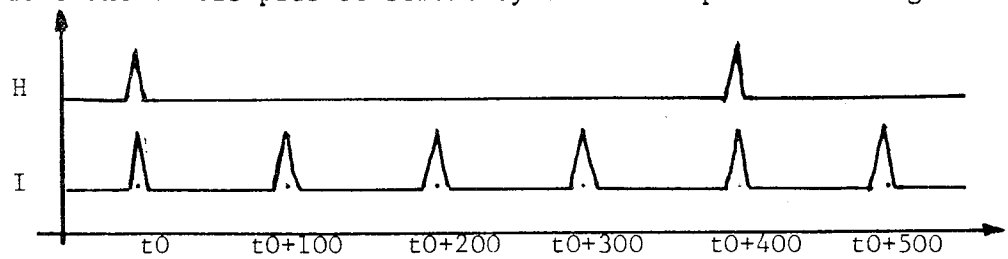
c) L'unité de MEMOIRE

1) Description discursive

Nous ne préciserons pas la technologie susceptible de répondre aux spécifications ci-après, pas plus que nous ne poserons la question du coût d'une telle mémoire. Ces points ont été abordés dans (3).

La mémoire comporte un bloc de 65536 mots de 16 bits (soit encore 128 K octets). Le temps de lecture d'un mot sera supposé de 1,2 µs et le temps d'écriture de 1,6 µs.

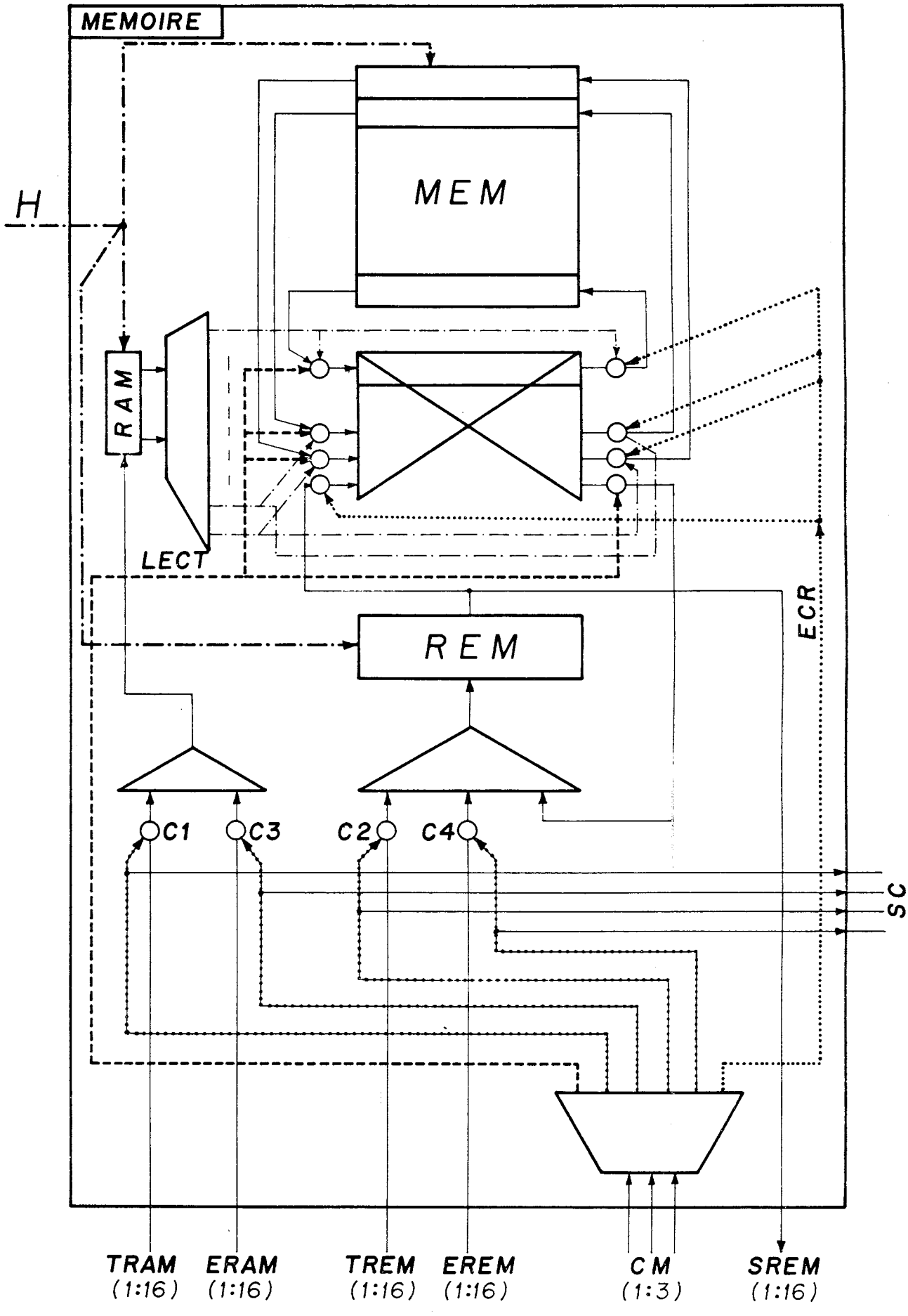
Dans tout ce qui suit, nous supposerons le temps découpé en intervalles de 100 Vs par une impulsion I. Les cycles élémentaires de traitement dureront 4 fois plus et seront synchronisés par une horloge H.



Le temps de lecture de la mémoire MEM sera donc de 3 cycles et son temps d'écriture de 4 cycles.

MEM a un registre d'échange REM de 16 bits et un registre d'adresse RAM (lui aussi de 16 bits).

Le contenu de REM peut être lu à l'extérieur de l'unité par la sortie SREM de 16 bits. L'unité de traitement peut envoyer à MEMOIRE, soit une adresse soit un mot à stocker, ces actions seront conditionnées respectivement par C1 et C2. L'unité d'échange elle aussi peut envoyer soit une adresse, soit un mot à stocker respectivement sous les conditions C3 et C4.



UNITE MEMOIRE

Lorsque RAM et REM contiennent des valeurs convenables on peut lancer les opérations de lecture ou écriture grâce aux signaux ECR et LEC. Il faudra alors garder inchangés les registres RAM et REM jusqu'à la fin de ces opérations, le signal BLOK remplira cette fonction.

Un cycle avec le signal ECR sera donc précédé par 3 cycles avec le signal BLOK et un cycle LEC par 2 cycles avec BLOK.

Il en résulte évidemment que les signaux ECR, LEC et BLOK sont mutuellement exclusifs. Il en est de même des signaux C1 et C3 qui conditionnent le chargement de RAM et C2 et C4 qui conditionnent le chargement de REM.

Comme nous souhaitons que TRAITEMENT n'ait qu'une seule liaison de 16 fils, cela entraîne l'exclusion mutuelle de C1 et C2. Nous supposerons que cela est vrai de plus pour C3 et C4. Enfin, C1, C2, C3, C4 sont chacun incompatibles avec LEC, ECR et BLOK. Nous avons 7 signaux dont un seul peut être vrai à la fois, on peut les coder à l'aide d'un nombre binaire de 3 bits CM (1 : 3)

2) Description en CASSANDRE

```
FILE: DESCRIPT CASSPGR  P1                          CAMBRIDGE  MONITOR  SYSTEM

'UNITE' MEMOIRE(H,IRAM(1:16),IREM(1:16),ERAM(1:16),EREM(1:16),CM(1:3);
SC(1:4),SREM(1:16)),
  'REGISTRE' REM(1:16),RAM(1:16),MEM(1:16,0:65535);
 'MEMOIRE' MEM(RAM,REM,D,3,4);
  'HORLOGE' H;
        <H>
           'SI' $CM 'ALORS'
             ()
             (  RAM<=IRAM)
             (  REM<=IREM)
             (  RAM<=ERAM)
             (  REM<=EREM)
             (  REM<=MEM(,$(RAM)))
             (  MEM(,$(RAM))<=REM)
             ();
             SREM:=REM;
            SC:='SI' $CM 'ALORS'
               ()
               (1000)
               (0100)
               (0010)
               (0001)
               ()
               ()
               ();
```

On voit que l'état BLOK est réalisé soit par CM = 000 soit CM = 111. On aurait pu mettre des compteurs locaux pour compter 3 ou 4 cycles, mais il est plus simple de faire ceci par microprogramme, donc de laisser au microprogramme le soin de faire précéder les signaux ECR et LEC du nombre convenable de signaux BLOK.

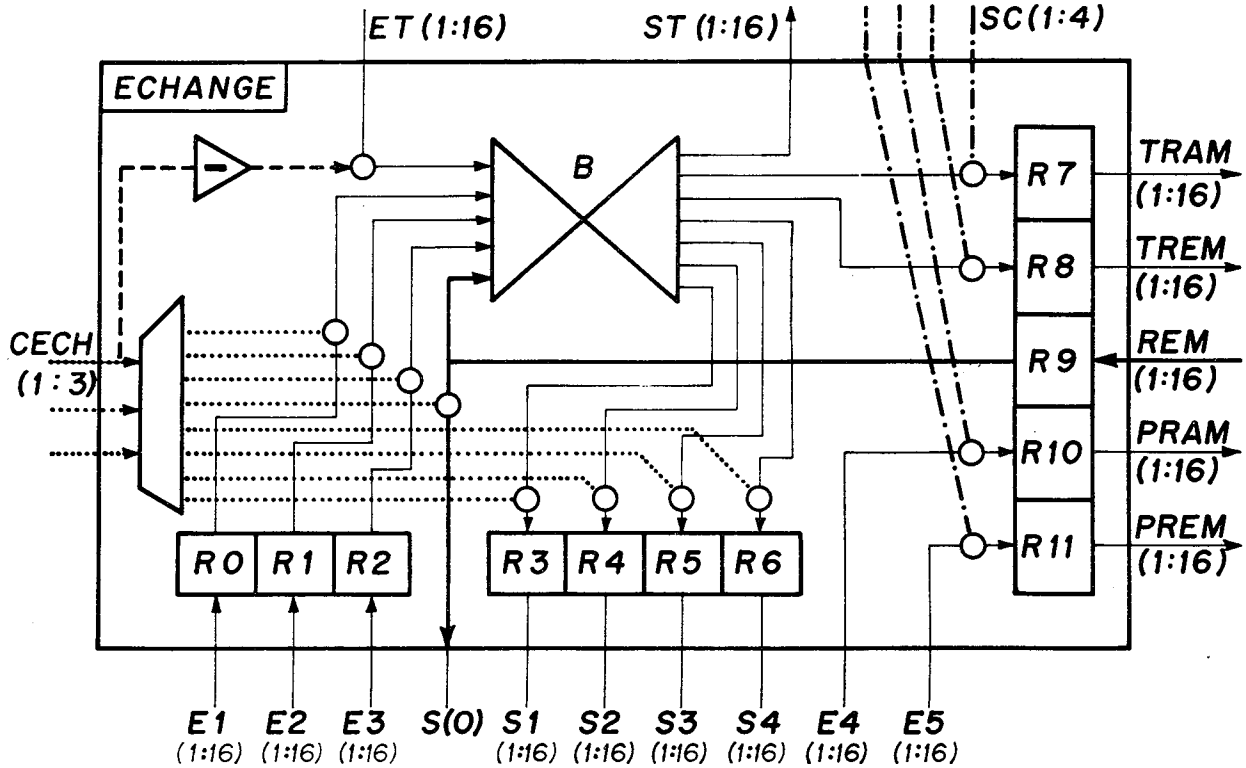
d) L'unité ECHANGE

1) Description discursive

Comme nous l'avons dit, c'est une version très simplifiée de cette unité, réduite à ses interfaces avec TRAITEMENT et MEMOIRE que nous décrirons.

Nous supposerons que les données stockées dans les registres d'ECHANGE sont aussitôt disponibles sur les sorties de ces registres (appellés quelquefois "latch""). Pour rendre compte de ceci en CASSANDRE nous chargerons ces registres avec l'impulsion I,4 fois plus fréquente que H.

ECHANGE reçoit de CONTROLE un signal CECH (1 : 3), de TRAITEMENT une ligne de 16 bits et de MEMOIRE une ligne de 16 bits et la sortie SC de 4 bits. ECHANGE envoit vers MEMOIRE quatre lignes de 16 bits et une vers TRAITEMENT. ECHANGE a aussi des liaisons avec les blocs d'entrées-sorties. ECHANGE permet à TRAITEMENT de dialoguer avec un périphérique pendant qu'un autre périphérique dialogue avec MEMOIRE, ou à TRAITEMENT de dialoguer avec MEMOIRE.



2) Description en CASSANDRE

```
FILE: DESCRIPT CASSPGR  P1                          CAMBRIDGE  MONITOR  SYSTEM

'UNITE' ECHANGE(I,CECH(1:3),SC(1:4),ET(1:16),REM(1:16),E(1:16,1:5);
PRAM(1:16),PREM(1:16),TRAM(1:16),TREM(1:16),ST(1:16),S(1:16,0:4));
  'REGISTRE' REG(1:16,0:11);
  'SIGNAL' B(1:16),
  'HORLOGE' I;
        <I>
           'SI' CECH 'ALORS'
             ()
             ()
             ()
             ()
             (  REG(,3)<=B)
             (  REG(,4)<=B)
             (  REG(,5)<=B)
             (  REG(,6)<=B),
           'SI' SC 'ALORS'
             (  REG(,7)<=B)
             (  REG(,8)<=B)
             (  REG(,10)<=E(,4))
             (  REG(,11)<=E(,5)),
           REG(,9)<=REM,
           REG(,0:2)<=E(,1:3);
           S:=0,
           PRAM:=REG(,10);
           PREM:=REG(,11);
           TREM:=REG(,8);
           TRAM:=REG(,7);
           S(,1:4):=REG(,3:6);
           S(,0):=REG(,9);
           B:='SI' CECH 'ALORS'
             (ET)
             (ET)
             (ET)
             (ET)
             (REG(,0))
             (REG(,1))
             (REG(,2))
             (REG(,9));
```

Les 4 entrées de R (,3:6) et les entrées du bus B se trouvent conditionnées chacune par une valeur de CECH, c'est-à-dire sont mutuellement exclusives. C'est indispensable pour les entrées de B mais pas à priori pour les 4 autres. Mais les 4 entrées de R (,3:6) ne peuvent provenir que de TRAITEMENT ce qui entraîne qu'elles sont aussi mutuellement exclusives 2 à 2. De plus, on veut que ET et ST emprunte la même ligne bidirectionnelle entre ECHANGE et TRAITEMENT, ce qui entraîne bien alors l'incompatibilité des 8 actions codées à l'aide de CECH.

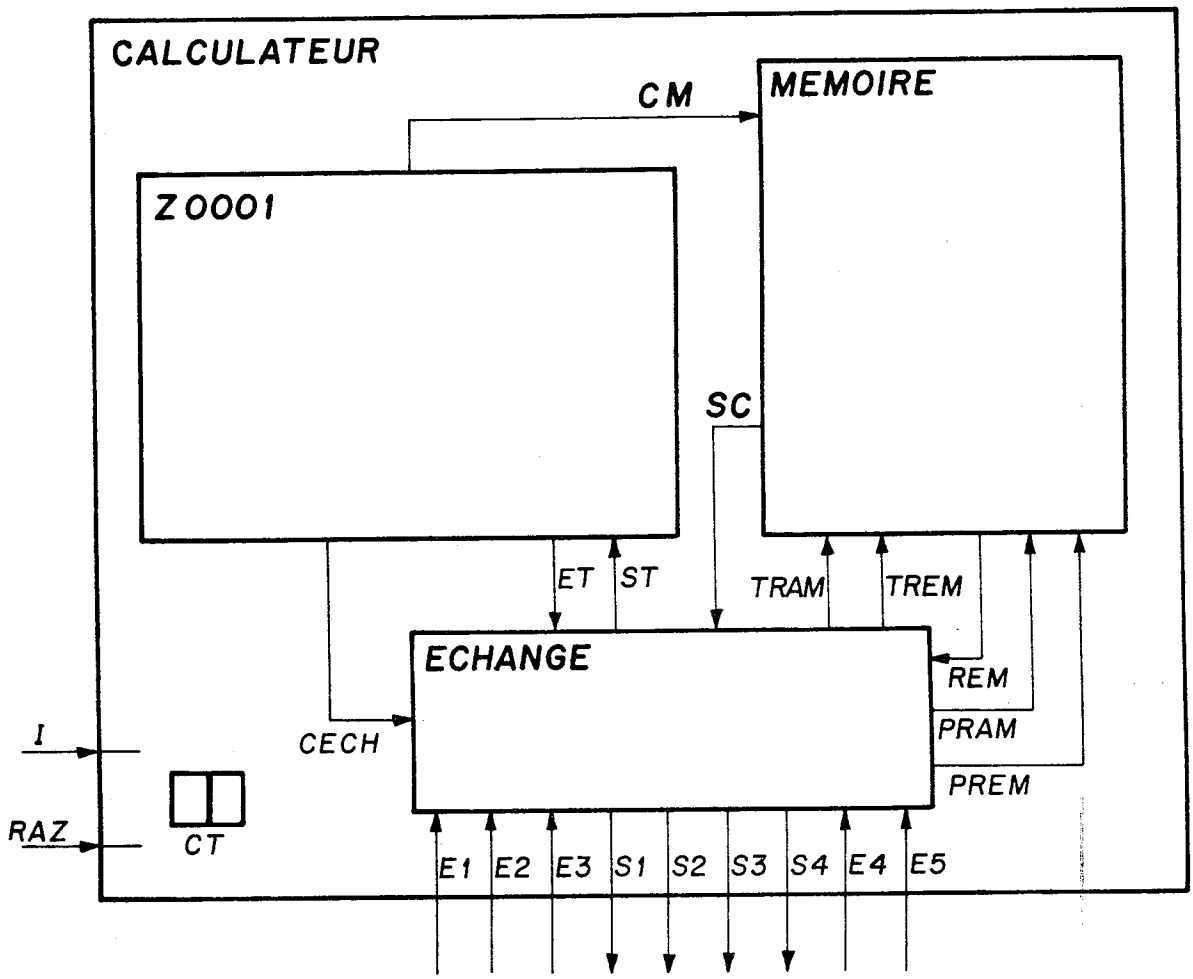
Les lignes bidirectionnelles sont rendues possibles grâce à la technologie des "circuits 3 états", mais n'existent pas en CASSANDRE (langage de description de réseau orienté). C'est pourquoi ET et ST sont séparées, bien que l'on tienne compte des incompatibilités que leur identification entrainera dans la réalité.

e) Unités Z0001 et CALCULATEUR :

Nous supposerons désormais que les blocs CONTROLE et TRAITEMENT sont intégrés dans une unité Z0001 et on appellera CALCULATEUR l'unité qui contient MEMOIRE, ECHANGE et Z0001.

```
'UNITE' CALCULATEUR (I,RAZ,E(1:16,1:5);S(1:16,1:4));
  'HORLOGE' I,
  'IMPULSION' H,
  'SIGNAL' E(1:16),S(1:16),TRAM(1:16),CM(1:3),CECH(1:3),PRAM(1:16),
  TREM(1:16),REM(1:16),PREM(1:16);
  'REGISTRE' CF(1:2),
  'EXTERNE' Z0001('HORLOGE','HORLOGE',,16;3,3,16),
  MEMOIRE('HORLOGE',,16,16,16,16,3;4,16),
  ECHANGE('HORLOGE',3,4,16,16,(16,3));
          Z0001(H,I,RAZ,E);CM,CECH,ST);
          MEMOIRE(H,TRAM,TREM,PRAM,C;SC,REM);
          ECHANGE(I,CECH,SC,ET,REM,E;PRAM,PREM,TRAM,TREM,ST,S);
    <I>     H:=/.CF,
          CF<= 'SI' SCF 'ALORS'
             (01)
             (10)
             (11)
             (00),
```

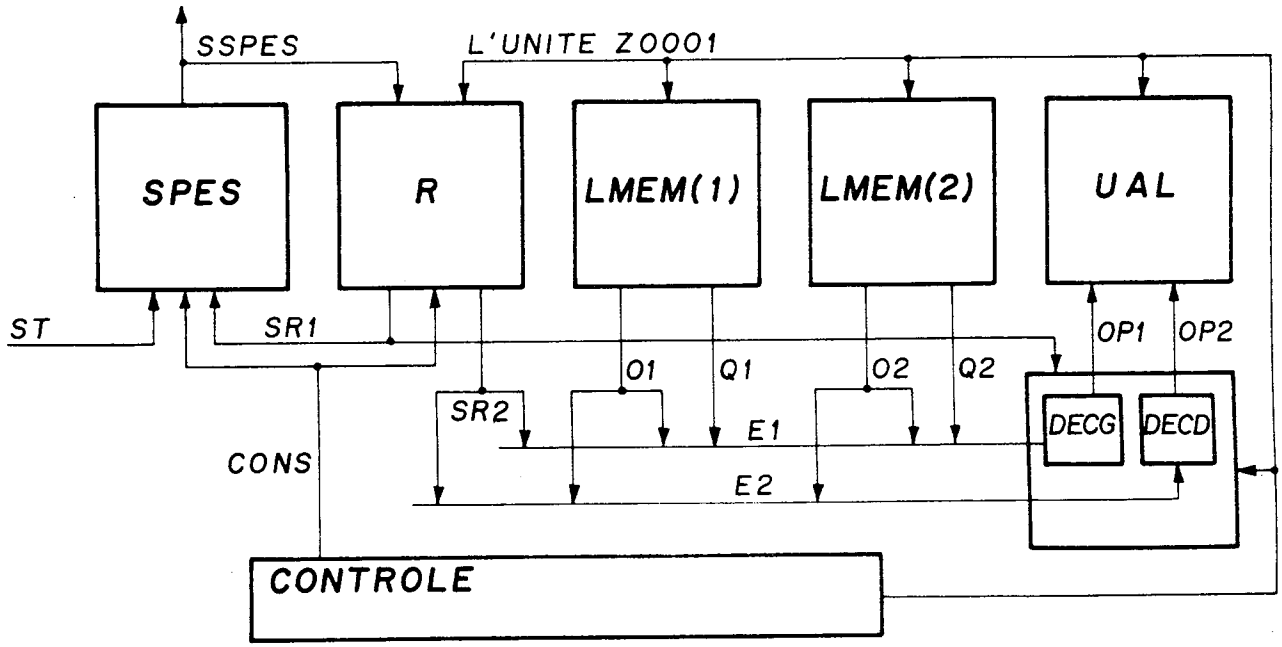
L'UNITE CALCULATEUR



Remarque :

CF fonctionne on peut le voir en compteur cyclique et génère une impulsion H toutes les 4 impulsions I.

L'UNITE Z0001



L'unité Z0001 contient une unité CONTROLE, une unité arithmétique et logique (UAL), une unité de décalage DEC, deux unités de mémoire locale 16 x 4 bits (LMEM) une mémoire locale 4 x 16 bits (SPES) et un registre 4 x 4 bits (R).

1) L'unité LMEM :

Elle contient une mémoire de 64 bits organisée en 16 mots de 4 bits et un registre AD de 4 bits pour adresser cette mémoire. AD peut être chargé à partir de la sortie S de UAL ou bien progresser ou régresser de 1, c'est un registre compteur-décompteur. Lorsque son contenu est 0000 en régressant ou 1111 en progressant il le signale. Ce compteur est cyclique (lorsqu'il est à 1111 et qu'il progresse la valeur suivante sera 0000).

Notons qu'un tel circuit existe en un seul boitier dans le commerce, de même que la mémoire 16 x 4 bits. L'unité LM contiendra donc 2 boites l'unité COMPTEUR et l'unité ML, plus un circuit de décodage.

```
'UNITE' COMPTEUR(H,P(0:3),PE,CU,CD,RAZ;Q(0:3),TC);
  'REGISTRE' CT(1:4);
  'SIGNAL' MD1(0:3),MD2(0:3);
  'HORLOGE' H;
              Q:=CT,
              MD1:=#D(1)(CT.MD2+CT.MD1+MD1.MD2)&0;
      'SI' CU&CD 'ALORS'
       (  MD2:=0001,
                  TC:=/.CT)
       (  MD2:=1111,
                  TC:=/.(-CT));
         <H>
            'SI' PE&(CU+CD)&RAZ 'ALORS'
              (  CT<=P)
              (  CT<=CT#(MD2#MD1))
              (  CT<=0000);
```

Suivant que les signaux CU, CD et PE sont à 1 on effectue respectivement +1, -1 et chargement de l'entrée P dans le compteur CT.

```
'UNITE' ML(I,A(0:3),D(0:3),CS;C(0:3));
  'REGISTRE' M(1:4,0:15);
  'HORLOGE' I;
     <I>      'SI' CS 'ALORS'
              M(,$(A))<=D;
              C:=M(,$(A));
```

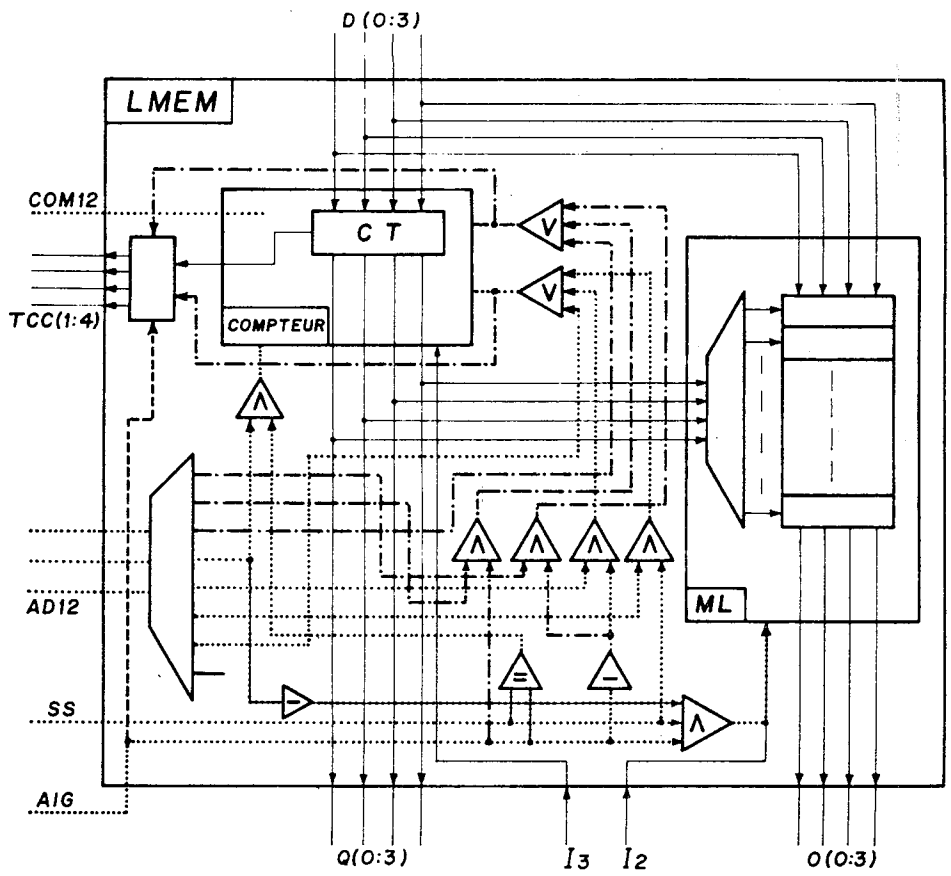
La condition permet d'écrire un mot de 4 bits en parallèle dans la mémoire M.

Les deux unités précédentes tiennent dans des boitiers à 14 "pattes" (plus les alimentations) (*)

(*) MSI Pocket Guide, Fairchild, janvier 1970.

```
'UNITE' LMEM(I2,I3,D(0:3),AD12(1:3),AIG,SS,COM12;Q(0:3),O(0:3),
        TCC(1:4));
  'HORLOGE' I2,I3;
  'SIGNAL' PE,CU,CD,CS,SC,TC;
  'EXTERNE' COMPTEUR('HORLOGE',4,,,,;4,),
            ML('HORLOGE',4,4,;4);
           COMPTEUR(H,D,PE,CU,CD,COM12;Q,TC);
           ML(I,Q,D,CS;O);
 'SI' $AD12 'ALORS'
       ()
       (   CU:=1)
       (   CU:=AIG)
       (   CU:=-AIG)
       (   PE:=SS=AIG;
                  SC:=1)
       (   CD:=1)
       (   CD:=AIG)
       (   CD:=-AIG);
            CS:=SS.AIG.-SC;
      'SI' AIG 'ALORS'
       (   'SI' CU&CD 'ALORS'
         (   TCC(1):=TC)
         (   TCC(3):=TC))
      'SINON'
        (   'SI' CU&CD 'ALORS'
          (   TCC(2):=TC)
          (   TCC(4):=TC));
```

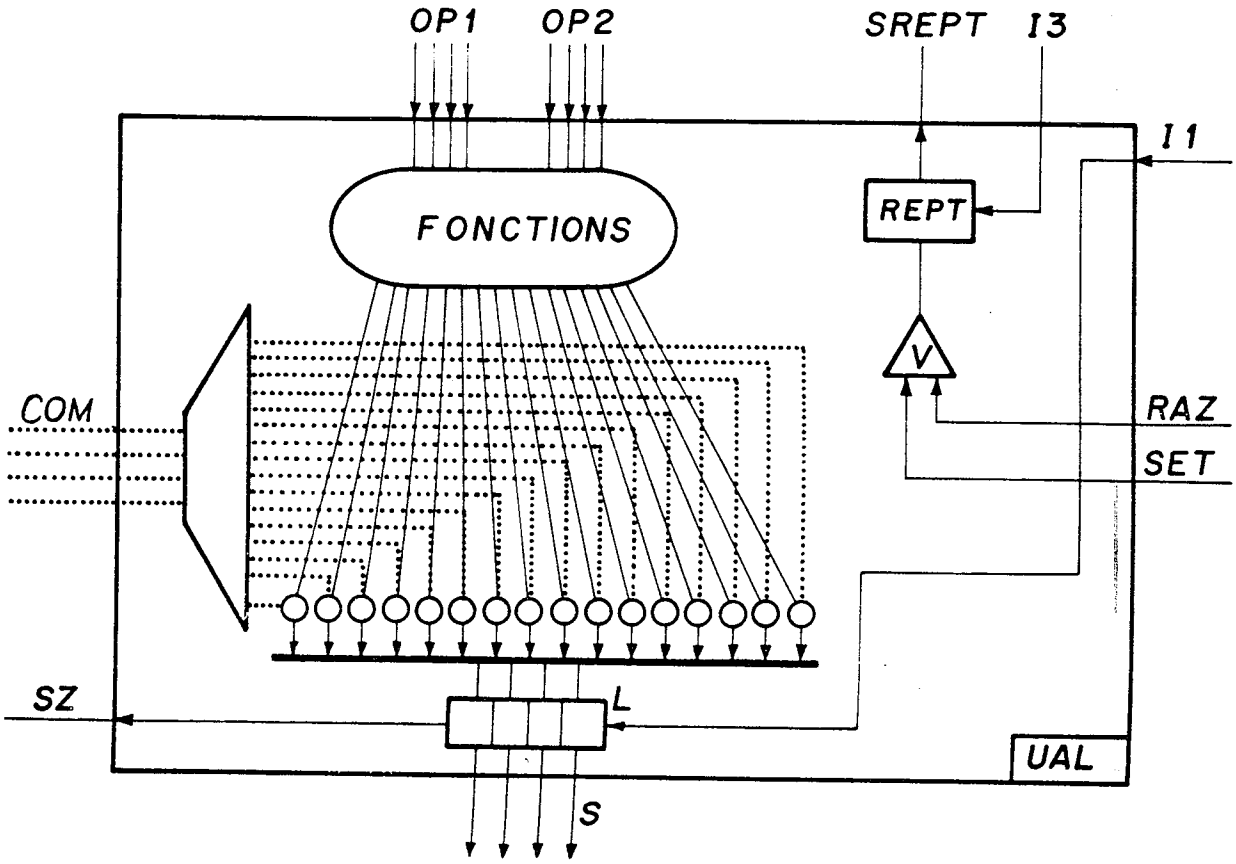
Suivant la valeur de AIG, l'unité LMEM sera considérée comme LMEM1 ou LMEM2 (ce qui est différent des 2 boites LMEM |1| et LMEM |2| , connectées dans Z0001, chacune à son tour pouvant être soit LMEM1 soit LMEM2. (voir microprogrammation)). C'est pourquoi, le signal AIG conditionne TCC (1:4), CU, CD et CS.



2) L'unité UAL :

Cette unité sera réalisée en une seule boite (il n'est pas sûr qu'elle existe dans un catalogue avec les fonctions que nous lui attribuons, mais il en existe de comparables en tous points). UAL est une unité formée de circuits combinatoires, capable d'effectuer sur deux opérandes de 4 bits 16 opérations logiques ou arithmétiques. UAL contient une bascule REPT qui peut stocker d'un cycle au suivant le report d'une addition binaire (ou d'une soustraction). Cette bascule a une entrée de remise à 0 et une entrée de remise à 1, on pourra de l'extérieur tester sa valeur.

Les fonctions sont commandées par signal COM (0:3)



Lorsque l'une des 16 fonctions possibles est sélecté son résultat est stocké dans L (0:3) et le report éventuel, si l'opération est une addition ou une soustraction, est stocké dans REPT. Comme on veut réutiliser L dans le même cycle on utilisera l'impulsion Il pour le charger. La sortie S indique si L est différent de 0. La "boite" UAL a 22 entrées-sorties.

```
FILE: DESCRIPT CASSPGR  P1                          CAMBRIDGE  MONITOR  SYSTEM

'UNITE' UAL(I1,I2,OP1(0:3),OP2(0:3),COM(0:3),SET,RAZ;S(0:3),SREPT,SZ);
  'REGISTRE' REPT,L(0:3);
  'SIGNAL' RS7(0:4),RS6(0:3),RS1(0:4),RS2(0:4),RS3(0:4),RS4(0:3),E1(0:
           3),E2(0:3),E3(0:3),E4(0:3),C,RS5(0:3),RS6(0:3);
  'HORLOGE' I1,I2;
              E1:=(OP1#OP2)#RS1(1:4);
              E2:=(OP1#-OP2)#RS7(1:4);
              E3:=(OP1#OP2)#RS3(1:4);
              E4:=(OP1#-OP2)#RS2(1:4);
              RS1:=RS4&REPT;
              RS2:=RS3&1;
              RS3:=RS6&0;
              RS7:=RS6&REPT;
              RS6:=*D11|(0&(OP1.-OP2)+0&CP1.RS7+0&-CP2.RS7);
              RS4:=*D11|(0&(OP1.OP2)+0&CP1.RS1+0&CP2.RS1);
              RS5:=*D11|(0&(OP1.-OP2)+0&CP1.RS2+0&-CP2.RS2);
              RS6:=*D11|(0&(OP1.OP2)+0&CP1.RS3+0&CP2.RS3);
         <I1>  L<=
            'SI' #COM 'ALORS'
              (E1)
              (E2)
              (E3)
              (E4)
              (-OP1.OP2)
              (OP2)
              (OP1#OP2)
              (OP1+OP2)
              (-(OP1+OP2))
              (-(OP1#OP2))
              (-OP2)
              (OP1+-OP2)
              (0000)
              (OP1.OP2)
              (OP1.-OP2)
              (OP1);
         <I2> REPT<=
            'SI' -/+COM(0:1)&RAZ&SET 'ALORS'
              (RS4(0))
              (0)
              (1);
              SREPT:=REPT;
              SZ:=/+L;
              S:=L;
```

Remarques :

On peut vérifier que les signaux E1, E3, E2, E4, correspondent respectivement au calcul de OP1 + OP2 + REPT, OP1 + OP2; OP1 - OP2 - 1 + REPT et OP1 - OP2. La soustraction s'obtient en ajoutant le complément vrai (corrigé de 1) du 2e opérande. Les fonctions booléennes de 2 variables qui ne sont pas effectuées par UAL sont (-OP1 V OP2), (-OP1 V -OP2), -OP1, 1111, seul OP2 peut être complémenté.

3) L'unité DEC :

Cette unité reçoit également deux opérandes de 4 bits OP1 et OP2 et elle les restitue après leur avoir fait subir éventuellement un décalage (à gauche pour OP1, à droite pour OP2). Elle contient un registre A (1:2) dont la valeur indique le nombre de positions de décalage à effectuer. A peut être chargé soit à partir de 2 bits du registre R (4 x 4 bits) on verra lesquels , soit à partir de la sortie S de UAL. Deux sous-unités DECG et DECD traitent OP1 et OP2.

DEC est un petit automate à 3 états, DECALG, DECALD et PASDECAL. Ces états peuvent être forcés de l'extérieur à l'aide de 3 commandes.

α) Unité DECG :

Elle a une entrée OP1 (1:4) une entrée commande de CO1(1:2) et une sortie SG (1:4), suivant la valeur de SG, OP1 est décalé de 0, 1, 2 ou 3 positions.

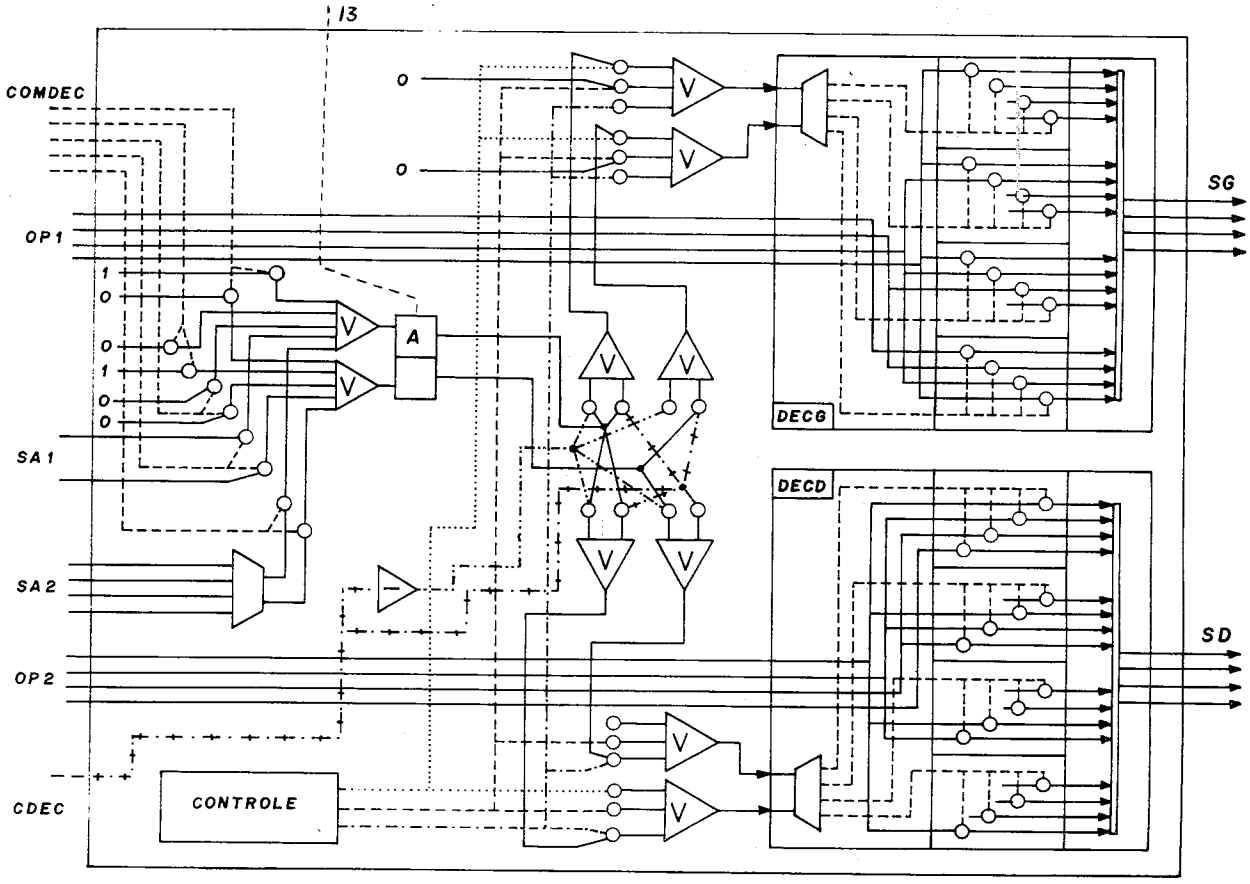
```
'UNITE' DECG(OP1(1:4),CO1(1:2);SG(1:4));
  'SIGNAL' OP(0:5),C(0:5);
  'EXTERNE' VALV2(2,4);
          VALV2(CO1,C);
          OP:=OP1&0000;
      'POUR' I=0 'A' 3
       'DEBUT'
        'SI' C(I) 'ALORS'
             SG:=OP(I:3+I);
       'FIN';
```

L'unité DEC a aussi une entrée-commande CDEC indiquant le mode, décalage gauche ou décalage droit, de fonctionnement. C'est une entrée de validation qui agit suivant l'état. Dans le mode décalage gauche, c'est la commande CO1 de DECG qui est égale au contenu de A, la commande CO2 est égale au complément à 4 de CO1. Dans le mode décalage droit CO2 = A et CO1 = complément à 4 de A. La microprogrammation montrera ultérieurement l'utilité de ceci.

β) Unité DECD :

```
FILE: DESCRIPT CASSPOR  P1

'UNITE' DECD(OP2(1:4),CO2(1:2);SD(1:4));
      'SI' $CO2 'ALORS'
       (  SD:=OP2)
       (  SD:=0&OP2(1:3))
       (  SD:=00&OP2(1:2))
       (  SD:=000&OP2(1));
```

```
'UNITE' DEC(13,CDEC,COMDEC(1:5),OP1(1:4),OP2(1:4),SA1(1:2),SA2(1:4);SG(
        1:4),SD(1:4));
  'REGISTRE' A(1:2);
  'SIGNAL' CO1(1:2),CO2(1:2);
  'HORLOGE' 13;
  'EXTERNE' DECG(4,2;4),DECD(4,2;4);
          DECG(OP1,CO1;SG);
          DECD(OP2,CO2;SD);
     <13> 'SI' COMDEC 'ALORS'
          (A<=SA1)
          ('SI' SA2 'ALORS'
              (A<=00)
              (A<=01)
              (A<=10)
              (A<=11))
          (A<=00)
          (A<=01)
          (A<=10);
PASDECAL: CO1:=00;
          CO2:=00;
DECALG  : 'SI' CDEC 'ALORS'
             (CO1:=A)
          'SINON'
             ('SI' $A 'ALORS'
                 (CO2:=00)
                 (CO2:=11)
                 (CO2:=10)
                 (CO2:=01));
DECALD  : 'SI' CDEC 'ALORS'
             (CO2:=A)
          'SINON'
             ('SI' $A 'ALORS'
                 (CO1:=00)
                 (CO1:=11)
                 (CO1:=10)
                 (CO1:=01));
```



4) L'unité SPES :

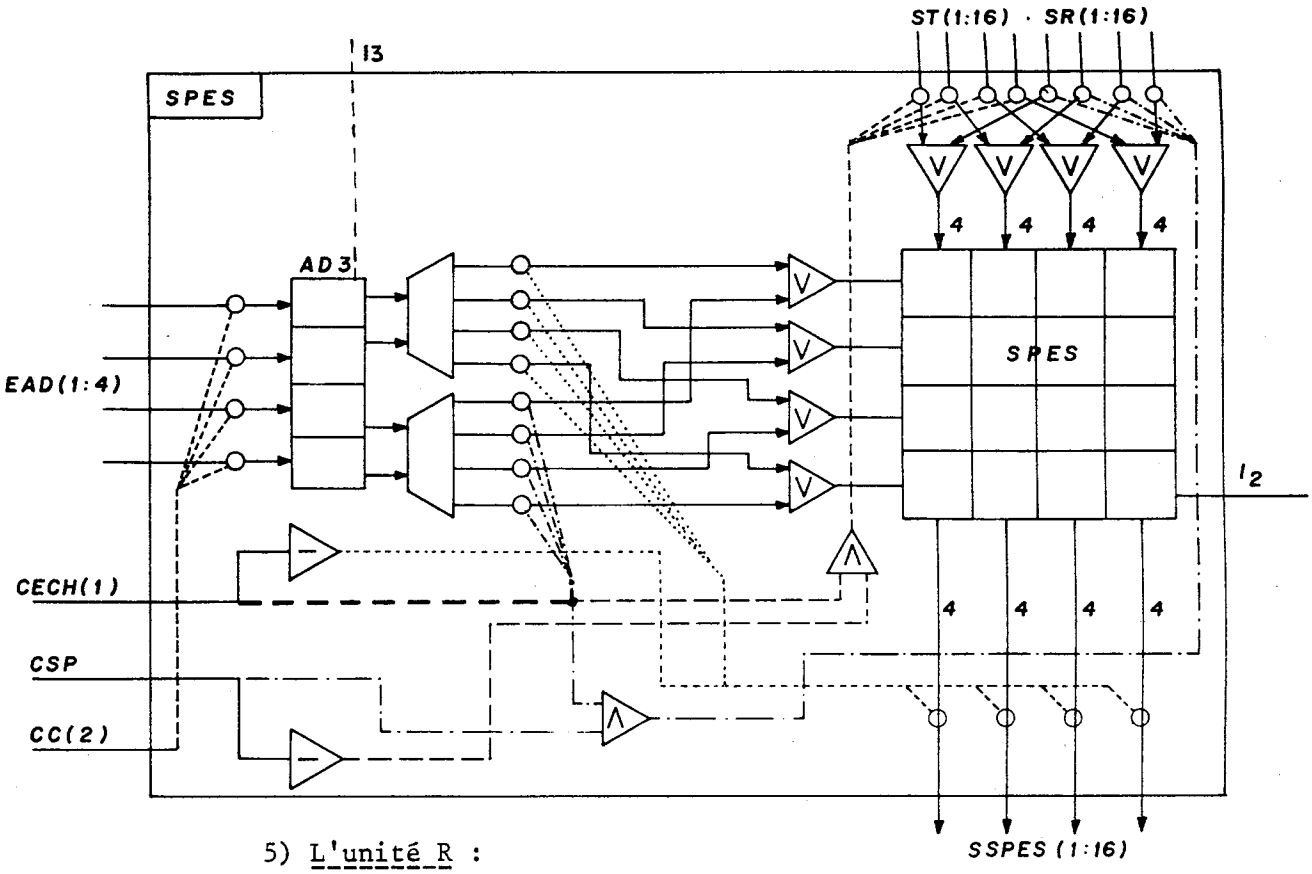
L'unité SPES contient 16 registres de 4 bits, comme l'unité RSPES largement décrite dans la partie A, mais ses fonctions sont moins élaborées pour des raisons d'économie.

Les registres peuvent être organisés en mots de 16 bits et on peut lire l'un des 4 mots pendant que l'on écrit dans un autre. Un registre d'adresse AD3 (1:4) est associé à SPES, les 2 bits poids forts AD3 (1:2) indiquant l'adresse du mot lu et AD3 (3:4) l'adresse du mot écrit.

On peut également lire un mot de 4 bits, nous verrons comment avec l'unité R suivante.

Outre les horloges I2 et I3, 3 signaux de commande CECH (1), CSP, CC(2) interviennent dans cette unité.

```
FILE: DESCRIPT CASSPOR  P1                          CAMBRIDGE  MONITOR  SYSTEM

'UNITE' SPES(I2,I3,SI(1:16),SR(1:16),EAD(1:4),CECH,CSP,CC(2:2);SSPES(1
        :16));
  'REGISTRE' AD3(1:4),SPE(1:16,0:3);
  'HORLOGE' I2,I3;
          SSPES:=SPE(,#(AD3(1:2)));
         <I2>
          SPE(,#(AD3(3:4)))<='SI' CECH 'ALORS'
             ('SI' CSP 'ALORS' ST 'SINON' SR);
         <I3>
            'SI' CC 'ALORS' AD3<=EAD3;
```



5) L'unité R :

Elle contient un registre de 4 mots de 4 bits RR et un registre d'adresse associé ADO, qui fonctionne en compteur-décompteur (on recopiera sa définition sur celle de l'unité COMPTEUR). Quant à RR, il peut recevoir soit 16 bits en parallèle en provenance de SPES, soit 4 bits en provenance de SPES ou de UAL. RR fonctionne également en compteur-décompteur de 16 bits (4 unités COMPTEUR).

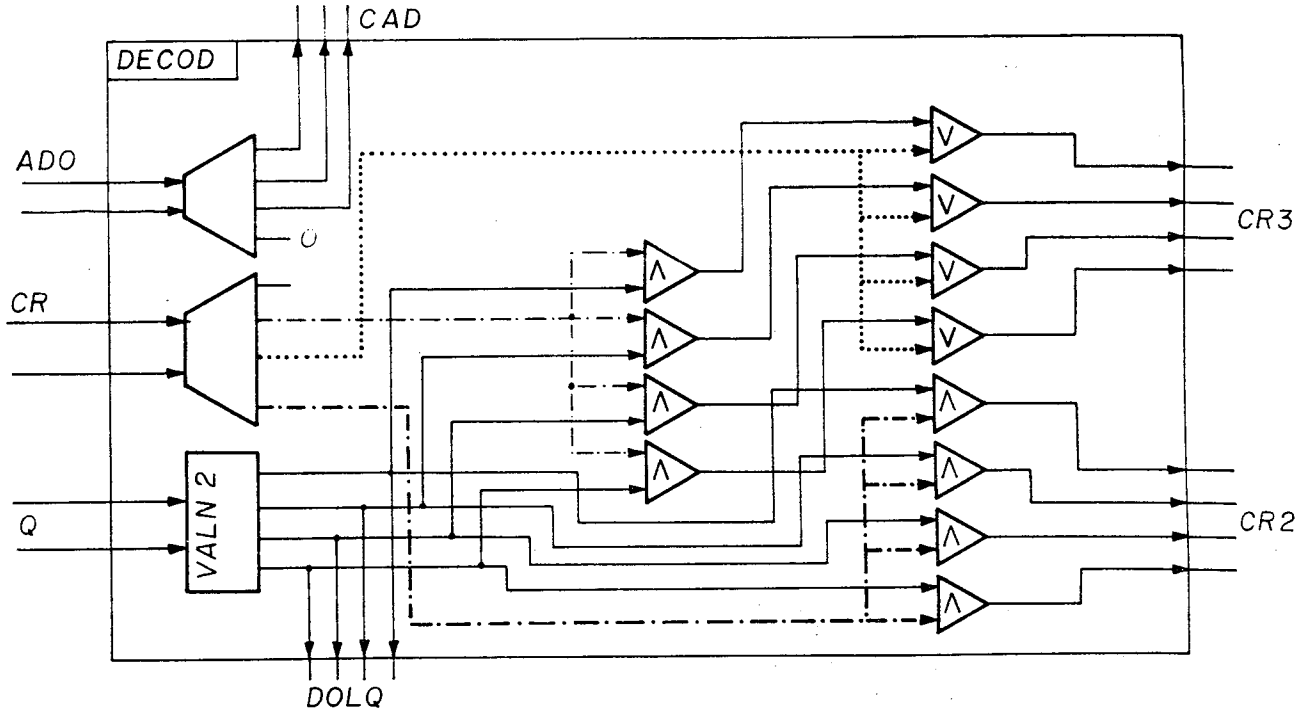
```
'UNITE' DECOD(ADO(1:2),CR(1:2),Q(0:1);CAD(1:3),DOLQ(1:4),CR3(1:4),CR2(
        1:4));
  'SIGNAL' CR1(1:4);
  'EXTERNE' VAL2(2;4),
      'SI' -CR(1).CR(2) 'ALORS'
                CR1:=DOLQ;
      'SI' CR(1).CR(2) 'ALORS'
                CR2:=DOLQ;
              CR3:='SI' CR(1).-CR(2) 'ALORS' 1111 'SINON' CR1;
      'SI' $ADO 'ALORS'
           ()
           (CAD(3):=1)
           (CAD(1):=1)
           (CAD(2):=2),
               VAL2(1|Q;DOLQ);
```

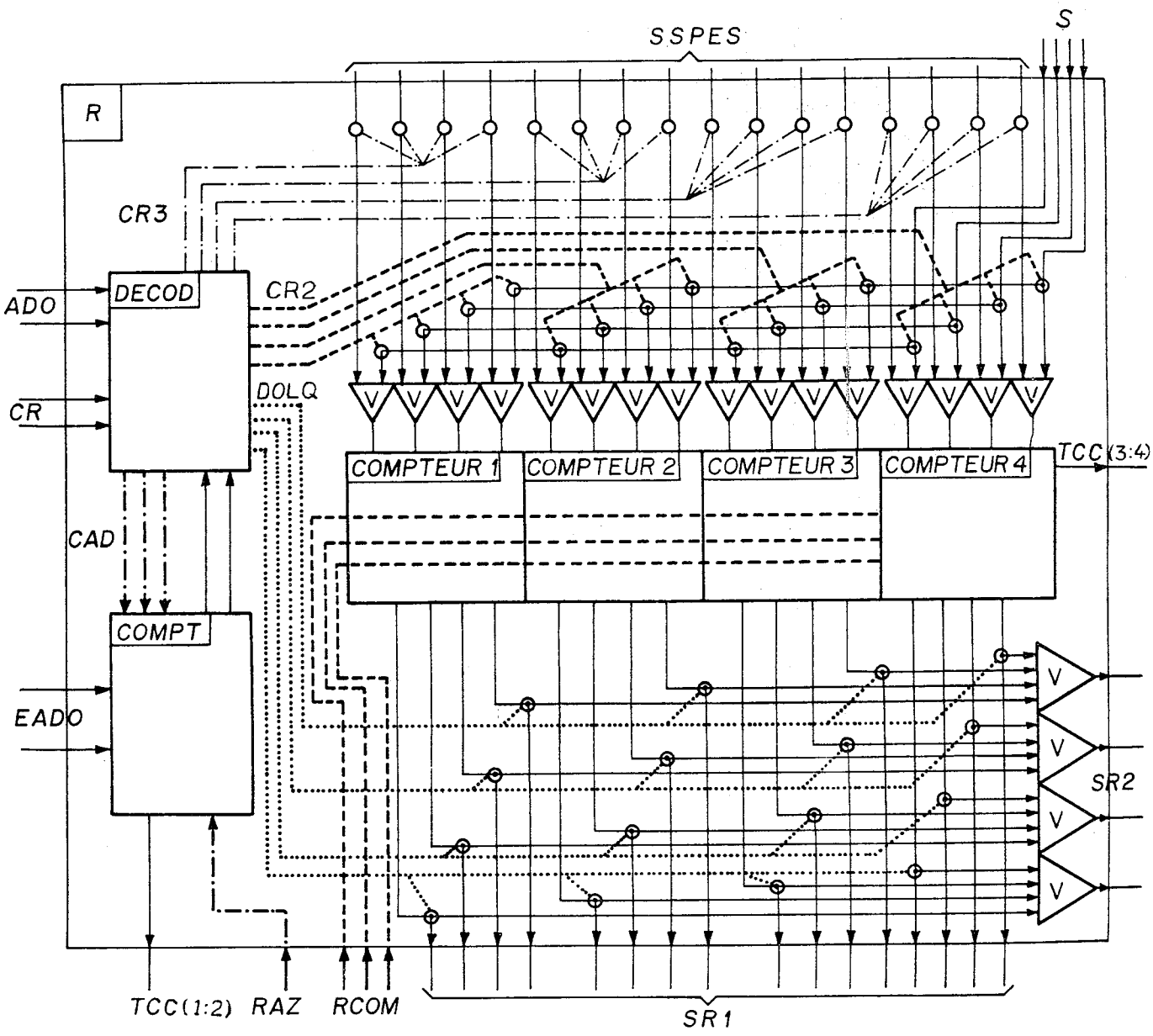
```
'UNITE' COMPT(H,P(0:1),PE,CO,CD,RAZ;Q(0:1),TC);
  'REGISTRE' ADD(1:2);
  'SIGNAL' M01(0:1),M02(0:1);
  'HORLOGE' H;
      'SI' CO&CD 'ALORS'
       ( M02:=01;
                 TC:=/.ADD)
       ( M02:=11;
                 TC:=/.-ADD);
            M01:=*D|11|(ADD.M02+ADD.M01+M01.M02)&0;
            Q:=ADD;
        <H>
          'SI' PE&(CO+CD)&RAZ 'ALORS'
           ( ADD<=P)
           ( ADD<=ADD#(M02#M01))
           ( ADD<=00);
```

FILE: DESCRIPT CASSPGR P1 CAMBRIDGE MONITOR SYSTEM

```
'UNITE' R(I2,I3,ADD(1:2),RAZ,CR(1:2),RCOM(1:3),EADD(1:2),S(1:4),SSPES(1:
        16);SR1(1:16),SR2(1:4),TCC(1:4));
  'SIGNAL' TC4,TC1,Q(0:1),DOUQ(1:4),TC3,TC2,CAD(1:3),CR3(1:4),CR2(1:4),
           RR2(0:3,0:3),RR1(0:3,0:3),BU(1:16),TC5;
  'HORLOGE' I3,I2;
  'EXTERNE' COMPT('HORLOGE',2,,,,;2,),
            COMPTEUR('HORLOGE',4,,,,;4,),
            DECOD(2,2,2;3,4,4,4);
      'SI' CR2&CR3 'ALORS'
       ( RR1(,0):=S)
       ( RR1(,1):=S)
       ( RR1(,2):=S)
       ( RR1(,3):=S)
       ( RR1(,0):=SSPES(1:4))
       ( RR1(,1):=SSPES(5:8))
       ( RR1(,2):=SSPES(9:12))
       ( RR1(,3):=SSPES(13:16));
            SR1:=RR2(,0)&RR2(,1)&RR2(,2)&RR2(,3);
      'SI' DOUQ 'ALORS'
       ( SR2:=RR2(,0))
       ( SR2:=RR2(,1))
       ( SR2:=RR2(,2))
       ( SR2:=RR2(,3));
            TCC:=CAD(3).TC&CAD(2).TC&RCOM(3).TC5&RCOM(2).TC5;
            COMPT(I3,EADD,CAD(1),CAD(2),CAD(3),RAZ;Q,TC);
            DECOD(ADD,CR,Q;CAD,DOUQ,CR3,CR2);
            COMPTEUR(1)(I2,RR1(,0),CR2(1)+CR3(1),TC2.RCOM(1),TC2.RCOM
            (2),RCOM(3);RR2(,0),TC1);
            COMPTEUR(2)(I2,RR1(,1),CR2(2)+CR3(2),TC3.RCOM(1),TC3.RCOM
            (2),RCOM(3);RR2(,1),TC2);
            COMPTEUR(3)(I2,RR1(,2),CR2(3)+CR3(3),TC4.RCOM(1),TC4.RCOM
            (2),RCOM(3);RR2(,2),TC3);
            COMPTEUR(4)(I2,RR1(,3),CR2(4)+CR3(4),RCOM(1),RCOM(2),RCOM
            (3);RR2(,3),TC4);
            TC5:=TC1.TC2.TC3.TC4;
```

On peut voir que le registre de 16 bits contenu dans les unités COMPTEUR, reçoit sous les conditions CR3, soit les 16 bits de SSPES en parallèle, soit un mot de 4 bits pris parmi les 16 bits de SSPES, qui est


CAD
DECOD
ADO
0
CR
Q
VALN 2
CR3
CR2
DOLQ


S
SSPES
R
CR3
ADO
DECOD
CR2
CR
DOLQ
COMPTEUR 1
COMPTEUR 2
COMPTEUR 3
COMPTEUR 4
TCC(3:4)
CAD
COMPT
EADO
SR2
TCC(1:2)
RAZ
RCOM
SR1

stocké dans la partie du registre adressée par les 2 bits ADO de COMPT. sous la condition CR2 il reçoit S, stocké dans la partie adressée par ADO. C'est cette partie dont les sorties sont branchées sur SR2 (1:4), alors que toutes les sorties de RR sont disponibles sur SR1 (1:16). Les signaux CAD (1:3) et RAZ commandent respectivement le stockage de EADO dans ADO, l'incrémentation de 1 de ADO, la décrémentation de 1 et sa remise à zéro.

6) L'unité Z0001 :

Nous avons décrit les 5 éléments qui constituent Z0001 avec l'adjonction de CONTROLE, il reste à les interconnecter.

L'unité de Z0001 contient 2 bus E1 et E2 de 4 bits, ils peuvent recevoir SR2, les sorties des deux unités LMEM|1| et LMEM|2| et les conduisent à l'unité DEC qui, elle même les transmet à UAL.

Le résultat des opérations de DEC et UAL peut être testé, et ainsi positionner des bascules ou encore il est envoyé par S dans R, LMEM |1| ou LMEM(2). LMEM |1| et LMEM |2| sont exactement du même modèle, on ne peut les distinguer que par une bascule AIG qui pointe sur l'unité qui sera appelée LMEM |1|, -AIG pointant sur LMEM |2|. On voit que pour permuter entièrement les contenus de LMEM |1| et LMEM |2| il suffit en réalité de changer AIG en -AIG, ceci peut économiser un transfert pouvant aller jusqu'à 16 mots de 4 bits donc 16 cycles de microinstruction.

```
'UNITE' ZOOOI(H,I,RAZ,ET(1:16);CM(1:3),CECH(1:3),ST(1:16));
  'SIGNAL' TE(1:8),E1(1:4),E2(1:4),S(1:4),AD1(1:4),O1(1:4),AD2(1:4),O2
  (1:4),SR1(1:16),SR2(1:4),OP1(1:4),OP2(1:4),CONS(1:4),COMMANDE(1:15),
  TEST(0:9),AIG,CDEC,SS,CSP,CE1(1:2),CE2(1:2),CR(1:2),CUAL(1:4),AD12(1
  :3),ADO(1:2),CONS2(1:2),CC(1:2),CONTR(1:43),SSPES(1:16);
  'HORLOGE' H,I;
  'IMPULSION' I1,I2,I3;
  'EXTERNE' VALN3(3;8),
            LMEM('HORLOGE','HORLOGE',4,3,,,;4,4,4),
            CONTROLE('HORLOGE','HORLOGE','HORLOGE',4,10,,8;43,3,3),
            UAL('HORLOGE','HORLOGE',4,4,4,,;4,,),
            DEC('HORLOGE',,8,4,4,2,4;4,4),
            SPES('HORLOGE','HORLOGE',16,16,4,,,;16),
            R('HORLOGE','HORLOGE',2,,2,3,2,4,16;16,4,4),
            HORLOGE('HORLOGE','HORLOGE';'HORLOGE','HORLOGE','HORLOGE');
              VALN3(1)(AIG&CE1;);
              VALN3(2)(AIG&CE2;);
              HORLOGE(H,I;I1,I2,I3);
   'SI' VALN3(1)(;*) 'ALORS'
    (  E1:=SR2)
    (  E1:=O1)
    (  E1:=O2)
    (  E1:=AD1)
    (  E1:=SR2)
    (  E1:=O2)
    (  E1:=O1)
    (  E1:=AD2);
   'SI' VALN3(2)(;*) 'ALORS'
    (  'SI' CC(1) 'ALORS'
             E2:=CONS)
    (  E2:=SR2)
    (  E2:=O1)
    (  E2:=O2)
    (  'SI' CC(1) 'ALORS'
             E2:=CONS)
    (  E2:=SR2)
    (  E2:=O2)
    (  E2:=O1);
         R(I2,I3,ADO,COMMANDE(11),CR,COMMANDE(8:10),CONS2,S,SSPES
         ;SR1,SR2,TEST(0:3));
         SPES(I2,I3,ET,SR1,CONS,CECH(1),CSP,CC(2);SSPES);
         DEC(I3,CDEC,COMMANDE(1:5)&COMMANDE(13:15),E1,E2,SR1(13:
         14),S;OP1,OP2);
         UAL(I1,I2,OP1,OP2,CUAL,COMMANDE(6),COMMANDE(7);S,TEST(4)
         ,TEST(5));
         LMEM(1)(I3,I2,S,AD12,AIG,SS,COMMANDE(12);AD1,C1,TE(1:4));
         LMEM(2)(I3,I2,S,AD12,-AIG,SS,COMMANDE(12);AD2,C2,TE(5:8));
         CONTROLE(H,I1,I3,S,TEST(0:9),,;CONTR(1:43),CM(1:3),CECH(
         1:3));
         CR:=CONTR(1:2);
         CSP:=CONTR(3);
         CUAL:=CONTR(4:7);
         CE1:=CONTR(8:9);
         SS:=CONTR(10);
         AD12:=CONTR(11:13);
         ADO:=CONTR(14:15);
         CE2:=CONTR(16:17);
         CONS2:=CONTR(18:19);
         CONS:=CONTR(20:23);
         CDEC:=CONTR(24);
         CC:=CONTR(25:26);
         AIG:=CONTR(27);
         COMMANDE:=CONTR(28:42);
         TEST(0:9):=TE(1:4)+TE(5:8);
         ST:=SSPES;
```

II - L'unite CONTROLE

Les unités déjà décrites constituent la partie opératoire de Z0001, leur fonctionnement dépend d'un certain nombre de commandes en provenance, à chaque cycle de synchronisation H, de la partie CONTROLE. Cette dernière connaît l'état des registres des unités opératoires grâce aux 10 signaux TEST et à la sortie S de l'unité UAL. Pour reprendre une comparaison de GLUSHKOV, les fils de commande en provenance de CONTROLE et les fils de test qui y vont sont comparables respectivement aux fibres motrices et sensitives dans un système nerveux. (On retrouve des considérations voisines dans (4)).

Une sous-unité TESTT peut tester une à une les entrées "sensitives", elle peut aussi positionner des bascules à l'aide de certaines commandes ou encore tester leur contenu. En fonction du résultat des tests de la partie CONTROLE ou de l'état des bascules qui lui appartiennent, la sous-unité SEQUENCE assure alors l'enchainement séquentiel des différents ensembles de commandes. Ces ensembles sont constitués de bits regroupés dans des sous-ensembles, ou "champs," de format fixe. Pour en faciliter la réalisation le format du "mot" lui-même qui contient tous les champs des commandes exécutables à un instant donné, est aussi fixe. On peut alors stocker les mots de commande ou "microinstructions" dans une mémoire de microprogrammes. Dans Z0001 cette mémoire aura des mots de 32 bits et pourra en avoir jusqu'à 512. Elle forme une sous-unité MEMC, l'adresse de la microinstruction courante dans MEMC est fournie à chaque cycle par SEQUENCE.

Nous allons décrire directement la réalisation et le fonctionnement de l'automate de contrôle microprogrammé de Z0001 quitte à le replacer par la suite (partie IV) dans une classification des machines microprogrammées.

Une sous-unité COMMAND affiche à chaque cycle toutes les commandes élémentaires en les scindant en 2 groupes, celles qui vont vers les parties opératoires (fibres motrices), celles qui vont vers les sous-unités SEQUENCE et TEST (commandes d'enchaînement séquentiel). Une dernière sous-unité CODDE contrôle la fabrication du mot de commande de 88 bits de COMMAND, à partir des 32 bits qui proviennent de MEMC. Pour cela il interprète un champ particulier "CODE" de 4 bits de ces microinstructions. Nous avons appelé (Ecole de printemps DRME) "double décodage", ce principe.

a) L'unité MEMC

Elle contient les 512 x 32 bits de la mémoire de contrôle, reçoit en entrée une adresse ADMM (1:9) en provenance de SEQUENCE et fournit en sortie une microinstruction MICRO (1:32) qui va dans COMMAND (MICRO (1:28) et dans CODDE (MICRO (29:32)).

```
'UNITE' MEMC(ADMM(0:8);MICRO(1:32));
  'REGISTRE' MEMM(1:32,0:511);
             MICRO:=MEMM(,#(ADMM));
```

b) L'unité CODDE

Elle reçoit en entrée la sortie de la bascule TESST et le champ CODE de MICRO. Elle fabrique un signal CINS (1:8) qui va dans COMMAND aiguiller les autres champs de MICRO vers les différentes parties du mot de commande et les signaux PLUS 1, PLUS 0, MOINS 1, PLUS 2, ADR et ADD qui vont à l'unité SEQUENCE.

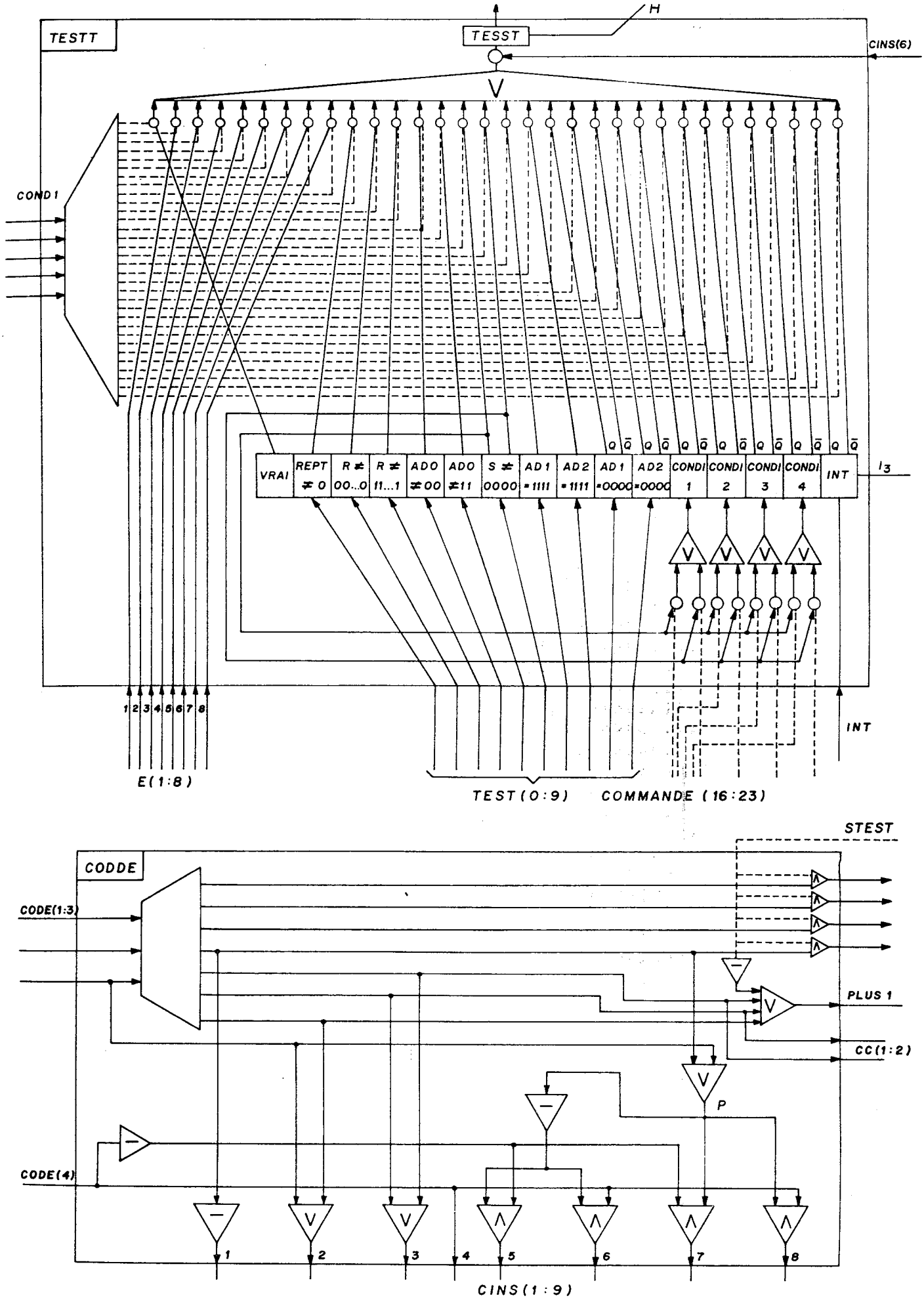
```
FILE: DESCRIPT CASSPUR  P1                          CAMBRIDGE  MONITOR  SYSTEM

'UNITE' CODDE(CODE(1:4),STEST;PLUS1,MOINS1,PLUS2,ADD,ADR,CC(1:2),CINS(
        1:8));
  'SIGNAL' P,COD(0:7);
             COD(#(CODE(1:3))):=1;
             CC:=COD(1:2);
      'SI' STEST 'ALORS'
       (  ADD:=COD(5);
                ADR:=COD(4);
                PLUS2:=COD(5);
                MOINS1:=COD(6));
             PLUS1:=-STEST+COD(0)+COD(1)+COD(2);
             P:=COD(3)+CODE(1);
             CINS(1:3):=-COD(3)&(CODE(1)+COD(0))&(COD(1)+COD(2));
      'SI' CODE(4) 'ALORS'
       (  CINS(4):=1;
                CINS(5):=-P;
                CINS(6):=P)
      'SINON'
        (  CINS(5):=-P;
                CINS(7):=P);
```

c) L'unité TESTT

1) Description

Cette unité contient 16 bascules positionnables par des commandes en provenance de COMMAND et par un fil INT qui vient de l'unité ECHANGE. La valeur de ces bascules et les entrées E, peuvent être testées une par une suivant le code COND 1, le résultat de ce test positionne la bascule TESST.

```
'UNITE' TESTT(13,H,COMMANDE(16:23),INT,CINS,E(1:8),TEST(0:9),
        COND1(1:3),STEST);
  'HORLOGE' 13,H;
  'REGISTRE' TESST,T(1:16);
      <H> 'SI' CINS 'ALORS'
           TESTT<='SI' $COND1 'ALORS'
                 T(1)&E(1:8)&T(2:7)&-T(7)&T(9:16)&-T(10:16);
      <13> 'SI' COMMANDE&INT 'ALORS'
                (T(12)<=T(7))
                (T(13)<=T(7))
                (T(14)<=T(7))
                (T(15)<=T(7))
                (T(12)<=-T(7))
                (T(13)<=-T(7))
                (T(14)<=-T(7))
                (T(15)<=-T(7))
                (T(16)<=1),
           'SI' TEST 'ALORS'
                (T(2)<=1)
                (T(3)<=1)
                (T(4)<=1)
                (T(5)<=1)
                (T(6)<=1)
                (T(7)<=1)
                (T(8)<=1)
                (T(9)<=1)
                (T(10)<=1)
                (T(11)<=1);
```


TESTT
TESST
H
CINS(6)
V
COND 1
VRAI
REPT ≠ 0
R ≠ 00...0
R ≠ 11...1
ADO ≠ 00
ADO ≠ 11
S ≠ 0000
AD1 = 1111
AD2 = 1111
AD1 = 0000
AD2 = 0000
CONDI 1
CONDI 2
CONDI 3
CONDI 4
INT
I3
1 2 3 4 5 6 7 8
E(1:8)
INT
TEST(0:9)
COMMANDE (16:23)
STEST
CODDE
CODE(1:3)
PLUS 1
CC(1:2)
P
CODE(4)
1
2
3
4
5
6
7
8
CINS(1:9)

2) Signification des commandes

En se reportant à I, on pourra vérifier que les entrées TEST (0:9) qui positionnent les bascules B (2 : 11) ont la signification suivante :

TEST	B	SIGNIFICATIONS
0	2	R = 1111111111111111
1	3	R = 0000000000000000
2	4	ADO = 11
3	5	ADO = OO
4	6	REPT
5	7	S ≠ 0
6	8	AD1 = 1111
7	9	AD2 = 1111
8	10	AD1 = 0000
9	11	AD2 = 0000

Les entrées E (1:8) quant à elles représentent les bascules AIG et poids faible de RAMM, plus 6 tests qui restent disponibles. Enfin, les entrées COMMANDE (16:23) ont la signification suivante :

COMMANDE	B	SIGNIFICATIONS
16	12	CONDI1 = (S ≠ 0)
17	13	CONDI2 = (S ≠ 0)
18	14	CONDI3 = (S ≠ 0)
19	15	CONDI4 = (S ≠ 0)
20	12	CONDI1 = (S = 0)
21	13	CONDI2 = (S = 0)
22	14	CONDI3 = (S = 0)
23	15	CONDI4 = (S = 0)

d) L'unité COMMAND

Elle reçoit en entrée MICRO (1 : 28) qui provient de MEMC, CC (1 : 2) et CINS (1 : 6) qui viennent de CODDE.

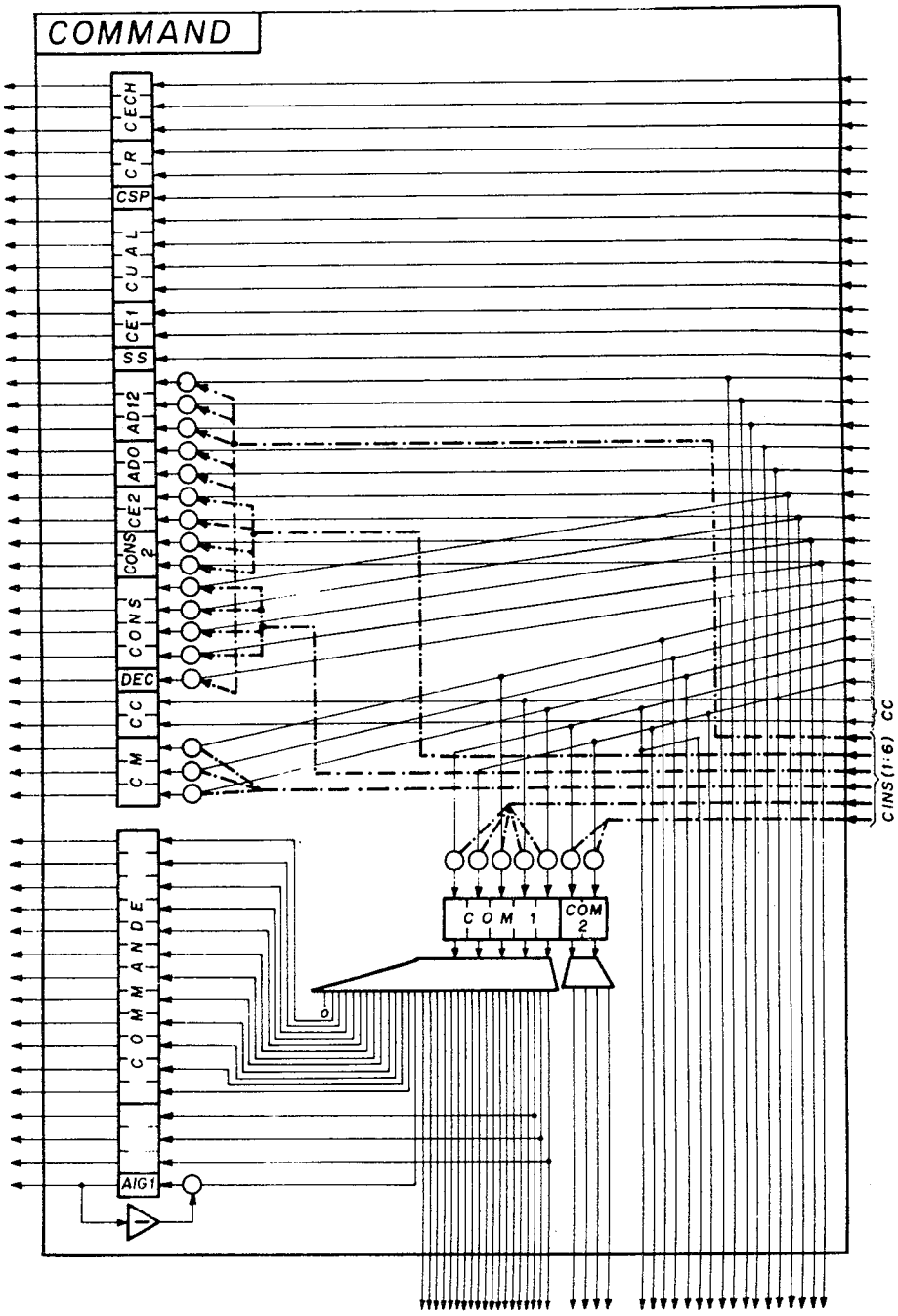
En sortie elle envoit CECH (1 : 3) à ECHANGE, CM (1 : 3) à MEM, CONTR (1 : 43) aux unités opérationnelles de ZOOO1, COMMANDE (16 : 23), COND1 (1 : 5) à TEST, C, COMMANDE (14 : 15) et ADRDEROUT à SEQUENCE.

Les sorties COM2 (1 : 4), COND2 (1 : 2) et COMMANDE (24 : 31) restent pour l'instant inutilisées.

MICRO (1 : 28) donne MICROCOM (1 : 51), COND, COND2, ADRDEROUT, après aiguillage des différents champs vers les sous-registres convenables du registre MICROCOM.

```
FILE: DESCRIPT CASSPGR P1                          CAMBRIDGE MONITOR SYSTEM

'UNITE' COMMAND(13,H,MICRO(1:28),CC(1:2),CINS(1:6);
        AIG,COMMANDE(14:31),CONTR(1:43),C,CECH(1:3),CM(1:3),
        COND1(1:5),COM2(1:4),COND2(1:2),ADRDROUT(1:9));
  'HORLOGE' H,13;
  'SIGNAL' Z(0:51);
  'REGISTRE' RAIG,MICROCOM(1:54),CR(1:2),CSP,COAL(1:4),CE1(1:2),
             SS,AD12(1:5),ADC(1:2),CE2(1:2),CONS2(1:2),CONS(1:4),DEC;
          MICROCOM(4:26)<=>CR&CSP&COAL&CE1&SS&AD12&ADC&CE2&
          CONS2&CONS&DEC;
          CONTR:=MICROCOM(1)&MICROCOM(4:30)&Z(1:15);
          COM2(&MICROCOM(50:51)):=1;
          Z(&(MICROCOM(45:49)):=1;
          COND1:=MICRO(27:28)&MICRO(24:26);
          COND2:=MICRO(27:28);
          C:=MICRO(23);
          ADRDROUT:=MICRO(14:22);
          CECH:=MICROCOM(1:3);
          CM:=MICROCOM(31:33);
          AIG:=RAIG;
    " COMMANDE(1:15) EST MICROCOM(33:47) "
          COMMANDE(16:31):=Z(16:31);
(MICROCOM(14:18)<=MICRO(14:18),MICROCOM(27)<=MICRO(23))
(MICROCOM(19:22)<=MICRO(19:22))
(MICROCOM(23:26)<=MICRO(19:22))
(MICROCOM(30:32)<=MICRO(24:26))
(MICROCOM(45:49)<=MICRO(27:28)&MICRO(24:26))
(MICROCOM(50:51)<=MICRO(27:28))
'SINON'
(MICROCOM(14:18)<=00000,MICROCOM(27)<=0)
(MICROCOM(19:22)<=0000)
(MICROCOM(23:26)<=0000)
(MICROCOM(30:32)<=000)
(MICROCOM(45:49)<=00000)
(MICROCOM(50:51)<=00)
,MICROCOM(1:13)<=MICRO(1:13),MICROCOM(28:29)<=CC,
MICROCOM(33:47)<=Z(1:12)&Z(29:31);
<13> 'SI' Z(13) 'ALORS' RAIG<=-RAIG;
```

Pour passer d'un mot de 28 bits à un mot de 88 bits il faut à l'aide des conditions CINS afficher des 0 dans les différents champs de MICROCOM dont les conditions d'entrées ne sont pas vérifiées. Les champs concernés sont AD12, ADO, CE2, CONS2, CONS, DEC, CM, COM1 et COM2. Comme on pourra le vérifier par la suite, pour aucun de ces champs le code qui n'est constitué que de 0 ne peut déclencher d'opérations. On peut ainsi "masquer" un champ , ce qui constitue le principe du "double décodage".



e) L'unité SEQUENCE

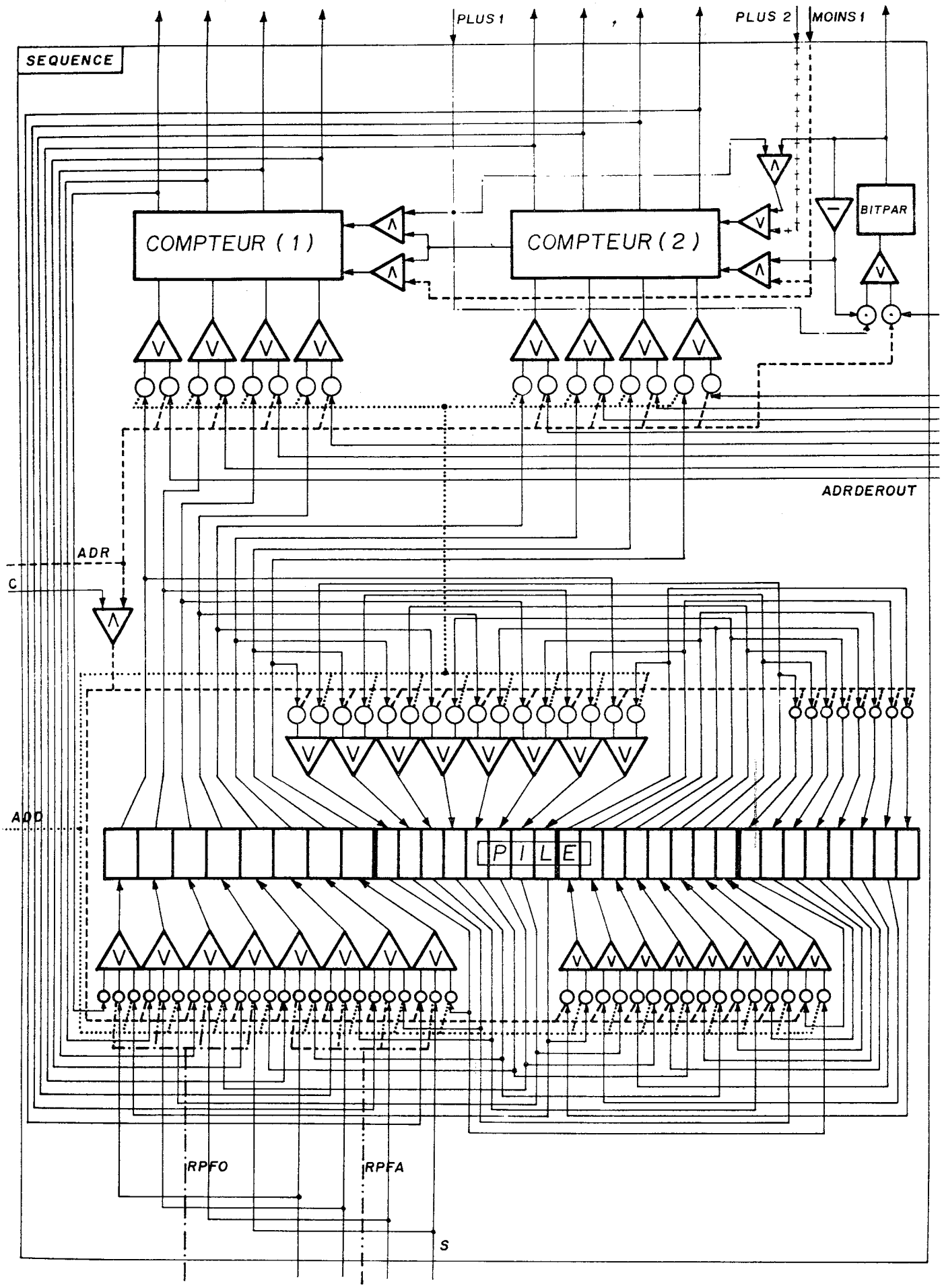
Elle contient le registre d'adresse de MEMC formé de 2 circuits compteur-décompteur (unité COMPTEUR) auxquels on adjoint une bascule BITPAR de poids faible. Elle contient aussi une PILE de 4 octets, dont le sommet peut recevoir une adresse de retour (on empile l'adresse courante), les 4 bits de sortie de l'unité UAL qui peuvent se positionner soit en poids faibles, soit en poids forts, enfin une adresse empilée et que l'on dépile (retour de sous-microprogramme).

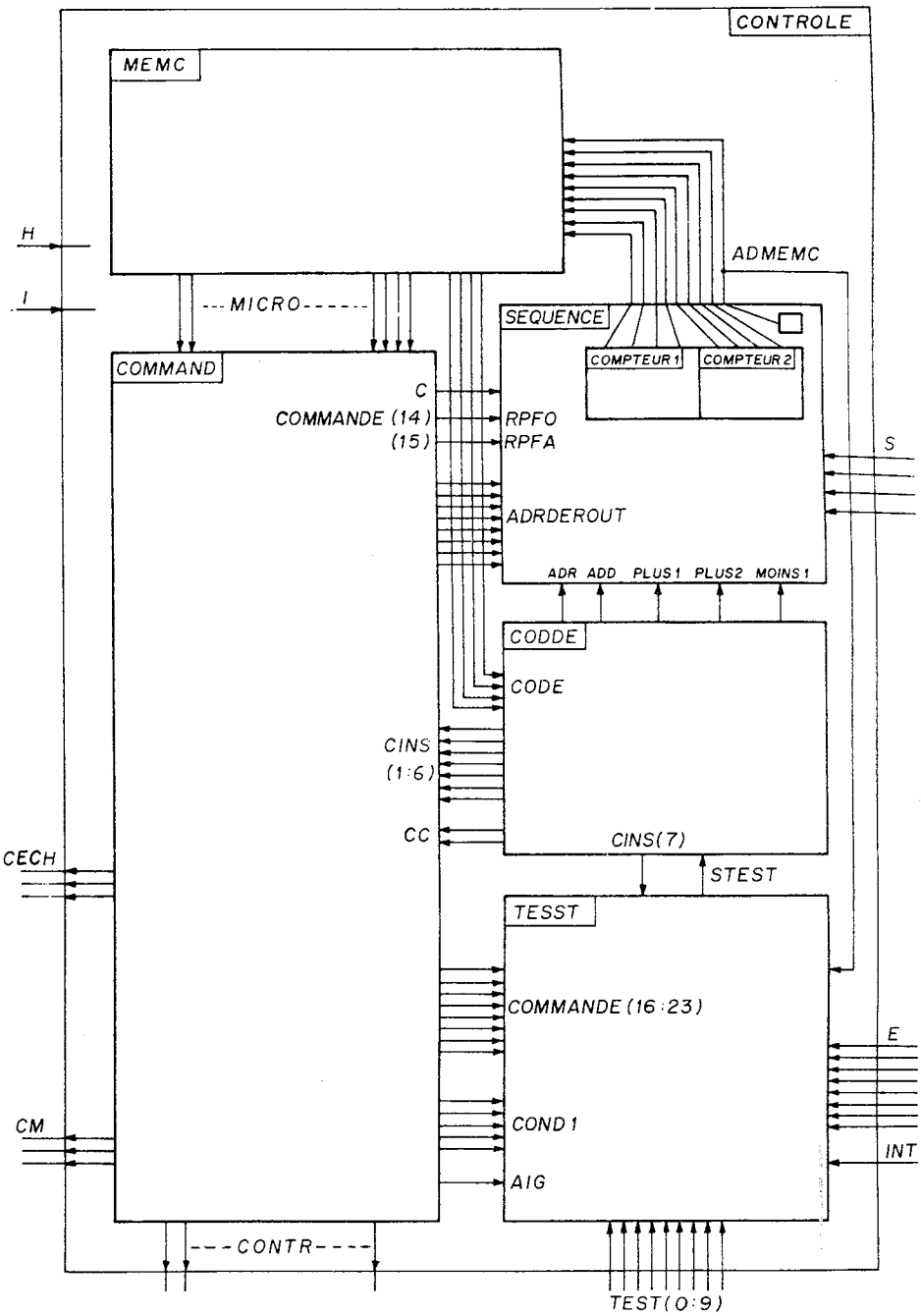
Le registre d'adresse de MEMC est cablé pour effectuer sous l'action des signaux PLUS1, MOINS1, PLUS 2, ADR, C, ADD respectivement l'incrémentation de son contenu par +1, -1, +2, le stockage de l'entrée ADRDEROUT (avec empilement, si C, de l'adresse courante) le stockage du sommet de pile (signal ADD).

On peut remarquer que les retours de sous-microprogrammes ne peuvent s'effectuer qu'à des adresses de même parité que l'adresse en cours.

Les signaux RPFO et RPFA permettant le stockage en poids forts ou en poids faibles de la sortie S de UAL. Ceci fait gagner beaucoup de temps dans la phrase d'interprétation d'un code instruction. Le décodage consist alors seulement en un branchement dans le microprogramme (CF système 360).

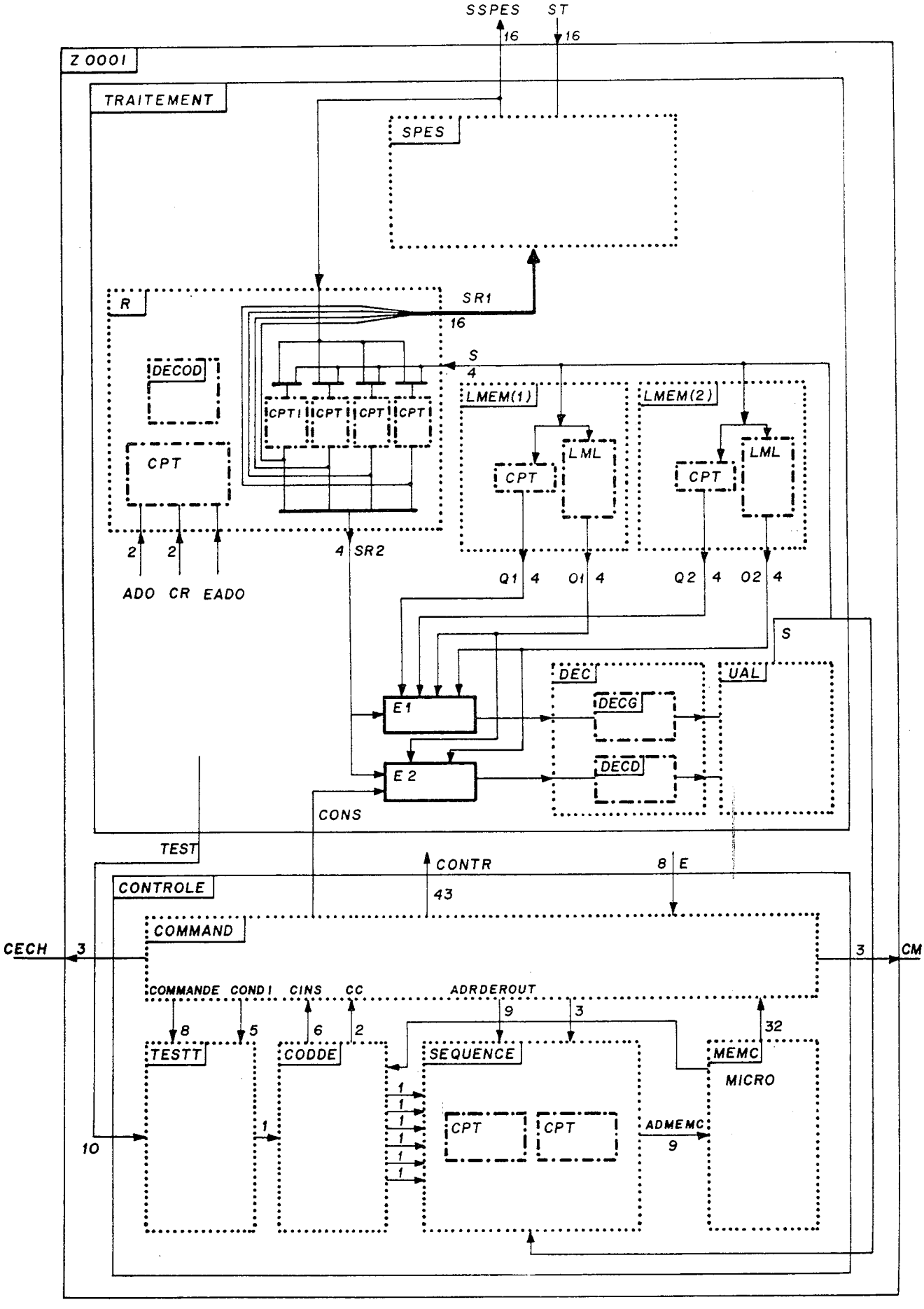
```
FILE: DESCRIPT.BASSPUR  P1                          CAMBRIDGE  MONITOR  SYSTEM

'UNITE' SEQUENCE(H,I1,S(1:4),ADRDEROUT(1:5),E1,E2,E3,E4,E5,E6,RPFO,RPFA;
        ADMEMC(1:9)),
'HORLOGE' I1,H;
'REGISTRE' PILE(1:3,0:3),BITPAR,PLUS1,MOINS1,PLUS2,ADR,C,ADD;
'EXTERNE' COMPTEUR('HORLOGE',3,,,,,;3,);
'SIGNAL' TC1,P(1:9),P2(1:4);
      'SI' ADD&ADR&PLUS1 'ALORS'
         (P(1:8):=PILE(,0))
         (P:=ADRDEROUT)
         (P(9):=-BITPAR);
      COMPTEUR1(I1,P(1:4),ADD+ADR,PLUS1.TC1,MOINS1.TC1,;ADMEMC(1:4),);
      COMPTEUR2(I1,P(5:8),ADD+ADR,PLUS2+PLUS1.BITPAR,MOINS1.-BITPAR,;
      ADMEMC(5:8),),
   <H> PLUS1<=E1,PLUS2<=E3,MOINS1<=E2,ADR<=E4,C<=E5,ADD<=E6;
   <I1> 'SI' RPFO&RPFA&ADD&(ADR.C) 'ALORS'
      (PILE(1:4,0)<=S)
      (PILE(5:8,0)<=S)
      (PILE(,0)<=PILE(,1))
      (PILE(,0)<=ADMEMC(1:8)),
      'SI' ADD&(ADR.C) 'ALORS'
         (PILE(,1)<=PILE(,2),PILE(,2)<=PILE(,3))
         (PILE(,1)<=PILE(,0),PILE(,2)<=PILE(,1),PILE(,3)<=PILE(,2)),
      'SI' ADR+PLUS1 'ALORS' BITPAR<=P(9);
```


PLUS 1
PLUS 2
MOINS 1
SEQUENCE
COMPTEUR (1)
COMPTEUR (2)
BITPAR
ADRDEROUT
ADR
C
ADD
PILE
RPFO
RPFA
S



f) L'unité CONTROLE

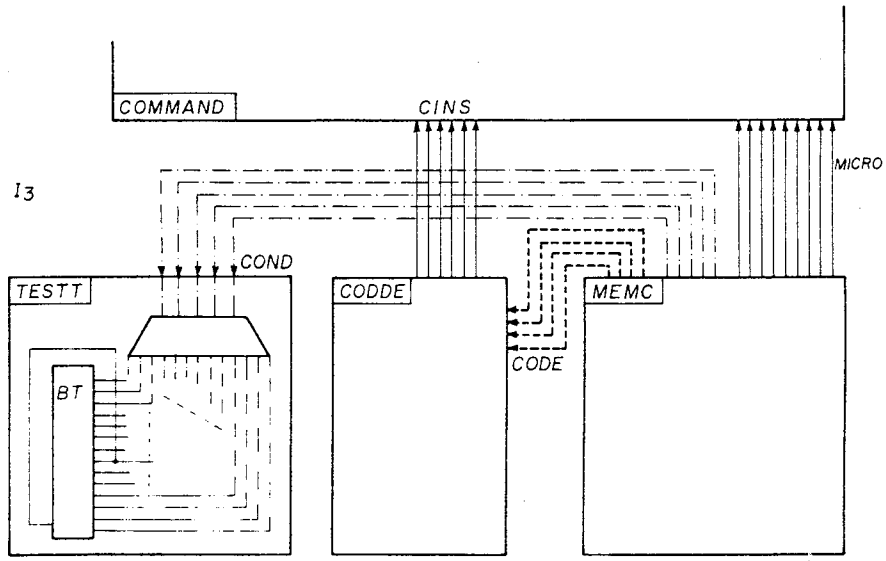
```
'UNITE' CONTROLE(H,I1,I3,S(1:4),TEST(0:9),INT,E(1:8);
        CONTR(1:43),CM(1:3),CECH(1:3));
'SIGNAL' ADMEMC(1:9),MICRO(1:32),STEST,CC(1:2),CINS(1:8),PLUS1,MOINS1,
         PLUS2,ADD,ADR,ADRDEROUT(1:9),C,RPFO,RPFA,COMMANDE(1:31),
         COND(1:5),AIG;
'EXTERNE' MEMC(9;32),
          CODDE(4;,,,,,,,2,8),
          COMMAND('HORLOGE','HORLOGE',28,2,6;,18,43,,3,3,5,4,2,9),
          TESTT('HORLOGE','HORLOGE',8,,,8,10,5;),
          SEQUENCE('HORLOGE','HORLOGE',4,9,,,,,,,,,;9);
'HORLOGE' H,I3,I1;
      MEMC(ADMEMC,MICRO);
      CODDE(MICRO(29:32),STEST;PLUS1,MOINS1,PLUS2,ADD,ADR,CC,CINS);
      SEQUENCE(H,I1,S,ADRDEROUT,PLUS1,MOINS1,PLUS2,ADR,C,ADD,RPFO,RPFA;
      ADMEMC);
      TESTT(I3,H,COMMANDE(16:23),INT,CINS(7) ,E,TEST,COND;STEST);
      COMMAND(I3,H,MICRO(1:28),CC,CINS(1:6);AIG,COMMANDE(14:31),CONTR,C,CECH
      CM,COND,,,ADRDEROUT(1:9));
```


SSPES
ST
16
16
Z 0001
TRAITEMENT
SPES
R
SR1
16
S
4
DECOD
CPT
CPT1
CPT
CPT
CPT
LMEM(1)
LMEM(2)
CPT
LML
CPT
LML
2
2
4
SR2
ADO
CR
EADO
Q1
4
01
4
Q2
4
02
4
S
DEC
DECG
DECD
UAL
E1
E2
CONS
TEST
CONTR
43
8
E
CONTROLE
COMMAND
CECH
3
3
CM
COMMANDE
CONDI
CINS
CC
ADRDEROUT
8
5
6
2
9
3
32
TESTT
CODDE
SEQUENCE
MEMC
MICRO
1
1
1
1
1
1
10
1
CPT
CPT
ADMEMC
9

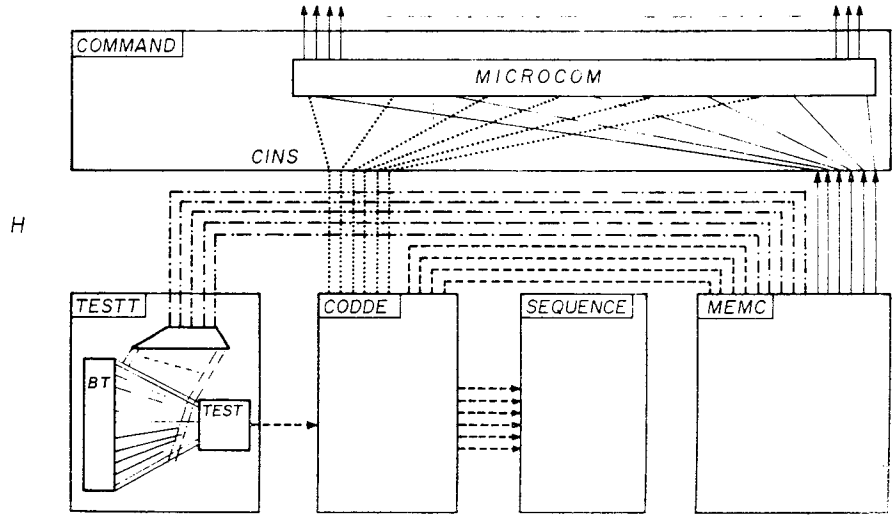
g) L'unité HORLOGE, la synchronisation de Z0001

Nous avons déjà dit que Z0001 est synchronisé par une impulsion I et une autre impulsion H quatre fois moins fréquente. Appelons Io,I1, I2, I3 les 4 impulsions de I qui se produisent entre 2 impulsions de H, Io coincide avec H.

1) Le mot MICRO (1 : 32) issu de la mémoire de contrôle MEMC est supposé disponible au top I3, alors commence immédiatement le décodage des champs CODE et COND.



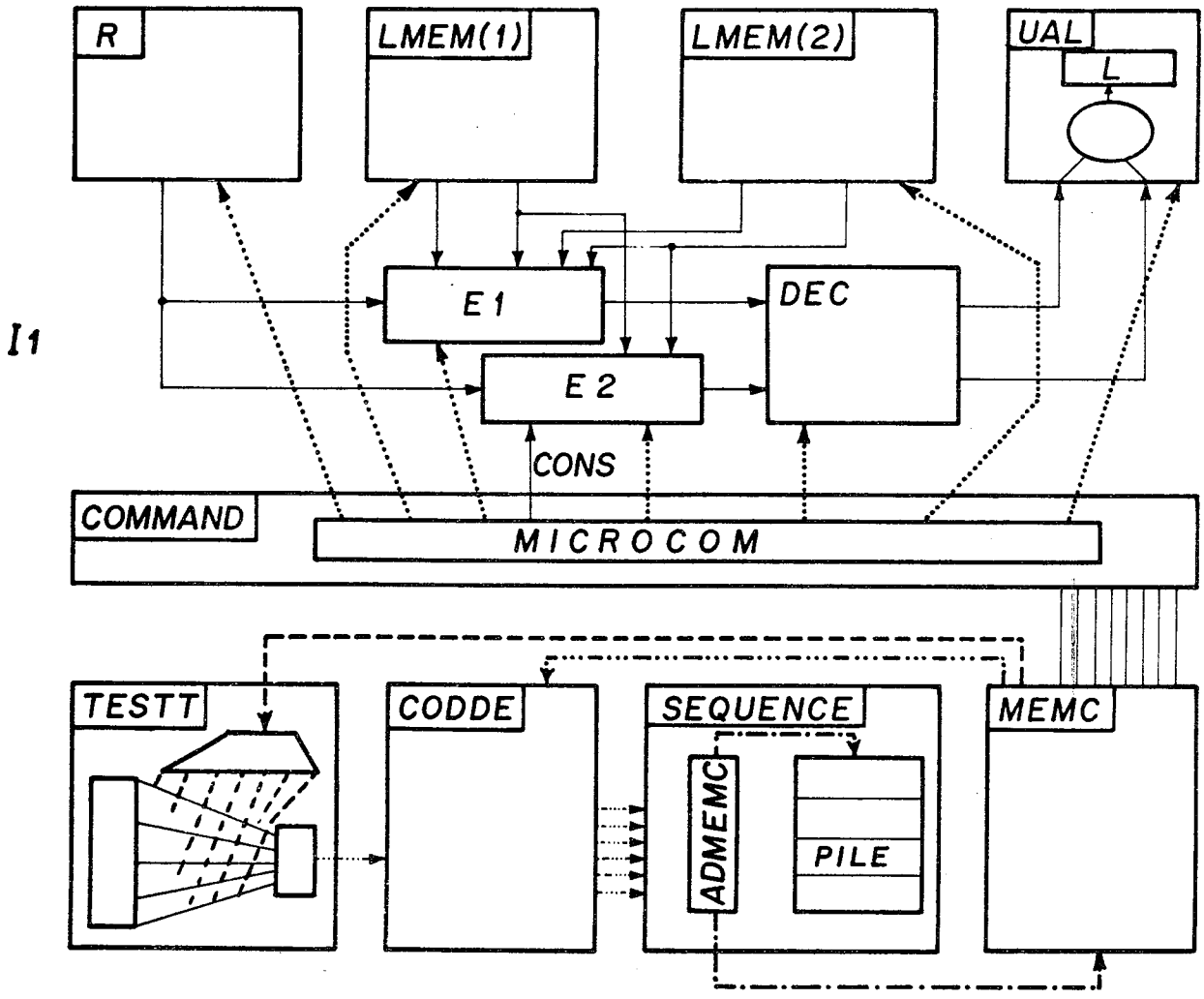
2) Le top suivant Io coincide avec H. On s'en sert pour charger le mot MICROCOM de COMMAND, la bascule TESST de TESTT. Le décodage de tous les champs de MICROCOM, en particulier le champ COM1, commence. Ces commandes décodées vont,dans les unités opératoires de TRAITEMENT, permettre l'exécution des opérations logiques et de transfert.



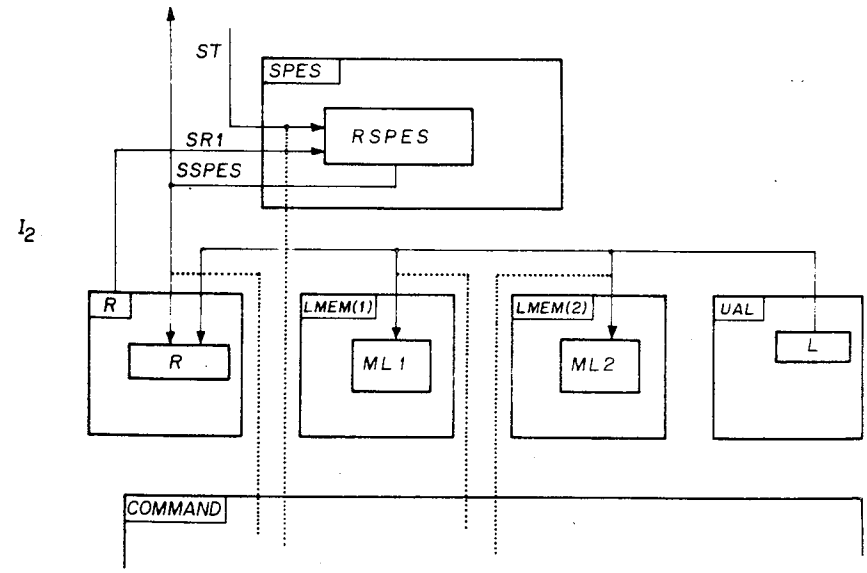
3) Au top I1 on supposera disponible la sortie S de l'unité UAL (pour simplifier. Dans la réalité on retarde le top I1, pour de plus de sûreté).

On charge alors L registre (Latch) de sortie de UAL. Mais SEQUENCE a en entrée les signaux qu'il faut pour donner au registre l'adresse de MEMC sa prochaine valeur. On charge donc ce registre en lançant la lecture de la microinstruction suivante.

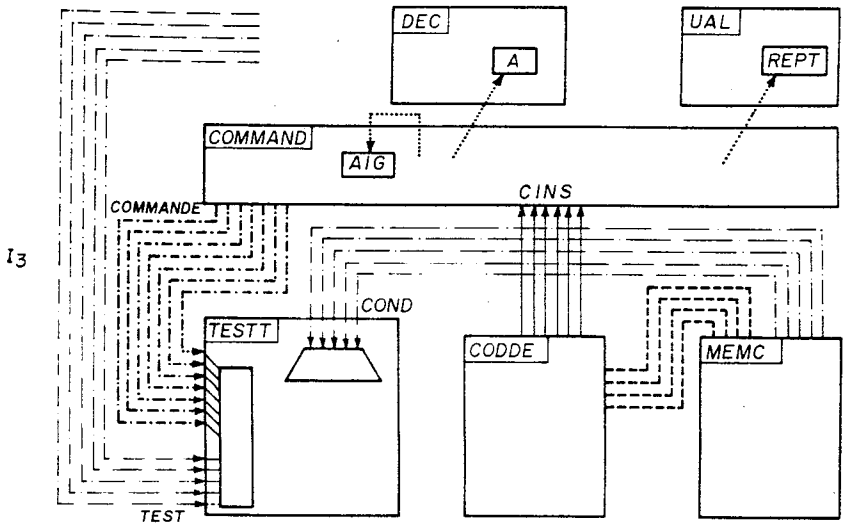
On charge également PILE s'il y a lieu.



4) Au top I2, la lecture de la microinstruction suivante se poursuit mais dans la microinstruction en cours les opérations de la partie TRAI TRAITEMENT doivent être terminées. On charge donc à l'aide de cette impulsion les registres M |1| et M |2| de ML |1| et ML |2| le registre R (c'est à dire COMPTEUR |1| , |2| , |3| , |4| de R) le registre RSPES de SPES.



5) Au top I3 la nouvelle instruction MICRO va être disponible. Les signaux COMMANDE et TEST sont prêts, on charge donc les 16 bascules testables de TESTT, on charge les bascules AIG, REPT (de UAL) et A(1:2) de DEC. On met aussi à jour tous les compteurs d'adresse AD0, CT(1) de LMEM |1| CT(2), LMEM |2| et AD3. On est revenu à 1), le cycle recommence avec H au top suivant.




```
'UNITE' HORLOGE(H,I,I1,I2,I3);
  'HORLOGE'H,I,I1,I2,I3;
  'REGISTRE' PHASE(3:1);
     <H>    PHASE<=01;
     <I>    'SI' $PHASE 'ALORS'
               ()
               (I1:=1,PHASE<=10)
               (I2:=1,PHASE<=11)
               (I3:=1,PHASE<=00);
```

h) Récapitulation sur Z0001

Z0001 est composé d'une partie opératoire TRAITEMENT et d'une partie CONTROLE.

L'unité TRAITEMENT en contient 6 autres.

- SPES, ensemble de 4 registres de 16 bits, dont on peut lire l'un pendant qu'on écrit dans un autre : ces deux registres sélectés sont adressés par AD3

- R, ensemble de 4 registres de 4 bits adressés par AD0. On peut lire ou écrire les 16 bits en parallèle, en liaison avec SPES. On peut aussi envoyer chacun des registres de 4 bits dans les bus E1 ou E2 et les charger à partir de S.

- LMEM, 2 ensembles LMEM|1| et LMEM|2| de 16 mémoires de 4 bits adressées par AD1 et AD2, on peut comme R les charger par S ou les envoyer dans AD1 ou AD2.

- DEC, reçoit en entrée les bus E1 et E2 et peut leur faire subir des décalages respectivement à gauche et à droite, le nombre de positions de décalage est indiqué par un registre A(1:2).

- UAL, reçoit OP1 et OP2 qui sont E1 et E2 après la traversée de DEC, peut alors appliquer 16 fonctions logiques ou arithmétiques.

- Les registres d'adresse AD0, AD1, AD2 peuvent être chargés en parallèle, être incrémentés ou décrémentés de 1.

L'unité CONTROLE en contient 5 autres :

MEMC, mémoire qui contient les microprogrammes, TESTT qui examine certains signaux caractéristiques de TRAITEMENT, CODDE qui décode la microinstruction, COMMAND qui valide les microactions élémentaires et SEQUENCE, qui assure en fonction des indications de CODDE et TESTT l'enchainement du microprogramme.

III - La microprogrammation de Z0001

a) Les champs et leur codage

On a pu voir que le séquencement de Z0001 est assuré par le seul automate SEQUENCE, à partir de l'état préanalysé par TESTT de certaines bascules des parties TRAITEMENT ou CONTROLE et en fonction des commandes interprétées par CODDE qui dictent la progression de l'adresse de MEMC.

Nous appelerons désormais ADMEMC, le compteur de microinstructions constitué par 2 unités COMPTEUR et le bit BITPAR de SEQUENCE.

En général ADMEMC progresse par +1, s'il n'y a pas de test ou si le résultat de ce test est faux (c.à.d. STEST = 0).

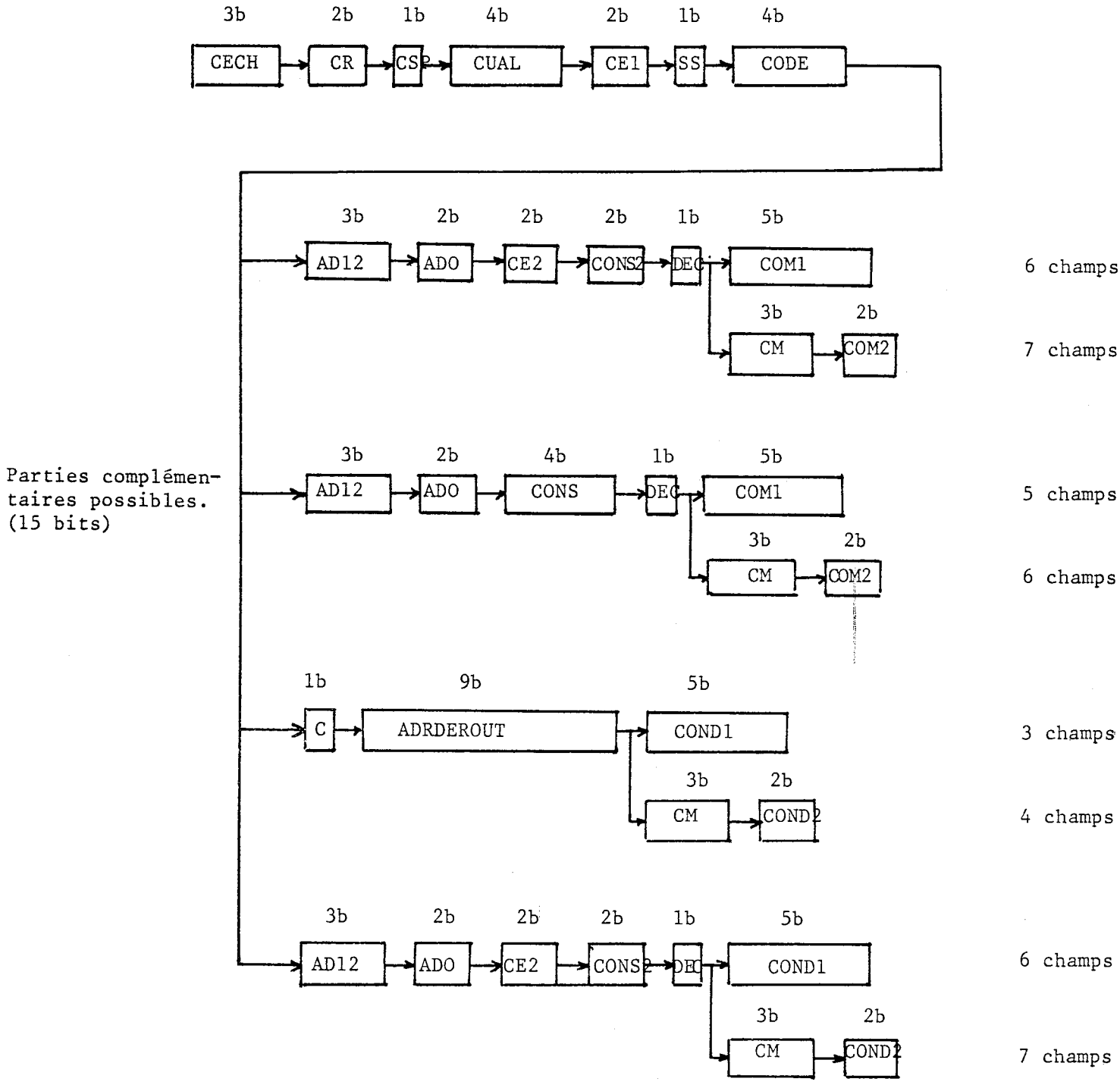
Sinon 6 cas sont possibles :

1) on charge dans ADMEMC l'adresse ADRDEROUT (1:9), prise dans MICRO.
2) on fait de même, mais au préalable on sauvegarde l'adresse qui était dans ADMEMC dans PILE.
3) on charge dans ADMEMC l'adresse contenue au sommet de PILE.
4) on fait progresser par +2 ADMEMC.
5) on fait progresser par -1 ADMEMC.
6) on ne fait pas progresser ADMEMC.
   (la microinstruction en cours se répète jusqu'à ce que le TEST devienne faux).

Une partie des commandes contenues dans la microinstruction MICRO (1:32) est donc destinée à la recherche de la microinstruction suivante. Il s'agit des champs COND (1:5), COND2 (1:2), COM1 (1:5), COM2 (1:2), C, ADRDEROUT (1:9) et surtout CODE (1:4). Le reste des commandes agit sur la partie opératoire : les champs CECH (1:3), CR (1:2), CSP, CUAL (1:4), CE1 (1:2), SS, AD12 (1:3), ADO (1:2) CE2 (1:2), CONS2 (1:2) CONS (1:4), DEC, CM (1:3).

b) Le mot de microinstruction peut donc avoir les formats suivants :

Partie immuable du microinstruction (17 bits 7 champs).



La microinstruction a sept champs invariables et suivant 8 formats possibles de 3 à 7 champs complémentaires.

Il s'agit bien, on le voit, de microprogrammation à mot long (ce qualificatif n'étant pas relatif au nombre de bits de la microinstruction, mais le nombre de ses champs indépendants lorsqu'ils avoisinent la dizaine).

Nous allons décrire en CASSANDRE les actions commandées par ces différents champs en expliquant en commentaire la signification de chaque action.

1) Le champ CECH et la bascule CSP

Il agit sur les unités SPES et ECHANGE

```
BDANS R4  : CECH :=000 ; "le bus B de ECHANGE va dans R4 "
BDANS R5  : CECH :=001 ; "    "          "          "   R5 "
BDANS R6  : CECH :=010 ; "    "          "          "   R6
BDANS R7  : CECH :=011 ; "    "          "          "   R7

R1 DANS B : CECH :=100 ; "R1 est branché sur l'entrée du bus B"
R2 DANS B : CECH :=101 ; "R2  "          "          "          "
R3 DANS B : CECH :=110 ; "R3  "          "          "          "
R9 DANS B : CECH :=111 ; "R9  "          "          "          "
```

Dans le cas des 4 premières actions il est clair qu'en entrée c'est ET (1:16) (encore appelé SSPES) qui va dans B. Dans les 4 suivantes il est non moins clair que la sortie validée de B est ST qui va dans SPES, il y a donc alors incompatibilité avec le chargement de R dans SPES par SR1. D'où pour le champ CSP :

```
SORSPES : CSP : = 0 ; "si CECH(1) vaut 0 la sortie de SPES va dans
                       B sinon il ne se passe rien".

ENTRSPES : CSP : = 1 ; "si CECH (1) vaut 0 l'entrée de SPES reçoit
                        R (et sa sortie va sur B)".
                        sinon si CECH (1) = 1, l'entrée de SPES reçoit B".
```

On voit que les champs CECH et CSP ne sont pas indépendants.

2) Le_champ_CR

CRIEN :	CR : = 00 ;	"pas d'opération"
SPE 4DANR :	CR : = 01 ;	"R reçoit 4 bits parmi les 16 qui viennent de SPES".
SPESDANR :	CR : = 10 ;	"Les 16 bits de SPES sont chargés dans R".
SDANR :	CR : = 11 ;	"la sortie S de UAL est chargée dans R".

3) Le_champ_CUAL

ADDREPT :	CUAL : = 0000 ;	"UAL effectue OP1 + OP2 + REPT"
ADD :	CUAL : = 0001 ;	" " OP1 + OP2 " "
SOUSREPT :	CUAL : = 0010 ;	" " OP1 - OP2 - REPT"
SOUS :	CUAL : = 0011 ;	" " OP1 - OP2 "
ABARB :	CUAL : = 0100 ;	" " -OP1 OP2 "
OP2 :	CUAL : = 0101 ;	" " OP2 "
DISJ :	CUAL : = 0110 ;	" " OP1 ≠ OP2 "
OU :	CUAL : = 0111 ;	" " OP1 V OP2 "
NI :	CUAL : = 1000 ;	" " -(OP1 V OP2) "
CONJ :	CUAL : = 1001 ;	" " OP1 = OP2 "
OP2BAR :	CUAL : = 1010 ;	" " -OP2 "
A ou BBAR :	CUAL : = 1011 ;	" " OP1V - OP2"
ZERO :	CUAL : = 1100 ;	" " 0000
INTER :	CUAL : = 1101 ;	" " OP1 ∧ OP2 "
ABBAR :	CUAL : = 1110 ;	" " OP1∧- OP2 "
OP1 :	CUAL : = 1111 ;	" " OP1 "

4) Le_champ_CE1 : si la bascule AIG = 0

RDANE1 : CE1 : = 00 ; "le bus E1 reçoit la sortie SR2 de R"
MDANE1 : CE1 : = 01 ; " " de LMEM |1| "
MBDANE1 : CE1 : = 10 ; " " de LMEM |2| "
ADDANE1 : CE1 : = 11 ; " " AD1 du compteur d'adresse de LMEM |1|"

si la bascule AIG = 1

RDANE1 signifie "le bus E1 reçoit la sortie SR2 de R "
MDANE1 " "le " " de LMEM |2| "
MBDANE1 " " " " de LMEM |1| "
ADDANE1 " " " " AD2 du compteur d'adresse de LMEM |2| "

5) Le_champ_SS

SDANAD : SS : = 0 ;"valable si SDANAD12 sinon pas d'opération "
SDANM : SS : = 1 ; "LMEM |1| reçoit S si AIG = 0 sinon LMEM |2| reçoit S "

6) Le_champ_AD12

ADRIEN : AD12 : = 000 ; "pas d'opération"
AD12PS1 : AD12 : = 001 ; "les compteurs d'adresse de LMEM |1| et LMEM |2| sont tous deux incrémentés de 1"
ADPS1 : AD12 : = 010 ; "Si AIG = 0 l'adresse de LMEM |1| est augmentée de 1 sinon l'adresse de LMEM |2|"
ADBPS1 : AD12 : = 011 ; "si AIG = 0 l'adresse de LMEM |2| est augmentée de 1 sinon celle de LMEM |1|"
SDANAD12 :AD12 : = 100 ; "le compteur d'adresse de LMEM |1| ou LMEM |2| reçoit S suivant que SS vaut AIG ou -AIG"
AD12MS1 : AD12 : = 101 ; "les compteurs d'adresse de LMEM |1| et LMEM |2| sont tous deux décrémentés de 1"
ADMS1 : AD12 : = 110 ; "si AIG = 0 l'adresse de LMEM |1| est décrémentée de 1 sinon celle de LMEM |2|"
ADBMS1 : AD12 : = 111 ; "si AIG = 0 l'adresse de LMEM |2| est décrémentée de 1 sinon celle de LMEM |1|.

7) Le champ CE2

CSE2 : CE2 : = 00 ; "si CSDANE2 est vrai alors le bus E2 reçoit CONS sinon pas d'opération"

RDANE2 : CE2 : = 01 ; "le bus E2 reçoit la sortie SR2 de R"

MDANE2 : CE2 : = 10 ; "si AIG = 0, E2 reçoit la sortie de LMEM |1| sinon celle de LMEM |2|"

MBDANE2 : CE2 : = 11 ; "si AIG = 0, E2 reçoit la sortie de LMEM |2| sinon celle de LMEM |1|"

On peut remarquer le rôle de la bascule AIG dans les 4 champs qui précèdent. Son inversion permet la permutation de LMEM |1| et LMEM |2|.

8) Le champ ADO

ADORIEN : ADO : = 00 ; "pas d'opération"

ADOPS1 : ADO : = 11 ; "ADO, l'adresse de R est incrémentée de 1"

ADOMS1 : ADO : = 01 ; " " " décrémentée de 1"

CONS2ADO : ADO : = 10 ; "le champ CONS2 est chargé dans ADO"

9) Les champs C et ADRDEROUT

DEROUT : C : = 0 ; ADRDEROUT : = ADR ; "le microprogramme est branché à l'adresse ADR de 9 bits"

CDEROUT : C : = 1 ; ADRDEROUT : = ADR ; "même opération que précédemment mais l'adresse courante est empilée"

10) Les champs CONS et CONS2

Ces champs représentent respectivement une constante de 4 bits et une constante de 2 bits que l'on peut injecter dans le chemin de données de l'unité de traitement (voir les descriptions précédentes).

11) Le bit DEC

DECGO : DEC : = 1 ; "validation du mode décalage gauche"

DECDR : DEC : = 0 ; " " " droit"

12) Le champ CM

CMRIEN1 : CM : = 000 ; "pas d'opération"

CMLEC : CM : = 001 ; 'lancement de lecture dans MEMOIRE"

CMAD1 : CM : = 010 ; "chargement d'une adresse de MEMOIRE en provenance de Z0001"

CMAD2 : CM : = 011 ; "chargement d'une adresse de MEMOIRE en provenance des entrées-sorties"

CMMOT1 : CM : = 100 ; "chargement d'un mot dans le registre tampon de MEMOIRE en provenance de Z0001"

CMMOT2 : CM : = 101 ; "idem, en provenance des entrées-sorties"

CMECR : CM : = 110 ; "lancement d'une écriture dans MEMOIRE"

CMRIEN2 : CM : = 111 ; "pas d'opération"

13) Les champs COM1 et COM2

Ils représentent tous deux respectivement sur 5 et 2 bits le codage de commandes que l'on a décidé non simultanées.

Nous n'utiliserons pas COM2 ici.

COMRIEN : COM1 : = 00000 ; "pas d'opération"

- RDANSA : COM1 : = 00001 ; "les bits 13 et 14 de SR1 sont chargés dans A"
- ZEDANSA : COM1 : = 00010 ; "00, est chargé dans A"
- UNDANSA : COM1 : = 00011 ; "01, " " "
- DEUDANSA : COM1 : = 00100 ; "10, " " "
- SDANSA : COM1 : = 00101 ; " les 4 bits de S qui ne doivent contenir qu'un seul 1 sont encodés et chargés dans A"
- UNDSREPT : COM1 : = 00110 ; "le report de UAL est mis à 1"
- ZEDSREPT : COM1 : = 00111 ; " " " " 0"
- RPS1 : COM1 : = 01000 ; "incrémentation de R"
- RMS1 : COM1 : = 01001 ; "décrémentation de R"
- ZEDANSR : COM1 : = 01010 ; "tous les bits de R sont remis à zéro"
- ZEDOADO : COM1 : = 01011 ; " " ADO " "
- ZEDOAD12 : COM1 : = 01100 ; " " AD1 et AD2 " "

- COMAIG : COM1 : = 01101 ; "inversion de la bascule AIG"
  RPFA : COM1 : = 01110 ; "SR2 est chargé en poids faibles de PILE"
  RPFO : COM1 : = 01111 ; "SR2 est chargé en poids forts de PILE"
  CONDI1 : COM1 : = 10000 ; "la bascule CONDI1 reçoit S ≠ 0000"
  CONDI2 : COM1 : = 10001 ; " " CONDI2 " "
  CONDI3 COM1 : = 10010 ; " " CONDI3 " "
  CONDI4 : COM1 : = 10011 ; " " CONDI4 " "
  NCONDI1 : COM1 : = 10100 ; "la bascule CONDI1 reçoit S = 0000"
  NCONDI2 : COM1 : = 10101 ; " " CONDI2 " "
  NCONDI3 : COM1 : = 10110 ; " " CONDI3 " "
  NCONDI4 : COM1 : = 10111 ; " " CONDI4 " "

Les 8 dernières commandes restent disponibles. L'une RAZM servira à remettre à 0, LMEM |1| et RAZMB, LMEM |2|. FDECALG, FDECALD et FPASDECAL servent à forcer DEC dans l'état que l'on veut lui donner.

14) Les champs COND1 et COND2

Ils représentent respectivement sur 5 et 2 bits le codage d'un certain nombre de conditions que l'on peut tester afin d'effectuer des branchements conditionnels dans le microprogramme. COND2 ne sera pas utilisé ici.

VRAI : COND1 : = 00000 ; "condition toujours vraie"
TESTAIG : COND1 : = 00001 ; "test de la bascule AIG"
BITPAR : COND1 : = 11111 ; "test du poids faible de l'adresse de la microinstruction"
TREPT : COND1 : = 01001 ; "test de la bascule REPT ≠ 0"
TRZERO : COND1 : = 01010 ; " " " R ≠ 00...0"
TRUN : COND1 : = 01011 ; " " R ≠ 11...1"
TADOZERO : COND1 : = 01100 ; " " " ADO ≠ 00"
TADOUN : COND1 : = 01101 ; " " " ADO ≠ 11"
TSNZERO : COND1 : = 01110 ; "test de la bascule S ≠ 0000"
TAD1UN : COND1 : = 01111 ; " " " AD1 = 1111"
TAD2UN : COND1 : = 10000 ; " " " AD2 = 1111"
TAD1ZERO : COND1 : = 10001 ; " " " AD1 = 0000"
TAD2ZERO : COND1 : = 10010 ; " " " AD2 = 0000"
TCONDI1 : COND1 : = 10011 ; " " " CONDI1"

```
TCONDI2 :     COND1 : = 10100 ; "test de la bascule  CONDI2"
TCONDI3 :     COND1 : = 10101 ; "test      "    "    CONDI3"
TCONDI4 :     COND1 : = 10110 ; "  "       "    "    CONDI4"
TINT :        COND1 : = 10111 ; "  "       "    "    INT"
TAD1NZERO :   COND1 : = 11000 ; "test du complément de la bascule AD1=0000'
TAD2ZERO :    COND1 : = 11001 ; "      "                "         AD2= ""
TNCONDI1 :    COND1 : = 11010 ; "      "                "         CONDI1"
TNCONDI2 :    COND1 : = 11011 ; "      "                "         CONDI2"
TNCONDI3 :    COND1 : = 11100 ; "      "                "         CONDI3"
TNCONDI4 :    COND1 : = 11101 ; "      "                "         CONDI4"
TNINT    :    COND1 : = 11110 ; "      "                "         INT"
TSZERO :      COND1 : = 00111 ; "      "                "       S ≠ 0000"
```

Les conditions COND1 : = 00011 à COND1 : = 01000, restent disponibles.

15) Le champ CODE (1:4)

```
SEKNOR :      CODE (2:4) : = 000 ; "l'adresse de microinstruction est
                                    incrémentée de 1"
CONSE2 :      CODE (2:4) : = 001 ; "l'adresse de microinstruction est
                                    incrémentée de 1 et on envoie CONS
                                    dans E2"
CONSAD3 :     CODE (2:4) : = 010 ; "comme ci-dessus mais on envoie CONS
                                    dans AD3"
BCADD :       CODE (2:4) : = 011 ; "branchement si TESST à l'adresse de
                                    déroutement contenu dans la microins-
                                    truction si C l'adresse courante est
                                    au préalable empilée"
BCADR :       CODE (2:4) : = 100 ; "branchement si TESST à une adresse
                                    de retour de sous microprogramme que
                                    l'on sort du sommet de PILE"
BCPLUS2 :     CODE (2:4) : = 101 ; "branchement si TESST à l'adresse
                                    courante +2"
BCMOINS1 :    CODE (2:4) : = 110 ; "     "          "              -1"
BCPLUSO :     CODE (2:4) : = 111 ; "     "          "              +0"
```

Dans les cinq derniers cas, si TESST est faux, l'adresse de microinstruction est incrémentée de 1 et le branchement conditionnel n'est pas exécuté.

Si CODE (1) vaut 0 la microinstruction contient le champ COM1 dans les 3 cas de branchement non conditionnel et COND1 dans les 5 cas de branchement conditionnel. Si CODE (1) vaut 1, les champs COM1 et COND1 sont respectivement remplacés par CM COM2 et CM COND2, il s'agit alors de microinstructions agissant sur l'unité MEMOIRE.

Comme nous l'avons vu certains champs dont nous venons de décrire les codages possibles empiètent les uns sur les autres.

"Une microinstruction est un ensemble de champs recouvrant les 32 bits de MICRO sans redondance".

b) CASSANDRE langage de microprogrammation

1) Ecriture symbolique d'une microinstruction du Z0001

Nous avons vu sur un exemple assez important comment CASSANDRE remplissait sa fonction de description du hardware.

Nous venons de voir comment, on pouvait aussi, sous forme de listes d'états décrire les tables de codage des champs des microinstructions, nous allons voir comment décrire toujours en CASSANDRE les microprogrammes eux-mêmes.

Ceux-ci vont être enregistrés dans la mémoire de contrôle MEMC dont nous rappelons la description : Unité MEMC (ADMM(1:9) ; MICRO (1:32)) ;
Registre MEMM (1:32, 0:511) ;
MICRO : = MEMM (, S ADMM) ;

Une première méthode consisterait à remplir MEMM avec la suite de bits, parfaitement définie maintenant, que constitue chaque microinstruction. C'est une méthode longue et fastidieuse, qui présente deux inconvénients majeurs : il est impossible d'écrire ainsi plusieurs microprogrammes sans un grand nombre d'erreurs et une fois écrits, ces microprogrammes sont illisibles. C'est d'ailleurs pour ces deux raisons que les microprogrammeurs ont réalisé des assembleurs de microprogrammes qui leur permettent d'écrire les microinstructions dans un langage symbolique d'assemblage et qui vérifient la correction formelle des microinstructions écrites ainsi.

Bien que non conçu pour cela CASSANDRE rempli aussi cette fonction, en permettant en plus de simuler l'exécution des microprogrammes et donc de s'assurer de leur correction fonctionnelle.

On arrive à ce résultat en écrivant les microprogrammes sous forme de liste d'états (1 état par microinstruction) et en employant l'ordre faire.

Exemple :

```
ADDITION : 'faire' BDANR4; 'faire ENTRSPES ; 'faire' SDANR ; 'faire' ADDREPT;
           'faire' MDANE1 ; 'faire' MBDANE2 ; 'faire' SDANAD ; 'faire' AD12PS1 ;
           'faire' ADOPS1 ; CONS2 : = 00 ; 'faire' CMLEC;
           'faire' SEKNOR ;
```

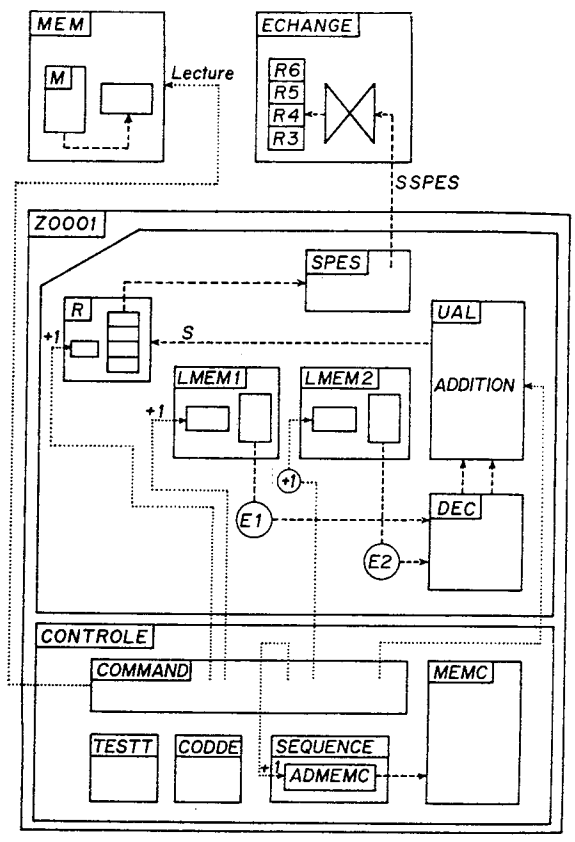
Cette instruction CASSANDRE produit à l'aide des tables de codage précédemment décrites :

	CECH	CR	CSP	CUAL	CE1	SS	CODE	AD12	ADO	CE2	CONS2	DEC	CM	CONS2
MICRO =	000	11	1	0000	01	0	0000	001	11	11	00	0	001	00

Les champs DEC et CONS2 qui n'étaient pas spécifiés sont mis automatiquement à 0, par contre si deux champs empiètaient l'un sur l'autre, définissant ainsi une microinstruction incorrecte, le simulateur CASSANDRE enverrait un message :

"chargement multiple de la variable MICRO"

Expliquons sur un schéma les actions commandées par la microinstruction précédente :



Bien sûr dans cette façon de décrire les microinstructions, l'unité SEQUENCE envoie bien à chaque cycle, une adresse à MEMC, mais cette adresse n'est pas utilisée pour sélecter la microinstruction suivante puisque cette dernière n'est plus rangée dans MEMM. Il faut alors décrire le séquencement d'une microinstruction à la suivante, on se sert des ordres 'allera' et dans l'exemple considéré on va les synchroniser par I3.

Cette conséquence n'est pas un inconvénient, au contraire. On peut en effet, vérifier à chaque cycle que l'adresse générée par SEQUENCE est bien celle que l'on désirait et on peut aussi mettre au point les microprogrammes avec des adresses symboliques et non des adresses absolues en binaire, comme ce serait obligatoire par tout autre procédé. On voit que l'on dispose bien à la fois d'un assembleur et d'un simulateur de microprogramme grâce aux ordres 'faire'.

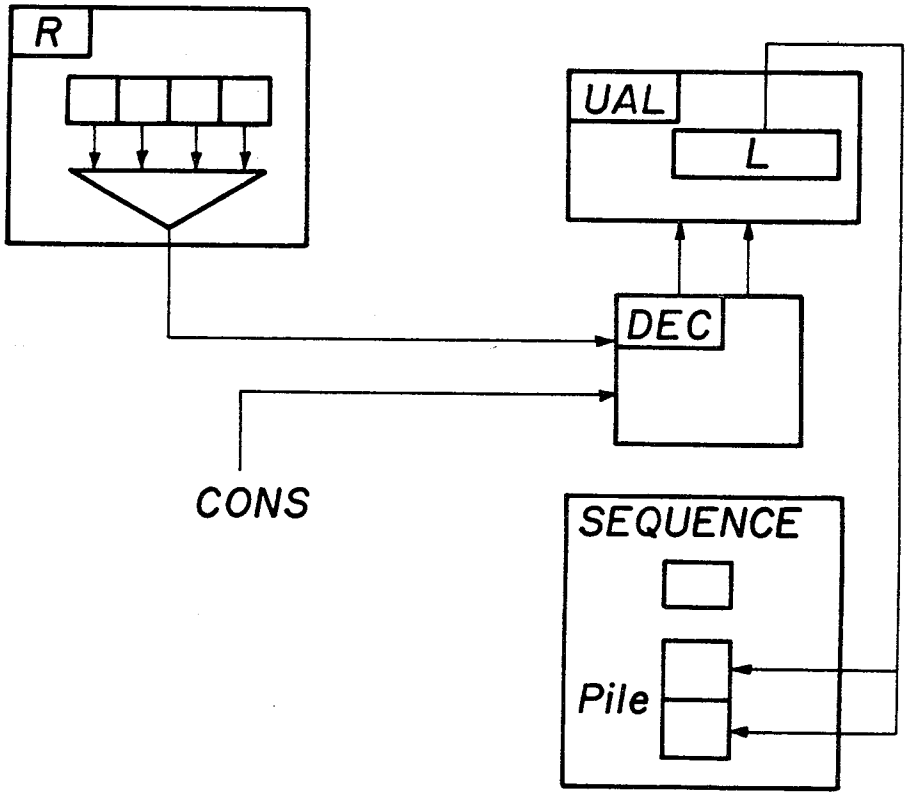
Lorsque tous les microprogrammes sont écrits il ne reste plus alors qu'à transformer les adresses symboliques en adresses absolues, en essayant de remplir de la façon la plus dense la mémoire morte. C'est l'opération de chargement.

2) Décodage d'une instruction

l'interprétation d'une instruction comporte deux phases consécutives le décodage et l'exécution.

Dans les machines microprogrammées l'une ou l'autre ou les deux phases peuvent être réalisées par microprogrammes. D'une façon générale, la phase décodage est commune à toutes les instructions bien qu'elle exécute des séquences différentes suivant les cas, la phase exécution est propre à chaque instruction. Lorsque les deux sont microprogrammées, la première phase se termine par un branchement à la première microinstruction du microprogramme d'exécution convenable.

Supposons une instruction dont le code opération ait 6 bits, suivi d'un bit indiquant s'il y a ou non indexation et d'un autre indiquant si l'adressage est indirect. Après lecture cette instruction se trouve dans R et ADO positionné sur 11.



D1 : 'faire' RDANE1 ; 'faire' CSE2 ; 'faire' OP1 ; 'faire ADOMS1 ;
'faire' RPFO ; 'faire' SEKNOR ;

Après cette microinstruction ADO contient 10 car il a été décrémenté de 1 et les 4 premiers bits du code opération ont été empilés en poids forts de PILE.

D2 : 'faire RDANE1 ; 'faire' CSE2 ; 'faire' CSDANE2 ; CONS : = 0010 ;
'faire' INTER ; 'faire' SEKNOR ;

On teste alors le 7e bit (bit d'index) en faisant l'intersection des 4 bits suivants de l'instruction avec la constante 0010.

D3 : 'faire' RDANE1 ; 'faire' CSE2 ; 'faire' CSDANE2 ; CONS : = 0001 ;
'faire' INTER ; 'faire' CONDI1 ; 'faire' SEKNOR ;

On teste de la même façon cette fois le 8e bit (bit d'indirection et on mémorise dans CONDI1 le résultat du précédent test ($S \neq 0$).

D5 : 'faire' COND12 ; 'faire' SEKNOR;

On mémorise dans CONDI2 le résultat du 2e test.

D6 : 'faire' RDANE1 ; 'faire' CSE2 ; 'faire' CSDANE2 ; CONS : = 1100 ;
'faire' INTER ; 'faire' RPFA ; 'faire' SEKNOR ;

On sélecte les 4e et 5e bits du code opération en faisant une intersection avec 1100 (pour éliminer les bits de test et d'indirection) et on les envoie en poids faibles dans PILE.

Bien sûr les microinstructions que nous venons de donner sont incomplètes et beaucoup d'autres actions peuvent d'effectuer simultanément à celles décrites. Mais on a pu voir comment les tests de bits particuliers sont opérés et comment on sauvegarde dans PILE le code opération.

A la fin du microprogramme de décodage de l'instruction, le branchement au microprogramme convenable d'exécution se fera très simplement par l'ordre 'faire' BCADR ; qui dépile ce code opération alors interprété comme une adresse de microprogramme.

3) Décalage logique à gauche d'un mot double longueur

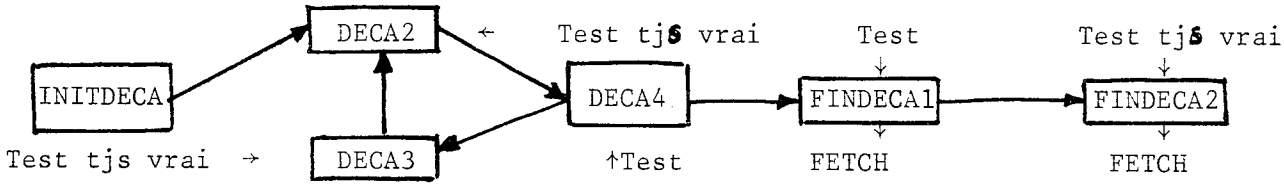
Une instruction de décalage se trouve dans R avec son code de R (1:6) et le nombre de positions de décalage dans R(9:14), la quantité double longueur à décaler se trouve dans LMEM |1|, AD0 : = 1, AD1 : = 0, AD2 : = 0, AIG : = 1 , A : = 0, LMEM |2| : = 0.

```
INITDECA : 'faire' OP2 ; 'faire' RDANE2 ; 'faire' SDANAD12 ; 'faire' SDANAD ;
           'faire' RDANA ; 'faire' SEKNOR ; 'faire' FDECALG ;

DECA2 : 'faire' DECG ; 'faire' MDANE2 ; 'faire' MBDANE1 ; 'faire' OU ;
        'faire' SDANM ; 'faire' BCPLUS2 ; 'faire' VRAI ;

DECA3 : 'faire' DECP ; 'faire' MBDANE2 ; 'faire' SDANM ; 'faire' ADBPS1 ;
        'faire' BCMOINS1 ; 'faire' VRAI ;

DECA4 : 'faire' ADPS1 ; 'faire' TAD1NZERO ; 'faire' BCMOINS1 ;

FINDECA1 : 'faire' OP1 ; 'faire' RDANE1 ; 'faire' SDANM ; ADRDEROUT : =
           'faire' BCADR ; 'faire' TAD2ZERO ;

FINDECA2 : 'faire' ; 'faire' SDANM ; ADRDEROUT : = FETCH ; 'faire' VRAI ; 'faire'
           BCADR ;
```

Explications :

α) La première microinstruction initialise les opérations. Soit n le nombre de décalages contenu dans R (9:14). R (9:12) représente le quotient de n par 4 et R (13:14) le reste. On charge R (13:14) dans A, c'est en effet le nombre (<4) de positions dont il faudra décaler chaque morceau de 4 bits de l'opérande contenu dans LMEM |2| (et non LMEM |1| puisque maintenant AIG = 1). On charge R (9:14) dans AD1, c'est le nombre de cases de 4 bits dont il faudra en plus translater l'opérande. Cette translation se résume donc à un changement de AD1.

β) Les 3 microinstructions qui suivent forment une boucle qui effectue le décalage dit.
Le graphe de séquencement est le suivant :

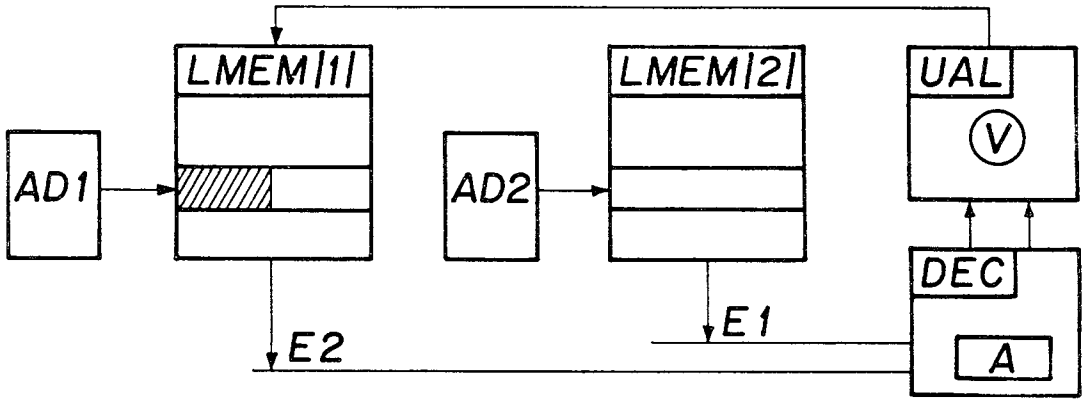


On voit comment le séquencement de Z0001 permet de concevoir des boucles de 3 microinstructions qui se répètent jusqu'à ce qu'un test indique la fin d'opérations. Nous verrons qu'il est aussi possible de concevoir des boucles de 2 microinstructions et même d'une seule microinstruction.

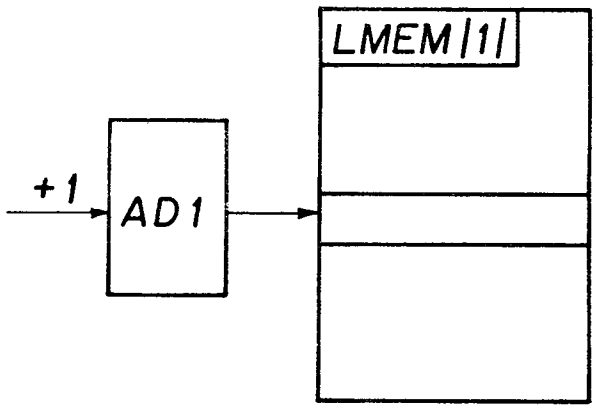
Cette propriété permet l'économie d'un nombre considérable de microinstructions.

γ) Dans DECA2, DECA3 et FINDECA2, un test de la condition (dont on voit ici la nécessité) assure la séquence par +2, -1 et déroutement à FETCH dans ADMEMC.
Dans DECA4 le test de AD1 ≠ 0000 aiguille sur -1 s'il est vrai et +1 s'il est faux.
Dans FINDECA1 le test porte sur AD2 = 0000 et assure le déroutement à FETCH s'il est vrai, sinon le passage à FINDECA2.

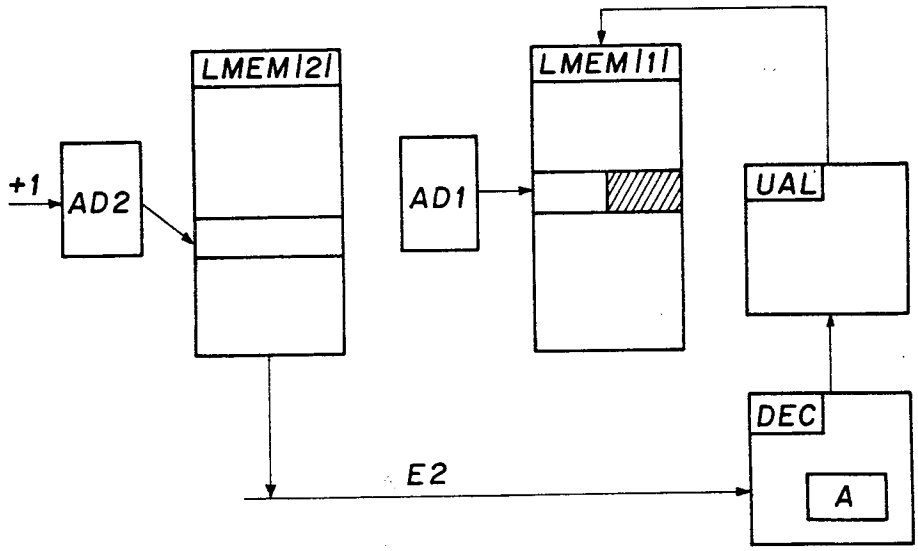
δ) DECA2 : un morceau de 4 bits est extrait de LMEM |2| décalé à gauche, on en fait le OU logique avec le mot courant de LMEM |1| qu'il remplace.



DECA4 : On teste AD1 = 0000



DECA3 : On décale à droite (du complément à 4 du contenu de A) le mot courant de LMEM |2|, en l'envoyant à DEC par E2 (voir le fonctionnement de l'unité DEC) et on stocke le résultat dans LMEM |1|, puis on incrémente AD2.



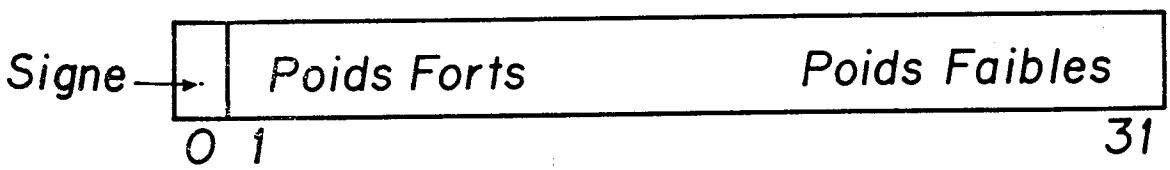
Le décalage d'un morceau de 4 bits de l'opérande se fait donc en 3 opérations, décalage à gauche, décalage à droite complémentaire du morceau précédent et OU logique des 2 résultats. Ce fonctionnement de l'unité DEC entraine la propriété fondamentale :

Dans Z0001 tous les décalages ont une durée indépendante du nombre de positions de décalage et proportionnelle à la longueur de l'opérande.

Ce qui assure une rapidité maximum à cette opération dès que le nombre de positions décalées dépasse 1.

ε) Les microinstructions FINDECA1 et FINDECA2, corrigent un effet de bord dû au test AD1 ≠ 0000 de sortie de boucle en tenant compte aussi du cas ou n < 4.

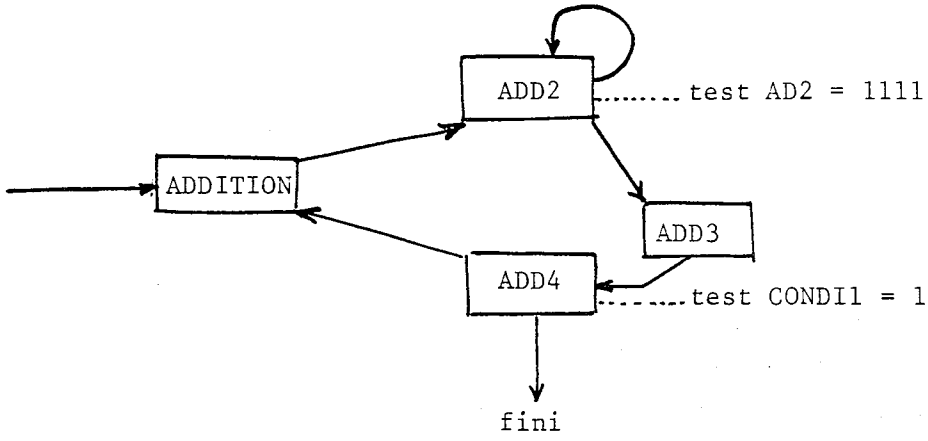
4) Addition de deux nombres de 31 bits en valeur absolue et signe



Le premier opérande est dans LMEM |1| (,0:7), le second dans RSPES (,1:2) AD1 = AD0 = AD2 = 0,AD3 =0100. Une première séquence de 6 microinstructions teste le signe des opérandes. Si tous deux ont le même signe on va à la séquence ADDITION, sinon à SOUSTRACTION1 et SOUSTRACTION2 (on soustrait l'opposé du nombre négatif, le résultat sera négatif si la valeur absolue du nombre soustrait est plus grande que l'autre).

Avec ces hypothèses l'opération de soustraction se ramène à changer de signe le 2e opérande puis à lui appliquer les séquences d'addition.

Graphe du séquencement d'ADDITION



ADDITION : 'faire' SPESDANR, 'faire' CONDI1 ; CONS : = 1101 ; 'faire'SDANM ;
'faire' SDANADI2 ; 'faire' SEKNOR ;

ADD2 : 'faire' MDANE1, 'faire' RDANE2 ; 'faire' ADD ; 'faire' AD12PSS ;
'faire' SDANM ; 'faire' ADOPS1 ; 'faire' BCPLUSO ; 'faire' TAD2UN ;

ADD3 : 'faire' CONSAD3 ; CONS : = 1000 ;

ADD4 : 'faire' ZERO ; 'faire' BCADR ; ADRDEROUT : = ADDITION ; 'faire' TCONDI1;

α) Dans la première microinstruction on charge les poids faibles du 2e opérande RSRES (,1) dans R, on positionne CONDI1 et au premier passage on fait en sorte que dans l'instruction qui précède S ≠ 0 donc CONDI1 est mis à 1, enfin on charge 1100 dans AD2.

β) La deuxième microinstruction fourni l'exemple d'une boucle réduite à une seule microinstruction : en effet tant que le test AD2 = 1111 n'est pas vrai on répète ADD2. Or ADD2 à chaque fois effectue l'addition de 4 bits issus de R avec 4 bits issus de LMEM |1|. A chaque pas AD1, AD2 et AD0 sont augmentés de 1. Lorsque AD0 et AD1 valent 10, AD2 vaut 1111, donc au pas suivant AD0 vaut 11, de même AD1, le test AD12 = 1111 est pris en compte car la bascule de test a pu être positionnée au pas précédent. On voit AD2 fonctionner comme compteur différentiel et servir à mettre fin à une boucle (réduite ici à une seule microinstruction) qui pourrait

avoir une longueur quelconque ($\leqslant 16$).

γ) Alors ADD3 sélecte la 2e partie du 2e opérande RSPES (2).

δ) Puis ADD4, teste CONDI1. Le branchement à ADDITION est effectué puisque CONDI1 vaut 1. En même temps ZERO provoque S = 0000, donc ADDITION va mettre cette fois CONDI1 à 0 ; au prochain passage sur ADD4 le branchement à ADDITION ne sera pas pris. C'est ce qu'il faut puisque les 31 bits seront alors additionnés.

Ce microprogramme montre l'utilisation de AD0, AD1 et AD2 comme compteurs testables. La séquence présentée coûte 14 cycles de microinstructions pour une addition (ou soustraction de 31 bits), si l'on rajoute les microopérations qui servent à tester les signes avant cette séquence, puis à les positionner après, on arrive à 25 cycles de 400 VS, soit 10 µS pour la phase complète d'exécution. (non compris la phase de décodage de l'instruction).

5) Multiplication de 2 nombres de 31 bits en valeur absolue et signe, avec accumulation.

Le multiplicande est rangé dans LMEM |2| (,8:15), une première opération de transfert (5 µS) range le multiplicateur dans LMEM |2| (,0:7).

Le résultat de la multiplication est ajouté au contenu de LMEM |1|(,0:15)

La séquence qui suit a été décrite dans (1).

On teste successivement les groupes de 4 bits du multiplicateur (dont l'adresse est maintenue dans R0) à l'aide de masques successifs 0001, 0010, 0100, 1000 conservés dans R1.

Lorsqu'on rencontre un 1 on ajoute en le décalant le multiplicande au résultat partiel stocké dans M2.

On a : AD0 = AD1 = AD2 = 0, A = 0 , AIG = 0; R=0 ;

α1) MULTIP : 'faire' FDECALG ; 'faire' ZERO ; 'faire' SDANR ; 'faire' SEKNOR ; 'faire' ADOPS1 ;

α2) LOOP1 : CONS = 0001 ; 'faire' CSE2 ; 'faire' CONSE2 ; 'faire' SDANR ; 'faire' ADOMS1 ;

α3) LOOP2 : 'faire' RDANE1 ; 'faire' OP1 ; 'faire' SDANM ; 'faire' SDANAD12 ;
'faire' ADOPS1 ;

α4) L1 : 'faire' RDANE1 ; 'faire' MBDANE2 ; 'faire' INTER ; 'faire' SDANA ;

α5) L2 : 'faire' TSZERO ; 'faire' BCADR ; ADRDEROUT : = LULU ;

α6) L3 : CONS : = 0100 ; 'faire' CSE2 ; 'faire' CONSE2 ; 'faire' DECD ;

α7) L4 : 'faire' CONDI1 ; 'faire' CONSE2 ; 'faire' CSE2 ; CONS : = 1000 ;
'faire' ADOMS1 ; 'faire' SDANM ; 'faire' SDANAD12 ;

α8) L5 : 'faire' RDANE2 ; 'faire' OP2 ; 'faire' SDANM ; 'faire' SDANAD12 ;

α9) 16 : 'faire' BCADR ; ADRDEROUT : = TOTO ; 'faire' TNCONDI1 ;

α10) MULT : 'faire' MBDANE2 ; 'faire' MDANE1 ; 'faire' ADD ; 'faire ADOPS1 ;
'faire' AD12PS1 ; 'faire' SDANM ; 'faire' SEKNOR ;

α11) MULTBIS : 'faire' RDANE2 ; 'faire' MDANE1 ; 'faire' SDANM ; 'faire'
TAD2UN ; 'faire' BCMOINS1 ;

α12) MULTTER : 'faire' BCADR ; 'faire' ADOMS1 ; ADRDEROUT : = LULU ; 'faire'
VRAI ;

α13) TOTO : 'faire' MBDANE2 ; 'faire' MDANE1 ; 'faire' DECG ; 'faire' ADD ;
'faire' ADMS1 ; 'faire' SDANM ; 'faire' BCPLUS2 ; 'faire' TAD2UN

α14) TOTO BIS : 'faire' MDANE2 ; 'faire' MBDANE1 ; 'faire' DECD ; 'faire'
ADD ; 'faire' ADBPS1 ; 'faire' SDANM ; 'faire' BC MOINS1 ;
'faire' VRAI ;

α15) TOTOER : 'faire' MDANE1 ; 'faire' CONSE2 ; 'faire' CSE2 ; CONS : = 0000
'faire' ADD ; 'faire' ADBPS1 ; 'faire' ZEDANSA ;

α16) LULU : 'faire' RDANE1 ; 'faire' RDANE2 ; 'faire' ADD ; 'faire' SDANR ;
'faire' ADOMS1 ;

α17) LULUBIS : 'faire' BCAD12 ; 'faire' TSNZERO ; ADRDEROUT : = LOOP2 ;

α18) SUITE : 'faire' RDANE1 ; 'faire' CONSE2 ; 'faire' CSE2 ; CONS : = 0001 ;
'faire' ADD ; 'faire' SDANR ;

α19) ENCORE : 'faire' RDANE1 ; 'faire' CONSE2 ; 'faire' CSE2 ; CONS : = 1000 ;
'faire' ADD ; 'faire' ADOPS1 ;

α20) TOUJOURS : 'faire' BCADR ; 'faire' TSNZERO ; ADRDEROUT : = LOOP1 ;

α21) REJOUR : 'faire' BCADD ; 'faire' PASDECAL ;

Explications

L'opération précédente a x x + b est valable dans le cas où a, x, et b sont positifs, ou encore a et x de même signe et b positif. Si on a a et x de signes contraires ou b négatif, il suffit de remplacer dans les microinstructions TOTO et MULT, l'opération ADD par SOUS.

L'opération a x x + b ici calculée est une opération de base dans l'évaluation d'un polynôme ou dans le calcul d'une moyenne pondérée. Ces deux calculs sont simples à microprogrammer sur Z0001.

β1) La première microinstruction positionne DEC sur l'état DECALG, qu'elle gardera durant toute l'opération.

β2) LOOP1, positionne R(,1) à 0001 premier masque.

β3) LOOP2, contient le chargement de R (,0) dans AD2.

β4) L1, voit l'intersection d'un morceau de 4 bits du multiplicateur (qui reste le même tant qu'on reste dans LOOP2) avec le masque contenu dans R1. Le résultat est envoyé à l'entrée avec en-codeur de A (dans DEC).

β5) L2. Si l'intersection précédente est nulle, c'est que le multiplicateur n'a pas de 1 dans la position testée, on va tout de suite à LULU.

β6) L3. Sinon, le multiplicateur a un 1, et par un décalage d'une constante on cherche à voir si ce 1 est en poids faibles, auquel cas on applique les opérations MULT au lieu de TOTO.

β7) L4. On charge 1000 dans AD2.

β8) L5. On charge R0 dans AD1.

β9) L6. Suivant le résultat précedemment testé on va à TOTO ou à MULT.

β10) MULT. On envoie le multiplicande dans E2, le résultat partiel dans E1. Le multiplicande est ajouté au résultat partiel (puisqu'il n'y a pas de décalage dans ce cas).

β11) MULTBIS . Complète seulement MULT en ajoutant un report éventuel et en testant la fin d'opérations par AD2 = 1111.

β12) MULTTER. Opère le saut inconditionnel à LULU.

β13) TOTO. opère le décalage à gauche du multiplicande et son addition au résultat partiel (le multiplicande peut être décalé de 1, 2 ou 3 positions suivant la place du 1 du multiplicateur qui a initialisé DEC).

β14) TOTOBIS. complète TOTO en faisant le décalage à droite du morceau suivant de 4 bits du multiplicande et en l'ajoutant au résultat intermédiaire. Comme on l'a déjà vu pour les décalages logiques, c'est une alternance de décalages à gauche suivie de décalages complémentaires à droite qui permet de récupérer les bits débordant .

β15) TOTOTER termine cette boucle de 2 microinstructions, dont on est sorti depuis TOTO par test de AD2 = 1111 , en ajoutant un éventuel report final et en remettant A à 0.

β16) LULU, fait progresser le masque contenu dans R1 en déplaçant le 1 d'une position à gauche.

β17) Si on n'a pas épuisé les 4 masques possibles on reboucle sur LOOP2.

β18) SUITE : sinon on incrémente R0 de 1.

β19) ENCORE : puis on teste R0 + 1000.

β20) TOUJOURS : Si R0 $<$ 8 on reboucle sur LOOP1, sinon on a traité tout le multiplicateur, l'opération est finie, le résultat est dans LMEM |1|(0:15)

β21) RETOUR : on dépile une adresse de retour car ce qui précède sera utilisé comme sous-microprogramme.

Durée de l'opération : elle dépend du nombre de 1 du multiplicateur : soit p ce nombre, la durée est 400 VS x (px12 +(31 - p) x 5 + 14) soit en moyenne : 110 µS.

IV - La simulation de Z0001

Le système CASSANDRE sous sa forme actuelle n'accepterait pas encore la description de Z0001 que nous avons donnée dans le début du chapitre. C'est donc la description suivante, légèrement modifiée mais strictement équivalente qui servira à la simulation et au traitement du chapitre suivant.

a) Description de Z0001

```
FILE: SOURCE   CASSPGR  P1                          CAMBRIDGE  MONITOR  SYSTEM

'UNITE' COMPTEUR(H,P(0:3),PE,CU,CD,RAZ;Q(0:3),TC);
  'REGISTRE' CT(1:4);
  'SIGNAL' MO1(0:3),MO2(0:3);
  'HORLOGEMERE' H,
                Q:=CT;
                MO1:=*D'1'(CT.MO2+CT.MO1+MO1.MO2)&0;
       'SI' CU&CD 'ALORS'
        (  MO2:='0001',
                   TC:=/.CT)
        (  MO2:='1111';
                   TC:=/.(-CT));
          <H>
             'SI' PE+(CU+CD)&RAZ 'ALORS'
               (  CT<=P)
               (  CT<=CT#(MO2#MO1))
               (  CT<='0000');


'UNITE' LMEM(H,I,D(0:3),AD12(1:3),AIG,SS,COM12;Q(0:3),C(0:3),TCC(1:4)
);
  'SIGNAL' FONCTION(0:7),TC,SC,PE,CU,CD,CS;
  'HORLOGEMERE' H,I;
  'EXTERNE' COMPTEUR('HORLOGE',(1:4),,,,;(1:4),),ML('HORLOGE',(1:4),(1
  :4),;(1:4)),VALN3((0:2);(0:7));
                COMPTEUR(H,D,PE,CU,CD,COM12;Q,TC);
                ML(I,Q,D,CS;C);
                VALN3(AD12;FONCTION);
       'SI' FONCTION 'ALORS'
        (
        (  CU:=1)
        (  CU:=AIG)
        (  CU:=-AIG)
        (  PE:=SS=AIG;
                   SC:=1)
        (  CD:=1)
        (  CD:=AIG)
        (  CD:=-AIG),
              CS:=SS.AIG.-SC;
       'SI' AIG 'ALORS'
        (  'SI' CU&CD 'ALORS'
           (  TCC(1):=TC)
           (  TCC(3):=TC))
       'SINON'
         (  'SI' CU&CD 'ALORS'
          (  TCC(2):=TC)
          (  TCC(4):=TC));
```

```
FILE: SOURCE    CASSPGR  P1                                  CAMBRIDGE  MONITOR  SYSTEM

'UNITE' ML(I,A(0:3),D(0:3),CS;D(C:3));
  'REGISTRE' M(1:4,0:15);
  'HORLOGEMERE' I;
  'EXTERNE' VALN4((1:4),(1:16));
        <I>
           'SI' CS 'ALORS'
             ('SI' VALN4(;*) 'ALORS'
                (  M(,0)<=D)
                (  M(,1)<=D)
                (  M(,2)<=D)
                (  M(,3)<=D)
                (  M(,4)<=D)
                (  M(,5)<=D)
                (  M(,6)<=D)
                (  M(,7)<=D)
                (  M(,8)<=D)
                (  M(,9)<=D)
                (  M(,10)<=D)
                (  M(,11)<=D)
                (  M(,12)<=D)
                (  M(,13)<=D)
                (  M(,14)<=D)
                (  M(,15)<=D));
      'SI' VALN4(A;*) 'ALORS'
       (  D:=M(,0))
       (  D:=M(,1))
       (  D:=M(,2))
       (  D:=M(,3))
       (  D:=M(,4))
       (  D:=M(,5))
       (  D:=M(,6))
       (  D:=M(,7))
       (  D:=M(,8))
       (  D:=M(,9))
       (  D:=M(,10))
       (  D:=M(,11))
       (  D:=M(,12))
       (  D:=M(,13))
       (  D:=M(,14))
       (  D:=M(,15));
```

```
FILE: SOURCE   CASSPOR  P1                                  CAMBRIDGE  MONITOR  SYSTEM

'UNITE' UAL(I1,I2,OP1(0:3),OP2(0:3),COM(0:3),SET,RAZ;S(0:3),SREPT,SZ);
  'REGISTRE' L(0:3),REPT;
  'SIGNAL' RS7(0:4),RS8(0:3),RS1(0:4),RS2(0:4),RS3(0:4),RS4(0:3),E1(0:
  3),E2(0:3),E3(0:3),E4(0:3),C,RS5(0:3),RS6(0:3);
  'HORLOGEMERE' I1,I2;
  'EXTERNE' VALN4(0:3);(0:15));
              E1:=(OP1#OP2)#RS1(1:4);
              E2:=(OP1#-OP2)#RS7(1:4);
              E3:=(OP1#OP2)#RS3(1:4);
              E4:=(OP1#-OP2)#RS2(1:4);
              RS1:=RS4&REPT;
              RS2:=RS3&1;
              RS3:=RS8&0;
              RS7:=RS8&REPT;
              RS8:=*D11|(0&(OP1.-OP2)+0&OP1.RS7+0&-OP2.RS7);
              RS4:=*D11|(0&(OP1.OP2)+0&OP1.RS1+0&OP2.RS1);
              RS5:=*D11|(0&(OP1.-OP2)+0&OP1.RS2+0&-OP2.RS2);
              RS6:=*D11|(0&(OP1.OP2)+0&OP1.RS3+0&OP2.RS3);
              VALN4(COM;);
         <I1>
            'SI' VALN4(;*) 'ALORS'
              (  L<=E1)
              (  L<=E2)
              (  L<=E3)
              (  L<=E4)
              (  L<=-OP1.OP2)
              (  L<=OP2)
              (  L<=OP1#OP2)
              (  L<=OP1+OP2)
              (  L<=-(OP1+OP2))
              (  L<=-(OP1#OP2))
              (  L<=-OP2)
              (  L<=OP1+-OP2)
              (  L<=0000)
              (  L<=OP1.OP2)
              (  L<=OP1.-OP2)
              (  L<=OP1);
         <I2>
            'SI' -/+COM(0:1)&RAZ&SET 'ALORS'
              (  REPT<=RS4(0))
              (  REPT<=0)
              (  REPT<=1);
              SREPT:=REPT;
              SZ:=/+L;
              S:=L;
```

```
FILE: SOURCE   CASSPGR  P1                                  CAMBRIDGE  MONITOR  SYSTEM

'UNITE' DEC(1,CDEC,COMDEC(1:8),DP1(1:4),DP2(1:4),SA1(1:2),SA2(1:4);SG(
1:4),SD(1:4));
  'REGISTRE' A(1:2),
  'SIGNAL' CDG(1:2),CDD(1:2),CD1(1:2),CD2(1:2);
  'HORLOGEMERE' 1;
  'EXTERNE' VALN2((0:1),(0:3)),DECG((1:4),(1:2);(1:4)),DECD((1:4),(1:2
  );(1:4));
              VALN2(A;);
      'SI' CDEC 'ALORS'
       (  CD1:=A;
          'SI' VALN2(;*) 'ALORS'
         (  CD2:=00)
         (  CD2:=11)
         (  CD2:=10)
         (  CD2:=01))
      'SINON'
        (  'SI' VALN2(;*) 'ALORS'
         (  CD1:=00)
         (  CD1:=11)
         (  CD1:=10)
         (  CD1:=01);
                CD2:=A);
        <I>
           'SI' COMDEC(1:5) 'ALORS'
             (  A<=SA1)
             ('SI' SA2 'ALORS'
               (  A<=00)
               (  A<=01)
               (  A<=10)
               (  A<=11))
             (  A<=00)
             (  A<=01)
             (  A<=10);
      'SI' -(COMDEC(0)+COMDEC(8)) 'ALORS'
              CDG:=CD1;
      'SI' -(COMDEC(7)+COMDEC(8)) 'ALORS'
              CDD:=CD2;
             DECG(DP1,CDG;SG);
             DECD(DP2,CDD;SD);
              CDD:=CD2
      'SINON'
                 CDD:=00;
             DECG(DP1,CDG;SG);
             DECD(DP2,CDD;SD);
```

```
FILE: SOURCE   CASSPOR  P1                                  CAMBRIDGE  MONITOR  SYSTEM

'UNITE' SPES(I2,I3,ST(1:16),SR(1:16),EAD(1:4),CECH,CSP,CC(2:2);SSPES(1
:16));
  'REGISTRE' ADS(1:4),SPE(1:16,0:3);
  'SIGNAL' INSP(1:16);
  'HORLOGEMERE' I2,I3;
  'EXTERNE' VALN2(0:1),(0:3));
              VALN2|1|(ADS(1:2));
              VALN2|2|(ADS(3:4));
      'SI' VALN2|1|(;*) 'ALORS'
       (  SSPES:=SPE(,0))
       (  SSPES:=SPE(,1))
       (  SSPES:=SPE(,2))
       (  SSPES:=SPE(,3));
         <I2>
            'SI' CSP 'ALORS'
              ('SI' VALN2|2|(;*) 'ALORS'
                (  SPE(,0)<=INSP)
                (  SPE(,1)<=INSP)
                (  SPE(,2)<=INSP)
                (  SPE(,3)<=INSP));
         <I3>
            'SI' CC 'ALORS'
                 ADS<=EAD;
              INSP:='SI' CECH 'ALORS' ST 'SINON' SR;



'UNITE' COMPT(H,P(0:1),PE,CO,CD,RAZ;Q(0:1),TC);
  'REGISTRE' ADD(1:2);
  'SIGNAL' MO1(0:1),MO2(0:1);
  'HORLOGEMERE' H,
      'SI' COCD 'ALORS'
       (  MO2:='0',
                 TC:=/.ADD)
       (  MO2:='1',
                 TC:=/.-ADD);
            MO1:='0'|1|(ADD.MO2+ADD.MO1+MO1.MO2)&0;
            Q:=ADD,
         <H>
            'SI' PEQ(CO+CD)&RAZ 'ALORS'
              (  ADD<=P)
              (  ADD<=ADD#(MO2#MO1))
              (  ADD<=CO);
```

```
FILE: SOURCE   CASSPGR  P1                                  CAMBRIDGE  MONITOR  SYSTEM


'UNITE' R(I,H,ADD(1:2),RAZ,CR(1:2),RCOM(1:3),EADC(1:2),S(1:4),SSPES(1:
16);SR1(1:16),SR2(1:4),TC,TC5);
  'SIGNAL' TC4,TC1,Q(0:1),DDLQ(1:4),TC3,TC2,CAD(1:3),CR3(1:4),CR2(1:4)
  ,RR2(0:3,0:3),RR1(0:3,0:3),DD(1:16);
  'HORLOGEMERE' H,1;
  'EXTERNE' COMPT('HORLOGE',(1:2),,,,;(1:2),),COMPTEUR('HORLOGE',(1:4)
  ,,,,;(1:4),),DECOD((1:2),(1:2),(1:2);(1:3),(1:4),(1:4),(1:4));
      'SI' CR2&CR3 'ALORS'
       (  RR1(,0):=S)
       (  RR1(,1):=S)
       (  RR1(,2):=S)
       (  RR1(,3):=S)
       (  RR1(,0):=SSPES(1:4))
       (  RR1(,1):=SSPES(5:8))
       (  RR1(,2):=SSPES(9:12))
       (  RR1(,3):=SSPES(13:16));
            SR1:=RR2(,0)&RR2(,1)&RR2(,2)&RR2(,3);
      'SI' DDLQ 'ALORS'
       (  SR2:=RR2(,0))
       (  SR2:=RR2(,1))
       (  SR2:=RR2(,2))
       (  SR2:=RR2(,3)),
            COMPT(H,EADO,CAD(1),CAD(2),CAD(3),RAZ;Q,TC);
            DECOD(ADD,CR,Q;CAD,DDLQ,CR3,CR2);
            COMPTEUR(1)(I,RR1(,0),CR2(1)+CR3(1),TC2.RCOM(1),TC2.RCOM
            (2),RCOM(3);RR2(,0),TC1);
            COMPTEUR(2)(I,RR1(,1),CR2(2)+CR3(2),TC3.RCOM(1),TC3.RCOM
            (2),RCOM(3);RR2(,1),TC2);
            COMPTEUR(3)(I,RR1(,2),CR2(3)+CR3(3),TC4.RCOM(1),TC4.RCOM
            (2),RCOM(3);RR2(,2),TC3);
            COMPTEUR(4)(I,RR1(,3),CR2(4)+CR3(4),RCOM(1),RCOM(2),RCOM
            (3),RR2(,3),TC4);
            TC5:=TC1.TC2.TC3.TC4;



'UNITE' DECOD(ADD(1:2),CR(1:2),Q(0:1);CAD(1:3),DDLQ(1:4),CR3(1:4),CR2(
1:4));
  'SIGNAL' CR1(1:4),
  'EXTERNE' VALN2((1:2),(1:4));
      'SI' -CR(1).CR(2) 'ALORS'
            CR1:=DDLQ;
      'SI' CR(1).CR(2) 'ALORS'
            CR2:=DDLQ;
           CR3:='SI' CR(1).-CR(2) 'ALORS' 1111 'SINON' CR1;
      'SI' VALN2(2)(ADD;*) 'ALORS'
       (  CAD:=000)
       (  CAD:=001)
       (  CAD:=100)
       (  CAD:=010),
            VALN2(1)(Q;DDLQ);
```

FILE: SOURCE CASSPCK P1 CAMBRIDGE MONITOR SYSTEM

```
'UNITE' VALN2(E(1:2);S(1:4));
                S(1):=-E(1).-E(2);
                S(2):=-E(1).E(2);
                S(3):=E(1).-E(2);
                S(4):=E(1).E(2);


'UNITE' VALN3(E(1:3);S(1:8));
                S(1):=-E(1).-E(2).-E(3);
                S(2):=-E(1).-E(2).E(3);
                S(3):=-E(1).E(2).-E(3);
                S(4):=-E(1).E(2).E(3);
                S(5):=E(1).-E(2).-E(3);
                S(6):=E(1).-E(2).E(3);
                S(7):=E(1).E(2).-E(3);
                S(8):=E(1).E(2).E(3);


'UNITE' VALN4(E(1:4);S(1:16));
                S(1):=-E(1).-E(2).-E(3).-E(4);
                S(2):=-E(1).-E(2).-E(3).E(4);
                S(3):=-E(1).-E(2).E(3).-E(4);
                S(4):=-E(1).-E(2).E(3).E(4);
                S(5):=-E(1).E(2).-E(3).-E(4);
                S(6):=-E(1).E(2).-E(3).E(4);
                S(7):=-E(1).E(2).E(3).-E(4);
                S(8):=-E(1).E(2).E(3).E(4);
                S(9):=E(1).-E(2).-E(3).-E(4);
                S(10):=E(1).-E(2).-E(3).E(4);
                S(11):=E(1).-E(2).E(3).-E(4);
                S(12):=E(1).-E(2).E(3).E(4);
                S(13):=E(1).E(2).-E(3).-E(4);
                S(14):=E(1).E(2).-E(3).E(4);
                S(15):=E(1).E(2).E(3).-E(4);
                S(16):=E(1).E(2).E(3).E(4);


'UNITE' DECG(GP1(1:4),G01(1:2);SG(1:4));
  'EXTERNE' VALN2((0:1);(0:3));
      'SI' VALN2(G01;*) 'ALORS'
        (  SG:=GP1)
        (  SG:=GP1(2:4)&00)
        (  SG:=GP1(3:4)&&0)
        (  SG:=GP1(4)&&&0);

'UNITE' DECD(GP2(1:4),G02(1:2);SD(1:4));
  'EXTERNE' VALN2((0:1);(0:3));
      'SI' VALN2(G02;*) 'ALORS'
        (  SD:=GP2)
        (  SD:=0&GP2(1:3))
        (  SD:=00&GP2(1:2))
        (  SD:=000&GP2(1));
```

FILE: SOURCE CASSPGR P1 CAMBRIDGE MONITOR SYSTEM

```
'UNITE' ZOOOI(H,I1,I2,I3,ST(1:16);SSPES(1:16),CECH(1:3),CM(1:3));
  'SIGNAL' TE(1:8),E1(1:4),E2(1:4),S(1:4),AD1(1:4),O1(1:4),AD2(1:4),O2
  (1:4),SR1(1:16),SR2(1:4),OP1(1:4),OP2(1:4),CONS(1:4),COMMANDE(1:15),
  TEST(0:9),AIG,CDEC,SS,CSP,CE1(1:2),CE2(1:2),CR(1:2),CUAL(1:4),AD12(1
  :3),ADO(1:2),CONS2(1:2),CC(1:2),CONTR(1:43);
  'HORLOGEMERE' H,I1,I2,I3;
  'EXTERNE' VALN3((0:2);(0:7)),LMEM('HORLOGE','HORLOGE',(1:4),(1:3),,,
  ;(1:4),(1:4),(1:4)),CONTROLE('HORLOGE','HORLOGE','HORLOGE',(1:4),(0:
  9),,(1:8);(1:43),(1:3),(1:3)),UAL('HORLOGE','HORLOGE',(1:4),(1:4),(1
  :4),,;(1:4),,),DEC('HORLOGE',,(1:8),(1:4),(1:4),(1:2),(1:4);(1:4),(1
  :4)),SPES('HORLOGE','HORLOGE',(1:16),(1:16),(1:4),,,;(1:16)),R(
  'HORLOGE','HORLOGE',(1:2),,(1:2),(1:3),(1:2),(1:4),(1:16);(1:16),(1:
  4),,);
              VALN3(1)(AIG&CE1;);
              VALN3(2)(AIG&CE2;);
      'SI' VALN3(1)(;*) 'ALORS'
        ( E1:=SR2)
        ( E1:=O1)
        ( E1:=O2)
        ( E1:=AD1)
        ( E1:=SR2)
        ( E1:=O2)
        ( E1:=O1)
        ( E1:=AD2);
      'SI' VALN3(2)(;*) 'ALORS'
        ( 'SI' CC(1) 'ALORS'
                E2:=CONS)
        ( E2:=SR2)
        ( E2:=O1)
        ( E2:=O2)
        ( 'SI' CC(1) 'ALORS'
                E2:=CONS)
        ( E2:=SR2)
        ( E2:=O2)
        ( E2:=O1),
            R(I2,I3,ADO,COMMANDE(11),CR,COMMANDE(8:10),CONS2,S,SSPES
            ;SR1,SR2,TEST(0),TEST(1));
            SPES(I2,I3,ST,SR1,CONS,CECH(1),CSP,CC(2);SSPES);
            DEC(I3,CDEC,COMMANDE(1:5)&COMMANDE(13:15),E1,E2,SR1(13:
            14),S,OP1,OP2);
            UAL(I1,I2,OP1,OP2,CUAL,COMMANDE(6),COMMANDE(7);S,TEST(4)
            ,TEST(5));
            LMEM(1)(I3,I2,S,AD12,AIG,SS,COMMANDE(12);AD1,O1,TE(1:4))
            ;
            LMEM(2)(I3,I2,S,AD12,-AIG,SS,COMMANDE(12);AD2,O2,TE(5:8)
            );
            CONTROLE(H,I1,I3,S,TEST(0:9),,;CONTR(1:43),CM(1:3),CECH(
            1:3));
            CR:=CONTR(1:2);
            CSP:=CONTR(3);
            CUAL:=CONTR(4:7);
            CE1:=CONTR(8:9);
            SS:=CONTR(10);
            AD12:=CONTR(11:13);
            ADO:=CONTR(14:15);
            CE2:=CONTR(16:17);
            CONS2:=CONTR(18:19);
            CONS:=CONTR(20:23);
            CDEC:=CONTR(24);
            CC:=CONTR(25:26);
            AIG:=CONTR(27);
            COMMANDE:=CONTR(28:42);
            TEST(0:9):=TE(1:4)+TE(5:8);
```

b) Simulation

1) La simulation d'une machine décrite en CASSANDRE a d'abord été étudiée par Monsieur LUSTMAN (5). Monsieur ANCEAU a ensuite réalisé, sur 360/67, avec l'aide de Monsieur PERRON le système de simulation synchrone (6) et étudié avec Monsieur COUTURIER la simulation asynchrone (6). La version que nous utiliserons ici, du simulateur synchrone a été mise au point par Monsieur MENARD qui réalise également le simulateur asynchrone.

La commande "range' (ou r) permet de donner une valeur aux entrées de l'unité simulée ou à ses registres. La commande cycle (ou c) permet de faire exécuter un cycle des horloges qui sont validées (ou égales à 1).

La commande "print" (ou p) permet de lire la valeur des variables. On peut créer des macrocommandes qui font exécuter une suite de commandes (exemple ici go ou initadd).

Pour la liste des commandes utilisables on pourra lire PERRON (6).

2) Simulation de l'unité COMPTEUR

```
                ***** traitement de l'unite  compteur  *****¢

***verification syntaxique et dimensionnelle de la description ***¢
cassys compteur

                EDITION

  U. S.   TEXTE SOURCE

 00001   'UNITE' COMPTEUR(H,P(0:3),PE,CU,CD,RAZ;S(0:3),TC);
 00044     'REGISTRE' CT(1:4);
 00053     'SIGNAL' MD2(0:3),MD1(0:3);
 00072     'HORLOGE/REPOS' H;
 00075       MD:=CT;
 00080       MD1:=*D|1|(CT.MD2+CT.MD1+MD1.MD2)¤;
 00114         'SI' CU&CD 'ALORS'
 00121          ( MD2:=0001;
 00131                    TC:=/.CT)
 00139          ( MD2:=1111;
 00149                    TC:=/.(-CT));
 00161            <H>
 00164               'SI' PE¤(CU+CD)¤RAZ 'ALORS'
 00180                 ( CT<=P)
 00186                 ( CT<=CT#(MD2#MD1))
 00205                 ( CT<=0000);

            FIN D'EDITION
```

MESSAGES DE VERIFICATION

***UNITE : COMPTEUR

-UNITE SANS ERREUR-

FIN VERIFICATION

R; S=0.04/1.41 16:43:14

***compilation de la description en un module simulable ***ç
cassin compteur ret2
R; T=1.57/2.40 16:46:02

***chargement du module simulable ***ç
simul ç
simul2 compteur
.* le programme simulateur attend des commandes
.*initialisation des variables d'entree
.r p=0101,pe=1
-
.*validation de l'horloge h
.r h=1
.*forcage parallele de 4 bits dans ct
.c
.p ct
0101
.*ct a bien pris la valeur de l'entree p

.*changement de commande (pour faire +1)
.r pe=0,cu=1
-
.*incrementation du compteur
.c
.p ct
0110
.*ct est bien passe de 5 a 6
.*initialisation de ct pour test passage a 1111
.r ct=1110
.*on va incrementer
.*visualisation de l'indicateur avant
.p tc
0
.c
.*visualisation de l'indicateur apres
.p tc
1
*valeur du compteur
.p ct
1111
.c
.p ct
0000
.p tc
0
*il repasse a zero (le compteur est cyclique)
*fin

.* le compteur cyclique repasse a zero
.*fin de la simulation

3) Simulation du microprogramme d'addition :

```
*   simulation du calculateur z0001
simul: z0001
.*addition de 2 nombres de 16 bits
.*contenus au depart dans lmem |1|( ,0:3) et lmem |2|(,0:3)
.*initialisations par une macro commande
.initadd
U L'MEM 1
U ML
R M(,0)=1111,M(,1)=1111 M(,2)=1111,M(,3)=0111
-
-
-
U Z0001
U L'MEM 2
U ML
R M(,0)=0001
U Z0001
U CONTROLE
R M(,0)=111 0000010001111100000000000000000000000000000000
U Z0001
U L'MEM 1
U COMPTEUR
R CT=0000
U Z0001
U L'MEM 2
U COMPTEUR
R CT=0000
U Z0001
U R
U COMPT
R ADD=00
U Z0001
.c 0
.*valeur des operandes : 0111111111111111 et 0000000000000001
.*on aura propagation de retenue sur les 4 groupes de 4 bits
.*validation horloge i1
.r i1=1
.c
.p e1,e2,s,test(4)
1111
-
0001
-
0000
-
0
.*e1 ,e2 sont les entrees de ual s la sortie test(4) le debordement memorisé
.*test(4 ) n'est pas encore etabli ,il sera charge par i2
.p test(4)
0
.validation de l'horloge i2
.r i2=1,i1=0
-
.*******************************
```

```
.*cycle d'horloge i2
.c
.p e1,e2,s,test(4)
1111
-
0000
-
0000
-
1
.*les entrées de ual ont changé,le debordement a ete pris en compte
.p ad1,ad2
0001
-
0001
.*ad1 et ad2 sont les adresse dans lmem 1 et lmem 2
.*validation de l'horloge i3
.r i3=1,i2=0
-
.c
.*validation de l'horloge h
.r h=1,i3=0
-
.c
.*nous avons effectue un pas de calcul pour faire les trois autres
.*nous ferons appel a une macro commande appelee go
.imp off
.go
.go
.go
.*visualisation du resultat
.p sr1
0000 0000 0000 1000
.*dans chaque tranche de 4 bits les poids faibles sont a droite
.*les 4 groupes sont disposes poids faibles a gauche
.*ceci n'est pas un defaut mais est du au mecanisme d'incrementation
.*d'adresse des registre compteur utilises dans r
.*fin de simulation
.fin
R; S=73.74/76.38 18:04:00
.go
R I1=1
C
R I1=0
C 0
R I2=1
C
R I2=0
C 0
R I3=1
C
R I3=0
C 0
R H=1
C
R H=0
C 0
.f   explicitation de la macro commande go
```

4) Simulation du microprogramme de décalage à gauche :

```
SIMUL Z0001
.*SIMULATION DU DECALAGE GAUCHE
.*SOURCE DANS LMEM 1 RESULTAT DANS LMEM 2
.*INITIALISATION DES VARIABLES
.INITDEC
.U LMEM 1
.U ML
.*VALEUR DE L'OPERANDE A DECALER
.P M
0110
1011
0100
0111
0101
1001
0111
0101
1011
1111
0100
1001
1100
1011
0110
1111
.U Z0001
.*EXECUTION DE LA PREMIERE MICROINSTRUCTION
.INITDECA
.P SR1
0000 0000 0011 0100
.P AD1,AD2
0000
0011
.U DEC
.P A
01
.U Z0001
.*DECALAGE GAUCHE DE LMEM 1(0) UNION AVEC LMEM 2(3)
.*RANGEMENT DANS LMEM 2(3)
.DECA2
.U LMEM 2
.U ML
.P M(,0:5)
0000
0000
0000
1100
0000
0000
.U Z0001
.*INCREMENTATION DE AD2
.DECA4
.P AD1,AD2
0000
0100
.*DECALAGE DROIT DE 3 POSITION DE LMEM 1(0)
.*RANGEMENT DANS LMEM 2(4) (POUR RECUPERER BIT DE GAUCHE)
.DECA3
.U LMEM 2
. ML [U ML
```

```
.P M(,0:5)
0000
0000
0000
1100
0000
0000
.U Z0001
.P AD1,AD2
0001
0100
.*AD1 A ETE INCREMENTE PAR DECA3
.*ON BOUCLE SUR DECA2
.DECA2
.U LMEM 2
.U ML
.P M(,0:5)
0000
0000
0000
1100
0110
0000
.U Z0001
.DECA4
.P AD1,AD2
0001
0101
.DECA3
.U LMEM 2
.U ML
.P M(,0:5)
0000
0000
0000
1100
0110
0001
.*LE BIT GAUCHE DE LMEM 1 (1) EST PASSE A DROITE DE LMEM 2(5)
.P AD1,AD2
0010
0101
.U Z0001
.DECA2
.U LMEM 2
.U ML
.P M(,0:5)
0000
0000
0000
1100
0110
1001
.*NOUS VOYONS L'UNION DANS LMEM 2(5) DE LMEMM1(2) DECALE A GAUCH
.*ET DE L'ANCIEN LMEM 2(5)
.*LA SIMULATION CONTINUERAIT AINSI JUSQU'A REMPLIR LMEM 2
```

BIBLIOGRAPHIE

1 - P. DARONDEAU, J. HERMET, "Projet d'étude d'un Minicalculateur Microprogrammé entièrement, compatible avec le 360, réalisable avec les moyens minimaux". Chap. 1 rapport final du contrat C.R.I., Industrialisation de CASSANDRE.

2 - P. PAULIAT, "Mémoire Vive Bipolaire"
Rapport de stage à la société TELEMECANIQUE, du 3 mai au 30 juin 71.

3 - GUERIN, LAGOUTIERE, "Hiérarchie de mémoire utilisant une technologie à domaines magnétiques", projet de fin d'études à l'ENSIMAG, juin 72.

4 - S.O. MKRTCHYAN, "Model of formal neuron with union fibers"Engineering cybernetics n° 3, 1970.

5 - LUSTMAN, "Simulation d'une machine digitale à partir d'une description en langage CASSANDRE", Revue bleue de l'A.F.I.R.O., B-2- 1969.

6 - F. ANCEAU, COUTURIER, J. DOUSSY, F. PERRON, "Compilation et simulation du langage CASSANDRE", chap. II, rapport final de contrat CRI, Industrialisation de CASSANDRE.

C - REALISATION LOGIQUE D'UN ENSEMBLE DIGITAL DECRIT EN CASSANDRE

I - Compilation de CASSANDRE en circuits logiques

a) Avertissement

La méthode que nous allons décrire a été étudiée et mise en oeuvre en collaboration avec, successivement, Messieurs ARCHER, LIDDELL et FANTINO. Le premier avait élaboré un début d'application de cette méthode, programmée en MAP 7044 et fondée sur l'utilisation d'un macrogénérateur.
Monsieur LIDDELL a décrit dans (1) sa thèse la mise en oeuvre sur le 360 IBM de cette méthode, par analyse syntaxique en se servant pour générer les analyseurs syntaxiques des programmes de Messieurs GRIFFITHS et PELTIER (comme c'était déjà le cas dans le système de simulation).
Monsieur FANTINO a repris la méthode et restructuré les programmes, en leur ajoutant ses apports personnels, pour en faire un système opérationnel et diffusable (2). Nous serons utilisateurs de ce système pour traiter l'exemple qui illustre cette partie, mais nous ne parlerons pas de la programmation en macro-assembleur 360, pour se limiter strictement à ce qui a fait l'objet de notre travail : la définition des fonctions sémantiques utilisées dans cette compilation.

b) Principes d'une compilation du hardware

Nous n'avons pas eu les premiers l'idée de passer par programme d'une description formelle de fonctions logiques (booléennes et séquentielles) à une réalisation de ces fonctions en circuits. Toutes les méthodes programmées de synthèses booléennes ou d'assignement d'un codage aux états d'un automate contiennent cette idée. Plus près de notre travail ceux de PROCTOR (3) et de FRIEDMANN avec le système ALERT (4) chez IBM traduisaient le même souci de génération automatique de circuits logiques. Mais nous avons systématisé ce processus en l'appliquant à un langage évolué de syntaxe complexe, en faisant de cette génération une simple compilation d'un langage origine en un langage cible mais surtout en prenant comme langage cible un sous ensemble du langage origine. C'est ce dernier point qui a les

conséquences méthodologiques les plus importantes.

Nous avons vu en effet, que CASSANDRE permet la description de réseaux de boites interconnectées (chaque boite pouvant à son tour contenir un réseau de même nature). Les seules notions du langage qui interviennent dans cette description sont :

- L'unité avec son entête.
- Les déclarations de type signal, externe, horloge et impulsion.
- Les connexions d'unité,de signaux-unités et de signaux.
- Eventuellement, les notions arithmétiques à fonction descriptive des connexions répétitives.

Toutes description en CASSANDRE-Général peut contenir ces notions, mais une description qui ne contient qu'elles sera appelée CASSANDRE-Elémentaire.

Réaliser une unité c'est passer de sa description en CASSANDRE-G à une description logiquement équivalente en CASSANDRE-E.
C'est l'opération inverse d'une extension, CASSANDRE-G pouvant considéré comme une extension de CASSANDRE-E.

Ce passage est donc une compilation particulière qui se traduit par un appauvrissement de la syntaxe. Pour l'effectuer il faut associer à toute notion syntaxique non contenue dans CASSANDRE-E un schéma d'expression de cette notion CASSANDRE-E, c'est ici notre but en faisant référence à LIDDELL (1) et FANTINO (2) chaque fois que possible.

Remarques :

1) Si l'entrée des programmes est décrite en CASSANDRE leur sortie l'est aussi, on peut donc en faire l'entrée de tout autre partie du système CASSANDRE (simulation, génération de microprogrammes...) nous en verrons en fin de chapitre, la nécessité dans l'utilisation conversationnelle des programmes qui est préconisée.

2) S'il existe dans la description en entrée des connexions simples de signaux ou d'unités, elles constituent déjà du CASSANDRE-E et demeurent inchangées dans le traitement. D'autre part une connexion conditionnelle d'unité subsistera sous une forme inconditionnelle après traitement. Les programmes laissent donc invariants les réseaux d'unités qui préexistaient, en leur superposant simplement d'autres réseaux, d'où le nom de SURDECOUPAGE pris par LIDDELL.

Exemple d'unité décrite en CASSANDRE-E (voir partie B)

```
'Unité' CALCULATEUR (I,H, RAZ, E(1:16,1:5) ; S(1:16, 1:4));
'Horloge' I,H ;
'Signal' ET (1:16), ST (1:16), TRAM (1:16), CM (1:3), PREM (1:16), CECH (1:3),
 PRAM (1:16), TREM (1:16), REM (1:16) ;
'Externe' ZOOO1 ('horloge', 'horloge', 1,1,16 ; 3,3,16) ,
          MEMOIRE ('horloge', 16,16,16,16,3 ; 4,16),
          ECHANGE ('horloge', 3,4,16,16, (16,5) ; 16,16,16,16,16,(16,4)) ;

ZOOO1 (H,I,RAZ,ET ; CM,CECH,ST) ;
MEMOIRE (H,TRAM,TREM,PRAM,C ; SC,REM) ;
ECHANGE (I,CECH,SC,ET,REM,E ; PRAM,PREM,TRAM,TREM,ST,S) ;
```

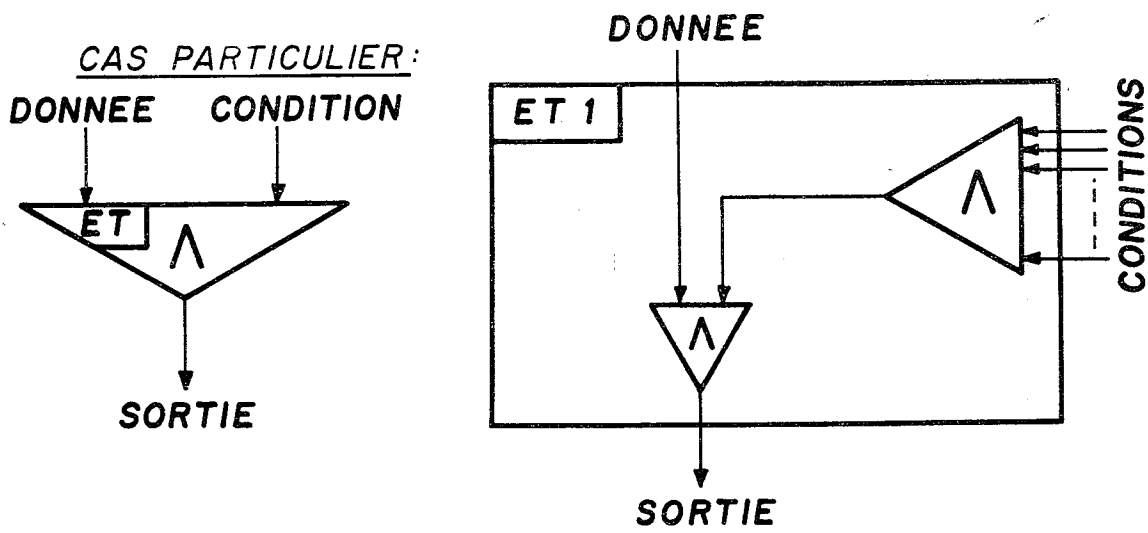
c) Unités terminales et unités primitives

On appelera unité terminale, une unité qui ne contient pas de déclaration 'externe' (donc pas d'autres unités). Une unité terminale peut être décrite en CASSANDRE-G et traitée par le SURDECOUPAGE*, elle ne sera plus terminale après traitement. Mais une unité terminale peut aussi correspondre à un macrocomposant existant et utilisable dans la réalisation d'autres unités. On évitera de lui appliquer le SURDECOUPAGE en la considérent comme un élément de base des circuits.

1) Unités primitives

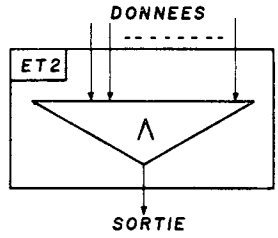
On appelera unité primitive une unité terminale qui correspond aux éléments de base "traditionnels" des schémas logiques, il y en a une douzaine :

- 'Unité' ET 1n (DONNEE, CONDITIONS (1:n) ; SORTIE) ;
  SORTIE : = DONNEE Λ (/Λ CONDITIONS) ;

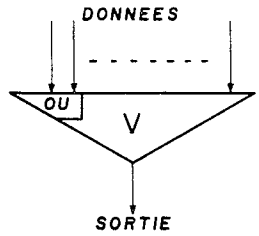


* Ou compilateur de hardware, c'est la même chose.

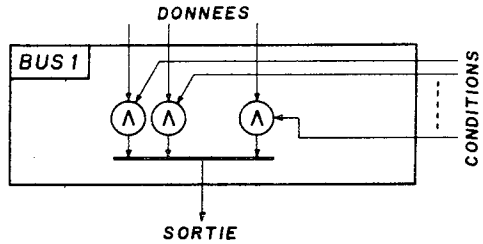
- 'Unité' ET2 n+1 (DONNEES (0:n) ; SORTIE);
  SORTIE : = /Λ DONNEES ;



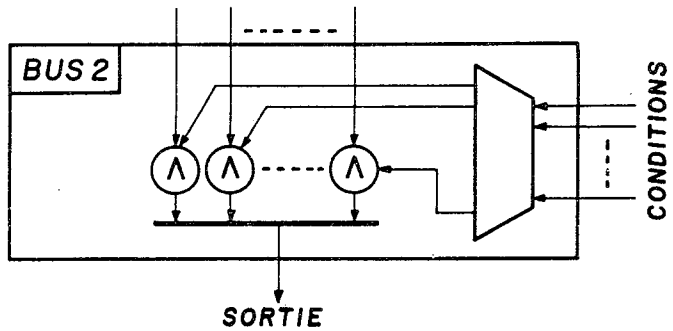
- 'Unité' OUn (DONNEES (1:n) ; SORTIE) ;
  SORTIE : = /V DONNEES ;



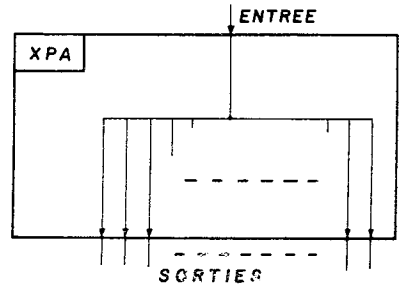
- 'Unité' BUS1n (CONDITIONS (1:n), DONNEES (1:n) ; SORTIE) ;
  SORTIE : = 'si' CONDITIONS alors DONNEES ;



- 'Unité' BUS2n (CONDITIONS (1:p), DONNEES (1:n) ; SORTIE) ;
  SORTIE : = 'si' &/ CONDITIONS alors DONNEES ;

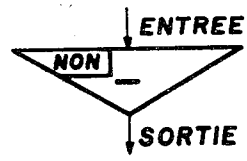


- 'Unité' XPAn (ENTREE ; SORTIES (1:n)) ;
  'Pour' I = 1 'a' n 'début'
  SORTIES (I) : = ENTREE ;



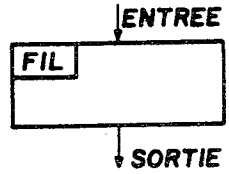
- `Unité' NON (ENTREE ; SORTIE) ;

SORTIE : = - ENTREE



- `Unité' FIL (ENTREE ; SORTIE) ;

SORTIE : = ENTREE ;

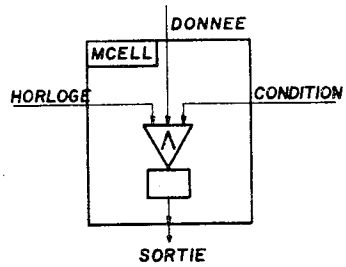


- `Unité' MCELL (HORLOGE,CONDITION,DONNEE ; SORTIE) ;

Horloge HORLOGE ; Registre M

<HORLOGE > si CONDITION alors M <= DONNEE ;

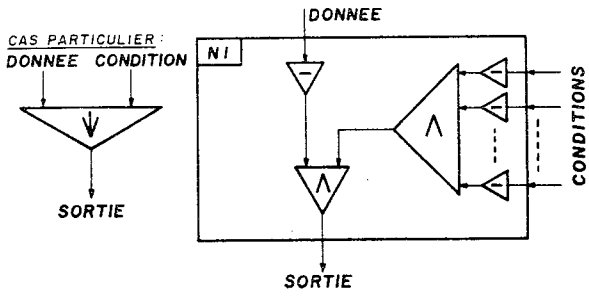
SORTIE : = M ;



- `Unité' NI 1n (DONNEE, CONDITIONS (1:n); SORTIE) ;

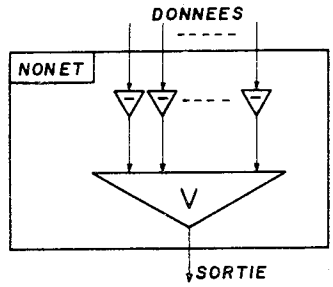
SORTIE : = -DONNEE Λ (/ Λ -CONDITIONS) ;

Cas particulier :



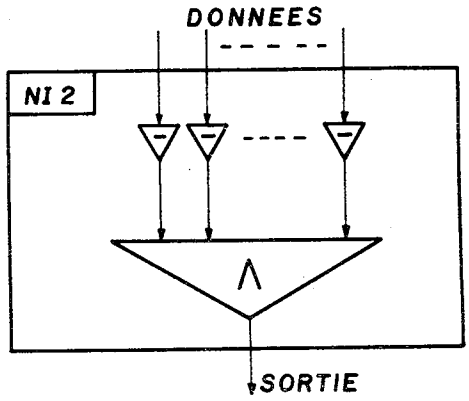
- `Unité' NONETn (DONNEES (1:n) ; SORTIE) ;

SORTIE : = / V - DONNEE ;



- Unité NI2n+1 (DONNEES (0:n) ; SORTIE) ;

SORTIE : = - DONNEES ;



Remarques :

α) Il convient d'être prudent si l'on utilise ces unités primitives dans une description. En effet, le compilateur de hardware va générer de telles unités avec un numéro d'ordre, si une unité de même type avec le même numéro existe il y aura une ambigüité que le programme ne peut lever.
On peut prendre, par exemple, un numéro d'ordre très élevé que le compilateur n'a que très peu de chance d'atteindre.

β) Les unités NI1 et ET1, qui ont une seule entrée "DONNEE" sont des portes. Dans ET1 si toutes les conditions valent VRAI, SORTIE : = DONNEE. Dans NI1 si toutes les conditions valent FAUX, SORTIE : = - DONNEE. Dans les autres cas SORTIE : = 0, ce qui correspond à notre définition de porte .

γ) Les unités NI2 et ET2, obtenues à partir de NI1 et ET1 en remplaçant les entrées CONDITIONS par des DONNEES, correspondent aux fonctions booléènnes ET et NI. Les unités FIL et NON correspondent à un simple transfert.

2) Expression homogène des unités primitives

(une expression est homogène si toutes ses entrées et sorties ont les mêmes dimensions).

- Unité ET2 (n+1) (DONNEE 0, DONNEE 1, ..., DONNEE n ; SORTIE) ;

  Externe ET 22 (1,1 ; 1)

  SORTIE : = ET 22 |1| (DONNEE 0, ET 22 |2| (DONNEE 1, ET 22 |3|
  (DONNEE 2, ... (ET 22 |n| (DONNEE n-1, DONNEE n ; *)
  ; *) ; .....) ; *) ;

- Unité ET1 (n) (DONNEEO, CONDITION1,... , CONDITIONn ; SORTIE) ;
  C'est une unité de type ET2 (n+1) en substituant CONDITION1.. CONDITIONn à DONNEE1... DONNEEn.

- Unité OUn (DONNEEO, DONNEE1,..., DONNEEn ; SORTIE);
  Externe OU2 (1, 1 ; 1) ;
  SORTIE : = OU2 |1| (DONNEEO, OU2 |2| (DONNEE1, OU2 |3| (...
  ..., OU2 |n-1| (DONNEE n-1, DONNEEn ; *) ;
  .....) ; *) ;

- Unité BUS 1n (CONDITION1, CONDITION2,..., CONDITIONn, DONNEE1, DONNEE2,..., DONNEEn ; SORTIE) ;
  Externe ET 22 (1, 1; 1) , OU n (1,1,..., 1 ; 1) ;
  SORTIE : = OUn (ET 22 |1| (CONDITION1, DONNEE1 ; *), ET 22 |2| (CONDITION2, DONNEE2 ; *),..., ET 22 |n| (CONDITIONn, DONNEEn ; *) ... ; *) ;

- Unité BUS 2n (CONDITIONS (1 : p), DONNEE1,..., DONNEEn ; SORTIE) ;
  Externe VALN p (p ; 1, 1...,1), BUS 1n (1,1,..., 1,1,1,...,1; 1) ;
  Signal COND1, COND2,..., CONDn ;
  SORTIE : = BUS 1n (COND1, COND2, ..., CONDn, DONNEE1, ...
  ... DONNEEn ; *) ;
  VALNp (CONDITIONS ; COND1, COND2,..., CONDn) ;

Dans le SURDECOUPAGE on produira à la demande des unités primitives sous forme terminale ou homogène.

Le surdécoupage se décompose en deux phases consécutives et indépendantes : partition en unités et réalisation des unités EXP (Expressions).

d) Partition en unités d'une description en CASSANDRE-G.

Ce premier traitement fait disparaître toutes les notions externes à CASSANDRE-E à l'exception des expressions qui représentent les circuits combinatoires. Ce traitement se décompose lui-même en :

α) Regroupement éventuel des notions séquentielles (état, allera, faire) dans une unité CONTROLE.

β) Création des unités EXP.

γ) Traitement des conditions.

δ) Traitement des registres.

ε) Création d'expanseurs et de bus.

1) L'unité de contrôle d'une unité U

Pour que cette boîte soit créée il faut et suffit que l'unité U en cours de traitement contienne au moins une étiquette. Si elle n'en a pas, c'est soit qu'elle ne contient que des fonctions combinatoires, soit que sa partie automate séquentiel est déjà réalisée (les états éventuels sont des registres, c'est le cas de toutes les unités définissant Z0001).

Quand elle existe, l'unité de contrôle va se voir attribuer toutes les expressions et variables d'états les ordres faire et allera, la liste des étiquettes de U.

L'unité de contrôle de U aura donc tous les états de U et l'unité combinatoire de U n'en aura plus, chaque état sera remplacé par une condition logique.

α) Entrées de l'unité de contrôle

Nous avons vu dans A,III,d,2 qu'il était possible de calculer les conditions cumulées portant sur chaque ordre allera ou faire de U, soit $\alpha 1$, $\alpha 2$,... αn, $\beta 1$, $\beta 2$... βp ces conditions résultantes. Leur calcul est effectué par des circuits, contenus dans l'unité opératoire de U dont les sorties constituent des entrées de l'unité de contrôle. Mais il est également possible (voir exemple suivant) de forcer un état de U à partir d'une autre unité externe et dans ce cas on crée pour U les entrées supplémentaires servant à ces forçages, ces entrées sont aussi des entrées de l'unité de contrôle de U. Enfin, la description en CASSANDRE-G peut contenir des chargements de variables déclarées Etat, comme celles-ci ont été attribuées à l'unité de contrôle, les conditions cumulées $\gamma 1$, $\gamma 2$,... γq qui valident ces chargements sont aussi des entrées de l'unité de contrôle de U.

Remarque : Dans le calcul de toutes les conditions cumulées mentionnées, les étiquettes n'interviennent pas. Donc par exemple, un ordre allera ou faire qui n'est conditionné que par l'état sous lequel on le trouve dans U n'introduira pas d'entrée de l'unité de contrôle de U.

Les horloges qui conditionnent les ordres allera et chargement de variable d'état, sont aussi des entrées de l'unité de contrôle de U.

Donc :

α1) Conditions cumulées sur les ordres faire, allera internes à U, et et les chargements de variables d'état (entrées internes à U).

α2) Conditions cumulées sur les ordres faire et allera dans une autre unité visant à forcer un état de U. (entrées également de U).

α3) Impulsions synchronisant les allera et chargement d'états affectées à l'unité contrôle (entrée d'horloge).

β) Sorties de l'unité de contrôle

Les expressions d'état de U peuvent servir à conditionner des opérations quelconques, or ces expressions ont été affectées à l'unité de contrôle, leur résultat en sera donc une sortie.

Les étiquettes qui sont transformées en conditions logiques dans l'unité opératoire de U et qui ont été placées dans l'unité de contrôle en seront également des sorties (sauf si elles ne portent que sur des opérations allera, faire ou chargement d'état qui sont internes à l'unité de contrôle).

Les ordres allera et faire qui servent éventuellement à forcer à partir de U des états dans une autre unité externe à U, créent eux aussi des sorties de l'unité de contrôle de U qui sont aussi des sorties de U.

Donc :

β 1) Etiquettes agissant sur l'unité opératoire (sorties de décodeur d'état).

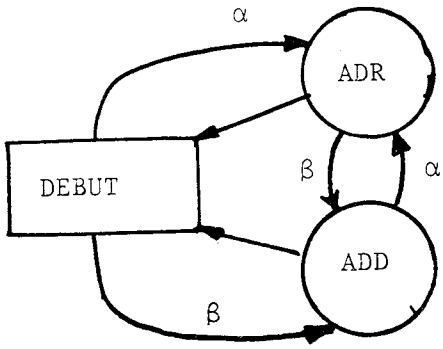
β 2) Signal de sortie d'une expression d'état (sortie d'état).

β 3) Conditions de forçage par allera ou faire d'un état dans une autre unité (sorties de forçage).

γ) Exemple : l'unité SEQUENCE de Z0001. Nous avons déjà dit que dans Z0001, qui est un calculateur entièrement défini (dont les automates sont déjà décrits par les circuits qui les réalisent) il ne reste plus d'unité séquentielle possédant d'étiquettes. Mais nous allons donner une nouvelle description de l'unité SEQUENCE, rigoureusement équivalente à celle de B, II, e mais contenant 3 états. De plus, nous supposerons que 2 de ces états peuvent être forcés à partir de l'unité CONTROLE.

```
'UNITE' SEQUENCE(H,I1,ADRDEBOUT(1:9),S(1:4),E1(1:4),CO'(1:2);
        ADMEMC(1:1,1:9));
  'REGISTRE' PLUS1,MOINS1,PLUS2,C,BITPAR,PILE(1:8,0:3);
  'HORLOGE' I1,H;
  'SIGNAL' Z,TC1,P(1:9),P2(1:4),X1,X2,X3,X4;
  'EXTERNE' COMPTEUR('HORLOGE',4,1,1,1,1;4,1);
        COMPTEUR|1|(I1,P(1:4),Z,X1,X2;ADMEMC(,1:4),);
        COMPTEUR|2|(I1,P(5:8),Z,X3,X4;ADMEMC(,5:8),TC1);
      X1:=PLUS1.TC1;
      X2:=MOINS1.TC1;
      X3:=PLUS2+PLUS1.BITPAR;
      X4:=MOINS1.-BITPAR;
      Z:=(SEQUENCE=ADD)+(SEQUENCE=ADD);
   'SI' PLUS1 'ALORS'
       (P(9):=BITPAR;<I1>BITPAR<=P(9));
    <H> PLUS1<=E(1),MOINS1<=E(2),PLUS2<=E(3),C<=E(4);
  DEBUT:
    <I1> 'SI' CON 'ALORS'
      (PILE(1:4,0)<=S)
      (PILE(5:8,0)<=S);
  ADR:
    P:=ADRDEBOUT;
    <I1> BITPAR<=P(9),
         'SI' C 'ALORS'
         PILE<=*P(ADMEMC(,1:8))*P(PILE(,1:3));
  ADD:
     P(1:8):=PILE(,0);
    <H> 'ALLERA' DEBUT;
    <I1> PILE(,0:2)<=PILE(,1:3);
```

Cette unité a un état DEBUT d'où elle ne sort pas, sauf action extérieure et deux états ADR et ADD qui vont à DEBUT.



Quel que soit l'état dans lequel elle se trouve les conditions venant de l'extérieur α et β peuvent la forcer dans les états ADR et ADD.

On va donc trouver dans l'unité CONTROLE :

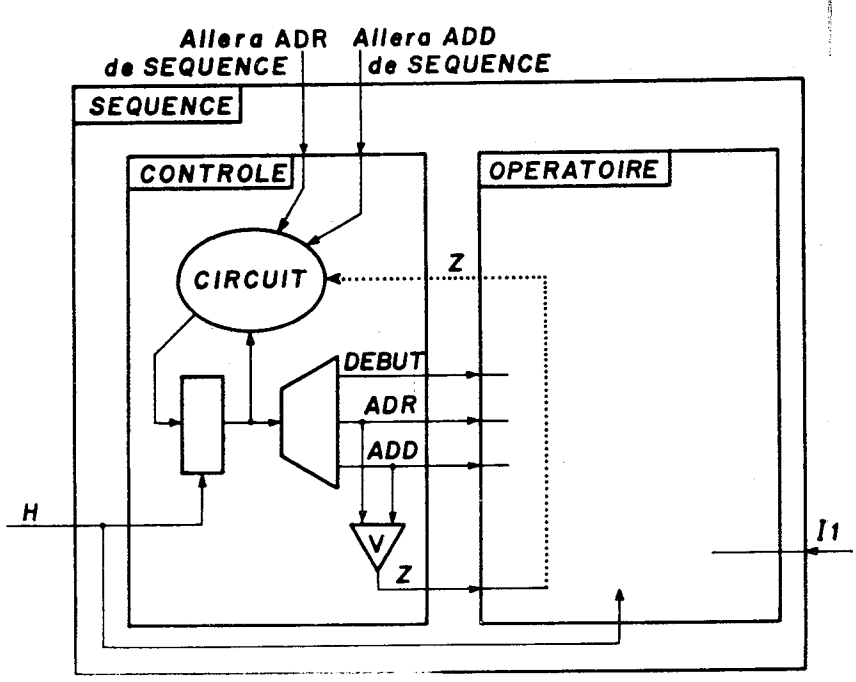
Externe SEQUENCE (ADD, ADR) ('horloge', 'horloge', 9,4,4,2 ; (1,9)) ;

- - - - - - - - - - - - - - - - - - - -

SEQUENCE (H, I1, ADRDEROUT, S, PLUS 1 & MOINS 1 & PLUS 2 & C, COMMANDE (14:15)
ADMEMC) ;

— — - - - - - - - - - - - - - - - - - - - —

<H > 'si' ADRR & ADDR 'alors' (allera ADR de SEQUENCE)
(allera ADD de SEQUENCE) ;



On peut voir dans la déclaration externe de SEQUENCE les 2 entrées d'état (allera ADR et allera ADD). Ces 2 entrées vont l'être aussi pour CONTROLE.

Les seules expressions d'état que l'on voit dans SEQUENCE concernent allera DEBUT. Ces deux ordres sont inconditionnels, donc ne vont pas créer d'entrée de CONTROLE en provenance de l'unité OPERATOIRE. Mais on pourrait aussi bien avoir, à la place des 2 allera DEBUT conditionnés par ADR et ADD, si Z alors allera DEBUT ; qui produirait bien le même effet. Mais dans cette hypothèse il y aurait dans l'unité OPERATOIRE une sortie allant vers l'unité CONTROLE, pour conditionner cet allera. (trait mixte). Comme par ailleurs Z est une sortie de CONTROLE vers OPERATOIRE, il y a création d'une boucle inutile, mais le programme ne peut le détecter, l'utilisateur au vue du résultat pourra corriger l'unité CONTROLE créée.

Le résultat est :

```
'UNITE' SEQUENCE (H,I1,ADRDEBOUT(1:S),S(1:4),E(1:4),FORCE000,FORCE001,
        CON'(1:2);/DMEMC(1:1,1:9));
  'HORLOGE' H,I1;
  'SIGNAL' ECOND000,SCOND001,SCOND002,SCOND003;
  'EXTERNE' CONTROLE('HORLOGE',1,1,1;1,1,1,1),
          OPERATOIRE('HORLOGE','HORLOGE',S,1,4,2,1,1,1,1;(1,9),1);
      CONTROLE(H,FORCE000,FORCE001,ECOND000;SCOND000,SCOND001,
              SCOND002,SCOND003);
      OPERATOIRE(H,I1,ADRDEBOUT,S,E,CON',SCOND000,SCOND001,SCOND002,
              SCOND003;/DMEMC,ECOND000);

'UNITE' CONTROLE(H,FORCE000,FORCE001,ECOND000;SCOND000,SCOND001,
                 SCOND002,SCOND003);
  'HORLOGE' H;
  'SIGNAL' Z;
      Z:=(CONTROLE=ADD)+(CONTROLE=ADR);
      SCOND000:=Z;
        <H> 'SI' FORCE000 'ALORS' 'ALLERA' ADR,
            'SI' FORCE001 'ALORS' 'ALLERA' ADD,
            'SI' ECOND000 'ALORS' 'ALLERA' DEBUT;
  DEBUT: SCOND001:=1;
  ADR  : SCOND002:=1;
  ADD  : SCOND003:=1;
```

Dans le cas où l'on n'a pas 'si' Z alors allera DEBUT, mais la description de la page C10 le signal ECOND000 n'est pas crée, il y a une entrée de moins à CONTROLE (et une sortie de moins à OPERATOIRE).

```
'UNITE' CONTROLE(H,FORCE000,FORCE001,ECOND000;SCOND000,SCOND001,
                 SCOND002,SCOND003);
  'HORLOGE' H;
  'SIGNAL' Z;
      Z:=(CONTROLE=ADD)+(CONTROLE=ADD);
      SCOND000:=Z;
        <H> 'SI' FORCE000 'ALORS' 'ALLERA' ADD,
            'SI' FORCE001 'ALORS' 'ALLERA' ADD;
  DEBUT: SCOND001:=1;
  ADR  : SCOND002:=1; <H> 'ALLERA' DEBUT;
  ADD  : SCOND003:=1; <H> 'ALLERA' DEBUT;
```

```
'UNITE' OPERATOIRE (H,I1,ADRDEBOUT(1:9),S(1:4),E(1:4),COM(1:2),Z,DEBUT,
                    ADR,ADD;ADMEMC(1:1,1:9));
  'HORLOGE' H,I1;
  'REGISTRE' PLUS1,MOINS1,PLUS2,C,BITPAR,PILE(1:8,0:3);
  'SIGNAL' TC1,P(1:9),P2(1:4),X1,X2,X3,X4;
  'EXTERNE' COMPTEUR('HORLOGE',4,1,1,1,1;4,1);
       COMPTEUR|1|(I1,P(1:4),Z,X1,X2;ADMEMC(,1:4),);
       COMPTEUR|2|(I1,P(5:8),Z,X3,X4;ADMEMC(,5:8),TC1);
      X1:=PLUS1.TC1;
      X2:=MOINS1.TC1;
      X3:=PLUS2+PLUS1.BITPAR;
      X4:=MOINS1.-BITPAR;
       'SI' PLUS1 'ALORS'
        (P(9):=BITPAR;<I1> BITPAR<=P(9));
        <H> PLUS1<=E(1),MOINS1<=E(2),PLUS2<=E(3),C<=E(4);
       'SI' DEBUT 'ALORS'
        (<I1> 'SI' CON 'ALORS'(PILE(1:4,0)<=S)(PILE(5:8,0)<=S));
       'SI' ADR 'ALORS'
        (P:=ADRDEBOUT;<I1> BITPAR<=P(9),'SI' C 'ALORS'
         PILE<=*P(ADMEMC(,1:8)%*P(PILE(,1:3)));
       'SI' ADD 'ALORS'
        (P(1:8):=PILE(,0);
        <I1> PILE(,0:2)<=PILE(,1:3));
```

Remarques :

Les unités créées ont été appelées CONTROLE et OPERATOIRE pour éclaircir les propos,en réalité le système génererait des noms d'unités UNITOOx , UNITOOy.

Dans OPERATOIRE les étiquettes d'état et le signal Z avec l'expression d'état qui le définissait ont disparus. On retrouve comme nouvelles entrées de l'unité les signaux qui leur correspondent.

Les états sont remplacés par des instructions conditionnelles englobant ce que les états conditionnaient à l'exception des ordres allera mis dans CONTROLE.

Dans CONTROLE on retrouve la déclaration et l'expression définissant Z. On génère autant d'instructions conditionnelles qu'il y a d'états à forcer à partir des entrées FORCE qui viennent de l'extérieur ou à partir des entrées ECOND qui viennent de OPERATOIRE.

Les états assurent seulement la transition à l'état suivant (qui peut être conditionnelle à partir d'entrées ECOND, voir LIDDELL (1) p.30) et qui valident des signaux de sortie SCOND destinés à OPERATOIRE.

2) Création des unités EXP :

Les expressions en CASSANDRE correspondent à des réseaux combinatoires dont les entrées sont les variables apparaissant dans l'expression et la sortie la valeur de l'expression. Comme les états disparaîtront de l'unité de CONTROLE d'une unité donnée, avant qu'on lui applique le traitement ici décrit, la sortie d'une expression sera soit un signal, soit une impulsion. Dans les deux cas le traitement appliqué sera le même. Il a été longuement décrit par LIDDELL (1) p.35à 54. nous y reviendrons dans la mesure où nos choix sont différents.

α) Expressions sans opérateur condition

Une expression à un niveau est une expression ne comportant pas de parenthèses. Les entrées de la boîte EXP que l'on crée sont les variables de l'expression. On se reporte à leur déclaration pour obtenir la dimension si elle n'est pas spécifiée localement.

Il n'est pas en général possible de déduire du contexte la dimension de la sortie. Si l'expression EXP se trouve en partie droite d'une affectation de registre ou d'une connexion, si EXP se trouve en entrée d'une connexion d'unité (une expression ne peut se trouver en sortie) alors la dimension de la sortie de EXP est celle du registre, du signal, de l'impulsion ou de l'entrée considérée.

Mais si EXP est une condition logique, dans un 'si' généralisé ce peut être un vecteur de dimension quelconque mais si EXP se trouve être une expression entre parenthèses plongée dans une autre expression plus grande (expression à plusieurs niveaux), le contexte ne définit pas forcément la dimension du résultat.

Exemple : dans l'unité COMPTEUR de Z0001 on trouve :

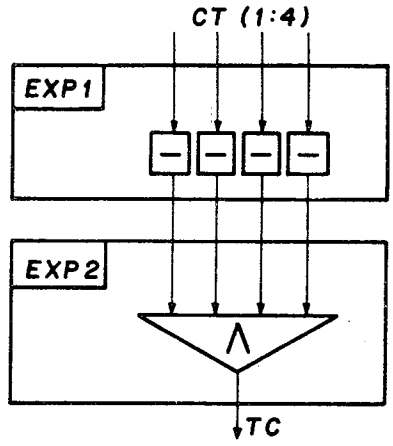
TC : = / Λ (-CT) ;

/ Λ (-CT) est une expression à 2 niveaux, au premier niveau on a -CT. L'entrée de cette expression est CT (1:4) que l'on trouve dans une déclaration Registre. Soit (q1, q2,...qn) les dimensions inconnues de la sortie du 1er niveau. Au 2e niveau /Λ EXP (1:q1, 1 :q2,...,1 :qn) aura pour dimensions en sortie (1:1, 1:q2,...1 :qn) et rien ne permet sans supplément d'analyse de connaître q1 (même si l'on connaît q2...qn grâce au membre gauche TC).

Dans un tel cas, seul l'interpréteur dimensionnel en traitant l'expression considérée pourra déterminer la dimension du résultat. Dans l'exemple, l'interprétation est triviale et donne q1 = 4, q2 = q3 = qn = 1. (les opérations auraient été correctes avec une autre valeur de q1).

D'où la structure :

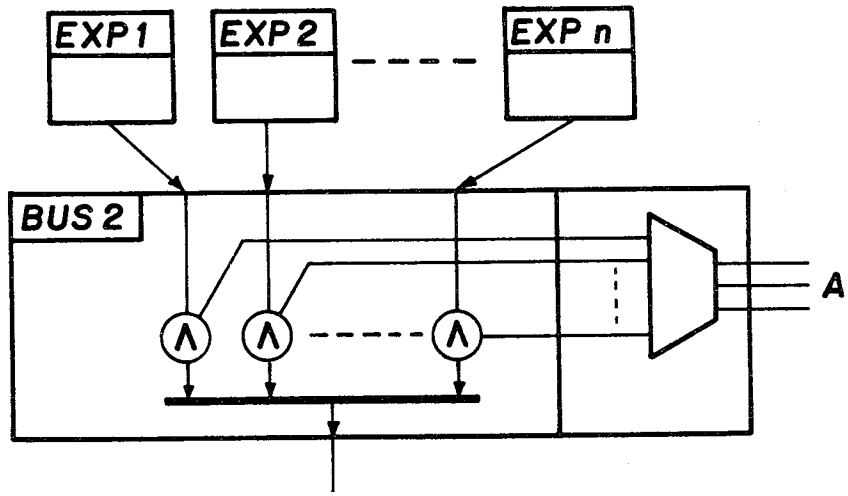
EXPOZOOO (EXPO1OOO(CT ; *) ; TC) ; correspondant au schéma ci-dessous.



β Traitement de l'opérateur condition

Nous distinguerons 2 cas suivant que l'opérateur comporte ou non un 'sinon' (seul le premier cas est traité par les programmes existant à ce jour).

Si $ A alors (<EXP1>) (<EXP2 >)...(<EXPn >) va être réalisé :



Dans le cas où les EXPi ont une sortie qui est un scalaire booléen.
Sinon si les dimensions des EXPi sont (1: q1, 1: q2...1: qn) on générera :

pour I = 1 à q1 début

pour J = 1 à qr début

pour N = 1 à qℓ début

BUS02000 f(I , J , ... , N)*(A (1:p), DONNEES (I,J,...,N, 1:n) ; S(I,J,...N)) ;

fin;

fin;

fin;

EXP101000 (, , ..., ; DONNEES (, , ..., 1)) ;

EXP201000 (, , ..., ; DONNEES (, , , ..., 2)) ;

EXPn01000 (, , ..., ; DONNEES (, , ... , n)) ;

* Voir LIDDELL (1) p. 15, ligne 19, pour le calcul de f.

Cas particulier : La condition S A se réduit à un scalaire, on retrouve le traitement effectué par Fantino (2) . Exemple :Signal ST (1:16) ;

si CSP alors ST ;

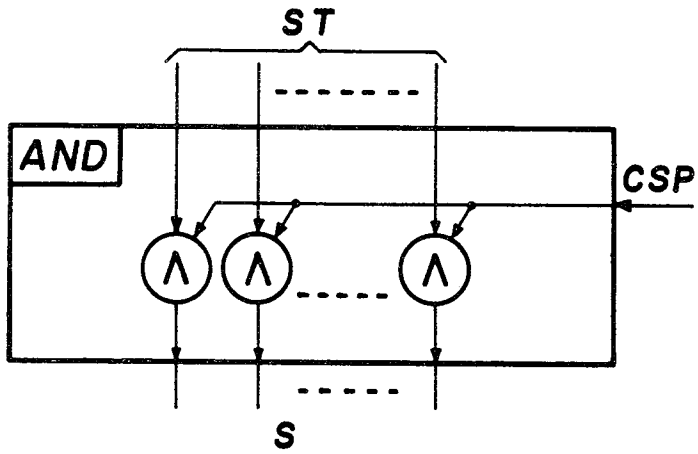
Donne : AND08002 (8) (CSP (1:1), ST (1:16) ; *) ;

Avec : Unité AND08002 (COND,E (0:15) ; S (0:15) ;

pour I = 0 à 15 début

S (I) : = COND Λ E (I) ;

fin;



L'opérateur-condition dans ce cas s'appelle AND.

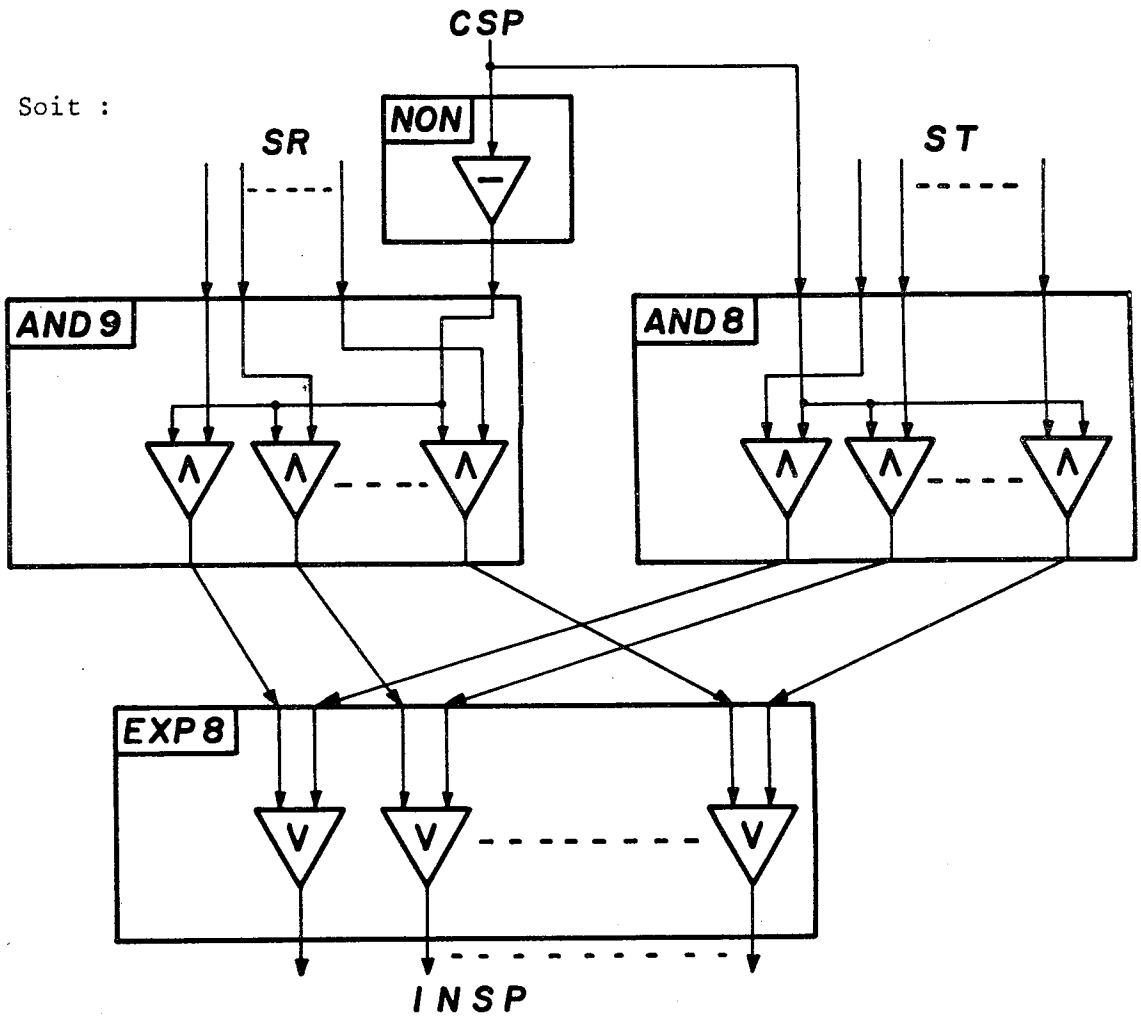
C'est seulement dans le cas où la condition est un scalaire qu'il peut y avoir un sinon. En effet, si la condition a plusieurs bits, un seul à la fois peut être égal à 1 sinon le résultat serait indéterminé. Il est bien clair que les conditions après le si étant deux à deux disjointes, leur complément après le sinon ne saurait l'être.

Exemple : On trouve dans l'unité SPES de Z0001, INSP : = si CSP alors ST sinon SR;

Après application des programmes de Fantino (2)on obtient :

EXP08001 (AND08002 |9| (NON |1| (CSP (1:1) ; *), SR (1:16) ; *),
AND08002 |8| (CSP (1:1), ST (1:16) ; *) ; INSP (1:16)) ;

Soit :



Mais comme la fonction de EXP8 n'est pas véritablement un OU logique, nous préfererons une réalisation à l'aide de boites CONNEXn (voir LIDDELL p 39) matérialisée par la connexion d'unité :

CONNEX16 (CSP, SR (1:16), ST (1:16) ; INSP)

Avec comme définition de CONNEX :

Unité CONNEXn (COND, DONNEE1 (1:n), DONNEE2 (1:n) ; SORTIE (1:n)) ;

Externe NON (1 ; 1),

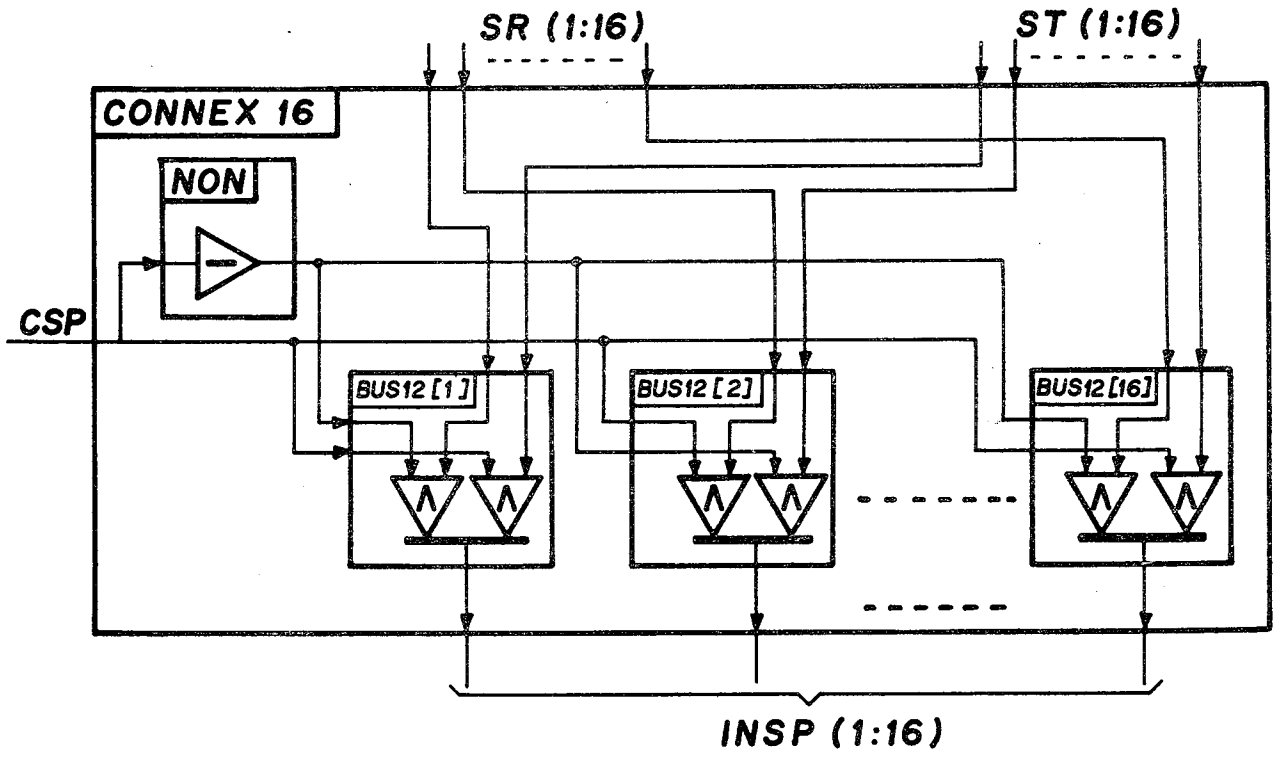
BUS12 (2, 2 ; 1) ;

pour I = 1 à n début

BUS12 |I| (COND&NON (COND ; *), DONNEE1 (I)& DONNEE2 (I); SORTIE (I)

fin ;

Soit :

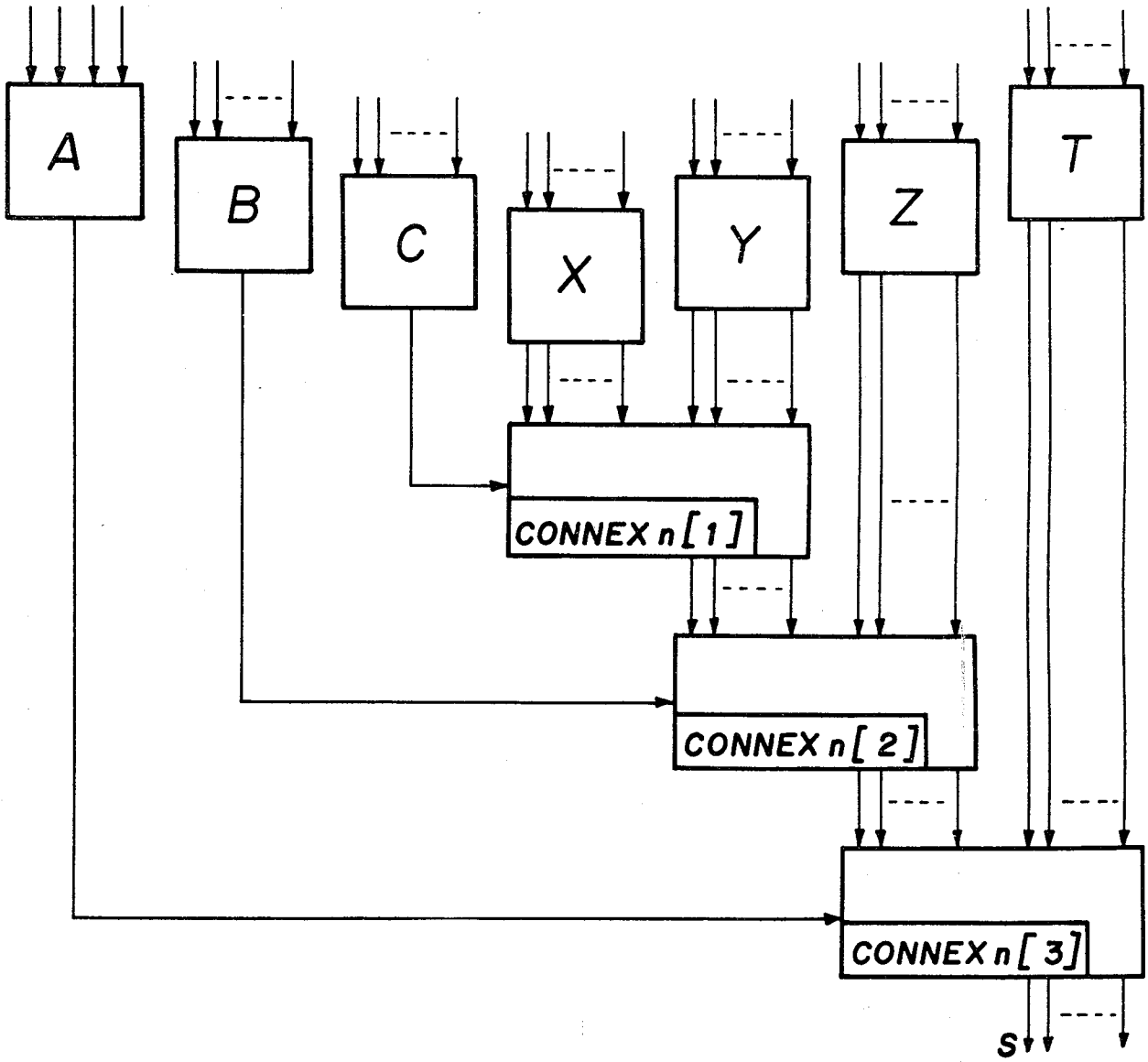


CONNEX, est donc un cas particulier de BUS (ou multiplexeur) à 2 entrées portant sur des opérandes de n bits.

γ) Opérateur condition dans un opérateur condition

Nous ferons le même choix que LIDDELL (1) pages 41-45 dont nous allons reprendre l'exemple page 44 , mais nous déciderons comme FANTINO (2)que "toutes les unités créées sont au même niveau, c'est à dire externe dans l'unité traitée". Ceci est très simple avec la boite CONNEX.

Exemple : S : = Si A alors (si B alors si C alors X sinon Y sinon Z) sinon T ;



3) Traitement des conditions :

Nous avons déjà parlé de conditions cumulées (sur les ordres allera et faire) au moment de la création de l'unité de contrôle d'une unité. En CASSANDRE on peut avoir une structure imbriquée d'instructions conditionnelles d'une complexité très grande, il importe cependant :

α1)de pouvoir réaliser pour chaque action élémentaire la condition résultante qui s'y applique.

α2) de ne pas réaliser 2 fois si possible la même condition cumulée si elle s'applique à plusieurs actions élémentaires.

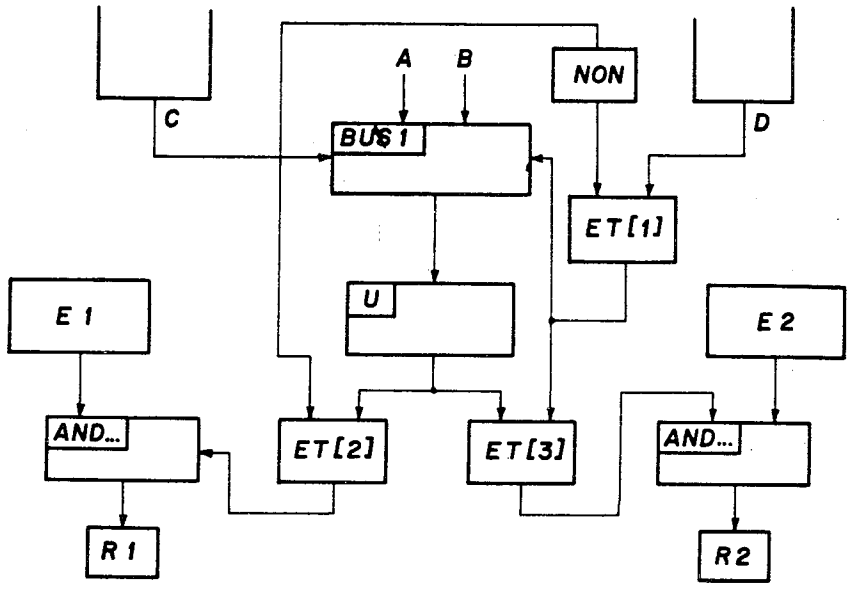
Nous avons vu aussi que les actions élémentaires sont soit de type A (affectation de registre) soit de type B (branchement). Dans le premier cas une horloge agit en même temps que la condition cumulée, toutes deux et aussi l'expression qui intervient dans l'affectation seront branchées sur les entrées de l'unité primitive MCELL réalisant le registre. Dans le second cas la condition cumulée et l'expression à connecter seront les entrées d'une boite de type BUS1 si on connecte plusieurs quantités au même point, ou de type AND si on n'en connecte qu'une.

β) Cas des connexions d'unités ou des signaux-unités

Nous avons dit que l'occurence sous la portée d'une condition d'un signal-unité ou d'une connexion d'unité provoque le conditionnement à la fois des entrées et des sorties de l'unité, c'est indispensable.

Exemple : 'si' C alors ('si' U (A ; *) alors R1 ← E1) 'sinon' ('si' D 'alors' 'si' U (B ; *) alors R2 ← E2) ;

va donner comme réalisation :

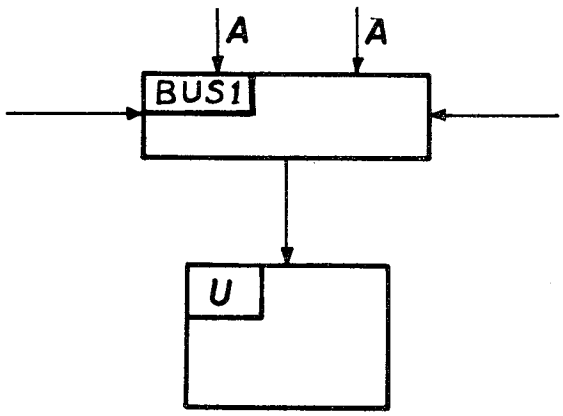


On voit aisément dans ce cas qu'il n'y a aucune redondance et que tout le matériel généré est indispensable au bon fonctionnement. Mais supposons maintenant que l'on ait en :

'si' C 'alors' ('si' U (A; *) 'alors' R1 ← E1)

'sinon' ('si' D 'alors' 'si' U (A; *) 'alors' RE ← E2

Comme précédemment, le conditionnement des entrées du signal-unité U va provoquer la réalisation de :



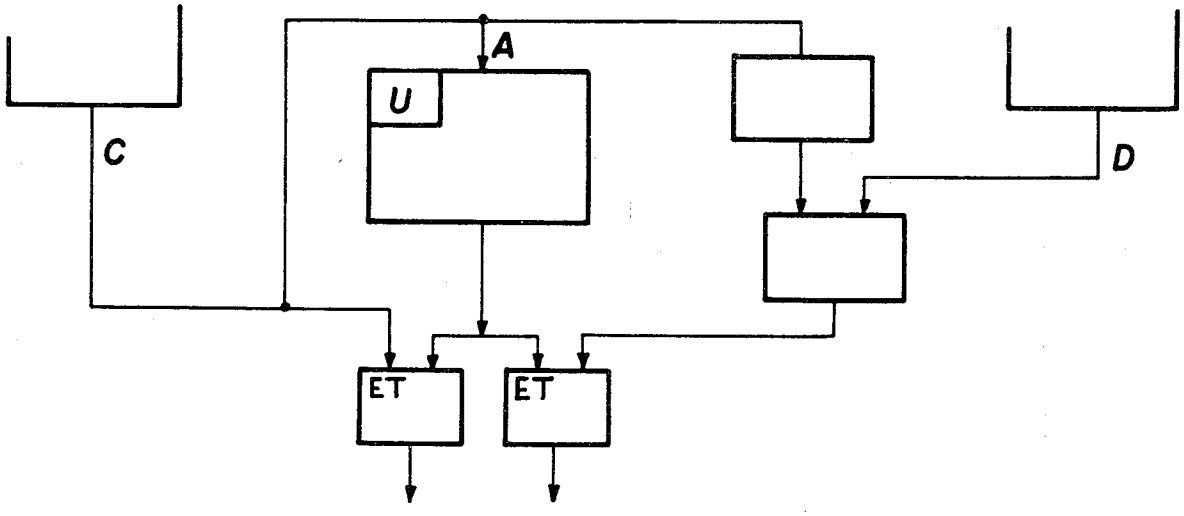
qui est cette fois parfaitement inutile. Pour éviter cela c'est à l'utilisateur d'écrire sa description :

U (A;) ;

'si' C 'alors' ('si' U (; *) 'alors' R1 ← E1) 'sinon'

('si' D 'alors' 'si' U (; *) 'alors' R2 ← E2) ;

Dont la réalisation donnera :



qui est non redondant.

γ) L'occurence d'une expression dans la description traitée provoque la création d'une unité EXP, c'est aussi le cas quand le résultat de l'expression est une condition logique. Une fois générée la connexion de l'unité EXPn créee on remplace la condition par le signal CNDn de sortie de EXPn qui est soit un scalaire soit un vecteur, suivant que l'on se trouve dans une instruction conditionnelle simple ou généralisée.

Exemple : Dans l'unité UAL de Z0001 on trouve :

'si' / Λ COM (0:1) & RAZ & SET 'alors'...

Après traitement UAL a généré UNIT 4 dans laquelle on trouve :(p.C74)

Externe EXP04035 (2,1,1; 2) ;

EXP04035 (COM (0:1), RAZ (1:1), SET (1:1); CND04001 (0:2)) ;

BUS04001 (ET |2| (CND04001 (0), RS4 (0) ; *) ; ET |3|...etc.

Une fois créee EXP04031, c'est sa sortie CND04001 qui est utilisée partout comme condition. On a, par ailleurs :

'Unité' EXP04031 (E000 (0:1), E001 (1:1), E002 (1:1) ; S (0:2) ;

S : = / Λ E000 &E0001 & E0002 ;

L'occurence d'une autre condition sous la portée de la première va créer, une autre unité EXP et les sorties des deux unités réalisant les conditions entreront dans une unité ET ou AND suivant que les deux conditions sont scalaires ou que la deuxième est un vecteur. La connexion de la deuxième unité EXP doit obligatoirement se faire hors de portée de la première condition, pour les raisons citées en β).

De la même façon l'occurence d'un sinon va créer une unité NONn ayant en entrée le signal CNDn crée lors de l'occurence du 'si' correspondant, et en sortie un nouveau signal CND n+1. Le 'sinon' est alors traité comme si CND n+1 alors.

Exemple : Dans l'unité DEC de Z0001, on trouve :

'si' CDEC 'alors' (C01 : = 1; 'si' VALN2 (; *) 'alors' (C02 : = 00) (C02 : = 11)(C02: = 10) (C02 : = 01)) 'sinon'('si' VALN2 (; *) alors (C01 : = 00) (C01 : = 11) (C01 : = 10) (C01 : = 01) ; C02 : = A) ; et par ailleurs :

VALN2 (A ; *) ;

Après traitement on obtient dans UNIT7 : (voir FANTINO (2), on obtient page C79, un résultat équivalent) :

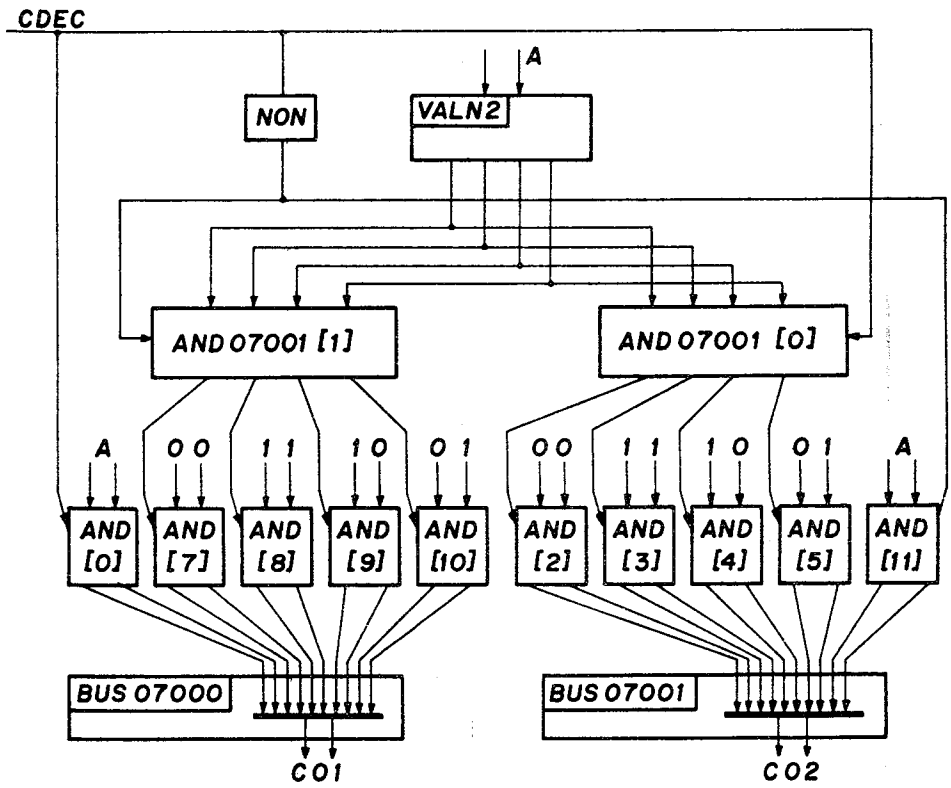
VALN (A,) ; NON |1| (CDEC ; CND07003) ;

AND07001 |0| (CDEC (1), VALN2 (; *) CND 07000(0:3)) ;

AND07001 |1| (CND07003, VALN2 (; *) ; CND07001 (0:3)) ;

BUS 07001 (AND07000 |2| (CND07000 (0) , 00 ; *), AND07000 |3| (CND07000 (1), 11 ; *), AND07000 |4| (CND07000 (2), 10 ; *), AND07000 |5| (CND07000 (3), 01 ; *), AND07000 |11| (CND07003, A (1:2) ; *); CO2 (1:2)) ;

BUS07000 (AND07000 |0| CDEC, A (1,2) ; *), AND07000 |7| (CND07001 (0), 00, *) ; AND07000 |8| (CND07001 (1), 11 ; *), AND07000 |9| (CND07001 (2), 10 ; *), AND07000 |10| (CND07001 (3), 01 ; *) ; CO1 (1:2)) ; ce qui correspond à :



Cette réalisation est sans gaspillage de matériel qui puisse être évité. (les 0 et 1 en entrée des boites AND pourraient aussi bien être des variables booléennes).

4) Traitement des registres

Au cours du traitement tous les registres sont transformés en unités, c'est MCELL unité primitive si le registre n'a qu'un bit ou une unité REG s'il en a plusieurs (l'unité REG étant elle-même réalisée à l'aide d'unités MCELL, voir Fantino (2)).

Exemple : Dans l'unité ML de Z0001, on trouve la déclaration :

Registre M(1:4, 0:15),dans l'unité UNIT2 produite, la déclaration de registre est remplacée par une déclaration :

Externe REG02000 ('horloge', (4,16), (4,16) ; (4, 16)) ; dont les entrées correspondent aux 16 x 4 conditions de chargement et aux 16 x 4 bits de donnée.

On a par ailleurs la définition : (page C69)

```
Unité REG02000 (H, ENABLE (0:3,0:15), INFO (0:3, 0:15) ; OUTPUT (0:3,
                                                          0:15)) ;

Horloge  H ;

Externe MCELL ('horloge',, ;) ;

Pour I = 0 'a' 3 'début'

'Pour' J = 0 'à' 15 'début'

MCELL |16 x I + J | (H, ENABLE (I,J), INFO (I,J); OUTPUT (I,J)) ;

                              'fin' ;

                    'fin'  ;
```

qui génère les 64 boites primitives MCELL qu'il faut pour réaliser la mémoire ML (voir exemple AId page 7 pour les boucles pour).

5) Création de bus et d'expanseurs

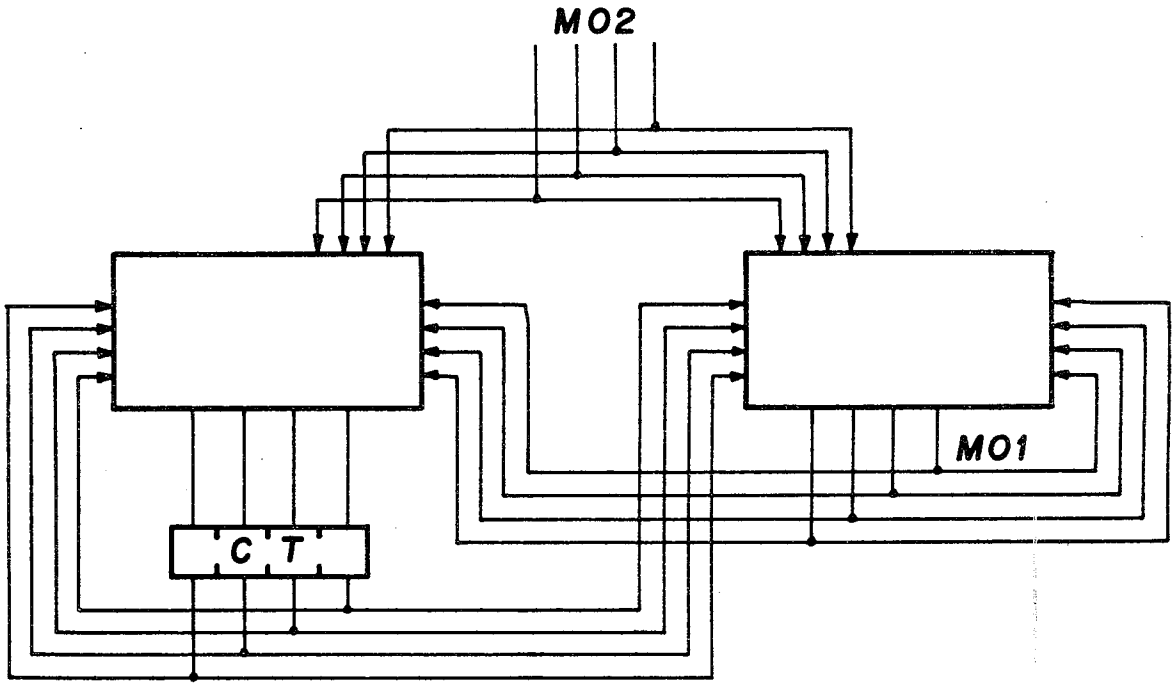
Il y a expanseur chaque fois qu'une même variable est utilisée à plusieurs endroits.

Exemple : Dans l'unité COMPTEUR de Z0001 on trouve :

Signal MO2 (0:3), MO1 (0:3); Registre CT (1:4) ;

MO1 : = * D |1 | (CT. MO2 + CT. MO1 + MO1. MO2)&0 ;

et plus loin : CT ← CT ≠ (MO1 ≠ MO2) ;

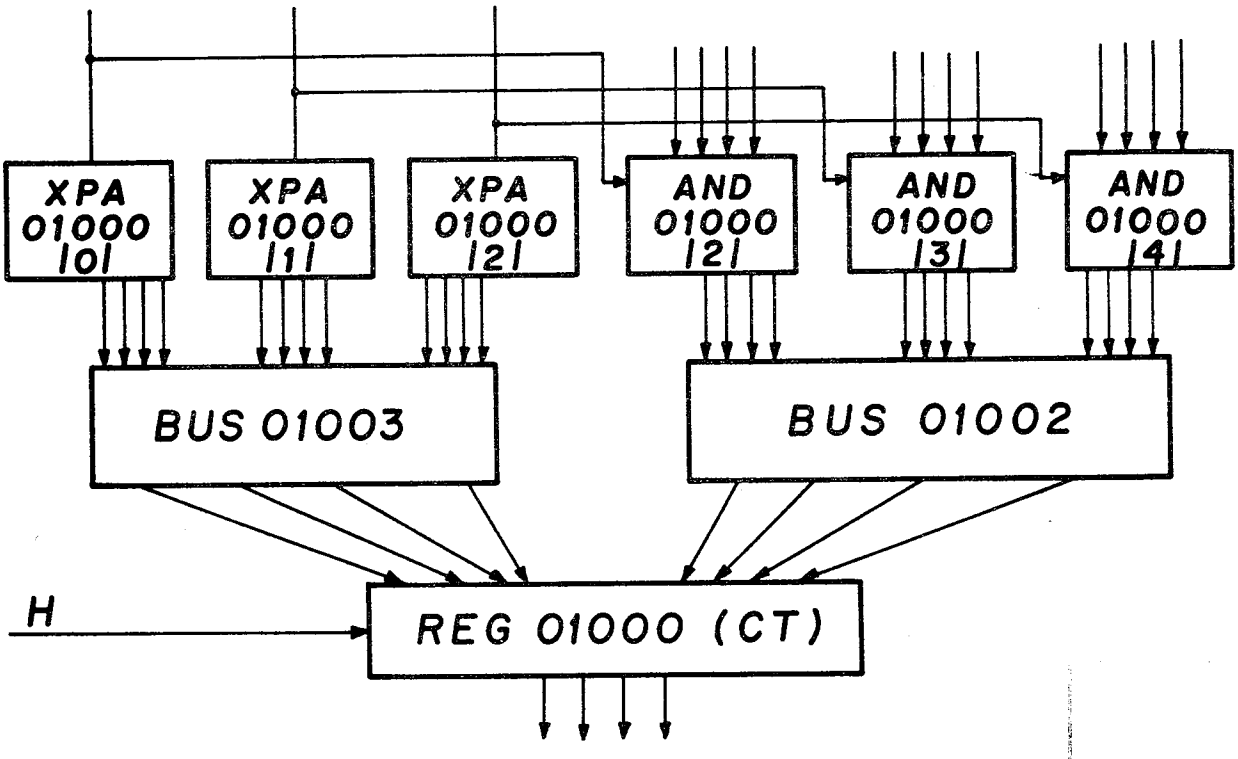


On voit 12 expanseurs à 2 branches. Mais on peut distinguer 2 cas dans leur emploi.

α) La même variable est utilisée plusieurs fois dans différentes expressions avec d'autres variables de dimensions comparables (exemple précédent), il n'y a pas génération d'unité XPA, on se contente de connecter les même fils de sortie (d'expression ou de registre) aux entrées des expressions qui les utilisent.

β) La même variable est utilisée avec chacune des composantes d'autres variables de dimensions supérieures aux siennes. C'est en général le cas pour une expression conditionnelle. Les programmes de traitement génèrent une unité XPA qui rend compatibles les variables conditionnées et les variables de condition.

Exemple : Toujours dans l'unité COMPTEUR après traitement on obtient :



La boite REG (qui représente ici CT) est constituée de boites MCELL (ayant en entrée H, une DONNEE et une CONDITION). Or il y a 4 bits de donnée donc 4 boites MCELL, il faut alors 4 conditions, c'est ce que produisent les 3 boites XPA01000 et la boite BUS01003 (qui n'est pas un bus comme nous l'avons défini mais qui simule sa fonction à l'aide d'un OU logique) en effet,

Unité BUS01003 (E000 (1:4), E001 (1:4), E002 (1:4); S (1:4)) ;

S : = E000 V E001 V E002 ;

On a par ailleurs :

Unité XPA01000 (IN; OUT (0:3));

'Pour' I = 0 'a' 3 'début'

OUT (I) : = IN; 'fin'

description déjà donnée.

La différence de traitement entre les 2 catégories d'expanseurs est arbitraire, car ils sont de même nature. Mais dans le 2e cas il y a création d'une boite (sans fonction logique)associée à la distribution d'une condition qui aura en général beaucoup plus de sorties que celle que l'on pourrait créer dans le premier cas. Les boites XPA peuvent être utiles dans la formalisation d'un réseau logique et le codage des microinstructions. Elles peuvent aussi subsister dans la réalisation effective.

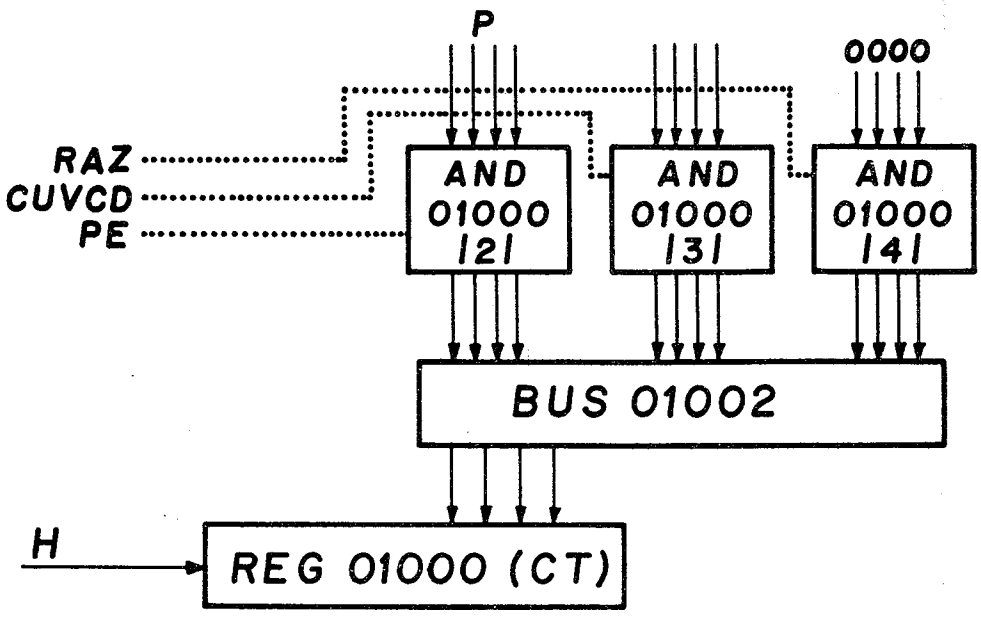
Nous avons déjà vu que les bus sont étroitement liés à la notion de conditions disjointes.

Or en CASSANDRE, une variable ne peut se voir affecter des quantités différentes que sous des conditions disjointes. Il y aura donc un bus (ou encore multiplexeur) en entrée de toute variable qui est en partie gauche de plusieurs affectations, c'est le cas, déjà vu, lorsque la partie droite est un "opérateur-condition".

Exemple : Reprenons encore l'unité COMPTEUR.

On trouve : <H> 'si' PE & (CUVCD) & RAZ 'alors'
(CT ← P) (CT ← CT ≠ (MO1 ≠ MO2)) (CT ← 0000) ;

Après traitement :



Dans ce cas on trouve localement les différentes valeurs affectées à CT qui sont dans un 'si' généralisé. En règle générale, c'est seulement à la fin du traitement d'une unité que l'on connaîtra toutes les quantités affectées à une même variable (donc générant un bus).

Une difficulté se présente lorsqu'on affecte une sous variable d'une variable donnée, soulignée par LIDDELL (1)dans sa thèse page 66. FANTINO la résoud d'une façon générale, mais qui risque de fabriquer du matériel superflu et donc demande à être complètée par un programme de simplification.

Registre R (1:8);

Exemple : < H > si C 1 alors R (3:7) ← E1 sinon (si C 2 alors R(5:8) ← E2) sinon R(1:4) ← E3) , si C3 alors R ← E4 ;

Cette expression amène à considérer 4 parties de R : R (1:2), R (3:4), R (5:7), R (8)

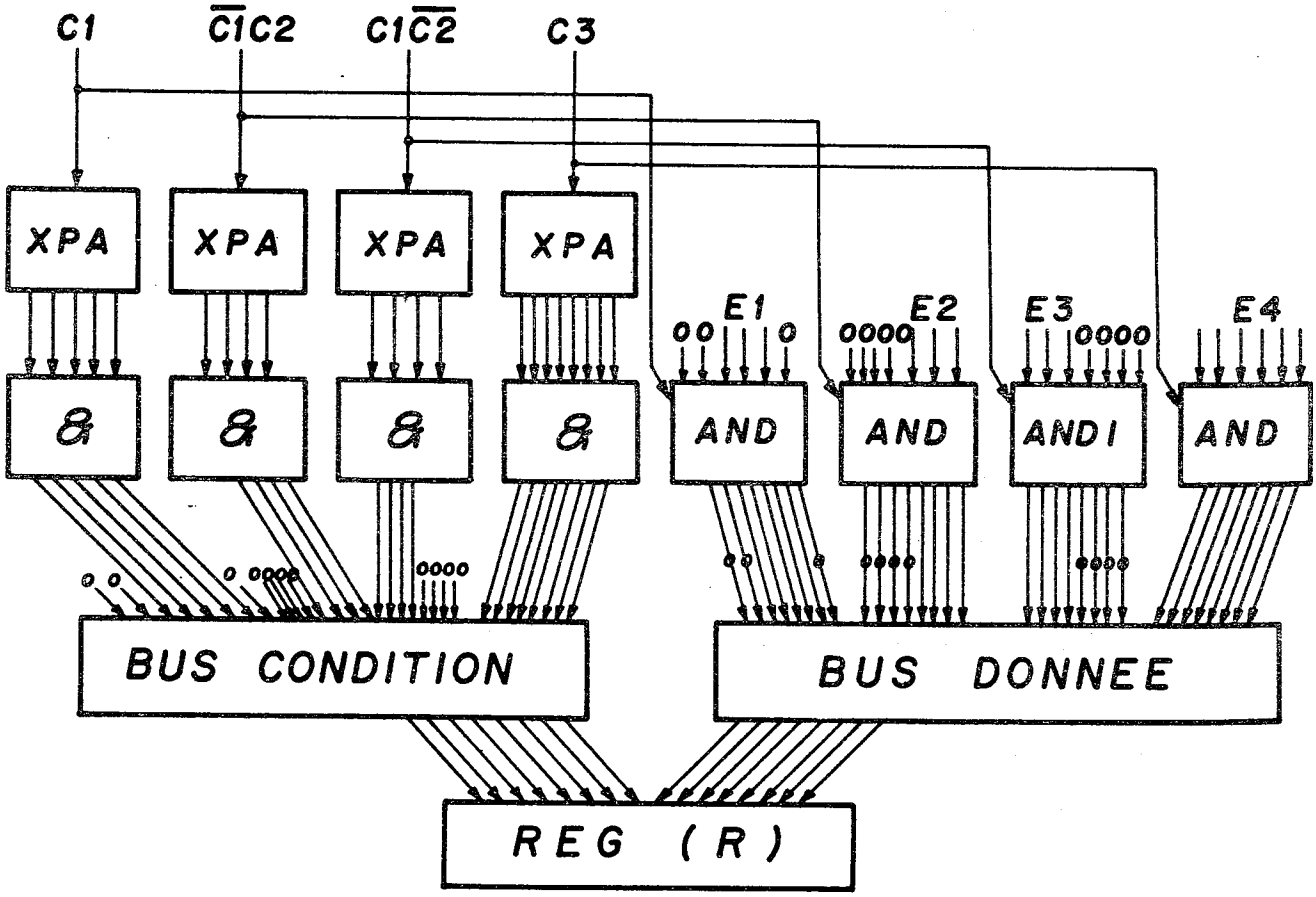
R (1:2) reçoit : E3 et E4, R (3:4) : E1, E3 et E4,

R (5:7) : E1, E2 et E4, et R8 reçoit E2 et E4.

Il n'est pas possible de construire une structure qui ne redescende au niveau du bit, ce qui fait perdre l'un des intérêts majeurs de CASSANDRE.

L'idée du traitement réalisé repose sur le fait qu'une variable de type registre n'est chargée que si la condition de chargement est vrai, sinon on peut lui affecter n'importe quoi en entrée (par exemple 0) ce ne sera pas chargé.

Les expressions et les conditions de chargement sont alors complètées par des 0 pour avoir la dimension de la variable affectée complète, soit dans l'exemple précédent :



Il s'agira de faire disparaître ensuite un certain nombre de portes ET ayant des 0 en entrée. Ce procédé a l'avantage d'être général et facile à programmer.

II. Relations avec les méthodes théoriques de synthèse booléenne et de codage

a) Réalisation des unités EXP :

D'après ce qui précède ces boîtes ont un nombre arbitraire d'entrées et une seule sortie S.

Intérieurement elles ne contiennent qu'un ordre de connexion, S : = < expression >.

On pourra le vérifier sur les nombreuses boites EXP crées par les programmes Fantino, traitant l'exemple de Z0001, listées ci-après dans III.

Par définition de EXP, l'expression contenue dans sa description n'a pas de parenthèse (1 seul niveau) ni d'opérateur condition, mais elle peut contenir les opérateurs de transposition ∿ , concaténation & (exemple EXP 010 02) de troncature ↕ (exemple EXP04012) de rotation ↑, de réduction / (exemple EXP01036) de négation - (exemple EXP01027) et tous les opérateurs binaires booléens (exemple EXP04025).

Propriété : "Une unité EXP qui ne contient que les opérateurs ∧ , V , -, est toujours sous forme d'une somme disjonctive de mônomes booléens (circuit bicouche)". (exemple EXP04013).

En effet, comme en CASSANDRE, les priorités de ces opérateurs sont celles employées habituellement, s'il n'en était pas ainsi, il y aurait obligatoirement des parenthèses.

α) Passage à une forme parenthèse

En analysant l'expression contenue dans EXP et en tenant compte des priorités des opérateurs en CASSANDRE, on met la dite expression sous forme totalement parenthésée.

Les priorités sont de la plus forte à la plus faible ∿ ; & ; - ; ∀, ∧ ; =, ≠, >, ≥, <, ≤ ; ∧ , V ; /, ↕ , ↑ ; soit 7 niveaux, le plus faible étant celui des opérateurs réduction troncature et rotation puisqu'ils portent sur tout ce qui se trouve à leur droite.

Exemple : S : = A & $\sim$ B $\wedge$ C V D $\neq$ E $\wedge$ F;

donnerait : S : = ((A & ($\sim$ B)) $\wedge$ C) V (D $\neq$ (E $\wedge$ F))

β) Equivalence d'une forme totalement parenthésée, avec une structure emboitée de signaux-unité

Au lieu d'introduire comme précédemment des parenthèses dans l'expression qui n'en avait pas, on remplace chaque opérateur par une nouvelle boite, connectée sous forme de signal-unité.

Exemple : en reprenant l'expression de α on obtient :

S : = GOU ox_1ooy_1 (GET ox_2ooy_2 |1| (ox_3ooy_3 (TRS ox_4ooy_4 (B; *),

A ; *), C ; *), DSJ ox_5ooy_5 (D, GET ox_2ooy_2 |2| (E, F ; *) ; *) ; *);

En introduisant bien sur les déclarations :

Externe : GOU ox_1ooy_1 ((n, p), (n, p) ; (n, p)),

GET ox_2ooy_2 ((n, p), (n, p) ; (n, p)),

CCT ox_3ooy_3 ((q, p), (r, p) ; (n, p)),

TRS ox_4ooy_4 (p, r) ; (r, p)),

DSJ ox_5ooy_5 ((n, p), (n, p) ; (n, p)) ;

Remarques : β1) La boite de type GET ox_2ooy_2 est utilisée en 2 exemplaires.

β2) Les boites remplaçant chaque opérateur sont des unités EXP de type particulier.

β3) Une structure imbriquée de signaux-unités de type EXP, est une expression totalement parenthésée en CASSANDRE-E, équivalente sémantique d'une expression booléenne de CASSANDRE-G sans signaux-unité.

γ) Description des unités-opérateurs EXP

γ1) Transposition

Soit A une variable de dimensions (d1, d2, ... dn) (p, q) ∿ A génère une boite :

Unité TRS oxooy (A (1:d1,...,1:dp,...,1:dq, ...,1:dn) ;
S (1:d1, ...,1:dq, ...,1:dp, ...,1:dn)) ;

Pour I = 1 a dp début

Pour J = 1 a dq début

S (,..., J,..., I,...,) : = A (,..., I,..., J,...,) ;

fin ; fin ;

γ2) Concaténation :

Soit A une variable de dimensions (d1, d2,..., dn)
et B " " " (d0, d2,..., dn),

avec s = d0 + d1, l'expression A & B génère une boite :

Unité CCT oxooy (A (1:d1, 1:d2,..., 1:dn), B (1:d0, 1:d2,..., 1:dn) ;

S (1:s, 1:d2,..., 1:dn)) ;

S (1:d1,,...,) : = A ;

S (d1+1:s, , ...,) : = B ;

γ3) Troncature :

Soit A une variable de dimensions (d1, d2,..., dn) p ↕ A va créer la boite (si p est positif) :

Unité TNK oxooy (A (1:d1, 1:d2, ..., 1:dn) ; S (1:d1-p, 1:d2,.., 1:dn));

S : = A (p+1:d1, , ...,) ;

Et si p est négatif :

Unité TNK (A (1:d1, 1:d2,..., 1:dn) ; S (-p:d1,1:d2,..., 1:dn)) ;

S : = A (1:d1+p, , ...,) ;

γ4) Rotation :

De la même façon p A va créer soit :

Unité ROT oxooy ((A (1:d1,1:d2, ..., 1:dn) ; S (1:d1,1:d2,..., 1:dn)) ;

S (1:d1-p, ,...,) : = A (p+1:d1, , ...,) ;

S (d1+p+1 : d1, , ...,) : = A (1:p, , ...,) ;

si p est positif sinon :

Unité ROT oxooy (A (1:d1,1:d2,..., 1:dn) ; S (1:d1,1:d2,..., 1:dn)) ;

S (-p+1:d1, , ...,) : = A (1:d1+p, ,...,) ;

S (1:-p, ,...,) : = A (d1+p+1:d1, , ...,) ;

γ5) Réduction

Pour la réalisation de la réduction par rapport aux opérateurs >, >, < , < , =, ≠, nous renvoyons à LIDDELL (1) p 16-20.

En reprenant la variable A, l'occurence de /Λ A va générer :

Unité RET 2d1 (A (1:d1,1:d2,...,1:dn) ; SORTIE (1:d2,1:d3,...,1:dn)) ;

Externe GET ((d2,d3,...,dn), (d2,d3,...,dn) ; (d2,d3,...,dn)) ;

SORTIE : = GET |1| (A (1,,...,), GET |2| (A (2,,..,), GET |3| ...,

GET |d1-1|(A (d1-1,,...,), A (d1,,...,) ; *) ; *)... ; *)

Dans le cas où d2 = d3 = ... =dn = 0 on retrouve l'unité primitive ET 2 d1 déjà rencontrée.

De même l'occurence / V A, va générer :

Unité ROU d1 (A (1:d1,1:d2,...,1:dn) ; SORTIE (1:d2,1:d3,...,1:dn)) ;

Externe GOU ((d2,d3...,dn, d2,d3,...,dn ; d2,d3,...,dn) ;

SORTIE : = GOU |1| (A (1,,...,), GOU |2| (A (2,,...,), GOU |3| ....
GOU |d_1 -1| A (d_1-1,,...,), A (d_1,,...,) ; *)...; *) ;

qui redonne une unité primitive OU d1 si A n'a qu'une dimension.

γ6) Opérateurs de relation, NI et NAND ;

A > B, A ≥ B, A < B, A ≤ B, A = B, A ≠ B, A ↓ B, A ↑ B vont produire respectivement :

Unité SUP (A (1:d1,1:d2,...,1:dn), B (1:d1,1:d2,...,1:dn) ; SORTIE (:d1, 1:d2,....,1:dn)) ;

Externe GET ((d1,d2,...,dn), (d1,d2,...,dn) ; (d1,d2,...,dn));

GNO ((d1,d2,...,dn) ; (d1,d2,...,dn)) ;

SORTIE : = GET (A, GNO (B ; *) ; *) ;

Unité INF (A (1:d1,1:d2,...,1:dn), B (1:d1,1:d2,...,1:dn) ; SORTIE (1:d1, 1:d2,...,1:dn)) ;

Externe GET ((d1,d2,...,dn), (d1,d2,...,dn) ; (d1,d2,...,dn)),

GNO ((d1,d2,...,dn) ; (d1,d2,...,dn)) ;

SORTIE : = GET (GNO (A ; *), B ; *) ;

Unité NIN (A(1:d1,1:d2,...,1:dn), B(1:d1,1:d2,...,1:dn);SORTIE(1:d1,1:d2, ...,1:dn)) ;
Externe GOU ((d1,d2,...,dn), (d1,d2,...,dn) ; (d1,d2,...,dn)),
GNO ((d1,D2,...,dn) ; (d1,d2,...,dn)) ;

SORTIE : = GOU (A, GNO (B ; *) ; *) ;

Unité NSU (A(1:d1,1:d2,...,1:dn), B(1:d1,1:d2,...,1:dn);SORTIE(1:d1,1:d2, ...,1:dn)) ;
Externe GOU ((d1,d2,...,dn), (d1,d2,...,dn) ; (d1,d2,...,dn)),

GNO ((d1,d2,...,dn) ; (d1,d2,...,dn) ;

SORTIE = GOU (GNO (A ; *), B ; *) ;

Unité EGA (A(1:d1,1:d2,...,1:dn), B(1:d1,1:d2,...,1:dn);SORTIE(1:d1,1:d2, ...,1:dn)) ;
Externe : GOU ((d1,d2,...,dn), (d1,d2,...,dn) ; (d1,d2,...,dn)),

GET ((d1,d2,...,dn), (d1,d2,...,dn) ; (d1,d2,...,dn)),

GNO ((d1,d2,...,dn) ; (d1,d2,...,dn) ;

SORTIE : = GOU (GET|1| (A,B ; *), GET|2| (GNO|1| (A ; *), GNO|2|(B ; *) ; *) ; *) ;

```
Unité DIF (A(1:d1,1:d2,...,1:dn), B(1:d1,1:d2,...,1:dn);SORTIE(1:d1,1:d2,
          ...,1:dn)) ;
Externe : GOU ((d1,d2,...,dn), (d1,d2,...,dn) ; (d1,d2,...dn)),

          GET ((d1,d2,...,dn), (d1,d2,...,dn) ; (d1,d2,...,dn)),

          GNO ((d1,d2,...dn) ; (d1,d2,...,dn) ;

SORTIE : = GOU (GET(1) (A, GNO( ) (B ; *), GET(2) (A ; *), B ; *) ; *) ;

Unité  NI (A(1:d1,1:d2,...1:dn), B(1:d1,1:d2,...,1:dn);SORTIE(1:d1,1:d2,
          ...,1:dn)) ;
Externe :  GET ((d1,d2,...dn), (d1,d2,...,dn) ; (d1,d2,...,dn)),

           GNO ((d1,d2,...dn) ; (d1,d2,...dn) ;

SORTIE : = GET (GNO (1) (A ; *), GNO (2) (B ; *) ; *) ;

Unité NAND (A(1:d1,1:d2,...,1:dn), B(1:d1,1:d2,...,1:dn);SORTIE(1:d1,1:d2
          ,...,1:dn)) ;
Externe : GOU ((d1,d2,...,dn), (d1,d2,...,dn) ; (d1,d2,...,dn)),

          GNO ((d1,d2,...,dn) ; (d1,d2,...,dn) ;

SORTIE : = GOU (GNO |1| (A ; *) ; GNO |2| (B ; *) ; *) ;
```

γ7) Opérateurs ET, OU et NON

Les occurences de A ∧B, A ∨B et - A génèrent respectivement :

```
Unité  GET (A (1:d1,1:d2,..., 1:dn), B (1:d1,1:d2,..., 1:dn) ;

       SORTIE (1:d1,1:d2,..., 1:dn)) ;

Externe ET2 (1,1; 1) ;

Pour I = 1 à d1 début

      Pour J = 1 à d2 début

           Pour N = 1 à dn début

SORTIE (I,J,...,N) : = ET2|*|(A (I,J,...,N), B (I,J,...,N) ; *) ;

                                fin ;

                           fin ;

                      fin ;
```

```
Unité GOU (A(1:d1,1:d2,...,1:dn), B(1:d1,1:d2,...,1:dn);SORTIE(1:d1,1:d2,
          ...,1:dn)) ;
Externe OU2 (2 ; 1)

Pour I = 1 a d1 début

SORTIE (I,J,...,N) : = OU2 |*|        (A (I,J,...,N), B (I,J,...,N) ; *);

                     fin ;         fin ;..... fin ;

Unité GNO ((d1,d2,...,dn) ; (d1,d2,...,dn) ;

Externe NON (1 ; 1) ;

Pour J = 1 à d2 début

    Pour N = 1 à d2 début

SORTIE (I,J,...,N) : = NON  |*|        (A (I,J,...,N) ; *) ;

                     fin ;    fin ; .....; fin ;
```

Dans le cas où d1 = 1, d2 = d3 = ... = dn = 0 on retrouve les unités promitives ET22, OU2, NON.

* Voir LIDDELL (1) page 15 ligne 19.

b) Réalisation de l'unité de contrôle d'une unité

Nous avons vu comment regrouper la partie séquentielle d'une unité lorsqu'elle existe, à l'intérieur d'une sous-unité "contrôle", nous allons indiquer comment cette dernière pourrait être réalisée, et quelle solution est employée dans le système actuel.

1) Méthode de Gerace

Nous avons emprunté à Gerace la division d'un système logique en partie contrôle et partie opératoire.

Dans l'article (5), une méthode pour réaliser chacune de ces parties comme un système séquentiel est indiquée. Cette méthode part d'un langage de transfert de registres (ce qui n'est pas le cas de CASSANDRE où des variables d'une nature autre que Registre existent) où les conditions logiques sont isolées (ce qui n'est pas non plus le cas en CASSANDRE à cause de l'imbrication des instructions conditionnelles).

On construit d'abord le tableau d'états de la partie contrôle, dont les états sont tous connus (comme en CASSANDRE ce sont des étiquettes) et les entrées, les conditions logiques qui peuvent influer sur les transitions entre ces états. Les sorties de cet automate seront des entrées des automates de la partie opératoire.

On choisi une partition de l'ensemble des registres du système logique décrit, qui entraîne une décomposition de celui-ci en sous-systèmes, chacun représenté par un tableau.

Les méthodes de réalisation optimisée à partir de ces tableaux, qui ont fait l'objet de multiples études, peuvent alors s'appliquer.

Nous ne retiendrons pas ici la méthode de décomposition de la partie opératoire en sous-systèmes réalisés comme automates séquentiels, pour plusieurs raisons :

α1) La clarté de la réalisation prime aujourd'hui sur l'économie du nombre de modules élémentaires.

α2) CASSANDRE (mentionné ci-dessus) est beaucoup plus complexe que la langage employé par Gerace et la complexité du traitement qu'il faudrait lui appliquer n'est pas justifiée par l'intérêt du résultat dans le cas général.

α3) Madame De Polignac a réalisé ce traitement dans sa thèse (6) , dans le cas d'une réalisation microprogrammée de la machine décrite. Par contre nous allons montrer qu'il est simple de construire le tableau d'états de la partie contrôle d'une unité.

Cette unité de contrôle a comme entrées (voir d1) les conditions cumulées portant sur :

β1) Les ordres allera internes à l'unité.

β2) Les ordres allera d'une autre unité pour forcer un état.

β3) Les ordres faire internes ou non à l'unité.

β4) Les chargements de variables déclarées état.

Nous prendrons seulement les entrées β1 pour en faire celles du tableau d'états (colonnes). Nous prendrons la liste des étiquettes dont chacune correspondra à une ligne. Les sorties sont plus simples à calculer que dans une seule associée à chaque état, elle est mise à 1 quand l'automate est dans cet état.

Exemple : Unité AUTOMATE (H,E (1:4) ; S (1:5)) ;

Horlogemère H ;

HE1 : si E (1) alors (S(1) : = 1 ; <H> allera HE2) ;

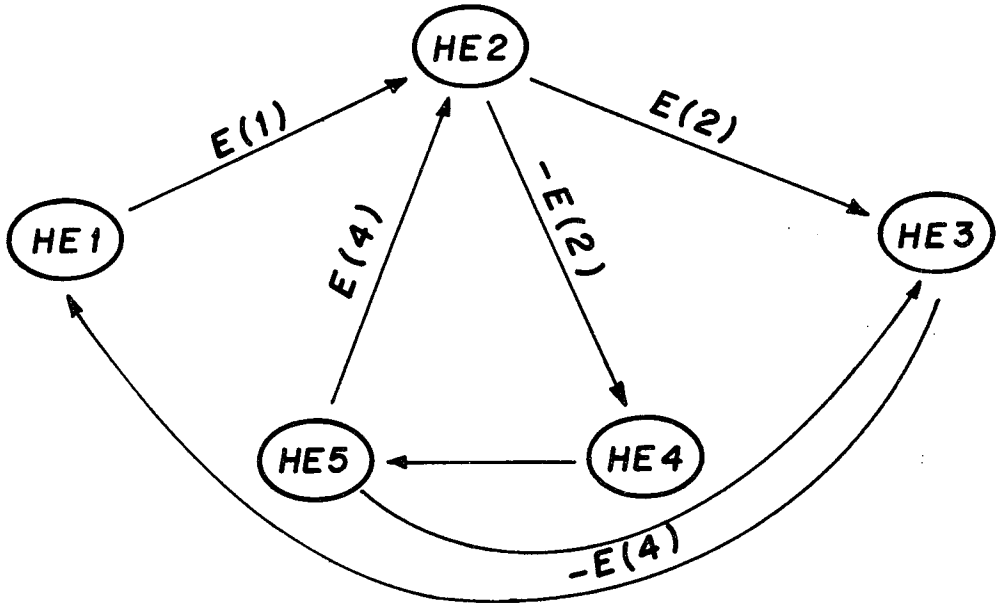
HE2 : S(2) : = 1 ; <H >allera si E(2) alors HE3 sinon HE4 ;

HE3 : S(3) : = 1 ;< H >allera HE1 ;

HE4 : si E(3) alors S(4) : = 1 ;< H >allera HE5 ;

HE5 : si E(4) alors (S(5) : = 1 ;< H >allera HE2) sinon (< H > allera HE3)

Son graphe d'états est :



Son unité de contrôle va être :

```
Unité CONTROLE (H,ECOND (1:4) ; SCOND (1:5) ;
Horloge mère H ;

    HE1 : SCOND(1) : = 1 ; < H >si ECOND(1) alors allera HE2 ;
    HE2 : SCOND(2) : = 1 ; < H >allera si ECOND(2) alors HE3 sinon HE4;
    HE3 : SCOND(3) : = 1 ; < H >allera HE1 ;
    HE4 : SCOND(4) : = 1 ; < H >allera HE5 ;
    HE5 : SCOND(5) : = 1 ; < H >si ECOND(3) alors allera HE2,
                                si ECOND(4) alors allera HE3 ;
```

On voit la différence de traitement entre HE2 où l'expression conditionnelle d'état "si E(2) alors HE3 sinon HE4" est reproduite dans CONTROLE et HE5 où l'instruction conditionnelle est décomposée en ses deux allera, allera HE2 sous la condition ECOND(3) ≡ E(4) et allera HE3 sous la condition ECOND(4) ≡ -E(4). Si on employait aussi une expression conditionnelle d'état dans le tableau d'états qui est le suivant :

	ECOND(1)	ECOND(2)	-ECOND(2)	ECOND(3)	ECOND(4)
1	2;SCOND(1)	1;SCOND(1)	1;SCOND(1)	1;SCOND(1)	1;SCOND(1)
2	2;SCOND(2)	3;SCOND(2)	4;SCOND(2)	2;SCOND(2)	2;SCOND(2)
3	1;SCOND(3)	1;SCOND(3)	1;SCOND(3)	1;SCOND(3)	1;SCOND(3)
4	5;SCOND(4)	5;SCOND(4)	5;SCOND(4)	5;SCOND(4)	5;SCOND(4)
5	5;SCOND(5)	5;SCOND(5)	5;SCOND(5)	2;SCOND(5)	3;SCOND(5)

Dans la réalisation de l'automate de contrôle d'une unité les sorties sont simplement celles d'un décodeur d'état (on pourra les omettre dans le tableau).

La partie opératoire d'AUTOMATE serait :

```
Unité OPERATOIRE (SCOND(1:5), E(1:4) ; S(1:5), ECOND(1:4)) ;
si SCOND(1) alors (si E(1) alors S(1) : = 1 ) ;
si SCOND(2) alors S(2) : = 1 ;
si SCOND(3) alors S(3) : = 1 ;
si SCOND(4) alors (si E(3) alors S(4) : = 1 ) ;
si SCOND(5) alors (si E(5) alors S(5) : = 1) ;
   ECOND : = H(1) & E(2) & E(4) & -E(4) ;
```

Remarques : On voit que son rôle consiste seulement à "filtrer" les sorties SCOND par les conditions E et à connecter ECOND.
Dans contrôle on ne saura pas que ECOND(4) ≡ - ECOND(3), on risque ainsi de manquer certaines simplifications. On pourrait de même décrire un automate asynchrone (en utilisant l'opérateur de dérivation) la construction de son tableau d'état serait un peu plus complexe que dans le cas synchrone que nous venons de voir.

Conclusion : Tout automate peut recevoir en CASSANDRE une description équivalente à son graphe d'état. On peut extraire automatiquement son tableau d'états de cette description en vue de lui appliquer les méthodes de synthèses qui partent de cette donnée. Si l'on prend certaines précautions d'écriture le résultat sera sans gaspillage de matériel (dû à sa description en CASSANDRE).

2) Codage des états :

Un certain nombre de bits est nécessaire pour coder les étiquettes qui représentent les valeurs de la variable interne d'une unité. Le minimum de ce nombre est Log 2 (n) lorsque n (nombre d'états) est une puissance de 2 et l'entier immédiatement supérieur, si ce n'est pas le cas. On appellera "codage minimum" celui qui fait appel à un tel nombre de bits, et "codage trivial" celui qui a autant de bits que d'états (une bascule pour chaque état). L'ensemble des codages peut s'étendre entre ces 2 solutions particulières, nous allons examiner une méthode d'assignement d'un codage aux états dans le cas minimum (méthode d'Amstrong ou Humphrey) avant de donner la réalisation canonique de l'unité de contrôle dans le système CASSANDRE actuel, qui est associée à un "codage trivial". Reprenons l'exemple précédent à 5 états.

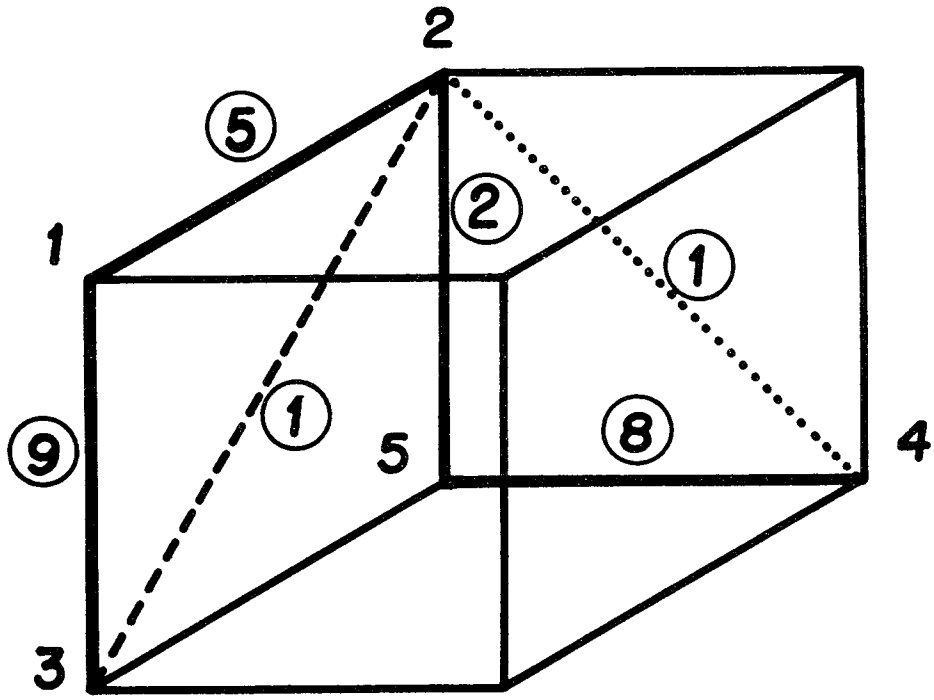
Le codage minimum nécessite 3 bits.

La méthode prend en compte 2 sortes d'adjacences.

α) Les états suivants associés à un même état présent, soit dans l'exemple les "paires de **1ère** espèce" (1,2), (2,3), (2,4), (3,1), (4,5), (5,2), (5,3).

β) Les états obtenus à partir de plusieurs autres sous l'effet de la même entrée soit les "paires de 2e espèce" (5,4,), (1,2), (1,3), (2,5) (toutes incluses dans les précédentes).

Représentons les adjacences sur le cube en pondérant les paires (dans l'ordre : 5,1,1,9,8,2,1).



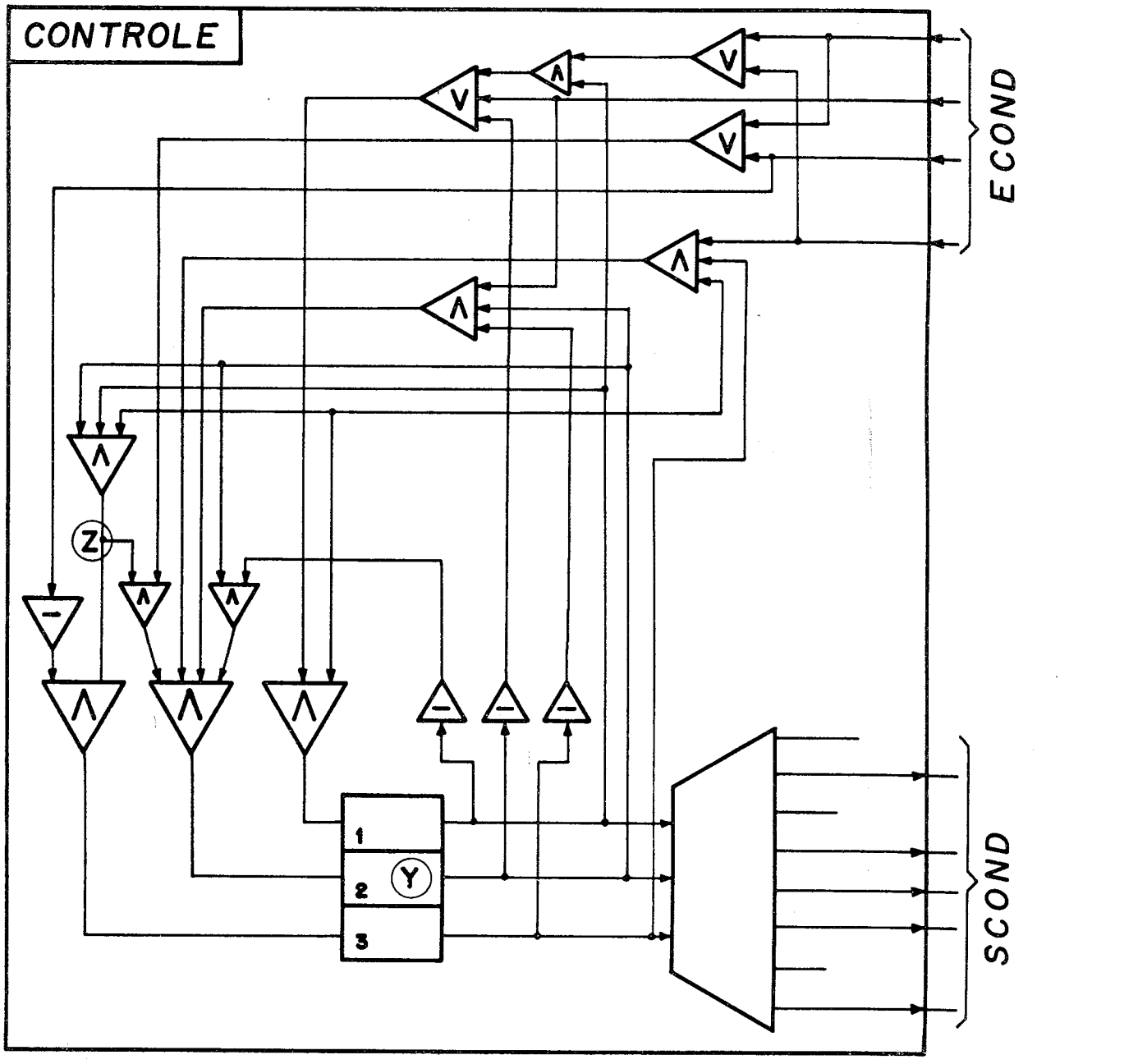
Le tableau est alors :

	x1	x2	-x2	x3	x5
100	110	100	100	100	100
110	110	000	011	110	110
000	100	100	100	100	100
011	010	010	010	010	010
010	010	010	010	110	000

On obtient les équations :

$$\begin{cases} z = y1 \wedge y2 \wedge \neg y3; \\ y1 = \neg y3 \wedge (ECOND(3) \vee \neg y2 \vee y \wedge (ECOND(1) \vee ECOND(4))) ; \\ y2 = (\neg y1 \wedge y2) \vee (y1 \wedge \neg y3 \wedge ECOND(1)) \vee z1 \wedge (\neg ECOND(2) \vee ECOND(4)) \\ \qquad\qquad \vee (y2 \wedge \neg y3 \wedge ECOND(3)) ; \\ y3 = z1 \wedge \neg ECOND(2) ; \end{cases}$$

Soient 4 inverseurs, 5 ET et 2 OU à 2 entrées. 3 ET à 3 entrées et 1 OU à 3 entrées, 1OU à 4 entrées.



3) Réalisation canonique :

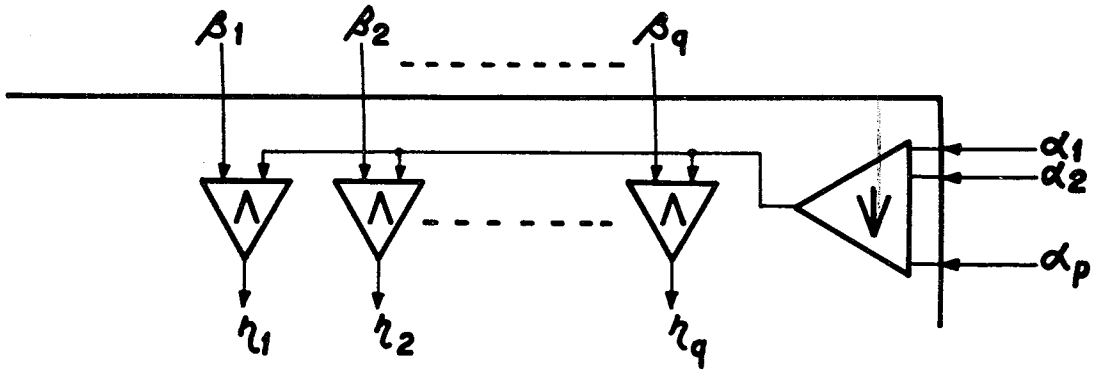
Nous adoptions cette dois un codage trivial, c'est à dire associant une variable interne indépendante à chaque état. Le passage d'un état dans un autre se fait par mise à 1 d'une nouvelle bascule et remise à 0 de celle qui était à 1.

α) Traitement des entrées de forcage d'état:

Les entrées de l'unité CONTROLE qui sont conditions de forcage d'un état à partir d'une autre unité sont prioritaires par rapport aux conditions de changements normaux d'états. Les secondes (soit β1, β2,.. βq)ne seront donc validés que par l'absence des premières (soit α1, α2,... αp). Désignons par γ1, γ2,... γr les conditions sur les ordres faire (indifféremment internes ou externes) et δ1, δ2,... δs les conditions sur les chargements de variables d'états, soient enfin ε1, ε2,... εn la liste des étiquettes.

Ces dernières génèrent un registre ETAT (1:n) qui va stocker la variable interne.

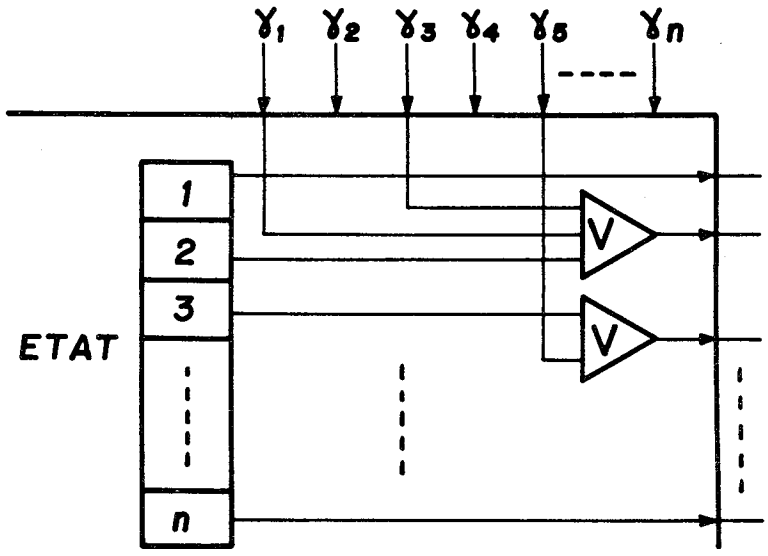
Le NI des conditions α1, α2,... αp vient valider par un étage de ET les entrées β1, β2, ... βq, qui deviennent η1, η2... ηq



β) Traitement des conditions sur les faire :

Les sorties du registre ETAT (1:n) sont autant de sorties de l'unité CONTROLE. Chaque condition γ1, γ2... γr agit sur la sortie d'une bascule parmi celles d'ETAT en créant un OU.

Exemple : γ1 et γ3 portent sur faire ε2 et γ5 sur faire ε3



Chaque fois qu'est apparu un ordre faire un étage de OU vient s'interposer entre les sorties correspondantes d'ETAT et de CONTROLE.

γ) Transitions entre états :

Dans le graphe des états, chaque noeud a en général des entrées et des sorties sur lesquelles partent des conditions η ou α.

A chaque noeud est associé une bascule dans le registre ETAT et un bus sur l'entrée de cette bascule. Ce bus permet de charger 1 sous toutes les conditions qui portent sur des branches en entrée du noeud et 0 sous toutes les conditions sur des branches en sortie.

Reprenons l'exemple précédent (pC40).

Après traitement l'unité CONTROLE devient :

Unité CONTROLE (H,ECOND(1:3) ; SCOND(1:5));

Horlogémère H ;

Externe MCELL (horloge,1,1; 1), BUS 12 (2,2; 1) ;

Signal S(1:4) ;

S(1) : = SCOND(1) Λ ECOND(1) ;

S(2) : = SCOND(5) Λ ECOND(3) Λ SCOND(1)Λ ECOND(1)

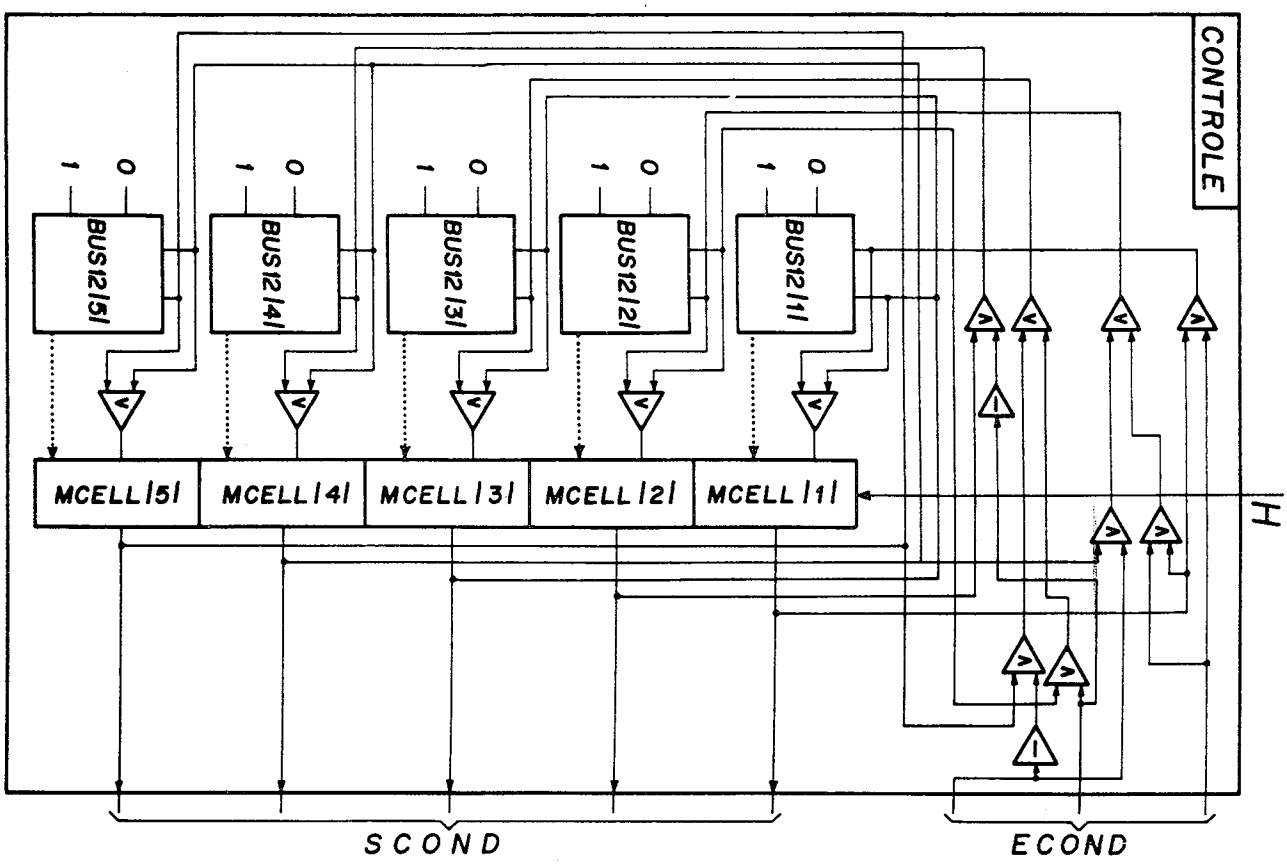
S(3) : = SCOND(2) Λ ECOND(2) ν SCOND(5)Λ -ECOND(3) ;

S(4) : = SCOND(2)Λ -ECOND(2) ;

MCELL|1| (H, S(1) V SCOND(3), BUS12 |1| (S(1)& SCOND(3), 0 &1; *); SCOND(1))

```
MCELL|2| (H,SCOND(2) V S(2), BUS12|2| (SCOND(2)& S(2), 0& 1; *) ; SCOND(2));
MCELL|3| (H,SCOND(3) V S(3), BUS12|3| (SCOND(3)& S(3), 0& 1; *) ; SCOND(3));
MCELL|4| (H,SCOND(4) V S(4), BUS12|4| (SCOND(4)& S(4), 0& 1; *);SCOND(4));
MCELL|5| (H,SCOND(5) V SCOND (4), BUS12 5  (SCOND(5)&SCOND(4), 0& 1 ; *) ;
                                                        SCOND(5)) ;
```

Soit le schéma :



Cette réalisation devrait coûter "plus cher" que la précédente, ce serait normal, en réalité, elle est plus simple.

Chaque unité BUS 12 ne coûte qu'un ET à 2 entrées, soit en tout 7 OU à 2 entrées, 11 ET à 2 entrées et 2 inverseurs. Le bilan par rapport à la précédente réalisation est donc 2 boites MCELL en plus, mais 2 inverseurs et le décodeur à 3 entrées en moins.

CONCLUSION : Lorsqu'on choisi une réalisation canonique pour obtenir une solution plus claire et plus régulière on devrait compenser cet avantage par un coût plus élevé en matériel. Paradoxalement on obtient presque toujours une solution avec moins de matériel logique que dans les solutions dites "optimisées" et on cumule économie et régularité. Ce résultat d'expérience qui admet facilement des contre-exemples, semble justifier "à postériori" le traitement sous forme de compilation, de la partie contrôle elle-même, d'une unité CASSANDRE, on évite ainsi en particulier les circuits décodeurs qui sont coûteux. Toutefois les méthodes classiques de synthèse des automates peuvent sans difficultés se brancher sur le système CASSANDRE comme nous l'avons montré.

c) Traduction en équations booléennes :

Les méthodes de synthèse de fonctions booléennes simples ou multiples ont fait l'objet d'innombrables travaux. Bien que peu employées à ce jour dans la conception des calculateurs dont les modules sont presque toujours réalisés à l'aide de macrocomposants "standard", elles restent quelquefois utiles dans d'autres domaines qui ressortissent également de CASSANDRE.

C'est pourquoi, comme pour les automates séquentiels, nous allons définir le branchement de ces méthodes sur le système CASSANDRE, au niveau des équations booléennes sous forme polynomiale qui constituent presque toujours les données d'entrée de ces méthodes.

Après le traitement de compilation en hardware les unités CASSANDRE initiales peuvent contenir des nouvelles unités "externes" qui elles-même en contiennent d'autres...etc.

Remarques :

- Une unité est combinatoire

  α) si elle ne contient pas d'externe MCELL

  β) si les externes qu'elle contient sont combinatoires

- Une unité combinatoire à plusieurs sorties est une fonction booléènne multiple.

- Une unité combinatoire à une sortie est une fonction booléenne simple.

1) Traitement des réseaux homogènes

Ce cas simplifié avait été étudié sur la 7044, IBM. On considère les unités GET, GOU et GNO comme des primitives, aucune des unités TRS, CCT, TNK, ROT ou Réduction n'existe. On dira qu'un tel réseau d'unités est homogène car toutes les entrées et sorties d'unités ont les mêmes dimensions c'est la juxtaposition d'autant de réseaux du même modèle qu'il y a de bits dans l'une quelconque des variables.

α) Elimination des signaux intermédiaires.

On peut trouver un même signal en partie gauche d'une connexion (ou en sortie d'une connexion d'unité) et en partie droite d'autres connexions (ou en entrée de connexions d'unités) on l'élimine au risque d'écrire plusieurs fois la même connexion d'unité.

Exemple : Dans l'unité COMPTEUR (voir dans la suite de cette partie)

On trouve : signal CT (1:4) ;

```
        Q (0:3) : = CT (1:4) ;
- - - - - - - - - - - - - - - - - - - - - - - -
    REG 01000|0| (H(1), SLG01001(1:4), SLG01000(1:4) ;
                 CT(1:4) ;
- - - - - - - - - - - - - - - - - - - - - - - -
    BUS01001 (ET|2| (CN01000(0), EXP01001 (CT (1:4) ;
                                  *) ; *) ,
                ET|3| (CND01000(1), EXP01002 (EXP01003
               (CT(1:4) ; *); *) ; *) ; TC(1)) ;
```

On va obtenir :

```
Q (0:3) : = REG01000|0| (H(1), SLG01001 (1:4), SLG01000(1:4) ;
                                                *) ;
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
BUS 01001 (ET|2| (CND01000 (0), EXP01001 (REG01000|0|(H(1),
          SLG01001(1:4), SLG01000(1:4) ; *) ; *); *),
          ET|3|(CND01000(1), EXP01002 (EXP01003 (REG01000|0|
          (H(1), SLG01001(1:4), SLG01000(1:4) ; *); *); *);
                                                *); TC(1));
```

β) Lorsqu'on a fait ce traitement pour toutes les unités, les variables qui apparaissent à gauche d'une connexion (ou en sortie d'une connexion d'unité) sont des sorties de l'unité dans laquelle elles apparaissent, et elles n'apparaissent qu'une fois. Toutes les variables déclarées signal ont disparu.

On peut alors éliminer les signaux-unités et les connexions d'unités, on obtiendra les équations booléennes en somme de produits de sommes de....des sorties de l'unité traitée.

```
Exemple : Unité UAL (A(1:4), B(1:4); S1(1:4), S2(1:4), S3(1:4));

          Externe GET(4,4;4), GOU(4,4;4), DIF(4,4;4);

          S1 : = GET (A,B;*);

          S2 : = GOU (A,B;*);

          S3 : = DIF (A,B;*);

          Unité DIF (E1(1:4), E2(1:4); S(1:4));

          Externe GOU(4,4;4), GET(4,4;4), GNO(4,4;4);

          S : = GOU(GET|1|(E1, GNO|1| (E2;*);*),

                    GET|2|(GNO|2| (E1;*), E2; *); *);

          Unité GOU (E1(1:4), E2(1:4); S(1:4)) ;

          S : = E1 V E2 ;

          Unité GET (E1(1:4), E2(1:4); S(1:4)) ;

          S : = E1 Λ E2 ;

          Unité GNO (E(1:4); S(1:4)) ;

          S : = - E ;
```

On obtient d'abord :

```
          Unité DIF (E1(1:4), E2(1:4); S(1:4)) ;

          S : = (E1 Λ (-E2)) V ((-E1) Λ E2) ;
```

Puis :

```
          Unité UAL (A(1:4), B(1:4); S1(1:4), S2(1:4), S3(1:4)) ;

          S1 : = A V B ;

          S2 : = A Λ B ;

          S3 : = (A Λ (-B)) V ((-A) ΛB) ;
```

Par un mécanisme analogue à une macro-substitution. (c'est d'ailleurs un programme macrogénérateur (9) qui faisait ce travail sur le 7044 IBM).

2) Réseaux quelconques

Le traitement est beaucoup plus délicat car les unités TRS, CCT, TNK, ROT, RET et ROU modifient les dimensions des variables traitées, ce traitement n'a pas encore été programmé (la difficulté a déjà été signalée par LIDDELL).

On peut trouver la difficulté en se ramenant au traitement de variables booléènnes scalaires uniquement.(mais le volume de la description risque de devenir prohibitif).

Il suffit pour cela de donner une nouvelle description des unités qui subsistent après compilation, afin qu'elles n'aient plus que des entrées-sorties scalaires, les unités génantes disparaîtront dans ce traitement, car leur description ne contiendra plus que des connexions élémentaires, mais le résultat va contenir beaucoup d'équations booléènnes semblables,aux indices des variables près.

Nous allons suggérer une autre méthode donnant un résultat optimum, sans détailler les difficultés possibles de sa mise en oeuvre.

Après le traitement de surdécoupage on construit grâce à un programme réalisé par Monsieur MILESI, l'arbre des connexions d'unités. Les feuilles de cet arbre seront des unités primitives ou les unités TRS, CCT, TNK, ROT, GOU, GET, GNO considérées comme terminales. Chaque noeud de l'arbre est une unité qui a comme ascendant celle qui la contient, comme descendants celles qu'elle contient et comme soeurs celles qui ont le même ascendant et avec qui elle peut être connectée.

Une unité terminale n'a pas forcément pour soeur que des unités terminales nous allons faire en sorte qu'il en soit ainsi. Une unité non terminale a en général un ascendant, nous allons faire en sorte qu'elle n'en n'ait pas. Le réseau ainsi obtenu sera réduit à deux couches : unité traitée et tous ses descendants constitués d'unités terminales.

α) Opération d'expansion :

Nous appelons ainsi le remplacement d'une unité non terminale, à l'intérieur d'une autre, par le réseau d'unités qu'elle contient, soit U l'unité à insérer dans W. Dans l'unité W ou U est externe, U est connectée soit sous forme de connexion d'unité inconditionnelle, soit sous forme de signal-unité dans une connexion inconditionnelle.

α1) Dans le cas d'un signal-unité on crée une nouvelle déclaration de signal dans W en prenant soin qu'il n'y ait pas confusion possible avec les signaux existant déjà.

On remplace l'occurence du signal-unité par le signal créé. On crée une connexion de U avec en sortie le signal créé, immédiatement après le ";" terminant l'ordre de connexion où apparaissait le signal-unité (en effet, on peut se trouver à l'intérieur d'une instruction arithmétique pour ou sia et il ne faut pas en sortir dans ce traitement).

α2) On crée dans W autant de nouvelles déclarations de signaux, qu'il y a de signaux déclarés dans U (sans tenir compte donc des entrées-sorties de U).

α3) On substitue, dans la définition de U, à ses variables d'entrée-sortie celles qui apparaissent dans la connexion de U, à l'intérieur de W (voir α1).

On substitue à ses variables déclarées signal celles créées en α 2). On substitue de nouvelles variables arithmétiques n'existant pas dans W, à celles qui peuvent être dans U. (ceci inclue les numéros de duplication d'unités du même modèle).

α4) On insère à la place de l'ordre de connexion de U dans W, la description de U obtenue par α 3, à l'exclusion de l'entête d'unité et des déclarations signal. S'il y a des déclarations externe, on ajoute à la liste des déclarations externe de W, celles de U qui n'y sont pas déjà. On supprime de cette liste la déclaration externe U.

Le résultat est la disparition de U dans W et son remplacement (par un mécanisme analogue à une macrosubstitution) par le réseau contenu dans U c'est à dire une liste de connexions de signaux, d'unités et de signaux unités, à l'intérieur peut-être d'instructions arithmétiques.

Exemple : On trouve dans Z0001 :

```
Unité LMEM (I2,I3,D(0:3), AD12(1:3), AIG,SS,COM12; Q(0:3), O(0:3)),
                                                        TCC(1:4)) ;

Horlogémère I2,I3,

Externe COMPTEUR ('Horloge',4,1,1,1,1; 4,1),
        ML (Horloge',4,4; 4) ;

Signal  PE,CU,CD,CS,SC ;
        COMPTEUR (I3,D,PE,CU,CD,COM12; Q,TC) ;
        ML(I2,Q,D,CS; O) ;

si  $ AD12 alors ( ) (CU : = 1) (CU : = AIG) (CU : = -AIG)(PE:=(SS=AIG),
            SC : = 1)(CD : = 1)(CD : = AIC)(CD : = -AIC) ;

si AIG 'alors' (si CU& CD alors (TCC(1) : = TC)(TCC(3) : = TC))
       'sinon' (si CU& CD alors (TCC(2) : = TC)(TCC(4) : = TC)) ;

CS : = SS. AIG. - SC ;
```

On obtient par expansion de COMPTEUR :

```
Unité LMEM (I2,I3,D(0:3), AD12(1:3), AIG,SS,COM12;
                          Q(0:3), O(0:3), TCC(1:4)) ;

Horlogémère I2,I3 ;

Externe ML ('Horloge',4,4; 4) ;

Signal  PE,CU,CD,CS,SC,SGNO1000(0:3), SGNO1001(0:3) ;

Registre CT(1:4)

'si' CU& CD 'alors'  (SGNO1001 : = 0001 ; TC : = /Λ CT)
                     (SGNO1001 : = 1111 ; TC : = /Λ (-CT)) ;

SGNO1000 : = 1 ↑ O (CT ΛSGNO1001 V CTΛSGNO1000 V SGNO1000Λ SGNO1001)& O ;

Q : = CT ;

 <I3 > 'si' PE & (CUVCD)& COM12 alors (CT ← D)
```

CT ← CT ≠ (SGNO1000 ≠ SGNO1001)) (CT ← 0000) ;

'si' § AD12 'alors'.....

On voit sur l'exemple que le mécanisme d'expansion est aussi simple pour une description en CASSANDRE.G que pour une description en CASSANDRE.E telle que nous l'avons définie.

Il faut rajouter toutes les déclarations en changeant le nom des variables s'il y a risque de confusion, puis substituer dans toute la description les variables nouvelles à celles de l'unité insérée.

β) Simplification

β1) Réduction du réseau d'unités à 2 couches

On applique l'opération d'expansion à toutes les unités sauf celles de type GET, GOU et GNO (ou aussi ET,OU,NON). Les unités TRS,CCT, TNK, ROT,RET et ROU disparaissent dans le traitement en ne laissant que des connexions.

β2) Disparition des instructions arithmériques

Des ordres de connexion peuvent se trouver à l'intérieur d'instruction pour (éventuellement imbriquées dans d'autres instructions pour) soit I, J, K,..., M les variables arithmétiques de ces instructions pour, on les range en 2 classes,

- Celles qui apparaissent en indice, en même position dans toutes les variables situées à l'intérieur de leur portée.
- Les autres

On fait disparaître les instructions pour relatives à la première classe en créant des unités GOU, GET et GNO plus grandes que celles qui se trouvaient sous leur portée.

Exemple : pour I = 1 à d1 début

pour J = 1 à d2 début

pour K = 1 à d3 début

A(I,J,K,1:d4,...,1:dn) : = GOU(GET|1|(B(I,J,1:d4,...,1:dn), C(I,J,1:d4,
...,1:dn); *),
GET|2|(D(I,J,1:dn,...,1:dn), E(I,J,1:d4,...,1:dn); *) ; *); fin;fin;fin;

Cette description va devenir :

pour K = 1 à d3 début

A(1:d1,1:d2,K,1:dn,...,1:dn) : = GOU(GET|1| (B(1:d2,1:d2,1:d4,...,1:dn),
C(1:d1,1:d2,1:d4,...1:dn); *), GET|2|(D(1:d1,1:d2,1:d4,...,1:dn),
E(1:d1,1:d2,1:d4,...,1:dn); *); *); fin ;

Il convient de vérifier au cours du traitement que les nouvelles unités GOU, GET et GNO sont bien déclarées externe, sinon le faire.

Le reste des notions arithmétiques disparaît par son exécution pour toutes les valeurs des indices, avec les duplications que cela implique des parties de description qui se trouvent sous leur portée.

γ) Découpage longitudinal du réseau :

Dans cette partie il s'agit de suivre les signaux depuis les signaux de sortie jusqu'à ceux d'entrée (c'est un traitement analogue qu'emploie DAVID pour la VISUALISATION pour le tracé des connexions, voir paragraphe suivant).

Une sortie de l'unité traitée se trouve en partie gauche de connexions de signaux ou en sortie de connexions d'unités. Si pour une même sortie il y a plusieurs connexions ce ne peut être que sur des ensembles différents de bits de cette sortie.

On divise alors la sortie en autant de parties qu'il convient et on reprend le traitement à la première de ces parties.

Exemple : Unité (, ,..., ; S (1:16) ,..,..,);
S (1:8) : = A ;
U (B,C; S(9:16)) ;

On remplace dans l'unité principale S par 2 sorties.

- Quand la sortie traitée se trouve à gauche d'une seule connexion de signal, on la remplace par le signal en partie droite et on reprend le traitement pour celui-ci.
- Si le signal traité se trouve en sortie d'une unité , on le remplacera successivement par chaque entrée de cette unité que l'on recopie en un nouvel exemplaire.

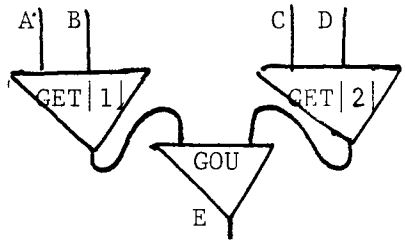
En poursuivant le même traitement sur chaque entrée on construit un arbre d'unités GOU, GET et GNO.

Deux difficultés peuvent se présenter :

γ1) Le signal en cours de traitement est connecté à plusieurs sorties d'unités ou autres signaux. Comme on l'a vu, ci-dessus, pour les sorties de l'unité principale on va décomposer ce signal en autant de parties disjointes qu'il y a de connexions.

Mais il faut aussi dupliquer en autant de parties tout l'arbre que l'on avait construit précédemment, après quoi on reprend le traitement où on en était pour chaque nouveaux arbres successivement.

Exemple :



on a : A (1:5) : = H (5:9);

A (6:16) : = I (1:10);

Il va falloir dédoubler A,B,C,D,E, en (SGL1(1:5), SGL2(1:11)), (SGL3(1:5), SGL4(1:11)), (SGL5(1:5), SGL6(1:11)), (SGL7(1:5), SGL8(1:11)), (SGL9(1:5), SGL10(1:11)), et les unités GET et GOU en Externe GET(5,5;5), GET(11,11;11), GOU(5,5;5) GOU(11,11;11) ;

Avec les réseaux :

SGL9 : = GOU |1| (GET |1| (SGL1, SGL3; *), GET |3| (SGL5, SGL7; *) ;
*) ;

SGL10 : = GOU |2| (GET |2| (SGL2, SGL4; *), CET |4| (SGL6, SGL8; *) ;
*) ;

Puis on reprend le traitement pour successivent SGL1, SGL2...SGL8.

γ2) Le signal en cours de traitement est connecté à une partie d'un autre signal ou à une partie d'une sortie d'unité (c'est toujours le cas en particulier dès que l'on a appliqué γ1)). L'unité du réseau original que l'on a atteinte dans la construction de l'arbre devra être recopiée sous forme d'un autre exemplaire ayant la bonne dimension en entrées et sortie.

On construit ainsi un arbre jusqu'au moment où le signal en cours de traitement est une entrée de l'unité principale (ou une partie de cette entrée).

On remplace l'entrée par autant d'entrées qu'il y a de feuilles dans lesarbres construits qui s'y rattachent.

N.B. Il se peut tout à fait après ce traitement que les mêmes composantes d'une même variable apparaissent dans plusieurs entrées différentes de l'arbre obtenu ou dans plusieurs de ses branches. Ce n'est pas génant car ceci est un traitement formel qui donne seulement un résultat intermédiaire avant les équations booléènnes.

Lorsque l'on a appliqué ce traitement depuis toutes les sorties de l'unité principale jusqu'à ses entrées, on a construit par recopie un certain nombre d'arbres dont les sorties (ou des parties de sorties) sont les racines. Chacun de ces arbres est homogène. On peut calculer comme déjà vu les équations booléènnes de sa racine. On obtient ainsi le plus petit nombre d'équations booléènnes qui peuvent être formellement différentes (certaines se trouvent peut être semblables avec des composantes différentes des mêmes variables).

III - Visualisation et manipulation des réseaux d'unités

a) Etablissement des documents d'un projet

1) La documentation qui accompagne un projet est en général constituée de 3 sortes d'éléments :

α1) Un texte en français.

α2) Des schémas à différents niveaux de détail.

α3) Une description formelle des fonctions logiques.

CASSANDRE associé à un processus de visualisation et de dessin des schémas logiques répond aux points α2 et α3, et permet α1 par introduction de commentaires dans la description.

Les avantages d'intégrer la fonction documentation dans l'ensemble du système CASSANDRE sont évidents : β1) La plus grande partie de la documentation étant soit du texte en CASSANDRE (après toutes les étapes de vérification) soit, des schémas produits automatiquement à partir d'elle, la documentation est sûre.

β2) Sa production imprimée ou dessinée est automatique, donc sans erreur d'interprétation ou de frappe, c'est indispensable pour un document de référence. De plus, on l'obtient sans délai.

β3) Comme l'évolution du projet se confond avec celle des fichiers CASSANDRE, à chaque instant la documentation est à jour.

2) Topologie et implantation des réseaux logiques

Les propriétés des réseaux d'unités (qui à un certain niveau peuvent correspondre aux circuits logiques réels) sont de deux ordres α) Topologiques, (et indépendantes de la technologie), ce sont les connexions et les permuttations d'entrées-sorties d'unités.

β) D'implantation (liées à la technologie de la réalisation) ce sont les dimensions et formes des boites, leur localisation, la largeur des connexions, leurs directions permises et la garde d'isolation entre elles.

Les propriétés topologiques font partie de la description en CASSANDRE, puisque sont exprimés aussi bien les connexions, que l'ordre des entrées et sorties à la périphérie d'une unité (quand cet ordre est différent de celui de la définition de l'unité).

Les études théoriques très nombreuses, qui ont été menées dans ce domaine portent, soit sur les propriétés topologiques (étude de planarité, nombre de coupures minimum dans les connexions d'un réseau de boites pour le rendre planaire, décomposition d'un réseau en plusieurs couches de réseaux planaires...), soit sur les problèmes d'implantation , algorithmes de tracé assez rapides pouvant prendre en compte des réseaux complexes (cinquantaine de boites) et les contraintes d'implantation.

Les premières études peuvent se connecter directement au système CASSANDRE, puisque la donnée de ces traitements c'est la description abstraite d'un réseau, qui peut se faire commodément en CASSANDRE.E. Le résultat des traitements, pour les mêmes raisons peut être exprimé sous forme d'une description en CASSANDRE.E.

Les études d'implantation sortent du cadre de CASSANDRE, car leur résultat est la liste des propriétés géométriques d'un réseau réel (où les fils ne sont pas des courbes mais des surfaces), informations qui ne s'expriment pas en CASSANDRE. Au niveau de telles études ni les propriétés logiques ni même les propriétés topologiques ne peuvent plus être changées.

Mais CASSANDRE.E constitue une description d'entrée satisfaisante pour ces traitements aussi.

L'étude de nombreux travaux faits dans ce domaine a montré, en effet, que la description du réseau à traiter se faisait toujours dans un formalisme très voisin de CASSANDRE.E. Leur connexion au système CASSANDRE ne présente aucune difficulté.

3) Visualisation et dessin de schémas logiques

Le but souhaité est d'obtenir un dessin clair avec des connexions dans deux directions perpendiculaires seulement et pouvant se croiser sans limitation. On peut se poser le problème du placement initial des boites et de l'évolution de ce placement pour rendre le dessin le plus simple. Différents algorithmes sont étudiés par DAVID (2) , une possibilité serait de disperser les boites autour d'un barycentre calculé d'après la densité des liaisons entre ces boites, méthodes utilisée par SAILLARD dans le programme DESMAG (10). Ce schéma doit pouvoir être amélioré, en mode conversationnel, devant la console de visualisation (problème étranger à CASSANDRE). Citons la thèse de Monsieur FARRENY (7)

Les opérations que l'on doit pouvoir réaliser après avoir produit les schémas logiques sont :

α) Modification d'une connexion.

β) Modification de la permuttation des entrées-sorties d'une unité.

γ) Expansion d'une unité (qui correspond à l'effet de zoom appliqué à une unité).

δ) Contraction (voir paragraphe suivant) de 2 ou plusieurs unités

Toutes ces opérations s'effectuent sur la description en CASSANDRE puis se répercuttent sur l'image (elles sont d'ailleurs plus faciles à commander du télétype que par le crayon lumineux).

Le terminal de visualisation ne sera donc dans le système qu'un moyen de sortie (complété par une sortie sur table traçante) et non un moyen de dialogue.

4) Opération de contraction

Cette opération appliquée à deux unités d'un réseau d'unités consiste à les faire disparaître en les remplaçant par une seule qui les contient et qui a la même configuration de connexions avec le reste du réseau que les 2 précédentes.

Cette opération est définie dans : "théorie des réseaux" cours du Professeur KUNTZMANN (8) (étudiée par Madame RAYNAUD) sous le nom de réduction. Nous l'appellerons contraction car l'opération de réduction existe déjà en CASSANDRE.

Soit U et V ces 2 unités dans l'unité R, on établit les 2 listes ordonnées de leurs signaux d'entrée-sortie apparaissant dans leur connexions.

On crée une définition d'unité W dans laquelle on place les déclarations externe U ( ) et externe V et les connexions de U et V. Puis on balaye la liste des signaux d'entrée sortie de U et V. Si un signal en entrée (resp. Sortie) dans la connexion de U ou V apparaît en entrée (resp. en sortie) de R, ou ce signal apparaît dans une autre connexion contenue dans R, c'est une entrée (resp. une sortie) de W, (sauf si l'on avait déjà rencontré en entrée (resp. sortie) auparavant) sinon on crée dans la définition de W une déclaration signal pour lui.

Quand on a fini le balayage on a construit l'entête de W et ses déclarations signal.

Dans R on remplace les déclarations en externe de U et V par celle de W et les connexions de U et V par une connexion de W avec les signaux qui connectaient U et V et qui ont résisté au balayage précédent.

Soit maintenant x1, x2,...xe,...xn et y1, y2,...ym,...yp les listes d'entiers (exprimant les permuttations d'entrées-sorties de U et V).

Soit respectivement l et m le nombre de leurs entrées. On crée la liste x1, x2,...,xe, y1+n, ym+n, x_{2+1},...xn, ym+1+n,..., yp+n. Dans cette liste on supprime tous les xi et yj correspondant à une entrée-(sortie) de U ou V qui n'est pas entrée-(sortie) de W.

A chaque suppression on diminue de 1 tous les xk > xi et yn > yj.
On obtient la liste d'entiers associés à la connexion de W.

Exemple :

```
Unité R (...., A(1:2),...;...S(1:2),     ) ;

Signal C (1:2), D(1:2), E(1:2), F(1:2), G(1:2), H(1:2) ;

Externe U (2,2,2,2;3), V (2,3,2,2;2,2) ;
         U (A,C,E,F;D) (1,3,4,5,2) ;
         V (E,D,G,H;F,S) (1,2,6,4,3,5) ;
```

Liste des entrées sorties :

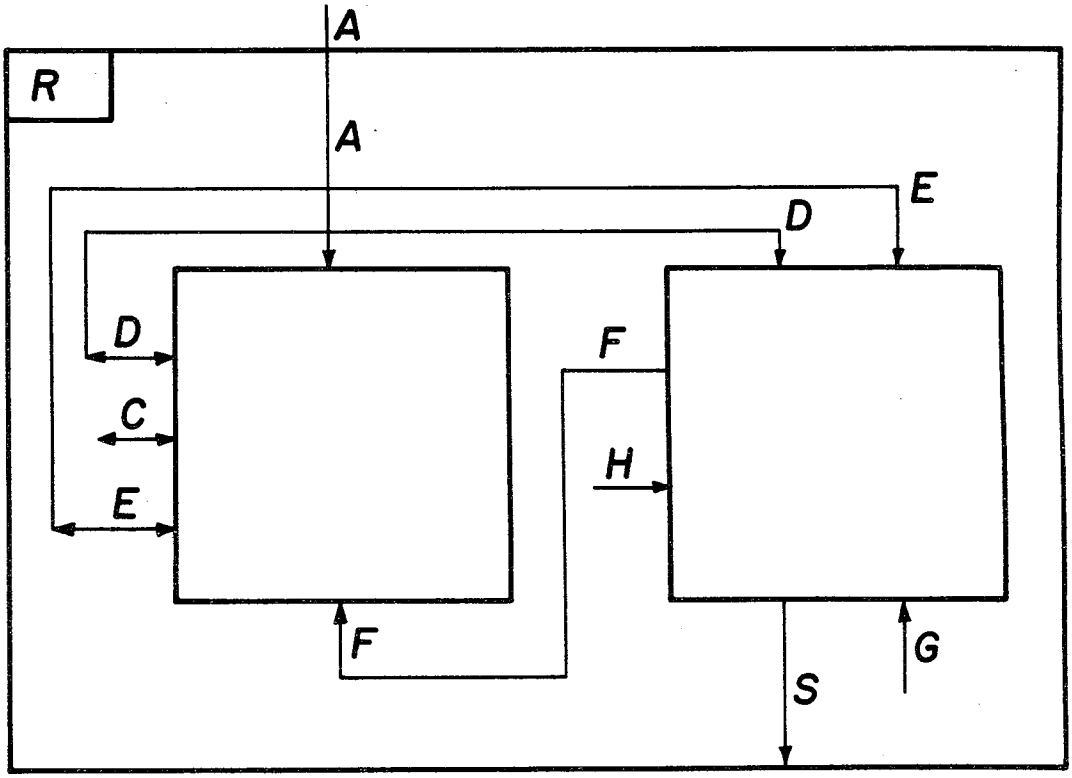
A,C,E,~~F~~,~~E~~,~~D~~,G,H;~~D~~,~~F~~,S

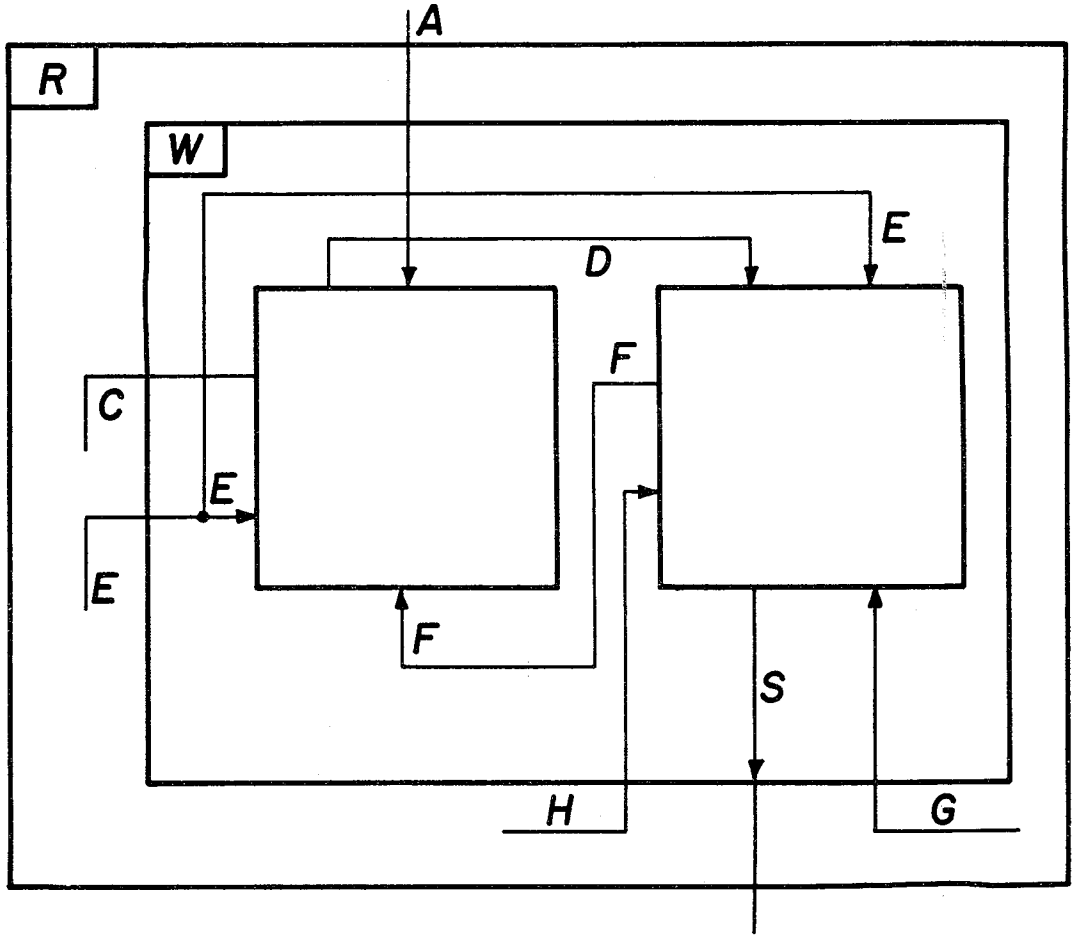
D'où :

```
Unité W (A(1:2), C(1:2), E(1:2), G(1:2) H(1:2); S(1:2)) ;

Signal F (1:2), D(1:3) ;

Externe U (2,2,2,2;3),  V(2,3,2,2;2,2) ;
         U (A,C,E,F;D) (1,3,4,5,2) ;
         V (E,D,G,H;F,S) (1,2,6,4,3,5) ;

Unité R (...,A(1:2),...;...; S(1:2),...) ;

Externe W  (2,2,2,2,2;2) ;
         W (A,C,E,G,H;S) (1,2,3,6,4,5) ;
```

On obtient la liste d'indices de W ainsi :

- (1,3,4,5,6,7,11,9,2,8,10)
- (1,3,4,5,6,10,8,2,7,9)
- (1,3,4,5,9,7,2,6,8)
- (1,3,4,8,6,2,5,7)
- (1,2,3,7,5,4,6)
- (1,2,3,6,4,5)


A
R
A
E
D
D
C
E
F
H
F
G
S


A
R
W
E
D
C
F
E
E
F
S
H
G

b) Surdécoupage et schémas de la partie opératoire de Z0001

Pour illustrer ce chapitre nous donnons, ci-après, le résultat de la compilation des unités de la partie opératoire de Z0001, et le résultat du tracé sur la table BENSON.

Les premiers résultats dûs aux programmes de FANTINO, sont des textes en CASSANDRE que l'on peut réintroduire si l'on veut en entrée du système (par exemple pour les simuler).

Les programmes de DAVID analysent ces unités créées et génèrent les arbres d'imbrication des unités les unes dans les autres, ainsi que le schéma du réseau contenu dans chaque unité. Ce dessin peut être optimisé à la console IBM 2250 en mode conversationnel. C'est au cours d'une telle optimisation que DAVID a employé à plusieurs reprises l'opération de contraction d'un ensemble d'unités (créant les unités BUSAND20, BUSAND21, BUSXPA22, dans ML ; BUSXPA42, FONCTION et LINPUTBS dans UAL, ARITHM dans R). On obtient toujours ainsi un schéma simple quelle que soit la complexité initiale du réseau à visualiser.

Les programmes de MENARD assurent le tracé à la table BENSON de tous les schémas produits par DAVID. Ils peuvent par superposition faire des dessins plus complexes que ceux que tolère la console 2250.

Tous ces traitements partent de la description de Z0001 utilisée dans le chapitre précédent pour sa simulation.

1) Unités COMPTEUR, ML, LMEM, UAL, DECD, DECG, DEC, SPES, COMPT, ZOOO1, R, DECOD, VALN2, VALN3, VALN4.

```
FILE: UNIT1    CASSPGR  P5                                  CAMBRIDGE  MONITOR  SYSTEM

'UNITE' EXP01000 (E000(0:3),E001;S (0:3));
          S:=*D|1| E000 & E001 ;
'UNITE' EXP01001 (E000(0:3),E001(0:3),E002(0:3),E003(0:3),E004(0:3),
E005(0:3);S (0:3));
          S:= E000 . E001 + E002 . E003 + E004 . E005 ;
'UNITE' EXP01002 (E000,E001;S (0:1));
          S:= E000 & E001 ;
'UNITE' AND01000 (COND,E (0:3);S (0:3));
          'POUR' I=0 'A' 3
        'DEBUT'
            S (I):=COND . E(I);
        'FIN';
'UNITE' EXP01003 (E000(0:3);S );
          S:=/. E000 ;
'UNITE' EXP01004 (E000(0:3);S );
          S:=/. E000 ;
'UNITE' EXP01005 (E000(0:3);S (0:3));
          S:=- E000 ;
'UNITE' EXP01006 (E000,E001,E002;S (0:2));
          S:= E000 & E001 & E002 ;
'UNITE' EXP01007 (E000,E001;S );
          S:= E000 + E001 ;
'UNITE' EXP01008 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 # E001 ;
'UNITE' EXP01009 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 # E001 ;
'UNITE' REG01000 (H,ENABLE(0:3),INFO(0:3);OUTPUT(0:3));
  'HORLOGEMERE' H;
  'EXTERNE' MCELL('HORLOGE',,;);
          'POUR' I=0 'A' 3
        'DEBUT'
            MCELL |I| (H,ENABLE(I),INFO(I);OUTPUT(I));
        'FIN';
'UNITE' XPA01000(IN;OUT(0:3));
          'POUR' I=0 'A' 3
        'DEBUT'
            OUT(I):=IN;
        'FIN';
'UNITE' BUS01000(E000(0:3),E001(0:3);S (0:3));
          S:=E000+E001;
'UNITE' BUS01001(E000(1:1),E001(1:1);S (1:1));
          S:=E000+E001;
'UNITE' BUS01002(E000(1:4),E001(1:4),E002(1:4);S (1:4));
          S:=E000+E001+E002;
'UNITE' BUS01003(E000(1:4),E001(1:4),E002(1:4);S (1:4));
          S:=E000+E001+E002;
'UNITE' COMPTEUR(H,P(0:3),PE,CU,CD,RAZ;Q(0:3),TC);
  'SIGNAL' CI(1:4);
  'SIGNAL' MU1(0:3),MU2(0:3);
  'SIGNAL' SGL01001(1:4),SGL01000(1:4),CND01001(0:2),CND01000(0:1);
  'HORLOGEMERE' H;
  'EXTERNE' EXP01000 ((0:3),;(0:3)),EXP01001 ((0:3),(0:3),(0:3),(0:3),
  (0:3),(0:3);(0:3)),EXP01002 (,;(0:1)),EXP01003 ((0:3);),EXP01004 ((0
  :3);),EXP01005 ((0:3);(0:3)),EXP01006 (,,;(0:2)),EXP01007 (,;),
  EXP01008 ((0:3),(0:3);(0:3)),EXP01009 ((0:3),(0:3);(0:3));
```

```
FILE: UNITI    CASSPGR  P5                          CAMBRIDGE  MONITOR  SYSTEM

  'EXTERNE' ET(,;),AND01000(,(C:3);(C:3));
  'EXTERNE' XPA01000(;(C:3));
  'EXTERNE' REG01000('HORLOGE',(C:3),(C:3);(C:3));
  'EXTERNE'BUS01000((C:3),(C:3);(C:3)),BLSG1C01((1:1),(1:1);(1:1)),
  BUS01002((1:4),(1:4),(1:4);(1:4)),BLSG1C03((1:4),(1:4),(1:4);(1:4));
          Q(0:3):=CT(1:4);
          EXP01000(EXP01001(CT(1:4),MD2(C:3),CT(1:4),MC1(0:3),MD1(0:3)
          ,MD2(0:3);*),0;MD1(C:3));
          EXP01002(CD(1),CD(1);CND01000(0:1));
          EXP01006(PE(1),EXP01007(CD(1),CD(1);*),RAZ(1);CND01001(0:2))
          ;
          REG01000(0)(H(1),SGL01C01(1:4),SGL01000(1:4);CT(1:4));
          BUS01003(AND01000|0|(CND01000(C),0001;*),AND01000|1|(
          CND01000(1),1111;*);MD2(0:3));
          BUS01001(ET|2|(CND01000(C),EXP01003(CT(1:4);*);*),ET|3|(
          CND01000(1),EXP01004(EXP01005(CT(1:4);*);*);*);TC(1));
          BUS01002(AND01000|2|(CND01001(C),P(0:3);*),AND01000|3|(
          CND01001(1),EXP01008(CT(1:4),EXP01009(MD2(0:3),MD1(0:3);*);*
          );*),AND01000|4|(CND01001(2),0CC0;*);SGL01000(1:4));
          BUS01003(XPA01000|0|(CND01001(C);*),XPA01000|1|(CND01001(1);
          *),XPA01000|2|(CND01001(2);*);SGL01001(1:4));
```

```
FILE: UNIT2    CASSPGR  P5                                CAMBRIDGE  MONITOR  SYSTEM

'UNITE' AND02000 (COND,E (0:15);S (0:15));
          'POUR' I=0 'A' 15
        'DEBUT'
            S (I):=COND . E(I);
        'FIN';
'UNITE' AND02001 (COND,E (0:3);S (0:3));
          'POUR' I=0 'A' 3
        'DEBUT'
            S (I):=COND . E(I);
        'FIN';
'UNITE' REG02000 (H,ENABLE(0:3,0:15),INFO(0:3,0:15);OUTPUT(0:3,0:15));
  'HORLOGEMERE' H;
  'EXTERNE' MCELL('HORLOGE',,,);
          'POUR' I=0 'A' 3
        'DEBUT'
            'POUR' J=0 'A' 15
          'DEBUT'
              MCELL [10.I+J] (H,ENABLE(I,J),INFO(I,J);OUTPUT(I,J));
          'FIN';
        'FIN',
'UNITE' XPA02000(IN;OUT(0:3));
          'POUR' I=0 'A' 3
        'DEBUT'
            OUT(I):=IN;
        'FIN';
'UNITE' BUS02000(E000(1:4,0:0),E001(1:4,1:1),E002(1:4,2:2),E003(1:4,3:
3),E004(1:4,4:4),E005(1:4,5:5),E006(1:4,6:6),E007(1:4,7:7),E008(1:4,8:
8),E009(1:4,9:9),E010(1:4,10:10),E011(1:4,11:11),E012(1:4,12:12),E013(
1:4,13:13),E014(1:4,14:14),E015(1:4,15:15);S (1:4,0:15));
          S:=*P(*P(E000)&*P(E001)&*P(E002)&*P(E003)&*P(E004)&*P(E005)&
          *P(E006)&*P(E007)&*P(E008)&*P(E009)&*P(E010)&*P(E011)&*P(
          E012)&*P(E013)&*P(E014)&*P(E015));
'UNITE' BUS02001(E000(0:3),E001(0:3),E002(0:3),E003(0:3),E004(0:3),
E005(0:3),E006(0:3),E007(0:3),E008(0:3),E009(0:3),E010(0:3),E011(0:3),
E012(0:3),E013(0:3),E014(0:3),E015(0:3);S (0:3));
          S:=E000+E001+E002+E003+E004+E005+E006+E007+E008+E009+E010+
          E011+E012+E013+E014+E015;
'UNITE' BUS02002(E000(1:4,0:0),E001(1:4,1:1),E002(1:4,2:2),E003(1:4,3:
3),E004(1:4,4:4),E005(1:4,5:5),E006(1:4,6:6),E007(1:4,7:7),E008(1:4,8:
8),E009(1:4,9:9),E010(1:4,10:10),E011(1:4,11:11),E012(1:4,12:12),E013(
1:4,13:13),E014(1:4,14:14),E015(1:4,15:15);S (1:4,0:15));
          S:=*P(*P(E000)&*P(E001)&*P(E002)&*P(E003)&*P(E004)&*P(E005)&
          *P(E006)&*P(E007)&*P(E008)&*P(E009)&*P(E010)&*P(E011)&*P(
          E012)&*P(E013)&*P(E014)&*P(E015));
'UNITE' ML(I,A(0:3),B(0:3),CS;O(0:3));
  'SIGNAL' M(1:4,0:15);
  'SIGNAL' SGL02001(1:4,0:15),SGLC2000(1:4,0:15),CND02001(1:16),
  CND02000(0:15);
  'HORLOGEMERE' I;
  'EXTERNE' VAL44((1:4);(1:16));
  'EXTERNE' AND02000(,(0:15);(0:15)),AND02001(,(0:3);(0:3));
  'EXTERNE' XPA02000(,(0:3));
  'EXTERNE' REG02000('HORLOGE',(0:3,0:15),(0:3,0:15);(0:3,0:15));
  'EXTERNE'BUS02000((1:4,0:0),(1:4,1:1),(1:4,2:2),(1:4,3:3),(1:4,4:4),
  (1:4,5:5),(1:4,6:6),(1:4,7:7),(1:4,8:8),(1:4,9:9),(1:4,10:10),(1:4,
```

```
FILE: UNIT2    CASSPGR  P5                                  CAMBRIDGE  MONITOR  SYSTEM

  11:11),(1:4,12:12),(1:4,13:13),(1:4,14:14),(1:4,15:15);(1:4,0:15)),
  BUS02001((0:3),(0:3),(0:3),(0:3),(0:3),(0:3),(0:3),(0:3),(0:3),(0:3)
  ,(0:3),(0:3),(0:3),(0:3),(0:3),(0:3);(0:3)),BUS02002((1:4,0:0),(1:4,
  1:1),(1:4,2:2),(1:4,3:3),(1:4,4:4),(1:4,5:5),(1:4,6:6),(1:4,7:7),(1:
  4,8:8),(1:4,9:9),(1:4,10:10),(1:4,11:11),(1:4,12:12),(1:4,13:13),(1:
  4,14:14),(1:4,15:15),(1:4,0:15));
          VALN4(A(0:3);CND02001(1:16));
          AND02000|0|(CS(1),VALN4(;*);CND02000(0:15));
          REG02000|0|(1(1),SGL02001(1:4,0:15),SGL02000(1:4,0:15);M(1:4
          ,0:15));
          BUS02000(AND02001|0|(CND02000(0),D(0:3);*),AND02001|1|(
          CND02000(1),D(0:3);*),AND02001|2|(CND02000(2),D(0:3);*),
          AND02001|3|(CND02000(3),D(0:3);*),AND02001|4|(CND02000(4),D(
          0:3);*),AND02001|5|(CND02000(5),D(0:3);*),AND02001|6|(
          CND02000(6),D(0:3);*),AND02001|7|(CND02000(7),D(0:3);*),
          AND02001|8|(CND02000(8),D(0:3);*),AND02001|9|(CND02000(9),D(
          0:3);*),AND02001|10|(CND02000(10),D(0:3);*),AND02001|11|(
          CND02000(11),D(0:3);*),AND02001|12|(CND02000(12),D(0:3);*),
          AND02001|13|(CND02000(13),D(0:3);*),AND02001|14|(CND02000(14
          ),D(0:3);*),AND02001|15|(CND02000(15),D(0:3);*);SGL02000(1:4
          ,0:15));
          BUS02001(AND02001|16|(CND02001(1),M(1:4,0);*),AND02001|17|(
          CND02001(2),M(1:4,1);*),AND02001|18|(CND02001(3),M(1:4,2);*)
          ,AND02001|19|(CND02001(4),M(1:4,3);*),AND02001|20|(CND02001(
          5),M(1:4,4);*),AND02001|21|(CND02001(6),M(1:4,5);*),AND02001
          |22|(CND02001(7),M(1:4,6);*),AND02001|23|(CND02001(8),M(1:4,
          7);*),AND02001|24|(CND02001(9),M(1:4,8);*),AND02001|25|(
          CND02001(10),M(1:4,9);*),AND02001|26|(CND02001(11),M(1:4,10)
          ;*),AND02001|27|(CND02001(12),M(1:4,11);*),AND02001|28|(
          CND02001(13),M(1:4,12);*),AND02001|29|(CND02001(14),M(1:4,13
          );*),AND02001|30|(CND02001(15),M(1:4,14);*),AND02001|31|(
          CND02001(16),M(1:4,15);*);D(0:3));
          BUS02002(XPA02000|0|(CND02000(0);*),XPA02000|1|(CND02000(1);
          *),XPA02000|2|(CND02000(2);*),XPA02000|3|(CND02000(3);*),
          XPA02000|4|(CND02000(4);*),XPA02000|5|(CND02000(5);*),
          XPA02000|6|(CND02000(6);*),XPA02000|7|(CND02000(7);*),
          XPA02000|8|(CND02000(8);*),XPA02000|9|(CND02000(9);*),
          XPA02000|10|(CND02000(10);*),XPA02000|11|(CND02000(11);*),
          XPA02000|12|(CND02000(12);*),XPA02000|13|(CND02000(13);*),
          XPA02000|14|(CND02000(14);*),XPA02000|15|(CND02000(15);*);
          SGL02001(1:4,0:15));
```

```
FILE: UNIT3    CASSPUR  P5                          CAMBRIDGE  MONITOR  SYSTEM

'UNITE' EXP03000 (E000,S );
          S:=- E000 ;
'UNITE' EXP03001 (E000,E001;S );
          S:= E000 = E001 ;
'UNITE' EXP03002 (E000,S );
          S:=- E000 ;
'UNITE' EXP03003 (E000,E001,E002;S );
          S:= E000 . E001 .- E002 ;
'UNITE' EXP03004 (E000,E001;S (0:1));
          S:= E000 & E001 ;
'UNITE' AND03000 (COND,E (0:1);S (0:1));
          'POUR' I=0 'A' 1
        'DEBUT'
           S (I):=COND . E(I);
        'FIN';
'UNITE' EXP03005 (E000,E001;S (0:1));
          S:= E000 & E001 ;
'UNITE' BUS03000(E000(1:1),E001(1:1),E002(1:1);S (1:1));
          S:=E000+E001+E002;
'UNITE' BUS03001(E000(1:1),E001(1:1),E002(1:1);S (1:1));
          S:=E000+E001+E002;
'UNITE' BUS03002(E000(1:1),E001(3:3),E002(2:2),E003(4:4);S (1:4));
          S:=E000&E002&E001&E003;
'UNITE' LMEM(H,I,D(0:3),AD12(1:3),AIG,SS,CCM12;Q(0:3),C(0:3),TCC(1:4)
);
  'SIGNAL' FONCTION(0:7),TC,SC,PE,CU,CD,CS;
  'SIGNAL' CND03001(0:1),CND03000(0:1);
  'HORLOGEMERE' H,I;
  'EXTERNE' COMPTEUR('HORLOGE' ,(1:4),,,,;(1:4),),ML('HORLOGE' ,(1:4),
  (1:4),;(1:4)),VALV3((0:2);(0:7));
  'EXTERNE' NON(,);
  'EXTERNE' EXP03000 (,),EXP03001 (,;),EXP03002 (;),EXP03003 (,,;),
  EXP03004 (,;(0:1)),EXP03005 (,;(0:1));
  'EXTERNE' ET(,;),AND03000(,(0:1);(0:1));
  'EXTERNE'BUS03000((1:1),(1:1),(1:1);(1:1)),BUS03001((1:1),(1:1),(1:1
  );(1:1)),BUS03002((1:1),(3:3),(2:2),(4:4);(1:4));
          ET|5|(FONCTION(4),EXP03001(SS(1),AIG(1);*);PE(1));
          ET|6|(FONCTION(4),1;SC(1));
          EXP03003(SS(1),AIG(1),SC(1);CS(1));
          COMPTEUR(H(1),D(0:3),PE(1),CU(1),CD(1),CCM12(1);Q(0:3),TC(1)
          );
          ML(I(1),Q(0:3),D(0:3),CS(1);D(0:3));
          VALV3(AD12(1:3);FONCTION(0:7));
          AND03000|0|(AIG(1),EXP03004(CU(1),CD(1);*);CND03000(0:1));
          NON|1|(AIG(1),);
          AND03000|1|(NON|1|(;*),EXP03005(CU(1),CD(1);*);CND03001(0:1)
          );
          BUS03000(ET|2|(FONCTION(1),1;*),ET|3|(FONCTION(2),AIG(1);*),
          ET|4|(FONCTION(3),EXP03000(AIG(1);*);*);CU(1));
          BUS03001(ET|7|(FONCTION(5),1;*),ET|8|(FONCTION(6),AIG(1);*),
          ET|9|(FONCTION(7),EXP03002(AIG(1);*);*);CD(1));
          BUS03002(ET|10|(CND03000(0),TC(1);*),ET|11|(CND03000(1),TC(1
          );*),ET|12|(CND03001(0),TC(1);*),ET|13|(CND03001(1),TC(1);*)
          ;TCC(1:4));
```

```
FILE: UNIT4    CASSPGR  P5                                  CAMBRIDGE  MONITOR  SYSTE

'UNITE' EXP04000 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 # E001 ;
'UNITE' EXP04001 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 # E001 ;
'UNITE' EXP04002 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 # E001 ;
'UNITE' EXP04003 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 #- E001 ;
'UNITE' EXP04004 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 # E001 ;
'UNITE' EXP04005 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 # E001 ;
'UNITE' EXP04006 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 # E001 ;
'UNITE' EXP04007 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 #- E001 ;
'UNITE' EXP04008 (E000(0:3),E001;S (0:4));
          S:= E000 & E001 ;
'UNITE' EXP04009 (E000(0:3),E001;S (0:4));
          S:= E000 & E001 ;
'UNITE' EXP04010 (E000(0:3),E001;S (0:4));
          S:= E000 & E001 ;
'UNITE' EXP04011 (E000(0:3),E001;S (0:4));
          S:= E000 & E001 ;
'UNITE' EXP04012 (E000(0:4);S (0:3));
          S:=*DIII E000 ;
'UNITE' EXP04013 (E000,E001(0:3),E002,E003(0:3),E004(0:4),E005,E006(0:
3),E007(0:4),S (0:4)),
          S:= E000 & E001 + E002 & E003 . E004 + E005 &- E006 . E007 ;
'UNITE' EXP04014 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 .- E001 ;
'UNITE' EXP04015 (E000(0:4);S (0:3));
          S:=*DIII E000 ;
'UNITE' EXP04016 (E000,E001(0:3),E002,E003(0:3),E004(0:4),E005,E006(0:
3),E007(0:4),S (0:4)),
          S:= E000 & E001 + E002 & E003 . E004 + E005 & E006 . E007 ;
'UNITE' EXP04017 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 . E001 ;
'UNITE' EXP04018 (E000(0:4);S (0:3));
          S:=*DIII E000 ;
'UNITE' EXP04019 (E000,E001(0:3),E002,E003(0:3),E004(0:4),E005,E006(0:
3),E007(0:4),S (0:4)),
          S:= E000 & E001 + E002 & E003 . E004 + E005 &- E006 . E007 ;
'UNITE' EXP04020 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 .- E001 ;
'UNITE' EXP04021 (E000(0:4);S (0:3));
          S:=*DIII E000 ;
'UNITE' EXP04022 (E000,E001(0:3),E002,E003(0:3),E004(0:4),E005,E006(0:
3),E007(0:4),S (0:4)),
          S:= E000 & E001 + E002 & E003 . E004 + E005 & E006 . E007 ;
'UNITE' EXP04023 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 . E001 ;
'UNITE' ANDC4000 (E000,E (0:3);S (0:3));
          'POUR' I=0 'A' 3
        'DEBUT'
```

```
FILE: UNIT4    CASSPGR  P5                          CAMBRIDGE  MONITOR  SYSTEM

            S (I):=COMD . E(I);
          'FIN';
'UNITE' EXP04024 (E000(0:3),E001(C:3);S (C:3));
            S:=- E000 . E001 ;
'UNITE' EXP04025 (E000(0:3),E001(0:3);S (C:3));
            S:= E000 # E001 ;
'UNITE' EXP04026 (E000(0:3),E001(C:3);S (C:3));
            S:= E000 + E001 ;
'UNITE' EXP04027 (E000(0:3);S (C:3));
            S:=- E000 ;
'UNITE' EXP04028 (E000(0:3),E001(0:3);S (C:3));
            S:= E000 + E001 ;
'UNITE' EXP04029 (E000(0:3);S (C:3));
            S:=- E000 ,
'UNITE' EXP04030 (E000(0:3),E001(C:3);S (C:3));
            S:= E000 # E001 ;
'UNITE' EXP04031 (E000(0:3);S (C:3));
            S:=- E000 ;
'UNITE' EXP04032 (E000(0:3),E001(C:3);S (C:3));
            S:= E000 +- E001 ;
'UNITE' EXP04033 (E000(0:3),E001(C:3);S (C:3));
            S:= E000 . E001 ;
'UNITE' EXP04034 (E000(0:3),E001(C:3);S (C:3));
            S:= E000 .- E001 ;
'UNITE' EXP04035 (E000(0:1),E001,E002;S (C:2));
            S:=-/+ E000 & E001 & E002 ;
'UNITE' EXP04036 (E000(0:3);S );
            S:=/+ E000 ;
'UNITE' REG04000 (H,ENABLE(0:3),INFO(0:3);OUTPUT(0:3));
  'HORLOGEMERE' H;
  'EXTERNE' MCELL('HORLOGE',,,);
            'POUR' I=0 'A' 3
          'DEBUT'
              MCELL (I) (H,ENABLE(I),INFO(I);OUTPUT(I));
          'FIN',
'UNITE' XPA04000(IN,OUT(0:3));
            'POUR' I=0 'A' 3
          'DEBUT'
              OUT(I):=IN;
          'FIN';
'UNITE' BUS04000(E000(0:3),E001(C:3),E002(C:3),E003(0:3),E004(0:3),
E005(0:3),E006(0:3),E007(C:3),E008(0:3),E009(C:3),E010(0:3),E011(0:3),
E012(0:3),E013(0:3),E014(0:3),E015(0:3);S (0:3));
            S:=E000+E001+E002+E003+E004+E005+E006+E007+E008+E009+E010+
            E011+E012+E013+E014+E015;
'UNITE' BUS04001(E000(1:1),E001(1:1),E002(1:1);S (1:1));
            S:=E000+E001+E002;
'UNITE' BUS04002(E000(0:3),E001(C:3),E002(0:3),E003(0:3),E004(0:3),
E005(0:3),E006(0:3),E007(C:3),E008(0:3),E009(C:3),E010(0:3),E011(0:3),
E012(0:3),E013(0:3),E014(C:3),E015(0:3);S (0:3));
            S:=E000+E001+E002+E003+E004+E005+E006+E007+E008+E009+E010+
            E011+E012+E013+E014+E015;
'UNITE' BUS04003(E000(1:1),E001(1:1),E002(1:1);S (1:1));
            S:=E000+E001+E002;
'UNITE' UAL(I1,I2,OP1(0:3),OP2(C:3),COM(0:3),SET,RAZ;S(0:3),SREPT,SZ);
```

```
FILE: UNIT4    CASSPGR  P5                                CAMBRIDGE  MCNITGR  SYSTEM

  'SIGNAL' REPT,L[0:3];
  'SIGNAL' RS7[0:4],RS6[0:3],RS1[0:4],RS2[0:4],RS5[0:4],RS4[0:3],E1[0:
  3],E2[0:3],E3[0:3],E4[0:3],C,RS5[0:3],RS6[0:3];
  'SIGNAL' SGL04005,SGL04002,SGLC4001[0:3],SGLC4000[0:3],CND04001[0:2]
  ,CND04000[0:15];
  'HORLOGEMERE' I1,I2;
  'EXTERNE' VAL94[0:3];[0:15]);
  'EXTERNE' EXP04000 [[0:3],[0:3];[0:3]),EXP04001 [[0:3),[0:3];[0:3)),
  EXP04002 [[0:3),[0:3];[0:3)),EXP04003 [[0:3),[0:3];[0:3)),EXP04004 [
  [0:3),[0:3),[0:3)),EXP04005 [[0:3),[0:3);[0:3)),EXP04006 [[0:3),[0:3
  );[0:3)),EXP04007 [[0:3),[0:3);[0:3)),EXP04008 [[0:3),;[0:4)),
  EXP04009 [[0:3),;[0:4)),EXP04010 [[0:3),;[0:4)),EXP04011 [[0:3),;[0:
  4)),EXP04012 [[0:4),[0:3)),EXP04013 [,[0:3),,[0:3),[0:4),,[0:3),[0:4
  );[0:4)),EXP04014 [[0:3),[0:3);[0:3)),EXP04015 [[0:4);[0:3)),
  EXP04016 [,[0:3),,[0:3),[0:4),,[0:3),[0:4);[0:4)),EXP04017 [[0:3),[0
  :3);[0:3)),EXP04018 [[0:4);[0:3)),EXP04019 [,[0:3),,[0:3),[0:4),,[0:
  3),[0:4);[0:4)),EXP04020 [[0:3),[0:3);[0:3)),EXP04021 [[0:4);[0:3)),
  EXP04022 [,[0:3),,[0:3),[0:4),,[0:3),[0:4);[0:4)),EXP04023 [[0:3),[0
  :3);[0:3)),EXP04024 [[0:3),[0:3);[0:3)),EXP04025 [[0:3),[0:3);[0:3)
  ),EXP04026 [[0:3),[0:3);[0:3)),EXP04027 [[0:3);[0:3)),EXP04028 [[0:3
  ),[0:3);[0:3)),EXP04029 [[0:3);[0:3)),EXP04030 [[0:3),[0:3);[0:3)),
  EXP04031 [[0:3);[0:3)),EXP04032 [[0:3),[0:3);[0:3)),EXP04033 [[0:3),
  [0:3);[0:3)),EXP04034 [[0:3),[0:3);[0:3)),EXP04035 [[0:1),,;[0:2)),
  EXP04036 [[0:3);),
  'EXTERNE' E1[,;),AND04000[,[0:3);[0:3));
  'EXTERNE' XPA04000[,[0:3));
  'EXTERNE' MCELL['HORLOGE',,;),REG04000['HORLOGE',[0:3),[0:3);[0:3));
  'EXTERNE' BUS04000[[0:3),[0:3),[0:3),[0:3),[0:3),[0:3),[0:3),[0:3),[0
  :3),[0:3),[0:3),[0:3),[0:3),[0:3),[0:3),[0:3);[0:3)),BUS04001[[1:1),
  [1:1),[1:1),[1:1)),BUS04002[[0:3),[0:3),[0:3),[0:3),[0:3),[0:3),[0:3
  ),[0:3),[0:3),[0:3),[0:3),[0:3),[0:3),[0:3),[0:3);[0:3)),
  BUS04003[[1:1),[1:1),[1:1);[1:1));
          SREPT[1]:=REPT[1];
          S[0:3]:=L[0:3];
          EXP04000[EXP04001[OP1[C:3),OP2[C:3);*),RS1[1:4);E1[0:3));
          EXP04002[EXP04003[OP1[C:3),OP2[C:3);*),RS7[1:4);E2[0:3));
          EXP04004[EXP04005[OP1[0:3),OP2[C:3);*),RS3[1:4);E3[0:3));
          EXP04006[EXP04007[OP1[C:3),OP2[C:3);*),RS2[1:4);E4[0:3));
          EXP04008[RS4[0:3),REPT[1);RS1[C:4));
          EXP04009[RS5[0:3),1;RS2[C:4));
          EXP04010[RS6[0:3),0;RS3[0:4));
          EXP04011[RS6[0:3),REPT[1);RS7[C:4));
          EXP04012[EXP04013[0,EXP04014[OP1[0:3),OP2[0:3);*),0,OP1[0:3)
          ,RS7[0:4),0,OP2[0:3),RS7[0:4);*);RS6[0:3));
          EXP04015[EXP04016[0,EXP04017[OP1[0:3),OP2[0:3);*),0,OP1[0:3)
          ,RS1[0:4),0,OP2[0:3),RS1[0:4);*);RS4[0:3));
          EXP04018[EXP04019[0,EXP04020[OP1[0:3),OP2[0:3);*),0,OP1[0:3)
          ,RS2[0:4),0,OP2[0:3),RS2[0:4);*);RS5[0:3));
          EXP04021[EXP04022[0,EXP04023[OP1[0:3),OP2[0:3);*),0,OP1[0:3)
          ,RS3[0:4),0,OP2[0:3),RS3[0:4);*);RS6[0:3));
          EXP04036[L[0:3);SZ[1));
          VAL94[COM[0:3);CND04000[0:15));
          EXP04035[COM[0:1),RAZ[1),SET[1);CND04001[0:2));
          REG04000[U[1:1],SGL04001[0:3),SGLC4000[0:3);L[0:3));
          MCELL2[I2[1),SGL04003[1),SGLC4002[1);REPT[1));
```

FILE: UNIT4 CASSPGR P5 CAMBRIDGE MONITOR SYSTEM

```
BUS04000(AND04000|0|(CND04000(0),E1(0:3);*),AND04000|1|(
CND04000(1),E2(0:3);*),AND04000|2|(CND04000(2),E3(0:3);*),
AND04000|3|(CND04000(3),E4(0:3);*),AND04000|4|(CND04000(4),
EXP04024(OP1(0:3),OP2(0:3);*);*),AND04000|5|(CND04000(5),OP2
(0:3);*),AND04000|6|(CND04000(6),EXP04025(OP1(0:3),OP2(0:3);
*);*),AND04000|7|(CND04000(7),EXP04026(OP1(0:3),OP2(0:3);*);
*),AND04000|8|(CND04000(8),EXP04027(EXP04028(OP1(0:3),OP2(0:
3);*);*);*),AND04000|9|(CND04000(9),EXP04029(EXP04030(OP1(0:
3),OP2(0:3),*);*);*),AND04000|10|(CND04000(10),EXP04031(OP2(
0:3),*),*),AND04000|11|(CND04000(11),EXP04032(OP1(0:3),OP2(0
:3);*);*),AND04000|12|(CND04000(12),0000;*),AND04000|13|(
CND04000(13),EXP04033(OP1(0:3),OP2(0:3);*);*),AND04000|14|(
CND04000(14),EXP04034(OP1(0:3),OP2(0:3);*);*),AND04000|15|(
CND04000(15),OP1(0:3);*);SGL04000(0:3));
BUS04001(ET|2|(CND04001(0),RS4(0);*),ET|3|(CND04001(1),0;*),
ET|4|(CND04001(2),1;*);SGL04002(1));
BUS04002(XPA04000|0|(CND04000(0);*),XPA04000|1|(CND04000(1);
*),XPA04000|2|(CND04000(2);*),XPA04000|3|(CND04000(3);*),
XPA04000|4|(CND04000(4);*),XPA04000|5|(CND04000(5);*),
XPA04000|6|(CND04000(6);*),XPA04000|7|(CND04000(7);*),
XPA04000|8|(CND04000(8);*),XPA04000|9|(CND04000(9);*),
XPA04000|10|(CND04000(10);*),XPA04000|11|(CND04000(11);*),
XPA04000|12|(CND04000(12);*),XPA04000|13|(CND04000(13);*),
XPA04000|14|(CND04000(14);*),XPA04000|15|(CND04000(15);*);
SGL04001(0:3));
BUS04003(CND04001(0),CND04001(1),CND04001(2);SGL04003(1));
```

```
FILE: UNIT6    CASSPGR  P5                                  CAMBRIDGE  MONITOR  SYSTEM

'UNITE' AND06000 (COND,E (0:3);S (0:3));
          'POUR' I=0 'A' 3
        'DEBUT'
            S (I):=COND . E(I);
        'FIN';
'UNITE' EXP06000 (E000,E001(0:2);S (0:3));
          S:= E000 * E001 ;
'UNITE' EXP06001 (E000(0:1),E001(0:1);S (0:3));
          S:= E000 * E001 ;
'UNITE' EXP06002 (E000(0:2),E001;S (0:3));
          S:= E000 * E001 ;
'UNITE' BUS06000(E000(1:4),E001(1:4),E002(1:4),E003(1:4);S (1:4));
          S:=E000+E001+E002+E003;
'UNITE' DECO(DP2(1:4),CO2(1:2);SD(1:4));
  'SIGNAL' CND06000(0:3);
  'EXTERNE' VALN2((0:1);(0:3));
  'EXTERNE' EXP06000 (,(0:2);(0:3)),EXP06001 ((0:1),(0:1);(0:3)),
  EXP06002 ((0:2),,(0:3));
  'EXTERNE' AND06000(,(0:3);(0:3));
  'EXTERNE'BUS06000((1:4),(1:4),(1:4),(1:4);(1:4));
          VALN2(CO2(1:2);CND06000(0:3));
          BUS06000(AND06000|0|(CND06000(0),DP2(1:4);*),AND06000|1|(
          CND06000(1),EXP06000(0,DP2(1:3);*);*),AND06000|2|(CND06000(2
          ),EXP06001(00,DP2(1:2);*);*),AND06000|3|(CND06000(3),
          EXP06002(000,DP2(1);*);*);SD(1:4));
```

```
FILE: UNIT5|   CASSPGR  P5                          CAMBRIDGE  MONITOR  SYSTEM

'UNITE' AND05000 (COND,E (0:3);S (0:3));
          'POUR' I=0 'A' 3
        'DEBUT'
            S (I):=COND . E(I);
        'FIN';
'UNITE' EXP05000 (E000(0:2),E001;S (0:3));
          S:= E000 & E001 ;
'UNITE' EXP05001 (E000(0:1),E001(0:1);S (0:3));
          S:= E000 & E001 ;
'UNITE' EXP05002 (E000,E001(0:2);S (0:3));
          S:= E000 & E001 ;
'UNITE' BUS05000(E000(1:4),E001(1:4),E002(1:4),E003(1:4);S (1:4));
          S:=E000+E001+E002+E003;
'UNITE' DECG(DP1(1:4),C01(1:2);SG(1:4));
  'SIGNAL' CND05000(0:3);
  'EXTERNE' VALN2((0:1);(0:3));
  'EXTERNE' EXP05000 ((0:2),;(0:3)),EXP05001 ((0:1),(0:1);(0:3)),
  EXP05002 (,(0:2);(0:3));
  'EXTERNE' AND05000(,(0:3);(0:3));
  'EXTERNE'BUS05000((1:4),(1:4),(1:4),(1:4);(1:4));
          VALN2(C01(1:2);CND05(CC(0:3));
          BUS05000(AND05000|0|(CND05000(0),DP1(1:4);*),AND05000|1|(
          CND05000(1),EXP05000(DP1(2:4),C;*);*),AND05000|2|(CND05000(2
          ),EXP05001(DP1(3:4),C0;*);*),AND05000|3|(CND05000(3),
          EXP05002(DP1(4),C0C;*);*);SG(1:4));
```

```
FILE: UNIT7    CASSPUR  P5                          CAMBRIDGE  MONITOR  SYSTEM

'UNITE' AND07000 (COND,E (0:1);S (0:1));
          'POUR' I=0 'A' 1
        'DEBUT'
            S (I):=COND . E(I);
        'FIN';
'UNITE' AND07001 (COND,E (0:3);S (0:3));
          'POUR' I=0 'A' 3
        'DEBUT'
            S (I):=COND . E(I);
        'FIN';
'UNITE' EXP07000 (E000,S );
          S:=- E000 ;
'UNITE' EXP07001 (E000,E001;S );
          S:= E000 + E001 ;
'UNITE' EXP07002 (E000,S );
          S:=- E000 ;
'UNITE' EXP07003 (E000,E001;S );
          S:= E000 + E001 ;
'UNITE' REG07000 (H,ENABLE(0:1),INPU(0:1);OUTPUT(0:1));
  'HORLOGEMERE' H,
  'EXTERNE' MODELE('HORLOGE',,;);
          'POUR' I=0 'A' 1
        'DEBUT'
            MODELE (I) (H,ENABLE(I),INPU(I);OUTPUT(I));
        'FIN',
'UNITE' XPA07000(IN,OUT(0:1));
          'POUR' I=0 'A' 1
        'DEBUT'
            OUT(I):=IN;
        'FIN',
'UNITE' BUS07000(E000(1:2),E001(1:2),E002(1:2),E003(1:2),E004(1:2);S (
1:2));
          S:=E000+E001+E002+E003+E004;
'UNITE' BUS07001(E000(1:2),E001(1:2),E002(1:2),E003(1:2),E004(1:2);S (
1:2));
          S:=E000+E001+E002+E003+E004;
'UNITE' BUS07002(E000(1:2),E001(1:2),E002(1:2),E003(1:2),E004(1:2),
E005(1:2),E006(1:2),E007(1:2);S (1:2));
          S:=E000+E001+E002+E003+E004+E005+E006+E007;
'UNITE' BUS07003(E000(1:2),E001(1:2),E002(1:2),E003(1:2),E004(1:2),
E005(1:2),E006(1:2),E007(1:2);S (1:2));
          S:=E000+E001+E002+E003+E004+E005+E006+E007;
'UNITE' DEC(I,CDEC,COMDEC(1:6),DP1(1:4),DP2(1:4),SA1(1:2),SA2(1:4);SG(
1:4),SD(1:4)),
  'SIGNAL' A(1:2);
  'SIGNAL' CDC(1:2),CDD(1:2),CD1(1:2),CD2(1:2);
  'SIGNAL' SGL07001(1:2),SGL07000(1:2),CND07004(0:0),CND07003(0:0),
  CND07002(0:3),CND07001(0:3),CND07000(0:3);
  'HORLOGEMERE' I;
  'EXTERNE' VAL(2((0:1),(0:3)),DEC3((1:4),(1:2);(1:4)),DECD((1:4),(1:2
  );(1:4));
  'EXTERNE' NON(;);
  'EXTERNE' EXP07000 (,),EXP07001 (,;),EXP07002 (;),EXP07003 (,;);
  'EXTERNE' AND07000(,(0:1);(0:1)),AND07001(,(0:3);(0:3));
  'EXTERNE' XPA07000(,(0:1));
```

FILE: UNIT7 CASSPGR P5 CAMBRIDGE MONITOR SYSTEM

```
  'EXTERNE' REG07000('HORLOGE',(0:1),(0:1);(0:1));
  'EXTERNE'BUS07000((1:2),(1:2),(1:2),(1:2),(1:2);(1:2)),BUS07001((1:2
  ),(1:2),(1:2),(1:2),(1:2);(1:2)),BUS07002((1:2),(1:2),(1:2),(1:2),(1
  :2),(1:2),(1:2),(1:2);(1:2)),BUS07003((1:2),(1:2),(1:2),(1:2),(1:2),
  (1:2),(1:2),(1:2),(1:2));
          AND07000|18|(CND07003(0),CG1(1:2);CGG(1:2));
          AND07000|19|(CND07004(0),CG2(1:2);CGD(1:2));
          VALN2(A(1:2),);
          DECG(GP1(1:4),CGG(1:2);SG(1:4));
          DECD(GP2(1:4),CGD(1:2);SD(1:4));
          AND07001|0|(CDEC(1),VALN2(;*);CND07000(0:3));
          NON|1|(CDEC(1););
          AND07001|1|(NON|1|(;*),VALN2(;*);CND07001(0:3));
          AND07001|2|(CGMDEC(2),SA2(1:4);CND07002(0:3));
          EXP07000(EXP07001(CGMDEC(6),CGMDEC(8);*);CND07003(0));
          EXP07002(EXP07003(CGMDEC(7),CGMDEC(8);*);CND07004(0));
          REG07000|0|(1(1),SGLG7001(1:2),SGLG7000(1:2);A(1:2));
          BUS07000(AND07000|0|(CDEC(1),A(1:2);*),AND07000|5|(CND07001(
          0),00;*),AND07000|6|(CND07001(1),11;*),AND07000|7|(CND07001(
          2),10;*),AND07000|8|(CND07001(3),01;*);CG1(1:2));
          BUS07001(AND07000|1|(CND07000(0),00;*),AND07000|2|(CND07000(
          1),11;*),AND07000|3|(CND07000(2),10;*),AND07000|4|(CND07000(
          3),01;*),AND07000|9|(NON|1|(;*),A(1:2);*);CG2(1:2));
          BUS07002(AND07000|10|(CGMDEC(1),SA1(1:2);*),AND07000|11|(
          CND07002(0),00;*),AND07000|12|(CND07002(1),01;*),AND07000|13
          |(CND07002(2),10;*),AND07000|14|(CND07002(3),11;*),AND07000|
          15|(CGMDEC(5),00;*),AND07000|16|(CGMDEC(4),01;*),AND07000|17
          |(CGMDEC(5),10;*);SGLG7000(1:2));
          BUS07003(XPA07000|0|(CGMDEC(1);*),XPA07000|1|(CND07002(0);*)
          ,XPA07000|2|(CND07002(1);*),XPA07000|3|(CND07002(2);*),
          XPA07000|4|(CND07002(3);*),XPA07000|5|(CGMDEC(3);*),XPA07000
          |6|(CGMDEC(4);*),XPA07000|7|(CGMDEC(5);*);SGL07001(1:2));
```

```
FILE: UNIT8    CASSPUR  P5                                  CAMBRIDGE  MONITOR  SYSTEM

'UNITE' ANDO8000 (COND,E (0:15,0:0);S (0:15,0:0));
          'PUUR' I=0 'A' 15
        'DEBUT'

          'PUUR' J=0 'A' 0
          'DEBUT'
            S (I,J):=COND . E(I,J);
          'FIN';
        'FIN';
'UNITE' ANDO8001 (COND,E (0:3);S (0:3));
          'PUUR' I=0 'A' 3
        'DEBUT'
            S (I):=COND . E(I);
        'FIN';
'UNITE' EXPC8000 (E000(0:15),EC01(0:15);S (0:15));
          S:= E000 + E001 ;
'UNITE' REGC8000 (H,ENABLE(0:15,0:3),INFO(0:15,0:3);OUTPUT(0:15,0:3));
  'HORLOGEMERKE' H,
  'EXTERNE' MCELL('HORLOGE',,,);
          'PUUR' I=0 'A' 15
        'DEBUT'
          'PUUR' J=0 'A' 3
          'DEBUT'
            MCELL (4.I+J) (H,ENABLE(I,J),INFO(I,J);OUTPUT(I,J));
          'FIN';
        'FIN';
'UNITE' XPAC8000(IN;OUT(0:15));
          'PUUR' I=0 'A' 15
        'DEBUT'
            OUT(I):=IN;
        'FIN',
'UNITE' XPAO8001(IN,OUT(0:3));
          'PUUR' I=0 'A' 3
        'DEBUT'
            OUT(I):=IN;
        'FIN';
'UNITE' REGO8001 (H,ENABLE(0:3),INFO(0:3);OUTPUT(0:3));
  'HORLOGEMERKE' H;
  'EXTERNE' MCELL('HORLOGE',,,);
          'PUUR' I=0 'A' 3
        'DEBUT'
            MCELL (I) (H,ENABLE(I),INFO(I);OUTPUT(I));
        'FIN';
'UNITE' BUSC8000(E000(1:16),E001(1:16),E002(1:16),E003(1:16);S (1:16)
);
          S:=E000+E001+E002+E003;
'UNITE' BUSO8001(E000(1:16,0:0),EC01(1:16,1:1),E002(1:16,2:2),E003(1:
16,3:3);S (1:16,0:3));
          S:=*P(*P(E000)&*P(E001)&*P(E002)&*P(E003));
'UNITE' BUSO8002(E000(1:16,0:0),EC01(1:16,1:1),EC02(1:16,2:2),E003(1:
16,3:3);S (1:16,0:3));
          S:=*P(*P(E000)&*P(E001)&*P(E002)&*P(E003));
'UNITE' SPES(12,13,SI(1:16),SK(1:16),EAD(1:4),CECH,CSP,CC(2:2);SSPES(1
:16));
  'SIGNAL' AD3(1:4),SPE(1:16,0:3);
```

FILE: UNIT8 CASSPGR P5 CAMBRIDGE MONITOR SYSTEM

```
  'SIGNAL' INSP(1:16),
  'SIGNAL' SGLC8001(1:16,0:3),SGLC8000(1:16,0:3),CND08001(0:3),
  CND08000(0:3);
  'HORLOGEMERE' 12,13,
  'EXTERNE' VALN2((0:1),(0:3));
  'EXTERNE' NON(;),
  'EXTERNE' EXP08000 ((0:15),(C:15);(0:15));
  'EXTERNE' AND08000(,(0:15);(C:15)),AND08001(,(0:3);(0:3));
  'EXTERNE' XPAC8000(;(0:15)),XPAC8001(;(C:3));
  'EXTERNE' REG08000('HORLOGE',(C:15,0:3),(0:15,0:3);(0:15,0:3)),
  REG08001('HORLOGE',(0:3),(0:3);(C:3));
  'EXTERNE' BUS08000((1:16),(1:16),(1:16),(1:16);(1:16)),BUS08001((1:16
  ,0:0),(1:16,1:1),(1:16,2:2),(1:16,3:3);(1:16,0:3)),BUS08002((1:16,0:
  0),(1:16,1:1),(1:16,2:2),(1:16,3:3);(1:16,C:3));
          EXP08000(AND08000|9|(NON|1|(;*),SR(1:16);*),AND08000|8|(CECH
          (1),ST(1:16);*);INSP(1:16));
          VALN2|1|(ADS(1:2);CND08000(0:3));
          VALN2|2|(ADS(3:4););
          AND08001|0|(CSP(1),VALN2|2|(;*);CND08001(0:3));
          NON|1|(CECH(1););
          REG08000|0|(12(1),SGLC8001(1:16,0:3),SGLC8000(1:16,0:3);SPE(
          1:16,0:3));
          REG08001|0|(13(1),XPAC8001|0|(CC(2);*),AND08001|1|(CC(2),EAD
          (1:4);*);ADS(1:4));
          BUS08000(AND08000|0|(CND08000(0),SPE(1:16,0);*),AND08000|1|(
          CND08000(1),SPE(1:16,1);*),AND08000|2|(CND08000(2),SPE(1:16,
          2);*),AND08000|3|(CND08000(3),SPE(1:16,3);*);SSPES(1:16));
          BUS08001(AND08000|4|(CND08001(0),INSP(1:16);*),AND08000|5|(
          CND08001(1),INSP(1:16);*),AND08000|6|(CND08001(2),INSP(1:16)
          ;*),AND08000|7|(CND08001(3),INSP(1:16);*);SGL08000(1:16,0:3)
          );
          BUS08002(XPAC8000|0|(CND08001(0);*),XPAC8000|1|(CND08001(1);
          *),XPAC8000|2|(CND08001(2);*),XPAC8000|3|(CND08001(3);*);
          SGLC8001(1:16,0:3));
```

```
FILE: UNIT9    CASSPGR  P5                                  CAMBRIDGE  MONITOR  SYSTEM

'UNITE' EXP09000 (E000,E001;S (0:1));
          S:= E000 & E001 ;
'UNITE' AND09000 (COND,E (0:1);S (0:1));
          'POUR' I=0 'A' 1
        'DEBUT'
           S (I):=COND . E(I);
        'FIN',
'UNITE' EXP09001 (E000(0:1);S );
          S:=/. E000 ;
'UNITE' EXP09002 (E000(0:1);S );
          S:=/.- E000 ;
'UNITE' EXP09003 (E000(0:1),E001;S (0:1));
          S:=*D|1| E000 & E001 ;
'UNITE' EXP09004 (E000(0:1),E001(0:1),E002(0:1),E003(0:1),E004(0:1),
E005(0:1);S (0:1));
          S:= E000 . E001 + E002 . E003 + E004 . E005 ;
'UNITE' EXP09005 (E000,E001,E002;S (0:2));
          S:= E000 & E001 & E002 ;
'UNITE' EXP09006 (E000,E001;S );
          S:= E000 + E001 ;
'UNITE' EXP09007 (E000(0:1),E001(0:1);S (0:1));
          S:= E000 # E001 ;
'UNITE' EXP09008 (E000(0:1),E001(0:1);S (0:1));
          S:= E000 # E001 ;
'UNITE' REG09000 (H,ENABLE(0:1),INFO(0:1);OUTPUT(0:1));
  'HORLOGEMERE' H,
  'EXTERNE' MCELL('HORLOGE',,;);
          'POUR' I=0 'A' 1
        'DEBUT'
           MCELL (I) (H,ENABLE(I),INFO(I);OUTPUT(I));
        'FIN';
'UNITE' XPA09000(IN,OUT(0:1));
          'POUR' I=0 'A' 1
        'DEBUT'
           OUT(I):=IN;
        'FIN';
'UNITE' BUS09000(E000(0:1),E001(0:1);S (0:1));
          S:=E000+E001;
'UNITE' BUS09001(E000(1:1),E001(1:1);S (1:1));
          S:=E000+E001;
'UNITE' BUS09002(E000(1:2),E001(1:2),E002(1:2);S (1:2));
          S:=E000+E001+E002;
'UNITE' BUS09003(E000(1:2),E001(1:2),E002(1:2);S (1:2));
          S:=E000+E001+E002;
'UNITE' COMPT(H,P(0:1),PE,CU,CD,RAZ;Q(0:1),TC);
  'SIGNAL' ADD(1:2),
  'SIGNAL' MUX(0:1),MUX(0:1),
  'SIGNAL' SGL09001(1:2),SGL09000(1:2),CND09001(0:2),CND09000(0:1);
  'HORLOGEMERE' H;
  'EXTERNE' EXP09000 (,;(0:1)),EXP09001 ((0:1);),EXP09002 ((0:1);),
  EXP09003 ((0:1),;(0:1)),EXP09004 ((0:1),(0:1),(0:1),(0:1),(0:1),(0:1
  );(0:1)),EXP09005 (,,;(0:2)),EXP09006 (,;),EXP09007 ((0:1),(0:1);(0:
  1)),EXP09008 ((0:1),(0:1);(0:1));
  'EXTERNE' BT(,;),AND09000(,(0:1);(0:1));
  'EXTERNE' XPA09000(,(0:1));
```

```
FILE: UNIT9     CASSPGR  P5                          CAMBRIDGE  MONITOR  SYSTEM

  'EXTERNE' REG09000('HORLOGE',(C:1),(0:1);(0:1));
  'EXTERNE'BUS09000((0:1),(0:1);(0:1)),BUSC9C01((1:1),(1:1);(1:1)),
  BUS09002((1:2),(1:2),(1:2);(1:2)),BUSC9C03((1:2),(1:2),(1:2);(1:2));
          Q(0:1):=ADD(1:2);
          EXP09005(EXP09004(ADD(1:2),M02(C:1),ADD(1:2),M01(0:1),M01(0:
          1),M02(0:1),*),C;M01(C:1));
          EXP09000(CU(1),CD(1);CND09000(C:1));
          EXP09005(PE(1),EXP09006(CU(1),CD(1);*),RAZ(1);CND09001(0:2))
          ;
          REG09000|0|(H(1),SGLC9C01(1:2),SGLC9000(1:2);ACC(1:2));
          BUS09000(AND09000|0|(CND09000(0),C1;*),AND09000|1|(CND09000(
          1),11;*);M02(0:1));
          BUS09001(ET|2|(CND09000(0),EXP09001(ADD(1:2);*);*),ET|3|(
          CND09000(1),EXP09002(ADD(1:2);*);*);TC(1));
          BUS09002(AND09000|2|(CND09001(0),P(0:1);*),AND09000|3|(
          CND09001(1),EXP09007(ADD(1:2),EXP09008(M02(0:1),M01(0:1);*);
          *);*),AND09000|4|(CND09001(2),OC;*);SGL09000(1:2));
          BUS09003(XPA09000|0|(CND09001(0);*),XPA09000|1|(CND09001(1);
          *),XPA09000|2|(CND09001(2);*);SGL09001(1:2));
```

```
FILE: UNIT10   CASSPGR  P5                                  CAMBRIDGE  MONITOR  SYSTE

'UNITE' EXP10000 (E000,E001(0:1);S (0:2));
          S:= E000 & E001 ;
'UNITE' EXP10001 (E000,E001(0:1);S (0:2));
          S:= E000 & E001 ;
'UNITE' AND10000 (COND,E (0:3);S (0:3));
          'POUR' I=0 'A' 3
        'DEBUT'
            S (I):=COND . E(I);
        'FIN',
'UNITE' EXP10002 (E000(0:4),E001(0:2);S (0:7));
          S:= E000 & E001 ;
'UNITE' EXP10003 (E000;S );
          S:=- E000 ,
'UNITE' EXP10004 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 + E001 ;
'UNITE' BUS10000(E000(1:4),E001(1:4),E002(1:4),E003(1:4),E004(1:4),
E005(1:4),E006(1:4),E007(1:4);S (1:4));
          S:=E000+E001+E002+E003+E004+E005+E006+E007;
'UNITE' BUS10001(E000(1:4),E001(1:4),E002(1:4),E003(1:4),E004(1:4),
E005(1:4),E006(1:4),E007(1:4);S (1:4));
          S:=E000+E001+E002+E003+E004+E005+E006+E007;
'UNITE' BUS10002(E000(0:9),E001(0:0),E002(1:1),E003(4:4),E004(5:5);S (
0:9));
          'SIGNAL' S001 (0:9),S002 (0:9),S003 (0:9),S004 (0:9),S005 (0
          :9),
          'POUR' I=0 'A' 9
        'DEBUT'
            'SI' I>=0 & I<=9
             'ALORS' 'DEBUT'S001(I):=E000(I);'FIN'
             'SINON' 'DEBUT' S001(I):=0;'FIN';
            'SI' I=0
             'ALORS' 'DEBUT'S002(I):=E001(I);'FIN'
             'SINON' 'DEBUT' S002(I):=0;'FIN';
            'SI' I=1
             'ALORS' 'DEBUT'S003(I):=E002(I);'FIN'
             'SINON' 'DEBUT' S003(I):=0;'FIN';
            'SI' I=4
             'ALORS' 'DEBUT'S004(I):=E003(I);'FIN'
             'SINON' 'DEBUT' S004(I):=0;'FIN';
            'SI' I=5
             'ALORS' 'DEBUT'S005(I):=E004(I);'FIN'
             'SINON' 'DEBUT' S005(I):=0;'FIN';
            S(I):=S001(I)+S002(I)+S003(I)+S004(I)+S005(I);
        'FIN';
'UNITE' BUS10003(E000(1:4),E001(5:8);S (1:8));
          S:=E000&E001;
'UNITE' Z0001(H,I1,I2,I3,ST(1:16);SSPES(1:16),CECH(1:3),CM(1:3));
  'SIGNAL' TE(1:8),E1(1:4),E2(1:4),S(1:4),AD1(1:4),C1(1:4),AD2(1:4),C2
  (1:4),SR1(1:16),SR2(1:4),OP1(1:4),OP2(1:4),CONS(1:4),COMMANDE(1:15),
  TEST(0:9),AIG,CDEC,SS,CSP,CE1(1:2),CE2(1:2),CR(1:2),CUAL(1:4),AD12(1
  :3),ADD(1:2),CONS2(1:2),CC(1:2),CONTR(1:43);
  'SIGNAL' CND10003(0:0),CND10002(0:0),CND10001(0:7),CND10000(0:7);
  'HORLOGEMERE' H,I1,I2,I3;
  'EXTERNE' VALS((0:2);(0:7)),LMEM('HORLOGE' ,'HORLOGE' ,(1:4),(1:3),
  ,,;(1:4),(1:4),(1:4)),CONTROLE('HORLOGE' ,'HORLOGE' ,'HORLOGE' ,(1:4
```

```
FILE: UNIT10   CASSPGR  P5                                  CAMBRIDGE  MONITOR  SYSTEM

),(0:9),,(1:6),(1:43),(1:3),(1:3)),UAL('HORLOGE' ,'HORLOGE' ,(1:4),(
1:4),(1:4),,,(1:4),,),DEC('HORLOGE' ,,(1:8),(1:4),(1:4),(1:2),(1:4);
(1:4),(1:4)),SPES('HORLOGE' ,'HORLOGE' ,(1:16),(1:16),(1:4),,,;(1:16
)),R('HORLOGE' ,'HORLOGE' ,(1:2),,(1:2),(1:3),(1:2),(1:4),(1:16);(1:
16),(1:4),,);
'EXTERNE' EXP10000 (,(0:1);(0:2)),EXP10001 (,(0:1);(0:2)),EXP10002 (
(0:4),(0:2);(0:7)),EXP10003 (;),EXP10004 ((0:3),(0:3);(0:3));
'EXTERNE' ET(,;),AND10000(,(0:3);(0:3));
'EXTERNE'BUS10000((1:4),(1:4),(1:4),(1:4),(1:4),(1:4),(1:4),(1:4);(1
:4)),BUS10001((1:4),(1:4),(1:4),(1:4),(1:4),(1:4),(1:4),(1:4);(1:4)
),BUS10002((6:9),(0:0),(1:1),(4:4),(5:5);(0:9)),BUS10003((1:4),(5:8)
;(1:8));
        CR(1:2):=CONTR(1:2);
        CSP(1):=CONTR(3);
        CUAL(1:4):=CONTR(4:7);
        CE1(1:2):=CONTR(8:9);
        SS(1):=CONTR(10);
        AD12(1:3):=CONTR(11:13);
        ADD(1:2):=CONTR(14:15);
        CE2(1:2):=CONTR(16:17);
        CONS2(1:2):=CONTR(18:19);
        CONS(1:4):=CONTR(20:23);
        CDEC(1):=CONTR(24);
        CC(1:2):=CONTR(25:26);
        AIG(1):=CONTR(27);
        COMMANDE(1:15):=CONTR(28:42);
        VAL03|1|(EXP10000(AIG(1),CE1(1:2);*);CND10000(0:7));
        VAL03|2|(EXP10001(AIG(1),CE2(1:2);*);CND10001(0:7));
        R(12(1),13(1),ADD(1:2),COMMANDE(11),CR(1:2),COMMANDE(8:10),
        CONS2(1:2),S(1:4),SSPES(1:16);SR1(1:16),SR2(1:4),,);
        SPES(12(1),13(1),ST(1:16),SR1(1:16),CONS(1:4),CECH(1),CSP(1)
        ,CC(2),SSPES(1:16));
        DEC(13(1),CDEC(1),EXP10002(COMMANDE(1:5),COMMANDE(13:15);*),
        E1(1:4),E2(1:4),SR1(13:14),S(1:4);OP1(1:4),OP2(1:4));
        UAL(11(1),12(1),OP1(1:4),OP2(1:4),CUAL(1:4),COMMANDE(6),
        COMMANDE(7),S(1:4),,);
        LME01|1|(13(1),12(1),S(1:4),AD12(1:3),AIG(1),SS(1),COMMANDE(
        12);AD1(1:4),01(1:4),);
        LME02|2|(13(1),12(1),S(1:4),AD12(1:3),EXP10003(AIG(1);*),SS(1
        ),COMMANDE(12);AD2(1:4),02(1:4),);
        CONTROLE(M(1),I1(1),I3(1),S(1:4),TEST(0:9),,;CONTR(1:43),CM(
        1:3),CECH(1:3));
        ET|2|(CND10001(0),CC(1);CND10002(0));
        ET|3|(CND10001(4),CC(1);CND10003(0));
        BUS10000(AND10000|0|(CND10000(0),SR2(1:4);*),AND10000|1|(
        CND10000(1),01(1:4);*),AND10000|2|(CND10000(2),C2(1:4);*),
        AND10000|3|(CND10000(3),AD1(1:4);*),AND10000|4|(CND10000(4),
        SR2(1:4);*),AND10000|5|(CND10000(5),02(1:4);*),AND10000|6|(
        CND10000(6),01(1:4);*),AND10000|7|(CND10000(7),AD2(1:4);*);
        E1(1:4));
        BUS10001(AND10000|8|(CND10002(0),CONS(1:4);*),AND10000|9|(
        CND10001(1),SR2(1:4);*),AND10000|10|(CND10001(2),C1(1:4);*),
        AND10000|11|(CND10001(3),02(1:4);*),AND10000|12|(CND10003(0)
        ,CONS(1:4);*),AND10000|13|(CND10001(5),SR2(1:4);*),AND10000|
        14|(CND10001(6),02(1:4);*),AND10000|15|(CND10001(7),01(1:4);
```

FILE: UNIT10 CASSPGR P5 CAMBRIDGE MONITOR SYSTEM

```
*);L2(1:4));
BUS10002(EXP10004(TE(1:4),TE(5:8);*),R(,,,,,,,,;,,*,),R(,,,,
,,,;,,,*),UAL(,,,,,,;,*,),UAL(,,,,,,;,,*);TEST(0:9));
BUS10003(LMEM(1)(,,,,,,;,,*),LMEM(2)(,,,,,,;,,*);TE(1:8));
```

```
FILE: UNIT11   CASSPUR  P5                                  CAMBRIDGE  MONITOR  SYSTEM

'UNITE' EXP11000 (E000(0:3),E001(0:3);S (0:7));
          S:= E000 & E001 ;
'UNITE' AND11000 (COND,E (0:3);S (0:3));
          'POUR' I=0 'A' 3
        'DEBUT'
            S (I):=COND . E(I);
        'FIN';
'UNITE' EXP11001 (E000(0:3),E001(0:3),E002(0:3),E003(0:3);S (0:15));
          S:= E000 & E001 & E002 & E003 ;
'UNITE' EXP11002 (E000,E001;S );
          S:= E000 + E001 ;
'UNITE' EXP11003 (E000,E001;S );
          S:= E000 . E001 ;
'UNITE' EXP11004 (E000,E001;S );
          S:= E000 . E001 ;
'UNITE' EXP11005 (E000,E001;S );
          S:= E000 + E001 ;
'UNITE' EXP11006 (E000,E001;S );
          S:= E000 . E001 ;
'UNITE' EXP11007 (E000,E001;S );
          S:= E000 . E001 ;
'UNITE' EXP11008 (E000,E001;S );
          S:= E000 + E001 ;
'UNITE' EXP11009 (E000,E001;S );
          S:= E000 . E001 ;
'UNITE' EXP11010 (E000,E001;S );
          S:= E000 . E001 ;
'UNITE' EXP11011 (E000,E001;S );
          S:= E000 + E001 ;
'UNITE' EXP11012 (E000,E001,E002,E003;S );
          S:= E000 . E001 . E002 . E003 ;
'UNITE' BUS11000(E000(0:3,0:0),E001(0:3,1:1),E002(0:3,2:2),E003(0:3,3:
3),E004(0:3,0:0),E005(0:3,1:1),E006(0:3,2:2),E007(0:3,3:3);S (0:3,0:3)
);
          'SIGNAL' S001 (0:3,0:3),S002 (0:3,0:3),S003 (0:3,0:3),S004 (
          0:3,0:3),S005 (0:3,0:3),S006 (0:3,0:3),S007 (0:3,0:3),S008 (
          0:3,0:3);
          'POUR' I=0 'A' 3
        'DEBUT'
            'POUR' J=0 'A' 3
          'DEBUT'
              'SIA'  J=0
               'ALORS' 'DEBUT'S001(I,J):=E000(I,J);'FIN'
               'SINON' 'DEBUT' S001(I,J):=0;'FIN';
              'SIA'  J=1
               'ALORS' 'DEBUT'S002(I,J):=E001(I,J);'FIN'
               'SINON' 'DEBUT' S002(I,J):=0;'FIN';
              'SIA'  J=2
               'ALORS' 'DEBUT'S003(I,J):=E002(I,J);'FIN'
               'SINON' 'DEBUT' S003(I,J):=0;'FIN';
              'SIA'  J=3
               'ALORS' 'DEBUT'S004(I,J):=E003(I,J);'FIN'
               'SINON' 'DEBUT' S004(I,J):=0;'FIN';
              'SIA'  J=0
               'ALORS' 'DEBUT'S005(I,J):=E004(I,J);'FIN'
```

```
FILE: UNIT11   CASSPOR  P5                                CAMBRIDGE  MONITOR  SYSTEM

                'SINON' 'DEBUT' SOC5(I,J):=0;'FIN';
              'SIA'  J=1
               'ALORS' 'DEBUT'SOC6(I,J):=EOC5(I,J);'FIN'
               'SINON' 'DEBUT' SOO6(I,J):=0;'FIN';
              'SIA'  J=2
               'ALORS' 'DEBUT'SOC7(I,J):=EOO6(I,J);'FIN'
               'SINON' 'DEBUT' SOC7(I,J):=0;'FIN';
              'SIA'  J=3
               'ALORS' 'DEBUT'SOO8(I,J):=EOC7(I,J);'FIN'
               'SINON' 'DEBUT' SOC8(I,J):=0;'FIN';
              S(I,J):=SOO1(I,J)+SOO2(I,J)+SOO3(I,J)+SOO4(I,J)+SOO5(I,J
              )+SOO6(I,J)+SOO7(I,J)+SOC8(I,J);
          'FIN';
        'FIN';
'UNITE' BUS11001(EOOO(1:4),EOO1(1:4),EOO2(1:4),EOO3(1:4);S (1:4));
          S:=EOOO+EOO1+EOO2+EOO3;
'UNITE' BUS11002(EOOO(0:3,0:0),EOO1(0:3,1:1),EOO2(0:3,2:2),EOO3(0:3,3:
3);S (0:3,0:3));
          S:=*P(*P(EOOO)&*P(EOO1)&*P(EOO2)&*P(EOO3));
'UNITE' R(I,H,ADO(1:2),RAZ,CR(1:2),RCOM(1:3),EADO(1:2),S(1:4),SSPES(1:
16);SR1(1:16),SR2(1:4),TC,TC5);
  'SIGNAL' TC4,TC1,Q(0:1),DOLQ(1:4),TC3,TC2,CAD(1:3),CR3(1:4),CR2(1:4)
  ,RR2(0:3,0:3),RR1(0:3,0:3),BL(1:16);
  'SIGNAL' CND11000(0:7);
  'HORLOGEMERE' H,1;
  'EXTERNE' COMPT('HORLOGE' ,(1:2),,,,,;(1:2),),COMPTEUR('HORLOGE' ,(1:
  4),,,,,;(1:4),),DECOD((1:2),(1:2),(1:2);(1:3),(1:4),(1:4),(1:4));
  'EXTERNE' EXP11000 ((0:3),(0:3);(0:7)),EXP11001 ((0:3),(0:3),(0:3),(
  0:3);(0:15)),EXP11002 (,;),EXP11003 (,;),EXP11004 (,;),EXP11005 (,;
  ),EXP11006 (,;),EXP11007 (,;),EXP11008 (,;),EXP11009 (,;),EXP11010 (
  ,;),EXP11011 (,;),EXP11012 (,,,,;);
  'EXTERNE' AND11000(,(0:3);(0:3));
  'EXTERNE'BUS11000((0:3,0:0),(0:3,1:1),(0:3,2:2),(0:3,3:3),(0:3,0:0),
  (0:3,1:1),(0:3,2:2),(0:3,3:3);(0:3,0:3)),BUS11001((1:4),(1:4),(1:4),
  (1:4);(1:4)),BUS11002((0:3,0:0),(0:3,1:1),(0:3,2:2),(0:3,3:3);(0:3,0
  :3));
          EXP11001(RR2(0:3,0),RR2(0:3,1),RR2(0:3,2),RR2(0:3,3);SR1(1:
          16));
          EXP11012(TC1(1),TC2(1),TC3(1),TC4(1);TC5(1));
          COMPT(H(1),EADO(1:2),CAD(1),CAD(2),CAD(3),RAZ(1);Q(0:1),TC(1
          ));
          DECOD(ADO(1:2),CR(1:2),Q(0:1);CAD(1:3),DOLQ(1:4),CR3(1:4),
          CR2(1:4));
          COMPTEUR(1)((1(1),RR1(0:3,0),EXP11002(CR2(1),CR3(1);*),
          EXP11003(TC2(1),RCOM(1);*),EXP11004(TC2(1),RCOM(2);*),RCOM(3
          );,TC1(1));
          COMPTEUR(2)((1(1),RR1(0:3,1),EXP11005(CR2(2),CR3(2);*),
          EXP11006(TC3(1),RCOM(1);*),EXP11007(TC3(1),RCOM(2);*),RCOM(3
          );,TC2(1));
          COMPTEUR(3)((1(1),RR1(0:3,2),EXP11008(CR2(3),CR3(3);*),
          EXP11009(TC4(1),RCOM(1);*),EXP11010(TC4(1),RCOM(2);*),RCOM(3
          );,TC3(1));
          COMPTEUR(4)((1(1),RR1(0:3,3),EXP11011(CR2(4),CR3(4);*),RCOM(1
          ),RCOM(2),RCOM(3);,TC4(1));
          EXP11000(CR2(1:4),CR3(1:4);CND11000(0:7));
```

FILE: UNIT11 CASSPGR P5 CAMBRIDGE MONITOR SYSTEM

```
BUS11000(AND11000|0|(CND11000(0),S(1:4);*),AND11000|1|(
CND11000(1),S(1:4);*),AND11000|2|(CND11000(2),S(1:4);*),
AND11000|3|(CND11000(3),S(1:4);*),AND11000|4|(CND11000(4),
SSPES(1:4);*),AND11000|5|(CND11000(5),SSPES(5:8);*),AND11000
|6|(CND11000(6),SSPES(9:12);*),AND11000|7|(CND11000(7),SSPES
(13:16);*);RR1(0:3,0:3));
BUS11001(AND11000|8|(DCLG(1),RR2(0:3,0);*),AND11000|9|(DCLG(
2),RR2(0:3,1);*),AND11000|10|(DCLG(3),RR2(0:3,2);*),AND11000
|11|(DCLG(4),RR2(0:3,3);*);SR2(1:4));
BUS11002(COMPTEUR|1|(,,,,,,;*,),COMPTEUR|2|(,,,,,,;*,),
COMPTEUR|3|(,,,,,,;*,),COMPTEUR|4|(,,,,,,;*,);RR2(0:3,0:3));
```

```
FILE: UNIT12   CASSPGR  P5                              CAMBRIDGE  MONITOR  SYSTE

'UNITE' EXP12000 (E0C0,E0C1;S );
          S:=- E000 . E001 ;
'UNITE' AND12000 (COND,E (0:3);S (0:3));
          'POUR' I=0 'A' 3
        'DEBUT'
            S (I):=COND . E(I);
        'FIN';
'UNITE' EXP12001 (E000,E0C1;S );
          S:= E000 . E001 ;
'UNITE' EXP12002 (E0C0(0:3),E0C1(0:3);S (0:3));
          S:= E000 + E001 ;
'UNITE' EXP12003 (E000,E001;S );
          S:= E000 .- E001 ;
'UNITE' AND12001 (COND,E (0:2);S (0:2));
          'POUR' I=0 'A' 2
        'DEBUT'
            S (I):=COND . E(I);
        'FIN';
'UNITE' BUS12000(E000(1:3),EC01(1:3),E002(1:3),E003(1:3);S (1:3));
          S:=E000+E001+E002+E0C3;
'UNITE' DECOD(ADD(1:2),CR(1:2),Q(0:1);CAD(1:3),DCLQ(1:4),CR3(1:4),CR2(
1:4));
  'SIGNAL' CR1(1:4),
  'SIGNAL' CND12003(1:4),CND12002(0:0),CND12001(0:0),CND12000(0:0);
  'EXTERNE' VAL42(1:2);(1:4));
  'EXTERNE' NON(;);
  'EXTERNE' EXP12000 (,,),EXP12001 (,;),EXP12002 ((0:3),(0:3);(0:3)),
  EXP12003 (,,);
  'EXTERNE' AND12000(,(0:3);(0:3)),AND12001(,(0:2);(0:2));
  'EXTERNE'BUS12000((1:3),(1:3),(1:3),(1:3);(1:3));
          AND12000|0|(CND12000(0),DCLQ(1:4);CR1(1:4));
          AND12000|1|(CND12001(0),DCLQ(1:4);CR2(1:4));
          EXP12002(AND12001|3|(NON|1|(;*),CR1(1:4);*),AND12000|2|(
          CND12002(0),1111;*);CR3(1:4));
          VAL42|2|(ADD(1:2);CND12003(1:4));
          VAL42|1|(Q(0:1);DCLQ(1:4));
          EXP12000(CR(1),CR(2);CND12000(0));
          EXP12001(CR(1),CR(2);CND12001(0));
          EXP12003(CR(1),CR(2);CND12002(0));
          NON|1|(CND12002(0););
          BUS12000(AND12001|0|(CND12003(1),000;*),AND12001|1|(CND12003
          (2),001;*),AND12001|2|(CND12003(3),100;*),AND12001|3|(
          CND12003(4),010;*);CAD(1:3));
```

FILE: UNIT13 CASSPGR P5 CAMBRIDGE MONITOR SYSTEM

```
'UNITE' EXP13000 (E000,E001;S );
          S:=- E000 .- E001 ;
'UNITE' EXP13001 (E000,E001;S );
          S:=- E000 . E001 ;
'UNITE' EXP13002 (E000,E001;S );
          S:= E000 .- E001 ;
'UNITE' EXP13003 (E000,E001;S );
          S:= E000 . E001 ;
'UNITE' BUS13000(E000(1:1),E001(2:2),E002(3:3),E003(4:4);S (1:4));
          S:=E000&E001&E002&E003;
'UNITE' VALN2(E(1:2),S(1:4));
  'EXTERNE' EXP13000 (,;),EXP13001 (,;),EXP13002 (,;),EXP13003 (,;);
  'EXTERNE' BUS13000((1:1),(2:2),(3:3),(4:4);(1:4));
          BUS13000(EXP13000(E(1),E(2);*),EXP13001(E(1),E(2);*),
          EXP13002(E(1),E(2);*),EXP13003(E(1),E(2);*);S(1:4));
```

```
FILE: UNIT14   CASSPGM  P5                                CAMBRIDGE  MONITOR  SYSTE

'UNITE' EXP14000 (E000,E001,E002;S );
          S:=- E000 .- E001 .- E002 ;
'UNITE' EXP14001 (E000,E001,E002;S );
          S:=- E000 .- E001 . E002 ;
'UNITE' EXP14002 (E000,E001,E002;S );
          S:=- E000 . E001 .- E002 ;
'UNITE' EXP14003 (E000,E001,E002;S );
          S:=- E000 . E001 . E002 ;
'UNITE' EXP14004 (E000,E001,E002;S );
          S:= E000 .- E001 .- E002 ;
'UNITE' EXP14005 (E000,E001,E002;S );
          S:= E000 .- E001 . E002 ;
'UNITE' EXP14006 (E000,E001,E002;S );
          S:= E000 . E001 .- E002 ;
'UNITE' EXP14007 (E000,E001,E002;S );
          S:= E000 . E001 . E002 ;
'UNITE' BUS14000(E000(1:1),E001(2:2),E002(3:3),E003(4:4),E004(5:5),
E005(6:6),E006(7:7),E007(8:8);S (1:8));
          S:=E000&E001&E002&E003&E004&E005&E006&E007;
'UNITE' VALN5(E(1:5);S(1:8));
  'EXTERNE' EXP14000 (,,;),EXP14001 (,,;),EXP14002 (,,;),EXP14003 (,,;
  ),EXP14004 (,,;),EXP14005 (,,;),EXP14006 (,,;),EXP14007 (,,;);
  'EXTERNE' BUS14000(1:1),(2:2),(3:3),(4:4),(5:5),(6:6),(7:7),(8:8);(1
  :8));
          BUS14000(EXP14000(E(1),E(2),E(3);*),EXP14001(E(1),E(2),E(3);
          *),EXP14002(E(1),E(2),E(3);*),EXP14003(E(1),E(2),E(3);*),
          EXP14004(E(1),E(2),E(3);*),EXP14005(E(1),E(2),E(3);*),
          EXP14006(E(1),E(2),E(3);*),EXP14007(E(1),E(2),E(3);*);S(1:8)
          );
```

FILE: UNIT15 CASSPGK P5 CAMBRIDGE MONITOR SYSTEM

```
'UNITE' EXP15000 (E000,E001,E002,E003;S );
          S:=- E000 .- E001 .- E002 .- E003 ;
'UNITE' EXP15001 (E000,E001,E002,E003;S );
          S:=- E000 .- E001 .- E002 . E003 ;
'UNITE' EXP15002 (E000,E001,E002,E003;S );
          S:=- E000 .- E001 . E002 .- E003 ;
'UNITE' EXP15003 (E000,E001,E002,E003;S );
          S:=- E000 .- E001 . E002 . E003 ;
'UNITE' EXP15004 (E000,E001,E002,E003;S );
          S:=- E000 . E001 .- E002 .- E003 ;
'UNITE' EXP15005 (E000,E001,E002,E003;S );
          S:=- E000 . E001 .- E002 . E003 ;
'UNITE' EXP15006 (E000,E001,E002,E003;S );
          S:=- E000 . E001 . E002 .- E003 ;
'UNITE' EXP15007 (E000,E001,E002,E003;S );
          S:=- E000 . E001 . E002 . E003 ;
'UNITE' EXP15008 (E000,E001,E002,E003;S );
          S:= E000 .- E001 .- E002 .- E003 ;
'UNITE' EXP15009 (E000,E001,E002,E003;S );
          S:= E000 .- E001 .- E002 . E003 ;
'UNITE' EXP15010 (E000,E001,E002,E003;S );
          S:= E000 .- E001 . E002 .- E003 ;
'UNITE' EXP15011 (E000,E001,E002,E003;S );
          S:= E000 .- E001 . E002 . E003 ;
'UNITE' EXP15012 (E000,E001,E002,E003;S );
          S:= E000 . E001 .- E002 .- E003 ;
'UNITE' EXP15013 (E000,E001,E002,E003;S );
          S:= E000 . E001 .- E002 . E003 ;
'UNITE' EXP15014 (E000,E001,E002,E003;S );
          S:= E000 . E001 . E002 .- E003 ;
'UNITE' EXP15015 (E000,E001,E002,E003;S );
          S:= E000 . E001 . E002 . E003 ;
'UNITE' BUS15000(E000(1:1),E001(2:2),E002(3:3),E003(4:4),E004(5:5),
E005(6:6),E006(7:7),E007(8:8),E008(9:9),E009(10:10),E010(11:11),E011(
12:12),E012(13:13),E013(14:14),E014(15:15),E015(16:16);S (1:16));
          S:=E000&E001&E002&E003&E004&E005&E006&E007&E008&E009&E010&
          E011&E012&E013&E014&E015;
'UNITE' VALN4(E(1:4);S(1:16));
  'EXTERNE' EXP15000 (,,,;),EXP15001 (,,,;),EXP15002 (,,,;),EXP15003 (
  ,,,;),EXP15004 (,,,;),EXP15005 (,,,;),EXP15006 (,,,;),EXP15007 (,,,;
  ),EXP15008 (,,,;),EXP15009 (,,,;),EXP15010 (,,,;),EXP15011 (,,,;),
  EXP15012 (,,,;),EXP15013 (,,,;),EXP15014 (,,,;),EXP15015 (,,,;);
  'EXTERNE' BUS15000((1:1),(2:2),(3:3),(4:4),(5:5),(6:6),(7:7),(8:8),(9
  :9),(10:10),(11:11),(12:12),(13:13),(14:14),(15:15),(16:16);(1:16));
          BUS15000(EXP15000(E(1),E(2),E(3),E(4);*),EXP15001(E(1),E(2),
          E(3),E(4);*),EXP15002(E(1),E(2),E(3),E(4);*),EXP15003(E(1),E
          (2),E(3),E(4);*),EXP15004(E(1),E(2),E(3),E(4);*),EXP15005(E(
          1),E(2),E(3),E(4);*),EXP15006(E(1),E(2),E(3),E(4);*),
          EXP15007(E(1),E(2),E(3),E(4);*),EXP15008(E(1),E(2),E(3),E(4)
          ;*),EXP15009(E(1),E(2),E(3),E(4);*),EXP15010(E(1),E(2),E(3),
          E(4),*),EXP15011(E(1),E(2),E(3),E(4);*),EXP15012(E(1),E(2),E
          (3),E(4),*),EXP15013(E(1),E(2),E(3),E(4);*),EXP15014(E(1),E(
          2),E(3),E(4);*),EXP15015(E(1),E(2),E(3),E(4);*);S(1:16));
```

2) Unités ML, UAL, R après contraction de certains de leurs sous réseaux en les unités BUSAND, BUSXPA, LINPUTBS, FONCTION, ARITHM.

```
FILE: UNIT2T   CASSPGK  P1                                  CAMBRIDGE  MONITOR  SYSTEM

'UNITE' AND02000 (COND,E (0:15);S (0:15));
          'POUR' I=0 'A' 15
        'DEBUT'
            S (I):=COND . E(I);
        'FIN';
'UNITE' AND02001 (COND,E (0:3);S (0:3));
          'POUR' I=0 'A' 3
        'DEBUT'
            S (I):=COND . E(I);
        'FIN';
'UNITE' REG02000 (H,ENABLE(0:3,0:15),INFO(0:3,0:15);OUTPUT(0:3,0:15));
  'HORLOGEMERE' H;
  'EXTERNE' MCELL('HORLOGE',,,);
          'POUR' I=0 'A' 3
        'DEBUT'
            'POUR' J=0 'A' 15
          'DEBUT'
              MCELL (16.I+J) (H,ENABLE(I,J),INFO(I,J);OUTPUT(I,J));
          'FIN';
        'FIN';
'UNITE' XPA02000(IN;OUT(0:3));
          'POUR' I=0 'A' 3
        'DEBUT'
            OUT(I):=IN;
        'FIN';
'UNITE' BUS02000(E000(1:4,0:0),E001(1:4,1:1),E002(1:4,2:2),E003(1:4,3:
3),E004(1:4,4:4),E005(1:4,5:5),E006(1:4,6:6),E007(1:4,7:7),E008(1:4,8:
8),E009(1:4,9:9),E010(1:4,10:10),E011(1:4,11:11),E012(1:4,12:12),E013(
1:4,13:13),E014(1:4,14:14),E015(1:4,15:15);S (1:4,0:15));
          S:=*P(*P(E000)&*P(E001)&*P(E002)&*P(E003)&*P(E004)&*P(E005)&
          *P(E006)&*P(E007)&*P(E008)&*P(E009)&*P(E010)&*P(E011)&*P(
          E012)&*P(E013)&*P(E014)&*P(E015));
'UNITE' BUS02001(E000(0:3),E001(0:3),E002(0:3),E003(0:3),E004(0:3),
E005(0:3),E006(0:3),E007(0:3),E008(0:3),E009(0:3),E010(0:3),E011(0:3),
E012(0:3),E013(0:3),E014(0:3),E015(0:3);S (0:3));
          S:=E000+E001+E002+E003+E004+E005+E006+E007+E008+E009+E010+
          E011+E012+E013+E014+E015;
'UNITE' BUS02002(E000(1:4,0:0),E001(1:4,1:1),E002(1:4,2:2),E003(1:4,3:
3),E004(1:4,4:4),E005(1:4,5:5),E006(1:4,6:6),E007(1:4,7:7),E008(1:4,8:
8),E009(1:4,9:9),E010(1:4,10:10),E011(1:4,11:11),E012(1:4,12:12),E013(
1:4,13:13),E014(1:4,14:14),E015(1:4,15:15);S (1:4,0:15));
          S:=*P(*P(E000)&*P(E001)&*P(E002)&*P(E003)&*P(E004)&*P(E005)&
          *P(E006)&*P(E007)&*P(E008)&*P(E009)&*P(E010)&*P(E011)&*P(
          E012)&*P(E013)&*P(E014)&*P(E015));
'UNITE' BUSAND20(CND02000(0:15),D(0:3);SGL02000(1:4,0:15));
'EXTERNE' AND02001(,(0:3);(0:3));
  'EXTERNE'BUS02000((1:4,0:0),(1:4,1:1),(1:4,2:2),(1:4,3:3),(1:4,4:4),
  (1:4,5:5),(1:4,6:6),(1:4,7:7),(1:4,8:8),(1:4,9:9),(1:4,10:10),(1:4,
  11:11),(1:4,12:12),(1:4,13:13),(1:4,14:14),(1:4,15:15);(1:4,0:15));
          BUS02000(AND02001|1|(CND02000(0),D(0:3);*),AND02001|2|(
          CND02000(1),D(0:3);*),AND02001|3|(CND02000(2),D(0:3);*),
          AND02001|4|(CND02000(3),D(0:3);*),AND02001|5|(CND02000(4),D(
          0:3);*),AND02001|6|(CND02000(5),D(0:3);*),AND02001|7|(
          CND02000(6),D(0:3);*),AND02001|8|(CND02000(7),D(0:3);*),
          AND02001|9|(CND02000(8),D(0:3);*),AND02001|10|(CND02000(9),D
```

```
FILE: UNIT2T   CASSPOR  P1                                CAMBRIDGE  MONITOR  SYSTE

          (0:3);*),AND02001|11|(CND02000(10),D(0:3);*),AND02001|12|(
          CND02000(11),D(0:3);*),AND02001|13|(CND02000(12),D(0:3);*),
          AND02001|14|(CND02000(13),D(0:3);*),AND02001|15|(CND02000(14
          ),D(0:3);*),AND02001|16|(CND02000(15),D(0:3);*);SGL02000(1:4
          ,0:15));
'UNITE' BUSAND21(CND02001(1:16),M(1:4,0:15);D(0:3));
'EXTERNE' AND02001(,(0:3);(0:3)),
  BUS02001((0:3),(0:3),(0:3),(0:3),(0:3),(0:3),(0:3),(0:3),(0:3),(0:3)
  ,(0:3),(0:3),(0:3),(0:3),(0:3),(0:3);(0:3));
          BUS02001(AND02001|17|(CND02001(1),M(1:4,0);*),AND02001|18|(
          CND02001(2),M(1:4,1);*),AND02001|19|(CND02001(3),M(1:4,2);*)
          ,AND02001|20|(CND02001(4),M(1:4,3);*),AND02001|21|(CND02001(
          5),M(1:4,4);*),AND02001|22|(CND02001(6),M(1:4,5);*),AND02001
          |23|(CND02001(7),M(1:4,6);*),AND02001|24|(CND02001(8),M(1:4,
          7);*),AND02001|25|(CND02001(9),M(1:4,8);*),AND02001|26|(
          CND02001(10),M(1:4,9);*),AND02001|27|(CND02001(11),M(1:4,10)
          ;*),AND02001|28|(CND02001(12),M(1:4,11);*),AND02001|29|(
          CND02001(13),M(1:4,12);*),AND02001|30|(CND02001(14),M(1:4,13
          );*),AND02001|31|(CND02001(15),M(1:4,14);*),AND02001|32|(
          CND02001(16),M(1:4,15);*);D(0:3));
'UNITE' BUSXPA22(CND02000(0:15);SGL02001(1:4,0:15));
'EXTERNE' XPA02000(,(0:3)),
  BUS02002((1:4,0:0),(1:4,1:1),(1:4,2:2),(1:4,3:3),(1:4,4:4),(1:4,
  5:5),(1:4,6:6),(1:4,7:7),(1:4,8:8),(1:4,9:9),(1:4,10:10),(1:4,11:11)
  ,(1:4,12:12),(1:4,13:13),(1:4,14:14),(1:4,15:15);(1:4,0:15));
          BUS02002(XPA02000|0|(CND02000(0);*),XPA02000|1|(CND02000(1);
          *),XPA02000|2|(CND02000(2);*),XPA02000|3|(CND02000(3);*),
          XPA02000|4|(CND02000(4);*),XPA02000|5|(CND02000(5);*),
          XPA02000|6|(CND02000(6);*),XPA02000|7|(CND02000(7);*),
          XPA02000|8|(CND02000(8);*),XPA02000|9|(CND02000(9);*),
          XPA02000|10|(CND02000(10);*),XPA02000|11|(CND02000(11);*),
          XPA02000|12|(CND02000(12);*),XPA02000|13|(CND02000(13);*),
          XPA02000|14|(CND02000(14);*),XPA02000|15|(CND02000(15);*);
          SGL02001(1:4,0:15));
'UNITE' ML(I,A(0:3),D(0:3),CS;D(0:3));
  'SIGNAL' M(1:4,0:15);
  'SIGNAL' SGL02001(1:4,0:15),SGL02000(1:4,0:15),CND02001(1:16),
  CND02000(0:15),
  'HORLOGEMERE' I;
  'EXTERNE' VALN4((1:4),(1:16));
  'EXTERNE' AND02000(,(0:15);(0:15));
  'EXTERNE' REG02000('HORLOGE',(0:3,0:15),(0:3,0:15);(0:3,0:15));
 'EXTERNE' BUSAND20((0:15),(0:3);(1:4,0:15)),
  BUSAND21((1:16),(1:4,0:15);(0:3)),
  BUSXPA22((0:15),(1:4,0:15));
          BUSAND20(CND02000(0:15),D(0:3);SGL02000(1:4,0:15));
          BUSAND21(CND02001(1:16),M(1:4,0:15);D(0:3));
          BUSXPA22(CND02000(0:15);SGL02000(1:4,0:15));
          VALN4(A(0:3),CND02001(1:16));
          AND02000|0|(CS(1),VALN4(;*);CND02000(0:15));
          REG02000|0|(I(1),SGL02001(1:4,0:15),SGL02000(1:4,0:15);M(1:4
          ,0:15)),
```

```
FILE: UNIT4T   CASSPGR  P1                                  CAMBRIDGE  MONITOR  SYSTEM

'UNITE' EXP04000 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 # E001 ;
'UNITE' EXP04001 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 # E001 ;
'UNITE' EXP04002 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 # E001 ;
'UNITE' EXP04003 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 #- E001 ;
'UNITE' EXP04004 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 # E001 ;
'UNITE' EXP04005 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 # E001 ;
'UNITE' EXP04006 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 # E001 ;
'UNITE' EXP04007 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 #- E001 ;
'UNITE' EXP04008 (E000(0:3),E001;S (0:4));
          S:= E000 & E001 ;
'UNITE' EXP04009 (E000(0:3),E001;S (0:4));
          S:= E000 & E001 ;
'UNITE' EXP04010 (E000(0:3),E001;S (0:4));
          S:= E000 & E001 ;
'UNITE' EXP04011 (E000(0:3),E001;S (0:4));
          S:= E000 & E001 ;
'UNITE' EXP04012 (E000(0:4);S (0:3));
          S:=*D|1| E000 ;
'UNITE' EXP04013 (E000,E001(0:3),E002,E003(0:3),E004(0:4),E005,E006(0:
3),E007(0:4),S (0:4));
          S:= E000 & E001 + E002 & E003 . E004 + E005 &- E006 . E007 ;
'UNITE' EXP04014 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 .- E001 ;
'UNITE' EXP04015 (E000(0:4);S (0:3));
          S:=*D|1| E000 ;
'UNITE' EXP04016 (E000,E001(0:3),E002,E003(0:3),E004(0:4),E005,E006(0:
3),E007(0:4),S (0:4));
          S:= E000 & E001 + E002 & E003 . E004 + E005 & E006 . E007 ;
'UNITE' EXP04017 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 . E001 ;
'UNITE' EXP04018 (E000(0:4);S (0:3));
          S:=*D|1| E000 ;
'UNITE' EXP04019 (E000,E001(0:3),E002,E003(0:3),E004(0:4),E005,E006(0:
3),E007(0:4);S (0:4)),
          S:= E000 & E001 + E002 & E003 . E004 + E005 &- E006 . E007 ;
'UNITE' EXP04020 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 .- E001 ;
'UNITE' EXP04021 (E000(0:4);S (0:3));
          S:=*D|1| E000 ;
'UNITE' EXP04022 (E000,E001(0:3),E002,E003(0:3),E004(0:4),E005,E006(0:
3),E007(0:4);S (0:4)),
          S:= E000 & E001 + E002 & E003 . E004 + E005 & E006 . E007 ;
'UNITE' EXP04023 (E000(0:3),E001(0:3);S (0:3));
          S:= E000 . E001 ;
'UNITE' AND04000 (COND,E (0:3);S (0:3));
          'PUSH' 1=0 'A' 3
        'DEBUT'
```

```
FILE: UNIT4T   CASSPGR  P1                                CAMBRIDGE  MONITOR  SYSTE
            S (I):=COND . E(I);
          'FIN';
'UNITE' EXP04024 (E000(0:3),E001(0:3);S (C:3));
            S:=- E000 . E001 ;
'UNITE' EXP04025 (E000(0:3),E001(0:3);S (C:3));
            S:= E000 # E001 ;
'UNITE' EXP04026 (E000(0:3),E001(0:3);S (C:3));
            S:= E000 + E001 ;
'UNITE' EXP04027 (E000(0:3);S (C:3));
            S:=- E000 ;
'UNITE' EXP04028 (E000(0:3),E001(0:3);S (C:3));
            S:= E000 + E001 ;
'UNITE' EXP04029 (E000(0:3);S (C:3));
            S:=- E000 ;
'UNITE' EXP04030 (E000(0:3),E001(0:3);S (C:3));
            S:= E000 # E001 ;
'UNITE' EXP04031 (E000(0:3);S (C:3));
            S:=- E000 ;
'UNITE' EXP04032 (E000(0:3),E001(0:3);S (C:3));
            S:= E000 +- E001 ;
'UNITE' EXP04033 (E000(0:3),E001(0:3);S (C:3));
            S:= E000 . E001 ;
'UNITE' EXP04034 (E000(0:3),E001(0:3);S (C:3));
            S:= E000 .- E001 ;
'UNITE' EXP04035 (E000(0:1),E001,E002;S (C:2));
            S:=-/+ E000 & E001 & E002 ;
'UNITE' EXP04036 (E000(0:3);S );
            S:=/+ E000 ;
'UNITE' REG04000 (H,ENABLE(0:3),INFO(0:3);OUTPUT(0:3));
  'HORLOGEMERE' H;
  'EXTERNE' MODEL('HORLOGE',,,);
            'POUR' I=0 'A' 3
          'DEBUT'
              MODEL (I) (H,ENABLE(I),INFO(I);OUTPUT(I));
          'FIN';
'UNITE' XPA04000(IN,OUT(0:3));
            'POUR' I=0 'A' 3
          'DEBUT'
              OUT(I):=IN;
          'FIN';
'UNITE' BUS04000(E000(0:3),E001(0:3),E002(0:3),E003(0:3),E004(0:3),
E005(0:3),E006(0:3),E007(0:3),E008(0:3),E009(0:3),E010(0:3),E011(0:3),
E012(0:3),E013(0:3),E014(0:3),E015(0:3);S (0:3));
            S:=E000+E001+E002+E003+E004+E005+E006+E007+E008+E009+E010+
            E011+E012+E013+E014+E015;
'UNITE' BUS04001(E000(1:1),E001(1:1),E002(1:1);S (1:1));
            S:=E000+E001+E002;
'UNITE' BUS04002(E000(0:3),E001(0:3),E002(0:3),E003(0:3),E004(0:3),
E005(0:3),E006(0:3),E007(0:3),E008(0:3),E009(0:3),E010(0:3),E011(0:3),
E012(0:3),E013(0:3),E014(0:3),E015(0:3);S (0:3));
            S:=E000+E001+E002+E003+E004+E005+E006+E007+E008+E009+E010+
            E011+E012+E013+E014+E015;
'UNITE' BUS04003(E000(1:1),E001(1:1),E002(1:1);S (1:1));
            S:=E000+E001+E002;
 'UNITE' FUNCTION(OP1(0:3),OP2(0:3),REPT;E1(0:3),E2(0:3),E3(0:3),E4(
```

```
FILE: UNIT4T   CASSPGR  P1                          CAMBRIDGE  MONITOR  SYSTEM

0:3), RS4(0:3)),
  'SIGNAL' RS7(0:4),RS8(0:3),RS1(C:4),RS2(C:4),RS3(0:4),RS5(0:3),
 RS6(0:3);
  'EXTERNE' EXP04000 ((0:3),(0:3);(0:3)),EXP04001 ((0:3),(0:3);(0:3)),
  EXP04002 ((0:3),(0:3);(C:3)),EXP04003 ((0:3),(0:3);(0:3)),EXP04004 (
  (0:3),(0:3);(0:3)),EXP04005 ((C:3),(0:3);(0:3)),EXP04006 ((0:3),(0:3
  );(0:3)),EXP04007 ((0:3),(C:3);(C:3)),EXP04008 ((C:3),;(0:4)),
  EXP04009 ((0:3),;(0:4)),EXP04010 ((0:3),;(0:4)),EXP04011 ((0:3),;(0:
  4)),EXP04012 ((0:4),(0:3)),EXP04013 (,(C:3),,(0:3),(0:4),,(0:3),(0:4
  );(0:4)),EXP04014 ((0:3),(0:3);(C:3)),EXP04015 ((0:4);(0:3)),
  EXP04016 (,(0:3),,(0:3),(0:4),,(0:3),(C:4);(0:4)),EXP04017 ((0:3),(0
  :3);(0:3)),EXP04018 ((0:4);(0:3)),EXP04019 (,(0:3),,(0:3),(0:4),,(0:
  3),(0:4);(0:4)),EXP04020 ((0:3),(C:3);(C:3)),EXP04021 ((0:4);(0:3)),
  EXP04022 (,(0:3),,(0:3),(0:4),,(C:3),(C:4);(0:4)),EXP04023 ((0:3),(0
:3);(0:3));
          EXP04000(EXP04001(OP1(C:3),OP2(C:3);*),RS1(1:4);E1(0:3));
          EXP04002(EXP04003(OP1(C:3),OP2(C:3);*),RS7(1:4);E2(0:3));
          EXP04004(EXP04005(OP1(C:3),OP2(C:3);*),RS3(1:4);E3(0:3));
          EXP04006(EXP04007(OP1(C:3),OP2(C:3);*),RS2(1:4);E4(0:3));
          EXP04008(RS4(0:3),REPT(1);RS1(0:4));
          EXP04009(RS5(0:3),1;RS2(0:4));
          EXP04010(RS6(0:3),0;RS3(0:4));
          EXP04011(RS8(0:3),REPT(1);RS7(C:4));
          EXP04012(EXP04013(0,EXP04014(OP1(0:3),OP2(0:3);*),0,OP1(0:3)
          ,RS7(0:4),0,OP2(C:3),RS7(0:4);*);RS8(0:3));
          EXP04015(EXP04016(0,EXP04017(OP1(0:3),OP2(0:3);*),0,OP1(0:3)
          ,RS1(0:4),0,OP2(0:3),RS1(0:4);*);RS4(0:3));
          EXP04018(EXP04019(0,EXP04020(OP1(0:3),OP2(0:3);*),0,OP1(0:3)
          ,RS2(0:4),0,OP2(0:3),RS2(0:4);*);RS5(0:3));
          EXP04021(EXP04022(0,EXP04023(OP1(0:3),OP2(0:3);*),0,OP1(0:3)
          ,RS3(0:4),0,OP2(C:3),RS3(0:4);*);RS6(0:3));
'UNITE'LINPUTBS(CND04000(0:15),E1(0:3),E2(0:3),E3(0:3),E4(0:3),
 OP1(0:3),OP2(0:3);SGL04000(C:3));
  'EXTERNE' EXP04024 ((0:3),(0:3);(0:3)),EXP04025 ((0:3),(0:3);(0:3)
  ),EXP04026 ((0:3),(0:3);(0:3)),EXP04027 ((0:3);(0:3)),EXP04028 ((0:3
  ),(0:3);(0:3)),EXP04029 ((0:3);(0:3)),EXP04030 ((0:3),(0:3);(0:3)),
  EXP04031 ((0:3);(0:3)),EXP04032 ((0:3),(0:3);(0:3)),EXP04033 ((0:3),
  (0:3);(0:3)),EXP04034 ((C:3),(C:3);(0:3));
  'EXTERNE' AND04000(,(0:3);(0:3));
  'EXTERNE' BUS04000((0:3),(0:3),(0:3),(0:3),(0:3),(0:3),(0:3),(0:3),(0
  :3),(0:3),(0:3),(0:3),(0:3),(0:3),(0:3),(0:3);(C:3));
          BUS04000(AND04000|0|(CND04000(C),E1(0:3);*),AND04000|1|(
          CND04000(1),E2(0:3);*),AND04000|2|(CND04000(2),E3(0:3);*),
          AND04000|3|(CND04000(3),E4(0:3);*),AND04000|4|(CND04000(4),
          EXP04024(OP1(0:3),OP2(C:3);*);*),AND04000|5|(CND04000(5),OP2
          (0:3);*),AND04000|6|(CND04000(6),EXP04025(OP1(0:3),OP2(0:3);
          *);*),AND04000|7|(CND04000(7),EXP04026(OP1(0:3),OP2(0:3);*);
          *),AND04000|8|(CND04000(8),EXP04027(EXP04028(OP1(0:3),OP2(0:
          3);*);*);*),AND04000|9|(CND04000(9),EXP04029(EXP04030(OP1(0:
          3),OP2(0:3),*);*);*),AND04000|10|(CND04000(10),EXP04031(OP2(
          0:3);*),*),AND04000|11|(CND04000(11),EXP04032(OP1(0:3),OP2(0
          :3);*);*),AND04000|12|(CND04000(12),0000;*),AND04000|13|(
          CND04000(13),EXP04033(OP1(0:3),OP2(0:3);*);*),AND04000|14|(
          CND04000(14),EXP04034(OP1(C:3),OP2(C:3);*);*),AND04000|15|(
          CND04000(15),OP1(0:3);*);SGL04000(0:3));
```

```
FILE: UNIT4T   CASSPGR  P1                                  CAMBRIDGE  MONITOR  SYSTE

 'UNITE' BUSXPA42(CND04000(0:15);SGL04001(0:3));
  'EXTERNE' XPA04000(,(0:3));
  'EXTERNE' BUS04002((0:3),(0:3),(0:3),(0:3),(0:3),(0:3),(0:3
  ),(0:3),(0:3),(0:3),(0:3),(0:3),(0:3),(0:3),(0:3);(0:3));
            BUS04002(XPA04000|0|(CND04000(0);*),XPA04000|1|(CND04000(1);
            *),XPA04000|2|(CND04000(2);*),XPA04000|3|(CND04000(3);*),
            XPA04000|4|(CND04000(4);*),XPA04000|5|(CND04000(5);*),
            XPA04000|6|(CND04000(6);*),XPA04000|7|(CND04000(7);*),
            XPA04000|8|(CND04000(8);*),XPA04000|9|(CND04000(9);*),
            XPA04000|10|(CND04000(10);*),XPA04000|11|(CND04000(11);*),
            XPA04000|12|(CND04000(12);*),XPA04000|13|(CND04000(13);*),
            XPA04000|14|(CND04000(14);*),XPA04000|15|(CND04000(15);*);
            SGL04001(0:3));
'UNITE' UAL(I1,I3,OP1(0:3),OP2(0:3),COM(0:3),SET,RAZ;S(0:3),SREPT,SZ);
  'SIGNAL' REPT,L(0:3);
 'SIGNAL' C,RS4(0:3),E1(0:3),E2(0:3),E3(0:3),E4(0:3);
  'SIGNAL' SGL04003,SGL04002,SGL04001(0:3),SGL04000(0:3),CND04001(0:2)
  ,CND04000(0:15);
  'HORLOGEMERE' I1,I3,
  'EXTERNE' VALN4((0:3);(0:15));
 'EXTERNE' EXP04035((0:1),,;(0:2)),  EXP04036 ((0:3););
  'EXTERNE' ET(,;);
  'EXTERNE' MCELL('HORLOGE',,;),REG04000('HORLOGE',(0:3),(0:3);(0:3));
  'EXTERNE' BUSXPA42((0:15);(0:3)),
            BUS04001((1:1),(1:1),(1:1);(1:1)),
  BUS04003((1:1),(1:1),(1:1);(1:1));
  'EXTERNE' FONCTION((0:3),(0:3),(1:1);(0:3),(0:3),(0:3),(0:3),(0:3)),
  LINPUTBS((0:15),(0:3),(0:3),(0:3),(0:3),(0:3),(0:3);(0:3));
            SREPT(1):=REPT(1);
            S(0:3):=L(0:3);
             FONCTION(OP1(0:3),OP2(0:3),REPT;E1(0:3),E2(0:3),E3(0:3),
             E4(0:3),RS4(0:3));
            EXP04036(L(0:3);SZ(1));
            VALN4(COM(0:3);CND04000(0:15));
            EXP04035(COM(0:1),RAZ(1),SET(1);CND04001(0:2));
            REG04000(0)((I1(1),SGL04001(0:3),SGL04000(0:3);L(0:3));
            MCELL|2|(I3(1),SGL04003(1),SGL04002(1);REPT(1));
            LINPUTBS(CND04000(0:15),E1(0:3),E2(0:3),E3(0:3),E4(0:3)
            ,OP1(0:3),OP2(0:3);SGL04000(0:3));
            BUS04001(ET|2|(CND04001(0),RS4(0);*),ET|3|(CND04001(1),0;*),
            ET|4|(CND04001(2),1;*);SGL04002(1));
            BUSXPA42(CND04000(0:15);SGL04001(0:3));
            BUS04003(CND04001(0),CND04001(1),CND04001(2);SGL04003(1));
```

```
FILE: UNIT11T  CASSPGR  P1                          CAMBRIDGE  MONITOR  SYSTEM

'UNITE' EXP11000 (E000(0:3),E001(0:3);S (0:7));
          S:= E000 & E001 ;
'UNITE' AND11000 (COND,E (0:3);S (0:3));
          'POUR' I=0 'A' 3
        'DEBUT'
           S (I):=COND . E(I);
        'FIN';
'UNITE' EXP11001 (E000(0:3),E001(0:3),E002(0:3),E003(0:3);S (0:15));
          S:= E000 & E001 & E002 & E003 ;
'UNITE' EXP11002 (E000,E001;S );
          S:= E000 + E001 ;
'UNITE' EXP11003 (E000,E001;S );
          S:= E000 . E001 ;
'UNITE' EXP11004 (E000,E001;S );
          S:= E000 . E001 ;
'UNITE' EXP11005 (E000,E001;S );
          S:= E000 + E001 ;
'UNITE' EXP11006 (E000,E001;S );
          S:= E000 . E001 ;
'UNITE' EXP11007 (E000,E001;S );
          S:= E000 . E001 ;
'UNITE' EXP11008 (E000,E001;S );
          S:= E000 + E001 ;
'UNITE' EXP11009 (E000,E001;S );
          S:= E000 . E001 ;
'UNITE' EXP11010 (E000,E001;S );
          S:= E000 . E001 ;
'UNITE' EXP11011 (E000,E001;S );
          S:= E000 + E001 ;
'UNITE' EXP11012 (E000,E001,E002,E003;S );
          S:= E000 . E001 . E002 . E003 ;
'UNITE' BUS11000(E000(0:3,0:0),E001(0:3,1:1),E002(0:3,2:2),E003(0:3,3:
3),E004(0:3,0:0),E005(0:3,1:1),E006(0:3,2:2),E007(0:3,3:3);S (0:3,0:3)
);
          'SIGNAL' S001 (0:3,0:3),S002 (0:3,0:3),S003 (0:3,0:3),S004 (
          0:3,0:3),S005 (0:3,0:3),S006 (0:3,0:3),S007 (0:3,0:3),S008 (
          0:3,0:3);
          'POUR' I=0 'A' 3
        'DEBUT'
           'POUR' J=0 'A' 3
          'DEBUT'
              'SIA'  J=0
               'ALORS' 'DEBUT'S001(I,J):=E000(I,J);'FIN'
               'SINON' 'DEBUT' S001(I,J):=0;'FIN';
              'SIA'  J=1
               'ALORS' 'DEBUT'S002(I,J):=E001(I,J);'FIN'
               'SINON' 'DEBUT' S002(I,J):=0;'FIN';
              'SIA'  J=2
               'ALORS' 'DEBUT'S003(I,J):=E002(I,J);'FIN'
               'SINON' 'DEBUT' S003(I,J):=0;'FIN';
              'SIA'  J=3
               'ALORS' 'DEBUT'S004(I,J):=E003(I,J);'FIN'
               'SINON' 'DEBUT' S004(I,J):=0;'FIN';
              'SIA'  J=0
               'ALORS' 'DEBUT'S005(I,J):=E004(I,J);'FIN'
```

```
FILE: UNIT111  CASSPOR  P1                                  CAMBRIDGE  MONITOR  SYSTE
                'SINON' 'DEBUT' SOC5(I,J):=0;'FIN';
               'SIA'  J=1
                'ALORS' 'DEBUT'SOC6(I,J):=E005(I,J);'FIN'
                'SINON' 'DEBUT' SCC6(I,J):=0;'FIN';
               'SIA'  J=2
                'ALORS' 'DEBUT'SOC7(I,J):=EOO6(I,J);'FIN'
                'SINON' 'DEBUT' SCC7(I,J):=0;'FIN';
               'SIA'  J=3
                'ALORS' 'DEBUT'SOO8(I,J):=EOO7(I,J);'FIN'
                'SINON' 'DEBUT' SOO8(I,J):=0;'FIN';
               S(I,J):=SOO1(I,J)+SOO2(I,J)+SOO3(I,J)+SOO4(I,J)+SOO5(I,J
               )+SOO6(I,J)+SOO7(I,J)+SOO8(I,J);
          'FIN';
        'FIN';
'UNITE' BUS11001(EOOO(1:4),E001(1:4),E002(1:4),E003(1:4);S (1:4));
          S:=EOOO+EOO1+EOO2+EOO3;
'UNITE' BUS11002(EOOO(0:3,0:0),EOO1(0:3,1:1),E002(0:3,2:2),E003(0:3,3:
3);S (0:3,0:3));
          S:=*P(*P(EOOO)&*P(E001)&*P(EOO2)&*P(E003));
 'UNITE' ARITHM (1,RCOM(1:3),CR2(1:4),CR3(1:4),RR1(0:3,0:3);RR2(0:3,
  0:3),TC5);
'HORLOGEMERE' 1,
 'SIGNAL' TC1,TC2,TC3,TC4;
  'EXTERNE'BUS11002((0:3,0:0),(0:3,1:1),(0:3,2:2),(0:3,3:3);(0:3,0
  :3));
  'EXTERNE'EXP11002 (,,),EXP11003 (,;),EXP11004 (,;),EXP11005 (,;
  ),EXP11006 (,;),EXP11007 (,;),EXP11008 (,;),EXP11009 (,;),EXP11010 (
  ,;),EXP11011 (,;),EXP11012 (,,,,;);
 'EXTERNE' COMPTEUR('HORLOGE',(1:4),,,,,;(1:4),);
          EXP11012(TC1(1),TC2(1),TC3(1),TC4(1);TC5(1));
          COMPTEUR(1)(1(1),RR1(0:3,0),EXP11002(CR2(1),CR3(1);*),
          EXP11003(TC2(1),RCOM(1);*),EXP11004(TC2(1),RCOM(2);*),RCOM(3
          );,TC2(1));
          COMPTEUR(2)(1(1),RR1(0:3,1),EXP11005(CR2(2),CR3(2);*),
          EXP11006(TC3(1),RCOM(1);*),EXP11007(TC3(1),RCOM(2);*),RCOM(3
          );,TC2(1));
          COMPTEUR(3)(1(1),RR1(0:3,2),EXP11008(CR2(3),CR3(3);*),
          EXP11009(TC4(1),RCOM(1);*),EXP11010(TC4(1),RCOM(2);*),RCOM(3
          );,TC3(1));
          COMPTEUR(4)(1(1),RR1(0:3,3),EXP11011(CR2(4),CR3(4);*),RCOM(1
          ),RCOM(2),RCOM(3);,TC4(1));
          BUS11002(COMPTEUR(1)(,,,,,;*,),COMPTEUR(2)(,,,,,;*,),
          COMPTEUR(3)(,,,,,;*,),COMPTEUR(4)(,,,,,;*,);RR2(0:3,0:3));
'UNITE' R(I,H,ADD(1:2),RAZ,CR(1:2),RCOM(1:3),EADD(1:2),S(1:4),SSPES(1:
16);SR1(1:16),SR2(1:4),TC,TC5);
  'SIGNAL' Q(0:1),DOLL(1:4),CAD(1:3),CR3(1:4),CR2(1:4)
  ,RR2(0:3,0:3),RR1(0:3,0:3),BL(1:16);
  'SIGNAL' CND11000(0:7);
  'HORLOGEMERE' H,1;
  'EXTERNE' COMPT('HORLOGE' ,(1:2),,,,,;(1:2),)
  ,DECOD((1:2),(1:2),(1:2);(1:3),(1:4),(1:4),(1:4));
  'EXTERNE' EXP11000 ((0:3),(0:3);(0:7)),EXP11001 ((0:3),(0:3),(0:3),(
 0:3);(0:15)),
  'EXTERNE' ARITHM('HORLOGE',(1:3),(1:4),(1:4),(0:3,0:3);(0:3,0:3),
 (1:1));
```

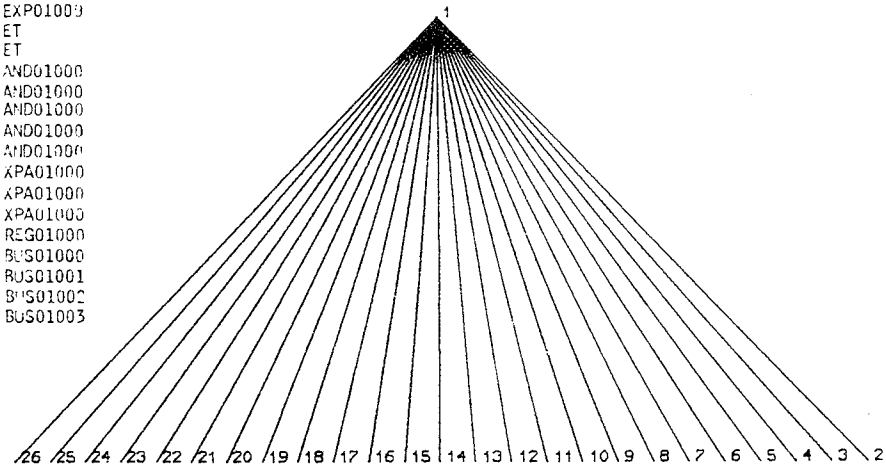
FILE: UNIT111 CASSPUR P1 CAMBRIDGE MONITOR SYSTEM

```
  'EXTERNE' AND11000(,(0:3);(0:3));
  'EXTERNE'BUS11000((0:3,0:0),(0:3,1:1),(0:3,2:2),(0:3,3:3),(0:3,0:0),
  (0:3,1:1),(0:3,2:2),(0:3,3:3);(0:3,0:3)),BUS11001((1:4),(1:4),(1:4),
 (1:4);(1:4)),
            EXP11001(RR2(0:3,0),RR2(0:3,1),RR2(0:3,2),RR2(0:3,3);SR1(1:
            16));
            COMPT(H(1),CAD0(1:2),CAD(1),CAD(2),CAD(3),RAZ(1);Q(0:1),TC(1
            ));
            DECOD(ADD(1:2),CR(1:2),Q(0:1);CAD(1:3),DOLQ(1:4),CR3(1:4),
            CR2(1:4));
            EXP11000(CR2(1:4),CR3(1:4);CND11000(0:7));
            BUS11000(AND11000|0|(CND11000(0),S(1:4);*),AND11000|1|(
            CND11000(1),S(1:4);*),AND11000|2|(CND11000(2),S(1:4);*),
            AND11000|3|(CND11000(3),S(1:4);*),AND11000|4|(CND11000(4),
            SSPES(1:4);*),AND11000|5|(CND11000(5),SSPES(5:8);*),AND11000
            |6|(CND11000(6),SSPES(9:12);*),AND11000|7|(CND11000(7),SSPES
            (13:16),*);RR1(0:3,0:3));
            BUS11001(AND11000|8|(DOLQ(1),RR2(0:3,0);*),AND11000|9|(DOLQ(
            2),RR2(0:3,1);*),AND11000|10|(DOLQ(3),RR2(0:3,2);*),AND11000
            |11|(DOLQ(4),RR2(0:3,3);*);SR2(1:4));
            ARITHM(1,RCOM(1:3),CR2(1:4),CR3(1:4),RR1(0:3,0:3);
            RR2(0:3,0:3),IC5);
```

3) Schémas des unités précédentes.

INSTITUT DE MATHEMATIQUES APPLIQUEES DE GRENOBLE

1 == COMPTEUR
2 == EXP01000
3 == EXP01001
4 == EXP01002
5 == EXP01003
6 == EXP01004
7 == EXP01005
8 == EXP01006
9 == EXP01007
10 == EXP01008
11 == EXP01009
12 == ET
13 == ET
14 == AND01000
15 == AND01000
16 == AND01000
17 == AND01000
18 == AND01000
19 == XPA01000
20 == XPA01000
21 == XPA01000
22 == REG01000
23 == BUS01000
24 == BUS01001
25 == BUS01002
26 == BUS01003



1 == EXP01000
2 == EXP01001
3 == EXP01002
4 == EXP01003
5 == EXP01004
6 == EXP01005
7 == EXP01006
8 == EXP01007
9 == EXP01008
10 == EXP01009
11 == ET|3|
12 == ET|2|
13 == AND01000|4|
14 == AND01000|3|
15 == AND01000|2|
16 == AND01000|1|
17 == AND01000|0|
18 == XPA01000|2|
19 == XPA01000|1|
20 == XPA01000|0|
21 == REG01000|0|
22 == BUS01000
23 == BUS01001
26 == BUS01003

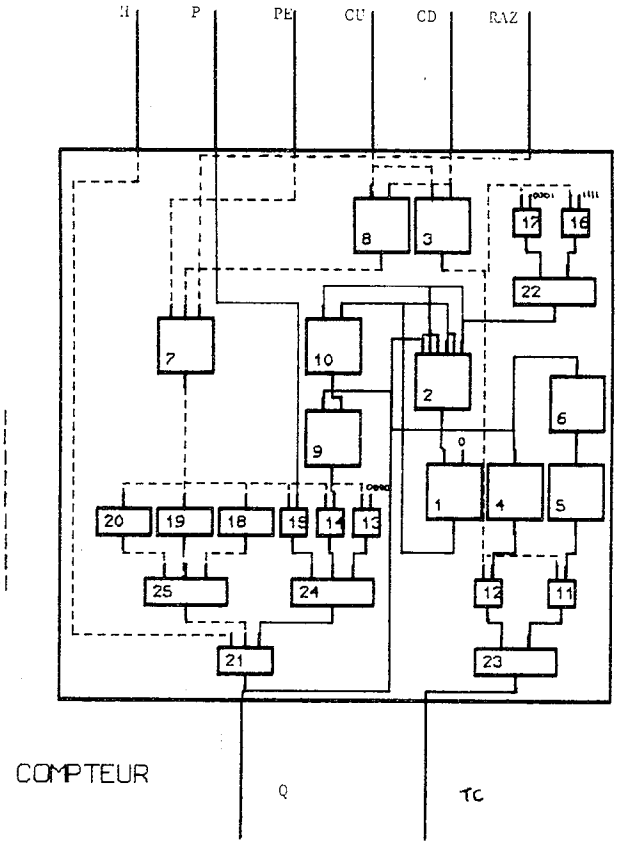
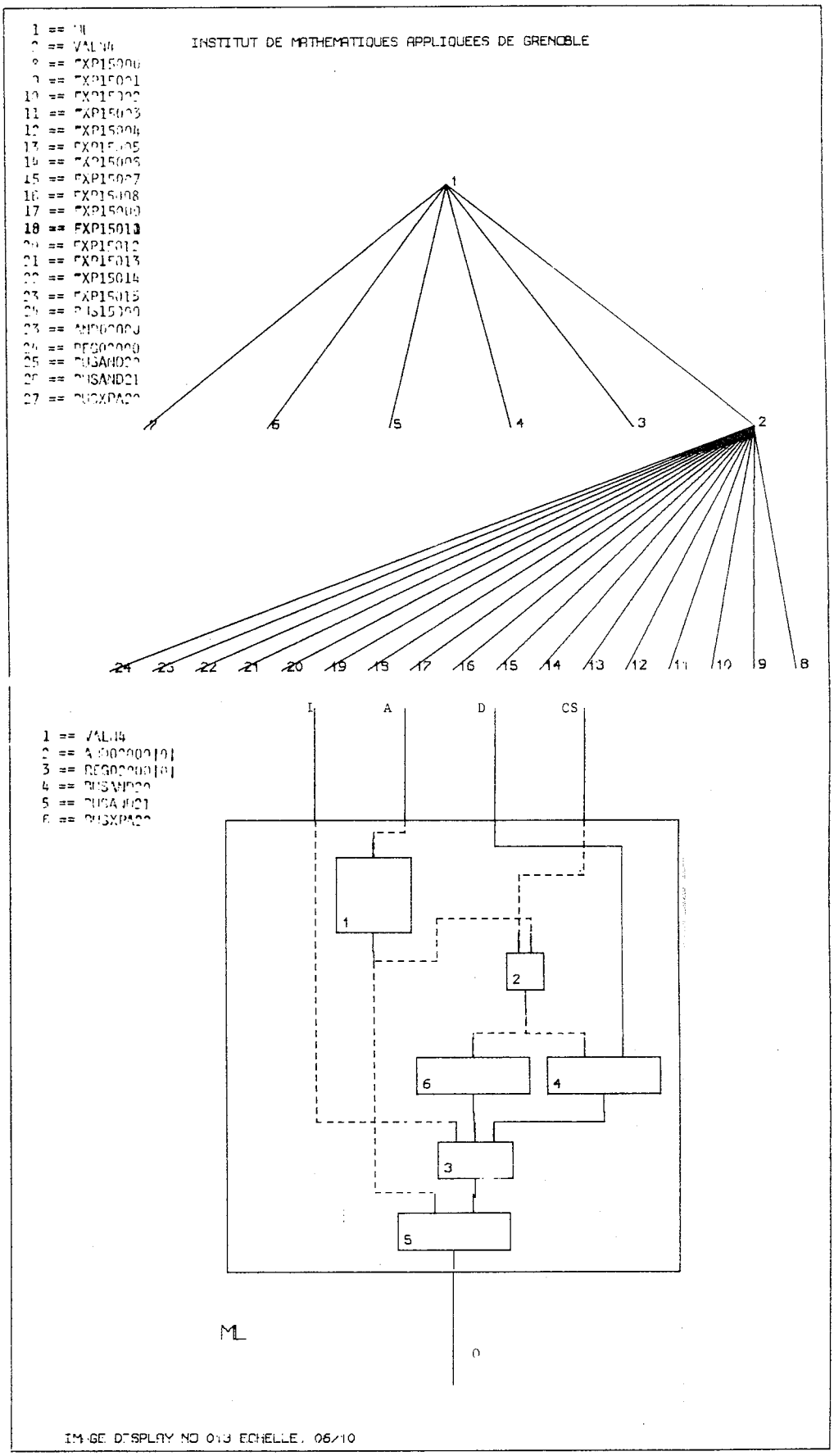
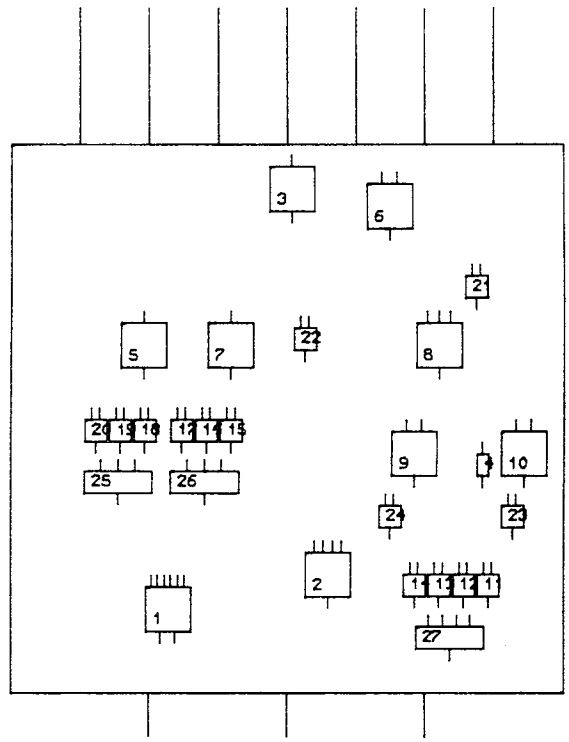
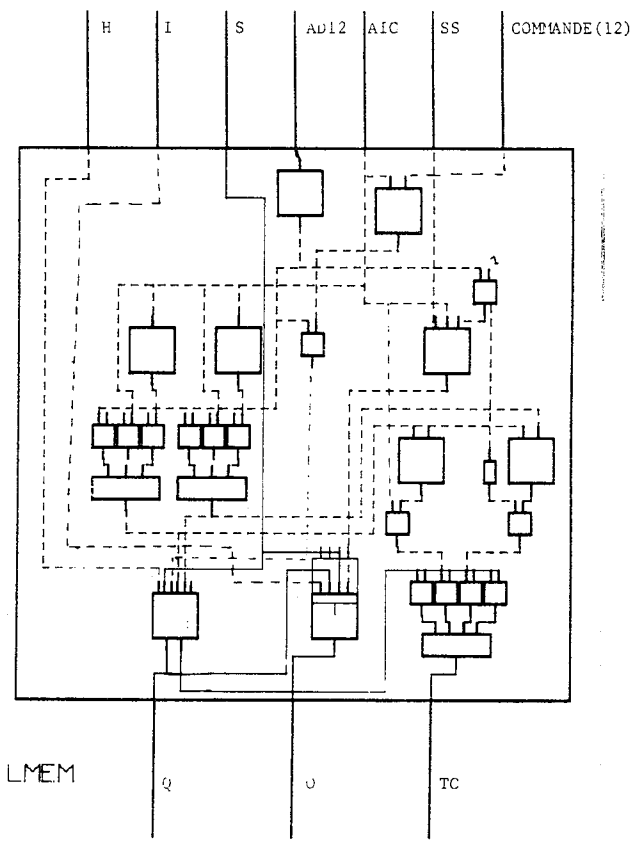


IMAGE DISPLAY NO 010 ECHELLE: 06 10


INSTITUT DE MATHEMATIQUES APPLIQUEES DE GRENOBLE
1 == M
2 == VALM
9 == EXP15000
9 == EXP15001
10 == EXP15002
11 == EXP15003
12 == EXP15004
13 == EXP15005
14 == EXP15006
15 == EXP15007
16 == EXP15008
17 == EXP15009
18 == EXP15010
20 == EXP15012
21 == EXP15013
22 == EXP15014
23 == EXP15015
24 == EXP15000
25 == AMP02000
26 == REG02000
27 == BUSAND21
28 == BUSAND21
29 == BUSXPA22
1
7
6
5
4
3
2
24
23
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8
1 == VALM
2 == AMP02000[0]
3 == REG02000[0]
4 == BUSAND22
5 == BUSAND21
6 == BUSXPA22
I
A
D
CS
1
2
6
4
3
5
ML
0
IMAGE DISPLAY NO 013 ECHELLE. 06/10

INSTITUT DE MATHEMATIQUES APPLIQUEES DE GRENOBLE

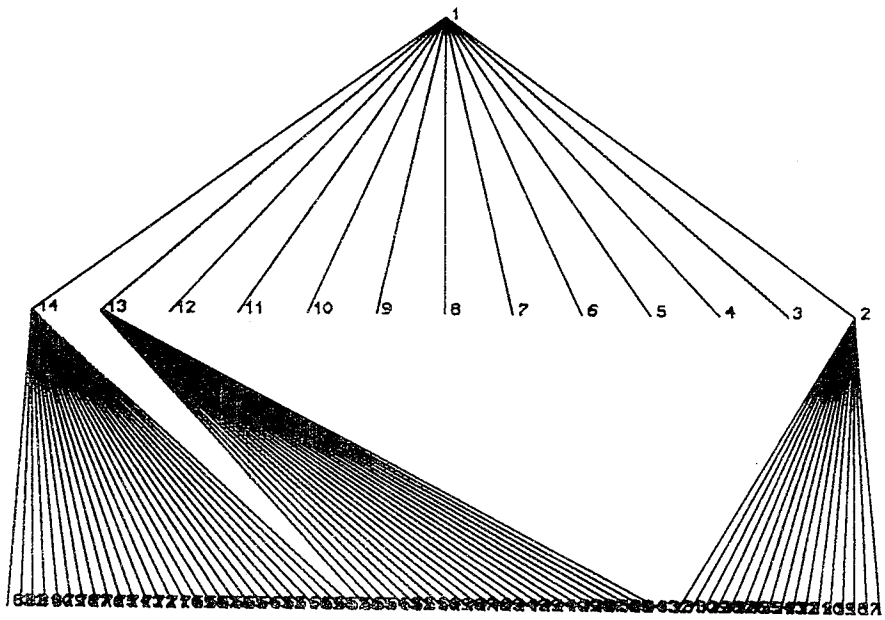




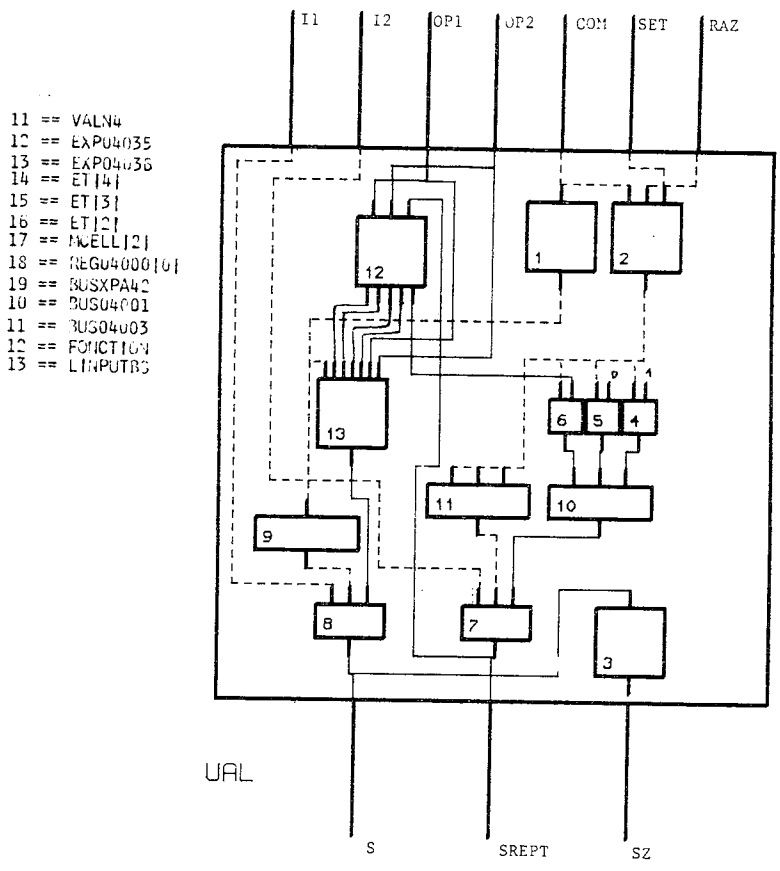
1 == COMPTEUR
2 == NIL
3 == VALN3
4 == NON|1|
5 == EXP03000
6 == EXP03001
7 == EXP03002
8 == EXP03003
9 == EXP03004
10 == EXP03005
11 == ET|13|
12 == ET|12|
13 == ET|11|
14 == ET|10|
15 == ET|9|
16 == ET|8|
17 == ET|7|
18 == ET|4|
19 == ET|3|
20 == ET|2|
21 == ET|6|
22 == ET|5|
23 == AND03000|1|
24 == AND03000|0|
25 == BUS03000
26 == BUS03001
27 == BUS03002

IMAGE DISPLAY NO 033 ECHELLE: 06/10

INSTITUT DE MATHEMATIQUES APPLIQUEES DE GRENOBLE



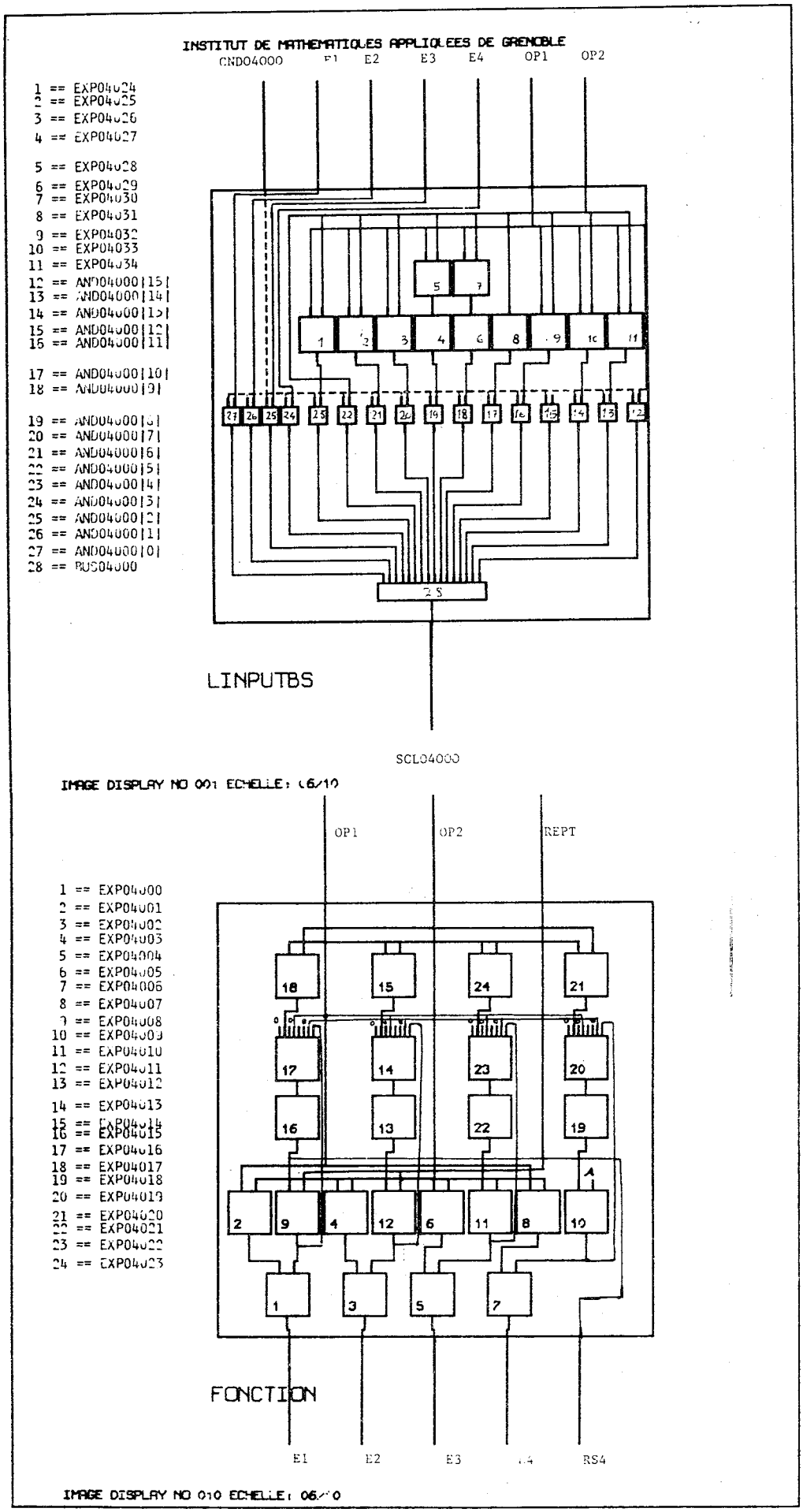
UAL

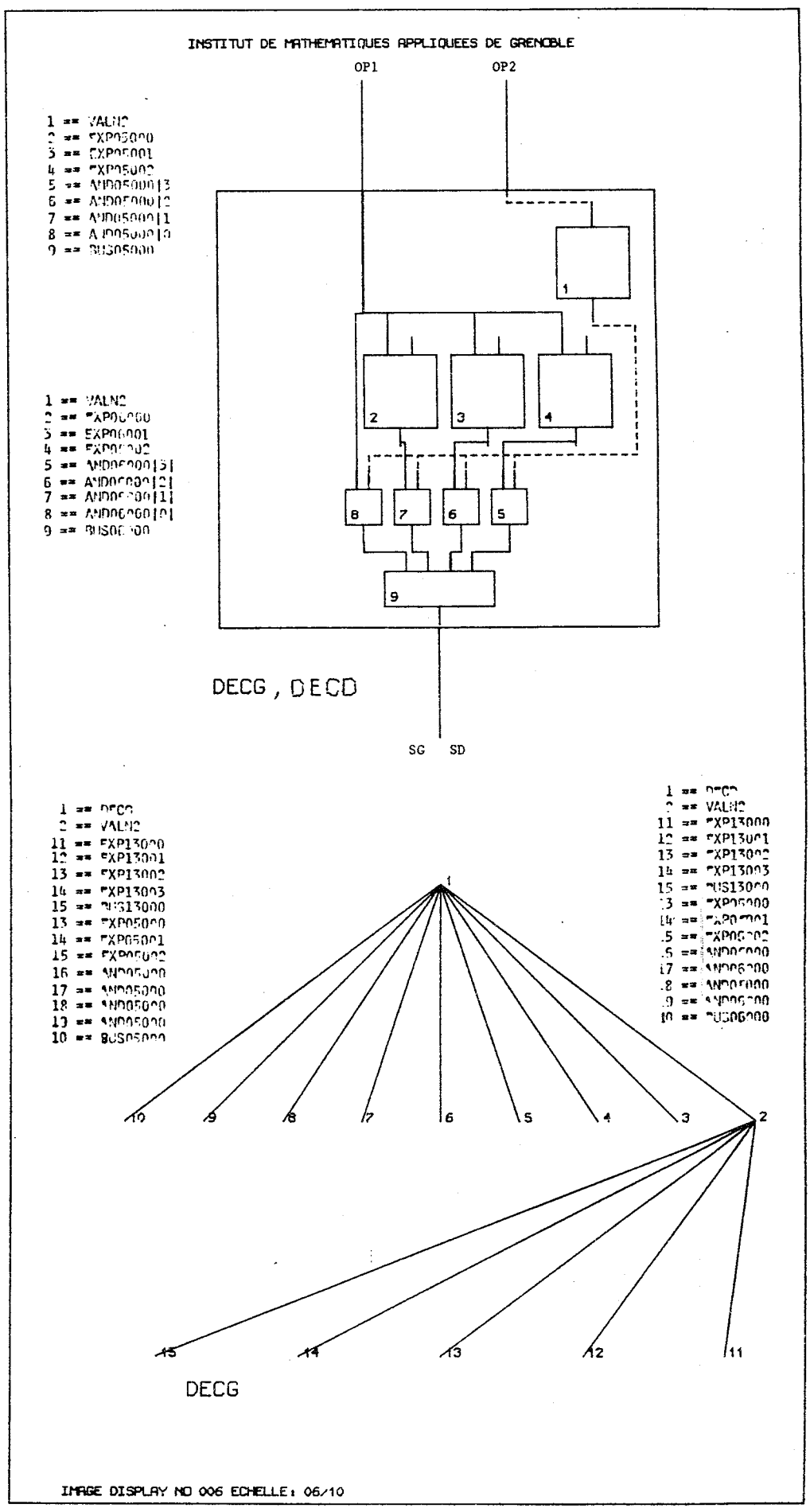


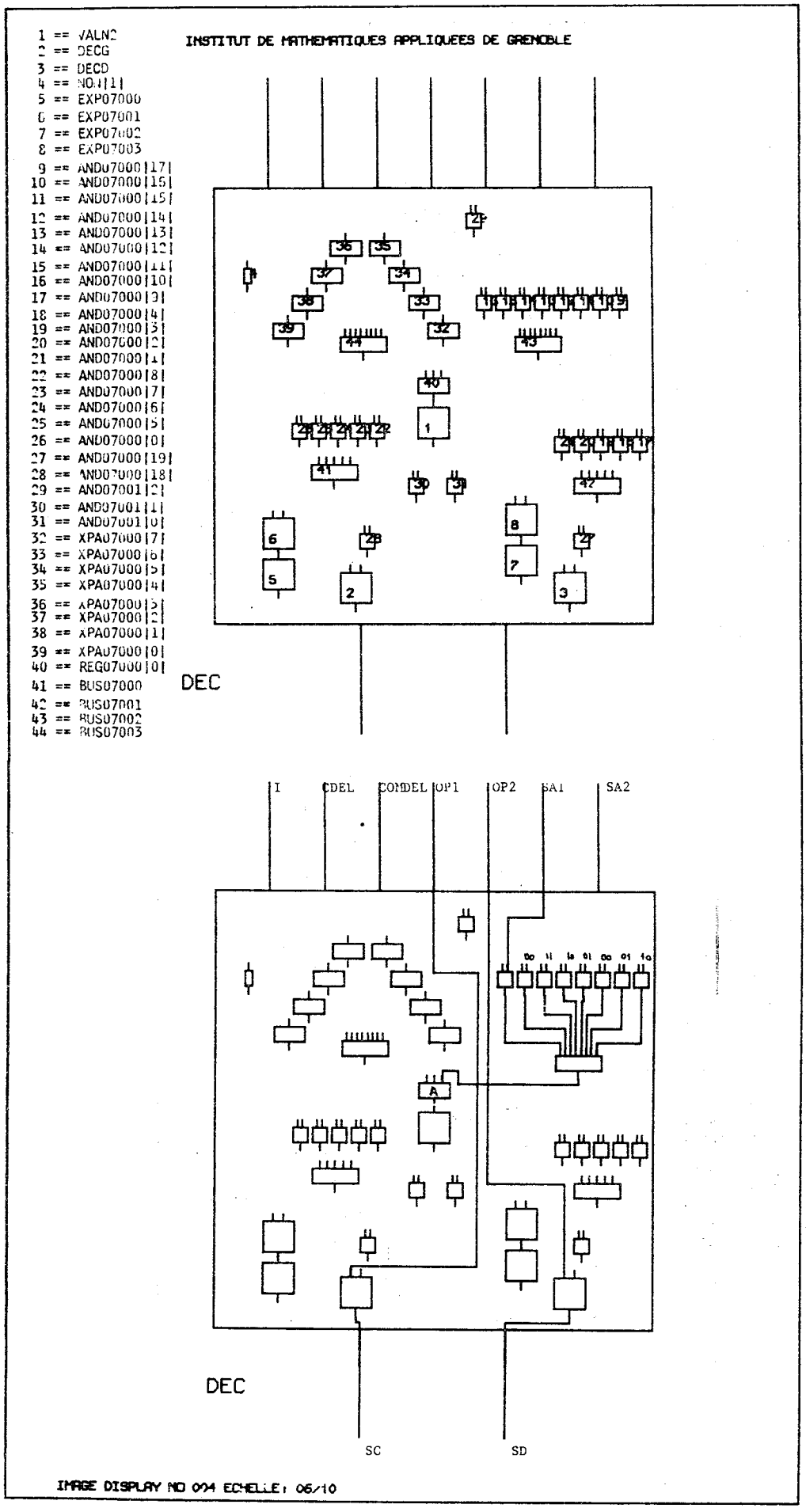
11 == VALN4
12 == EXP04035
13 == EXP04036
14 == ET[4]
15 == ET[3]
16 == ET[2]
17 == MCELL[2]
18 == REG04000[0]
19 == BUSXPA42
10 == BUS04001
11 == BUS04003
12 == FONCTION
13 == LINPUTBS

UAL

IMAGE DISPLAY NO 020 ECHELLE: 06/10

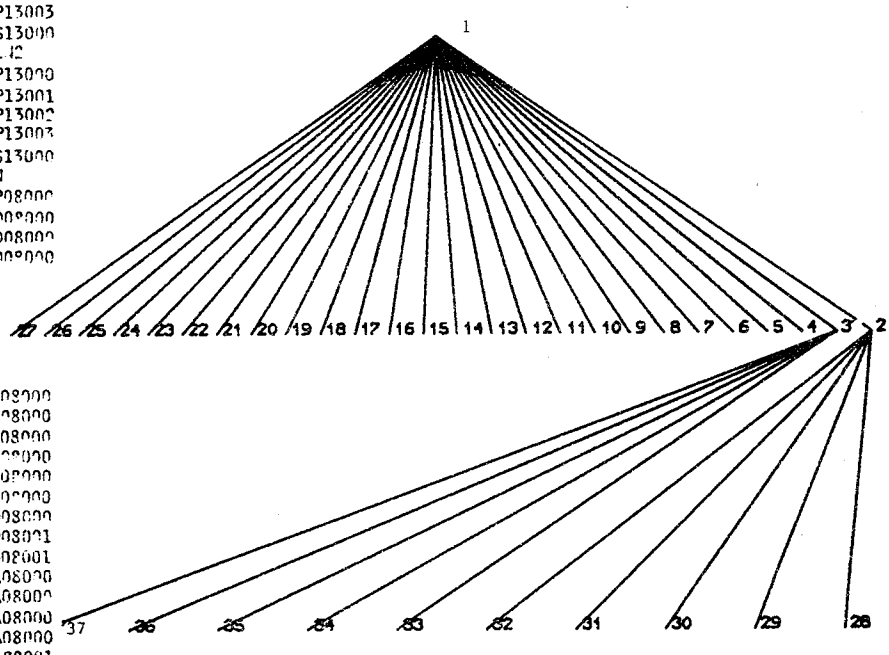

INSTITUT DE MATHEMATIQUES APPLIQUEES DE GRENOBLE
CND04000
E1
E2
E3
E4
OP1
OP2
1 == EXP04024
2 == EXP04025
3 == EXP04026
4 == EXP04027
5 == EXP04028
6 == EXP04029
7 == EXP04030
8 == EXP04031
9 == EXP04032
10 == EXP04033
11 == EXP04034
12 == AND04000|15|
13 == AND04000|14|
14 == AND04000|13|
15 == AND04000|12|
16 == AND04000|11|
17 == AND04000|10|
18 == AND04000|9|
19 == AND04000|8|
20 == AND04000|7|
21 == AND04000|6|
22 == AND04000|5|
23 == AND04000|4|
24 == AND04000|3|
25 == AND04000|2|
26 == AND04000|1|
27 == AND04000|0|
28 == BUS04000
LINPUTBS
SCL04000
IMAGE DISPLAY NO 001 ECHELLE: 06/10
OP1
OP2
REPT
1 == EXP04000
2 == EXP04001
3 == EXP04002
4 == EXP04003
5 == EXP04004
6 == EXP04005
7 == EXP04006
8 == EXP04007
9 == EXP04008
10 == EXP04009
11 == EXP04010
12 == EXP04011
13 == EXP04012
14 == EXP04013
15 == EXP04014
16 == EXP04015
17 == EXP04016
18 == EXP04017
19 == EXP04018
20 == EXP04019
21 == EXP04020
22 == EXP04021
23 == EXP04022
24 == EXP04023
FONCTION
E1
E2
E3
E4
RS4
IMAGE DISPLAY NO 010 ECHELLE: 06/10


INSTITUT DE MATHEMATIQUES APPLIQUEES DE GRENOBLE
OP1
OP2
1 == VALN2
2 == EXP05000
3 == EXP05001
4 == EXP05002
5 == AND05000|3
6 == AND05000|2
7 == AND05000|1
8 == AND05000|0
9 == BUS05000
1
2
3
4
8
7
6
5
9
DECG , DECD
SG
SD
1 == DECG
2 == VALN2
11 == EXP13000
12 == EXP13001
13 == EXP13002
14 == EXP13003
15 == BUS13000
13 == EXP05000
14 == EXP05001
15 == EXP05002
16 == AND05000
17 == AND05000
18 == AND05000
19 == AND05000
10 == BUS05000
1 == DECG
2 == VALN2
11 == EXP13000
12 == EXP13001
13 == EXP13002
14 == EXP13003
15 == BUS13000
3 == EXP05000
4 == EXP05001
5 == EXP05002
6 == AND05000
7 == AND05000
8 == AND05000
9 == AND05000
10 == BUS05000
1
10
9
8
7
6
5
4
3
2
15
14
13
12
11
DECG
IMAGE DISPLAY NO 006 ECHELLE: 06/10


1 == VALN2
2 == DECG
3 == DECD
4 == NO.7|1|
5 == EXP07000
6 == EXP07001
7 == EXP07002
8 == EXP07003
9 == AND07000|17|
10 == AND07000|15|
11 == AND07000|15|
12 == AND07000|14|
13 == AND07000|13|
14 == AND07000|12|
15 == AND07000|11|
16 == AND07000|10|
17 == AND07000|9|
18 == AND07000|4|
19 == AND07000|3|
20 == AND07000|2|
21 == AND07000|1|
22 == AND07000|8|
23 == AND07000|7|
24 == AND07000|6|
25 == AND07000|5|
26 == AND07000|0|
27 == AND07000|19|
28 == AND07000|18|
29 == AND07001|2|
30 == AND07001|1|
31 == AND07001|0|
32 == XPA07000|7|
33 == XPA07000|6|
34 == XPA07000|5|
35 == XPA07000|4|
36 == XPA07000|3|
37 == XPA07000|2|
38 == XPA07000|1|
39 == XPA07000|0|
40 == REG07000|0|
41 == BUS07000
42 == BUS07001
43 == BUS07002
44 == BUS07003
INSTITUT DE MATHEMATIQUES APPLIQUEES DE GRENOBLE
DEC
I
CDEL
COMDEL
OP1
OP2
SA1
SA2
DEC
SC
SD
IMAGE DISPLAY NO 094 ECHELLE: 06/10

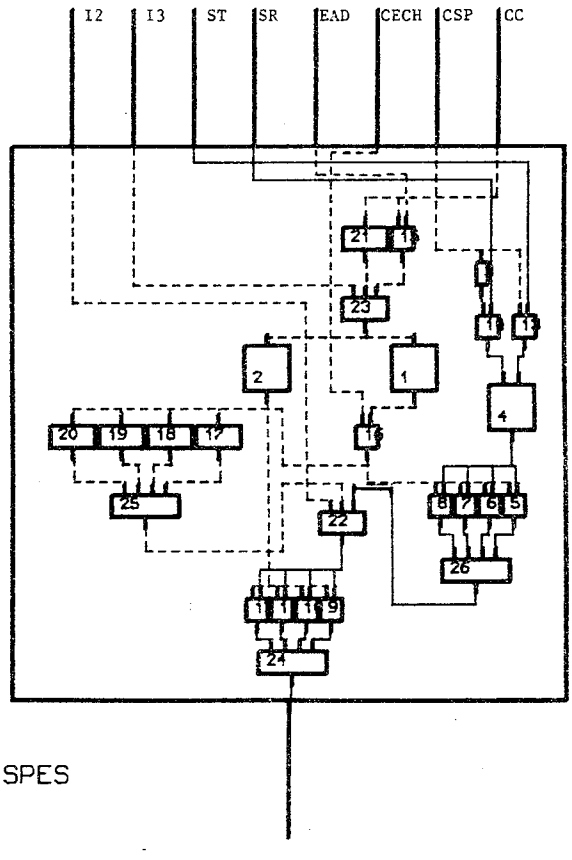
INSTITUT DE MATHEMATIQUES APPLIQUEES DE GRENOBLE

1 == SPES
2 == VALN2
28 == EXP13000
29 == EXP13001
30 == EXP13002
31 == EXP13003
32 == BUS13000
3 == VALN2
33 == EXP13000
34 == EXP13001
35 == EXP13002
36 == EXP13003
37 == BUS13000
4 == NON
5 == EXP08000
6 == AND08000
7 == AND08000
8 == AND08000
9 == AND08000
10 == AND08000
11 == AND08000
12 == AND08000
13 == AND08000
14 == AND08000
15 == AND08000
16 == AND08001
17 == AND08001
18 == XPA08000
19 == XPA08000
20 == XPA08000
21 == XPA08000
22 == XPA08001
23 == REG08000
24 == REG08001
25 == BUS08000
26 == BUS08001
27 == BUS08002



SPES

1 == VALN2|2|
2 == VALN2|1|
3 == NON|1|
4 == EXP08000
5 == AND08000|7|
6 == AND08000|6|
7 == AND08000|5|
8 == AND08000|4|
9 == AND08000|3|
10 == AND08000|2|
11 == AND08000|0|
12 == AND08000|8|
13 == AND08000|3|
14 == AND08001|1|
15 == AND08001|0|
16 == XPA08000|3|
17 == XPA08000|2|
18 == XPA08000|1|
19 == XPA08000|0|
20 == XPA08001|0|
21 == REG08000|0|
22 == REG08001|0|
23 == BUS08000
24 == BUS08001
25 == BUS08002



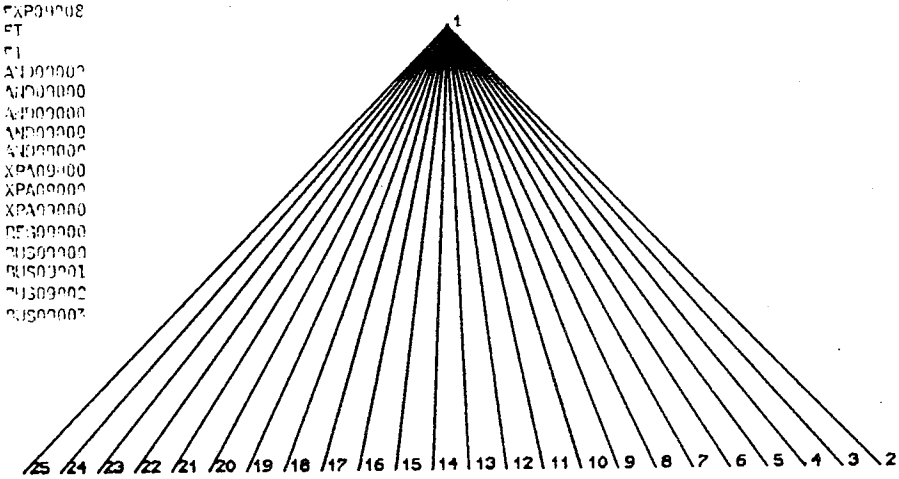
SPES

SSPES

IMAGE DISPLAY NO 017 ECHELLE: 06/10

INSTITUT DE MATHEMATIQUES APPLIQUEES DE GRENOBLE

1 == CO|PT
2 == EXP09000
3 == EXP09001
4 == EXP09002
5 == EXP09003
6 == EXP09004
7 == EXP09005
8 == EXP09006
9 == EXP09007
10 == EXP09008
11 == PT
12 == P|
13 == AND09000
14 == AND09090
15 == AND09000
16 == AND09000
17 == AND09000
18 == XPA09000
19 == XPA09000
20 == XPA09000
21 == DEC09000
22 == BUS09000
23 == BUS09001
24 == BUS09002
25 == BUS09003



*** CO|PT

1 == EXP09000
2 == EXP09001
3 == EXP09002
4 == EXP09003
5 == EXP09004
6 == EXP09005
7 == EXP09006
8 == EXP09007
9 == EXP09008
10 == PT|3|
11 == PT|2|
12 == AND09000|4|
13 == AND09000|3|
14 == AND09000|2|
15 == AND09000|1|
16 == AND09000|0|
17 == XPA09000|2|
18 == XPA09000|1|
19 == XPA09000|0|
20 == DEC09000|0|
21 == BUS09000
22 == BUS09001
23 == BUS09002
24 == BUS09003

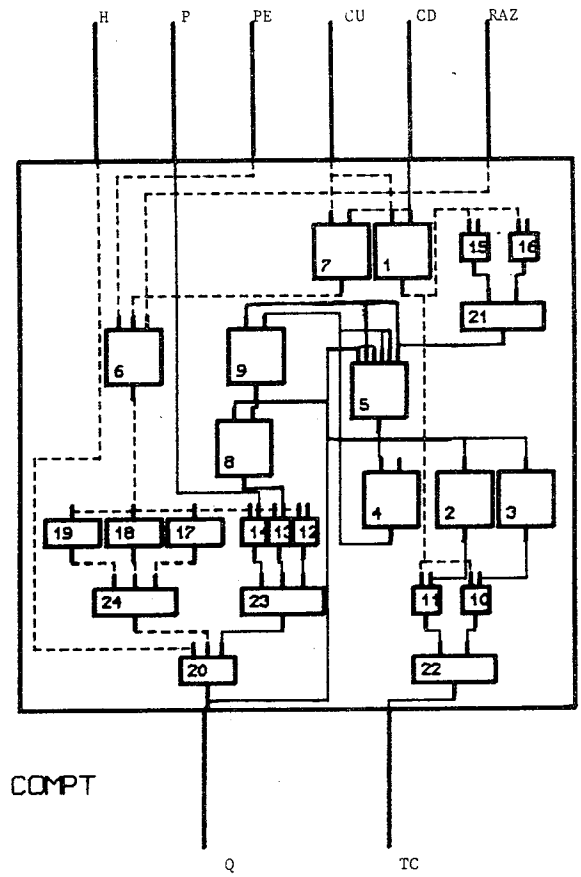
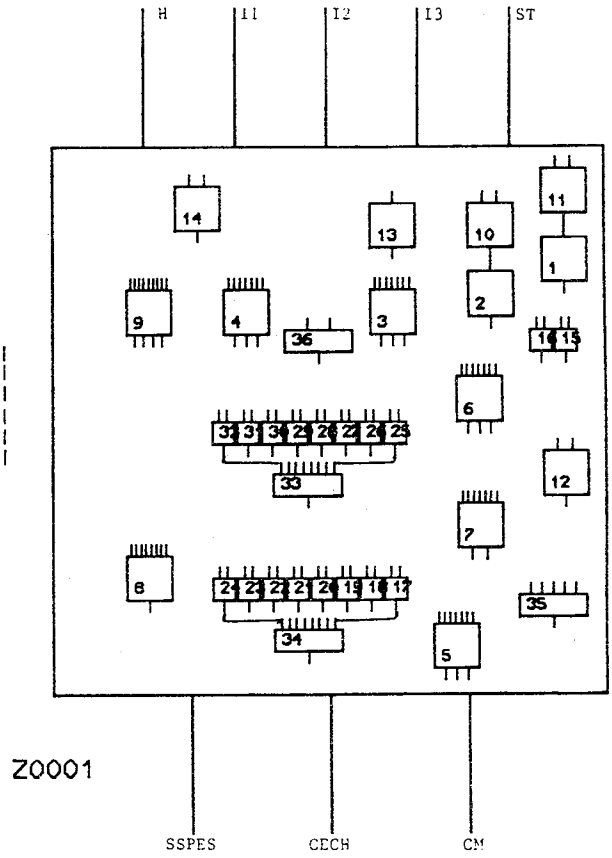


IMAGE DISPLAY NO 006 ECHELLE: 06/10

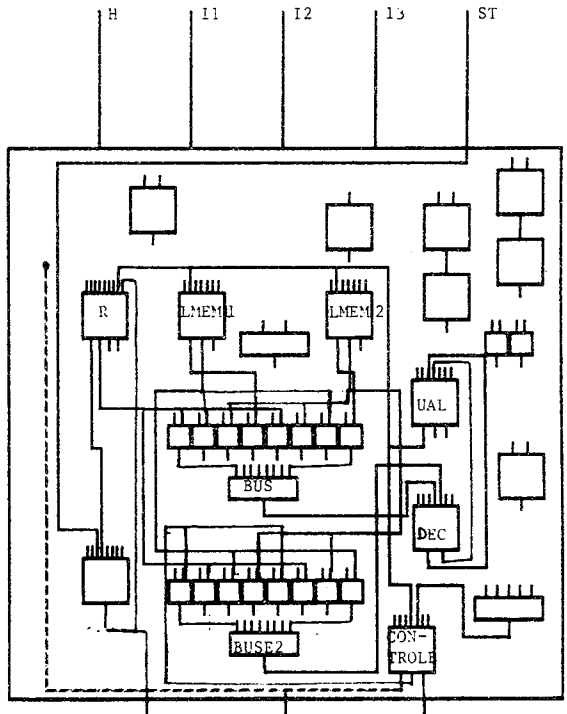
INSTITUT DE MATHEMATIQUES APPLIQUEES DE GRENOBLE

1 == VALN3|2|
2 == VALN3|1|
3 == LMEM|2|
4 == LMEM|1|
5 == CONTROLE
6 == UAL
7 == DEC
8 == SPES
9 == R
10 == EXP10000
11 == EXP10001
12 == EXP10002
13 == EXP10003
14 == EXP10004
15 == ET|3|
16 == ET|2|
17 == AND10000|15|
18 == AND10000|14|
19 == AND10000|13|
20 == AND10000|12|
21 == AND10000|11|
22 == AND10000|10|
23 == AND10000|9|
24 == AND10000|8|
25 == AND10000|7|
26 == AND10000|6|
27 == AND10000|5|
28 == AND10000|4|
29 == AND10000|3|
30 == AND10000|2|
31 == AND10000|1|
32 == AND10000|0|
33 == BUS10000
34 == BUS10001
35 == BUS10002
36 == BUS10003



Z0001

SSPES CECH CM

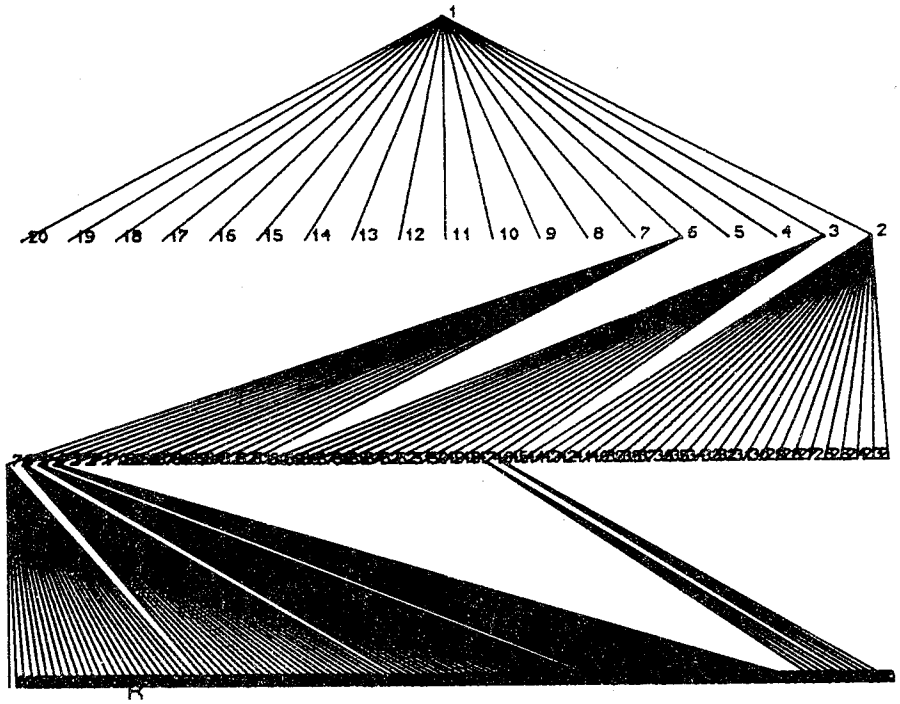


Z0001

SSPES CECH CM

IMAGE DISPLAY NO 005 ECHELLE: 06/10

INSTITUT DE MATHEMATIQUES APPLIQUEES DE GRENOBLE



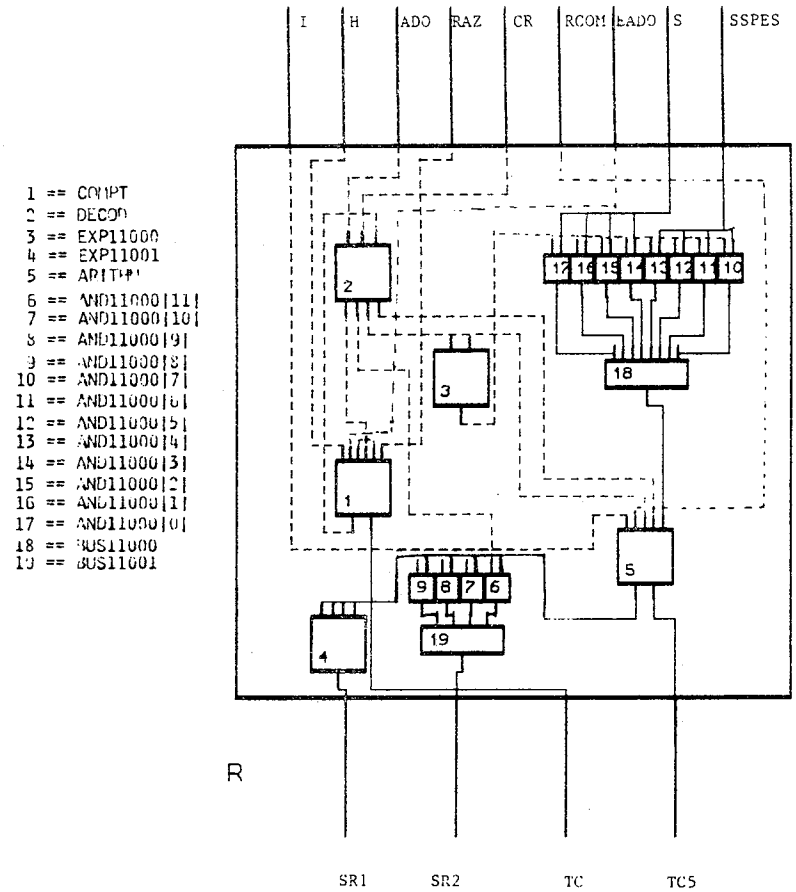
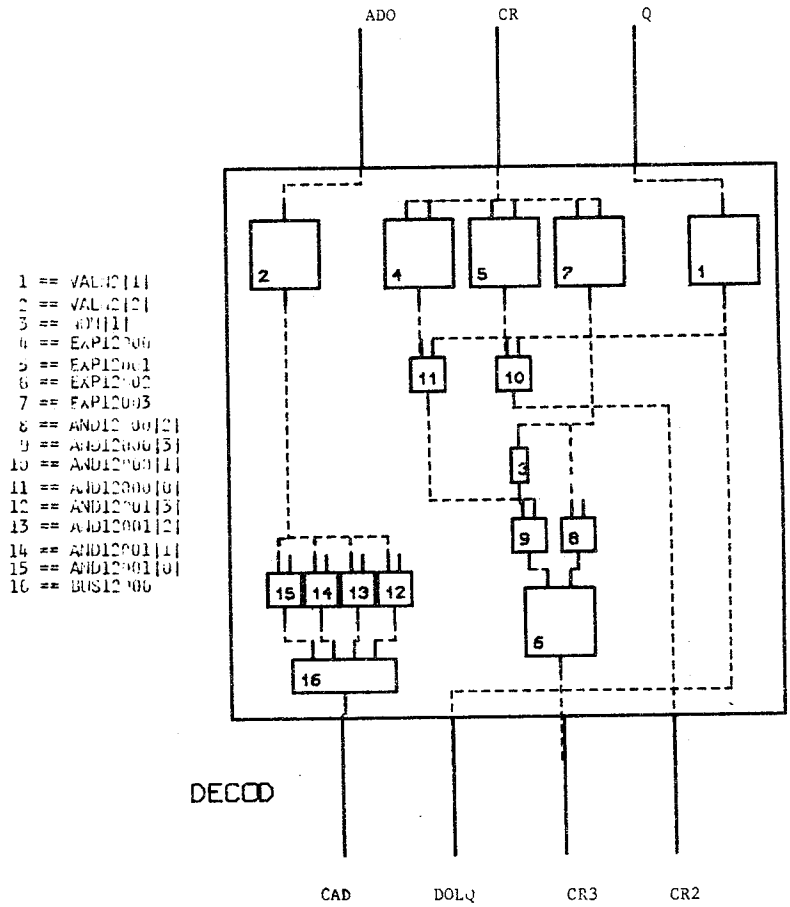


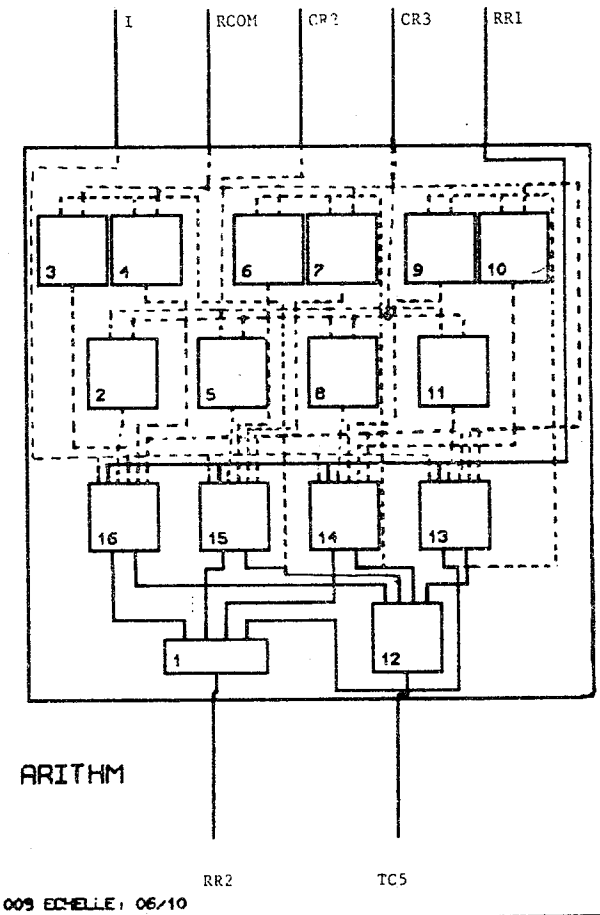
IMAGE DISPLAY NO 025 ECHELLE: 06.10

INSTITUT DE MATHEMATIQUES APPLIQUEES DE GRENOBLE



1 == VAL12[1]
2 == VAL12[2]
3 == ADI[1]
4 == EXP12000
5 == EXP12001
6 == EXP12002
7 == EXP12003
8 == AND12000[2]
9 == AND12000[3]
10 == AND12000[1]
11 == AND12000[0]
12 == AND12001[3]
13 == AND12001[2]
14 == AND12001[1]
15 == AND12001[0]
16 == BUS12000

IMAGE DISPLAY NO 002 ECHELLE: 06/10



*** ARITH

1 == BUS11000
2 == EXP11002
3 == EXP11003
4 == EXP11004
5 == EXP11005
6 == EXP11006
7 == EXP11007
8 == EXP11008
9 == EXP11009
10 == EXP11010
11 == EXP11011
12 == EXP11012
13 == COMPTEUR[4]
14 == COMPTEUR[3]
15 == COMPTEUR[2]
16 == COMPTEUR[1]

IMAGE DISPLAY NO 009 ECHELLE: 06/10

INSTITUT DE MATHEMATIQUES APPLIQUEES DE GRENOBLE

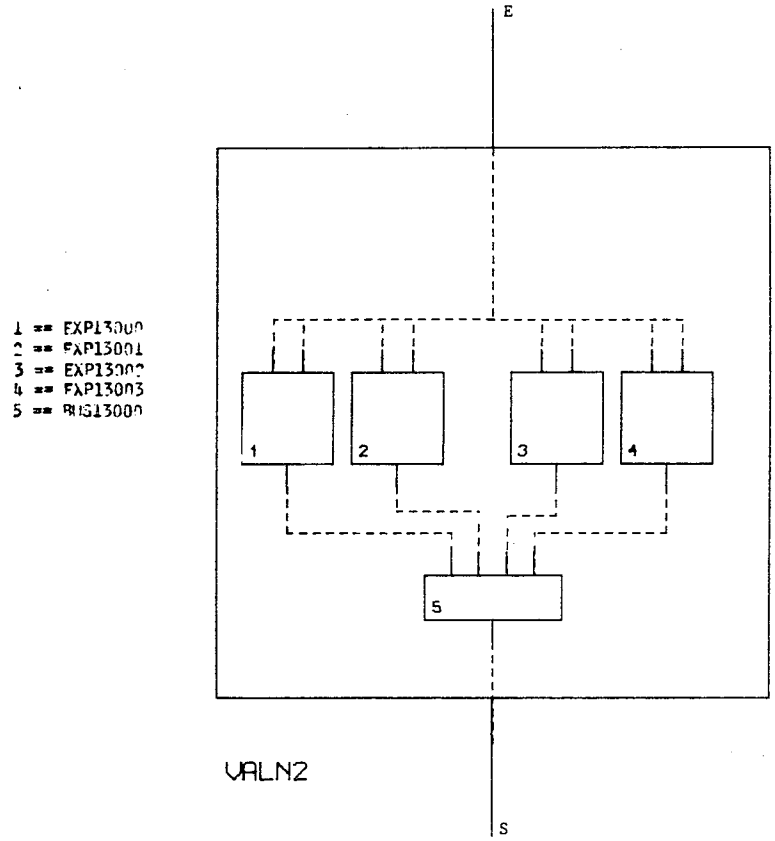


IMAGE DISPLAY NO 002 ECHELLE: 06/10

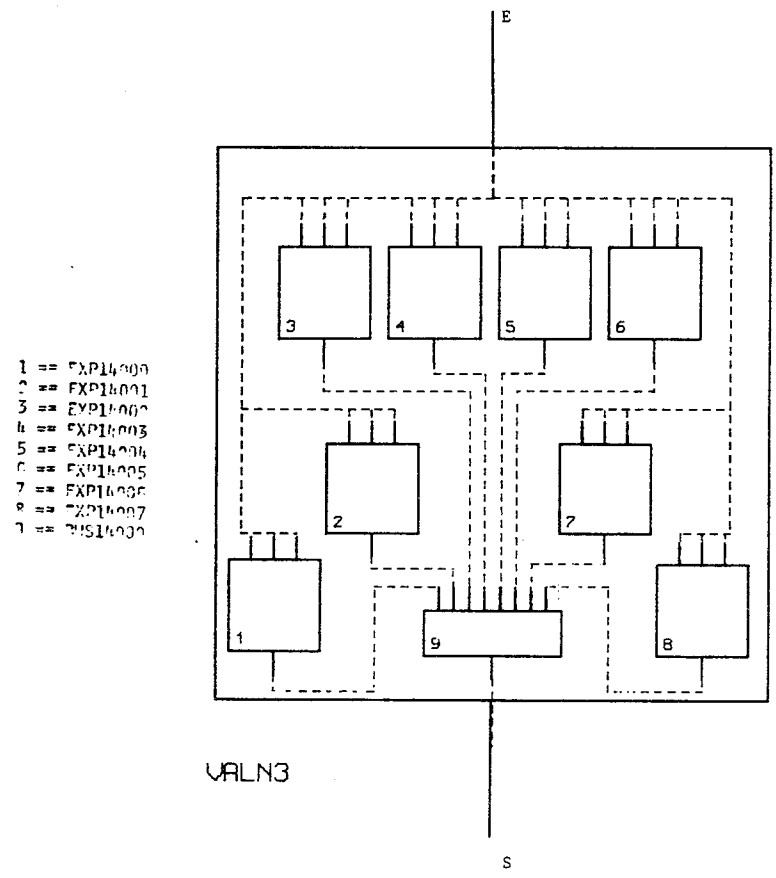
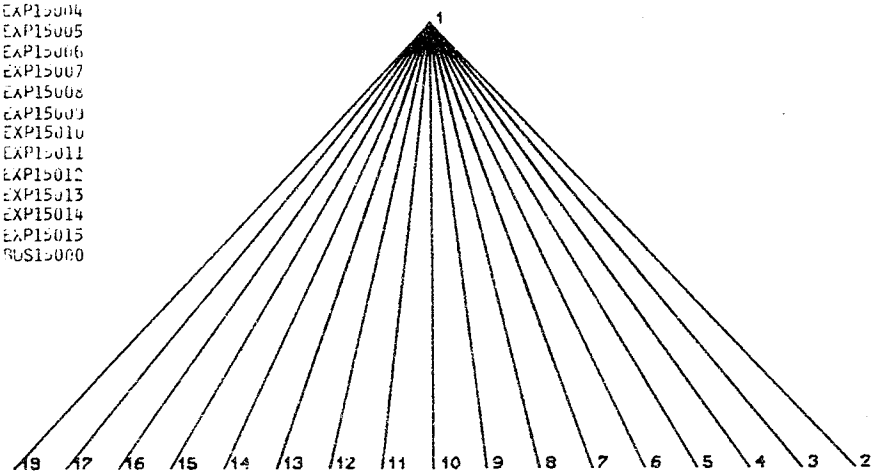


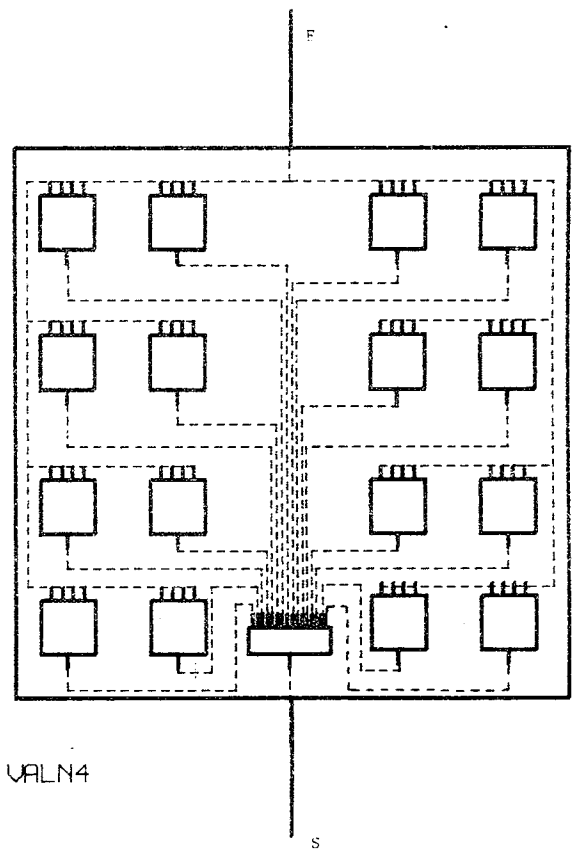
IMAGE DISPLAY NO 003 ECHELLE: 06/10

INSTITUT DE MATHEMATIQUES APPLIQUEES DE GRENOBLE

1 == VALN4
2 == EXP15000
3 == EXP15001
4 == EXP15002
5 == EXP15003
6 == EXP15004
7 == EXP15005
8 == EXP15006
9 == EXP15007
10 == EXP15008
11 == EXP15009
12 == EXP15010
13 == EXP15011
14 == EXP15012
15 == EXP15013
16 == EXP15014
17 == EXP15015
18 == BUS15000



1 == EXP15000
2 == EXP15001
3 == EXP15002
4 == EXP15003
5 == EXP15004
6 == EXP15005
7 == EXP15006
8 == EXP15007
9 == EXP15008
10 == EXP15009
11 == EXP15010
12 == EXP15011
13 == EXP15012
14 == EXP15013
15 == EXP15014
16 == EXP15015
17 == BUS15000



VALN4

IMAGE DISPLAY NO 001 ECHELLE: 06 10

1 - LIDDELL, "Découpage syntaxique de systèmes logiques décrits en CASSANDRE" Thèse 3e cycle Mars 1970.

2 - FANTINO, DAVID, MENARD, "Outils pour le découpage en module, l'implantation et la documentation automatique de machines digitales décrites en CASSANDRE", Rapport interne, ENSIMAG 1973.

3 - R. PROCTOR, "A logic design translator experiment demonstrating relations hip of language to systems and logic design", IEEE Trans. on Electronic Computers, col. EC-13, pp 422-430. August 1964.

4 - T.D. FRIEDMAN, "Alert : A program to produce logic design from preliminarv machine descriptions", RC 1578, march 24, 1966, IBM Research.

5 - G.B. GERACE, G. GESTRI, "A method for designing a digital system as a sequential network system", Université de Pise, Juin 1967.

6 - De POLIGNAC, "Utilisation du langage CAS pour la conception des machines microprogrammées", Thèse 3e cycle 1973.

7 - FARRENY, "Edition automatique de schémas logiques", Rapport final du contrat CRI 70016.

8 - F. REYNAUD-GAROCHE, "Treillis des écoulements dans un graphe orienté", Séminaire d'Algèbre Appliquée et Conception de l'IMAG, séance du 26 avril 1971.

9 - G. AURIAUX, J. MERMET, "Etude d'un macrogénérateur", projet de fin d'étude, 1966.

10 - J.C. SAILLARD, J.P. LUSINCHI, "Programme conversationnel pour l'aide à l'implantation de circuits intégrés", revue de microélectronique avancée.

D - RESEAUX EN CASSANDRE

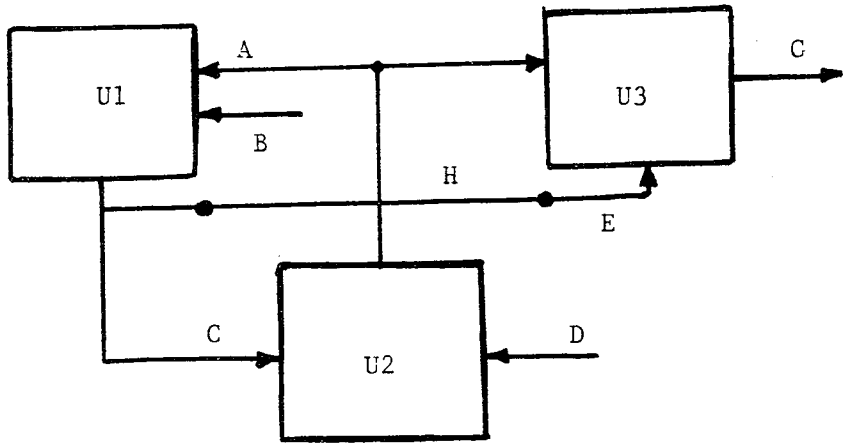
Une description en CASSANDRE Elémentaire contient des connexions d'unités des signaux-unités et des connexions de signaux, elle correspond donc à un réseau orienté d'unités et de branches. (si l'on associe à chaque signal une branche).

Exemple : U1 (A, B; C) ;

U2 (C, D; A) ;

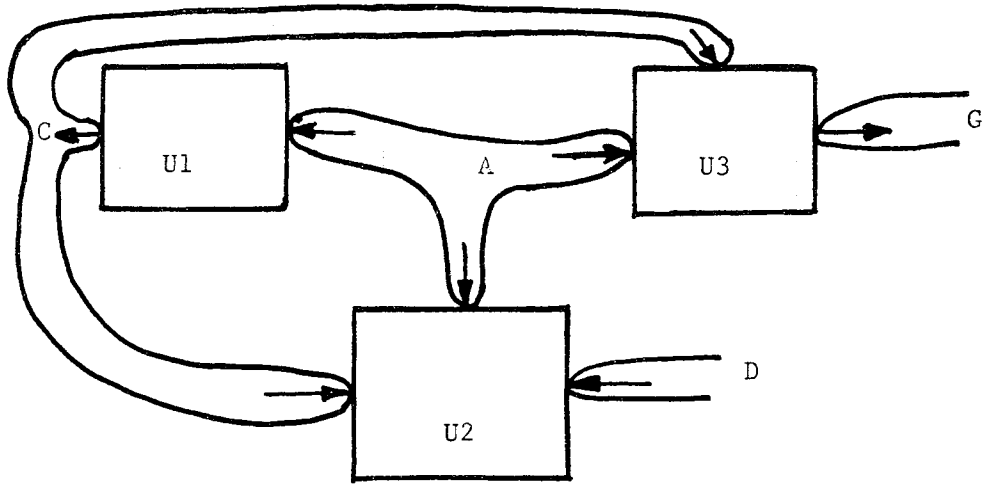
U3 (A, E; G) ;

E : = H; H = C ;



On peut réduire les branches par la relation d'équivalence qu'est la connexion de signaux. On donne à la classe d'équivalence le nom d'un de ces représentants (par exemple le plus en amont puisqu'il y a orientation). On obtient ainsi une première catégorie de réseau noeuds-étoiles (où les noeuds sont les unités et les étoiles les classes de branches).

Exemple :



Les étoiles sont aussi ce que nous avons désigné par expanseur.

Une description en CASSANDRE Général contient également un tel réseau explicite, mais aussi un réseau que nous ne nommerons implicite car c'est celui décrit en CASSANDRE.E qui est produit par le compilateur de Hardware (chapitre précédent).

En réalité c'est une structure plus générale qu'un réseau que nous obtenons. Un réseau est une structure à un seul niveau d'articulations interconnectées, or chacune de nos articulations de type "Unité" peut à son tour contenir un réseau de même nature que celui dans lequel est inséré cette unité. Nous appellerons surréseau cet emboitement hiérarchisé de réseaux d'unités.

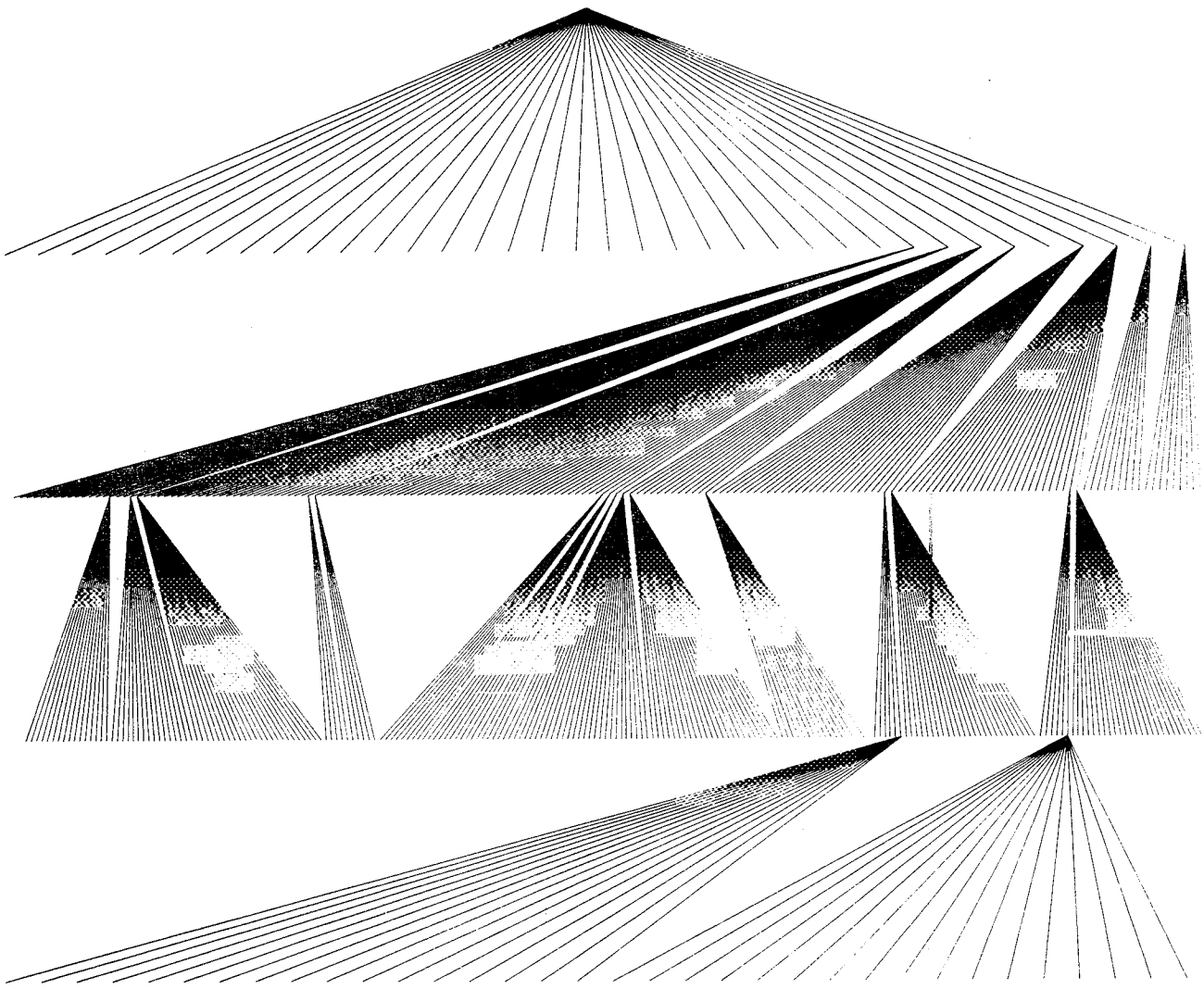
Toute unité CASSANDRE représente un surréseau implicite (celui qui est généré par sa compilation).

Exemple : le résultat de la compilation de Z0001, chapitre précédent.

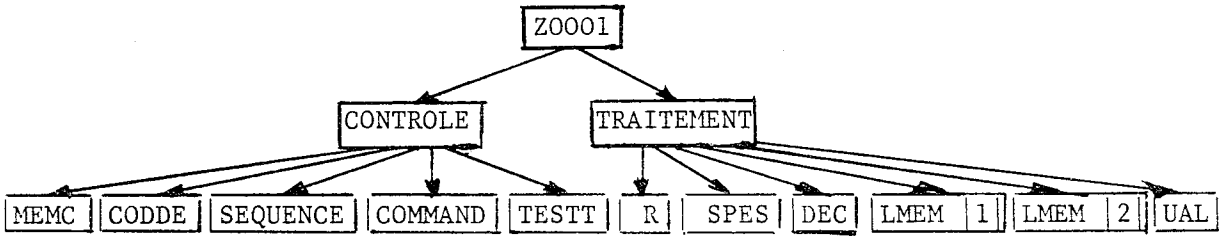
I- Sous-réseaux, surréseaux et opérations associées :

L'ensemble des articulations apparaissant à différents niveaux dans un surréseau, forme un arbre. Deux articulations ne peuvent être interconnectées que si elles ont le même antécédent dans cet arbre.

Exemple : Arbre des unités de Z0001.



Les 3 premières couches de l'arbre étant :



On définira aussi la notion de niveau dans un surréseau en disant que deux unités qui peuvent être interconnectées se trouvent au même niveau et au niveau inférieur à celui de l'unité qui les contient.

Les unités primitives se trouvent toujours au niveau le plus bas, ce sont des feuilles de l'arbre associé au surrréseau.

Le "compilateur de hardware" ou traducteur de CASSANDRE.G en CASSANDRE.E réalise une extension de l'arbre A1 associé à l'unité initiale traitée. Le résultat est un arbre A2 "compatible" avec A1 dont toutes les feuilles sont des unités primitives.

(On a A1 $\subset$ A2 = A1 et A1 $\cap$ A2 = A2).

a) Opération de contraction

Cette opération définie au chapitre précédent pour deux unités peut s'appliquer à un nombre quelconque d'unités formant un sous réseau.

Nous la noterons K (U1, U2,..., Un) où U1, U2,... Un sont les articulations du réseau d'unités contracté.

Le résultat de l'opération est une unité qui a pour entrées et sorties les connexions entre une unité U1, U2,..ou Un du sous réseau et une unité Uj qui ne lui appartient pas ou une entrée sortie de l'unité au niveau supérieur. (Pour l'ordre dans lequel apparaissent ces entrées-sorties à la périphérie de l'unité créée voir (1)).

L'opération de contraction augmente d'un niveau l'une des branches du surréseau considéré. (Les unités du sous-réseau contracté descendent d'un niveau).

Dans l'hypothèse où le sous-réseau contracté se réduit à une seule articulation nous considérerons le résultat comme isomorphe au surréseau de départ (en pratique on pourrait justifier la considération inverse : exemple d'une plaquette ne portant qu'un seul boitier de circuits intégrés). Il en résulte que le nombre de surréseaux différents pouvant être obtenus par contraction à partir d'un surréseau donné est fini.

b) Opération d'expansion

C'est l'inverse de l'opération précédente. Elle a un seul opérande, l'unité U remplacée par le réseau d'unités qu'elle contient. Nous le noterons $\mathfrak{E}$ (U) le problème des entrées-sorties ne se pose pas, pour l'unité supprimée comme pour l'unité créée par l'opération de contraction : elles restent inchangées et dans le même ordre.

L'opération $\mathfrak{E}$ ne peut pas s'appliquer à une unité primitive. On peut ramener à 2 niveaux tout surréseau par expansion de toutes ses unités non primitives (on ne peut pas le ramener à un seul niveau car il y a toujours en CASSANDRE une unité qui contient toutes les autres). Nous nommerons réseau de base d'un surréseau, le réseau au 2e niveau obtenu après expansion de toutes ses unités non primitives.

Il y a un nombre fini de surréseaux ayant le même réseau de base, ils correspondent tous au même système digital décrit en CASSANDRE et ont tous les mêmes fonctions logiques (si l'on interprète les unités primitives). Le réseau de base ainsi considéré est un réseau marqué, chaque unité primitive différente ayant un nom différent.

La réciproque n'est évidemment pas vraie : deux réseaux de base différents peuvent parfaitement correspondre aux mêmes fonctions logiques.

c) Classe de surdécoupage

Soit un réseau R1 ayant une racine U (unité racine de l'arbre associé) il correspond au système digital décrit en l'unité U. Supposons que les composants élémentaires d'une technologie aient tous comme correspondance une unité primitive. On peut dans ce cas associer un niveau dans un surréseau (c.a.d. une unité et un réseau) à chaque niveau de découpage technologique (plaquette imprimée, bacs de cartes, armoires...), on peut donc associer un surréseau R2 à la réalisation dans une technologie quelconque de l'unité U. Une condition suffisante pour que la réalisation technologique effectue les fonctions de U est que les surréseaux R1 et R2 aient le même réseau de base .

L'unité U est susceptible d'être décrite de plusieurs façons fonctionnellement équivalentes (exemple : introduction de redondance), donc d'être associée à différents réseaux de base.

Nous nommerons classes de surdécoupage les ensembles de surréseaux associés à chaque description de l'unité U.

"Tous les surréseaux d'une classe de surdécoupage peuvent être générés à partir du réseau de base par une suite d'applications de K".

Comme le réseau de base peut être obtenu à partir d'un surréseau quelconque par une suite d'applications de $\mathcal{E}$:

"Tout surréseau d'une classe peut être obtenu à partir de l'un quelconque des éléments de la classe par composition d'applications $\mathcal{E}$ et K".

Le problème du surdécoupage en sous-ensembles réalisables dans une technologie donnée d'un ensemble logique (décrit en CASSANDRE par une unité) équivaut à trouver l'application (les applications K et $\mathcal{E}$) qui fait passer du surréseau associé à U compilé à celui associé à la réalisation technologique.

d) Union, intersection, complément, disjonction de sous réseaux

Soit un réseau R d'unités (ou ce qui revient au même, une unité U qui le contient). On peut définir l'ensemble des sous réseaux de ce réseau et les opérations union, intersection, complémentation, disjonction sur cet ensemble.

Un sous réseau est défini par un ensemble A d'unités U1, U2,...Un qui sont déclarées externe dans U l'ensemble B des entrées-sorties de ces unités et de U, l'ensemble C des couples formés d'une sortie $s \in B$ et d'un signal e (soit entrée d'une unité dans U, soit sortie de U). Cet ensemble C se subdivise lui-même en deux sous-ensembles CI (connexions internes) des couples tels que $e \in B$ et CE (connexions externes) des autres. Ce sont les connexions des unités du sous réseau entre elles et avec le reste de R.

α) Union : On peut faire l'union de 2 sous-réseaux R1, R2, le résultat est un sous-réseau de R défini par les ensembles $A = A1 \cup A2$ donc $B = B1 \cup B2$, $C = C1 \cup C2$, $CI1 = CI1 \cup CI2$.

Mais on n'a pas $CE = CE1 \cup CE$ car les externes de l'intersection sont devenues des connexions internes.

L'union est commutative, associative et idempotente, seule l'idempotence pourrait poser des problèmes, on vérifie sans mal que pour $R1 \cup R1$, $A = A1$, $B = B1$, $C = C1$, $CI = CI1$, $CE = CE1$.

β) Intersection : l'intersection de 2 sous-réseaux R1 et R2 est le sous-réseau qui est défini par les ensembles : $A = A1 \cap A2$ donc $B = B1 \cap B2$ et $C = C1 \cap C2$. On a $CI = CI1 \cap CI2$, mais on n'a pas $CE = CE1 \cap CE2$.
L'intersection est commutative, associative et idempotente.

γ) Complément : Le complément d'un sous-réseau R1 de R est le sous-réseau R2 tel que $R1 \cup R2 = R$ et $R1 \cap R2 = 0$.

Ce complément existe toujours, c'est simplement le sous-réseau qui contient toutes les unités de U non contenues dans R1

δ) Disjonction : La disjonction de 2 sous-réseaux R1 $\oplus$ R2 est le sous-réseau défini par les ensembles A=(A1 $\cup$ A2) - (A1 $\cap$ A2) et donc B=(B1 $\cup$ B2) - (B1 $\cap$ B2) C = (C1 $\cup$ C2) - (C1 $\cap$ C2), mais CE = CE1 $\cup$ CE2, et CI = (CI1 $\cup$ CI2) -(CI1 $\cap$ CI2)

Cette opération est commutative, associative (à cause de l'associativité de le disjonction sur les ensembles). Elle a un élément neutre le sous-réseau vide. Chaque élément a un inverse : lui-même. On a donc un groupe abélien.

ε) Treillis associé à une unité CASSANDRE :

Les propriétés : $(R1 \cap R2) \cup R1 = R1$

$(R1 \cup R2) \cap R1 = R1$

sont évidentes, on peut le vérifier sur les ensembles A, B, C, CI et CE. Donc l'ensemble des sous-réseaux d'un réseau forme un treillis dont l'élément maximal est ce réseau et l'élément minimal le sous-réseau vide.

Cet ensemble avec la disjonction et l'intersection forme un anneau (l'intersection est distributive par rapport à la disjonction, la difficulté pourrait venir des connexions externes, mais comme il s'agit d'ensembles disjoints, ce n'est pas le cas).

ι) Contraction de ces opérations :

Les opérations K, $\cup$, $\cap$, $\oplus$ s'appliquent aux sous-réseaux d'un réseau R quelconque. K n'est pas interne car le résultat de son application est une articulation nouvelle et non un sous-réseau de R, et dans le nouveau réseau R' que l'on obtient on peut redéfinir les opérations que nous venons de voir. Si R1 est un sous-réseau de R, on voit que K est l'application K (R1, R) $\longrightarrow$ R' faisant correspondre à chaque réseau ceux qui peuvent s'en déduire par une seule contraction.

Si $\oplus$ et $\dot{\oplus}$ sont la disjonction dans R et R', R1 $\oplus$ R2 et K(R1) $\dot{\oplus}$ K(R2) existent mais on n'a pas : K(R1 $\oplus$ R2) = K(R1) $\dot{\oplus}$ K(R2) on a par contre : K(K(R1 $\oplus$ R2)) = K(R1 $\oplus$ R2) = K(K(R1) $\dot{\oplus}$ K(R2)) (en réalité K(R1,R) $\longrightarrow$ R" qui contient R2 aussi, K(R2,R3) $\longrightarrow$ R' et K(R1 $\oplus$ R2) donne R"' isomorphe à R'). On voit qu'a un isomorphisme près il est possible de définir l'application de K non à un seul sous-réseau de R mais à plusieurs sous-réseaux disjoints.

K(R1, R2,...Rj; R) est un ensemble de j articulations interconnectées qui sont les contractions de R1, R2,...Rj.

Supposons que nous ayons maintenant une somme non disjointe R1 $\cup$ R2. On ne peut pas appliquer K(R1) et K(R2) car après K(R1), R2 n'existe plus. On crée R3 = K(R1 $\cap$ R2) puis on duplique l'unité R3 en R3|1| et R3|2| chacune ayant les mêmes connexions. Il reste à la place de R1 et R2, R4 = R1 - (R1 $\cap$ R2) et R5 = R2 - (R1 $\cap$ R2).

On peut alors appliquer K(R3|1| $\cup$ R4) et K(R3|2| $\cup$ R5) on crée ainsi 2 articulations U1 et U2 et à l'intérieur de chacune d'elle on applique $\mathcal{E}$(R3).

On voit que ce processus réalise la mise en deux boites disjointes de deux sous-réseaux qui ne l'étaient pas, avec duplication de la partie commune. Cette opération est courante dans une réalisation technologique (ayant par exemple pour but de minimiser le nombre d'entrées-sorties des boites).

L'opération précédente de mise en boite de j réseaux disjoints est elle classique et employée au cours de toutes les réalisations technologiques.

C'est pourquoi les opérations K, $\mathcal{E}$, $\cup$, $\cap$, et $\oplus$, de manipulation des sous-réseaux du réseau d'une unité CASSANDRE doivent être introduites, parallèlement au compilateur de hardware, dans tout système conversationnel ou automatique de découpage et regroupement en parties technologiques. Leur usage est suffisant pour obtenir le surréseau ayant les mêmes fonctions logiques que l'unité traitée et permettant de vérifier les contraintes de la technologie choisie.

II - Hiérarchie de contrôle :

Nous avons vu dans les chapitres précédents comment une description en CASSANDRE formalisait la notion de "contrôle" dans un système digital. Nous avons vu également comment on pouvait au sein du système passer de la notion d'automate à celle de réseau logique réalisant les fonctions de cet automate.

Nous allons voir comment retrouver dans un réseau logique donné, les fonctions de contrôle, ce qui permet de poser dans un deuxième temps le problème de reconnaître les fonctions effectuées par un réseau logique lorsqu'on a identifié et isolé sa partie contrôle.

Les informations booléènnes qui circulent dans le réseau d'un système logique existant sont de deux natures différentes "contrôle" et "données". Selon la comparaison déjà faite ,l'action du contrôle peut se comparer à celle du système nerveux d'un organisme vivant : des fibres motrices viennent déclencher ou inhiber localement chaque action élémentaire, leur ensemble forme "la hiérarchie de contrôle", des fibres sensitives viennent tester localement l'état des composants du système permettant à l'automate de contrôle d'évoluer. Ces deux flux d'information sont matérialisés dans ZOOOl par les sorties CONTR, CECH et CM de l'unité CONTROLE et par son entrée TEST.

Nous supposerons qu'il est toujours possible (éventuellement après plusieurs essais) de connaître les points d'origine des fibres motrices de contrôle. Ce sont soit des éléments de mémorisation de l'automate, soit des signaux du réseau ou des entrées d'unités dont la description fait apparaître la fonction. Les informations de type données viennent toutes de l'extérieur du système, traversent le "chemin de donnée", restent le temps qu'il faut dans les éléments de mémorisation de celui-ci puis retournent à l'extérieur.

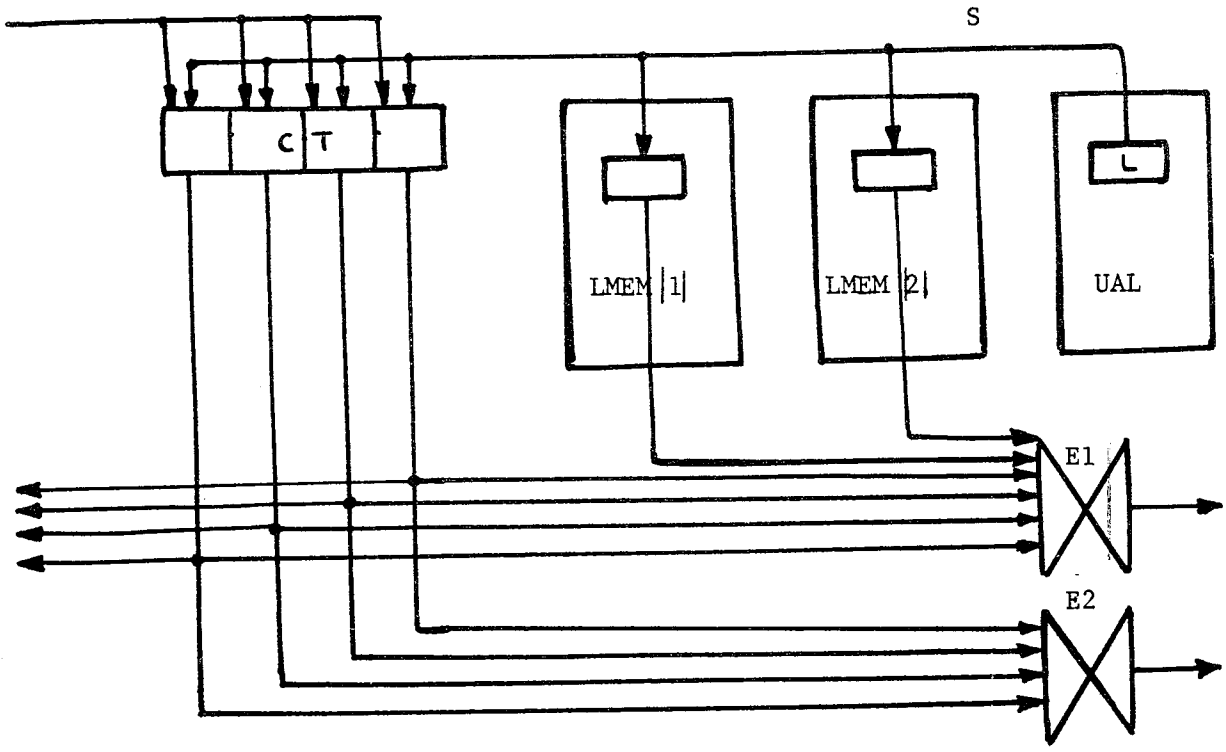
Nous rangerons dans un ensemble déclaré Contrôle les signaux et registres identifiés au départ comme tels :

Exemple : Dans ZOOOl seront ainsi déclarés les registres MEMM (de MEMC) et son signal de sortie MICRO(1:32), le registre A (1 : 2) (de DEC), les signaux AIG, CECH(1:3), CONTR(1:43), CM(1:3) de sortie à la fois des unités COMMAND et CONTROLE, et les registres RAIG et MICROCOM(1:54) de COMMAND, le signal d'entrée A (de ML) les signaux PE, CU, CD et RAZ d'entrée de COMPTEUR et le registre AD de COMPT.

Bien entendu nous oublions de nombreuses variables de contrôle dans cette liste, mais cela est délibéré car nous voulons définir un processus qui permette de les retrouver toutes à partir d'un ensemble de départ suffisant.

REMARQUES :

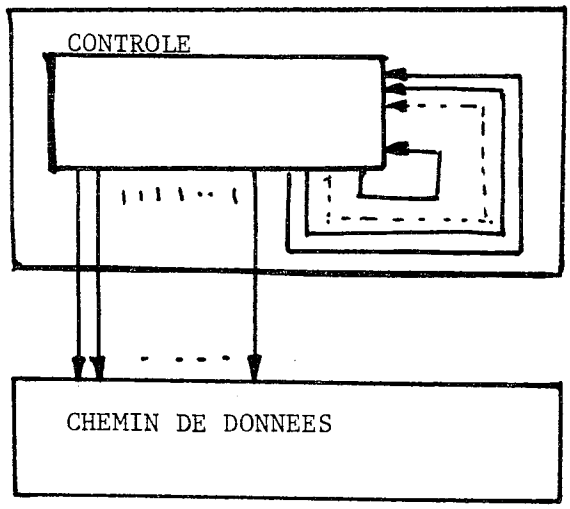
1) Nous n'avons pas déclaré contrôle le registre CT des unités COMPTEUR, bien que manifestement son contenu soit interprété par l'unité ML (mémoire locale) comme tel. D'ailleurs le registre ADO de COMPT qui joue le même rôle dans R est déclaré contrôle. Mais le contenu de CT(dans les 6 unités de type COMPTEUR de R, LMEM |1| et LMEM |2|)provient du chemin de donnée par la sortie S de UAL et peut retourner au chemin de donnée par les bus E1 et E2.



On résoud cette apparente contradiction dûe au rôle mixte des registres CT en déclarant contrôle les entrées A de ML, PE, CU, CD de COMPTEUR qui proviennent bien, à travers certains circuits combinatoires des registres CT mais dont tous les signaux qui se trouvent en aval sont exclus du chemin de donnée.

A, PE, CU, CD, deviendront dans ce cas, des racines de la hiérarchie de contrôle.

2) La hiérarchie de contrôle se subdivise en deux nappes de fibres motrices, l'une qui agit sur l'automate de contrôle lui-même, l'autre qui agit sur le chemin de donnée. Dans la suite nous ne nous intéresserons qu'à cette dernière, c'est pourquoi, nous ne considérerons plus MEMM (mémoire morte) ni MICRO(1:32) (sa sortie) ni RAIG, ni MICROCOM, comme racines de la hiérarchie, mais seulement AIG, CECH, CONTR et CM qui se trouvent en aval et à l'origine de la nappe de contrôle du chemin de données.



Le processus de construction pourrait s'appliquer aussi bien aux deux nappes.

3) Un registre déclaré contrôle peut être chargé par une donnée (exemple registre A de DEC chargé par 5). C'est une articulation pendante du graphe du chemin de données car son contenu n'y retourne jamais.

Nous avons donc comme classe de départ :
AIG, CECH(1:3), CONTR(1:43), CM(1:3), A(1:2) (de DEC), CU, CD, PE (des unités COMPTEUR |1|, COMPTEUR |2|, COMPTEUR |3| de R, car les 3 autres unités de type COMPTEUR ont nous le verrons sur ces entrées des signaux de l'arborescence de contrôle), A (1:4) (entrée de ML dans LMEM |1| et LMEM |2|), ADO (1:2)

Le processus s'applique au réseau de base d'une unité après action du compilateur de hardware (exemple Z0001 chapitre précédent). On suit ce réseau orienté en partant des racines déclarées contrôle et on augmente progressivement la classe des signaux de contrôle.

a) Propagation à travers les unités primitives :

1) Chaque fois qu'un signal de contrôle se trouve en entrée d'une unité de type OU, XPA, NON, on range la sortie de cette unité dans la classe des signaux de contrôle, si la description de l'unité au départ était cohérente les autres entrées des unités primitives ci-dessus seront aussi dans la classe de contrôle, mais on n'a pas besoin d'attendre de les avoir trouvé pour continuer le processus à partir des sorties.

2) Chaque fois qu'une entrée DONNEE d'une unité de type BUS 1, BUS 2, ET 1 ou NI 1 se trouve dans la classe de contrôle, on y range la sortie. Si la description de départ était cohérente, les entrées CONDITION des unités BUS 1, BUS 2, ET 1 ou NI 1 sont toujours dans la classe de contrôle. Si ce n'est pas le cas en fin de traitement il y a une erreur à corriger.

3) Pour les mêmes raisons si une entrée DONNEE d'une unité MCELL est dans la classe de contrôle, on y met le registre contenu dans MCELL et la sortie.

Exemple : Le registre AD3 dans l'unité SPES après compilation (voir schéma).

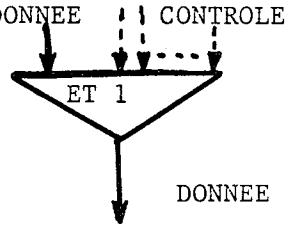
4) Si toutes les entrées d'une unité de type ET2 ou NI2 sont dans la classe de contrôle, leur sortie s'y trouve aussi.

5) Ce qui a été dit pour les unités ET 1, et MCELL se généralise aux unités non primitives AND et REG.

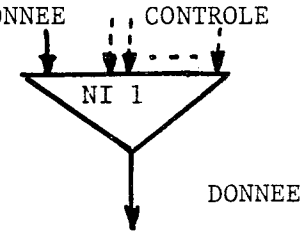
Exemple : L'unité AND08000 (n° 6 sur le dessin) dans SPES surdécoupé.

6) Ce qui a été dit pour NI 2 et ET 2 se généralise aux unités EXP, si toutes leurs entrées sont de type contrôle, leur sortie aussi.
Chaque fois qu'un signal de la classe de contrôle se trouve en entrée d'une unité et qu'on ne se trouve pas dans l'un des 6 cas précédents, on a :

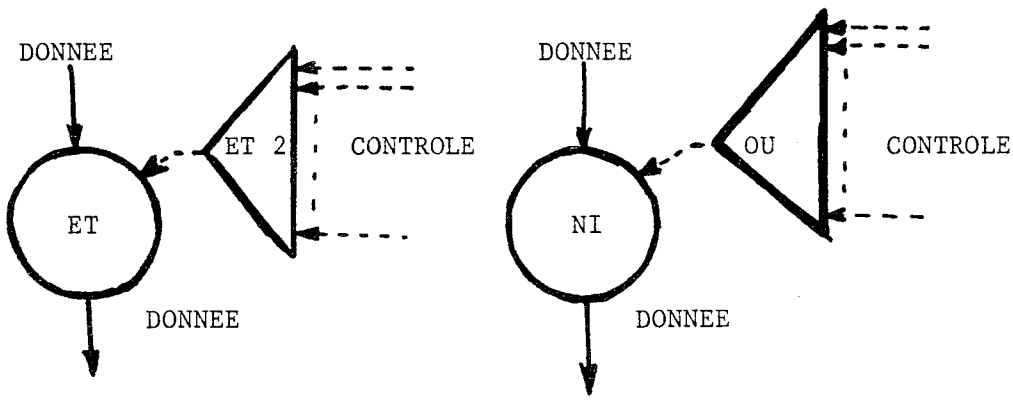
soit 7)



ou

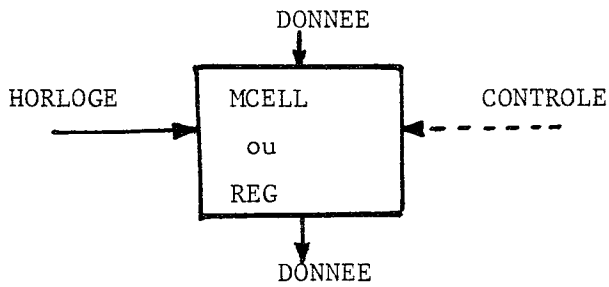


Que l'on remplace par les réseaux :



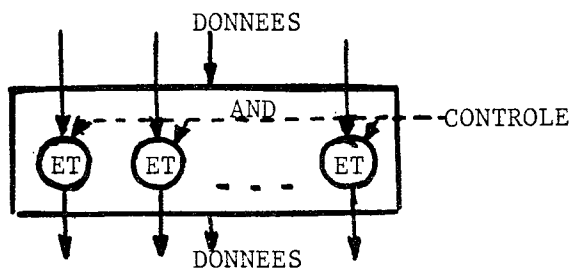
On a trouvé une <u>porte</u> (l'équivalent des synapses des fibres nerveuses). C'est une des feuilles de la hiérarchie de contrôle :

Soit 8)



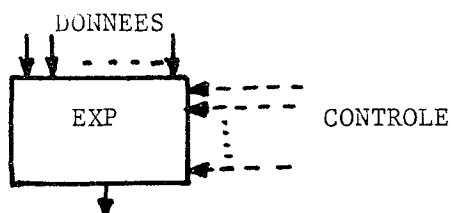
C'est l'un des éléments de mémorisation du chemin de donnée :

Soit 9)



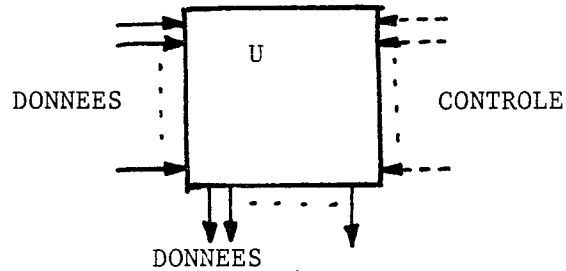
On a trouvé un ensemble de portes :

Soit 10)

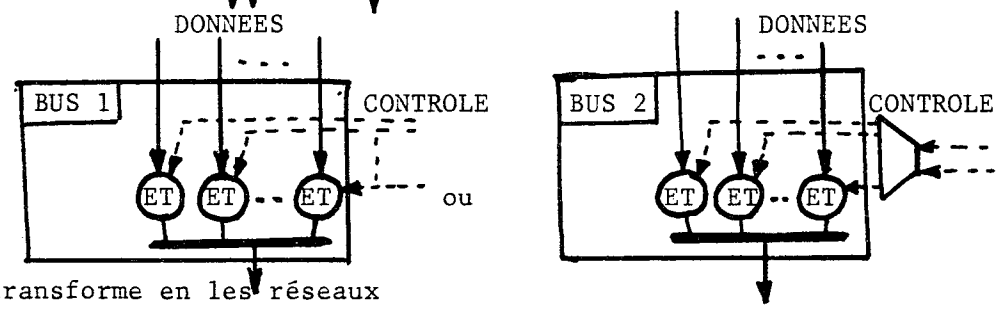


Dans ce cas on peut affirmer que l'unité EXP contient des portes mais il faut en faire l'expansion pour les mettre en évidence :

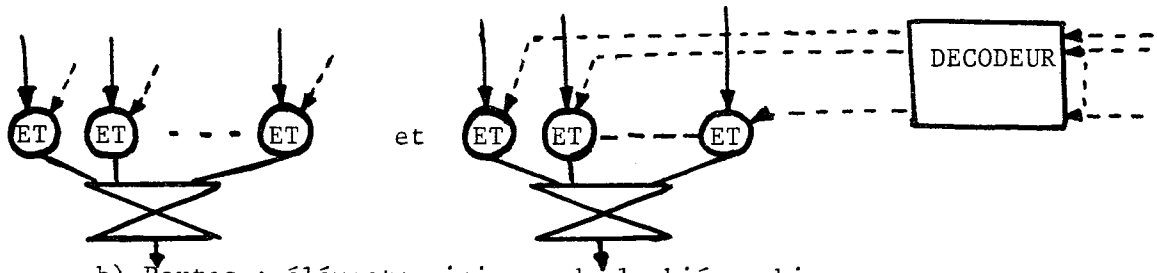
Soit 11) C'est également vrai pour une unité quelconque. On continue le traitement à partir des entrées CONTROLE après application de $\mathcal{E}$ (U).



Soit 12)



Que l'on transforme en les réseaux

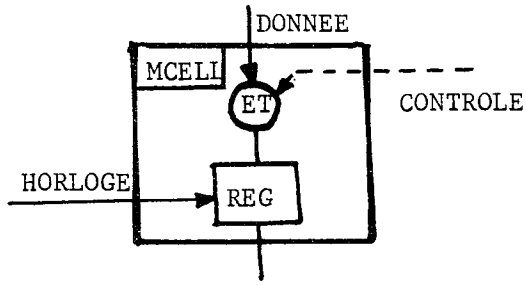


b) Portes : éléments minimums de la hiérarchie :

Les signaux de contrôle suivent un réseau totalement orienté sans circuit : c'est une hiérarchie. Ils vont de ce que nous avons appelé les racines (exemple : mémoire morte d'un calculateur) jusqu'aux points de contact du contrôle avec le chemin de donnée qui se fait par l'intermédiaire de portes (et seulement à travers elles), que nous avons représenté comme des opérateurs ET 11.

Ces portes se situent soit au milieu de réseaux combinatoires (Unités EXP ou autres), on verra ultérieurement leur traitement, soit en entrée des articulations de type bus ou registre.

Rappelons en effet, le schéma de principe du registre en CASSANDRE :



Il est toujours possible de numéroter les articulations d'une hiérarchie (Théorie des Réseaux de J. KUNTZMANN), cette numérotation préalable permettra au programme de suivre les signaux de contrôle en cas de besoin sans opération inutile (on connaît en arrivant à chaque articulation la nature de ses entrées).

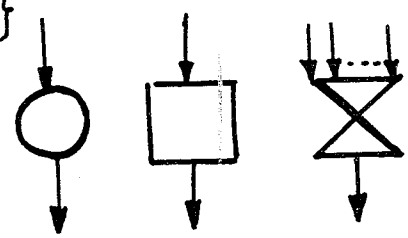
Le résultat du traitement est la création de 3 classes d'articulations.

$\mathcal{P}$ {ensemble des portes}

$\mathcal{R}$ {ensemble des éléments de mémorisation du chemin de données}

$\mathcal{B}$ {ensemble des bus du chemin de données}

schématisées respectivement



c) Exemple du Z0001 :

Nous avons tracé (chapitre précédent) la hiérarchie de contrôle de l'unité Z0001, à partir du résultat de sa compilation.Elle est constituée des sous réseaux dont les connexions sont en pointillés. Les classes ainsi trouvées sont : (désignées par le n° des boites sur le schéma).

$\mathcal{P}$ { (11,12,13,14,15,16,17) dans COMPTEUR, (10,11,12,13,14,15,16) dans COMPT, (6,7,8,9,10,11,12,13,14,15,16,17) dans R, (4,5) dans ML, (5,6,7,8,9,10, 11,12,13,14) dans SPES, (4,5,6) dans UAL, (5,6,7,8) dans DECG, (5,6,7,8) dans DECD, (9,10,11,12,13,14,15,16) dans DEC.

Les 8 entrées de BUS 10000 et de BUS 10001 dans Z0001, les 16 entrées BUS 04000 dans LINPUTBS , les entrées des registres, 40 dans DEC, 7 et 8 dans UAL, 22 dans SPES, 3 dans ML, 20 dans COMPT, et 21 dans COMPTEUR.}

R { 21 de COMPTEUR, 20 de COMPT, 3 de ML, 22 de SPES, (7 et 8) de UAL, 40 de DEC. }

B { 9 de DECG, 9 de DECD, 43 de DEC, (5,28) de SPES, 28 de LINPUTBS 10 de UAL, (BUS 10000, BUS 10001) de Z0001, (4,5) de ML, (16,19) de R, (21,22,23) de COMPT et (22,23,24) de COMPTEUR. }

REMARQUES : 1) Si on admet que CONS2 venant de la mémoire morte est de la classe contrôle, alors l'unité COMPT ayant toutes ses entrées dans CONTROLE y a aussi toutes ses sorties et tous ses éléments qu'il faut donc soustraire des classes précédentes.

Par contre, le registre A (n° 40) de DEC, reçoit son contenu du chemin de données et donc appartient à celui-ci, bien que sa sortie soit déclarée contrôle donc qu'il soit l'une des racines de la hiérarchie de contrôle.

2) On peut voir que la majorité des boîtes générées par la compilation de Z0001 appartiennent au contrôle, seules les unités DECG, DECD, COMPTEUR et UAL contiennent des boites qui ne sont ni dans la classe du contrôle ni dans l'une des 3 classes que nous venons de définir. C'est à ces réseaux résiduels que nous nous intéressons dans le paragraphe suivant. Ils réalisent les fonctions combinatoires du chemin de données, donc les fonctions de la machine.

III - Réseaux du chemin de donnée

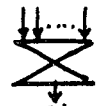
a) Réseau primaire de noeuds et d'étoiles :

1) Articulations de base

Les unités primitives du compilateur de hardware peuvent du point de vue qui est le notre, maintenant être remplacées par un réseau équivalent afin d'arriver à un ensemble plus restreint d'articulations type.

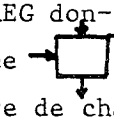
α) BUS

Les unités appelées bus donnent des articulations de base de type porte et bus.

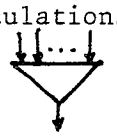
L'articulation bus (notée ) a n entrées et une sortie. Ces entrées proviennent chacune d'une porte et par définition le contrôle agissant sur ces portes permet d'en ouvrir une seule à la fois.

L'articulation bus que l'on pourrait nommer multiplexeur est la fonction "1 parmi n" de ces entrées.

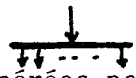
β) Registres

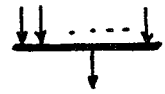
Les unités primitives de type MCELL ou terminales de type REG donnent des articulations de base de type porte et registre (notée ).

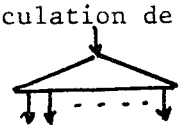
Le contrôle permet en ouvrant la porte en entrée du registre de charger celui-ci (lorsqu' arrive l'impulsion d'horloge).

γ) Les unités primitives de type ET, OU, NON, NI, NONET et les non primitives EXP, donnent des articulations à n entrées et 1 sortie du type opérateur booléen (notées ).

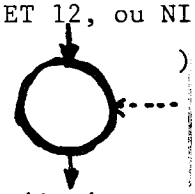
δ) Dans la version asynchrone de CASSANDRE apparaissent des opérateurs de retard (τ). Ceux-ci le cas échéant, donneront des articulations de type retard (notée  qui ont une signification analogue à celle des registres. Mais au lieu de stocker l'information pour une durée qui ne dépend que du contrôle, les retards la stockent pour une durée fixe puis la détruisent (d'ailleurs leur entrée ne sera pas conditionnée par une porte).

ε) La sortie de chacune des articulations précédentes peut être connectée en entrée de plusieurs autres articulations. Chaque fois que ce sera le cas, nous allons générer une nouvelle articulation de type <u>expanseur</u> (notée ) celles-ci s'ajouteront aux articulations-expanseurs générées par les unités primitives du même nom.

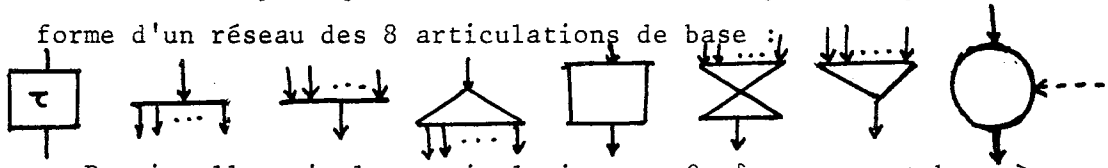
i) Les unités EXP du type concaténation vont générer des articulations de concaténation à n entrées et une sortie (notées ).

η) Une même sortie d'unité (ou d'articulation) est en CASSANDRE une variable générale (de type tableau). On peut utiliser une partie (sous tableau). Chaque fois qu'une variable sera utilisée plusieurs fois partiellement il nous faudra générer une articulation de type <u>DELTA</u> (par analogie avec le delta des fleuves, notée ).

<u>Exemple</u> : dans ZOOO1, l'entrée SSPES (0:3, 1:16) de R est aussi l'entrée d'une articulation de type DELTA dont les sorties sont SSPES (0,), SSPES (1,) SSPES (2,), SSPES (3,).

κ) Les unités primitives ET 1, NI 1 et les non primitives AND peuvent s'exprimer on l'a vu, à l'aide d'unités ET 12, ou NI 12, qui donnent des articulations de type <u>porte</u> (notées ), et d'unités ET 2 et OU.

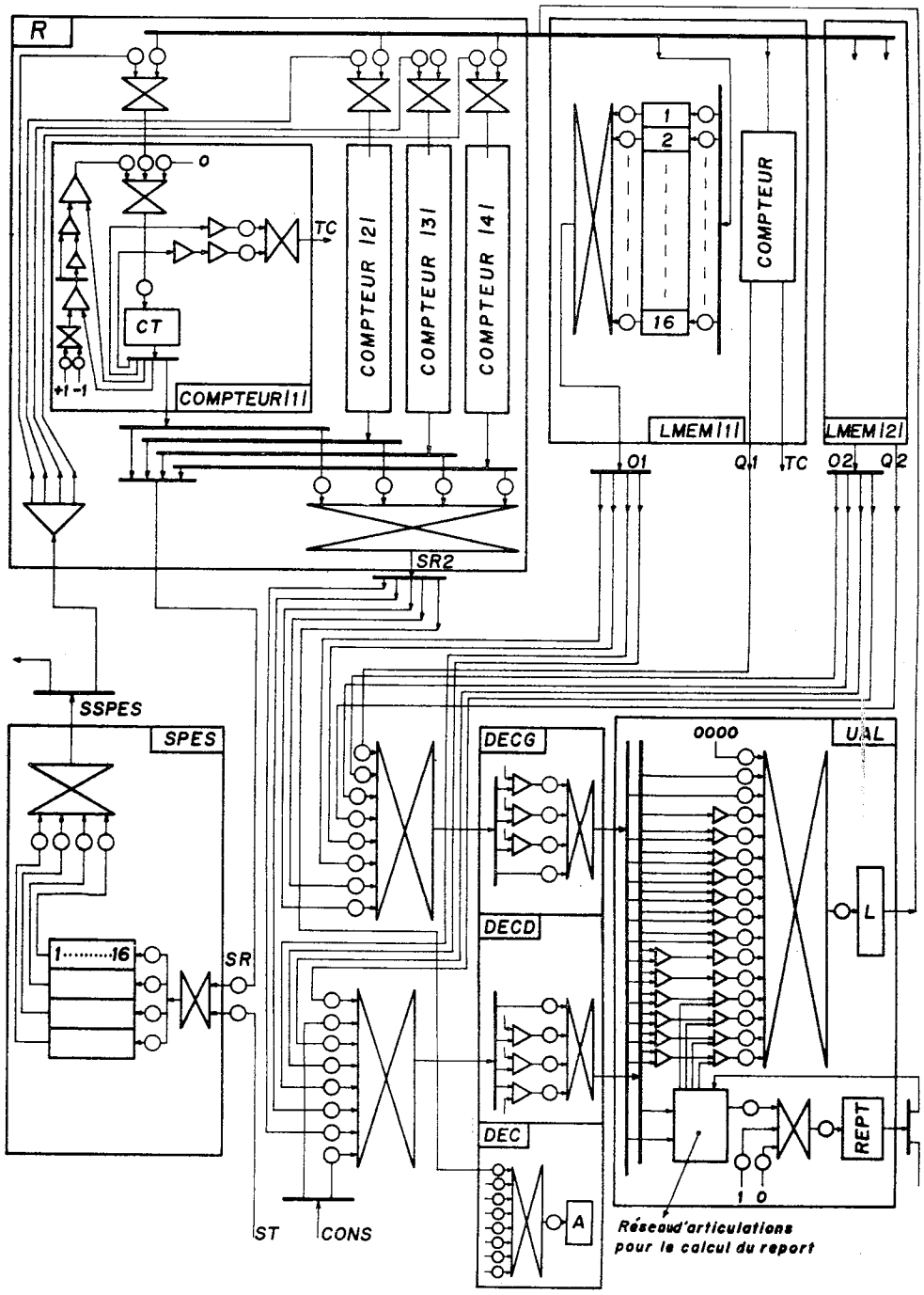
Toute description CASSANDRE après compilation et après les transformations simples que nous venons de définir peut s'exprimer sous forme d'un réseau des 8 articulations de base :



Parmi celles-ci, les articulations α, β, δ, ε, κ sont homogènes c'est à dire le nombre de bits qui rentrent à un instant donné est égal à celui qui sort par toute sortie) γ , i et η ne sont pas homogènes.

2) Exemple du Z0001 :

Nous pouvons extraire du schéma précédent le réseau suivant :

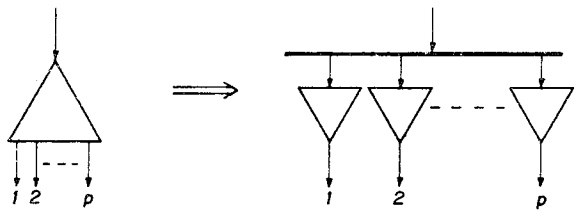


Réseau, encore complexe qui représente le chemin de données de Z0001.

3) Simplifications primaires :

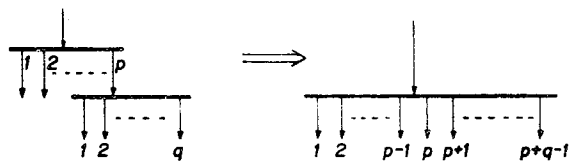
Le réseau précédent est calqué sur le surréseau que l'on a après compilation, nous allons le réduire par des applications $\mathfrak{E}$ à un seul niveau.

α) On remplace une articulation de type Δ par le réseau :



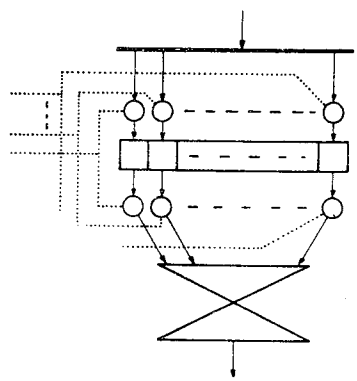
Composé d'un expanseur et de p opérateurs booléens (chacun fait en réalité l'opération d'extraction d'une sous-variable de la variable d'entrée).

β) On réduit ensuite les expanseurs consécutifs



γ) Articulations mémoire

Chaque fois que l'on trouve le réseau suivant :



Si l'on a le même signal de contrôle sur la porte qui se trouve en entrée d'un registre et sur celle qui se trouve en sortie, on crée une nouvelle articulation mémoire (notée ) avec une porte en entrée pour conditionner son chargement.

Comme les portes en sortie de registre sont en entrée d'un bus, on peut affirmer que les signaux de contrôle sont des conditions logiques exclusives (un seul d'entre eux est vrai à un instant donné). Dans ce

réseau on ne peut accéder à la fois qu'à un seul registre en entrée et en sortie (le même).

DOnc l'articulation mémoire est fonctionnellement équivalente à une articulation registre (bien que ce registre change au cours du temps). On lui associe deux informations $\gamma 1$) Le nombre de registres qu'elle contient.

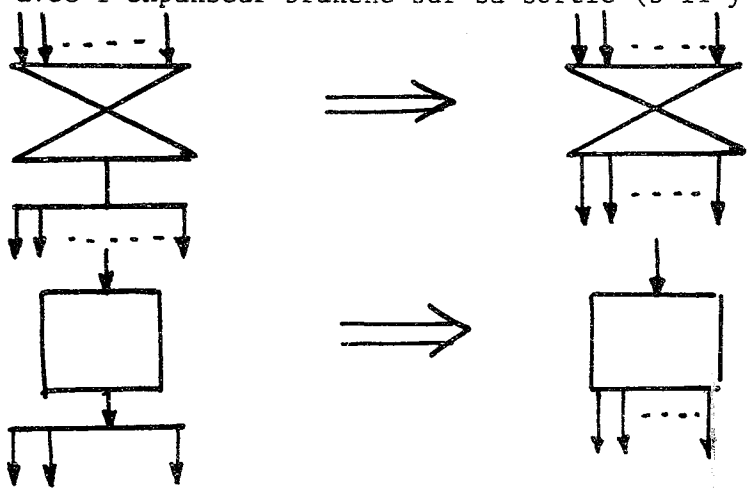
$\gamma 2$) Le nom du registre d'adresse (c'est à dire dont la sortie déclarée "contrôle" vient après décodage commander les portes de la "mémoire").

δ) Articulations "fonction booléenne"

Repartons des 3 classes, définies dans II, c, $\mathcal{P}, \mathcal{R}, \mathcal{B}$.

Pour ce traitement nous regroupons chaque élément soit de $\mathcal{R}$ soit de $\mathcal{B}$ avec l'expanseur branché sur sa sortie (s'il y en a un).

Exemple :

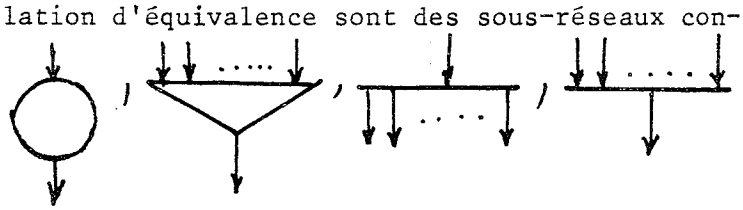


On obtient des articulations à plusieurs sorties formant deux classes $\mathcal{R}'$ et $\mathcal{B}'$. Toutes les portes de $\mathcal{P}$ dont la sortie est connectée à une entrée d'un élément de $\mathcal{R}'$ ou $\mathcal{B}'$ appartient à une nouvelle classe, $\mathcal{P}'$ (ou a $\mathcal{P}' \subset \mathcal{P}$).

On défini la relation d'équivalence suivante :

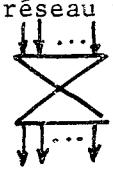
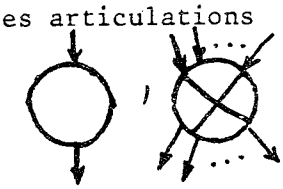
- Deux articulations n'appartenant pas à $\mathcal{P}'$ ou $\mathcal{B}'$, connectées entre elles sont équivalentes . (Relation clairement réflexive, symétrique et transitive).

Les classes de cette relation d'équivalence sont des sous-réseaux connexes d'articulations

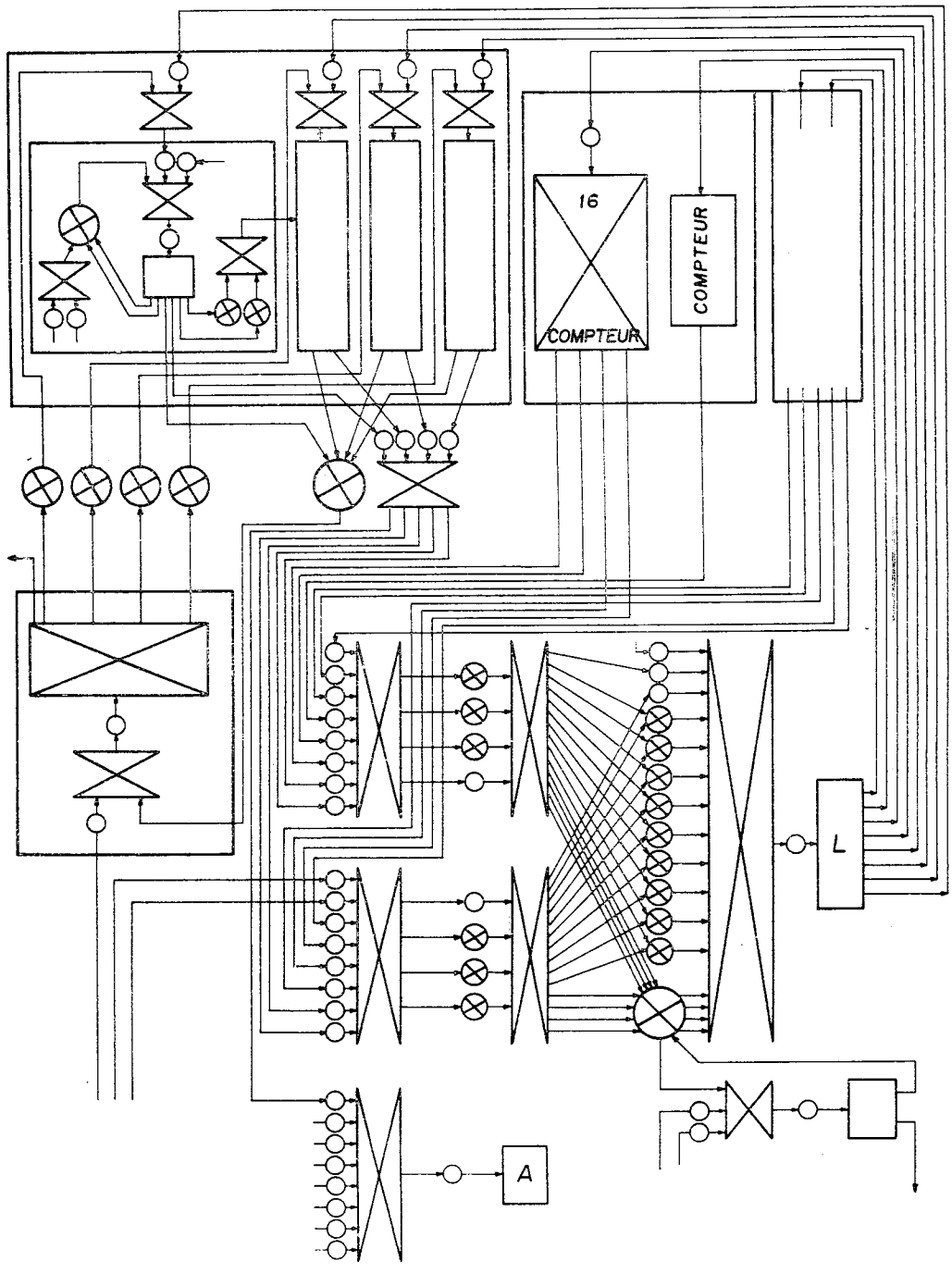


Chacun constitue une articulation du type "fonction booléenne" (notée )

Le réseau obtenu après les simplifications primaires sera appelé réseau primaire .

C'est un réseau noeuds-étoiles où les noeuds sont les articulations de type  et les étoiles du type 

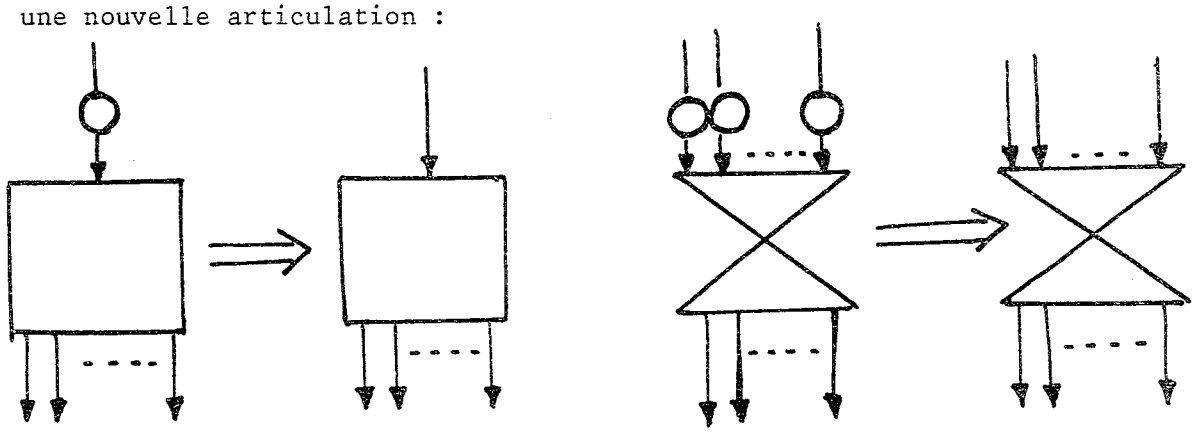
ε) Exemple du Z0001 : transformation du réseau précédent en réseau primaire.



b) Réseau secondaire "canonique" :

Le réseau primaire a 4 sortes d'articulations. Toutes les sorties d'une articulation "fonction booléenne" vont sur une entrée d'une articulation  ou  , donc proviennent d'une porte  située à l'intérieur. (On peut facilement vérifier qu'au cours du traitement en vue d'obtenir le réseau primaire, on a absorbé dans les articulations  une porte sur chaque sortie).

On les extrait maintenant. De nouveau chaque entrée d'une articulation registre ou bus provient de la sortie d'une porte. On crée à chaque fois une nouvelle articulation :



On fait disparaître par absorption dans les classes $\mathcal{R}'$ et $\mathcal{B}'$, toute la classe $\mathcal{P}'$.

(Il faut noter qu'il reste d'autres portes à l'intérieur des articulations "fonctions booléennes" mais elles ne sont pas connectées à ses sorties)

Après ce premier regroupement nous ne nous intéresserons plus qu'aux articulations ).

1) Extraction des portes appartenant à $\mathcal{P} - \mathcal{P}'$:

α) Ces portes appartiennent donc aux réseaux contenus dans les articulations  . On va les chasser vers les sorties de celles-ci, en modifiant son réseau tout en le maintenant fonctionnellement invariant.

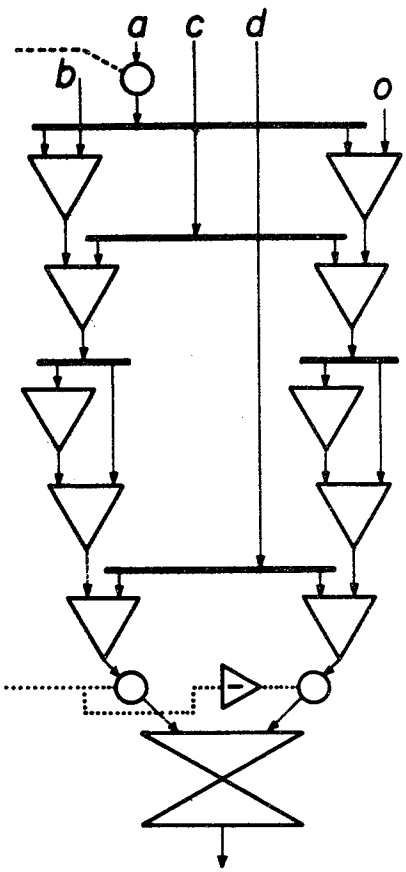
Le réseau dans l'articulation est lui aussi une hiérarchie. Considérons une porte P comme l'élément minimum d'une autre hiérarchie

(sous-réseau du précédent). Nous appliquons κ à ce sous-réseau pour obtenir une autre articulation (soit U) de type ⊗ .

Dupliquons cette articulation en U |1| et U |2|. L'une des entrées de U était par construction la sortie de la porte P traitée, dans U |1| nous la connectons avec l'entrée de cette porte et dans U |2| avec la valeur 0. Les autres entrées de U |1| et U |2| sont connectées aux mêmes sorties d'articulations que les entrées correspondantes de U.

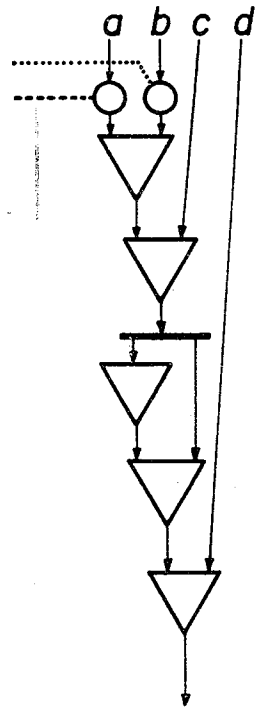
Nous considérons deux à deux les sorties correspondantes de U |1| et U |2| et nous créons pour chaque couple un bus à deux entrées conditionnées l'une par le même signal de contrôle que P, l'autre par son complément. Une fois ce traitement appliqué, la porte P a disparu de l'articulation ⊗ dans laquelle elle se trouvait. On recommence le traitement pour les autres portes, s'il en reste.

β) Exemple : Il n'y a pas dans le réseau primaire de Z0001 d'articulation ⊗ qui contienne de porte. Prenons l'unité COMPTEUR et remplaçons pour les besoins de l'exemple le bus correspondant à la boite n° 22 par un opérateur OU. Le réseau obtenu (fonctionnellement équivalent) est le suivant :

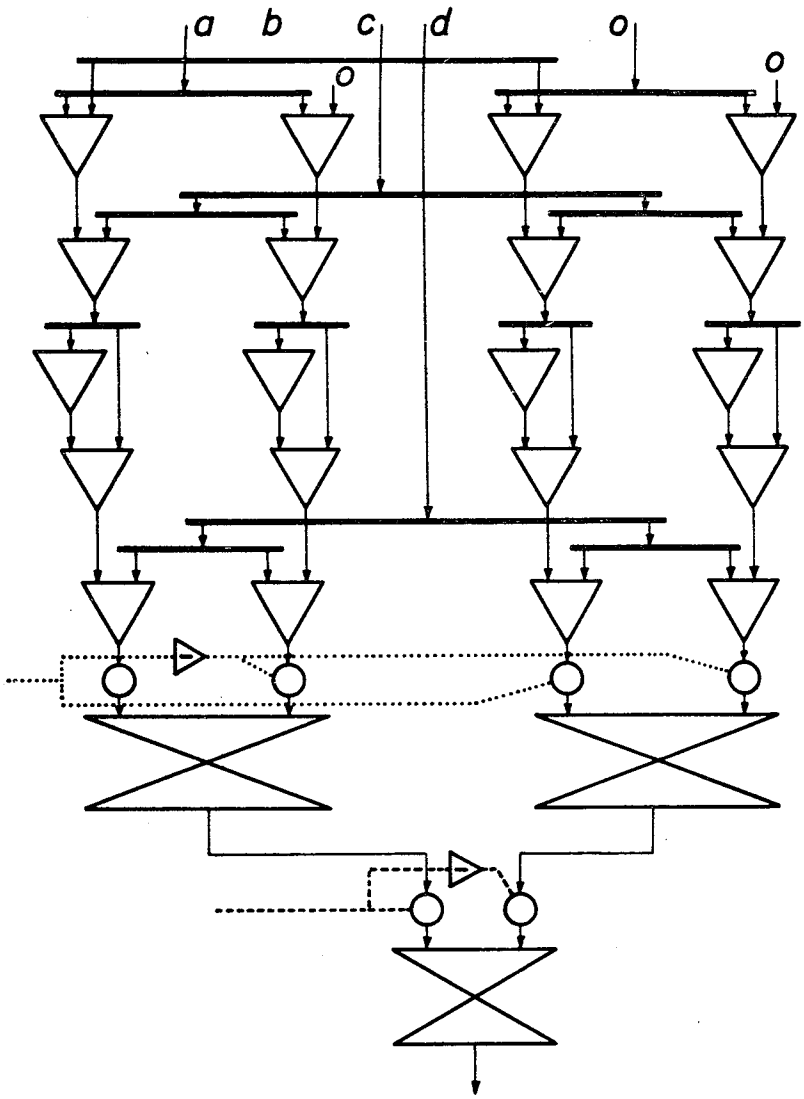


Comme les portes sont en entrée la hiérarchie dont elles sont l'élément minimum se confond avec le réseau tout entier.

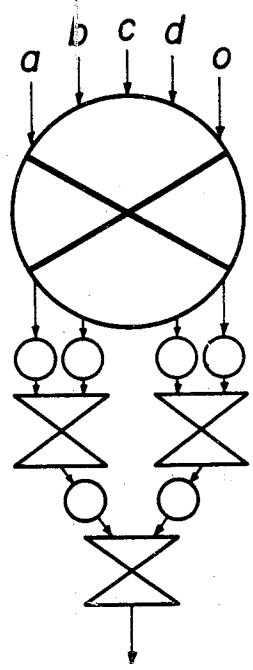
Traitement de la 1ère porte



Traitement de la 2e porte



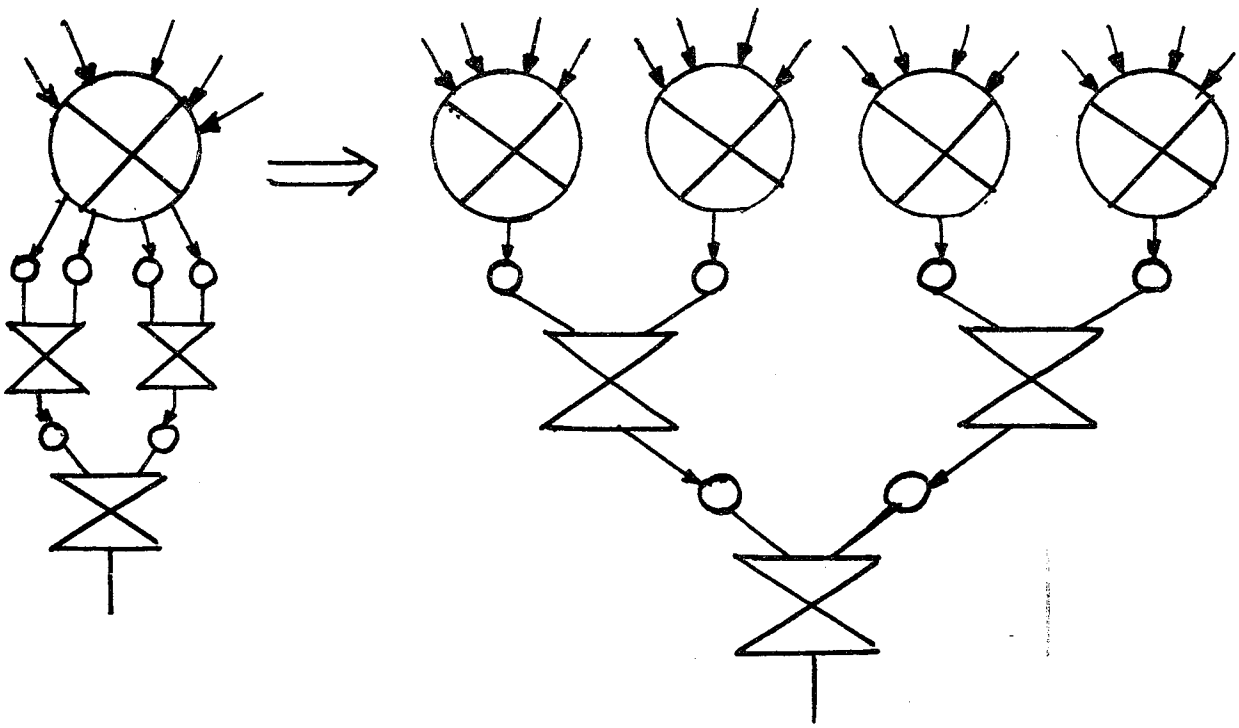
On obtient donc le réseau suivant
qui correspond aux mêmes fonctions :



2) Découpage longitudinal des articulations "fonctions booléennes"

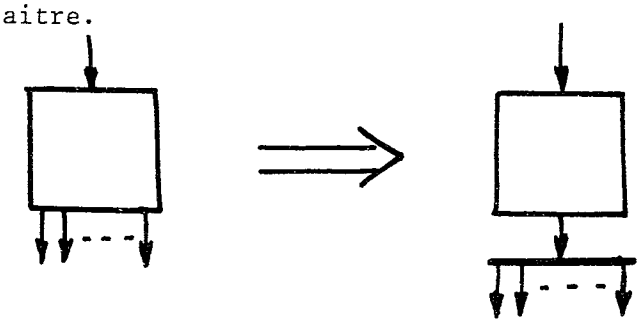
Nous avons décrit dans C - c-2 - un processus de découpage longitudinal de réseaux combinatoires. Nous l'appliquons à toutes les articulations qui ont plusieurs sorties, on obtient autant de nouvelles articulations qu'il y avait de sorties. On sépare ainsi toutes les donctions différentes de la machine dont on est en train de traiter le réseau primaire.

Exemple : Reprenons l'exemple précédent :



Après avoir appliqué ce traitement à toutes les articulations "fonctions booléennes" on n'a plus que des articulations "fonctions booléennes simples" (à une sortie). Pour les articulations de type , , on fait l'opération inverse de celle que l'on a faite pour obtenir le réseau primaire. Si elles ont plusieurs sorties, il y a un expanseur que l'on fait apparaitre.

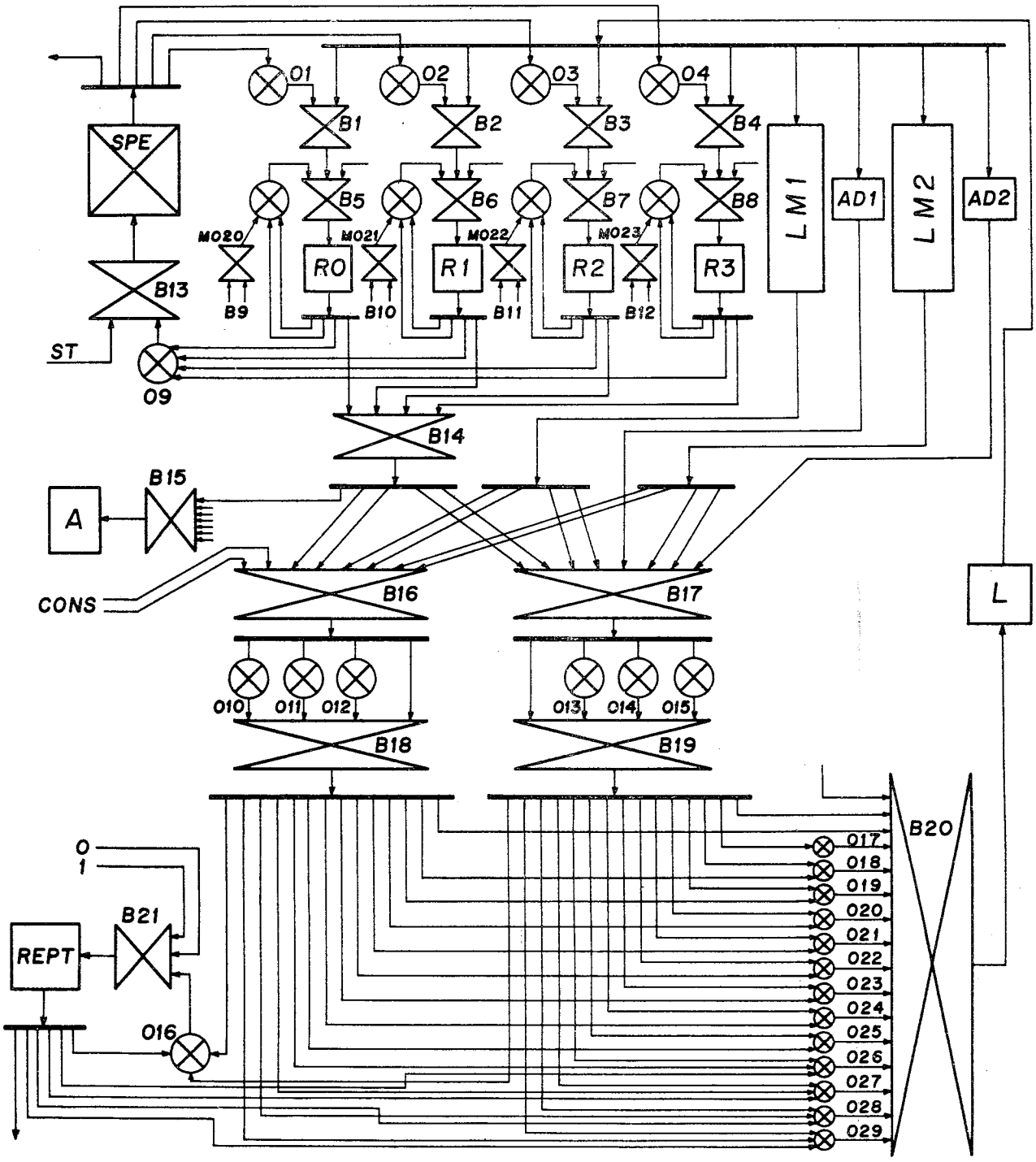
Exemple :



3) Propriétés du réseau canonique secondaire :

α) Exemple du Z0001 :

Nous simplifions le réseau en enlevant de COMPTEUR les parties relatives au sur-débordement et sous-débordement.

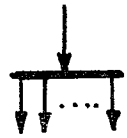


β) Le réseau secondaire contient 4 sortes d'articulations :

β1) Registres  (ou mémoires) qui ont une entrée et une sortie.

β2) Bus  , n entrées, 1 sortie

β3) Opérateur booléen  , p entrées, 1 sortie

β4) Expanseur  , 1 entrée, q sorties.

- Le sous-réseau formé uniquement des articulations opérateur-booléen est disconnecté.

- Les signaux de la classe de contrôle conditionnent chacune des entrées des bus et des registres.

- Réciproquement, aucun signal de contrôle ne s'applique ailleurs qu'en entrée d'un bus ou d'un registre (les entrées d'opérateurs booléens ne sont jamais conditionnées).

- La hiérarchie de contrôle est restée invariante dans le traitement qui permet d'obtenir le réseau canonique (on ajoute seulement la complémentation de certaines de ses sorties, lorsqu'on chasse les portes des articulations fonctions booléennes).

Soit une machine digitale M quelconque décrite en CASSANDRE et soit H sa hiérarchie de contrôle : "Il existe toujours une machine M' ayant les mêmes éléments de mémorisation, les mêmes fonctions, les mêmes transferts possibles de données et la même hiérarchie de contrôle H que M, telle qu'il y ait une bijection entre les sorties de H et les entrées des registres et des bus de M'".

Nous appelerons M' la réalisation canonique de M . Vues de l'extérieur les boites noires M et M' sont impossibles à distinguer. On pourra dire aussi qu'elles sont équivalentes et identiquement contrôlées (plus loin on verra des machines équivalentes n'ayant pas la même hiérarchie de contrôle).

- Tout réseau canonique peut se décrire à l'aide d'une grammaire contexte-libre.

4) Grammaire associée au Z0001 :

α) Les éléments non terminaux sont :

α1) les sorties des éléments de mémorisation soit :

SPE, RO, R1, R2, R3, LM1, AD1, LM2, AD2, L, REPT.

α2) Les bus et opérateurs booléens soit :

B1, B2,..., B21, O1, O2,..., O29.

β) Les éléments terminaux sont les entrées des registres et mémoires (à l'exception de L qui est dans Z0001 utilisé comme retard). soit :

spe, ro, r1, r2, r3, lm1, ad1, lm2, ad2, rept.

γ) Règles : / indique l'alternative, " (", " ; ", ")" les identifieurs (entre ' ') et les constantes 0 et 1 sont d'autres terminaux

< CHARGEMENT - DE - REGISTRE >::=

spe ← B13 / ro ← B5 / r1 ← B6 / r2 ←B7 / r3 ← B8 /

lm1 ← L / ad1 ← L / lm2 ← L / ad2 ← L / rept ← B21 ;

B1 ::= L / O1 ;

B2 ::= L / O2 ;

B3 ::= L / O3 ;

B4 ::= L / O4 ;

B5 ::= O5 / B1/ 0000 ; B6 ::= O6 / B2/ 0000 ; B7 ::= O7 / B3/ 0000 ;

B8 ::= O8 / B4/ 0000 ; B9 ::= 0001 / 111 ; B10 ::= 0001 / 111 ;

B11 ::= 0001 / 1111 ; B12 ::= 0001 / 1111 ; B13 ::= O9 / ST ;

B14 ::= RO / R1 / R2 / R3 ; B15 ::= B14 / 00 / 01 / 10 / 11 ;

B16 ::= CONS / B14 / LM1 / LM2 ;

B17 ::= B14 / LM1 / AD1 / LM2 / AD2 ;

B18 ::= O10 / O11 / O12 / B16 ; B19 ::= B17 / O13 / O14 / O15 ;

B20 ::=0000 / B19 / B18 / 017 / ... / 029 ;

B21 ::= 0 / 1 / 016 ;

01 ::= 'OP1' (SPE) ; 02 ::= 'OP2' (SPE) ; 03 ::= 'OP3' (SPE) ;

04 ::= 'OP4' (SPE) ;

05 ::= 'OP5' (B9,R0, R0) ; 06 ::= 'OP6' (B10, R1, R1) ;

07 ::= 'OP7' (B11,R2,R2) ; 08 ::= 'OP8' (B12,R3,R3) ;

09 ::= 'OP9' (R0, R1, R2, R3) ; 010 ::= 'OP10' (B16) ;

011 ::= 'OP11' (B16); 012 ::= 'OP12' (B16) ; 013 ::= 'OP13' (B17) ;

014 ::= 'OP14' (B17) ; 015 ::= 'OP15' (B17) ;

016 ::= 'OP16' (B18,B19,REPT) ; 017 ::= 'OP17' (B19) ;

018 ::= 'OP18' (B18 ,B19) ;.....; 025 ::= 'OP25' (B18,B19) ;

026 ::= 'OP26' (B18,B19,REPT) ; ... ; 029 ::= 'OP29' (B18,B19,REPT) ;

δ) Si la machine est microprogrammée on peut rajouter la règle :

< MICROINSTRUCTION > ::= < LISTE-DE-CHARGEMENTS-DE-REG > en fixant les contraintes de simultanéité de chargement si on le veut.

c) Réseau structurel :

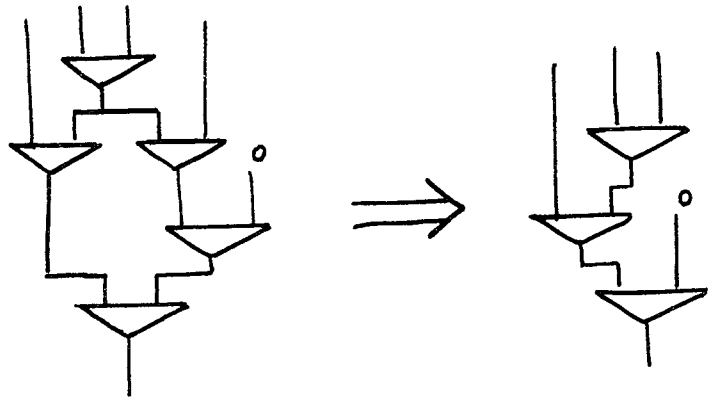
Nous avons jusqu'alors laissée invariante la hiérarchie de contrôle. Les simplifications que nous allons faire maintenant vont au contraire la modifier, mais dans le sens d'une extension seulement (la hiérarchie initiale sera un sous-réseau de celle que nous obtiendrons).

1) Simplifications booléennes

Dans les traitements précédents on a pu introduire dans les réseaux des articulations "opérateurs-booléen" des constantes 1 et 0. (Le compilateur de hardware lui-même peut en avoir un). On va opérer les simplifications les plus élémentaires.

α) Tout 0 en entrée d'un opérateur ET se traduit par un zéro sur sa sortie et la suppression de l'opérateur (ou de l'arbre qui y aboutit)

Exemple :



Au cours de ces simplifications, le nombre des entrées peut être modifié.

β) Tout opérateur OU qui a un 1 en entrée est supprimé (avec éventuellement l'arbre qui y aboutit). Sa sortie est mise à 1.

γ) Toute entrée à 1 sur un opérateur ET ou à 0 sur un opérateur OU est supprimée.

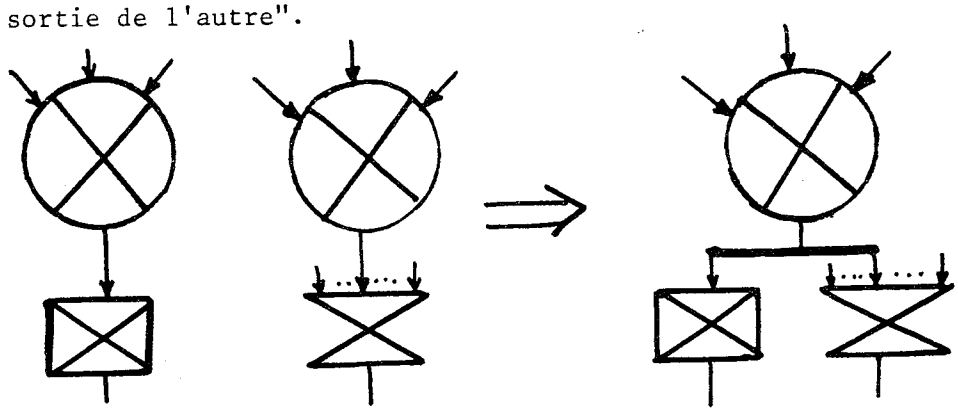
δ) Tout opérateur (autre que NON) qui n'a qu'une entrée est supprimé, sa sortie est branchée sur l'entrée.

2) Suppression de la redondance :

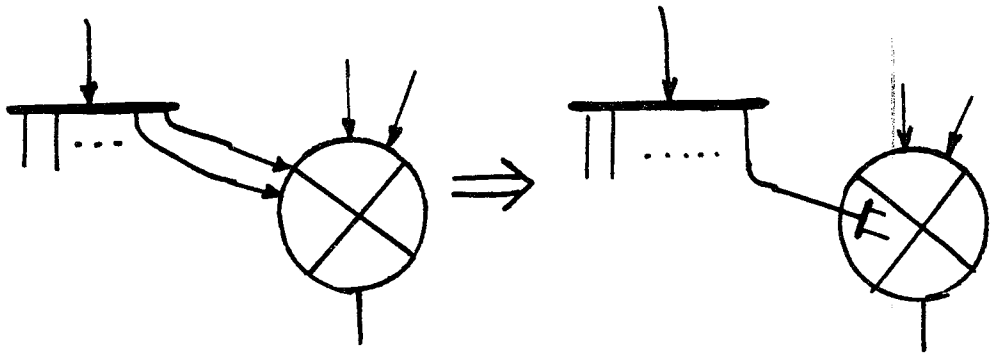
α) On traduit en équations booléennes (voir chapitre précédent) tout opérateur booléen qui subsiste après les simplifications.

Puis on compare deux à deux, par disjonction les équations des opérateurs qui ont les mêmes entrées.

"Si deux opérateurs qui ont les mêmes entrées effectuent la même fonction, on supprime l'un d'entre eux et on met un expanseur sur la sortie de l'autre".



β) Si un opérateur a plusieurs entrées en provenance du même expanseur on les réduit à une seule et on met un expanseur dans l'opérateur.



Après ce traitement tous les opérateurs booléens qui restent, effectuent des fonctions différentes, ou traitent des variables différentes, chacun a le minimum d'entrées et une sortie.

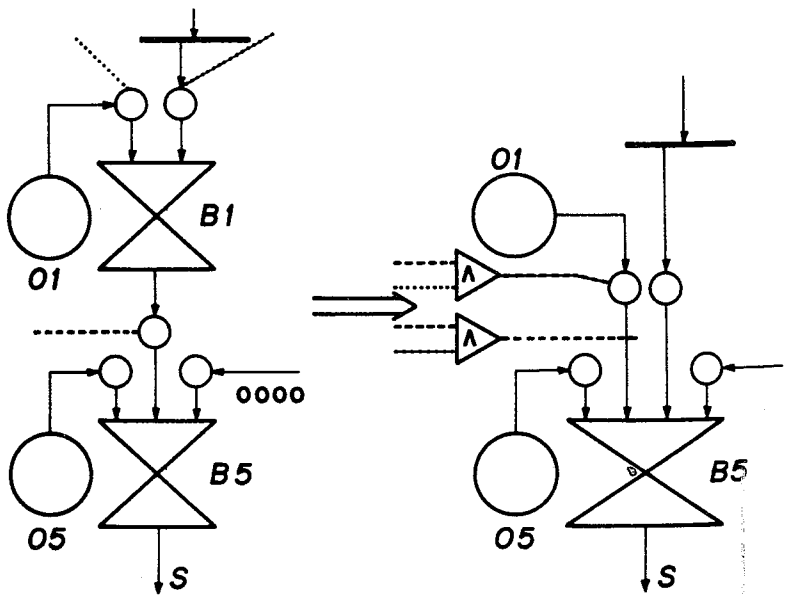
3) Réduction des bus :

Nous définissons cette réduction "à parallélisme constant".

Tout bus dont la sortie est connectée directement (et non par l'intermédiaire d'un expanseur) à un autre bus disparaît.

On fait l'intersection de la condition logique qui conditionnait sa sortie (une entrée du bus suivant) avec toutes les conditions logiques qui conditionnent ses entrées et on branche ces entrées sur le bus suivant.

Exemple :

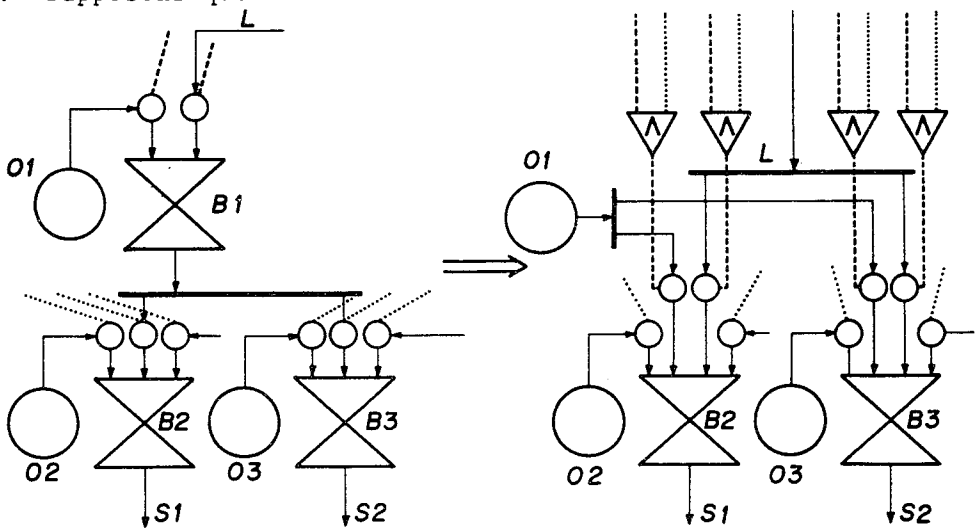


Un seul des 4 signaux d'entrée des 2 réseaux est disponible à un instant donné sur la sortie S dans les 2 cas. Même si l'on décide maintenant de coder le contrôle des entrées de B'5 avec d'autres variables, une seule parmi les 4 sera sélectée à un instant donné. La transformation est à parallélisme constant.

Cette réduction fait disparaître certains bus introduits lors du traitement de D - b - 1 , mais également d'autres qui pouvaient exister dans le réseau de départ.

Montrons que la réduction d'un bus qui est branché en sortie sur plusieurs bus n'est pas à parallélisme constant.

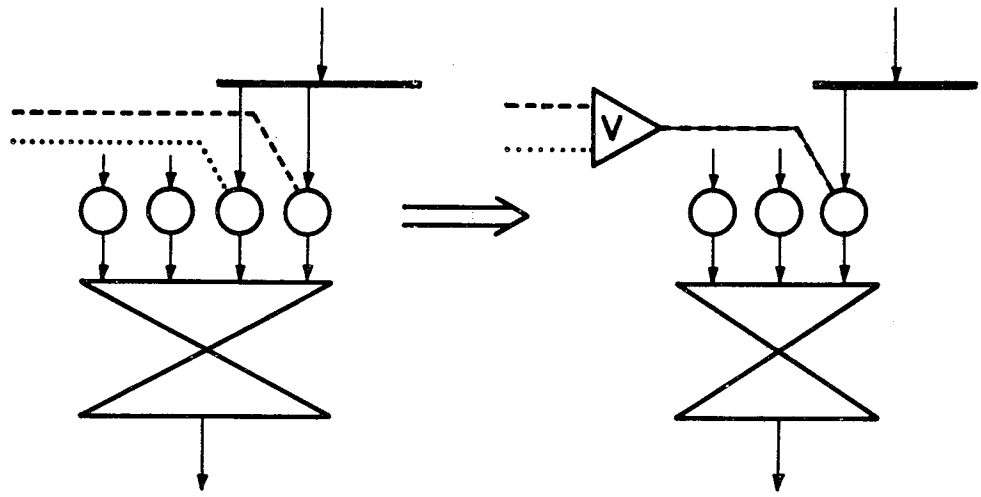
Exemple : Supposons que l'on ait :



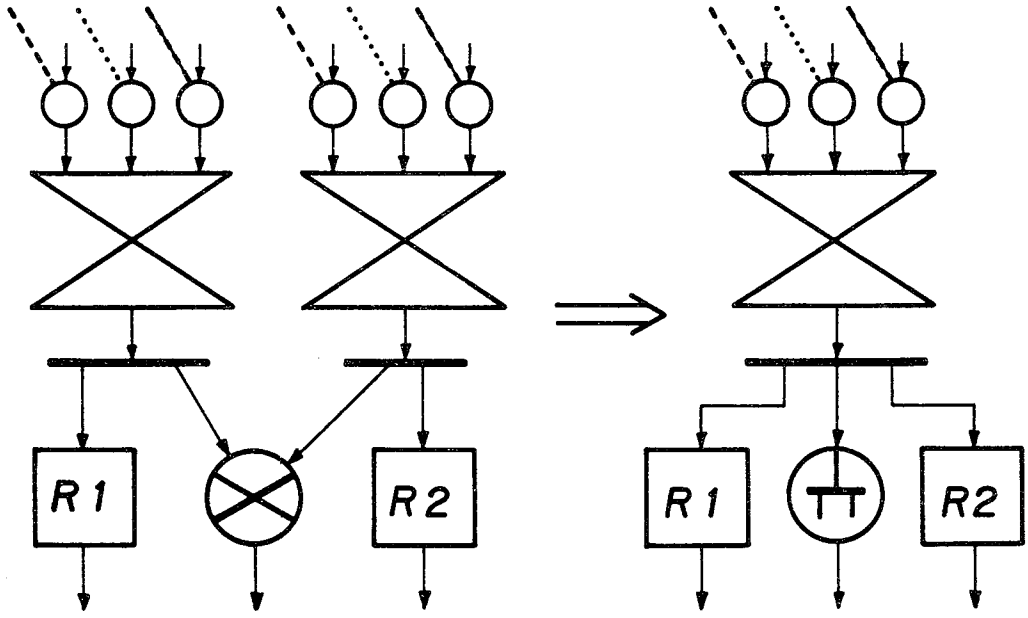
Les simultanéités permises restent les mêmes car la transformation opérée sur la hiérarchie de contrôle les maintient. Mais si nous décidons de coder avec de nouvelles variables indépendantes les signaux de contrôle des entrées de B'2 et B' 3, rien n'empêchera ensuite d'envoyer par exemple O1 sur S1 et L sur S2, alors que c'était impossible avant transformation.

La "structure" du chemin de données a donc été modifiée.

Sur les bus qui demeurent après traitement nous faisons si nécessaire les réductions suivantes . Si plusieurs entrées viennent du même expanseur on les réduit à une seule conditionnée par le OU logique des conditions précédentes.



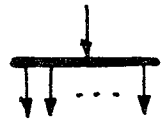
Si deux bus sont connectés aux mêmes entrées avec les mêmes conditions de contrôle, on supprime l'un d'entre eux et on conserve toutes les connexions qui existaient en sortie.



Il est clair, parmi les simplifications successives que nous venons d'indiquer, que le résultat de certaines en entraîne d'autres. Lorsque plus aucune ne peut s'appliquer, le réseau obtenu est le "réseau structurel" du système traité.

4) Propriétés du réseau structurel :

α) Il est constitué de 4 types d'articulations :

éléments de mémorisation,  bus,  opérateurs booléens,  expanseurs. 

β) Chacun des sous-réseaux constitués d'un seul type d'articulations est totalement disconnecté. Le réseau structurel n'est pas moins de 4 coloriable.
Chaque articulation d'un type donné peut être connectée à des articulations des 3 autres types.

γ) La hiérarchie de contrôle n'est en contact avec le réseau structurel que par les entrées des bus et l'entrée des registres.

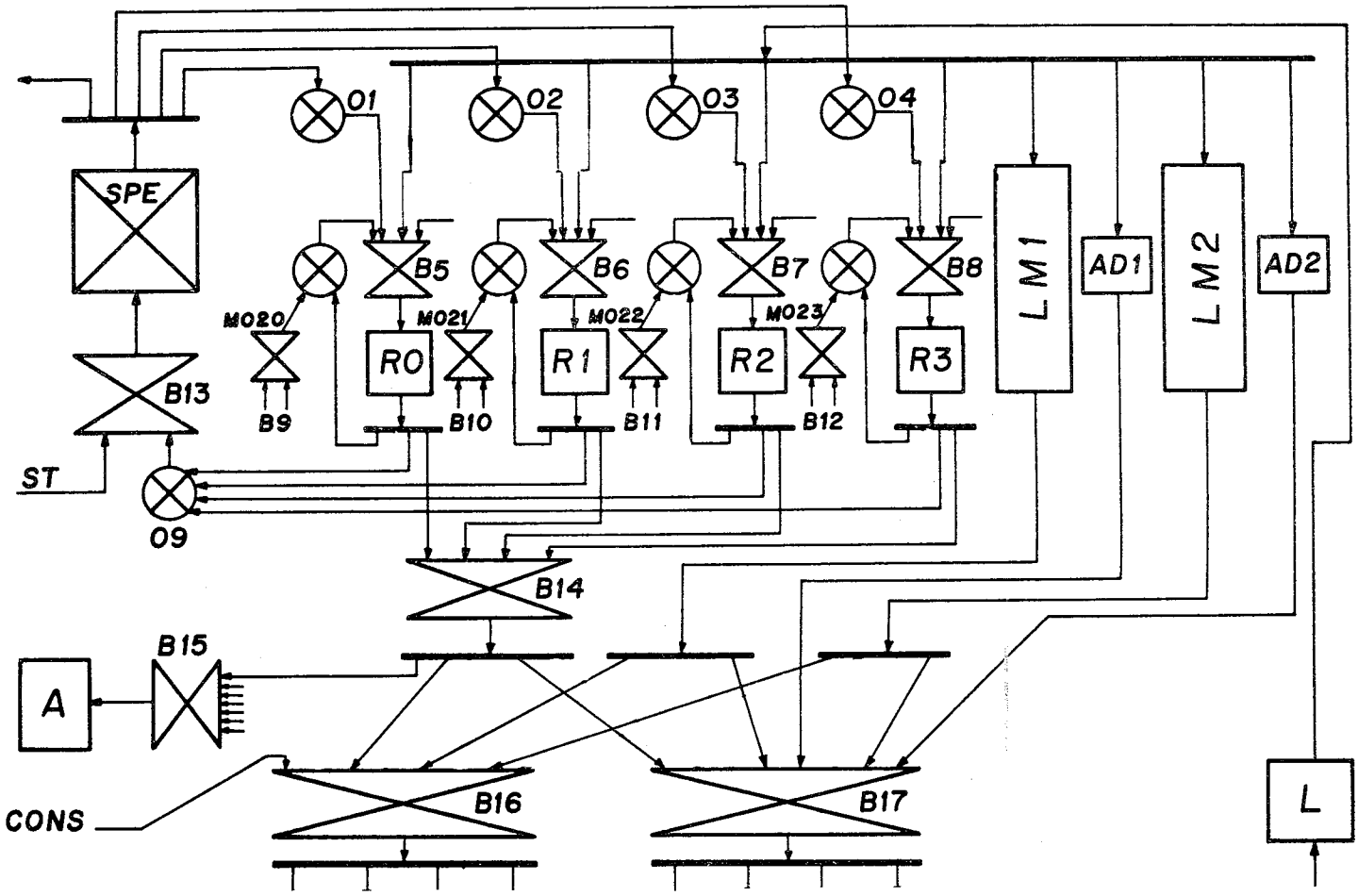
δ) Deux machines qui ont le même réseau structurel seront dites "de même structure".

ε) Deux machines de même structure ayant des éléments de mamorisation semblables deux à deux seront dites "semblables".

ι) Deux machines semblables ayant des opérateurs booléens identiques deux à deux sont "fonctionnellement équivalentes".

5) Exemple de Z0001 :

Seule la partie haute du réseau secondaire subit des modifications.



d) Applications à la microprogrammation :

1) Métacompilateur de microprogrammes :

Le problème consiste à compiler la description dans un certain langage (exemple Algol W) d'algorithmes et d'opérations, en un ensemble de microinstructions exécutables sur une machine que l'on connaît par sa description en CASSANDRE. ANCEAU a donné en séminaire puis dans (2) une méthode pour réaliser une partie du travail : si l'on peut représenter sous forme d'une grammaire les transferts, les simultanéités et les opérations de la machine (celle qui devra recevoir les microprogrammes compilés) on utilise cette grammaire pour analyser la description des algorithmes à compiler.

Il faut : α1) que les symboles de base du langage analysé soient les mêmes que ceux qui apparaissent dans la grammaire.

α2) que l'on puisse introduire dans les règles de celle-ci les fonctions sémantiques qui vont générer chaque partie du code de la microinstruction.

α3) que l'on associe à l'analyseur syntaxique ainsi obtenu une table d'utilisation dynamique des éléments de mémorisation de la machine (ceux-ci sont en effet affectés durant la compilation.

Nous pensons :

β 1) qu'il est impossible de fabriquer à la main sans erreur une grammaire équivalente à une machine donnée. (l'exemple précédent du Z0001 peut le montrer). C'est pourquoi, nous proposons une méthode automatisable pour le faire.

β2) que l'on peut ajouter à la grammaire ainsi obtenue les règles qui font passer des opérateurs qui sont symboles de bases du langage à compiler (+, -, + ...) à ceux qui existent dans la machine ('OPn' par exemple addition parallèle de 4 bits avec introduction de report).

β3) que l'on peut déduire du réseau structurel la table des éléments de mémorisation effectivement employée dans le chemin de données.

2) Matrice du réseau structurel

Une machine microprogrammée a été souvent représentée sous forme de graphe (LUCONI (3) ,RUTLEDGE (4) , BOULAYE (5)).

Pour que cela ait un sens dans le cas d'une machine réelle il faut définir rigoureusement le passage de la machine (ou de sa description, car la description est la machine) au graphe.

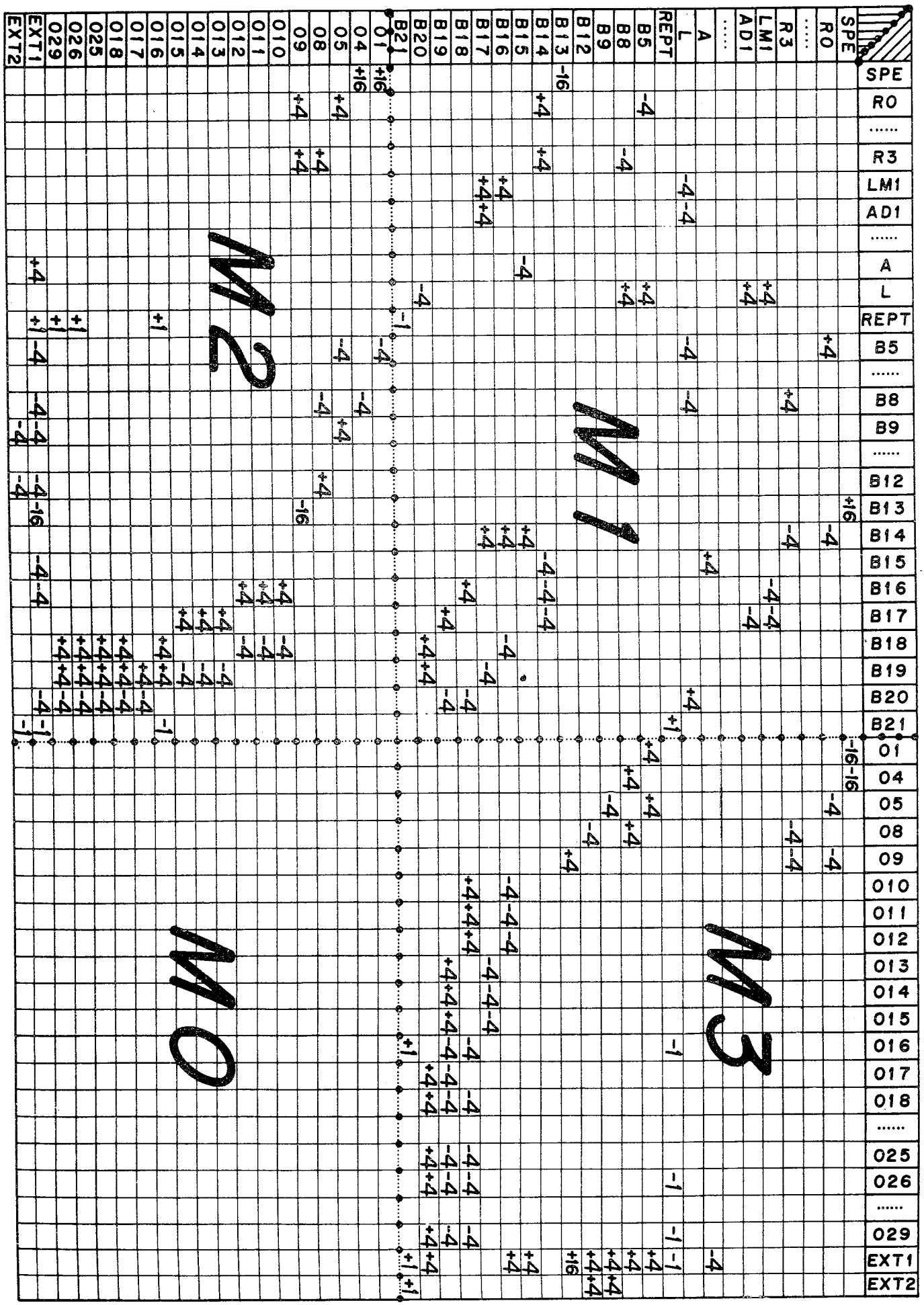
Par ailleurs, la notion de réseau est plus applicable que celle de graphe.

On peut donc associer une matrice (carrée) au réseau structurel. On associe une ligne et une colonne à chaque articulation de type □,⧖,⊗.

A l'intersection de la colonne d'une articulation a1 et de la ligne d'une articulation a2, on met le nombre de bits qui vont de a1 vers a2 (ce nombre peut être négatif ou nul).

Exemple du Z0001. Il y a 12 registres, 17 bus et 29 opérateurs, soit 58 lignes et colonnes. Nous ne les représenterons pas toutes (les cases vides sont en fait remplies de 0).

Matrice du Z0001



Alors, bien sûr, on peut représenter une microinstruction comme sous-matrice de cette matrice, mais on a :

"Une condition nécessaire et suffisante pour qu'une matrice M' de même dimension que M représente une microinstruction,est que l'on ait :

- M'O ne contient que des 0.
- M'2 $\wedge$ M2 contient tous les nombres positifs de M2.
- M'3 $\wedge$ M3 " " négatifs de M3.
- Les colonnes de M' communes à M'1 et M'2 ne contiennent qu'un seul nombre négatif.
- M $\vee$ M' = M. (pour justification voir (6)).

Les matrices associées aux réseaux structurels sont antisymétriques.

3) Codage du micromot :

Nous n'avons fait aucune hypothèse jusqu'alors sur la hiérarchie de contrôle, mais nous savons qu'elle est en contact avec le réseau structurel uniquement aux entrées des bus et des registres.

α) La limitation intrinsèque de simultanéité parmi toutes les actions élémentaires de la machine traitée, s'exprime simplement :

"A un instant donné une seule entrée peut être connectée sur chaque bus".

Nous allons comparer le "parallélisme" (c.a.d. le nombre d'actions élémentaires possibles simultanément) dans des machines de même structure.

-"Le parallélisme maximum d'une machine de structure donnée est obtenu lorsqu'à chaque instant une entrée peut être validée sur chaque bus ou registre".

-"Une condition suffisante de parallélisme maximum est que chaque bus et registre soit contrôlé par un ensemble indépendant de bits du mot de contrôle".

Nous appelerons codage par champs indépendants, ou "de parallélisme maximum" ce codage du micromot.

Exemple du Z0001 :

On va avoir 12 champs de 1 bit pour les registres et mémoires et les champs C5, C6, C7, C8 (de 2 bits) C9, C10, C11, C12, C13 (1 bit), C14 (2 bits), C15, C16, C17 (3 bits), C18, C19, C21 (2bits) et C20 (4 bits) associés au bus.

Dans ce cas le micromot aura donc 46 bits. Or dans Z0001 il y a seulement 27 bits racines de la hiérarchie de contrôle du réseau structurel précédent.

Pour des raisons d'économie de mémoire morte, le codage que nous avons adopté n'est donc pas "de parallélisme maximum".

β) Exemple de réduction du parallélisme

Si on organise R0, R1, R2, R3 comme une mémoire de 4 mots de 4 bits, on peut contrôler les entrées de ces registres par le même champ que le bus B14 sur lequel vont leur sortie. L'économie réalisée dans le nombre de bits du micromot est donc de 4 bits.

Autres exemples de réduction du micromot :

- S'il y a un bus à l'entrée d'un registre et si le nombre des entrées n de ce bus n'est pas une puissance de 2, on peut contrôler l'entrée du registre par l'un des codes inemployés du champ de contrôle du bus (il y en a $2^p > n$). Cette réduction du micromot n'est pas une réduction du parallélisme.

- Si l'on peut charger systématiquement (exemple registre L) un registre on peut supprimer du micromot le signal qui code son entrée.

Ces réductions appliquées à Z0001 sont :

- suppression du contrôle de L;
- codage avec B15 du contrôle de A, avec B21 de celui de REPT. L'économie réalisée est de 3 bits.

γ) Codage de parallélisme minimum :

Ce n'est pas celui qui permet de contrôler une seule action élémentaire à la fois, mais un seul circuit élémentaire à la fois.

- "Un circuit élémentaire est un cycle du réseau structurel qui ne contient qu'un registre".

On peut à l'aide de la matrice précédente calculer tous les circuits élémentaires possibles. Le codage minimum s'il y en a n est celui qui demandera p bits, avec $2^{p-1} < n \leqslant 2^{p}$.

Tout codage d'une machine microprogrammée se situe entre les codages de parallélisme maximum et minimum.

BIBLIOGRAPHIE

1 - J. KUNTZMANN, "Théorie des réseaux", DUNOD, Paris 1972.

2 - F. ANCEAU, P. LIDDELL, J. MERMET, C. PAYAN, "A language to describe Digital Systems, Application to logic Design", C.O.I.N.S. Miami, December 18-20- 1969.

3 - F. LUCONI, "Asynchronous Computational Structures", Thesis M.I.T. Cambridge, Massachusetts, February 1968.

4 - RUTLEDGE, "La microprogrammation", cours de D.E.A. 1967-68. Université de Grenoble

5 - G. BOULAYE, "La microprogrammation", DUNOD 1971.

6 - J. MERMET, "Définition du Langage CASSANDRE", Thèse de Docteur Ingénieur, Université de Grenoble, Mars 1970.

CONCLUSION

Nous pouvons faire le bilan de six années d'étude mais aussi de trois ans d'utilisation chez nous, ou dans diverses entreprises, du langage et du système CASSANDRE.

La conception assistée par calculateur est une discipline qui s'est développée, bien que l'utilisation de ses premiers résultats ne soit pas encore entrée dans les moeurs (il faudra encore plusieurs années pour cela).

Il semble que les systèmes les plus évolués s'articulent maintenant autour d'une structure interne de données définie au préalable pour stocker facilement les informations relatives au problème considéré et simplifier au maximum les traitements qui peuvent les utiliser.

C'est la démarche que nous avons suivie lorsque le système CASSANDRE a été conçu autour d'une bibliothèque d'unités, interne au système, mais accessible par tous les programmes d'utilisation. Nous sommes même allés plus loin en programmant des applications dont le résultat lui même est décrit en CASSANDRE (compilateur de hardware) et donc peut servir d'entrée à d'autres applications.

Mais, si le système était à refaire, un plus grand effort devrait être consacré à la structuration interne des données CASSANDRE pour faciliter la programmation des nouvelles applications (qui est assez délicate dans l'état actuel).

Par ailleurs un travail est en cours pour la réalisation d'une version asynchrone et très rapide du système de simulation. Ce système n'acceptera que des textes simples (en CASSANDRE-E) mais pourra servir à la mise au point de séquences de tests de circuits logiques. Il permettra aussi de traiter les circuits existant chez de nombreux utilisateurs (et qui n'ont pas été conçus à partir d'une description en CASSANDRE).

Un point important consiste à rendre opérationnelles les fonctions qui assurent la production automatique de la documentation d'un projet (texte et schémas); nous avons montré à la fin de la partie C ce qu'il pourrait en être. Cette fonction reste l'une des plus coûteuses et des moins bien résolues chez beaucoup d'industriels.

Nous avons mis un accent particulier sur CASSANDRE-langage-de-microprogrammation. C'est en effet le point sur lequel notre étude s'est révélée la plus riche en applications aussi bien dans l'aide à la conception de machines microprogrammées, que la simulation de microprogrammes où leur compilation sur une machine donnée.

Enfin l'utilisation de CASSANDRE pour l'enseignement de la structure des machines digitales et la formalisation des notions du hardware doit demeurer un objectif prioritaire. C'est d'ailleurs l'un des domaines où nos réalisations se comparent favorablement aux plus intéressantes autres tentatives (CASSANDRE a déjà été installé dans trois universités étrangères).

Nous avons essayé d'utiliser le formalisme des réseaux d'articulations pour expliciter certaines opérations qui constituent le "modelage des systèmes logiques". Ce formalisme nous semble prometteur mais il faut savoir conclure... et céder la place à plus compétent. Exprimons le voeux que les algébristes développent quelque théorie intéressante à partir de nos réflexions.

-o-o-o-o-o-o-o-o-

BIBLIOGRAPHIE GENERALE

-o-o-o-o-o-o-o-o-o-o-

1 - ANCEAU F. "La Compilation de microprogrammes", Séminaire de logique, mars 1969.

2 - ANCEAU F., LIDDELL P., MERMET J., PAYAN C., "CASSANDRE : Conception assistée des systèmes digitaux", Onde électrique. Numéro Spécial, janvier 1969.

3 - ANCEAU F., LIDDELL P., MERMET J., PAYAN C., "A Language to Describe Digital Systems, Application to logic Design", C.O.I.N.S, Miami, december 18-20 1969.

4 -ANCEAU F., COUTURIER, DOUSSY J., PERRON F., "Compilation et simulation du langage CASSANDRE", chap II, rapport final de contrat CRI, Industrialisation de CASSANDRE.

5 - AURIAUX G., MERMET J., "Etude d'un macrogénérateur", projet de fin d'études, 1966.

6 - BARLOW, "Simulation as a design tool, "in proc. of Design Automation Seminar, May 64, sponsored by MESA Scientific Corp, Inglewood. Calf., 90303, pp 164-192.

7 - BOGO G., GUYOT A., LUX A., MERMET J., PAYAN C., "CASSANDRE and the computer aided logical Systems design", Congrès de l'IFIP, 23-28 août 1971.

8 - BOULAYE G., "La microprogrammation", DUNOD 1971.

9 - BOULAYE G., MERMET J., Editors, "Microprogramming", Hermann, 480 pages, 1972.

10 - BOUCHET A., PAYAN C., "Synthèse des automates en fonctions croissantes. Immersion d'ordonnés, Séminaire de Logique Grenoble juin 1968.

11 - BOULAYE G., "Logique et organes des calculatrices", DUNOD-UNIVERSITE 1970.

12 - BREUER M.A., "Général Survey of design automation of digital computers", proc of IEEE vol 54, n° 12, décembre 1966.

13 - BREUER M.A., "Techniques for the simulation of computer logic", Commun. ACM, pp 443-446, july 1964.

14 - BREUER M.A., "Design Automation of Digital Systems" Prentice Hall 1972.

15 - BROZOWSKI J.A., "Derivation of regular expressions", J. Assoc. Com. March. 11, 481-494, 1964.

16 - BURNETT C.J., "A Design Language for Digital Systems", Masters Thesis MIT août 1965.

17 - Mc CLURE R.M., "A programming language for simulating digital systems", J.A.C.M., vol 12, pp. 14-22 january 1965.

18 - CHU Y., "An Algol-like computer design language", Communications of the ACM, vol. 8 n° 10, october 1965.

19 - CHU Y., "Introduction to computer organization and programming", Prentice Hall 1970.

20 - DARONDEAU P., MERMET J., "Projet d'étude d'un minicalculateur Microprogrammé, entièrement compatible avec le 360, réalisable avec les moyens minimaux", chap. 1 rapport final du contrat C.R.I., Industrialisation de CASSANDRE.

21 - DAVID, FANTINO, MENARD, "Outils pour le découpage en module, l'implantation et la documentation automatique de machines digitales décrites en CASSANDRE", rapport interne, ENSIMAG 1973.

22 - DULEY J.R., DIETMEYER D.L., "A Digital system design language DDL" IEEE trans-comput vol 17, n° 9 septembre 1969.

23 - DULEY J.R.; DIETMEYER D.L., "Translation of a DDL digital system specification to boolean equations" IEEE, vol C18, n° 4 Avril 1969.

24 - ESTRIN G. MANDELL R., "Metacompiler as a design automation tool" 1966 Proc. Share Design Automation Conference.

25 - FALKOFF A.D., IVERSON K.E., SUSSENGUTH E.H., "Formal description of system/360", IBM Sys. J., vol 3, pp. 198-262, 1964.

26 - FARRENY, "Edition automatique de schémas logiques", rapport final du contrat CRI, 70016.

27 - FRIEDMAN T.D., "Alert : A program to produce logic design from preliminary machine descriptions", RC 1578, march 24 1966, IBM Research.

28 - GAVRILOV, "Langage LIAPAS permettant de traiter les algorithmes de synthèse des structures logiques", Editeur NAOUKA, Moscou 1966.

29 - GERACE G.B., GESTRI G., "A method for Designing a digital system as a sequential network system", Université de Pise, juin 1967.

30 - GIESE A., "Hargol - A hardware oriented algol language", Internal report n° VA5, août 1966, A/S Regnecentralen Copenhague, Danemark.

31 - GIESE A., "Formal definition of Hargol", Regnecentralen Falhoneralle 1 Copenhague (rapport interne) février 1969.

32 - GORMAN D.F., ANDERSON J.P., "A Logic design translator", 1962 Proc. FJCC pp. 86-96.

33 - GORMAN D.F., "A system descriptive language and its uses", ph D. Université de Pensylvanie, avril 1968.

34 - GUERIN, LAGOUTIERE, "Hiérarchie de mémoire utilisant une technologie à domaines magnétiques", projet de fin d'études à l'ENSIMAG, juin 1972.

35 - IVERSON K.E., "A programming language", J. Wiley and Sons. Inc. N.Y. London 1962.

36 - KNUTH D.E., Mc NELEY J.L., "Sol-A symbolic Language for general-purpose systems simulation", IEEE Trans. on Electronic Computers, vol. EC-13, pp. 401-408, august 1964.

37 - KUNTZMANN J., "Algèbre de BOOLE", DUNOD.

38 - KUNTZMANN J., "Théorie des réseaux", DUNOD, Paris 1972.

39 - LIDDELL P., "Découpage syntaxique de systèmes logiques décrits en CASSANDRE, thèse 3e cycle, mars 1970.

40 - LE FAOU C., "Problèmes d'analyse numérique rencontrés dans le programme IMAG 2", séminaire d'analyse numérique, Université de Grenoble, Mathématiques Appliquées, février 1972.

41 - LITAIZE D., "Tracé automatique de Plaquettes de circuits imprimés en Technologie Multicouche Haute Densité", rapport final contrat C.R.I. n° 70015, Octobre 1972.

42 - LUCONI F., "Asynchronous Computational Structures", Thesis M.I.T. Cambridge, Massachusetts, février 1968.

43 - LUSTMAN, "Langages exprimant le fonctionnement d'un système" rapport DGRST Oct 1967.

44 - LUSTMAN, "Simulation d'une machine digitale à partir d'une description en langage CASSANDRE", revue bleue de l'AFIRO, B-2, 1969.

45 - LUSTMAN, MERMET J., "CASSANDRE - Un langage de description de machines digitales", revue de l'AFIRO n° 15 1969.

46 - LUX A., MERMET J., "Utilisation du système CASSANDRE pour l'enseignement de la structure des calculateurs et de la microprogrammation", congrès de Salzbourg (à paraître), avril 1972.

47 - MERMET J., "Description formelle d'un langage orienté vers le hardware", colloque sur les calculateurs embarqués sur fusées et satellites, 3-6 décembre 1968 CNES Paris.

48 - MERMET J., "Le langage CASSANDRE", rapport DGRST, contrat 66 00 069, 31 mars 1968.

49 - MERMET J., "Synthèse de fonctions booléennes par des opérateurs en ligne", Bull-Inst Polytechnique de JASSY, tome XIII 1967, Fasc. 1-2.

50 - MERMET J. "Microprogrammation", cours à l'ecole de printemps de la DRME, 19-23 avril 1971.

51 - MERMET J., "Définition du langage CASSANDRE", thèse de Docteur Ingénieur, Université de Grenoble, mars 1970.

52 - MESZTENYI C.K., "Translator and simulator for Computer design and simulation", Program (CDSP) version I, University of Maryland, Computer Science Center, College Park, Maryland.

53 - METZE G., SESHU S., "A proposal for a computer compiler", 1966 Proc. SJCC, pp 253-264.

54 - MKRTCHYAN S.O., "Model of formal neuron with union fibers" Engineering cubernetics n° 3, 1970.

55 - NYGAARD K., DAHL O.J., "Simula, an Algol, Based-Simulation language", communication of the ACM vol 9, n° 9 sept. 1966.

56 - PARNAS D.L., "A language for describing the functions of synchronous systems", communication of the ACM, vol 9 n° 2 février 1966.

57 - PARNAS D.L., DARRINGER J.A., "Sodas and methodology for system design", Fall joint computer conference, 1967.

58 - PAYAN C., "Description et synthèse des automates-Notion "dès que" "Colloque international systèmes logiques, conception et applications, Bruxelles sept 1969.

59 - PAYAN C., "Langage de description de systèmes séquentiels", chap. V, rapport final du contrat C.R.I. "Industrialisation de CASSANDRE".

60 - PAULIAT P., "Mémoire vive bipolaire", rapport de stage à la Société TELEMECANIQUE du 3 mai au 30 juin 1971.

61 - De POLIGNAC, "Utilisation du langage CASSANDRE pour la conception des machines microprogrammées", thèse de 3e cycle 1973.

62 - PROCTOR R., "A logic design translator experiment demonstrating relationship of language to systems and logic design", IEEE Trans. on Electronic Computers, col. EC-13, pp. 422-430, august 1964.

63 - RAYNAUD-GAROCHE F., "Treillis des écoulements dans un graphe orienté", séminaire d'algèbre appliquée et conception de l'IMAG, séance du 26 avril 1971.

64 - ROTH J.P., "Hardware implementation of microprograms", IBM Rapport interne, RC 1456 août 1965.

65 - ROTH J.P., BOURICIUS W.G., SCHNEIDER P.R., "Programmed algorithms to computer tests to detect and distinguish between failures in logic circuits", IEEE Trans on electronic computers, col EC-16, n° 5 october 1967.

66 - RUTLEDGE, "La microprogrammation", cours de D.E.A. 1967-68, Université de Grenoble.

67 - RUTMAN, "Logik a syntax-directed compiler for computer BIT-TIME simulation", M.S. thesis University of California, Los Angeles, august 1964.

68 - SAILLARD J.C., LUSINCHI J.P., "Programme conversationnel pour l'aide à l'implantation de circuits intégrés", revue de microélectronique avancée.

69 - SARRET M;, "Problèmes d'implantation, étude de la planérité pour les réseaux électroniques", contrat DRME, 6634215/6544.

70 - SAUCIER G., "Codage des états internes de systèmes séquentiels asynchrones", thèse 3e cycle, Université de Grenoble.

71 - SCHLAEPPI H.P., "A formal language describing machine logic, timing and sequencing (LOTIS)", IEEE Trans. on electronic computers, vol EC-13, pp. 439-448, august 1964.

72 - SCHORR H., "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.

73 - SEMPE B., "Conception et réalisation d'un calculateur industriel", Faculté de GRENOBLE, thèse de Docteur Ingénieur.

74 - SMITH D.L., "Models and data structures for digital logic simulation", MIT, Projet MAC, août 1966.

75 - SOKOLOFF M., "Tentatives d'emploi du produit tensoriel généralisé en langage programme iversion pour la compilation automatique des schémas logiques", colloque d'algèbre de Boole, Grenoble 1965.

76 - SRINIVASAN C.V., "An introduction to CDLI - A computer description language", RCA Lab N.Y. contrat ,° AF 319 (628) 4789, proj. 5682 Task n° 563202, sept 1967 report n° 2 octobre 1967.

77 - STRACKEY C., "A general Purpose Macrogenerator", computer journan, vol 8 n° 3 octobre 1965.

78 - Table ronde sur les langages pour la conception aidée des systèmes logiques. Laboratoire de Mathématiques Appliquées, St-Martin-d'Hères. 9-10 octobre 1967.

79 - TEICHROEW D., LUBIN J.F., "Computer simulation - Discussion of the technique and comparaison languages", communications of the ACM vol 9 n° 10 octobre 1966.

80 - TISON P., "Etude des systèmes logiques", rapport de stage aux Etats Unis, 16-11-1966.

81 - VINCENT-CARREFOUR, "Utilisation d'un calculateur pour la définition des ensembles logiques", l'onde électrique, janvier 1968.

82 - WILBER J.A., "A language for describing digital computer", report n° 197, Dept of com. Science, University of Illinois, feb. 15, 1966.

83 - WILKES M.V., "The best way to design an automatic calculation machine", Manchester University Computer Inaugural Conference, p 16 1951.

84 - ZUCKER M.S., "Locs : an EDP machine logic and control simulator", 1965 IEEE Conv. Rec, pt 3, pp. 28-51.

85 - ZURCHER F.W., RANDELL B., "Iterative multi-level modelling - A methodology for computer system design", RC 1938, novembre 10, 1967, IBM research.