Un compilateur de microprogrammes

Alain Guyot

THESE présentée à

UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

POUR OBTENIR LE GRADE DE
DOCTEUR DE 3ème CYCLE
INFORMATIQUE

Alain GUYOT

Un Compilateur de Microprogrammes

Soutenue le 25 octobre 1975 devant la Commission d'Examen :

Président : Monsieur L.BOLLIET

Examinateurs : Monsieur F. ANCEAU
Monsieur J. MERMET
Monsieur J.C. SYRE

UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE		INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE
M. Michel SOUTIF	Présidents	M. Louis NEEL
M. Gabriel CAU	Vice-Présidents	MM. Lucien BONNETAIN Jean BENOIT

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

PROFESSEURS TITULAIRES

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

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

PROFESSEURS ASSOCIES

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

PROFESSEURS SANS CHAIRE

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

MAITRES DE CONFERENCES ET MAITRES DE CONFERENCES AGREGES

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

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

MAITRES DE CONFERENCES ASSOCIES

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

CHARGES DE FONCTIONS DE MAITRES DE CONFERENCES

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

Fait à Saint Martin d'Hères, OCTOBRE 1974.

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

PROFESSEURS TITULAIRES

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

PROFESSEUR ASSOCIE

M.	BOUDOURIS Georges	Radioélectricité

PROFESSEURS SANS CHAIRE

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

MAITRES DE CONFERENCES

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

MAITRE DE CONFERENCES ASSOCIE

M.	LANDAU Ioan Doré	Automatique

CHARGE DE FONCTIONS DE MAITRES DE CONFERENCES

M.	ANCEAU François	Mathématiques appliquées

Fait à Saint Martin d'Hères JANVIER 1974

Un Compilateur

de Microprogrammes

TABLE DES MATIERES

Je suis très sensible à l'honneur que me fait Monsieur L. BOLLIET, Professeur à l'Université Scientifique et Médicale de Grenoble en acceptant de présider le Jury. Qu'il veuille trouver ici témoignage de ma reconnaissance.

Monsieur J. MERMET a guidé ma recherche de ses conseils précieux et en a suivi le développement. Je l'en remercie très vivement.

Je dois beaucoup à Monsieur F. ANCEAU pour ses encouragements et ses idées audacieuses.

Ma gratitude va également à Monsieur J. C. SYRE qui m'a aidé dans ma rédaction.

Je tiens à remercier aussi mes camarades de "l'équipe CASSANDRE" pour leur aide amicale. Nous faisions tous un travail semblable : "truffer" un suiveur syntaxique de fonctions sémantiques.

Je ne saurais oublier dans mes remerciements Madame DIAZ pour sa frappe précise ni le Service de Reprographie pour son amabilité et sa grande compétence.

Le terme de "microprogrammation" a été introduit par WILKES en 1951 [54] pour désigner une méthode permettant de simplifier et de systématiser la conception des calculateurs. Cette méthode fit ses preuves à l'Université de MANCHESTER (U.K.) dans la définition de l'EDSAC II.

Avec l'apparition, dix ans plus tard, de mémoires de contrôle rapides, fiables et bon marché, la microprogrammation devint une technique de fabrication de plus en plus utilisée. Cela entraînera l'écriture de microprogrammes gros et complexes et des perfectionnements éloignant la microprogrammation de la définition de WILKES.

Il est vraisemblalble que les raisons économiques qui assurent son succès actuel se maintiendront de longues années encore. Actuellement, seuls les microprocesseurs, dont la majorité est encore à contrôle câblé, peuvent constituer une menace potentielle à ce succès.

Il serait trop long d'énumérer ici les perfectionnements que la programmation a fournis à la microprogrammation et qui tendent à faire de cette dernière une programmation particulière.

En général, les microprogrammes demandent à être beaucoup plus optimisés, au double point de vue de l'encombrement et de la durée d'exécution, que les programmes, ces derniers étant plus faciles à écrire.

Or, d'une part, très rapidement les ingénieurs chargés de concevoir de nouvelles lignées d'ordinateurs se sont aidés des anciens et, d'autre part, on a chargé l'ordinateur des parties automatiques et fastidieuses de l'écriture de programmes.

Il est donc normal que les diverses aides développées,citons pêle-mêle simulateur, émulateur, assembleur, compilateur... soient adaptées à l'écriture de microprogrammes. Cependant, pour être rentables, ces aides nécessitent une évaluation précise et prudente des besoins. Il faut en effet, que les parties automatiques de l'écriture soient prises en charge par un programme et que reste possible l'astuce qui fait gagner à la fois du code et du temps d'exécution.

PREMIER CHAPITRE

oooooooooooooooo

La conception du compilateur de microprogrammes a été guidée par le schéma suivant:

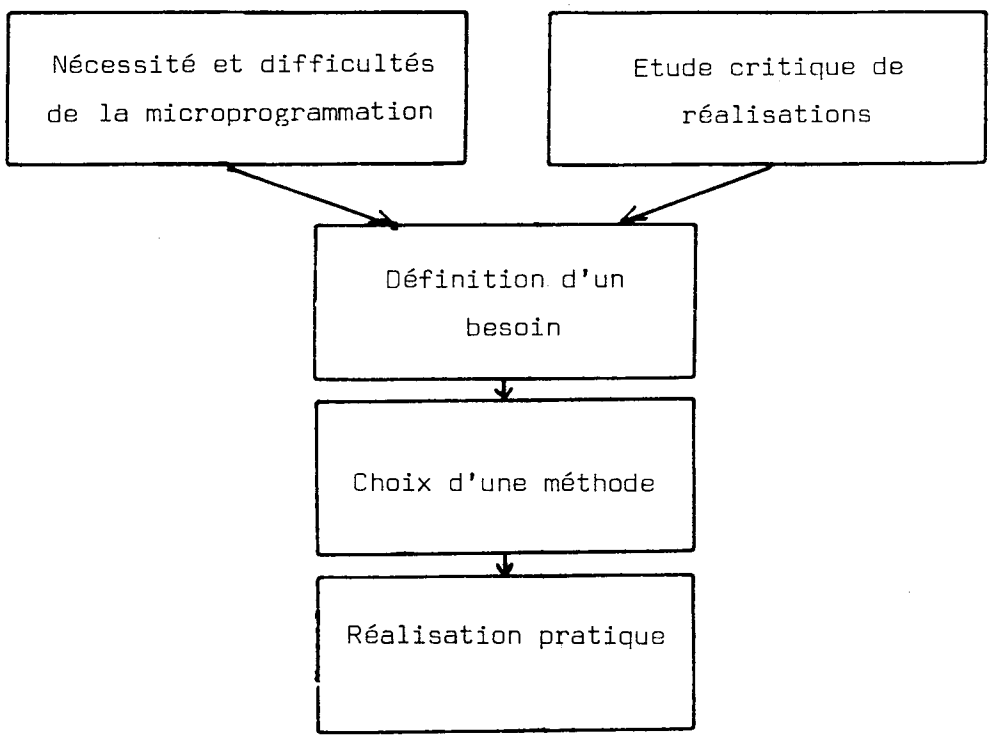


figure 1.1.

Ce schéma est simplifié, de nombreux liens n'y figurent pas. Par exemple, l'essai de la méthode sur des problèmes concrets et sa comparaison avec des méthodes existantes. Il y manque également les liens avec les problèmes connexes comme la définition et la simulation de la machine microprogrammée.

Enfin, il y manque surtout les inévitables bouclages entre la définition et la réalisation qui font de ce schéma un graphe complet.

Un programme ayant été écrit, la partie réalisation de ce schéma est à elle seule plus importante que tout le reste et sera décrite par un autre organigramme.

1.- UNE DEFINITION DE LA MICROPROGRAMMATION

Les définitions de la microprogrammation sont aussi nombreuses que variées. Cela peut être causé par la volonté de faire de la microprogrammation un concept alors que ce n'est probablement qu'une technique.

La définition donnée par WILKES peut se résumer grâce à la figure suivante:

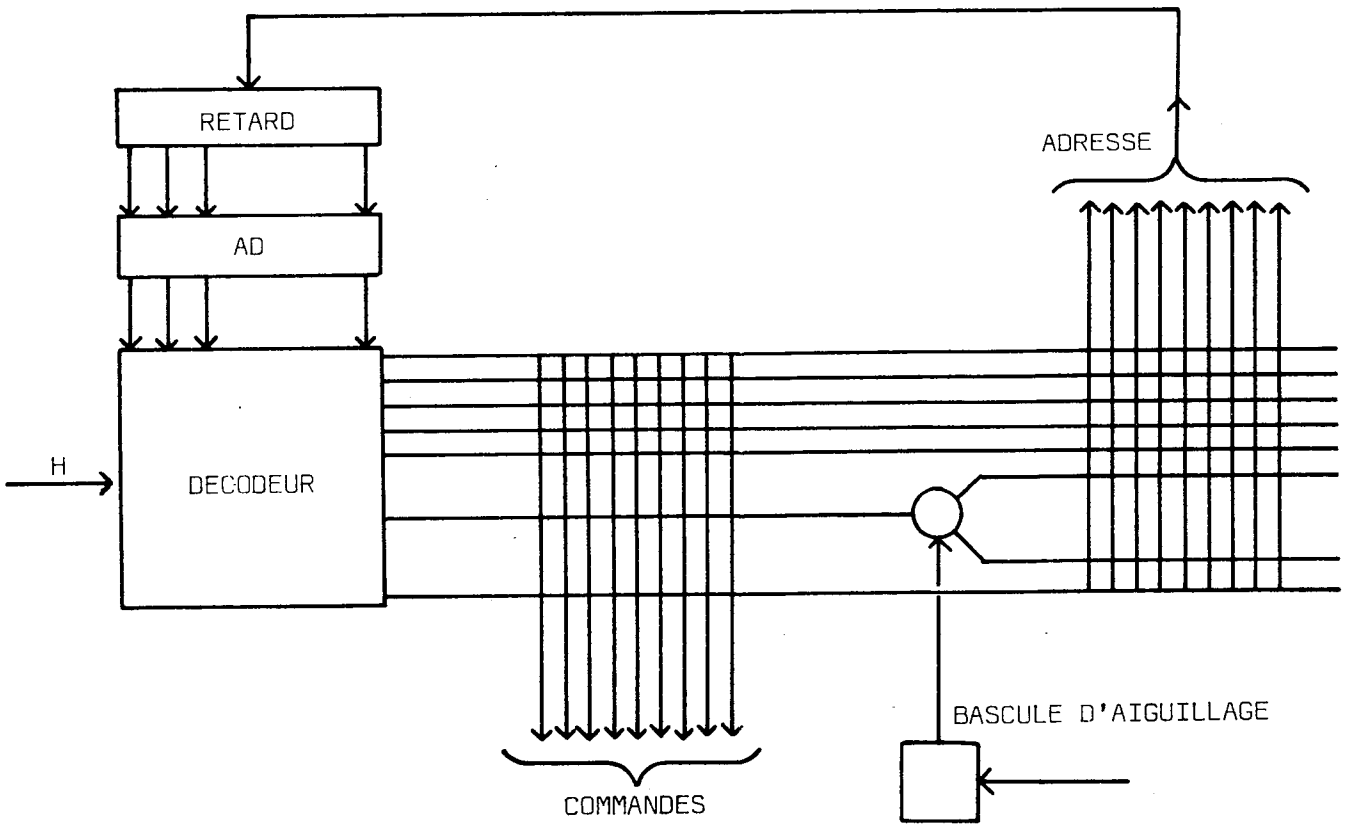


figure 1.2.

où l'adresse de la micro-instruction en cours d'exécution se trouve dans le registre AD et a pour effet d'aiguiller par un décodeur l'impulsion d'horloge H sur l'une des sorties de ce dernier.

Le schéma de WILKES montre également un dispositif permettant de choisir l'adresse de la micro-instruction suivante en fonction de la valeur d'une bascule d'aiguillage. On réalise ainsi un "branchement conditionnel".

Pour dériver des techniques existantes,nous adopterons une définition de la microprogrammation qui tend à faire de celle-ci une programmation particulière. Cette définition est liée à l'existence, dans un ordinateur, d'un mini-ordinateur, souvent appelé micromachine, et qui exécute des micro-instructions. La structure apparente des ordinateurs, canaux, unités de contrôle etc... n'a souvent rien à voir avec la structure de la micromachine plus ou moins universelle ou spécialisée qui les réalise.

La micromachine est physiquement réalisée. Elle se décompose classiquement, de façon plus ou moins naturelle ou arbitraire, en une partie "opérative", destinée à traiter des "données" et une partie "contrôle" qui commande la précédente.

La partie contrôle de ces micromachines comprend une mémoire de micro-instructions (à lecture seule), souvent totalement distincte de la mémoire de données, un registre d'adresse de cette mémoire, ou compteur ordinal de la micromachine, un séquenceur plus ou moins complexe, assurant la mise à jour de ce compteur ordinal, et enfin, un décodeur de micro-instructions faisant interface entre la partie opérative et la partie contrôle de la micromachine. Un automate est souvent chargé de définir une succession d'instants élémentaires dans l'exécution d'une micro-instruction. Par convention, cet automate fait alors partie du contrôle.

La séparation physique entre séquenceur et partie opérative, si elle existe, est souvent mise à profit pour exécuter des micro-instructions qui

sont simultanément opératives et de branchement. Le gain de vitesse d'exécution est appréciable pour des algorithmes ayant un organigramme "touffu", c'est-à-dire comportant beaucoup de branchements.

Nous conviendrons que le décodeur de micro-instructions àppartient à la partie opérative dont il commande combinatoirement les portes logiques et les registres.

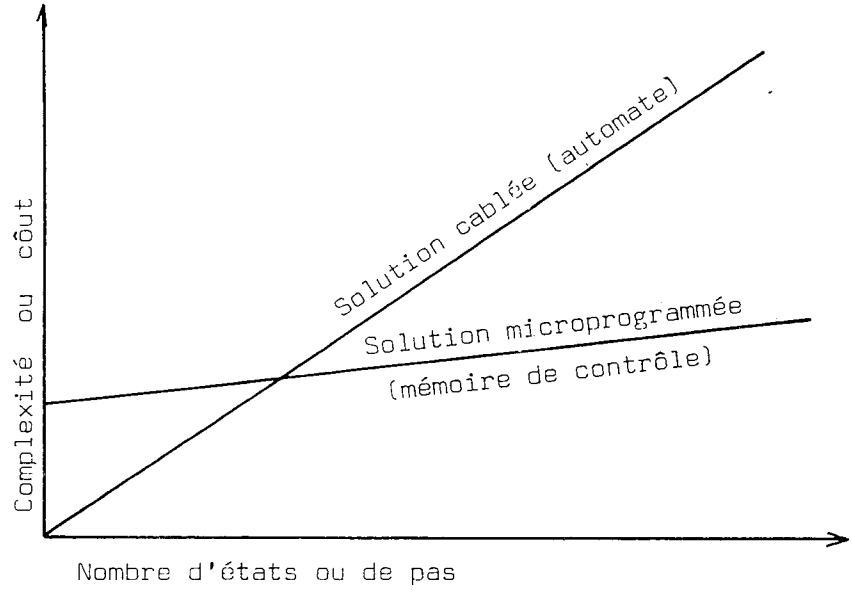
Ce décodeur ne doit pas comporter d'éléments de mémorisation.

Selon l'usage, nous appellerons "réalisation particulière d'une macro-machine" l'union indivisible d'une micromachine et d'un microprogramme. L'ensemble des différentes réalisations d'une macromachine constitue une série (exemple: la série 360).

2.- NECESSITE DE LA MICROPROGRAMMATION

2.1. Pour un constructeur de machine

Les études de fabricants de composants (circuits intégrés) et de constructeurs de machines, qui assemblent ces derniers, s'accordent sur l'économie de conception et de production des machines microprogrammées.



Appréciation du rapport nombre-de-pas/côut dans les parties contrôle.

Ce graphe se justifie de la façon suivante:

La complexité de la solution câblée croît avec le nombre d'états et cette complexité est faible lorsque le nombre d'états est petit. La solution microprogrammée, par contre, demande au départ une mémoire de contrôle et un dispositif de séquencement. Mais ajouter un état revient à ajouter un mot dans la mémoire de contrôle, peu chère.

Pour des machines rapides (peu de pas d'exécution), la solution câblée garde ses avantages et la solution microprogrammée est intéressante pour des machines ayant un jeu d'instructions riche et comportant peu de matériel, tout en conservant une grande souplesse d'utilisation.

2.1.1. Jeu d'instructions riche

Un programme réalisant un algorithme fixé tient d'autant moins de place en mémoire et, dans un moindre degré, s'exécute d'autant plus rapidement qu'il est écrit avec des instructions puissantes et bien adaptées.

Or le coût d'exécution d'un programme est fonction de la taille de la mémoire demandée et du temps d'exécution. La mémoire est souvent plus chère que l'unité de traitement.

Mais la complexité de l'unité de traitement va augmenter avec la complexité du jeu d'instructions, et ces instructions doivent avoir un format permettant de les distinguer toutes.

2.1.2. Peu de matériel

Les fabricants intègrent de plus en plus de portes logiques dans leurs circuits, ceux-ci réalisent des fonctions plus nombreuses et plus complexes. Le coût unitaire de la fonction réalisée s'abaisse et une certaine standardisation des structures de micromachines de puissance comparable apparaît, causée par la taille croissante des macrocomposants.

Cependant, le grand nombre de fonctions réalisées par chaque circuit exige pour ce dernier un grand nombre de commandes, c'est-à-dire de bits. Certaines configurations binaires de ces commandes correspondent à des opérations farfelues et ne sont jamais utilisées. Ce grand nombre de commandes favorise le contrôle microprogrammé ou le décodage par mémoire morte.

2.1.3. Souplesse de la microprogrammation

La microprogrammation rend largement indépendante la structure apparente d'une machine, qui se réduit à un jeu d'instructions, de la structure de la micromachine qui l'a réalisée, par interposition d'un niveau intermédiaire qui est le microprogramme. Ce niveau permet d'adapter la richesse et la souplesse des instructions à la pauvreté et à la rigidité du matériel.

Les constructeurs peuvent ainsi proposer des modèles différents de la même macromachine constituant une gamme, ou série, et différent dans leur réalisation, prix et performances, et cependant semblable dans l'exécution des instructions.

La microprogrammation permet parfois d'adjoindre facilement des fonctions à la liste de celles réalisées par une macromachine. Cette possibilité est exploitée pour émuler, pour réaliser des fonctions de canal ou de coupleurs (canaux intégrés) et enfin pour contrôler la micromachine grâce à des microprogrammes de test.

Pour ces dernières applications, il est intéressant de faire de la multi-micro-programmation, et donc de doter la micromachine d'un mécanisme, généralement câblé, d'interruption, de synchronisation et de sauvegarde de contexte, permettant de gérer des processus.

2.2. Pour un utilisateur de machine

La microprogrammation est un argument publicitaire cité dans tous les dépliants. Malheureusement cette microprogrammation reste en général totalement hors de portée de l'utilisateur.

Certaines micromachines sont très adaptées à leur usage et leur réutilisation à d'autres fins est délicate.

Exceptionnellement, l'utilisateur peut définir certaines fonctions spécifiques qui seront réalisées par microprogramme sous le contrôle du constructeur. Mieux, l'utilisateur peut obtenir un assembleur et un simulateur sous sa responsabilité. Si la structure de la micromachine le permet, ceux-ci seront simplement ajoutés à ceux livrés par le constructeur.

Cependant, la modification par un utilisateur du microprogramme livré par un constructeur entraîne au moins deux difficultés.

La plus lourde est d'avoir à modifier le code instruction de la machine, car pour ajouter des fonctions, il est parfois obligatoire d'en supprimer d'autres. Il faut alors modifier en conséquence tout le système de programmes livré avec la machine, qui ne pourra plus bénéficier d'améliorations de versions ultérieures de ce système.

En général, certains codes d'instructions sont volontairement et prudemment laissés inemployés par le constructeur en vue d'enrichissements ultérieurs.

La seconde est la perte de fiabilité. En effet, contrairement aux systèmes de programmes, les microprogrammes livrés par un constructeur sont en général absolument sûrs, c'est-à-dire sans aucune erreur. Toute modification du microprogramme fait évidemment tomber cette garantie.

3.- DIFFICULTE DE LA MICROPROGRAMMATION

L'outil d'aide à l'écriture de microprogrammes le plus utilisé étant certainement l'assembleur, il est intéressant de comparer les difficultés d'écriture d'un microprogramme et d'un programme à ce niveau (assembleur), ce qui nous entraînera à en comparer les avantages (toujours au même niveau).

3.1. Causes de difficulté

Les causes de difficulté que nous allons énumérer peuvent se résumer en recherche contradictoire de vitesse et d'économie.

- Manque d'efficacité des micro-instructions, primitives ou proches du matériel.
- Défaut de clarté et de symétrie des interconnexions de matériel, minimisé pour coûter moins cher.
- Contraintes dues au codage, souvent dans le cas dit "à mot court".
- Manque de place pour les variables et pour les micro-instructions. Chaque variable coûte un registre rapide et chaque micro-instruction, un mot de mémoire morte.
- Particularité du séquenceur qui rend délicate l'implantation et l'adressage de la mémoire de contrôle.

Mais en général, sur une machine à vocation universelle, on peut programmer un simulateur de la micromachine qui la réalise. Sauf applications particulières, programme et microprogramme sont comparables en ce sens que tout algorithme qui peut être microprogrammé peut être programmé et réciproquement.

3.1.1. Avantage du programme sur le microprogramme

L'avantage le plus immédiat est la facilité d'écriture accrue par l'écriture de compilateurs et de langages de haut niveau adaptés à des problèmes particuliers. Pour ne considérer que le niveau assembleur, il est beaucoup plus facile d'écrire des instructions que des micro-instructions.(On compte une instruction par heure et par personne pour un programme analysé, codé et complètement vérifié ; pour le MITRA 15, on aurait écrit une micro-instruction par jour, ce qui semble peu).

Comme corollaire, la facilité de test et de mise au point dûe à l'existence de protection contre les instructions anormales, qui n'existent pas contre les micro-instructions, et la beaucoup plus grande facilité de modification d'un programme comparé à un microprogramme.

En fait, les instructions sont étudiées en vue de faciliter leur utilisation, les micro-instructions en vue d'optimiser leur exécution.

Il semblerait que l'inaccessibilité de la micromachine soit continentale, voire nationale. Beaucoup de mini-ordinateurs outre atlantique sont conçus de façon modulaire, la mémoire de contrôle est extensible et des programmes d'aide à l'écriture de microprogrammes mis à la disposition des utilisateurs.

Un autre avantage, plus sérieux, du programme est la souplesse du support. Contrairement au microprogramme, un programme n'occupe de la place en mémoire centrale (chère) que pendant son exécution. Le reste du temps il est sur un support très bon marché. Cet avantage peut être annulé par l'utilisation de mémoires de contrôle dynamique.

3.1.2. Avantage du microprogramme sur le programme

En premier lieu, on citera toujours la vitesse, sans toujours expliquer pourquoi. En transcrivant un programme en microprogramme de façon que le comportement, par exemple de la mémoire, soit exactement le même, fera gagner du temps, mais on peut gagner beaucoup plus:

- En limitant l'utilisation de la mémoire de masse, toujours lente. Les registres rapides de la micromachine, accessibles par microprogrammes, ne le sont en général pas au programmeur, ils logeront les variables les plus utilisées. (Hiérarchie de mémoires)
- En utilisant le parallélisme dont les possibilités, pourtant réelles, sont complètement cachées au programmeur.

Les autres avantages sont:

- Débarrasser la mémoire vive de programmes souvent exécutés, donc résidents (fonctions du superviseur par exemple).
- Accroître la sécurité car un microprogramme en mémoire morte est pratiquement indestructible et plus difficile à espionner.

4.- BESOIN D'AIDE A L'ECRITURE DE MICROPROGRAMMES

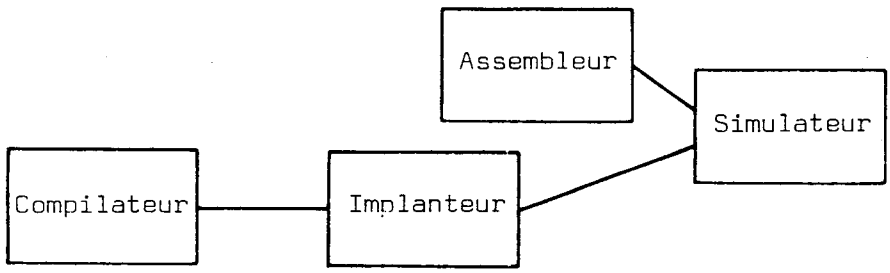
La microprogrammation ayant actuellement tendance à se généraliser et présentant des difficultés certaines, il s'est avéré rentable de développer des outils d'aide à l'écriture de microprogrammes de plus en plus puissants.

Malheureusement ces outils, souvent spécifiques, sont révélateurs de détails sur la structure interne de la micromachine auxquels ils correspondent et sur lesquels les constructeurs aiment à garder une certaine discrétion.

Nous nous bornerons donc à ne citer que quelques réalisations françaises, suffisamment anciennes pour que le secret soit tombé.

5.- EXEMPLES DE REALISATIONS

5.1. Le MITRA 15 CII

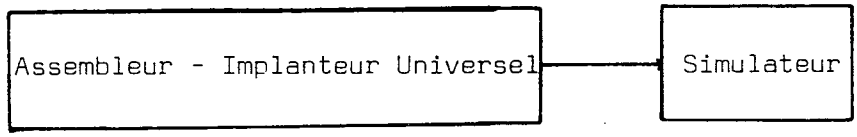


Deux voies ont été utilisées:

- Un assembleur-simulateur, écrit par la CII. Le langage source est en symbolique, et traduit en une seule passe, et l'implantation en mémoire est faite par le programmeur. L'assembleur et le simulateur détectent toutes les violations des contraintes imposées par la micromachine[32].

- Un compilateur-implanteur, défini à l'IRIA. Le langage utilisé est du niveau ALGOL, mais permet de plus l'utilisation de Macros. Ce compilateur est lié à la structure interne du MITRA 15 (exécutifs).

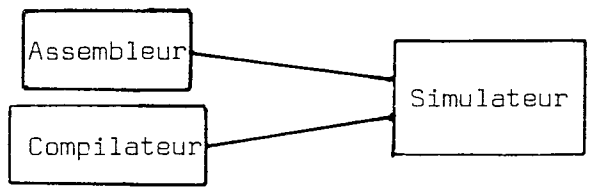
5.2. Le T 1600 TELEMECANIQUE



Entre autres produits d'aide, la TELEMECANIQUE a développé un Assembleur-Implanteur universel, qui utilise une description des micro-instructions sous forme de listes, une description symbolique du langage source et du code à générer, le tout de façon souple. Moins convaincante est l'implantation, mais à notre connaissance, c'est le seul implanteur universel. Le simulateur est propre au T 1600.

L'actuel développement des qualités d'universalité de ce produit assurerait vraisemblablement son succès s'il était diffusé.

5.3. Le MULTI 8 Intertechnique (microdata)



Le micro-assembleur, écrit en FORTRAN, pour être facilement transportable, fournit un code hexadécimal. Ce code peut être chargé dans la mémoire vive du MULTI 8 qui peut simuler ses propres micro-instructions. Un compilateur et un langage de microprogrammation ont été étudiés au Centre d'Etudes Nucléaires de GRENOBLE, intéressant pour l'optimisation de l'allocation des éléments de mémorisation [46].

5.4. Réalisations non spécifiques

Plus intéressantes pour nous sont les réalisations à vocation universelle.

La réalisation la plus ancienne est à notre connaissance celle de la R.C.A. La description du hardware, l'affectation des variables aux éléments de mémorisation, l'état initial ainsi que les variables à sauvegarder sont décrits

dans les déclarations. D'excellentes idées pour la description du microprogramme ont été reprises plus tard. Le microprogramme, mis sous forme arborescente (les boucles ne sont pas traitées) est optimisé après génération (par exploitation du parallélisme) en trois passages.

Au Centre Commun de Traitement de l'Information, du groupe THOMSON, a été développé un générateur de microprogrammes optimisés en temps à partir de la description de la micro-instruction (champs, codes, incompatibilités), d'une description de la structure câblée (éléments de mémorisation, connexions, contraintes) et d'une description de l'algorithme. On passe par l'intermédiaire d'une hiérarchie d'actions élémentaires pour optimiser le microprogramme en regroupant le maximum d'actions dans chaque micro-instruction [6].

5.5. Avantages d'un compilateur universel

Il est évidemment attrayant de cumuler, dans la mesure du possible, les avantages énumérés, c'est-à-dire de réaliser un compilateur universel de microprogrammes, lié de façon simple à un simulateur également universel et tel que l'utilisateur puisse faire varier le niveau du langage d'entrée pour limiter les choix faits par le compilateur, plus mauvais en général que ceux faits par l'utilisateur.

6.- PROBLEMES LIES A LA COMPILATION

Ces problèmes, sur lesquels nous reviendrons plus loin, peuvent être partagés en trois classes. Cependant leur solution ne peut être que globale.

6.1. Allocation des éléments de mémorisation

Pour décrire un algorithme à compiler, on utilise des identificateurs de variable. La machine ne possède que des éléments de mémorisation physiques. Un lien doit être établi entre nom et lieu , c'est-à-dire entre chaque variable et l'élément de mémorisation qui la contient.

Ce lien est une injection. Certains éléments de mémorisation peuvent, à un instant donné, ne pas contenir d'information utilisée plus tard, on dit qu'ils sont "vides" et une même variable peut être liée simultanément à plusieurs éléments de mémorisation. Ce lien est une fonction de temps.

Ce lien dépend des opérations exécutées sur chaque variable. Trivialement, une variable doit être aussi proche que possible de l'opérateur qui va la traiter, et d'autant plus facile d'accès qu'elle est localement dans le temps plus utilisée.

6.2. Allocation des opérateurs de la machine

Dans la description du programme à compiler, les opérations sont identifiées par un nom, et la machine possède des "boîtes à opération" (40), capables d'exécuter ces dernières. Le compilateur a la charge de reconnaître ces opérations, d'amener les opérandes vers les entrées de l'opérateur câblé qui peut les réaliser, et de diriger la sortie vers les variables qui reçoivent les résultats.

6.3. Implantation des micro-instructions

Le mécanisme de séquencement de la micromachine impose des contraintes sur l'implantation des micro-instructions dans la mémoire de contrôle.

Parfois, les champs contrôlant le déroulement du microprogramme recoupent certains champs opératifs de la machine et rendent les problèmes de l'implantation dépendants des problèmes précédents.

7.- DIVERSES SORTIES DU COMPILATEUR

Les compilateurs de programmes fournissent en général un module objet qui est repris avant exécution par un chargeur. Ils fournissent également, sur option, une liste des instructions générées dans un langage symbolique lisible.

Comme il est important que le code fourni par un compilateur de microprogramme soit optimisé la sortie du compilateur doit être claire et lisible, de façon à permettre une éventuelle amélioration du code généré par l'utilisateur du microprogramme.

D'après [41] ces sorties peuvent être de trois types.

7.1. Les transferts dans les chemins de données

Le langage de sortie sert à décrire, micro-instruction par micro-instruction, les transferts de données à travers la partie opérative, et les opérations effectuées. Nous appellerons ce type de sortie un "microprogramme formel" [41].

7.2. La séquence de contrôle

Le microprogramme fourni est alors une suite des valeurs de micro-instruction émises par la partie contrôle de la micromachine. Le séquencement est donné explicitement par des instructions de contrôle.

Si on considère la partie contrôle de la micromachine à microprogrammer comme un automate, suivant les descriptions de [20], cette suite est la séquence de sortie de l'automate de contrôle.

Nous ne nous préoccupons pas, dans ce cas, de l'implantation des micro-instructions à l'intérieur de la mémoire de contrôle.

7.3. L'image de la mémoire de contrôle

Le microprogramme est un tableau des valeurs binaires, ou code, de chaque mot de la mémoire de commande. Les valeurs, chargées dans la mémoire d'ordre de la micromachine, permettent d'exécuter l'algorithme à compiler.

8.- REALISATION PROPOSEE

Le compilateur de microprogramme que nous proposons peut être décrit par le schéma suivant:

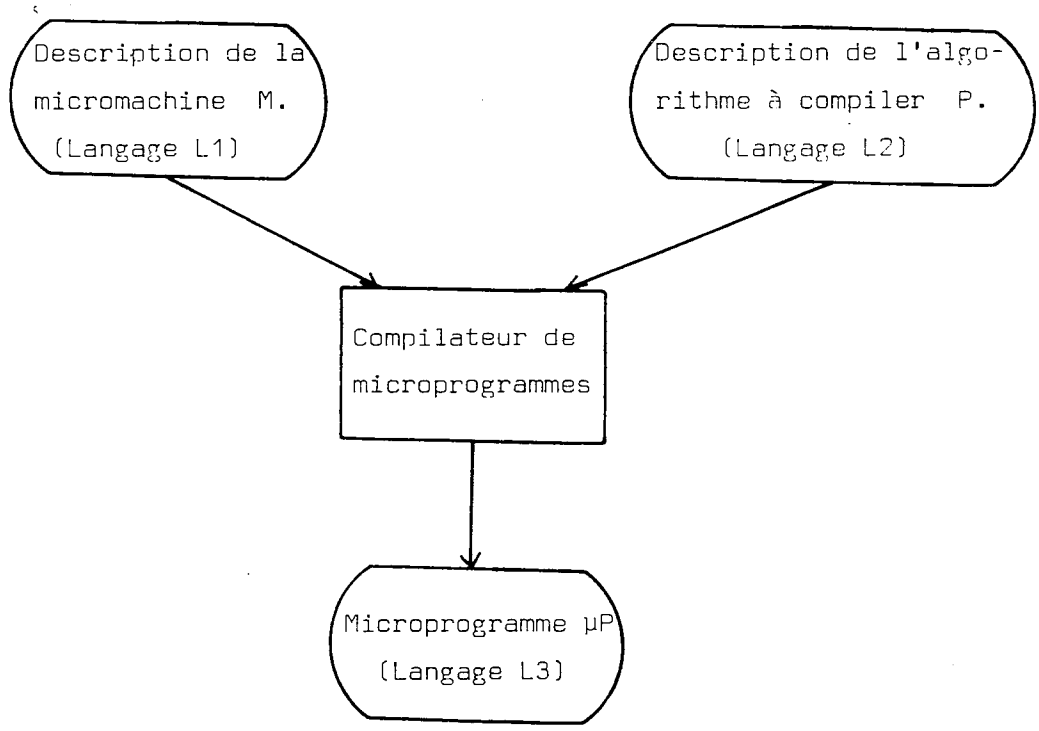


figure 1.3.

Le microprogramme µP exécute directement l'algorithme P sur la machine M.

8.1. Choix du langage L1

Le langage L1 doit être au moins apte à décrire les machines microprogrammées. Le niveau doit être logique, assez fin pour pouvoir rendre compte des particularités du matériel, tout en étant clair, souple et concis. Le langage CASSANDRE est de ceux qui répondent le mieux à ces exigences. Il offre de plus les avantages suivants:

Ce langage est connu, commercialisé et a de nombreuses utilisations dont la plus importante pour la compilation de microprogrammes est la simulation. Cette simulation permet de vérifier dynamiquement que la micromachine décrite correspond bien à son cahier.

Les autres applications de ce langage, comme la "compilation de hardware" et l'"aide à la conception des machines microprogrammées" sont entièrement compatibles avec la description de la machine M en vue de la compilation de microprogrammes, comme montré en annexe.

De plus, l'éditeur-vérificateur de CASSANDRE [15], exploité en commun par toutes les applications de ce langage, facilite l'écriture de suiveurs syntaxiques et assure l'unicité, au moins syntaxique, du langage.

8.2. Choix du langage L2

Le choix, que nous avons fait, de CASSANDRE comme langage de description de l'algorithme P à compiler en microprogramme pour la machine M est beaucoup moins immédiat, car il a comme langage algorithmique, de nombreux concurrents performants.

Cependant, son utilisation pour décrire des algorithmes n'est pas nouvelle: il est utilisé dans [48] pour obtenir la description d'une machine microprogrammée et le microprogramme aptes à l'exécution d'un algorithme donné, et, plus récemment dans [19] pour obtenir la description d'un automate câblé réalisant un algorithme donné (Flowcharting).

Ce choix se justifie par le fait que les algorithmes à microprogrammer manipulent des quantités booléennes. Un exemple d'algorithme peut être un interpréteur d'instructions, les instructions étant des chaînes de bits, de longeur fixe ou variable, dont le découpage en champs et la signification des champs sont variables. Un tel interpréteur peut être considéré comme une définition sémantique extrêmement précise et rigoureuse d'un langage d'instructions, dont la syntaxe, très simple, est le format.

Cette remarque reste valable pour les canaux, les contrôleurs et même les machines-langage, bien qu'une certaine syntaxe, qui interdit certaines successions d'instructions à interpréter, soit alors introduite.

Bien qu'APL ait décrit le 360 IBM, il n'existe pas encore de bons langages de description de langages d'instructions. Le langage de VIENNE, toujours IBM, est une tentative en ce sens. Le langage PASCAL, avec ses"records" conditionnés par des "tagfields", ses types "scalar" et "powerset", est bien adapté pour décrire des langages avant leur codage, c'est-à-dire des langages en cours de conception et donc sujets à modification.

CASSANDRE offre des variables booléennes vectorielles, de longueur quelconque, permettant la désignation de champs, des décodeurs de ces champs, sous la forme de "si-valnum", et enfin des valeurs d'état qui permettent d'étiqueter les différents pas d'un algorithme et de décrire par là des opérations purement séquentielles ou parallèles. Ce langage a été utilisé pour décrire l'interpréteur des instructions d'une machine qui devait être réalisée en LSI par PHILIPS.

Les avantages énumérés pour la description de la machine M restent valables pour le programme P.

Il ne faut pas non plus négliger l'intérêt d'utilisation du même langage pour la machine et le programme. Elle autorise l'emploi de mêmes identificateurs pour désigner les mêmes objets, apparaissant dans la machine et dans l'algorithme, comme par exemple les opérateurs "externes", considérés comme des macrocomposants.

L'utilisateur peut, grâce à ce dernier point, fixer le niveau de la description du programme P, qui peut aller de l'assembleur, ne laissant aucune initiative au compilateur, à des niveaux plus élevés où ce dernier a la charge d'ordonancer les opérations et d'allouer les variables intermédiaires.

8.3. Choix du langage L3

Nous avons encore choisi CASSANDRE pour exprimer les deux types de sorties possibles du compilateur.

Le résultat est lisible sans apprentissage et peut être simulé par simple concatenation du microprogramme et de la machine à microprogrammer.

Les sorties étant des descriptions d'algorithmes en CASSANDRE, à très bas niveau, elles sont éventuellement reprises, après amélioration "à la main", comme entrée P du compilateur.

9.- UTILISATION DU COMPILATEUR

Le compilateur peut fournir un microprogramme exécutant un algorithme donné sur une machine figée. Il est plus intéressant d'exploiter ses qualités d'auto-adaptation en modifiant pas-à-pas la description de la machine ou de l'algorithme, en vue de les adapter l'un à l'autre.

9.1. Adaptation de la machine

Les choix suivants peuvent être guidés par le compilateur:

- Dans la partie opérative de la machine:
interconnexion des éléments, c'est-à-dire définition des chemins de données, choix des opérateurs et de la taille des mémoires locales.

- Dans la partie contrôle de la micromachine:
codage de la micro-instruction en vue d'un compactage de sa longueur.

Comme montré en annexe, la deuxième sortie du compilateur est exploitée par un programme d'aide à la conception qui exhibe les éléments ou interconnexions jamais (ou rarement) utilisés, et propose le codage de la micro-instruction le plus compact compte tenu de l'utilisation. Malheureusement il n'est pas tenu compte des possibilités de décodage.

9.2. Adaptation du programme

A l'inverse, en fixant le nombre de variables de l'algorithme à compiler en fonction des éléments de mémorisation pouvant les recevoir, en imposant certaines localisations, en utilisant judicieusement les opérateurs, en bref, en simplifiant la tâche du compilateur, on peut améliorer le temps d'exécution des microprogrammes générés.

10.- METHODE PROPOSEE

On a la description d'une micromachine en CASSANDRE, pour générer des micro-instructions pour cette machine, nous proposons une méthode qui est calquée, en la précisant et en généralisant, sur celle qu'utilise naturellement le microprogrammeur [32].

10.1. Méthode naturelle

Le microprogrammeur qui connaît mal la micromachine pour laquelle il doit écrire, se sert d'un schéma synoptique représentant les chemins de données de la micromachine et sur lequel se trouvent des éléments de type connu (registres, opérateurs...) ayant un code graphique particulier. Le microprogrammeur utilise également des tables décrivant les propriétés et caractéristiques de ces éléments (opération réalisée, nombre de bits,...) et leur commande (incompatibilité, code,...).

On exprime alors l'algorithme à transcrire dans des primitives qui correspondent aux éléments de la machine (Variable≡registre, opération≡opérateur, transfert≡chemin de données, ...).

10.2. Méthode programmée

Cette méthode sera décrite en détail dans les chapitres suivants. Elle consiste premièrement en la construction automatique d'un schéma de la micromachine et de ses tables de propriétés, puis deuxièmement en la transformation de l'algorithme à compiler en un schéma, puis enfin troisièmement en une couverture régulière du schéma de l'algorithme à compiler par un

certain nombre de copies du schéma de la machine, ce qui donne les micro-instructions. Ces micro-instructions peuvent s'exprimer suivant les différents types de sorties.

11.- LIMITATION DE LA METHODE

Il importe de distinguer ce qui n'est pas réalisable dans le cadre de la méthode proposée mais qui pourrait éventuellement l'être par d'autres techniques, de ce qui est réalisable et qui se subdivise en programmes concrets et en parties non encore écrites.

Notons également que l'optimisation des microprogrammes semble raisonnable si le compilateur est utilisé correctement.

11.1.- Implantation

Le problème de l'implantation a été séparé de celui de la génération de la partie opérative de la micro-instruction. Cette séparation n'influe en rien sur les performances de certaines machines [18], mais peut être grave pour d'autres [33].

De toute façon, en dépit de tentatives de rationalisation des parties contrôle [54], ou d'études d'implanteurs universels [17], le problème de l'implantation n'a pas, à notre connaissance, reçu de solution satisfaisante.

Ceci exclut le troisième type de sortie du compilateur que nous avons proposé, c'est-à-dire l'image de la mémoire de contrôle, présentée au paragraphe 7.3.

11.2. Découpage des variables

Les variables utilisées dans le programme à compiler ont une certaine taille en bit , nécessairement précisée lors de la déclaration obligatoire de ces variables.

Le tronçonnage de ces variables, pour les rendre compatibles avec la taille des chemins de données internes de la micromachine, n'est pas du ressort de la compilation mais de l'algorithmique.

On ne saurait exiger d'un compilateur qu'il retrouve, par exemple, l'algorithme de multiplication de deux entiers sur trente deux bits avec un multiplieur de huit bits [38].

11.3. Décomposition des opérations

De même, on ne saurait demander que, par exemple, la multiplication soit décomposée automatiquement en une série d'additions et de décalages, avec tests et comptage si l'opération multiplication n'est pas câblée dans la machine considérée.

11.4. Allocation des variables

Le problème est celui-ci: le nombre de variables intermédiaires nécessaires pour mener un calcul et qui ne figurent pas dans l'algorithme à microprogrammer, dépend de l'implantation de ces mêmes variables dans les éléments de mémorisation et réciproquement.

Ici encore, le problème a été décomposé en une allocation des variables figurant dans l'algorithme, ce qui constitue une optimisation globale de l'allocation, suivie d'une allocation des variables intermédiaires crées en cours de compilation, et qui est locale car elle ne se fait qu'en tenant compte que d'une petite partie du programme à compiler.

On trouve facilement des exemples où l'union de l'optimisation globale et de l'optimisation locale ne donnent pas le meilleur résultat.

11.5. Amélioration du programme

Les optimisations qui sont faites, entre autres, dans le compilateur de FORTRAN 4H et qui consistent en une détection et correction des maladresses du programmeur, ne sont pas du tout abordées.
A mauvais microprogrammeur, mauvais microprogrammes.

12.- CONCLUSION

Conscient d'avoir laissé de côté un certain nombre de problèmes, nous avons voulu réaliser un système aisément améliorable, donc modifiable. Pour cette raison, le système est un assemblage de modules constituant une sorte de réseau dont les arêtes sont des fichiers et les noeuds des modules programmes. Cette structure permet d'intercaler des programmes dans ce réseau pour apporter des améliorations au système.

DEUXIEME CHAPITRE

oooooooooooooooo

La technique de production de microprogrammes utilisée, qui sera exposée dans ce chapitre un peu plus précisément, implique une modélisation de la machine à microprogrammer. En effet, les calculateurs microprogrammés ne constituent qu'un sous-ensemble restreint des systèmes logiques qui peuvent être décrits en CASSANDRE, il faut donc imposer des contraintes qui permettent d'assurer que le système décrit est une machine microprogrammée et qui facilitent la reconnaissance de cette machine par le compilateur de microprogrammes. Il n'est nécessaire d'imposer aucune restriction syntaxique au langage de description.

1.- DESCRIPTION DE LA MACHINE

La machine sera décrite sous forme de deux unités, la partie de contrôle et la partie opérative, réunies dans une unité englobante qui constitue le calculateur.

Les connexions de ces deux unités comprennent:

- Dans le sens contrôle-opération une nappe de fils, appelés parfois nerfs moteurs, par analogie avec un organisme vivant évolué. Cette nappe de fils est un signal qui a pour valeur à chaque instant celle de la micro-instruction en cours d'exécution. Nous appellerons micromot ce signal. Le compilateur de microprogramme fournit comme sortie du premier type la suite des valeurs de ce micromot.

- Dans le sens opération-contrôle une nappe de fils indique l'état de la partie opérative, c'est-à-dire donne des renseignements sur les résultats d'opérations, occupations d'éléments, etc... On peut les appeler nerfs sensitifs.

1.1. Partie contrôle

Suivant la définition donnée dans le premier chapitre, cette unité doit contenir une mémoire, dite mémoire de contrôle, qui contient des micro-instructions (assimilables aux instructions des machines à programmes enregistrés) et un registre contenant en permanence l'adresse de la micro-instruction en cours d'exécution. Nous n'avons pas cherché à faire un modèle qui rende compte des problèmes d'adressage, d'implantation et de séquencement de la mémoire de contrôle, les réalisations étant trop diverses.

La partie contrôle est un automate dont l'état est donné par la valeur de son registre d'adresse, compteur ordinal de la micromachine, le vocabulaire de sortie étant constitué par l'ensemble des valeurs de micromots et le vocabulaire d'entrée par l'ensemble des valeurs de fils de tests.

1.2. Partie opérative

La partie opérative de la machine doit être décrite en une seule unité [43]. Peut-être des outils de fusion formelle d'unités emboitées décrites en CASSANDRE viendront faire sauter cette limitation [43]. Cette unité doit posséder un ou plusieurs signaux spécifiés CONTROLE et qui correspondent au micromot dont le compilateur déterminera les valeurs successives. Ces signaux vont, par l'intermédiaire d'un certain nombre de décodeurs organisés en couches plus ou moins nombreuses correspondant à des codages à mot court, mot long ou direct [45] , commander l'ouverture de chemins de données ou le chargement d'éléments de mémorisation.

Les instructions ou expressions conditionnelles seront largement utilisées pour exprimer l'influence des signaux de contrôle sur les chemins de données, l'introduction du "si vectoriel" et du "si valnum" [43], améliorant la concision de la description. Cette description va servir à la construction d'un réseau dont les noeuds appartiennent à un petit nombre de types différents.

Comme la partie contrôle, la partie opérative est un automate dont nous définirons les états dans le chapitre 4.

1.3. Modèle de machine microprogrammée

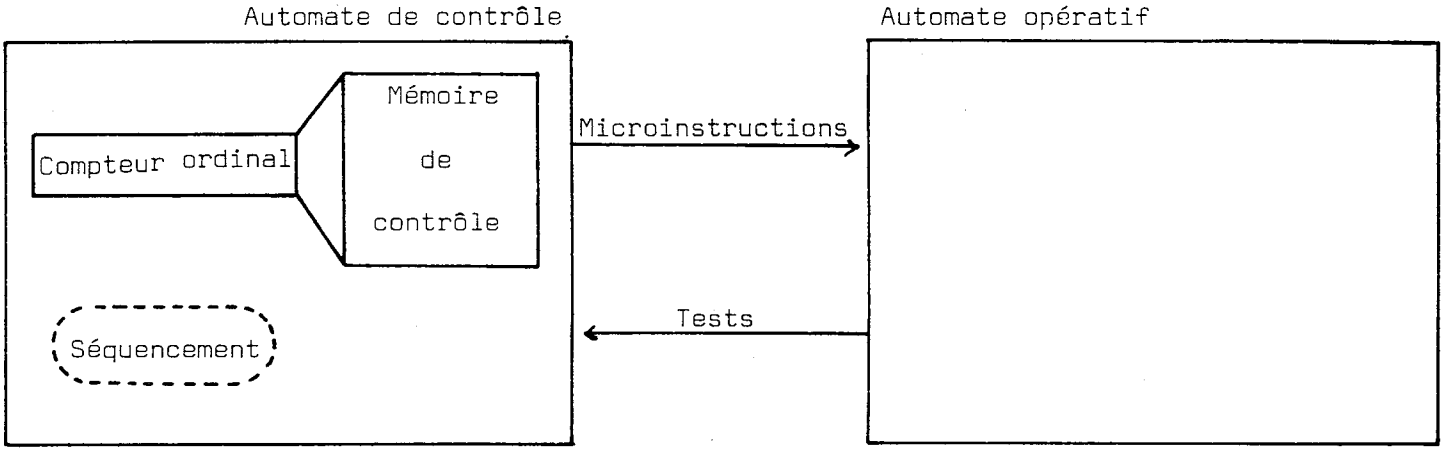


figure 2.1

Vocabulaire de sortie:

- valeur de micro-instruction,

Vocabulaire d'entrée:

- valeur de tests.

REMARQUE: Les relations de ce modèle avec le monde extérieur ne sont pas considérées.

1.4. Modèle de la partie opérative

La partie opérative contient deux hiérarchies. La première est la hiérarchie de contrôle dont les éléments maximaux sont les origines du contrôle, c'est-à-dire les signaux spécifiés tels dans la description, et les éléments minimaux sont des portes logiques point de contact entre la hiérarchie de contrôle et les chemins de données (synapses, pour reprendre la comparaison avec les organismes vivants), les noeuds intermédiaires réalisant des décodages, découpages ou restructurations de signaux.

1.5. Exemple de hiérarchie de contrôle

Contrôle de la machine décrite en annexe.

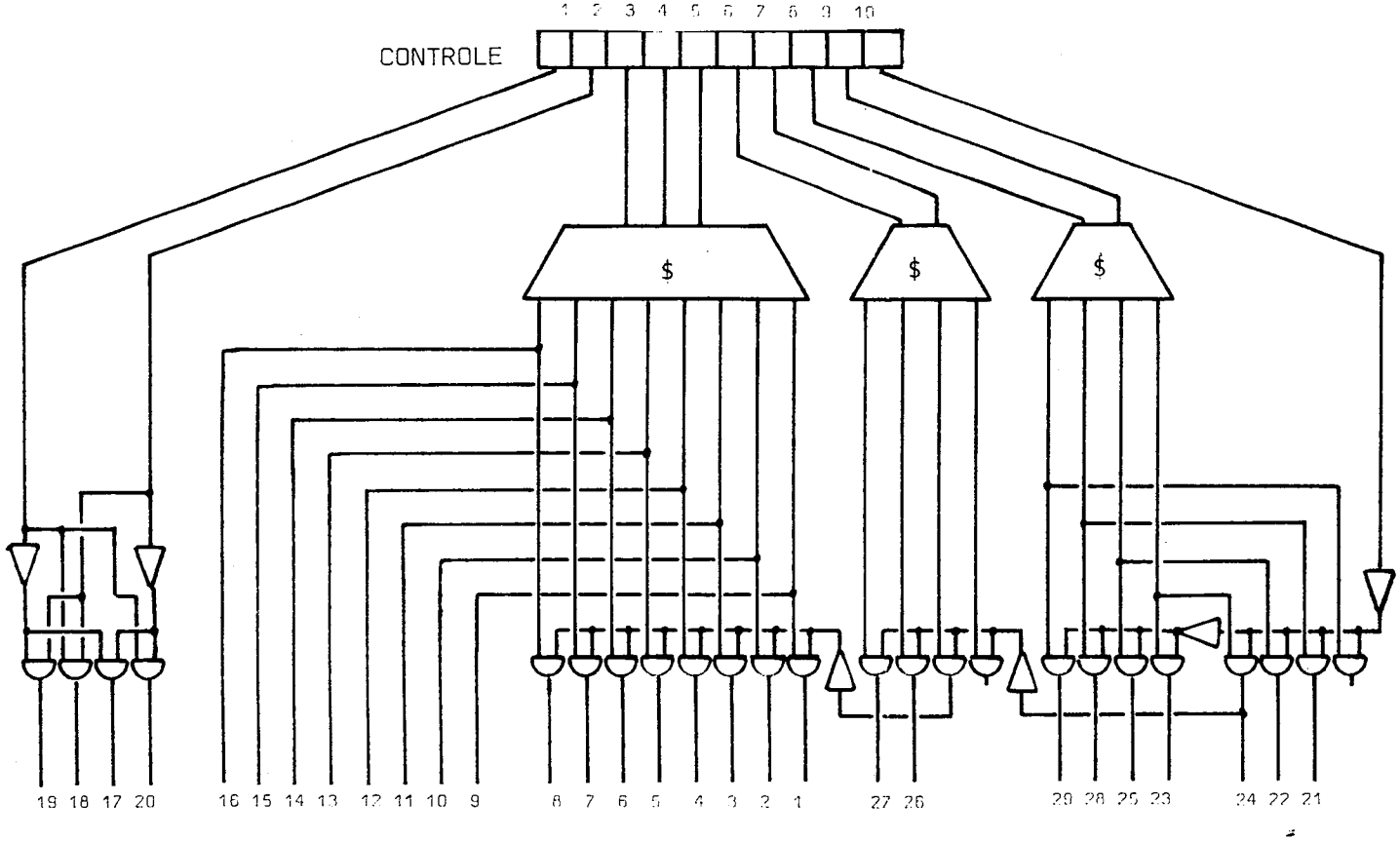


figure 2.2

REMARQUE: Les boîtes marquées $ sont les décodeurs 1 parmis n.
Cette hiérarchie n'est pas connexe en général.

1.6. Hiérarchie opérative

La deuxième hiérarchie est formée des registres, opérateurs et chemins de données. Il est d'usage de représenter une machine synchronisée par un schéma avec les éléments habituels, bus, registres, noeuds opératifs etc... Dans ce schéma les registres forment des barrières temporelles qui ne sont franchies que sous l'influence d'un top d'horloge. Il y a donc à chaque instant deux valeurs associées à chaque registre, l'ancienne valeur qui y est mémorisée et la nouvelle valeur en cours de stabilisation. A l'impulsion, la nouvelle valeur, qui est stable si le délai entre impulsions est suffisant, devient l'ancienne [2]. Nous éclaterons donc les registres en deux. Le graphe de la machine devient une hiérarchie (pas nécessairement connexe). En effet, ce graphe est sans circuit, car ces circuits seraient sans barrière temporelle et donc sujets à l'instabilité.

2.- PRESENTATION DU PROGRAMME A COMPILER

Le choix du langage de description du programme à compiler s'étant porté sur CASSANDRE, cette description prendra la forme d'une hypothétique machine ou unité. Le programme, qui se décomposera en général en une série d'instructions s'enchaînant dans le temps, peut être vu comme un automate de MOORE, chaque instruction, ou groupe d'instructions, étant validée par un état. L'enchaînement des instructions devient alors explicite et ne s'effectue plus dès que l'exécution de l'instruction est terminée, mais chaque fois qu'il y a une transition d'état. L'intervalle de temps entre deux transitions est suffisamment grand pour que les instructions soient correctement exécutées.

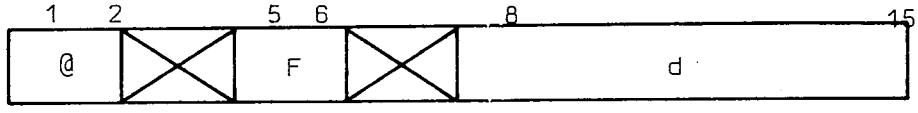
Les débranchements du programme à compiler deviennent des ordres de transition, conditionnels ou non. Les instructions conditionnées sont groupées en états atteints lorsque la condition est réalisée et sont ainsi ramenées aux débranchements.

2.1. Exemple de programme à compiler

L'ensemble des descriptions faites en CASSANDRE a été regroupé en annexe. Le deuxième exemple de cette annexe décrit, par le truchement d'un interpréteur, un jeu réduit d'instructions d'une machine à accumulateur.

2.1.1. Format

Format unique des macroinstructions:



@: mode d'adressage,

F: fonction,

d: déplacement.

figure 2.3.

Les zônes inutilisées peuvent servir à étendre le code.

Ce format reprend un fond commun à de nombreuses petites machines.

2.1.2 Organigramme

Organigramme du programme à compiler donné en annexe.

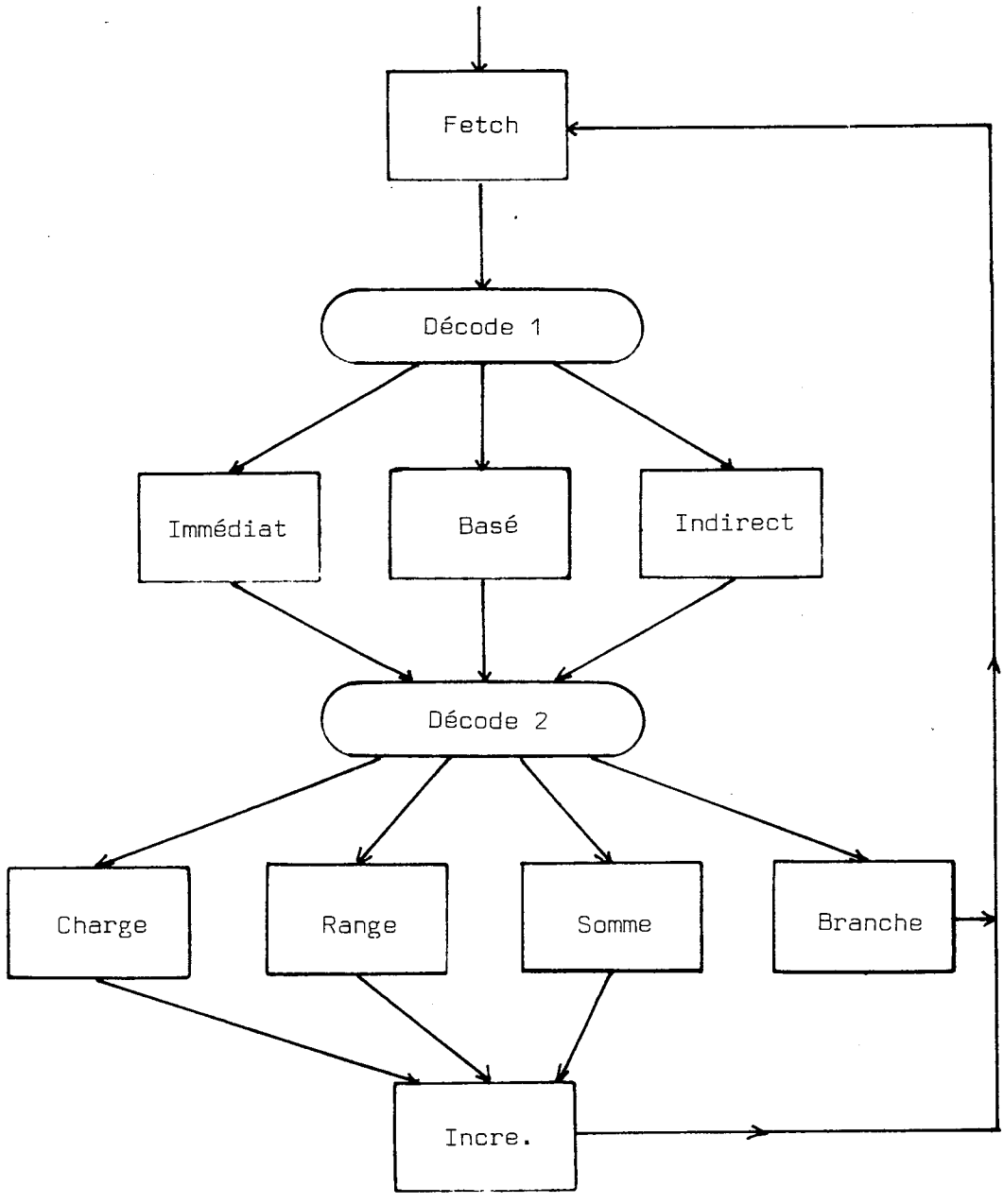


figure 2.4

2.2. Analyse du programme à compiler

Nous nommerons Incrément l'ensemble des instructions validées par un état. L'ordre d'écriture de ces instructions est sémantiquement sans importance. Le simulateur CASSANDRE permet en effet leur exécution en "parallèle". Le compilateur de microprogramme les examinera donc simultanément. Si une seule micro-instruction ne peut suffire pour commander l'exécution de toutes les instructions contenues dans un incrément, le compilateur a alors la charge de déterminer un ordre d'exécution dont le résultat soit strictement le même que celui obtenu en parallélisme synchrone.

Par exemple, l'exécution en parallèle des deux instructions d'affectation "A→B, B→A" est parfaitement définie et correspond à une permutation des valeurs respectives de A et B.

2.3. Séquence dans un incrément

Le compilateur est également chargé de déterminer un ordre d'exécution pour les instructions dont certaines se servent des résulats générés dans le même temps par d'autres. Ceci toujours dans le cas où une seule micro-instruction ne suffit pas à commander l'exécution simultanée de toutes ces instructions.

Si cet ordre temporel d'exécution n'existe pas, ce peut être dû à un bouclage et le compilateur le découvre. Par exemple, l'ensemble des instuctions de branchment "A := B; B := A", qui a une solution en simulation, n'est pas accepté par le compilateur, non plus que l'ensemble "A := B; B := $\overline{A}$", qui lui n'a pas de solution.

L'ordre d'exécution se matérialise dans la micromachine par la propagation d'un "front" de stabilisation entre opérateurs en cascade, et éventuellement par le franchissement de "barrières temorelles" lors de phases ou de micro-instructions successives.

La possibilité de décrire des séquences d'opérations, séquences exécutées en parallèle et regroupées dans des états s'exécutant en série, peut être comparée aux propositions sérielles et collatérales d'ALGOL 68.

En bouleversant l'ordre d'écriture d'instructions parallèles pour les rapprocher de leur ordre d'exécution optimum, on peut réduire considérablement le temps de calcul nécessaire à leur simulation. Un optimiseur, basé sur ce principe, a été écrit.

2.4. Construction de hiérarchie

Un incrément est une collection d'instructions, chacune peut être mise classiquement sous forme d'arborescence dont la racine est la partie gauche, les feuilles sont les variables ou valeurs, et les noeuds sont les opérateurs. En fusionnant racines et feuilles, qui sont des signaux de même nom et en réduisant les noeuds ainsi créés, on obtient une hiérarchie pas forcément connexe, dont les extrémités sont des variables (registres) ou des valeurs (constantes booléennes). Cette hiérarchie, est nommée schéma de l'incrément.

La fusion, puis réduction des signaux, correspond au caractère non fonctionnel de ceux-ci. Ce rôle descriptif est attesté par le fait qu'à toute description CASSANDRE on peut en faire correspondre une autre, certe moins élégante, mais sémantiquement équivalente et sans aucun signal.

2.5. Fusion de feuilles et noeuds

En CASSANDRE, les grandeurs logiques peuvent être organisées en tableaux ayant un nombre quelconque de directions. Pour faciliter la construction du schéma d'incrément, les signaux ont été limité à des nappes, ou vecteurs, de fils. En fonction de la paire de bornes qui le suit éventuellement, un même nom de signal peut désigner des sous-ensembles différents d'une même nappe de fils. La fusion de signaux peut nécessiter d'intercaler un noeud opérateur d'adaptation qui est implicite dans la description. Ce problème, qui se présente également lors de la construction de la hiérarchie de contrôle ou de la hiérarchie opération d'une machine, est traité dans le septième chapitre.

2.6. Lien entre schéma d'incrément et hiérarchie opérative

Pour l'essentiel, un schéma d'incrément est constitué d'opérations et de variables et une hiérarchie opérative de registre et d'opérateurs, les éléments de chaque ensemble étant reliés. Les opérateurs sont parfois nommés "boîte à opérations", ce qui exprime leur faculté d'exécuter exclusivement plusieurs opérations.

La relation variable-registre et opération-boîte à opération s'impose. Au niveau logique (et non analogique) où nous nous sommes placés, cette relation est une fonction discrète du temps, les opérations effectuées et les allocations de registre à variables pouvant varier par exemple à chaque phase, ou micro-instruction.

2.7. Reconnaissance des identificateurs d'opération

Les opérations du programme à compiler doivent avoir le même nom que les opérateurs correspondants de la machine qui va exécuter le microprogramme. Ces opérations peuvent appartenir au vocabulaire CASSANDRE, comme par exemple toutes les opérations LOGIQUES (monadiques ou diadiques) ou bien être des circuits plus complexes qui sont globalement décrits comme des unités externes (macrocomposants).

Remarque:
Si une opération de la micromachine doit recevoir une certaine valeur binaire sur une entrée contrôle pour exécuter une certaine opération, il incombe à l'utilisateur du compilateur de préciser cette valeur.

Exemple:

ALU (1, entrée 1, entrée 2; sortie)
ALU (0, entrée 1, entrée 2; sortie)

L'unité ALU exécutant une addition si son fil de commande a le niveau 1 et une soustraction avec le niveau 0.

On peut considérer que ALU(1,,;) et ALU(0,,;) sont des identificateurs d'opérations différentes.

Les identificateurs d'opérateurs, d'opérandes et de contrôle sont les seuls pouvant être communs à la description de la machine et de l'algorithme à compiler.

2.8. Couverture

On va essayer de ramener les problèmes de compilation de microprogramme à un problème de couverture de schéma d'incrément par une hiérarchie opérative.

Donnons-nous un petit problème qui servira d'illustration:

2.8.1. Programme

On veut faire exécuter l'ensemble d'opérations qui suit:

$A \leftarrow \alpha$,

$B \leftarrow \alpha$,

$\alpha := A + B + C$;

où α est une variable intermédiaire.

2.8.2. Micromachine

On dispose d'une micromachine dont le schéma est le suivant:

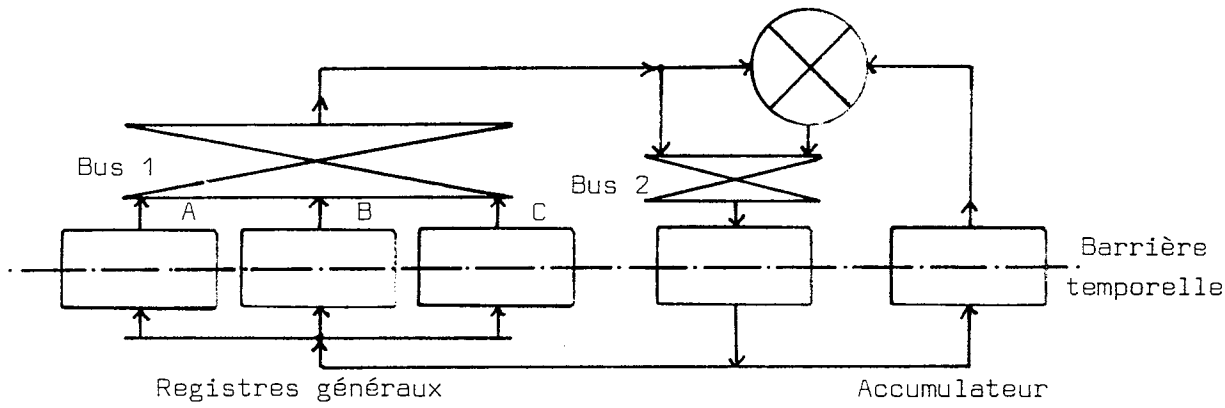
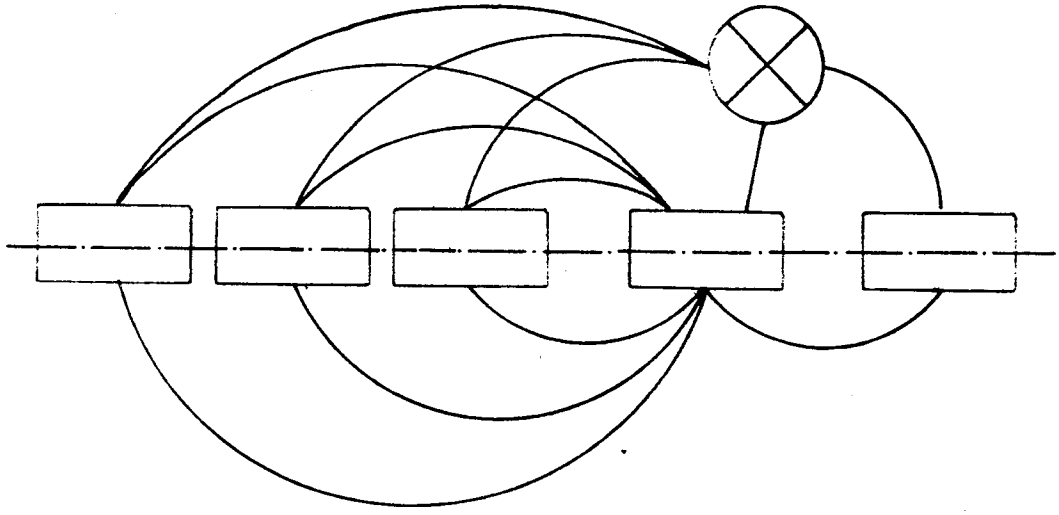


figure 2.5.

Cette machine ne possède qu'un champ de la micro-instruction pour désigner un registre général, ce qui implique qu'un seul de ces registres est accessible par micro-instruction. Les symboles utilisés pour ce dessin sont tirés de [43].

2.8.3. Transformation du schéma

En ne gardant que les éléments fonctionnels, on obtient un réseau équivalent qui n'exprime plus de contrainte.



Puis en décomposant les registres en deux, obtenir un sur-ensemble des opérations réalisables par la machine en un cycle.

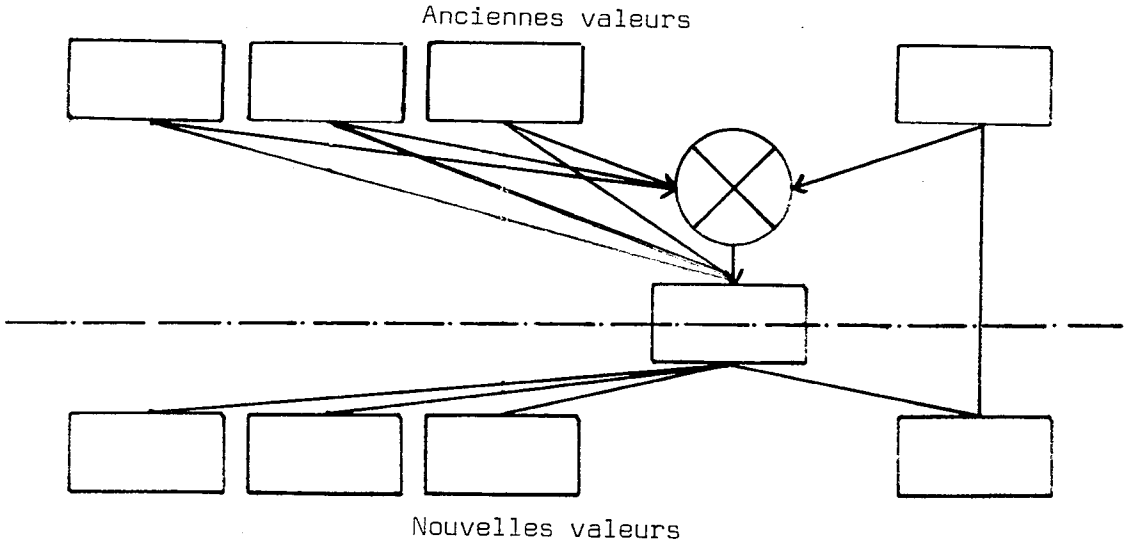


figure 2.8.

En dupliquant n fois les éléments de la machine qui sont réutilisables dans le temps, on peut obtenir un sur-ensemble des opérations réalisables par la machine en n cycles, ou n micro-instructions.

Voici le schéma obtenu pour n = 3, après suppression des variables intermédiaires.

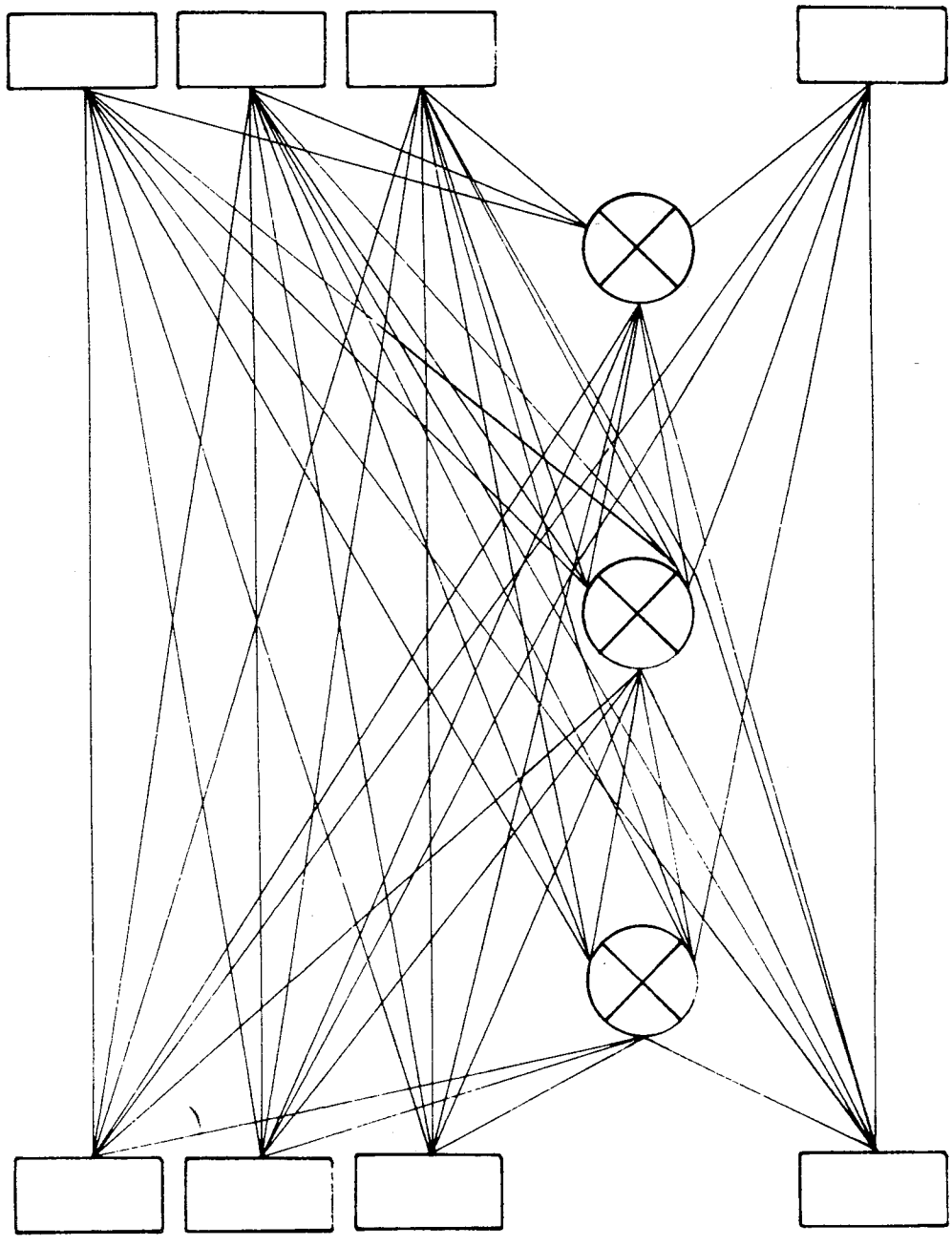


figure 2.9.

Ce réseau est inutilisable car il devient gigantesque au bout d'un petit nombre de micro-instructions. De plus, en supprimant les registres intermédiaires on ne rend plus compte de la contrainte suivante: un registre ne peut abriter qu'une seule variable à un moment donné.

2.8.4. Incrément

L'ensemble des opérations à faire exécuter par cette machine donne le schéma d'incrément suivant:

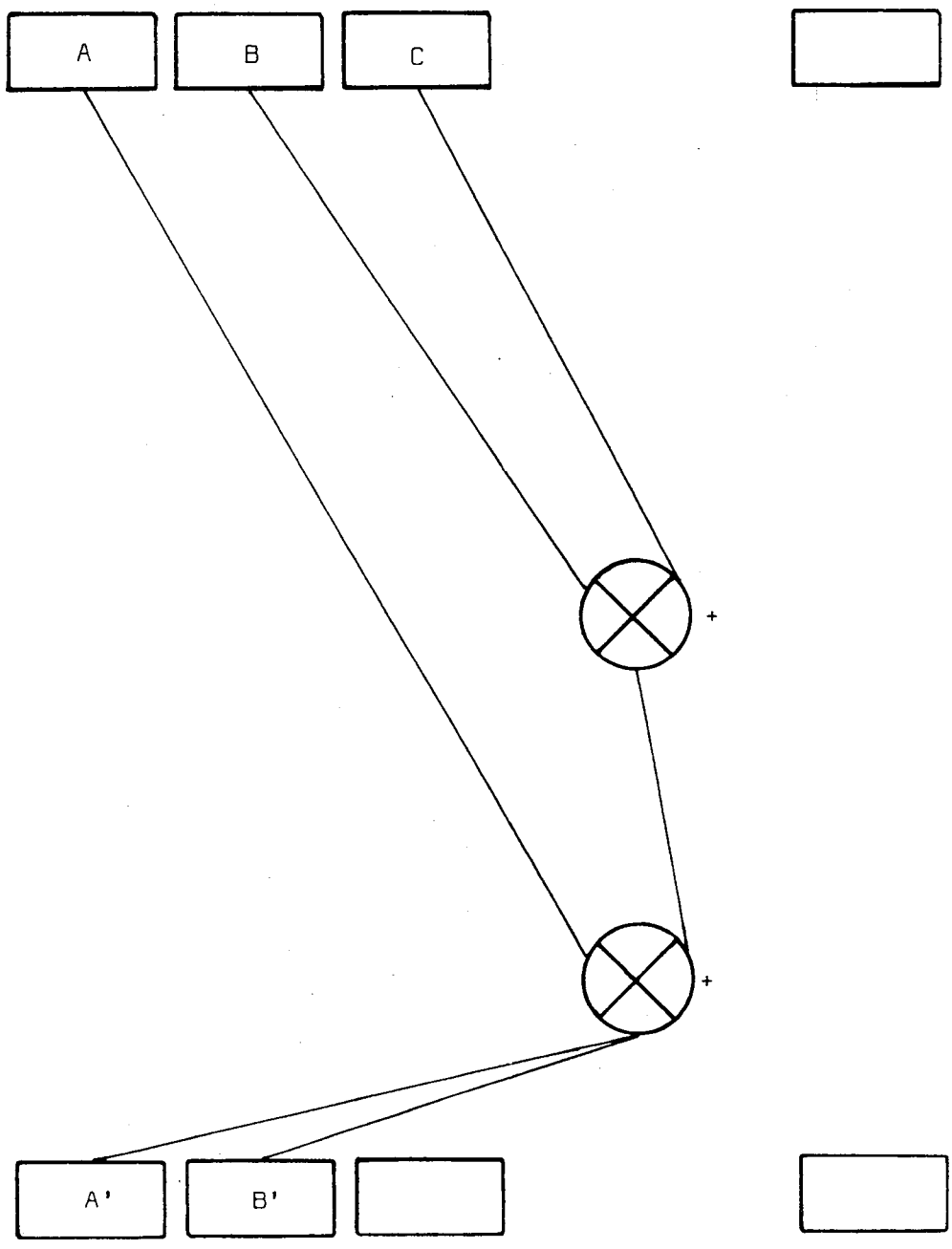


figure 2.10.

Ce schéma est contenu dans le schéma donné à la page précédente.

2.8.5. Duplication

Plutôt que de chercher à obtenir un réseau, difficile à manipuler, représentant l'ensemble des opérations exécutables en n cycles, on va dupliquer n fois le même réseau

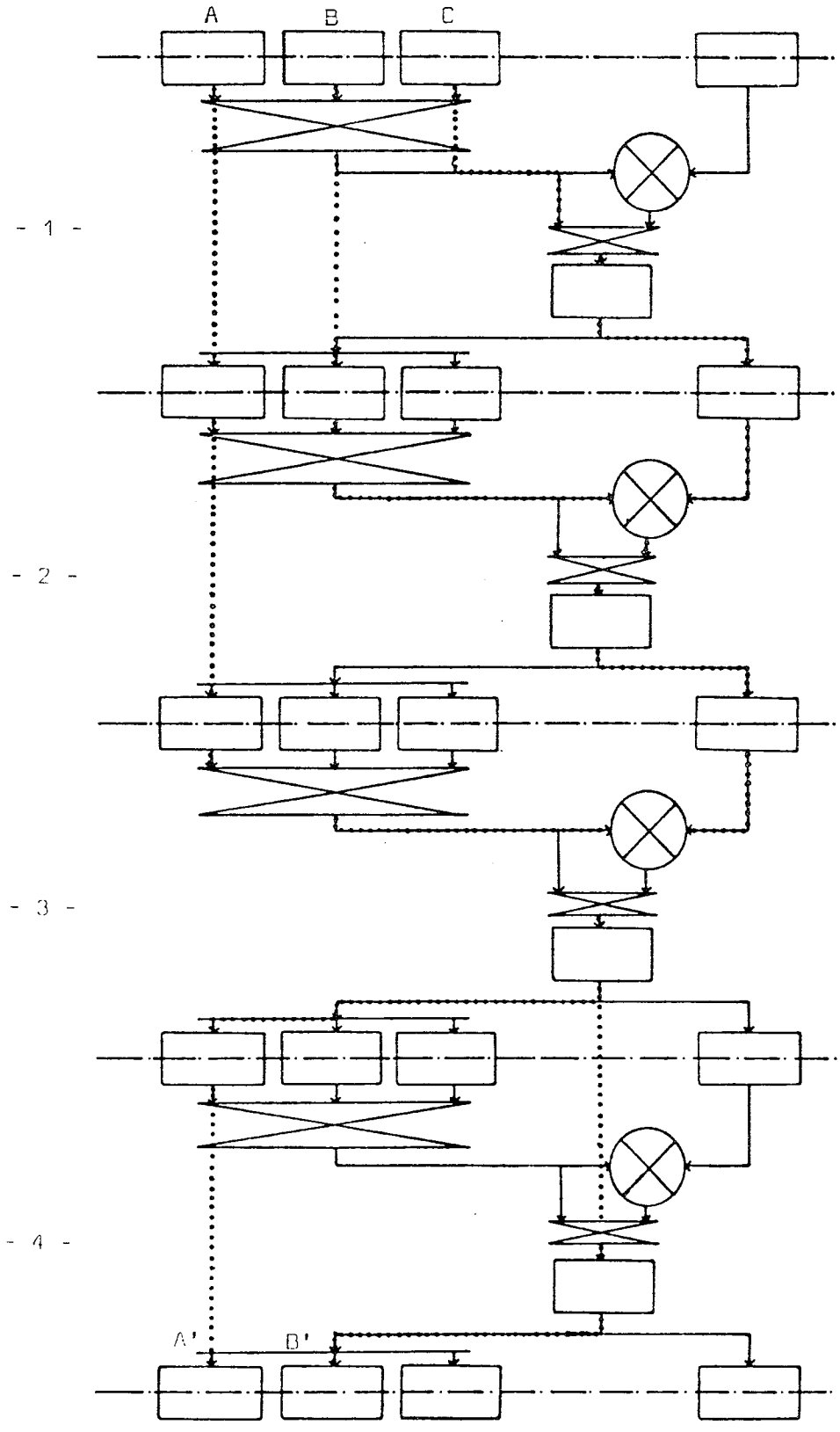


figure 2.11.

Le réseau en pointillé sur la gauche de la figure 2.11 est homomorphe au réseau de droite. Les arêtes du réseau en pointillés sont soit des arêtes du réseau de gauche (chemins de données), soit relient le même registre dans le temps.

3.- INSUFFISANCE DU RECOUVREMENT

La méthode de couverture n'est pas satisfaisante car elle ne tient compte en aucune façon des incompatibilités dûes à la partie opérative ou au contrôle et pas plus du fonctionnement des registres qui conservent une valeur aussi longtemps qu'ils ne sont pas rechargés et qui ne peuvent contenir qu'une seule valeur.

On peut utiliser la méthode de recouvrement d'un schéma d'incrément par un minimum de copies du schéma d'une machine pour proposer des solutions à condition de disposer d'un mécanisme capable d'assurer la validité de ces solutions.

4.- FORMAT DE SORTIE DU COMPILATEUR

Le résultat du compilateur est sous la forme d'un automate décrit en CASSANDRE à chaque état duquel correspond une micro-instruction.

La première micro-instruction correspondant à la compilation d'un incrément est étiquetée du nom de cet incrément s'il existe ; les autres ne le sont pas. A chaque incrément correspond au moins une micro-instruction générée, ce qui implique qu'en faisant boucler la sortie de ce compilateur sur son entrée, on obtienne un microprogramme au moins aussi long.

Comme nous l'avons dit au premier chapitre, les micro-instructions peuvent se présenter sous deux formes:

- une suite de transferts et d'opérations appelées actions élémentaires, ce qui constitue la deuxième sortie du compilateur ou "microprogramme formel", et fait le lien avec d'autres utilisations.

- la valeur binaire de la micro-instruction qui commande l'exécution de ces diverses actions.

Notons qu'une valeur de micro-instruction commande l'exécution d'un ensemble bien défini d'actions élémentaires, cet ensemble pouvant être commandé par plusieurs valeurs de la micro-instruction.

Lorsque le codage des micro-instructions est particulièrement efficace, il y a bi-jection.

Notons enfin que le compilateur peut transformer sa première sortie en la seconde et réciproquement.

TROISIEME CHAPITRE

oooooooooooooooo

Les schémas de machine,ou hiérarchie opérative,et schémas d'incrément, introduits pour utiliser une méthode de couverture, sont précisés dans ce chapitre. Nous en donnons les éléments constituants et indiquons comment on les obtient par un programme de construction.

Les réseaux structurels définis par J.MERMET [43] sont formés d'articulations d'un petit nombre de sortes, connectées entre elles pour former un réseau. Les articulations, ou éléments, forment une sorte de "mécano" qui permet, par assemblage, d'obtenir un modèle ayant un comportement équivalent à celui d'une machine donnée.

La machine d'une part, et les éléments ainsi que leur interconnexion d'autre part, peuvent être décrits en CASSANDRE, et l'équivalence entre la machine et le modèle est alors définie par leur comportement à la simulation.

Ces éléments peuvent être utilisés pour construire le schéma de la partie opérative d'une machine aussi bien que pour construire un schéma d'incrément.

Cependant, pour d'une part, indiquer les connexions entre la partie contrôle et la partie opérative d'une machine, et, d'autre part, faciliter la construction et la reconnaissance de ces réseaux, de nouveaux types d'articulations ont été créés et des renseignements ajoutés à chacune d'elles.

1.- MEMORISATION DU RESEAU

L'espace mémoire dans lequel sera construit le réseau est formé de mots dont le premier demi-mot contient soit une information, soit un pointeur sur un autre mot et le deuxième toujours un pointeur.

Une valeur particulière de pointeur ne pointe sur rien et indique une fin de liste.

2.- HIERARCHIE

Dans le réseau, les registres qui sont chargés sous l'influence d'une impulsion d'horloge, forment des "barrières temporelles". Ces barrières temporelles découpent le réseau en un ensemble de hiérarchies, car toute boucle qui n'est pas coupée par au moins une barrière temporelle est susceptible d'être instable.

Nous appellerons source ou puits [49] les noeuds extrêmes de ces hiérarchies qui sont souvent des registres. Pour passer de la hiérarchie au réseau fonctionnel, il suffit de fusionner les registres source et puits de même nom, et de les séparer en deux pour faire l'inverse.

Exemple:

Graphe fonctionnel simplifié de la machine donnée en exemple en annexe

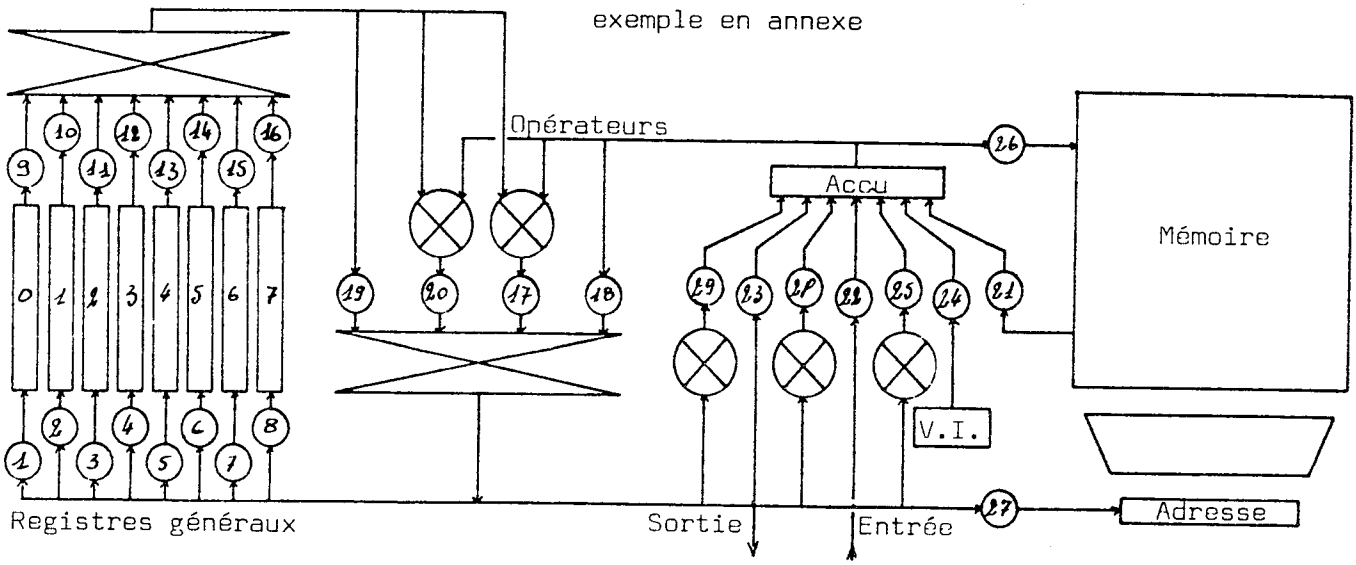


figure 3.1.

Hiérarchie correspondante :

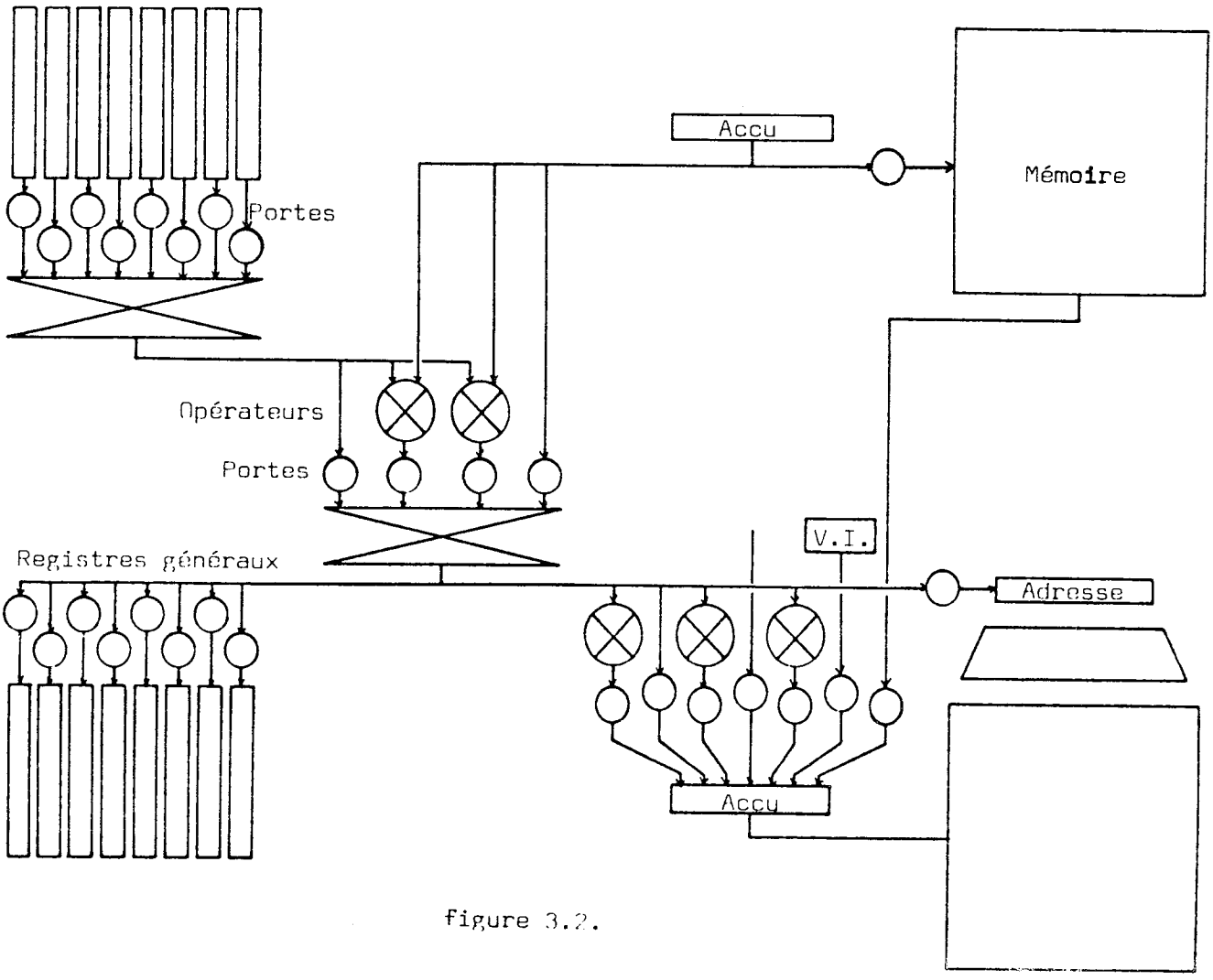


figure 3.2.

3.- ETOILE DU RESEAU

Les étoiles du réseau ci-dessus représentent des nappes de fils orientés, avec une entrée et une ou plusieurs sorties, chacune de ces nappes ayant un certain nombre d'équipotentielles qui est la taille de l'étoile. Ces étoiles sont donc capables de propager une variable booléenne vectorielle.

De même les étoiles du schéma d'un incrément sont des variables booléennes vectorielles.

4.- NOEUDS DU RESEAU

Voici la liste des différents types de noeuds constituant le réseau, les renseignements qui y sont attachés et éventuellement leur correspondance en CASSANDRE.

Sera également indiqué si ces noeuds appartiennent au schéma d'une machine, à celui d'un incrément ou aux deux.

Un fil venant de la partie contrôle vers la partie opérative sera appelé condition.

4.1. Bus ou multiplexeur

Réalise la fonction choix (exclusif) des entrées. Le noeud possède les renseignements ou marques suivants: type, nom, taille, et un ensemble de doublets (dont l'ordre est sans importance) dont chacun est formé du nom d'un noeud et d'une condition ; cette condition autorise, si elle est vraie (valeur 1), la connexion de l'entrée correspondante avec la sortie du bus. Toutes les entrées d'un bus ont la même taille. Dans les figures, un bus est représenté par deux triangles isocèles opposés par le sommet, les bus n'apparaissent que dans les schémas de machine.

Remarques:

- Une porte logique est un bus ayant une seule entrée.
- Les noeuds du réseau comportant plusieurs entrées ou sorties non équivalentes, il faudrait parler de sur-réseau.

4.2. Eléments de mémorisation

Nous distinguerons les registres, qu'ils soient adressables par la micro-instruction (registres généraux, scratch pad) ou non adressables (accumulateur, tampon, buffer), des mémoires dont l'adresse est une variable du compilateur et non une valeur d'un champ de micro-instruction.

Un registre apparaissant en partie gauche d'une affectation CASSANDRE est un puits de la hiérarchie. S'il apparait en partie droite c'est une source. Il possède les informations type, nom et taille, plus, si c'est un puits, un ensemble de doublets, comme un bus. Les conditions sont ici des conditions de chargement de registre.

Une mémoire se distingue d'un registre parce qu'il faut lui fournir une adresse pour faire une lecture ou une écriture. Une mémoire est un puits de la hiérarchie en écriture et c'est un opérateur booléen (tabulé) en lecture.

Les registres et mémoires sont représentés dans les figures par des rectangles, l'entrée de l'adresse dans une mémoire portant un trapèze.

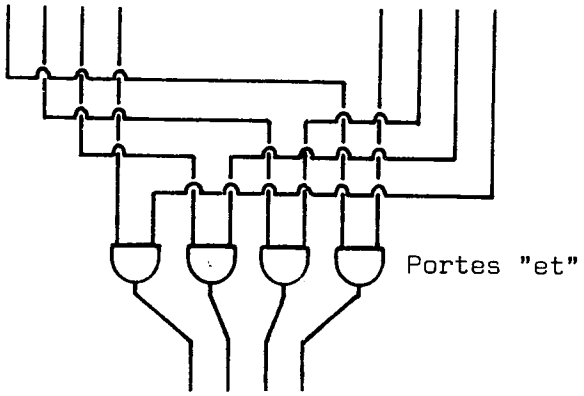
Lorsqu'un algorithme est recouvert par le schéma d'une machine, les puits d'une micro-instruction sont les sources portant le même nom de la micro-instruction suivante.

La représentation d'une variable dans un schéma d'incrément est la même que celle du registre dans un schéma de machine.

4.3. Opérateurs simples

Les opérateurs booléens répétitifs monadiques ou diadiques constituent les noeuds opératifs simples. Ils sont homogènes pour les tailles des entrées et sorties. Ils comportent type, nom, taille et liste ordonnée des noms de leurs entrées. Il y a un type par opérateur différent.

Exemple:



Porte "et" à quatre entrées

figure 3.3.

4.4. Opérateurs complexes

Ce sont les seules articulations du réseau, avec le delta que nous verrons plus tard, ayant un nombre de sorties non toujours réduit à un. Ils sont formés de type, nom, liste des noms de sorties avec leur taille, liste des noms d'entrées avec leur taille. Une entrée ne peut être qu'un bus ou une entrée de contrôle.

Exemple:

Additionneur avec retenue entrante et retenue sortante.

Les opérations simples ou complexes sont notés d'un cercle écartelé.

Le compilateur de microprogrammeur utilise les propriétés de régularité et commutativité des noeuds opératifs simples, ce qui explique la distinction.

Ces opérateurs apparaissent dans le schéma d'une micromachine aussi bien que d'un incrément.

4.5. Constantes booléennes

Ces noeuds ne possèdent pas d'entrée. Ils comportent nom, type, taille et valeur booléenne. On peut les considérer comme des opérateurs niladiques.

4.6. Valeurs immédiates

Il s'agit de valeurs passant directement de la hiérarchie de contrôle dans la hiérarchie opérative. Il est plus rapide de traiter spécialement les valeurs immédiates que de se ramener au cas d'un bus n'ayant que des constantes booléennes en entrée.

Les valeurs immédiates sont utilisées, soit pour introduire des constantes dans le chemin de données, soit encore pour commander le choix d'une fonction particulière d'un opérateur complexe.

Sur un dessin les constantes booleennes et les valeurs immédiates sont notées comme les opérateurs.

4.7. Entrées et sorties

Ce sont des signaux connectés à l'extérieur de la partie opérative. Ils ne constituent pas des bus mais ont les mêmes prérogatives que les registres sources et puits.

Ces signaux sont connus du compilateur de microprogramme par leur nom.

4.8. Confluences et deltas

Nous nommerons confluence, la juxtaposition de deux nappes de fils pour en faire une seule, et delta l'opération inverse, c'est-à-dire la séparation d'une nappe de fils en deux. Ces noeuds possèdent donc nom, type, taille des entrées et des sorties. Les entrées et sorties sont ordonnées. La somme des tailles des entrées est égale à la somme des tailles des sorties. Les noeuds se représentent par des triangles, la pointe du triangle indique l'entrée pour le delta et la sortie pour la confluence.

4.9. Retards

Les noeuds transmettent intégralement l'information qui les traverse, mais avec un certain délai. Nous interdirons l'utilisation de retards comme élément de mémorisation, leur usage est de permettre de calculer la durée d'un transfert et de tester ainsi sa validité.

5.- PROGRAMME CONSTRUISANT LE RESEAU

Le réseau représentant la partie opérative d'une machine est construit aisément à partir de la description en CASSANDRE de cette dernière. La technique habituelle est d'analyser la description CASSANDRE avec un suiveur syntaxique spécialisé, qui déclenche des fonctions sémantiques de construction. Quelques noeuds qui ne peuvent être construits en un seul passage, nécessitent un examen du réseau. Ce sont essentiellement les valeurs immédiates, confluence et delta. Certaines des tables qui ont permis de construire le réseau sont conservées pour faciliter l'analyse de ce dernier, les autres sont détruites. Les tables conservées sont essentiellement les tables des identificateurs, la table d'adresse des éléments de mémorisation et la table des conditions.

Le réseau n'est pas du tout construit pour être dessiné, le dessin est un autre problème, assez ardu [49]. Le réseau étant une hiérarchie dont on possède la liste des éléments extrêmes, nous en profiterons pour imprimer sous forme complètement parenthésée, toutes les arborescences conduisant aux puits (figure 5.1) et le tracé "à la main" et l'interprétation de la première de ces arborescences. Les noms externes sont ceux de la description donnée en CASSANDRE en annexe.

Un exemple de dessin, obtenu automatiquement sur table traçante, est également donné en annexe.

5.1. Exemple: Arborescences tirées de l'exemple du chapitre 2 (partie)

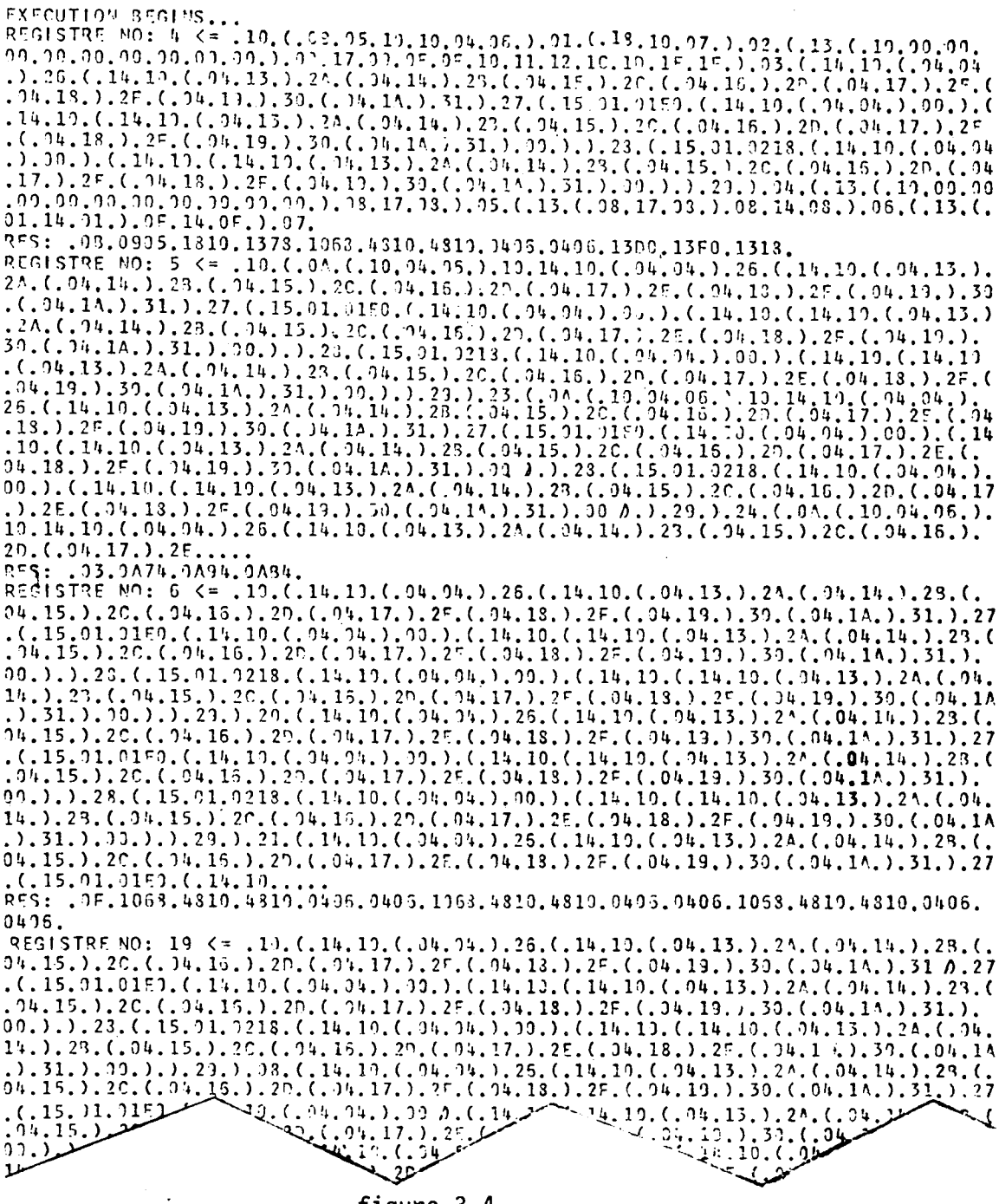


figure 3.4.

5.2. Table des codes internes utilisés

Elément insignifiant	00
Constante 0^*	01
Constante 1^*	02
Mélange des deux $(0^*\ 1^*)^*$	03
Registres sources et puits	04
Décalage	05
Rotation	06
Réduction	07
Négation	08
Mémoire	09
Adressage	0A
Ou logique	0B
Et logique	0C
Supérieur	0D
Inférieur	0E
Supérieur ou égal	0F
Inférieur ou égal	10
Egal	11
Différent	12
Confluence	13
Bus	14
Externe	15
Retard	16
Valeur immédiate	17
Entrée	18
Delta	19

figure 3.5.

5.3. Tracé d'une des arborescences à partir de sa forme parenthésée

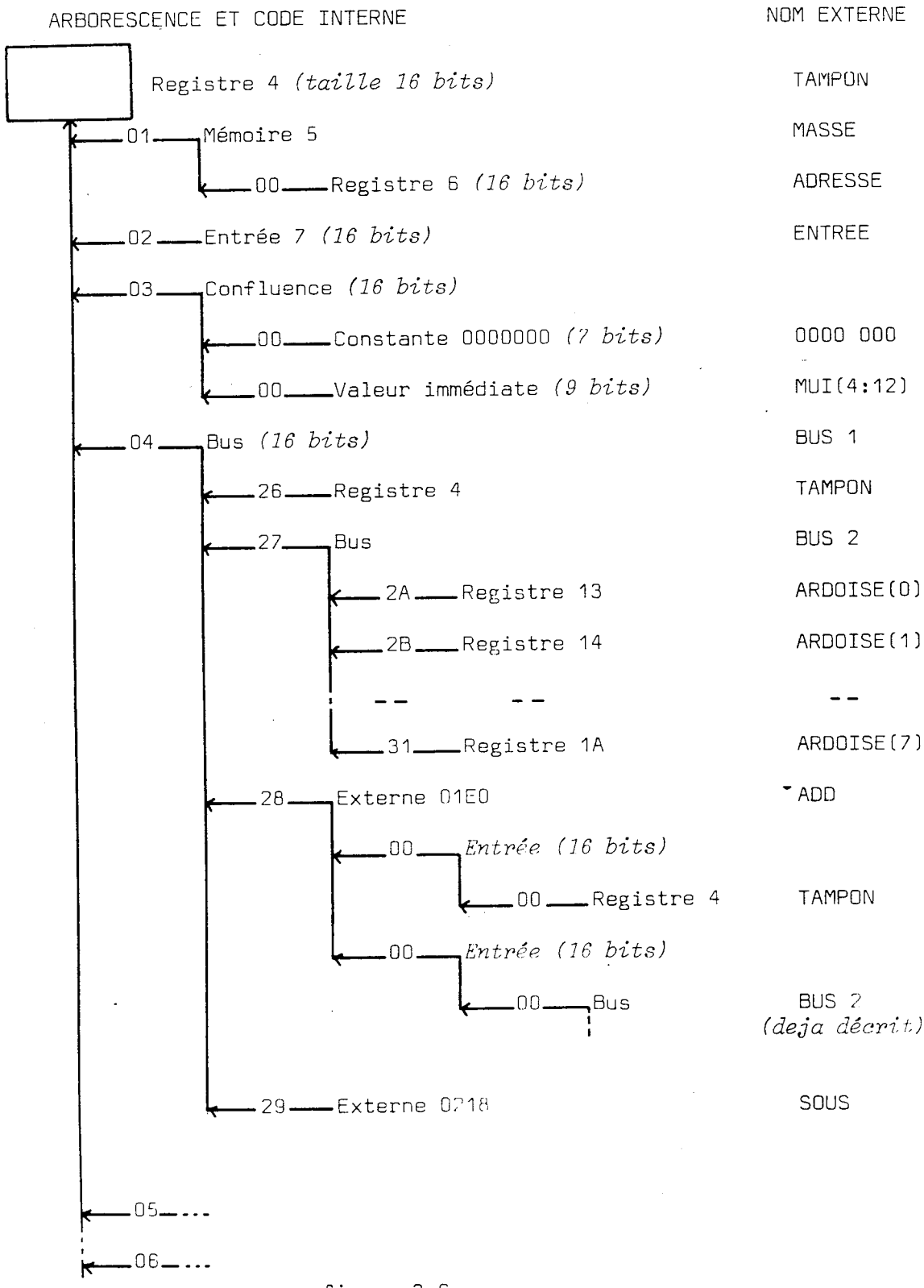


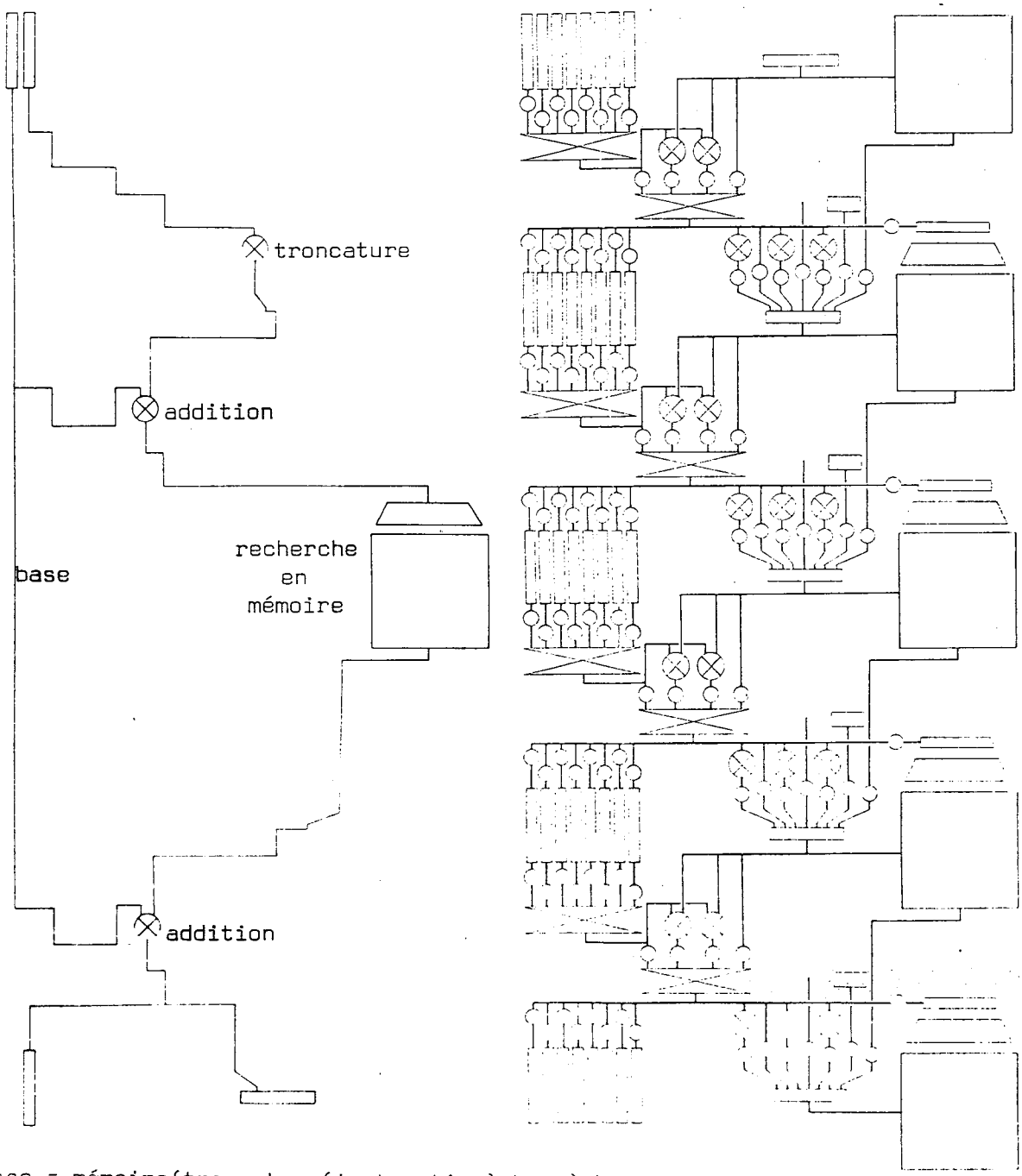
figure 3.6.

Remarque:

Il est facile de reconnaître dans cette arborscence la description de la micromachine, figure 2.9., à partir de laquelle elle est construite, ainsi que le graphe fonctionnel figure 3.1.

6.- EXEMPLE DE RECOUVREMENT

La schéma de la machine mis sous forme de hiérarchie (figure 3.4.), est utilisé pour recouvrir la hiérarchie représentant l'incrément étiqueté INDIRECT de l'exemple donné en annexe.



adresse = mémoire(troncature(instruction)+base)+base

figure 3.7.

QUATRIEME CHAPITRE

oooooooooooooooo

Au cours du précédent chapitre, un programme de construction de schéma de machine ou de schéma d'incrément a été présenté. Avec ces schémas ont obtient les micro-instructions exécutant un incrément en recouvrant ce dernier par une suite de schémas de machine.

Dans ce chapitre, et dans le suivant, un environnement permettant à l'algorithme de couverture de fonctionner est décrit.

En effet les schémas de machine et d'incrément seuls ne suffisent pas. Le recouvrement doit en outre se plier à deux impératifs: tenir compte de la micro-instruction et respecter les propriétés des éléments de mémorisation.

Plus le codage de la micro-instruction est compliqué, plus le sont les incompatibilités dans l'utilisation d'éléments ou d'ensembles d'éléments d'un même schéma de machine pour la couverture d'un incrément.

Les propriétés logiques des éléments de mémorisation intéressant la compilation sont:

- un élément de mémorisation ne contient qu'une seule variable à la fois.
- tant qu'elle n'est pas détruite, cette variable subsiste dans l'élément de mémorisation.

Nous appellerons parfois registres les éléments de mémorisation d'une micromachine.

Ces registres peuvent avoir une utilisation particulière:

Certains sont des "barrières temporelles", nécessaires à la synchronisation, c'est-à-dire à la régulation temporelle du flot d'informations dans la micromachine. Ces registres ne permettent pas d'utiliser, au cours d'une micro-instruction, la valeur qui y a été chargée au cours de la micro-instruction précédente.

Parmi les autres registres, c'est-à-dire ceux qui peuvent transmettre une information d'une micro-instruction à une autre, certains sont chargés (donc leur valeur précédente est détruite) sous le contrôle de la micro-instruction, alors que d'autres sont systématiquement chargés à chaque micro-instruction. Il est impératif, pour ces derniers, d'utiliser ou sauvegarder l'information qu'ils contiennent, si elle est intéressante, au cours de la micro-instruction qui suit immédiatement celle où elle a été produite.

Le compilateur de microprogramme n'accepte pas d'autre propriété dans le temps des registres que celle mentionnée ci-dessus. Cette restriction, qui n'est pas la dernière, vient s'ajouter à celles énumérées à la fin du premier chapitre.

1.- ELEMENTS DE MEMORISATION

Rien n'atteint dans le matériel d'un ordinateur, une diversité technologique aussi grande que les supports de l'information qui font appel à des phénomènes électroniques, électrostatiques, magnétiques statiques ou dynamiques, mécaniques, thermiques, optiques ou chimiques.

Il n'est pas question de demander au compilateur de microprogramme de gérer les modes d'accès à toutes les mémoires, ni de tenir compte de propriétés physiques dont la description en CASSANDRE est d'ailleurs délicate.

Les éléments de mémorisation seront classés en trois catégories:

- les registres obligés, c'est-à-dire les barrières temporelles, tampons ou accumulateurs, soit $B = \{B_i\}$ cet ensemble.
- les registres adressables par la micro-instruction, soit $R = \{R_i\}$ cet ensemble.
- les autres éléments de mémorisation, appelés mémoires et toujours considérés comme des périphériques de la micromachine.

Ces mémoires sont regroupées en classes d'équivalence pour la micromachine. Les diverses relations d'équivalences ont été étudiées [35]. Nous nous contenterons de l'équivalence triviale donnée par la description de la machine, soit $M = \{M_i\}$ l'ensemble de ces classes.

Dans l'exemple, MEMOIRE est une classe d'équivalence de 256 éléments.

2.- ETATS

Soit O l'ensemble des opérations arithmétiques, boolèennes et de restructuration faisant partie du langage source à compiler (exemple: addition, et logique, rotation, etc...).

Une expression finie est définie comme une chaîne finie d'éléments de $O \cup B \cup R \cup M$, respectant des règles de construction syntaxiques qui ne nous intéressent pas ici.*

Soit $\mathcal{O}'$ l'ensemble des expressions finies et $\mathcal{O} = \mathcal{O}' \cup \lambda$ ($\lambda \notin \mathcal{O}'$).

Soit n le cardinal de $B \cup R \cup M$, ordonné de façon arbitraire, c'est-à-dire le nombre des éléments de mémorisation distinguables au niveau de la micromachine, donc connus du compilateur de microprogrammes.
Nous appellerons état des éléments de mémorisation les éléments de $\mathcal{O}^n \cup \Lambda$, $\Lambda \notin \mathcal{O}^n$ [14]. La ième composante d'un vecteur de $\mathcal{O}^n$ est l'état de l'élément de rang i de $B \cup R \cup M$.

*Cette syntaxe est celle des expressions dans le langage intermédiaire présenté au chapitre 7 codées et mises sous forme préfixée.

Exemple d'état:

Soit R1, R2, R3, B1, B2 les registres d'une micromachine.

Soit α = R1, R2, R3, λ, λ
et β = R1 + R2, λ, R3, λ, λ

Dans l'état α, cette micromachine a ses trois premiers registres occupés, pour passer dans l'état β on ajoute (+) au premier registre le contenu du second, qui est d'ailleurs éventuellement détruit, et on conserve le contenu du troisième registre.

3.- ETATS SUCCESSEURS

On dit qu'un état succède à un autre s'il existe au moins une micro-instruction dont l'exécution complète permet de passer de celui-ci au premier. Nous donnerons une méthode pour construire la micro-instruction qui fait passer d'un état à un état successeur connaissant ces deux états.

S'il existe une micro-instruction ineffective, un état peut se succéder.

4.- ETATS ADJACENTS

Si l'exécution d'une micro-instruction se décompose en phases, à chacune d'elles va correspondre au moins un état intermédiaire entre états successeurs. Il existe donc au moins une suite (éventuellement vide) d'états adjacents entre deux états successeurs.

Remarque: Les états sont définis pour des micromachines synchronisées, où les registres sont chargés à intervalle fixe par des impulsions. Cette classe de machine se décrit aisément en CASSANDRE synchrone, le point de vue de ce langage ayant fortement influencé la conception de compilation de microprogrammes.

Les états adjacents ne sont qu'une facilité permettant de traiter les micromachines dont le cycle de base se décompose en un nombre fixé de phases. On peut d'ailleurs très facilement décrire une machine à cycles équivalente à une machine à phases, en utilisant la propriété des registres CASSANDRE d'être des maîtres-esclaves.

5.- ETATS PRINCIPAUX ET ETATS SECONDAIRES

Nous supposerons qu'il n'y a pas de recouvrement dans l'exécution des micro-instructions d'un même microprogramme. Cette exécution n'inclut pas le calcul de l'adresse et la recherche de la micro-instruction suivante, qui sont effectués par la partie contrôle de la micromachine. Cette restriction vise à exclure entre autres les micromachines pipeline ou barillet.

Nous appellerons états principaux, les états des éléments de mémorisation juste après l'impulsion de cycle de la micromachine, et secondaires les autres.

6.- ETAT INITIAL

Les états dont toutes les composantes sont soit λ, soit seulement un identificateur, à l'exclusion des expressions, sont des états initiaux.

Remarque:
Dans le cas où l'affectation des variables aux registres a déjà été faite, la ième composante ne peut être que λ ou l'identificateur du ième registre.

7.- ETAT FINAL POUR UN ENSEMBLE D'AFFECTATIONS

Soit une instruction affectant la valeur d'une expression à un élément de mémorisation de ième rang.

Alors un état est final pour cette affectation s'il porte l'expression comme ième composante. Les autres composantes n'interviennent pas.

Un état est final pour un ensemble d'affectations s'il est final pour chacune de ces affectations.

Cela entraîne évidemment que ces affectations se fassent à des éléments de mémorisation différents.

8.- CHEMINS D'ETATS ADJACENTS

Un chemin d'états adjacents est trivialement une suite d'états dont chacun, sauf le premier, est adjacent à son prédécesseur.

Reprenons la machine et l'opération données en exemple dans le deuxième chapitre, au paragraphe 2.3 (figures 2.4 à 2.8) pour illustrer quelques définitions données plus haut.

8.1. Schéma de la machine

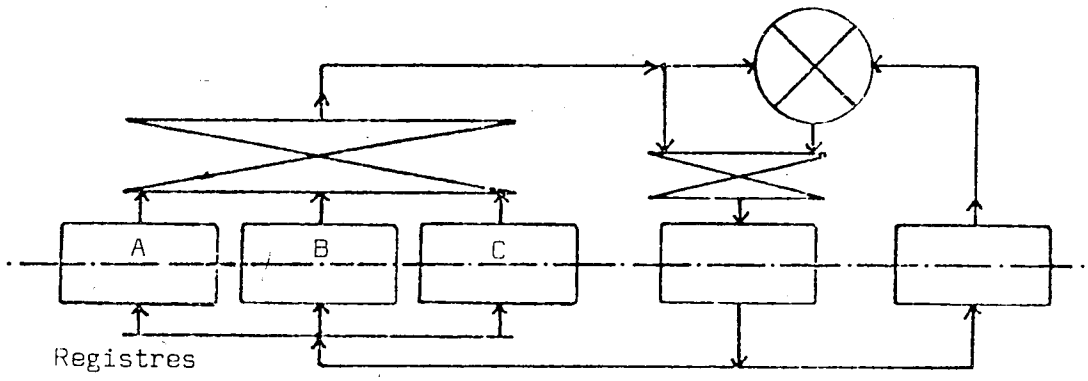


figure 4.1.

On veut réaliser A ← A+B+C.

8.2. Deux des chemins possibles

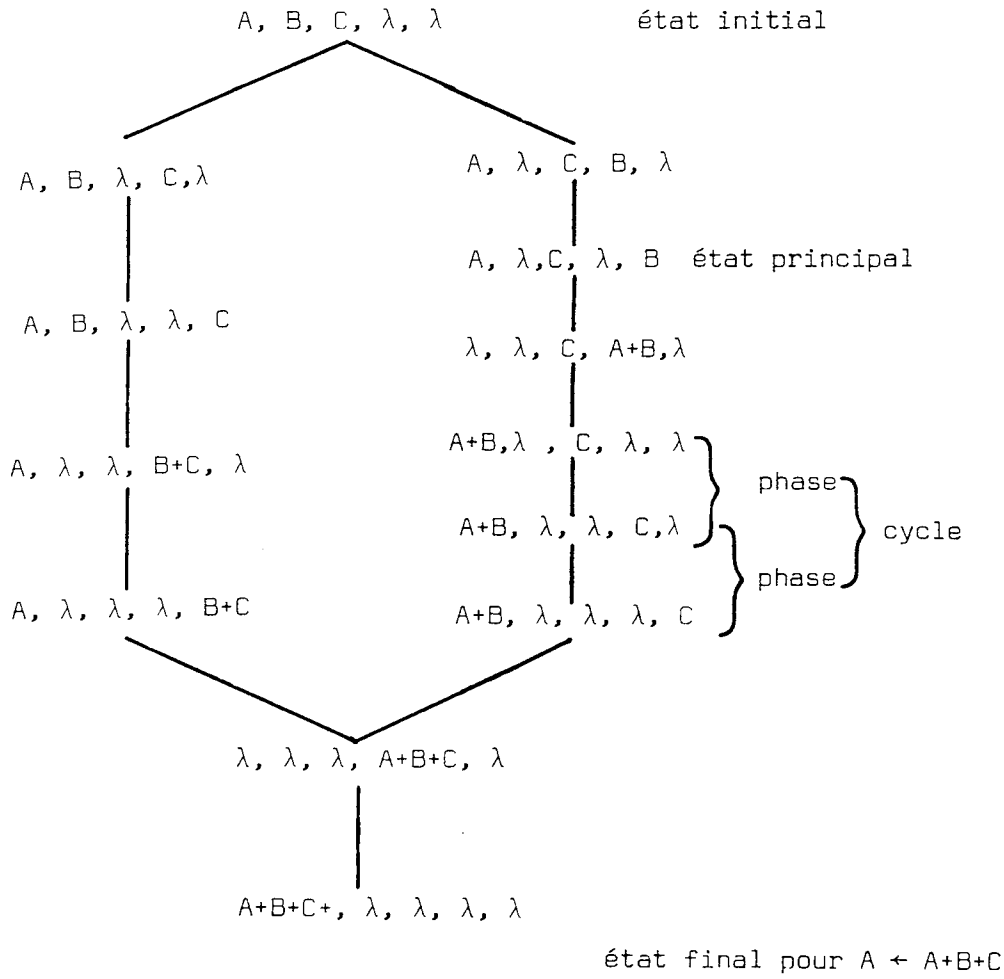


figure 4.2.

9.- INTERSECTION DE DEUX ETATS

L'opération interne d'intersection de deux états, notée $\wedge$, s'effectue composante à composante. Soit $E = e_1, e_2, \ldots e_n$ et $E' = e'_1, e'_2, \ldots e'_n$ et soit $E'' = E \wedge E' = e''_1, e''_2, \ldots e''_n$.

Les règles de calcul de E" sont les suivantes:

- Λ est absorbant pour l'intersection ($\Lambda \wedge E' = E \wedge \Lambda = \Lambda$)
- $E" = \Lambda$ si $\exists\ e_i$ et $e'_i \mid e_i \neq \lambda,\ e'_i \neq \lambda,\ e_i \neq e'_i$
- si $E" \neq \Lambda$ alors $\forall_{i \in 1,n}\quad e"_i =$ <u>si</u> $e_i \neq \lambda$ <u>alors</u> e_i <u>sinon</u> e'_i

Cette opération est commutative, associative et idempotente. On peut donc trivialement définir l'intersection d'ensembles d'états.

<u>Remarque</u>:

Signification de Λ : état physiquement impossible car contredisant une propriété logique des éléments de mémorisation.

Signification de l'intersection: une machine peut se trouver dans l'intersection d'un ensemble d'états si elle peut se trouver simultanément dans chacun des états de l'ensemble.

L'intersection d'états initiaux, si elle n'est pas Λ , est un état initial.

L'intersection d'états finals pour des affectations, si elle n'est pas Λ , est l'état final de l'ensemble de ces affectations.

10.- ENSEMBLE D'ETATS COMPATIBLES

Nous avons un ensemble d'états compatibles si l'intersection de ces états n'est pas Λ . La compatibilité n'est pas une relation d'équivalence.

11.- ETAT COMPLETEMENT INDETERMINE

Le vecteur dont toutes les composantes sont λ est un élément neutre pour l'intersection. Nous le nommerons Φ.

12.- CHEMINS D'ETATS COMPATIBLES

Un ensemble de chemins d'états de même longueur est dit compatible si les états sont compatibles à chaque niveau, c'est-à-dire les états initiaux, toujours compatibles, puis les états qui succèdent à l'état initial, et ainsi de suite.

13.- COMPATIBILITE ET PARALLELISMES

Un incrément peut être vu comme un ensemble non ordonné d'instructions d'affectations. Une condition nécessaire pour que ces affectations puissent s'exécuter en parallèle est qu'il existe pour chacune d'elles un chemin de l'état initial à l'état final tel que l'ensemble de ces chemins soit compatible.

Dans le chapitre suivant, nous verrons ce qu'il faut ajouter à cette condition pour la rendre suffisante en construisant un microprogramme.

Remarque:
Il faut entendre par parallélisme, le parallélisme synchrone.

CINQUIEME CHAPITRE

oooooooooooooooo

Pour que les éléments de mémorisation d'une micromachine passent d'un état èn un état successeur, la micromachine exécute un travail demandé par une micro-instruction. Il faut calculer la valeur, ou l'ensemble des valeurs possibles, de cette micro-instruction car le résultat de la compilation, première sortie, est une suite de micro-instructions.

Supposons que le travail que doit exécuter la micromachine pendant une micro-instruction soit parfaitement défini (nous verrons comment le définir au chapitre suivant), il reste deux problèmes à résoudre:

Le premier concerne le codage de la micro-instruction. Celui-ci peut être de complexité arbitraire et souvent la valeur d'un champ commande le découpage et la signification des autres champs de la micro-instruction.

Le deuxième est dû au fait que, lors de l'exécution d'une micro-instruction, seul le travail d'une partie de la micromachine est utile. Mais le reste du matériel est présent et fonctionne, car un circuit électronique délivre toujours une sortie. Il faut donc éviter que le travail de la partie de la machine inutile à l'exécution d'une micro-instruction, ne vienne parasiter le travail de la partie active.

On peut imaginer par exemple que certaine bascule de la micrmachine soit chargée à la fin de l'excéution de toute micro-instruction utilisant l'U.A.L. Si le déroulement du microprogramme nécessite la sauvegarde d'un résultat dans cette bascule, c'est parfait. Si la bascule est inutilisée, alors son chargement est inutile, mais non nuisible. Si par contre on veut conserver dans cette bascule la valeur qu'elle avait précédemment, ce chargement est désastreux.

Nous proposons donc un double mécanisme, qui dans une première phase fournit l'ensemble des valeurs de micro-instruction entraînant au moins l'exécution du travail demandé et, dans une deuxième phase ne garde, de cet ensemble de valeurs, que celles dont l'exécution ne détruit rien.

1.- ACTION ELEMENTAIRE ET POLYNOME CONDITION

La notion d'action élémentaire se retrouve dans [6, 8, 43]. Nous prendrons la définition de M. MERMET. Une action élémentaire est une action dont l'exécution ou la non exécution est commandée par la micro-instruction en cours.

Pour la partie opérative d'une machine, les actions élémentaires sont ramenées, par une modélisation convenable, à la sélection d'une entrée d'un bus ou au chargement d'un registre à partir d'une de ses entrées [43].

Pour chaque action élémentaire, un ensemble de valeurs de la micro-instruction commande l'exécution de cette action et toutes les autres la non exécution. En règle générale, seules les valeurs de certains champs de la micro-instruction sont importantes, les autres n'étant pas significatives pour l'action élémentaire considérée.

Exemple:

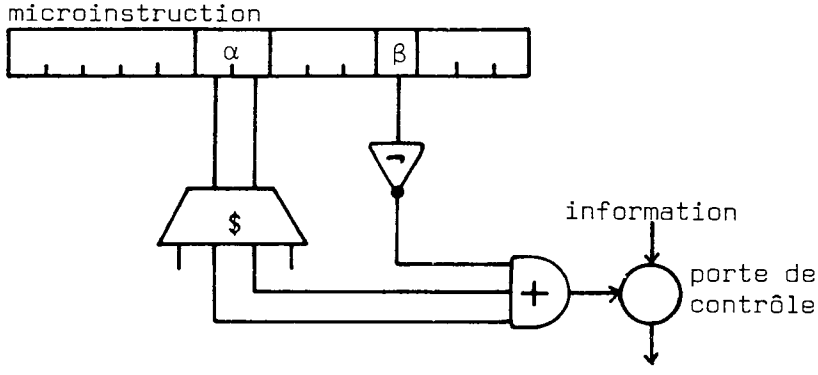


figure 5.1.

Les valeurs binaires 01 ou 10 du champ α ou la valeur 0 du champ β commandent l'ouverture de la porte logique représentée sur la figure.

Numérotons les positions binaires du micromot, ce qui donne un ensemble ordonné de variables booléennes. Pour toute action élémentaire il existe un polynôme booléen formé sur ces variables qui est vrai (vaut 1) si cette action doit être réalisée, et est faux (vaut 0) sinon.

Nous l'appellerons polynôme condition associé à une action

Exemple:
Si a_6, a_7, a_{11} sont les positions des champs α et β de l'exemple précédent, le polynôme booléen associé à l'action élémentaire précédente est:

$$P = \overline{a_6} \cdot a_7 + a_6 \cdot \overline{a_7} + \overline{a_{11}}$$

2.- MONOME CONDITION

Dans beaucoup de machines, le décodage est tel que les polynômes condition ne comportent qu'un seul monome. Cette constatation simplifiant le fonctionnement du compilateur, on s'y ramènera.

Exemple:

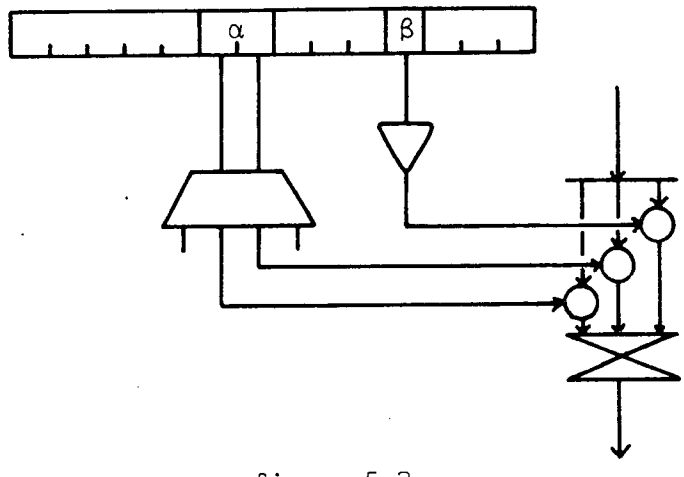


figure 5.2.
Schéma fonctionnellement équivalent à celui de la figure 5.1.

Nous verrons dans le chapitre 7 comment la transformation de la description de la machine, qui fait passer du premier au deuxième dessin, est réalisée avant la construction du graphe par un programme présenté au chapitre 3.

3.- ACTIONS ELEMENTAIRES COMPATIBLES

Deux actions élémentaires ne sont pas compatibles si le produit booléen de leurs monômes condition est nul [29], c'est-à-dire que leurs monômes condition sont orthogonaux. Nous verrons comment rendre cette condition nécessaire et obtenir ainsi une définition des actions élémentaires compatibles.

4.- MONOMES VRAIS ET FAUX

Nous appellerons "vrai" le monôme vide, c'est-à-dire ne comportant pas d'élément [4] et "faux" le monôme identiquement égal à 0. Le premier est élément neutre et le deuxième élément absorbant pour le produit.

5.- DEFINITION DES BLOCS

Soit P le produit cartésien de l'ensemble des états vus au chapitre 4 et de l'ensemble des monômes. On le munit trivialement d'une opération notée intersection ($\wedge$) définie à partir de l'intersection d'état et le produit booléen des monômes. Alors le couple état complètement indéterminé-monôme vrai est un élément neutre, et nous dirons qu'un couple est faux s'il comporte ou l'état Λ, ou bien le monôme faux. Soit F l'ensemble des couples faux et soit $\mathcal{P}$ l'ensemble des parties non ordonnées de P-F.

Nous appellerons bloc un élément de $\mathcal{P}$.

$\mathcal{P}$ est muni de l'union, au sens ensembliste, et de l'intersection définie comme suit:

Soit B1 et B2 deux blocs, alors B1$\wedge$B2 est l'ensemble des couples intersection de tous les couples de B1 avec chacun des couples de B2, moins les couples faux.

Ces deux opérations sur les blocs sont trivialement associatives commutatives, munies d'éléments neutres et distributives l'une par rapport à l'autre.

La propriété intéressante est:

- Le résultat d'un calcul sur des blocs ne dépend pas de l'ordre dans lequel les opérations sont exécutées.

6.- SIGNIFICATION INTUITIVE DE CES OPERATIONS

Les opérations que nous venons de décrire permettent de calculer progressivement un ensemble de valeurs de micro-instruction dont l'exécution réalise un travail donné. Appelons solution cet ensemble de valeurs.

Un couple état-monôme représente une ébauche de solution, comportant une ébauche d'état de la machine et une ébauche de valeur de micro-instruction.

Un bloc est une collection d'ébauches examinées simultanément (on espère que quelques unes de ces ébauches vont mener à la solution définitive).

Faire l'union de deux blocs, c'est vouloir examiner simultanément les solutions de ces deux blocs.

Faire l'intersection de deux blocs, c'est ne conserver que les solutions appartenant aux deux blocs à la fois.

L'opération d'intersection a été définie de telle manière qu'elle assure que les solutions respectent d'une part les propriétés logiques des éléments de mémorisation vues au chapitre précédent et d'autre part les contraintes de codage des micro-instructions.

Blocs et actions élémentaires

Reprenons l'exemple donné au début du chapitre:

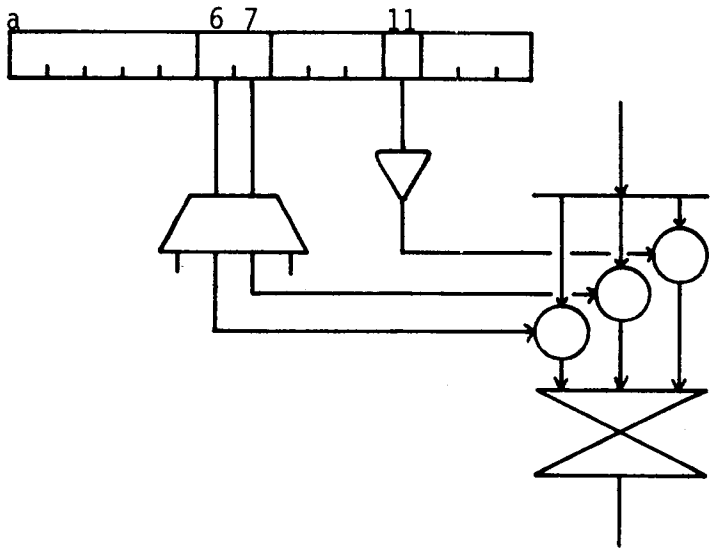


figure 5.3.

Le bloc correspondant à l'action élémentaire: passage de l'information à travers le bus est

$$(\Lambda, \overline{a_6}\ a_7) \cup (\Lambda, a_6\ \overline{a_7}) \cup (\Lambda, \overline{a_{11}})$$

Autre exemple:

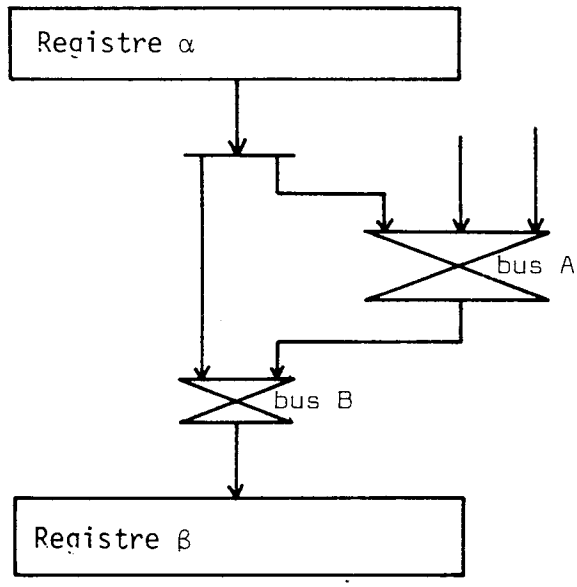


figure 5.4.

L'action: mettre le contenu du registre α dans le registre β se décompose en:

(occupation de β, sélection 1ère entrée de B) U
((occupation de β, sélection 1ère entrée de A) ∩
(occupation de β, sélection 2ème entrée de B))

7.- CONFUSION ET PERTES D'INFORMATION

Nous dirons qu'il y a confusion d'information si plusieurs entrées d'un bu sont ouvertes pour des informations différentes, et perte d'information si un élément de mémorisation est sensé conserver plusieurs informations différentes simultanément.

Quelques transformations du graphe représentant la machine en un graphe fonctionnellement équivalent [43] permettent d'éviter confusion ou perte d'information lors de la compilation de microprogramme.

- fusion des bus affluents :

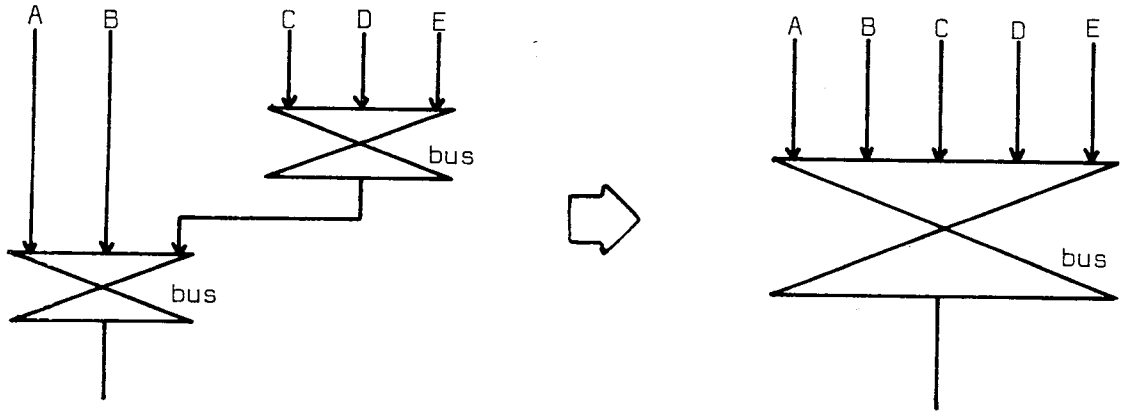


figure 5.5.

Remarque:

Nous avons adopté une représentation du bus dans la mémoire de l'ordinateur, qui rend triviale cette opération par manipulation de pointeurs. Les conditions d'entrée des bus sont recalculées.

- Regroupement des entrées physiques en entrées logiques. Les entrées physiques d'un bus provenant d'un même expanseur peuvent être sélectionnées simultanément.

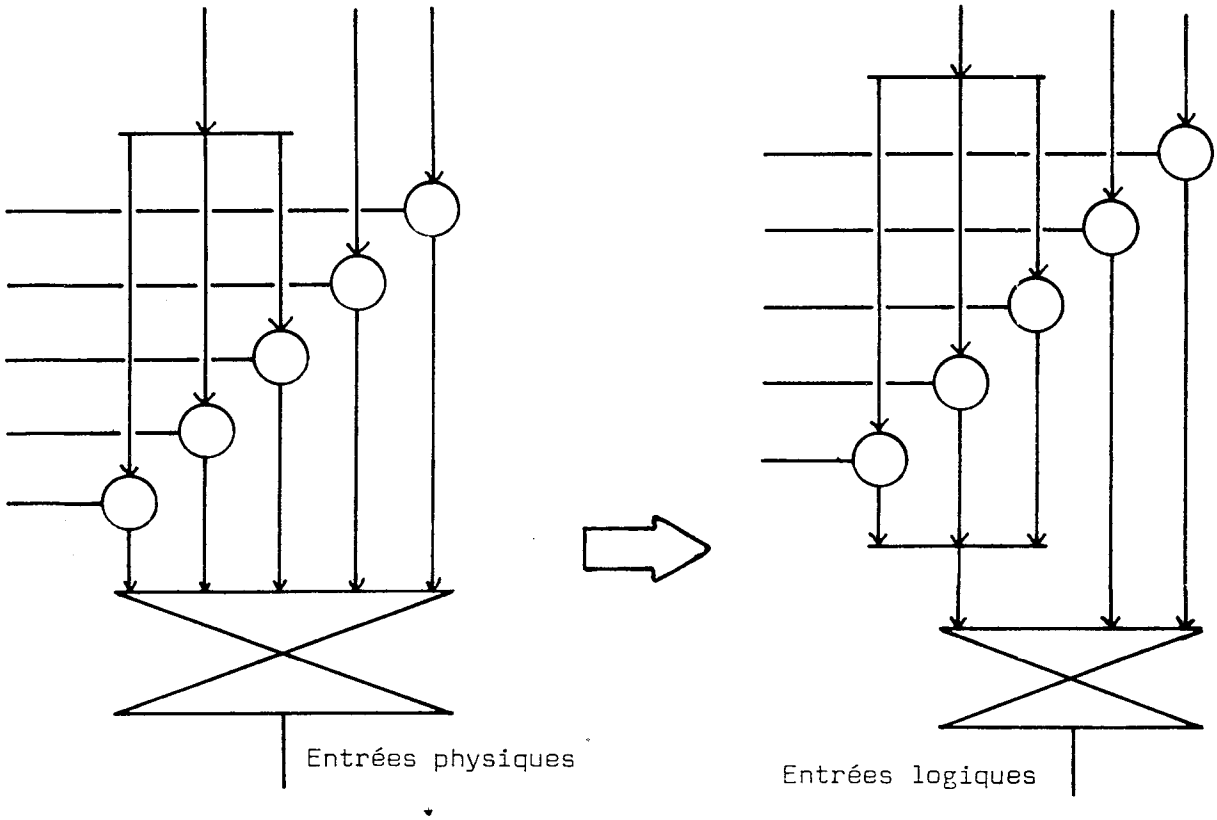


figure 5.6.

La condition associée à une entrée logique est la somme des monômes condition associée aux entrées physiques qui la composent. A une entrée logique est associé un polynôme condition.

- Rangement des polynômes condition des entrées logiques en classes maximales d'orthogonalité jusqu'à l'obtention d'un recouvrement. On ajoute alors le nombre de variables binaires nécessaires pour rendre ces classes orthogonales, ce nombre est le plus petit entier supérieur ou égal ou logarithme à base deux du nombre de classes.

Les variables booléennes sont ajoutées au contrôle. Il est bien entendu que ces variables binaires, ne figurant pas comme contrôle dans la description de la machine, sont internes au programme et n'apparaissent pas dans le résultat.

Exemple:

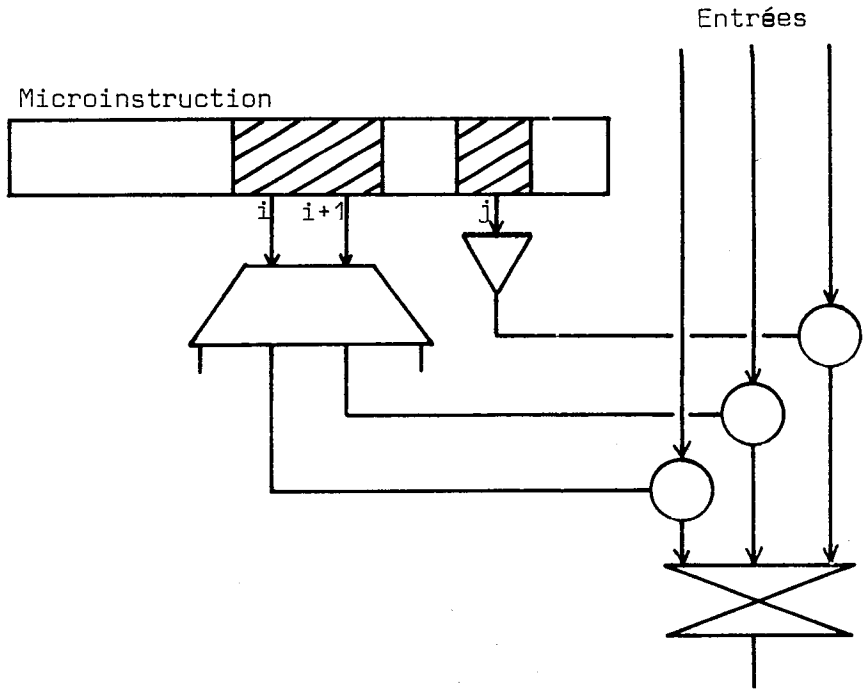


figure 5.7.

Les monômes sont:

$$\overline{\alpha_i}.\ \alpha_{i+1}\ ,\ \alpha_i.\ \overline{\alpha_{i+1}}\ ,\ \overline{\alpha_j}$$

qui se regroupent en classe :

$$(\overline{\alpha_i}.\ \alpha_{i+1}\ ,\ \alpha_i.\ \overline{\alpha_{i+1}})\ (\overline{\alpha_j})$$

Il faut donc ajouter une variable pour les distinguer, soit x. Les monômes deviennent:

$$\overline{\alpha_i}.\ \alpha_{i+1}.\ x,\ \alpha_i.\ \overline{\alpha_{i+1}}.\ x,\ \overline{\alpha_j}.\overline{x}$$

Ainsi complétés les monômes garantissent qu'il n'y a pas de micro-instruction telle que plusieurs entrées logiques de bus soient ouvertes simultanément.

Les mêmes transformations faites sur les entrées des registres garantissent contre les pertes d'information par chargement multiple.

Ces transformations justifient la définition suivante:

Deux actions élémentaires sont compatibles, c'est-à-dire

1/ il existe au moins une valeur de micro-instruction permettant l'exécution simultanée de ces deux actions,

2/ cette exécution simultanée n'entraîne ni perte ni confusion d'information,

3/ cette exécution simultanée n'implique pas qu'un registre contienne plusieurs variables à la fois,

si, et seulement si, le produit des blocs associés à ces actions élémentaires n'est pas nul (ou vide).

8.- ELIMINATION DES MAUVAISES SOLUTIONS

Le produit de bloc permet de trouver la solution, si elle existe, pour faire exécuter à une micromachine un certain travail. Il faut éliminer de cette solution les valeurs de micro-instructions qui détruisent le contenu de registre utile .

On se sert pour cela de l'état des éléments de mémorisation associé à chaque valeur de micro-instruction.

Trois cas peuvent se présenter pour les éléments de mémorisation:

- l'élément porte un λ et dans ce cas peut importe qu'il soit détruit

car c'est une place libre.

- L'exécution du travail exige le chargement de cet élément, ce qui est traduit par le fait que l'état de cet élément a changé et alors le mécanisme des intersections assure qu'il n'y a pas de perte d'information.
- Enfin, cet élément n'est pas libre, mais son état n'a pas changé au cours de l'exécution de la micro-instruction. Pour assurer la cohérence entre le vecteur d'état, qui indique qu'il n'a pas changé, et le codage de la micro-instruction, il faut s'assurer que celle-ci ne demande pas le chargement de l'élément. Cela se fait facilement en vérifiant que chacun des monômes condition de chargement est orthogonal à la micro-instruction.

Remarque:

Le cas de registres systématiquement chargés est inclus dans le cas précité.

9.- COMPARAISON ENTRE LA METHODE POLYNOMIALE ET LES AUTRES

Il n'y a guère que deux autres méthodes pour coder des micro-instructions:

- la méthode tabulée, utilisée dans les assemblages [6, 17], et qui à chaque action fait correspondre une seule valeur et un seul champ. Alors, deux actions sont incompatibles si leurs champs empiètent l'un sur l'autre.
- la méthode arborescente [16], qui permet de faire dépendre le découpage ou la signification de certains champs de la valeur d'autres champs situés plus haut dans l'arborescence. Elle impose bien sûr de coder la micro-instruction en parcourant les champs de la racine de l'arborescence jusqu'à une feuille.

La méthode polynomiale est plus puissante que celles vues ci-dessus, mais cette puissance n'est utile que dans des cas extrêmes, comme le montre l'exemple qui suit.

10.- EXEMPLE DE PROBLEME

Soit la machine microprogrammée:

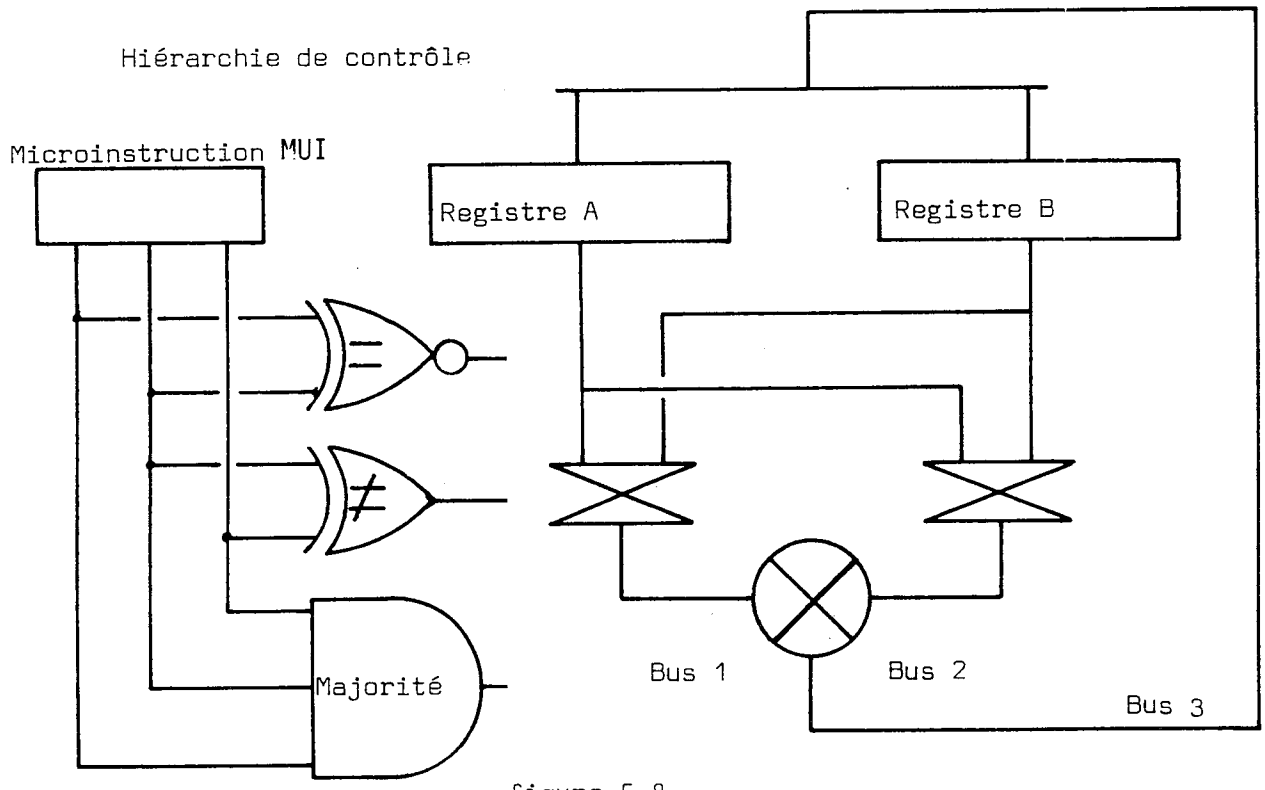


figure 5.8.

dont la description CASSANDRE est:

```
'UNITE' MICROMACHINE(H;)
  'HORLOGEMERE' H;
  'REGISTRE' A(1:4),B(1:4);
  'SIGNAUX' BUS1(1:4),BUS2(1:4),BUS3(1:4);
  'CONTROLE' MUI(1:3);
<H> 'SI' MUI(1)=MUI(2) 'ALORS'
  A<=BUS3 'SINON' B<=BUS3 ;
BUS1:='SI' MUI(2)#MUI(3) 'ALORS' A 'SINON' B ;
BUS2:='SI' MUI(1).MUI(2)+MUI(1).MUI(3)+MUI(2).MUI(3)
      'ALORS' B 'SINON' A ;
BUS3:=BUS1+BUS2 ;
```

figure 5.9

Si l'on veut réaliser A ← A+B, il faut que:

$(a_1 = a_2).\ (a_2 \neq a_3).$ majorité (a_1, a_2, a_3)

ce qui résoud en $a_1 = 1$, $a_2 = 1$, $a_3 = 0$.

Le compilateur de microprogramme fournit le bon codage, comme dans tous les cas où le décodage de la micro-instruction est combinatoire [52].

11.- EXEMPLE DE POLYNOMES CONDITION

Ce tableau donne les polynômes condition attachés aux portes logiques de la micromachine décrite en annexe et résume la figure 2.2 (page 30).

La première colonne indique un numéro de porte du graphe de cette micro-machine, donné figure 3.1. (page 49).

La notation du polynôme qui apparaît en deuxième colonne est inspirée par l'algorithme, donné dans [4], qui permet de faire en six instructions assembleur le produit avec simplification de deux monômes booléens ayant jusqu'à 1024 variables.

1	1..01000.. + .0.01000.. + ..001000..
2	1..01001.. + .0.01001.. + ..001001..
3	1..01010.. + .0.01010.. + ..001010..
4	1..01011.. + .0.01011.. + ..001011..
5	1..01100.. + .0.01100.. + ..001100..
6	1..01101.. + .0.01101.. + ..001101..
7	1..01110.. + .0.01110.. + ..001110..
8	1..01111.. + .0.01111.. + ..001111..
9	.....000..
10	.....001..
11	.....010..
12	.....011..
13	.....100..
14	.....101..
15	.....110..
16	.....111..
17	10
18	01
19	00
20	11
21	101.....
22	110.....
23	011.....
24	111.....
25	010.....
26	1..11..... + .0.11..... + ..011.....
27	1..10..... + .0.10..... + ..010.....
28	001.....
29	000.....

figure 5.10.

SIXIEME CHAPITRE

Il reste à préciser un certain nombre de points sur l'utilisation d'une méthode de couverture pour obtenir des micro-instructions.

- Avec quel noeud recouvrir quel noeud,
- Comment distinguer et utiliser les divers parallélismes apparaissant dans la machine et dans le programme à compiler.
- Quels critères utiliser pour ne conserver que les "bonnes" solutions de couverture parmi l'infinité des solutions possibles.
- Quel mécanisme de "contexte" associer à cette couverture pour rendre compte de l'influence essentielle de la couverture d'un ensemble de noeuds sur d'autres ensembles.

Le traitement de ces diverses questions interdit malheureusement de modifier ou d'ajouter des renseignements sur les schémas de la machine ou de l'algorithme.

Cependant, comme par construction ces schémas peuvent se décomposer en arborescences, on se ramene , sous certaines conditions, à un problème de recouvrement d'arborescences par les arborescences. La technique de résolution de ce problème rappelle celle de l'analyse syntaxique.

I.- RECOUVREMENT DES ELEMENTS

Les possibilités de recouvrement de chaque élément (articulation et arête) du graphe de l'algorithme par le graphe de la machine, sont résumées par le tableau suivant:

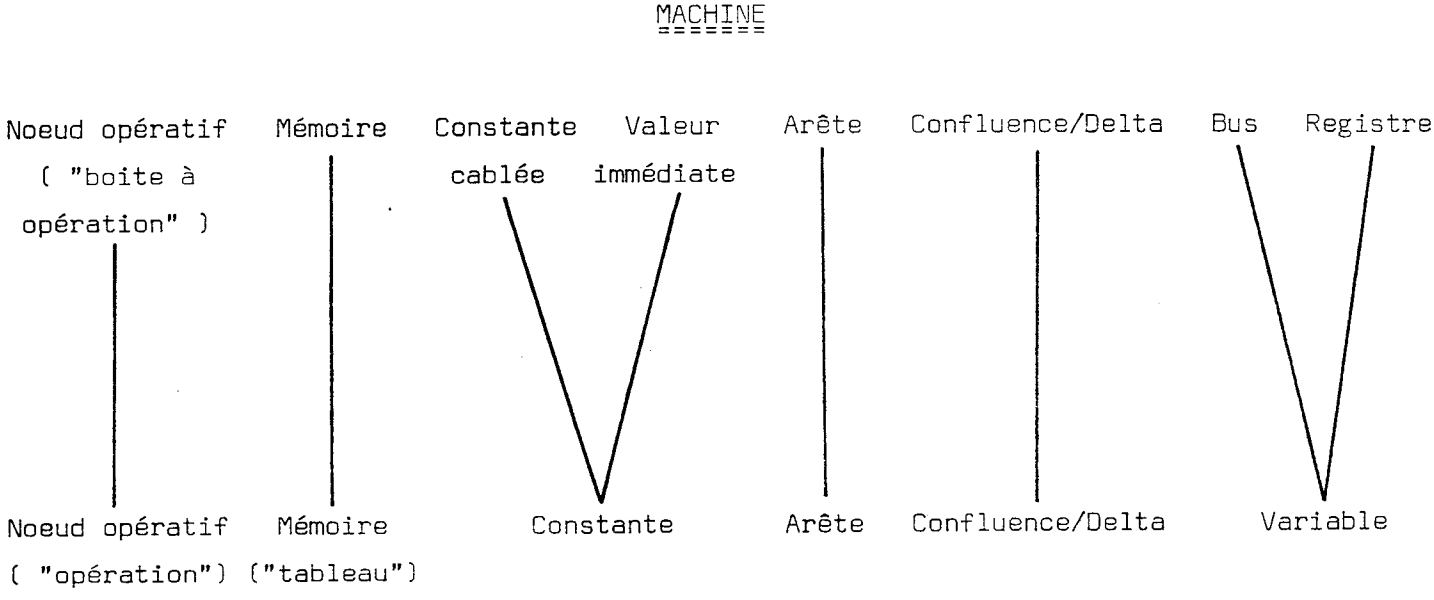


figure 6.1

Certains des recouvrements de ce tableau sont évidents: une opération apparaissant dans l'algorithme à compiler et mise sous forme de noeud opératif ne peut être recouverte que par un noeud opératif du graphe de la micro-machine qui représente la "boîte à opérations" qui peut exécuter celle-la (40). De même, une constante de l'algorithme est réalisée par une valeur immédiate venant d'une micro-instruction, ou par une constante câblée dans les micro-machines, les constantes câblées servent souvent de remplissage (une valeur immédiate est complétée à gauche par des zéros câblés), de bits entrants: zéro ou un, en cas de décalage et parfois l'unité arithmétique et logique délivre les constantes câblées de valeur entière, zéro, un, moins un ou autres dans la représentation entière en usage dans la micro-machine.

D'autres recouvrements sont moins immédiats: une variable apparaissant dans un programme sera toujours localisée par le compilateur dans un registre. Le compilateur peut également loger temporairement dans des registres des variables intermédiaires de l'algorithme qui n'apparaissent pas explicitement et qui sont représentées par des arêtes dans le graphe de l'algorithme. Un bus ou une arête du graphe de la machine recouvre une arête du graphe de l'algorithme.

Les noeuds "entrée de la micro-machine", "sortie de la micro-machine", "champs de la micro-instruction", sont connus dans l'algorithme à compiler par leurs noms dans la description de la machine.

Reste à examiner les opérateurs spéciaux que sont confluence et delta.

I.1. Confluences et deltas

Une arête du graphe d'une micro-machine représente une nappe de fils de la micro-machine réelle. Soit n le nombre de fils de cette nappe, qui peut alors véhiculer en parallèle des informations binaires de n bits au maximum. La variable véhiculée a donc une certaine structure. Il ne nous a pas semblé nécessaire de considérer des structures plus complexes que les tableaux à une seule direction ou vecteurs.

Deux questions se posent alors:

- sur les n fils d'une nappe, quels sont ceux qui transportent de l'information significative, c'est-à-dire utilisée, et ceux qui transportent de l'informations non significative.
- quelle est la permutation qui lie l'organisation d'une information en vecteur de bits à l'organisation d'une nappe en vecteur de fils.

Deux exemples concrets peuvent illustrer ces problèmes:

Exemple 1: Une machine ayant des chemins de donnée de 16 bits, traite un par un des caractères, chaque caractère tenant sur un octet.

Exemple 2: Sans qu'il y ait passage à travers une "boîte à opération" (ou noeud opératif), une restructuration d'une nappe de fils permet de faire la rotation d'une position à gauche [40]. Ces restructurations constituent des pseudo-opérations.

1.1. 1- Formalisation des pseudo-opérateurs

Soit $B = b_1\ b_2\ b_3 \ldots b_n$ une variable boléenne vectorielle. Une sous-variable de B est un ensemble ordonné quelconque de composants de B. Une sous-variable connexe de B est un ensemble ordonné de composants de B dont les indices sont consécutifs.

Exemple:

Sous-variable non répétitive — $- b_2\ b_3 \ldots b_n\ b_1$

sous-variable répétitive — $- b_2\ b_3 \ldots b_n\ b_n$

sous-variables connexes — $\{ b_2\ b_3 \ldots b_n \}$; $\{ b_1 \}$

Pour que ces questions puissent être traitées par le compilateur de micro-programme, c'est-à-dire ramenés au problème de couverture, les règles suivantes ont été dégagées:

- Toute variable est décomposée en un ensemble de sous-variables connexes qui seront couvertes simultanément (le contexte de l'analyse assure la cohérence de cette décomposition).

- A toute couverture d'une variable connexe de l'algorithme à compiler par une arête du graphe de la machine, sont associés quatre entiers positifs ou nuls:

  Deux sont associés au chemin de donnée:

  . cadrage C_1: numéro du premier fil significatif à gauche de la nappe représentée par l'arête du graphe de la machine.

  . nombre N_1: nombre de fils consécutifs significatifs à gauche du premier.

  Deux sont associés à la variable booléenne:

  . cadrage C_2: numéro du premier bit significatif à gauche de la variable booléenne vectorielle que représente l'arête du graphe de l'incrément.

  . nombre N_2: nombre de bits significatifs à gauche du premier.

Les seuls noeuds du graphe d'une machine ou des opérations de restructuration sont effectuées suivent.

- noeuds opératifs,
- confluences (fusion de deux nappes de fils),
- deltas (séparation d'une nappe de fils en deux).

Par exemple, un noeud opératif exécutant l'addition peut faire un décalage de une position binaire à gauche dans certains cas. Le compilateur ne peut reconnaître de telles fonctions, il ne sera pas tenu compte des propriétés de restructuration des noeuds opératifs.

La rencontre de noeuds confluence et deltas, que ce soit dans le graphe de la machine pour laquelle le compilateur fournit du microprogramme ou bien dans l'algorithme à compiler, fait calculer les entiers C ou N de l'analyse courante suivant des règles assez complexes.

Exemple:

Soit le schéma suivant:

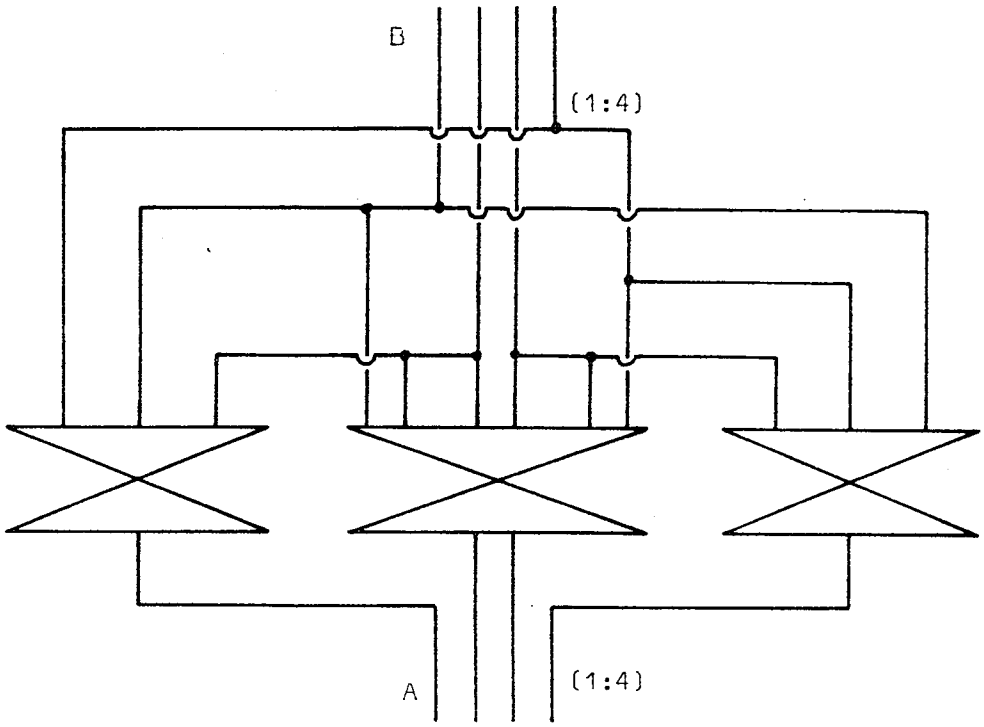


figure 6-2

où les traits représentent un conducteur.

Ce schéma se décrit en CASSANDRE:

```
'si'   COMM(1:2) 'alors'
(A(1) := B(4); A(2:3) := B(1:2); A(4) := B(3))
(A(1) := B(1); A(2:3) := B(2:3); A(4) := B(4))
(A(1) := B(2); A(2:3) := B(3:4); A(4) := B(1)) ( );
```

figure 6-3

Cet ensemble de trois multiplexeurs ayant une commande commune permet de réaliser les rotations de 1, 0 et -1 positions à droite, suivant les valeurs 00, 01, 10 de la commande, et peut donc recouvrir chacune des "arêtes" de l'algorithme dont le dessin suit:

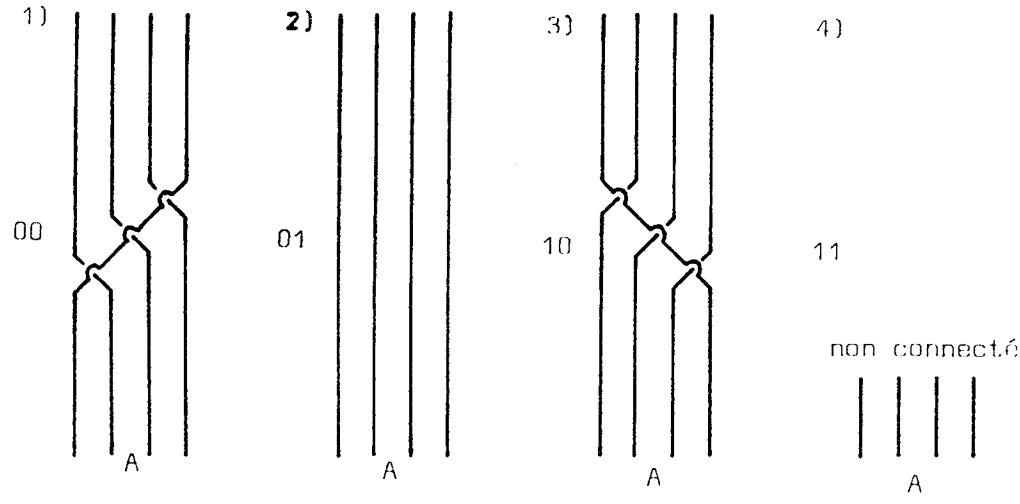


figure 6.4

Remarque:

- Les rotations et décalages exprimés en CASSANDRE sont décomposés en confluences et deltas lors de la construction des schémas.
- Ces noeuds sont très primitifs. L'expression du schéma (figure 6-3) nécessite 10 deltas, 2 confluences, 3 multiplexeurs, 10 coupures (figure 6-2).
- Les règles de calcul sur les entiers C et N rendent compte de toutes les propriétés de commutativité entre les opérateurs CASSANDRE de rotation, décalage et tous les opérateurs booléens (48).
- Il n'a pas été prévu dans cet exemple de valeur pour A, dans le cas où la commande vaut 11, et cela n'est pas nécessaire au bon fonctionnement du compilateur.

2 .- PARALLELISME

Sous le nom de parallélisme, on regroupe plusieurs notions qu'il est bon d'expliciter avant de montrer comment elles sont traitées par le compilateur de microprogramme.

2. 1. Parallélisme opératif

On regroupe sous ce nom le fait que les opérateurs (s'il y en a plusieurs) d'une micro-machine travaillent simultanément, dépendamment ou indépendamment les uns des autres, et également le fait que plusieurs registres de cette micro-machine peuvent être chargés sans l'influence d'un même impulsion d'horloge.

Deux opérateurs sont dépendants si une ou plusieurs sorties de l'un servent d'entrée à l'autre (il n'y a pas de boucle dans la micro-machine). Dans ce cas, un opérateur situé en aval ne commence à travailler efficacement que lorsque toutes les sorties des opérateurs situés en amont de lui sont stables. La phase durant laquelle un opérateur doit travailler efficacement c'est-à-dire délivrer un résultat à sauvegarder, est supposée durer suffisamment longtemps pour que les sorties des opérateurs puissent se stabiliser. La version actuelle ne tient pas compte des délais et retards (paragraphe III-4).

Deux opérateurs sont indépendants s'ils sont séparés par des barrières temporelles.

Dans le cas de la compilation de microprogramme, nous nous intéressons aux deux points suivants:

- les opérateurs fournissent un travail efficace au cours de la même phase.
- leur commande provient d'une même micro-instruction.

2.1.1 Exemple

Dans la machine décrite au deuxième chapitre, on peut "simultanément" exécuter une addition et une troncature du résultat de cette addition.

2 .2. Parallélisme de structure

Il s'agit ici de la structure des données traitées par la machine. L'efficacité du compilateur demande de considérer le plus globalement possible le groupe de fils qui constituent un chemin de données, cependant on peut

être amené à les décomposer en sous-groupe lorsque les opérations appliquées à ces sous-groupes sont différentes. Cette idée vient de CASSANDRE où on peut écrire indifféremment:

A(1:16) := B(1:16) ;

pour I=1 à 16 début A(I) := B(I) ; fin ;

ou toute autre combinaison.

Un chemin de donnée peut donc véhiculer en parallèle plusieurs sous-variables.

2.2.1 Exemple:

Dans la machine décrite au deuxième chapitre, les octets poids fort et poids faible qui chargent le registre tampon peuvent être traités de façon semblable ou différente. Dans un cas, on charge dans le tampon une donnée unique de 16 bits, et dans l'autre simultanément deux données en partie poids fort et partie poids faible.

2.3. Parallélisme synchrone

Dans le parallélisme synchrone, les variables peuvent changer de valeur, toutes ensemble, à des instants précis. Ce parallélisme, celui de CASSANDRE, qui sert de langage de description de l'algorithme à compiler.

Si dans un incrément, il y a plusieurs affectations de variables, le compilateur est chargé de générer la séquence de micro-instruction telle que les chargements de registres contenant ces variables, se fassent simultanément, ou a défaut, si la simultanéité n'est pas permise par la micro-machine, le compilateur produira une séquence de micro-instruction donnant le même résultat qu'une exécution simultanée.

2.3.1 Exemple:

On veut faire réaliser par la micro-machine décrite en première annexe les deux affectations suivantes:

ARDOISE (, 1) ← TAMPON,

TAMPON ← ARDOISE (, 1) ;

L'exécution de ces deux affectations permute les valeurs respectives de ARDOISE (, 1), qui est le premier mot de la mémoire locale ARDOISE et celle de TAMPON. Comme cette permutation ne peut se faire en une seule micro-instruction sur la micro-machine décrite, le compilateur va générer la

suite de micro-instructions :

- sauvegardant la valeur de TAMPON dans un registre général libre,
- chargeant TAMPON avec la valeur ARDOISE (,1),
- chargeant ARDOISE (,1) avec la valeur sauvegardée.

Remarque:-Si le parallélisme n'est plus synchrone, mais collatéral, le résultat des deux affectations est imprévisible.

- L'astuce de permutation de deux valeurs utilisant la disjonction n'est évidemment pas connue du compilateur. (Rappelons que $A \leftrightarrow B$ est équivalent à $A \leftarrow A \oplus B$; $B \leftarrow A \oplus B$; $A \leftarrow A \oplus B$;).

- Le mécanisme de gestion des éléments de mémorisation exposé dans le quatrième chapitre conduit de façon naturelle le compilateur à traiter le parallélisme synchrone.

- Dans le cas où il y a plusieurs affectations de variables à effectuer et qu'elles ne peuvent être effectuées simultanément, le compilateur examine toutes les séquences possibles de ces affectations.

2.4. Parallélisme d'examen des solutions

Le compilateur de microprogramme est amené en général à examiner plusieurs solutions pour un incrément donné pour les trois raisons suivantes:

. Si la micro-machine ne peut exécuter simultanément plusieurs actions d'un incrément, le compilateur est chargé de trouver un ordre d'exécution de ces actions tel que celle-ci soit le plus rapide possible. Le compilateur essaiera en parallèle les plus gros regroupements.

. Le compilateur doit choisir les éléments de mémorisation qui vont contenir les variables intermédiaires. Là aussi les possibilités qui se présentent seront examinées en parallèle.

. Il peut y avoir plusieurs valeurs de micro-instruction commandant une action élémentaire à exécuter par la machine. Le cas se présente surtout dans les décodages multiples. Là encore, ces différentes valeurs seront examinées en parallèle.

2.4.1 exemple

Le registre ADRESSE de la machine décrite en annexe est chargé si au moins une des trois conditions est réalisée:

Les bits 1, 4 et 5 de la micro-instruction valent respectivement 1, 1, 1 ou bien si les bits 2, 4 et 5 valent 0, 1, 1, ou encore les bits 3, 4 et 5 valent 0, 1, 1.

Le chargement de ce registre est indépendant des valeurs des autres bits de la micro-instruction.

3- OBTENTION DE RECOUVREMENT

1. Algorithme suiveur

Un algorithme suiveur permet d'examiner simultanément le graphe représentant la machine à microprogrammer et celui représentant l'algorithme à compiler. Si ce dernier peut être recouvert par un nombre fini de copies du graphe de la machine, suivant les règles indiquées au début de ce chapitre, une série de micro-instructions est générée. Elle constitue le microprogramme représentant l'algorithme à compiler.

Le fonctionnement de cet algorihtme suiveur est à rapprocher de celui d'un analyseur syntaxique descendant non déterministe. Pour reprendre la terminologie classique (22), il y a une fonction CHECK, qui permet soit de progresser, soit d'abandonner une règle, mais il n'y a pas de fonction DECIDE, qui permette de choisir une dérivation dans un ensemble de règles. La fonction DECIDE est remplacée par une fonction de prédiction, qui élimine un certain nombre de règles de cet ensemble, et par la dérivation en parallèle de toutes les règles non éliminées.

En réalité, le parallélisme des dérivations est évidemment simulé. Chaque règle sera suivie tant qu'elle ne peut être abandonnée avec la certitude qu'elle pourra être reprise, et, en principe, toutes les règles seront ainsi successivement abandonnées puis reprises de façon cyclique, jusqu'à ce que l'analyse soit terminée.

Ce parallélisme simulé est nécessaire car la grammaire représentant la micromachine est en général récursive. Tenter de suivre jusqu'au bout une règle, c'est risquer presque à coup sûr de s'enliser dans une boucle. Cette boucle correspond à un microprogramme de longueur infinie.

3.1. Pas de l'algorithme suiveur

Pour pouvoir abandonner une dérivation et la reprendre par la suite, il faut conserver l'état de l'analyseur. Or, au cours du calcul de la valeur d'une micro-instruction, cet état est dispersé dans la pile de récursion et divers indicateurs. Par contre, à la fin du calcul de chaque micro-instruction, l'état de l'analyseur est entièrement déterminé par l'état des éléments de mémorisation de la micromachine. La couverture du schéma de l'algorithme à compiler par des schémas de machine se fait donc pavé par pavé, chaque pavé étant un schéma complet de la machine.

Nous appellerons pas la pose d'un pavé. Chaque pas génère une micro-instruction dont la valeur est indépendante des valeurs générées avant ou après elle.

Dans la quatrième partie de ce chapitre, sera présenté le mécanisme d'enchaînement des pas.

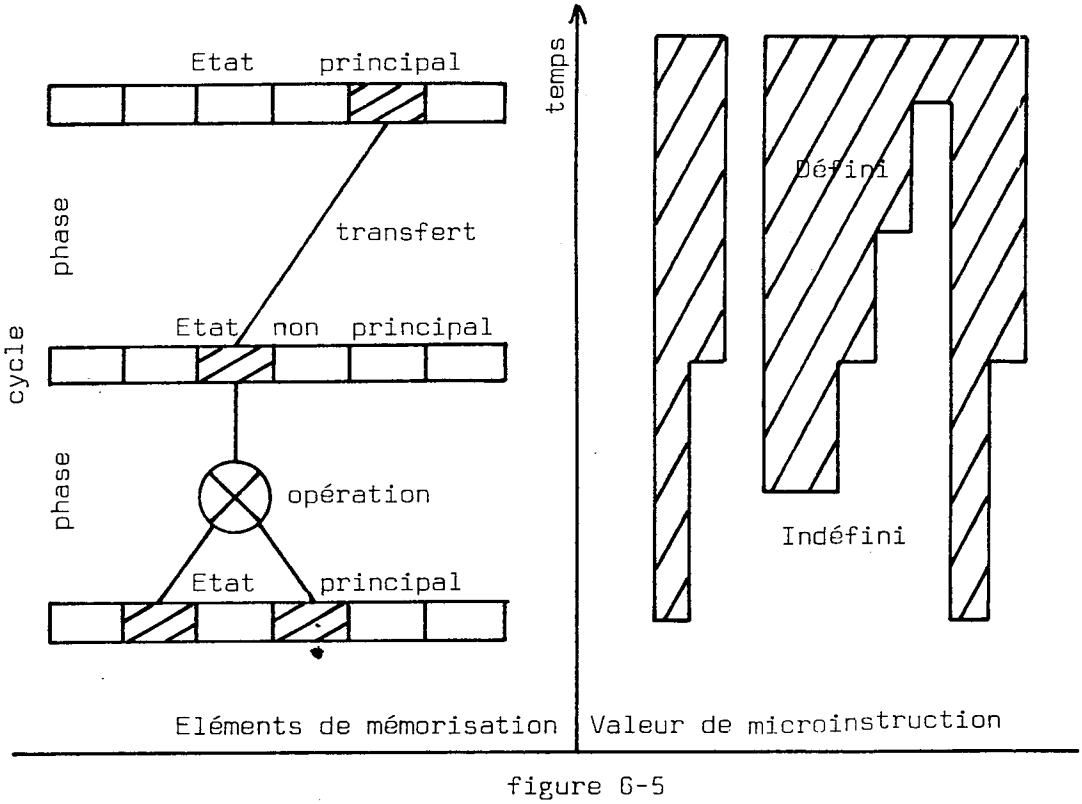


figure G-5

3.2. Analyse syntaxique sous contexte

Dans le chapitre 5, nous avons défini un bloc comme un ensemble non ordonné de doublets. Ces blocs seront rangés dans une pile et chaînés entre eux. Le bloc sommet de pile est toujours le contexte courant de l'analyse et le résultat d'une analyse est récursivement un contexte. Une deuxième pile sauvegarde des pointeurs sur les noeuds de la machine et les noeuds de l'incrément, qui correspondent aux règles syntaxiques et aux éléments de chaîne à analyser.

Analyse (Contexte, noeud, expression) → contexte

Rappelons que l'intersection de blocs est notée $\cap$ et l'union $\cup$.

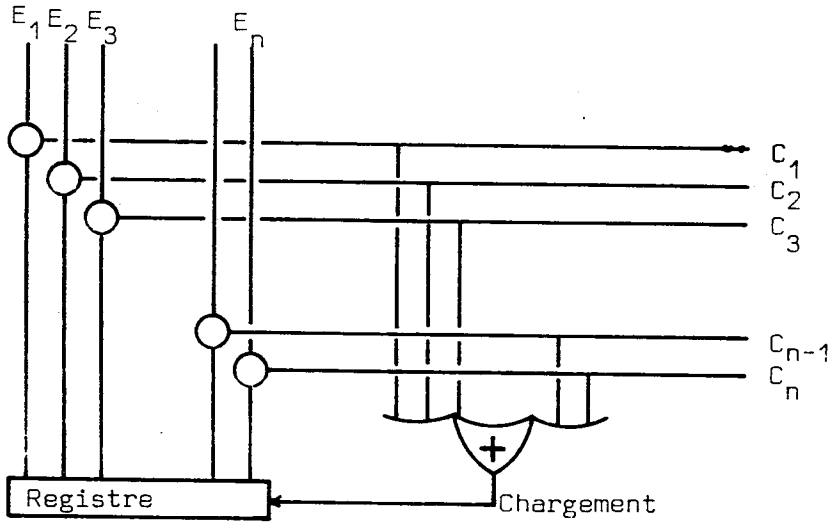
Voici cinq exemples de dérivation, choisis parmi les plus significatifs. La règle est un axiome, ensuite un bus de la machine puis un noeud opératif, ayant chacun n entrée et une sortie, puis un registre et enfin une valeur immédiate.

3.2.1 analyse d'un axiome

Les éléments de mémorisation de la machine peuvent être dans trois états. Ils sont, soit libres, soit occupés par une valeur qu'il ne faut pas détruire, soit à charger par le résultat du calcul d'une certaine expression. Cette expression est une arborescence de l'incrément. Dans ce troisième cas, cet élément de mémorisation constitue un axiome pour l'analyse, la chaîne à analyser est l'expression. Pour cette analyse, tous les éléments de mémorisation autres que l'axiome qui ne sont pas libres sont considérés comme occupés, le registre Axiome est libéré.

Cela constitue le contexte au début de l'analyse. La valeur de la micro-instruction contenue dans le contexte est entièrement indéterminée, puisque aucun calcul n'est encore fait.

Soit E_1, E_2, ... E_n, les n entrées possibles de ce registre, à chacune desquelles est attaché un polynome condition C_1, C_2, ...C_n qui permet, s'il est vrai, de charger le registre à partir de l'entrée correspondante.



Registre $((E_1, C_1), (E_2, C_2), \ldots (E_n, C_n))$

figure 6-6

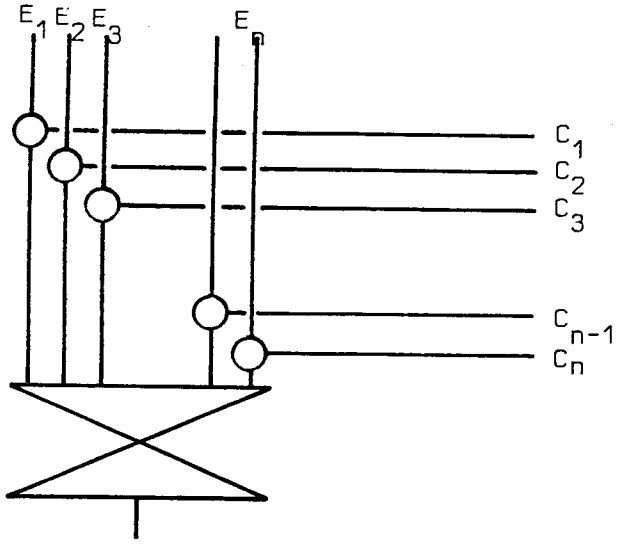
Analyse (Contexte, Registre $((E_1, C_1), (E_2, C_2), \ldots (E_n, C_n))$, expression)

$\rightarrow \bigcup_{i=1}^{n}$ Analyse (Contexte $\cap C_i$, E_i, expression)

Remarque: Dans ce cas, l'analyseur s'appelle récursivement.

3.2.2 analyse d'un bus

Soit un bus à n entrées $E_1, E_2, \ldots E_n$, à chacune desquelles est attaché un polynôme condition $C_1, C_2, \ldots C_n$ qui valide cette entrée.



Bus $((E_1, C_1), (E_2, C_2), \ldots (E_n, C_n))$

figure 6-7

Analyse (contexte, Bus $((E_1, C_1), (E_2, C_2), \ldots (E_n, C_n))$, expression →

$$\bigcup_{i=1}^{n} \text{Analyse}(\text{contexte} \cap C_i, E_i, \text{expression})$$

Ce qui traduit le fait que les entrées du bus sont à examiner simultanément avec un contexte rendu plus restrictif par le condition de cette entrée.

Remarque:

1) Certaines entrées, dont la condition est incompatible avec le contexte, ne sont pas à examiner. Cela est traduit par la règle suivante:

Analyse (FAUX, règle, incrément) → FAUX

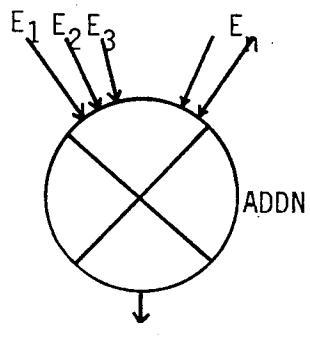
Dans ce cas, l'analyse de l'entrée correspondante est abandonnée. Rappelons que FAUX est élément neutre pour l'union.

2) L'analyse de bus conserve pour chacune des entrées le cadrage C et le nombre de bits significatifs N de la sortie du bus.

3.2.3 analyse d'un noeud opératif

Un noeud opératif est soit simple et appartient au vocabulaire CASSANDRE, soit complexe, et constitue un macrocomposant qui n'est pas une primitive pour le langage CASSANDRE, mais en est une pour le compilateur.

Soit un noeud opératif complexe de nom ADDN à n entrées ordonnées $E_1, E_2, \ldots E_n$; chacune de ces entrées a respectivement $b_1, b_2, \ldots b_n$ équipotentielles, et soit b_0 la taille de la sortie.



ADDN $(E_1, E_2, \ldots E_n)$

figure 6-8

Comme l'analyseur ne dispose pas de moyen de savoir quels sont les fils de S qui sont significatifs si tous les fils des entrées ne sont pas significatifs, l'analyse n'est faite que si tous les fils des entrées et des sorties sont significatifs, c'est-à-dire C = 0 et N = b_0.

L'expression étant mise sous forme arborescente, Racine de Expression a le sens habituel et $suite_i$ de expression désigne la $i^{ème}$ arborescence descendant direct de la racine.

Analyse (Contexte, ADDN(E_1, E_2, ...E_n) incrément) →
si C ≠ 0 ou N ≠ b_0 alors FAUX
sinon si racine (expression) ≠ ADDN alors FAUX
sinon $\bigcap_{i=1}^{n}$ Analyse (Contexte, E_i, $suite_i$ (expression))

Cette opération est le "CHECK" de l'analyse syntaxique descendante. Si l'incrément contient une opération que doit exécuter le noeud opératif considéré, alors on identifie les entrées de ce noeud opératif avec les opérandes de l'incrément. Dans le cas contraire, cette analyse est abandonnée en rendant un résultat FAUX.

Remarque:

1) Lorsque le résultat de l'analyse est FAUX, le contexte sommet de pile est détruit. Le travail nécessaire à sa construction est donc perdu.

2) Les opérateurs répétitifs booléens monadiques ou diadiques qui existent en CASSANDRE conservent les valeurs de C et de N pour leurs entrées.

3) Les mémoires en lecture et les constantes câblées sont considérées comme des noeuds opératifs respectivement monadiques et niladiques.

3.2.4 analyse d'un registre

L'occurrence d'un registre comme règle termine toujours une analyse. Soit REG le nom d'un registre de n bits.

Analyse (Contexte, REG, expression) →
si $N_1 = N_2$ et $C_1 = C_2$ alors
nouveau contexte
sinon FAUX

Le nouveau contexte peut rendre compte de trois cas:

a) l'expression se réduit à une variable et cette variable n'est pas dans REG, ou bien l'expression ne se réduit pas à une variable et REG est occupé. Dans ce cas, le nouveau contexte a la valeur FAUX et cette analyse est abandonnée.

b) l'expression se réduit à une variable et cette variable est dans REG. Alors, le nouveau contexte indique que REG est occupé, s'il ne l'était pas déjà, et ne peut plus servir à stocker des valeurs intermédiaires des calculs.

c) l'expression n'est pas une variable contenue dans REG et REG est libre. Alors pour valider cette analyse, le compilateur devra l'y amener. Le nouveau contexte indiquera que:

- REG est temporairement occupé (il sera libéré en servant d'axiome).
- REG doit recevoir l'expression.

Ces renseignements sont logés dans la partie "état des éléments de mémorisation" du contexte, et seront conservés pour fournir la liste des axiomes de l'analyse du pas suivant.

3.2.5 analyse d'une valeur immédiate

Rappelons qu'une valeur immédiate est un champ de la micro-instruction passant directement dans le chemin de donnée.

A un noeud valeur immédiate est attaché une liste ordonnée d'indices indiquant la provenance de chacun de ses bits.

L'expression à analyser doit être une constante booléenne. La cohérence des chemins de donnée de la micromachine assure l'égalité des tailles de cette constante booléenne et du champ valeur immédiate de la micro-instruction.

Exemple: Analyse (CONTEXTE, 010, VI(1, 5, 4)) →
contexte ∩ 0.. 01.....,Φ

Remarque:

1) la notation du polynôme est donnée en fin de chapitre V.

2) les valeurs immédiates sont communes en microprogrammation.

3) le même mécanisme de génération de polynôme condition en cours d'analyse permet éventuellement au programmeur de forcer la valeur de certain champ de la micro-instruction.

exemple: MUI (3:7) := 01011 ;

remarque:

Les vingt cinq types de noeud de la machine relevés à la fin du troisième chapitre confèrent à l'analyseur un fonctionnement plus ou moins équivalent aux cinq noeuds donnés en exemple. Il serait fastidieux de les détailler tous.

3. 3. Contexte au début d'un pas

Au début de l'analyse, le contexte ne rend compte que des éléments de mémorisation qui ne peuvent être utilisés, c'est-à-dire qui sont occupés définitivement, et de l'analyse à faire, c'est-à-dire des registres à charger avec des expressions. La micro-instruction elle-même est totalement indéfinie, ce qui signifie qu'il n'y a aucune contrainte sur ses valeurs possibles.

Imposer des contraintes initiales aux valeurs de micro-instruction permet de traiter de façon élégante certains types de codage, comme par exemple le décodage par mémoire morte.

Prenons l'exemple du MITRA 15 (32):

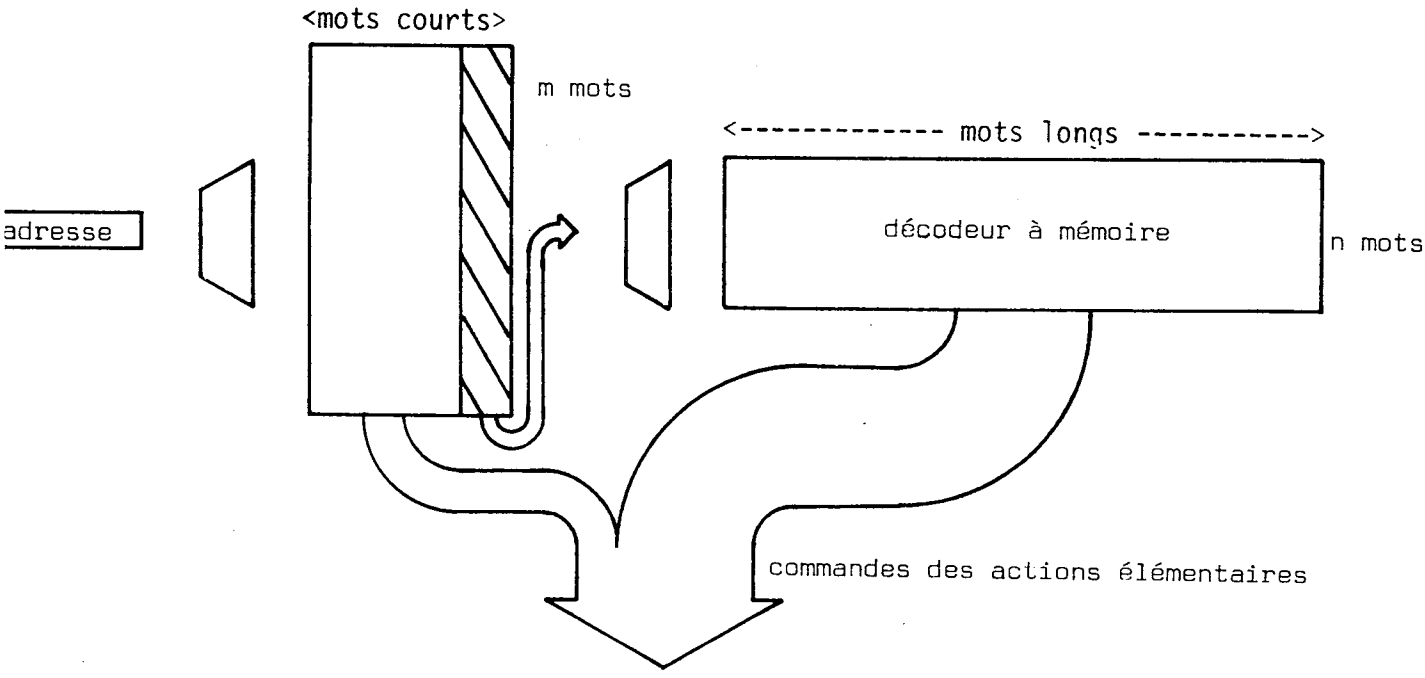


figure 6-9

La deuxième mémoire fait partie du décodage, donc de la description de la hiérarchie de contrôle de la machine. On se ramène au cas général grâce au modèle suivant:

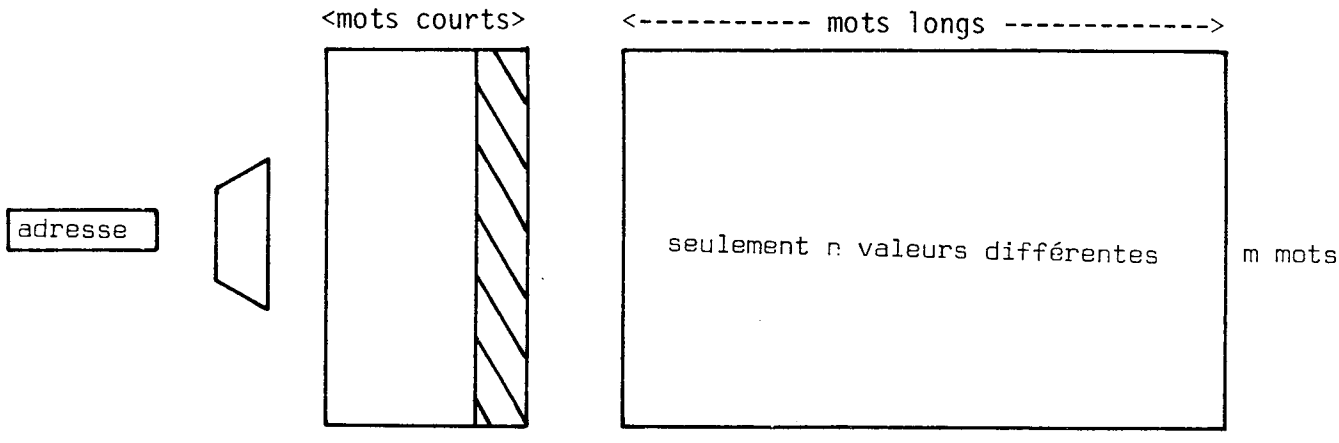


figure 6-10

La contrainte est: les valeurs d'un certain champ des microinstructions sont à choisir dans un tableau. Cette contrainte peut être mise dans le contexte de l'analyse au début de chaque pas. L'intérêt de cette méthode est de pouvoir changer la valeur de la mémoire morte de décodage sans changer la description de la machine.

3.4. Retards

Des opérateurs de retard existent en CASSANDRE asynchrone. L'analyseur syntaxique suivant les chemins de données de la machine, il serait aisé de cumuler tous les retards rencontrés lors d'un transfert pour vérifier qu'une micro-instruction est exécutable ou non. Soit t la durée d'un cycle et $\{\zeta\}$ les retards rencontrés, si $\sum_i \zeta_i$ t la micro-instruction n'est pas exécutable et la branche courante de l'analyse est abandonnée. Ceci n'est pas implémenté car ne présente pas beaucoup d'intérêt.

A ce propos, plus intéressant est le problème de description des opérateurs dont la sortie n'est pas disponible que n durées de micro-instructions après que les entrées soient stables, par exemple les multiplieurs câblés. Pour ramener ce cas au cas général, on utilise le modèle équivalent suivant:

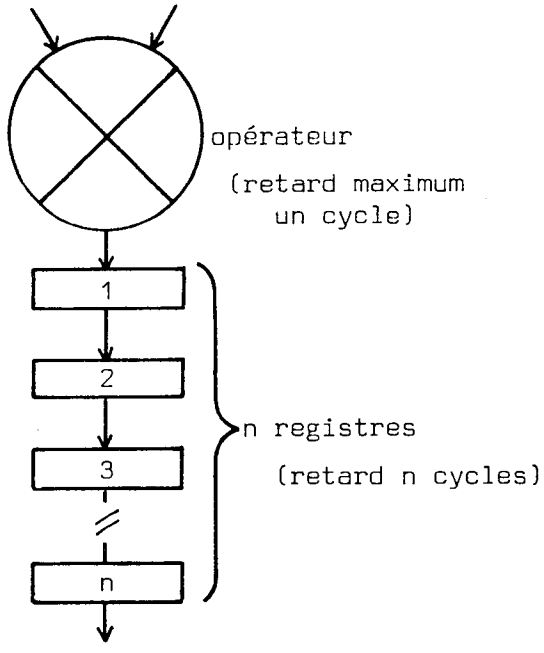


figure 6-11

L'utilisateur a la charge de décrire les n registres de façon que cela ne puisse fonctionner comme un "pipe-line" (chargement des registres mutuellement exclusif).

3.5. Enchaînement des pas de l'algorithme suiveur

Quand l'analyse d'un axiome se termine, un contexte se trouve au sommet de la pile. Ce contexte est un bloc, c'est-à-dire un ensemble de doublets. Chacun de ces doublets porte d'une part l'état des éléments de mémorisation en fin d'analyse et d'autre part la valeur de la micro-instruction qui permet de passer dans cet état.

Il y a en général dans un état des éléments de mémorisation plusieurs registres devant recevoir une valeur, donc occupés temporairement et qui constituent autant d'axiomes. Soit n leur nombre. Après analyse de chacun de ces axiomes, il y a au sommet de pile n blocs.

Un calcul d'intersections de ces n blocs détermine premièrement le plus grand ensemble d'actions exécutables en parallèle, deuxièmement les valeurs de la micro-instruction qui permettent cette exécution et troisièmement l'état des éléments de mémorisation qui résulte de cette exécution en parallèle.

Le mécanisme de vérification de la micro-instruction décrit dans le chapitre V vient alors éliminer les valeurs de micro-instruction commandant le chargement de registres à sauvegarder.

Le bloc résultat est alors chaîné au doublet qui l'a engendré, et chaque doublet de ce bloc servira éventuellement plus tard à initialiser une nouvelle analyse.

Lorsque plus aucun registre ne doit recevoir de valeur, c'est-à-dire qu'il n'y a plus d'axiome, l'état est final. En remontant le chaînage on retrouve la suite des valeurs de micro-instructions permettant d'arriver dans cet état final.

En bref, l'algorithme suiveur est un automate non déterministe qui fournit un ensemble de micro-instructions et d'états suivants en fonction de son état courant et d'un morceau d'incrément. Cet automate transforme progressivement un incrément à compiler en une arborescence de valeurs de micro-instructions.

3.6. Heuristique

Le peu de performance du mécanisme exposé ici paye d'une part les facilités de description de la machine et de l'algorithme, et d'autre part l'unicité dudit mécanisme face à des problèmes très différents, tels que codage de la micro-instruction, utilisation des éléments de mémorisation, exploitation des parallélismes du programme et de la machine et optimisation du microprogramme généré.

Dans le cas particulier où chaque incrément génère une unique micro-instruction, aucun de ces problèmes ne se pose. Alors le compilateur est un assembleur universel médiocrement performant.

Si par contre le schéma de l'incrément est une longue arborescence, ou plusieurs arborescences (hiérarchie), alors le compilateur tatonne en essayant des solutions et en les abandonnant parce qu'elles ne sont pas compatibles avec le codage de la micro-instruction ou les propriétés des éléments de mémorisation.

Une heuristique permet en outre de maximer les solutions examinées à chaque pas. La justification de cette heuristique est qu'un microprogramme a des chances d'être d'autant plus court que les opérations exécutées à chaque micro-instruction et qui ne sont pas des transferts sont plus nombreuses.

A chaque doublet et à chaque bloc, est associée une note. Le mode de calcul de ces notes est le classique MINIMAX (39). La note de l'union de deux blocs, qui contient les solutions appartenant à l'un ou à l'autre bloc, est le maximum des deux notes, et la note de l'intersection est le minimum. Seules les solutions les mieux notées sont conservées.

L'emploi de cette heuristique ne permet plus de garantir que la solution fournie par le compilateur est la meilleure.

3.7. Abandon

Lorsque la pile des contextes déborde, l'analyse est abandonnée. La solution est hors de portée du compilateur, soit parce qu'elle n'existe pas, soit parce qu'elle est trop complexe et ne peut être mémorisée.

3.8. Présentation de la première sortie

L'entrée programme du compilateur de microprogramme est présentée sous la forme d'un automate décrit en CASSANDRE, donc chaque état représente un pas de l'algorithme décrit.

La première sortie du compilateur de microprogramme se présente également comme un automate fournissant une valeur de micro-instruction pour chacun de ses états. Un pas de l'algorithme à compiler peut demander plusieurs micro-instructions pour son exécution, qui sont décrites dans un nombre égal d'états dont le premier sera étiqueté du nom du pas et les suivants ne seront plus étiquetés.

Le compilateur de microprogramme est transparent pour les ordres de séquencement qui se retrouvent dans la dernière micro-instruction générée pour chaque pas de l'algorithme.

4 - 1. OBTENTION DE LA DEUXIEME SORTIE

La deuxième sortie du compilateur est obtenue, à partir de la première sortie et de la description de la machine à microprogrammer, par substitution à la valeur de chaque micro-instruction de l'ensemble des actions élémentaires commandées par celle-ci.

Son exécution directe est beaucoup plus rapide que celle de la première sortie. Cela est dû à la suppression du décodage de chaque micro-instruction, ce décodage étant fait une fois pour toute.

4- 2. PRESENTATION DE LA DEUXIEME SORTIE

Cette deuxième sortie est beaucoup plus facilement lisible que la première. Les actions élémentaires qui ne dépendent pas de micro-instructions sont listées en tête.

La deuxième sortie permet de faire des mesures sur l'occurrence des actions élémentaires et peut servir à proposer un nouveau codage de la micro-instruction en utilisant les programmes développés par Madame de POLIGNAC (48).

Elle est essentiellement produite dans ce dernier but.

SEPTIEME CHAPITRE

Dans les précédents chapitres, on suppose avoir construit un schéma de machine, avec des noeuds distingués: registres, bus, etc... avec des conditions de chargement de registre ou de sélection d'entrées de bus mises sous forme polynomiale et des éléments de mémorisation distinguables au niveau de la micromachine, non regroupés.

Or, les descriptions de la machine à microprogrammer et de l'algorithme à compiler sont faites en CASSANDRE, langage qui a été précisé en vue de la simulation comme le révèlent les algorithmes de compilation et de simulation.

Plutôt que de construire directement le réseau d'une machine à partir de sa description, il a semblé plus aisé de définir un langage intermédiaire, de trouver les règles de traduction de CASSANDRE en ce langage, puis de construire le réseau à partir de la description traduite.

Le langage intermédiaire est un sous-langage de CASSANDRE, ce qui présente les avantages suivants:

- le langage intermédiaire est facilement compréhensible (un programme de traduction redonne un texte lisible).
- les symboles utilisés sont ceux de CASSANDRE.
- le système de programme CASSANDRE permet de vérifier l'équivalence entre la description source et la description en langage intermédiaire par leur comportement à la simulation.
- on peut utiliser ce langage intermédiaire, beaucoup plus simple, pour d'autres applications.

1.- SYNTAXE DU LANGAGE INTERMEDIAIRE

Une syntaxe simplifiée du langage intermédiaire est donnée par les règles suivantes:

<corps de description> ::= [<instructions>;]*

<instruction> ::= [<état>:]* ['si' <condition> 'alors'] <instruction simple>

<condition> ::= <contrôle> [,<contrôle>]*

<instruction simple> ::= (<horloge> <affectation>)| <branchement>

Remarque:

a) La syntaxe donnée ci-dessus est simplifiée en ce sens que pour obtenir la syntaxe complète du langage intermédiaire, il faut prendre l'intersection de la syntaxe du langage CASSANDRE [44, 48] et de cette syntaxe simplifiée.

b) Les déclarations ne sont pas touchées.

2.- EQUIVALENCES D'INSTRUCTIONS OU D'EXPRESSIONS CASSANDRE

Le langage intermédiaire doit avoir la même puissance que CASSANDRE, c'est-à-dire qu'à toute instruction CASSANDRE doit correspondre un ensemble d'une ou plusieurs instructions du langage intermédiaire qui lui soit équivalent.

L'équivalence d'instructions ou d'expressions a été bien étudiée dans [48] cependant l'équivalence de deux formes n'entraîne pas qu'il existe pour toute instruction appartenant à la première une instruction équivalente appartenant à la seconde. Nous noterons $\equiv$ l'équivalence au sens de [48].

2.1. Instruction et expression conditionnelles

On peut, par exemple, trouver l'équivalence de toute expression conditionnelle en instruction conditionnelle, la réciproque étant fausse.

Exemple: A := 'si' B 'alors' C 'sinon' D;
est équivalent à 'si' B 'alors' A := C 'sinon' A := D ;
et à 'si' B 'alors' A := C; 'si' $\neg$ B 'alors' A := D;

Remarque:

La dernière équivalence n'est pas vérifiée dans d'autres langages que CASSANDRE, si le calcul de la variable A influence la variable B.

2.2. Connexion d'externe

De même, on peut toujours mettre une connexion d'unité externe sous forme de signal unité, et pas l'inverse.

Exemple: ADD (ENTREE 1, ENTREE 2; SORTIE)

≡ ADD (*,;) := ENTREE1; ADD(,*;) := ENTREE2; SORTIE := ADD(,;*);

2.3. Constantes

Les constantes peuvent toujours s'écrire sous forme de tableaux booléens, l'inverse étant faux.

Exemple: * E1(3) ≡ 111

Pour chaque ensemble de formes équivalentes a été conservée une unique forme qui permet d'exprimer toutes les autres, l'ensemble de ces représentants constitue le langage intermédiaire. Nous appellerons transducteur le programme qui donne pour toute description une description équivalente en langage intermédiaire.

3.- ARITHMETIQUE ENTIERE EN CASSANDRE

L'arithmétique entière qui existe en CASSANDRE a, dans la version actuelle, une utilité strictement descriptive. Un programme écrit par Mr. FANTINO [18] donne pour toute description une description équivalente dans laquelle les "boucles pour" ont été exécutées et la valeur des expressions arithmétiques calculées.

Exemple:

forme primitive

```
'UNITE' DEDOUBL (E(1:5);S(1:10));
  'POUR' I=2 'A' 11 'DEBUT'
S(I-1):=E(I./2); 'FIN';
```

forme équivalente sans arithmétique

```
'UNITE' DEDOUBL (E(1:5);S(1:10));
              S(1):=E(1);
              S(2):=E(1);
              S(3):=E(2);
              S(4):=E(2);
              S(5):=E(3);
              S(6):=E(3);
              S(7):=E(4);
              S(8):=E(4);
              S(9):=E(5);
              S(10):=E(5);
```

figure 7.1.

4.- FONCTIONNEMENT DU TRANSDUCTEUR

Toute description en CASSANDRE peut se mettre syntaxiquement une forme arborescente orientée. Il suffit pour s'en convaincre de construire l'arbre d'analyse descendante de cette description avec la grammaire du langage CASSANDRE donnée par exemple dans [38, 44, 48] sous une forme arborescente ou développable en forme arborescente (infinie à cause de la récursivité du langage).

Un transducteur d'arborescence analyse et transforme une arborescence suivant le cycle:

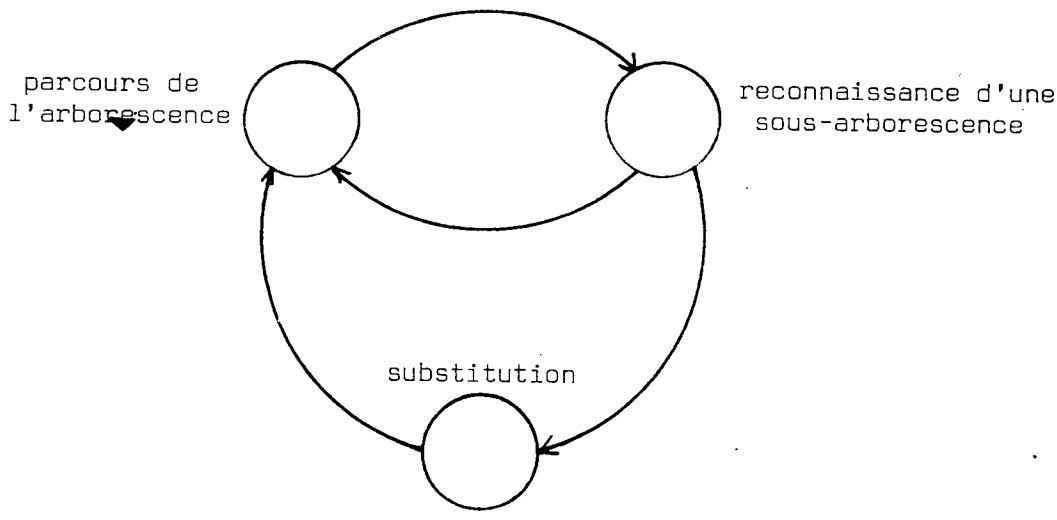


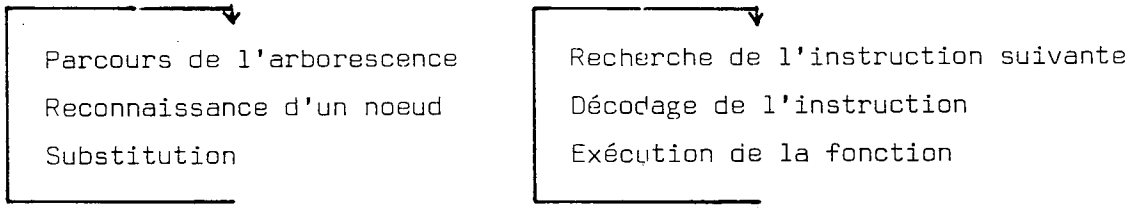
figure 7.2.

Le parcours d'une arborescence commence par sa racine, qui est le noeud le plus amont.

Cette arborescence étant renseignée (chaque noeud de l'arborescence ayant un nom), la reconnaissance consiste en une prise de décision en fonction des renseignements d'un noeud courant et éventuellement d'un certain nombre d'autres noeuds adjacents situés en aval sur l'arborescence.

La décision prise à l'étape de reconnaissance d'une part déclenche la substitution, qui consiste en une modification du renseignement du noeud courant, suivie éventuellement d'une restructuration de l'arborescence en aval de celui-ci, et, d'autre part, commande le parcours. Ce parcours peut être un abandon de la branche en cours d'analyse, une analyse du même noeud courant, dont le renseignement est modifié, ou bien l'analyse de tout ou partie des noeuds ou feuilles de l'arborescence situés en aval du noeud courant, c'est-à-dire descendant de celui-ci [12, 13].

Remarque:
Le fonctionnement d'un transducteur est semblable à celui d'un interpréteur d'instruction.



Dans le transducteur, les instructions et les données sont confondues.

L'organisation sous forme arborescente du programme parcouru par le transducteur n'est pas déterminante.

5.- REGLES DE TRANSDUCTION

Nous ne donnerons pas plus l'ensemble des règles du transducteur que nous n'avons donné dans le chapitre précédent l'ensemble des règles d'analyse. Toutes les règles de transduction ne s'expriment pas avec la notation linéaire non parenthésée utilisée par la suite pour faciliter la lecture. Il est très aisé d'ajouter des règles de transduction au programme transducteur à condition de prendre des précautions pour éviter les bouclages ou ambigüités.

Nous noterons avec des minuscules grecques les expressions, instructions ou listes d'instructions CASSANDRE, qui sont des sous-arborescences et par Λ l'arborescence ou sous-arborescence vide (réduite à sa racine). Nous noterons par une flèche → l'opération de transduction.

5.1. Règles de désimbrication

La définition syntaxique des instructions ou blocs d'instructions CASSANDRE étant récursive, une description peut avoir une structure emboîtée que le transducteur doit ne pas conserver. Les parenthèses "(" et "), permettent de délimiter des blocs d'instructions et les séparateurs ";" et "," permettent de créer des listes d'instructions.

5.1.1. Distribution d'horloges

$\langle\alpha\rangle\ \beta,\ \gamma;\ \rightarrow\ \langle\alpha\rangle\beta;\quad \langle\alpha\rangle\gamma;$

5.1.2. Décomposition des instructions conditionnelles

si α alors β sinon γ; → si α alors β; si $\bar{\alpha}$ alors γ;

si α alors (β;γ); → si α alors β; si α alors (γ);

si α alors (si β alors γ); → si (α). (β) alors γ;

si $\alpha+\beta$ alors γ; → si α alors γ; si β alors γ;

si 1 alors γ; → γ;

si 0 alors γ; → Λ

si α alors (Λ); → Λ

Remarque:

L'ordre des instructions d'une description peut être modifié, en CASSANDRE, sans modifier la signification de la description, ce qui n'est pas vrai dans d'autres langages purement séquentiels.

5.2. Règles de décomposition des expressions conditionnelles

Ces règles permettent de transformer les expressions conditionnelles (si-expressions) en instructions conditionnelles d'affectation (si-instructions).

(si β alors γ sinon δ) → si β alors (γ) sinon (δ)

α := ... si β alors γ sinon δ ...; →

si β alors α:= ... γ...; si $\overline{\beta}$ alors α:= ... δ...;

Remarque:

Chaque si-expression donne deux si-instructions et par combinaison n si-expressions au même niveau dans une expression peuvent donner 2^n si-instructions.

5.3. Règles de décomposition des expressions non-conditionnelles

Les êtres booléens de CASSANDRE et par extension les expressions booléennes sont des tableaux pouvant avoir pouvant avoir un nombre quelconque de directions. Sur chacune de ces directions, une expression peut être décomposée en sous-tableaux accolés ou concaténés, ayant les mêmes dimensions que l'expression initiale, sauf une [43]. Une succession de décompositions sur plusieurs directions permet d'isoler un sous-tableau quelconque d'un tableau qui est une expression. Nous le verrons, ces décompositions sont indispensables à la traduction en langage intermédiaire.

Une image "hardware" amusante permet d'illustrer cette décomposition: une expression peut représenter une carte logique sans boucle, réalisée en circuit imprimé et supportant un certain matériel (les opérateurs logiques) possédant un certain nombre de connexions d'entrées, de sorties, le tout interconnecté. Soulevons délicatement, avec une pince Bruxelles, une métallisation de sortie et décollons la. En supposant qu'elle ne casse pas,

elle entraîne avec elle en se détachant toutes les portes logiques qui lui sont soudées, ce qui entraîne d'autres métallisations et ainsi de suite. On a ainsi isolé tous les éléments, opérateurs ou signaux qui participent à l'élaboration du circuit de sortie que l'on tient avec la pince.

Cette opération de décomposition est formellement réalisée par le transducteur.

5.3.1. Descripteur d'une expression

Une expression CASSANDRE étant un tableau, on peut lui associer un descripteur qui est une suite ordonnée finie d'entiers positifs ou nuls, dont chacun, appelé dimension, est la différence entre une borne supérieure et une borne inférieure sur une direction. Les directions sont prises séquentiellement de gauche à droite. Le descripteur réduit se définit comme un descripteur dont on a supprimé tous les éléments nuls. Le descripteur réduit d'un scalaire est toujours vide.

Les descripteurs ont été introduits dans [15, page 19] pour permettre de vérifier la compatibilité de deux expressions CASSANDRE.

Deux expressions sont dites compatibles pour les opérations diadiques booléennes ainsi que pour les opérateurs d'affectation et de branchement si leurs descripteurs réduits sont égaux (les éléments des deux suites sont égaux deux à deux). Le descripteur du résultat est alors le descripteur du membre gauche de l'opération.

Deux expressions sont dites compatibles pour l'opération de concaténation si les éléments de leurs descripteurs sont égaux deux à deux, à l'exception des premiers éléments.

Remarque:
Ces définitions sont moins bonnes que celles de [43, pages A26 et A27] car elles conduisent à l'incohérence suivante:

Soient deux unités CASSANDRE :

```
00001  'UNITE' CONCAT1(A(1:3),B(,1:3);C(1:6));
00033              C:=A&(A+B);
00043  'UNITE' CONCAT2(A(1:3),B(,1:3);C(1:6));
00075              C:=A&(B+A);
                FIN D'EDITION

            MESSAGES DE VERIFICATION
***UNITE : CONCAT1
            -UNITE SANS ERREUR-
***UNITE : CONCAT2
U.S. NO. 00075     EXPRESSION NON COMPATIBLE
                FIN VERIFICATION
```

figure 7.3.

Les deux descriptions ne diffèrent que par la permutation des opérandes du ou logique qui est symétrique: (A+B) ≡ (B+A).
La première est juste, la seconde est fausse.

5.3.2. Opérateurs Tranche et Reste

Ces opérateurs ne font pas partie de CASSANDRE et sont introduits pour formaliser les décompositions.

Tranche (e, d, exp) où e et d sont des entiers non négatifs désignant l'épaisseur et la direction, et exp est une expression quelconque est un opérateur qui donne les e premiers éléments de l'expression exp sur le direction courante d. Cet opérateur s'exprime en CASSANDRE de la façon suivante:

Tranche (e,d,exp) ≡ * P|1||d| * D|e-E-1| * P|1||d| exp.
où E est le $d^{ième}$ élément du descripteur de exp.
L'opérateur Reste (e,d,exp) calcule le reste de l'expression exp quand on a oté Tranche (e,d,exp).

5.3.2.1. Tous les opérateurs de restructuration d'expression se décomposent en combinaison d'opérateurs Tranche et Reste.

Exemple:

* $D|n|exp \equiv$ si n>0 alors Reste $(n_{\text{mod } E+1}, 1, exp)$
sinon Tranche $((E-n+1)_{\text{mod } E+1}, 1, exp)$
où E est le 1er élément du descripteur de exp.

5.3.2.2. L'opérateur de réduction par rapport à un opérateur diadique booléen noté $\circ$ s'exprime récursivement par:

$/_\circ$ exp $\equiv$ si E = 0 alors exp
sinon Tranche (1,1,exp) $\circ$ ($/_\circ$ Reste (1,1,exp)).
où E est le 1er élément du descripteur de exp.

5.3.2.3. Le calcul formel du résultat de l'application de ces opérateurs aux variables ou valeurs booléennes est aisé.

Exemple:

Tranche $(e,d,A(i_1:s_1, \ldots, i_d:s_d, \ldots, i_n:s_n)) \equiv$

$A(i_1:s_1, \ldots, i_d:i_d+e, \ldots i_n:s_n)$.

où i_j et s_j désignent les bornes inférieure et supérieure de la $j^{ème}$ direction de la variable tableau A.

5.3.2.4. Les opérateurs Tranche et Reste se distribuent moyennant certains calculs avec tous les opérateurs CASSANDRE.

Exemple:

1/ Concaténation notée &
Tranche (e,d,exp1 & exp2) $\equiv$
si d $\neq$ 1 alors Tranche (e,d,exp1) & Tranche (e,d,exp2)
sinon si e<E+1 alors Tranche (e,1,exp1)
sinon Tranche (E+1,1,exp1) & Tranche (e-E-1,1,exp2).
où E est le 1er élément du descripteur de exp 1.

2/ Ou logique noté +

Tranche (e,d,exp1+exp2)

Tranche (e,d,exp1) + Tranche (e,d', exp2).

où d' est le rang du $n^{ième}$ entier non nul du descripteur de exp2 et n le nombre d'entiers non nuls parmi les d premiers entiers du descripteur de exp1.

Remarque:

a/ le calcul est valable pout tout opérateur diadique booléen à l'exception de &.

b/ pour ces opérateurs, la correspondance de direction se fait sur les descripteurs réduits. Le calcul qui précède exprime le passage du descripteur de exp1 au descripteur réduit, la correspondance avec le descripteur réduit de exp2 et le passage au descripteur de exp2.

5.4. Règles de complémentation

Ces règles sont nécessaires à la décomposition des instructions ou expressions conditionnelles comportant un "sinon" et à la mise sous forme polynomiale. Elles expriment les lois de DEMORGAN et autres règles de BOOLE. L'opération de complémentation est notée par un trait.

5.4.1. Règles de DEMORGAN

$$\overline{\alpha+\beta} \rightarrow \overline{\alpha} \cdot \overline{\beta}$$

$$\overline{\alpha-\beta} \rightarrow \begin{cases} (\overline{\alpha} \cdot \overline{\beta}) & \text{si les parenthèses sont nécessaires} \\ \overline{\alpha} + \overline{\beta} & \text{si les parenthèses ne sont pas nécessaires.} \end{cases}$$

Remarque:

La complémentation ne doit pas, en changeant les priorités, perturber l'ordre d'évaluation d'une expression: $\overline{\alpha+\beta.\gamma} \rightarrow \overline{\alpha} \cdot (\overline{\beta}+\overline{\gamma})$

5.4.2. Double négation et constantes

$\overline{-\alpha} \rightarrow \alpha$

$\overline{1} \rightarrow 0$

$\overline{0} \rightarrow 1$ ....

Notons que la complémentation se distribue par rapport à tous les opérateurs de restructuration ou commute avec eux.

5.5. Règles de simplification

Les lois d'absorbtion, de simplification de monôme et de polynôme, de décomposition d'opérateurs logiques dans la base et-ou-non permettent la mise sous forme polynomiale.

$\alpha=\beta \rightarrow \alpha.\ \beta + \overline{\alpha + \beta}$	
$\alpha\neq\beta \rightarrow \alpha = \overline{\beta}$	
$\alpha\cdot\alpha \rightarrow \alpha$	idempotence
$\alpha\cdot\overline{\alpha} \rightarrow 0^*$	la constante 0^* a le descripteur de α
$-\alpha \rightarrow \overline{\alpha}$	si α est une expression
$\alpha\cdot\ (\beta+\gamma) \rightarrow \alpha\cdot\beta + \alpha\cdot\gamma$	distribution
....	

6.- APPLICATION DES REGLES DE TRANSDUCTION

En raison de la relative complexité sémantique du langage, les règles de transduction sont nombreuses. Elles ont été distribuées entre une dizaine d'ensembles non disjoints, chaque ensemble de règles étant appliqué successivement, et ceci jusqu'à ce que plus aucune règle d'aucun ensemble ne puisse s'appliquer. Le partage pragmatique des règles en ensembles, dont certains ne contiennent qu'une seule règle, d'une part permet d'arriver au résultat assez rapidement et, d'autre part, facilite la programmation.

7.- EQUIVALENCE ENTRE DESCRIPTIONS

La simulation permet de vérifier l'équivalence entre la description source et la description en langage intermédiaire appauvri, la même séquence de signaux d'entrée et de commandes devant donner les mêmes résultats pour les deux descriptions. Le problème des signaux non connectés se pose alors. Un signal élémentaire de un bit ne peut prendre que deux valeurs à la simulation: 0 ou 1. La valeur indéfinie ϕ d'un signal non connecté en amont n'existe pas à la simulation et est soit 0 soit 1 d'une façon pseudo-aléatoire. Les règles de transduction ne garantissent pas que les valeurs de signaux non connectés soient les mêmes. Pour palier à cet inconvénient de taille il a été arbitrairement décidé que ϕ, valeur indéfinie, vaut toujours 0 (valeur initiale des signaux et registres) et le simulateur a été modifié à cet effet pour donner la version "Retour à Zéro" (RETZ).

8.-EXEMPLE

Un exemple de résultat de transduction est donné en annexe.

BIBLIOGRAPHIE

Graphes et recouvrement:

37 - 49 - 53

Graphes de machine:

8 - 43 - 44

Assembleurs et compilateurs de microprogramme:

1 - 6 - 16 - 33 - 45 - 51

Transducteur et analyseur syntaxiques:

12 - 13 - 22 - 23

Microprogrammation et structure de micromachines:

1 - 5 - 6 - 7 - 9 - 10 - 11 - 14 - 18 - 21 - 33 - 41 - 42 - 45 - 46 - 51 - 54

Description et application de CASSANDRE

2 - 7 - 15 - 38 - 43 - 44 - 48 .

1) ANCEAU F.
Compilation de Microprogrammes
Séminaire de logique, ENSIMAG, mars 1969.

2) ANCEAU F.
Application du langage CASSANDRE à la microprogrammation
GRENOBLE, Workshop on microprogramming, Juin 1970.

3) ANCEAU F, COUTURIER, DOUSSY J, PERRON F
Compilation et simulation du langage CASSANDRE
Rapport de contrat CRI, Industrialisation de CASSANDRE.

4) BENZAKEN C.
Contribution des structures algébriques ordonnées à la théorie des réseaux
Thèse d'état (pages 114/116), mars 1968.

5) BERNDT H.
Fonctional microprogramming as a logic design aid
I.E.E.E. Trans. on computer, octobre 1973.

6) BLAIN J.
Génération automatique de microprogrammes optimisés à partir d'une description algorithmique des macrocodes et d'une description formelle de la structure câblée.
R.A.I.R.O, septembre 1972
Contrats CRI 70.98 et 72.58.

7) BOGO G, GUYOT A, LUX A, MERMET J, PAYAN C
CASSANDRE and the computer aided logical system design
IFIP 1971 (pages 1056/1065).

8) BOULAYE G.
Graphes controles et instructions
R.I.R.O., 1971.

9) BOULAYE G.
La microprogrammation
DUNOD, Paris 1971.

10) BOULAYE G, MERMET J eds,
International advanced summer institute on microprogramming
HERMANN, Paris 1972.

11) CHATELIN M, HANCZAKOWSKI A.
Minimisation d'une mémoire morte de microprogramme
Rapport de contrat DRME, octobre 1970.

12) CHAUCHE J.
Arborescence et Transformation
Documentation G.E.T.A., juin 1972.

13) CHAUCHE J.
Transduction d'arborescences. Application aux manipulations de formule sur ordinateur.
Thèse de 3ème cycle, avril 1971.

14) COFFMAN E.G.
A formal microprogram model of parallelism and register sharing
University of Newcastle upon Tyne, février 1970
GRENOBLE WORKSHOP ON MICROPROGRAMMING , Juin 1970

15) DOUSSY J.
Edition et vérification de CASSANDRE
ENSIMAG, 1970.

16) DOUSSY J.
Réalisation d'un assembleur de microprogramme
Contrat CRI 70.090, juin 1972.

17) DROUET P.
Etude et réalisation d'un assembleur de microprogramme
Rapport de contrat IFP-ENSIMAG, 1973.

18) ECKHOUSE R.H.
A hight level microprogramming language
I.E.E.E. trans.on computer, juillet 1971 (pages 783/794).

19) FANTINO Y.
Transcription automatique d'algorithme en vue de leur réalisation hardware
ENSIMAG, 1974.

20) GERACE GB., GESII G.
A method for designing a digital system as sequential network system
Pise, juin 1967.

21) GLUSKOV VM.
Automata theory and microprogram transformation
Cybernetics 1965, (pages 1/9).

22) GRIFFITHS M.
Analyse déterministe et compilateurs
Thèse d'état, GRENOBLE.

23) GRIFFITHS M., PELTIER P.
Grammar transformation as an aid to compiler production
Centre scientifique IBM FRANCE, Mai 1968.

24) GUYOT A.
Compilation de microprogrammes pour une machine donnée
Rapport de contrat DRME N° 69.34.707.00.480.75.01, octobre 1970.

25) GUYOT A., MARTIN P., MERMET J.
Etude d'un métacompilateur de séquences de commandes
Journées AFCET-IRIA, octobre 1973.

26) GUYOT A., MARTIN P., MERMET J.
A microprogram metacompiler
EUROMICRO NEWSLETTERS ,Vol 1, Septembre 1974.

27) GUYOT A., MARTIN P.
Compilateur de microprogramme paramétré par une description du calculateur
(à paraître Revue Bleue AFCET)

28) GUYOT A., MARTIN P.
Métacompilateur de séquences de commandes
Rapport final de contrat CRI N° 172.57, septembre 1974.

29) GUYOT A.
Réalisation d'un compilateur de microprogramme
Séminaire d'algèbre appliquée et conception, IMAG 1973.

30) GUYOT A.
Compilation de microprogrammes
Séminaire de programmation IMAG, juin 1974.

31) GUYOT A.
The computer hardware description language CASSANDRE and the microprogram compilation
Presented at the Workshop of technishe hochshule DARMSTADT Germany A.C.M. German Chàpter , W 1974

32) GUYOT A.
Travaux dirigés de microprogrammation sur MITRA 15
ENSIMAG 1973.

33) HEBENSTREIT J.
Générateur de microprogramme
Thèse d'état, Université de PARIS, mai 1969.

34) HUSSON S.S.
Microprogramming, principles and practice
Prentice Hall, 1970.

35) KLEIR R.L., RAMAMOORTHY C.V.
Optimization strategies for microprograms
I.E.E.E. Trans. of computer, vol. C-20 N°9, juillet 1971

36) KUNTZMANN J.
Algèbre de Boole
Dunod, Paris 1972.

37) KUNTZMANN J.
Théorie des réseaux
DUNOD, Paris 1972.

38) LIDDELL P.
Découpage syntaxique de systèmes logiques décrits en CASSANDRE
Thèse de 3ème cycle, mars 1970.

39) LUX A.
Survol de la littérature en intelligence artificielle et dans les matières voisines.

40) NOGUEZ G.
La conception rationnelle du hardware
Institut de Programmation, Paris, 1974.

41) NOGUEZ G.
Design of a microprogramming language
Institut de Programmation, Paris, 1974.

42) MANOIR (du) H., MARTIN P.
Réduction du volume des microprogrammes par une méthode d'optimisation de l'allocation des registres
Revue Bleue, R.A.I.R.O., février 1974.

43) MERMET J.
Etude méthodologique de la conception assistée par ordinateurs des systèmes logiques: CASSANDRE
Thèse d'état, Grenoble, avril 1973.

44) MERMET J.
Définition du langage CASSANDRE
Thèse de Docteur-ingénieur, Grenoble, mars 1970.

45) MERMET J.
Microprogrammation
Cours à l'école de printemps de la DRME, avril 1971.

46) MINKLEY N., FARAUD C.
Définition d'un langage évolué d'écriture de microprogrammes sur MULTI 8

47) PELTIER M.
The macro system
Centre scientifique IBM, avril 1969.

48) POLIGNAC (de) K.
Utilisation du langage CASSANDRE pour la conception de machines microprogrammées
Thèse de 3ème cycle, 1973.

49) REYNAUD GAROCHE F.
Théorie des écoulements dans les réseaux
thèse de 3° cycle.

50) SIFAKIS J.
Modèles temporels des systèmes logiques
Thèse de 3° cycle, mars 1974.

51) TIRELL A.K.
A study of the application of compiler techniques
to the generation of microcode
Workshop on microprogramming, Grenoble, juin 1970.

52) TUDRUJ M., MARCZENSKI R.
Une approche de structuration des unités de commande microprogrammé
Rapport de recherche N°47, IRIA, janvier 1974.

53) TURCAT C.
Isomorphisme, immersion et recouvrement de graphes
Thèse de 3° cycle, avril 1974.

54) WILKES M.V.
The best way to design an automatic calculation machine
Manchester University computer in augural conference, 1951.

ANNEXE 1

EXEMPLES

VARIATIONS AUTOUR DE DEUX UNITES

Nous nous sommes donnés une description de machine micro-programmable et d'un algorithme.

L'ensemble des résultats présentés ici a été obtenu à partir de ces deux seules entrées du compilateur, et montre la cohérence de divers outils d'aide à la conception de machine construits autour du langage CASSANDRE.

Nous profitons de l'occasion pour remercier les auteurs des programmes fournissant ces résultats.

EXEMPLE 1

Voici la description de la partie opérative d'une micromachine, contrôlée par une micro-instruction de douze bits.

Construite autour d'un registre appelé TAMPON, qui sert à la fois d'accumulateur et de communication de données avec le milieu extérieur et la mémoire de masse, cette machine possède en outre une mémoire locale de huit registres, directement adressée par la micro-instruction, et une unité arithmétique exécutant l'addition (ADD) et la soustraction (SOUS).

La simplicité un peu spartiate de cette machine a été dictée par des soucis de clarté et de concision. La rendre plus complexe peut changer la dimension des problèmes rencontrés mais non leur nature.

Cette unité étant souvent prise comme exemple il est intéressant de rapprocher sa description CASSANDRE avec :

- son schéma	figure 3.1	Page 49
- sa hiérarchie opérative	figure 3.2	Page 49
- sa hiérarchie de contrôle	figure 2.2	Page 30
- ses polynômes-condition	figure 5.10	Page 86

```
'UNITE' MICROMACHINE(H,ENTREE(1:16) ; SORTIE(1:16)) ;
   'SIGNAUX' BUS1(1:16),BUS2(1:16) ;
   'SIGNAL' MOT(1:12) ;                          "MICROINSTRUCTION"
   'REGISTRES' ADRESSE(1:16,0:7),                "SCRATCH PAD"
         TAMPON(1:16),                           "TAMPON ET ACCUMULATEUR"
         ADRESSE(1:16) ;                         "ADRESSE MEMOIRE"
   'MEMOIRE' MASSE(1:16,0:255) ;                 "MEMOIRE DE MASSE"
   'EXTERNE' ADD((1:16),(1:16) ; (1:16)) ;       "ADDITIONNEUR"
   'EXTERNE' SOUS((1:16),(1:16) ; (1:16)) ;      "SOUSTRACTEUR"
   'HORLOGEINTERNE' H ;
"CHAMPS 1:3 COMMANDES DES CHARGEMENTS DE TAMPON"
   <H> 'SI' -MOT(1) 'ALORS' (
         'SI' $ MOT(2:3) 'ALORS'
         ()                                      "NEANT"
         (TAMPON <= MASSE(,$(ADRESSE)))          "LECTURE MEMOIRE"
         (TAMPON <= ENTREE )                     "PERIPHERIQUES"
         (TAMPON <= 0000 000 & MOT(4:12)))       "VALEUR IMMEDIATE"
         'SINON'
         ( 'SI' $ MOT(2:3) 'ALORS'
         (TAMPON <= BUS1)                        "NOT"
         (TAMPON <= 0000 0000 & BUS1(9:16))      "OCTET"
         (TAMPON <= BUS1(9:16) & BUS1(1:8))      "PERMUTATION"
         (TAMPON <= BUS1(16) & BUS1(1:15))) ;    "ROTATION"
"CHAMPS 4:5 COMMANDES DES CONNEXIONS DE BUS1"
   'SI' -/,(MOT(1:3)=011) 'ALORS' (              "NON VALEUR IMMEDIATE"
   <H> 'SI' $ MOT(4:5) 'ALORS'
         ()
         (ADRESSE(,$(MOT(6:8))) <= BUS1)         "SCRATCH PAD"
         (ADRESSE <= BUS1)                       "ADRESSE MEMOIRE"
         (MASSE(,$(ADRESSE)) <= BUS1)) ;         "MEMOIRE"
"CHAMPS 9:10 SELECTION DE L'OPERATION"
   ADD(TAMPON,BUS2;) ; SOUS(TAMPON,BUS2;) ;
   BUS1:='SI' MOT(9) 'ALORS'
         ('SI' MOT(10) 'ALORS' TAMPON 'SINON' BUS2) "TRANSFERT"
         'SINON'
         ('SI' MOT(10) 'ALORS' ADD(, ; *)          "ADDITION"
         'SINON' SOUS(, ; *)) ;                    "SOUSTRACTION"
"CHAMPS 6:8 SELECTION D'UN REGISTRE GENERAL"
         BUS2:=ADRESSE(,$(MOT(6:8)));
```

EXEMPLE 2

Soit une machine classique à accumulateur et mémoire de masse, et quatre de ses instructions

- Chargement de l'accumulateur,
- Rangement de l'accumulateur,
- Addition dans l'accumulateur,
- Branchement inconditionnel.

L'adresse en mémoire de masse a nécessaire à l'exécution de ces instructions est calculée à partir d'un déplacement d suivant trois modes d'adressage

- Immédiat a = d
- Basé a = d + (base)
- Indirect basé a = (d + (base))

La sémantique de ces instructions est précisée par une description d'interpréteur faite en CASSANDRE sous forme d'automate. Les variables nécessaires à cet interpréteur sont déclarées REGISTRE, et le mécanisme d'enchaînement des états n'est pas explicité.

L'algorithme d'interprétation est en somme décrit sous la forme d'une hypothétique machine qui l'exécute.

Cet algorithme à compiler en microprogramme est cité dans le texte. On trouve entre autre :

- Son organigramme figure 2.4 Page 33
- Le format des instructions figure 2.3 Page 32
- Le schéma de l'incrément INDIRECT figure 3.7 Page 58
- La couverture de cet incrément figure 3.8 Page 58

```
 'UNITE' INTERPRETEUR(H ;) ;
'HORLOGE/RE' H ;
'REGISTRES' AQ(1:16),                          "ACCUMULATEUR"
         CO(1:16),                             "COMPTEUR ORDINAL"
         INST(1:16),                           "INSTRUCTION A EXECUTER"
         BASE(1:16),                           "REGISTRE DE BASE"
         ADRESSE(1:16),                        "ADRESSE RESOLUE"
         MEMOIRE(1:16,0:255) ;                 "MEMOIRE DE MASSE"
'EXTERNE' ADD((1:16),(1:16) ; (1:16)) ;        "ADDITIONNEUR"
FETCH: <H> INST <= MEMOIRE(,$(CO)),'ALLERA' * ;  "RECHERCHE"
DECODE1: <H> 'SI' $ INST(1:2) 'ALORS'      "DECODAGE MODE D'ADRESSAGE"
         ('ALLERA' IMMEDIAT)
         ('ALLERA' BASEE)
         ('ALLERA' INDIRECT)() ;
IMMEDIAT: <H> ADRESSE <= 0000 0000 & INST(9:16),'ALLERA' DECODE2 ;
BASEE: <H> ADRESSE <= ADD|1|(,;*),'ALLERA' DECODE2 ;
         ADD|1|(BASE,0000 0000 & INST(9:16) ;) ;
INDIRECT: <H> ADRESSE <= ADD|1|(,;*),'ALLERA' DECODE2 ;
         ADD|1|(BASE,MEMOIRE(,$(ADD|2|(BASE,0000 0000 & INST(9:16) ;
         *))) ;) ;
DECODE2: <H> 'SI' $ INST(5:6) 'ALORS'      "DECODAGE FONCTION"
         ('ALLERA' CHARGE)
         ('ALLERA' RANGE)
         ('ALLERA' SOMME)
         ('ALLERA' BRANCHE) ;
   "CHARGEMENT DE L'ACCUMULATEUR"
CHARGE: <H> AQ <= MEMOIRE(,$(ADRESSE)),'ALLERA' INCRE ;
   "RANGEMENT DE L'ACCUMULATEUR"
RANGE: <H> MEMOIRE(,$(ADRESSE)) <= AQ,'ALLERA' INCRE ;
   "ADDITION ACCUMULATEUR - MEMOIRE"
SOMME: <H> AQ <= ADD|1|(,;*),'ALLERA' INCRE ;
      ADD|1|(AQ,MEMOIRE(,$(ADRESSE)) ;) ;
   "INCREMENTATION DU COMPTEUR ORDINAL"
INCRE: <H> CO <= ADD|1|(,;*),'ALLERA' FETCH ;
         ADD|1|(CO,0000 0000 0000 0001 ;) ;
   "BRANCHEMENTS"
BRANCHE: <H> CO <= ADRESSE,'ALLERA' FETCH ;
```

ANNEXE 1-6

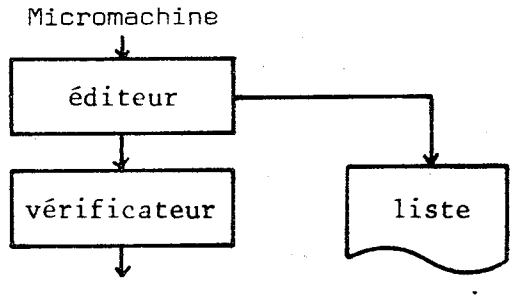
EXEMPLE 3

Liste de l'Editeur - Vérificateur de langage CASSANDRE appliqué à l'unité MICROMACHINE.

Utilisé en commun par toutes les applications de ce langage, ce programme fournit une chaîne intermédiaire syntaxiquement vérifiée, beaucoup plus facile à manipuler que le texte source.

Les numéros d'unité syntaxique rappelés en marge servent à repérer d'éventuelles erreurs.

Position de ce programme dans la chaine de traitement CASSANDRE :



Remarque: Les flèches représentent des fichiers.

Ont participé à la réalisation de ce programme:

F.ANCEAU

J.DOUSSY

EDITION

U. S. TEXTE SOURCE

```
 00001   'UNITE' MICROMACHINE(H,ENTREE(1:16) ; SORTIE(1:16)) ;
00044      'SIGNAUX' BUS1(1:16),BUS2(1:16) ;
00067      'SIGNAL' MUI(1:12) ;                          "MICROINSTRUCTION"
00078      'REGISTRES' ARDOISE(1:16,0:7),                "SCRATCH PAD"
00097            TAMPON(1:16),                           "TAMPON ET ACCUMULATEUR"
00110            ADRESSE(1:16) ;                         "ADRESSE MEMOIRE"
00124      'MEMOIRE' MASSE(1:16,0:255) ;                 "MEMOIRE DE MASSE"
00143      'EXTERNE' ADD((1:16),(1:16) ; (1:16)) ;       "ADDITIONNEUR"
00170      'EXTERNE' SOUS((1:16),(1:16) ; (1:16)) ;      "SOUSTRACTEUR"
00198      'HORLOGEMERE' H ;
00201   "CHAMPS 1:3 COMMANDES DES CHARGEMENTS DE TAMPON"
00201      <H> 'SI' -MUI(1) 'ALORS' (
00214            'SI' $ MUI(2:3) 'ALORS'
00225            ()                                     "NEANT"
00227            (TAMPON <= MASSE(,$(ADRESSE)))         "LECTURE MEMOIRE"
00254            (TAMPON <= ENTREE )                    "PERIPHERIQUES"
00269            (TAMPON <= 0000 000 & MUI(4:12)))      "VALEUR IMMEDIATE"
00296            'SINON'
00297            ( 'SI' $ MUI(2:3) 'ALORS'
00309            (TAMPON <= BUS1)                       "MOT"
00322            (TAMPON <= 0000 0000 & BUS1(9:16))     "OCTET"
00350            (TAMPON <= BUS1(9:16) & BUS1(1:8))     "PERMUTATION"
00379            (TAMPON <= BUS1(16) & BUS1(1:15))) ;   "ROTATION"
00409   "CHAMPS 4:5 COMMANDES DES CONNEXIONS DE BUS1"
00409      'SI' -/.(MUI(1:3)=011) 'ALORS' (            "NON VALEUR IMMEDIATE"
00429      <H> 'SI' $ MUI(4:5) 'ALORS'
00443            ()
00445            (ARDOISE(,$(MUI(6:8))) <= BUS1)       "SCRATCH PAD"
00473            (ADRESSE <= BUS1)                     "ADRESSE MEMOIRE"
00487            (MASSE(,$(ADRESSE)) <= BUS1)) ;       "MEMOIRE"
00514   "CHAMPS 9:10 SELECTION DE L'OPERATION"
00514      ADD(TAMPON,BUS2;) ; SOUS(TAMPON,BUS2;) ;
00551      BUS1:='SI' MUI(9) 'ALORS'
00564            ('SI' MUI(10) 'ALORS' TAMPON 'SINON' BUS2) "TRANSFERT"
00586            'SINON'
00587            ('SI' MUI(10) 'ALORS' ADD(, ; *)          "ADDITION"
00605            'SINON' SOUS(, ; *)) ;                "SOUSTRACTION"
00617   "CHAMPS 6:8 SELECTION D'UN REGISTRE GENERAL"
00617            BUS2:=ARDOISE(,$(MUI(6:8)));
```

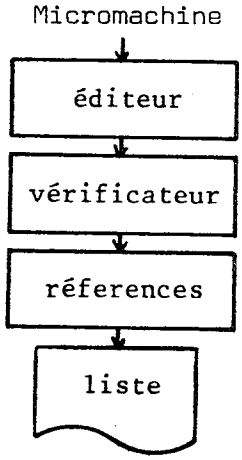
FIN D'EDITION

EXEMPLE 4

Ce tableau donne les numéros d'unité syntaxique repérant l'apparition d'un identificateur dans un texte source CASSANDRE.

Ce tableau est utile dans le cas de très grosses unités.

Position de ce programme :



Ont participé à la réalisation de ce programme:

Y.BRESSY

A.GUYOT

****
TABLE DE REFERENCES CROISEES DE L'UNITE MICROMACHINE
****

REGISTRES
****

ADRESSE
243 473 496

TAMPON
227 254 269 309 322 350 379 517
536 573

ARDOISE
445 621

REGISTRES 32767
****

MASSE
234 487

SIGNAUX D'ENTREE
****

ENTREE
261

SIGNAUX DE SORTIE
****

SORTIE

SIGNAUX INTERNES
****

BUS2
524 543 580 616

BUS1
316 338 357 368 386 395 467 481
506 550

MUI
205 215 284 299 413 433 456 556
565 588 632

HORLOGERIE
****

H
201 429

EXTERNES
****

ADD
513 596

SOUS
531 605

EXEMPLE 5

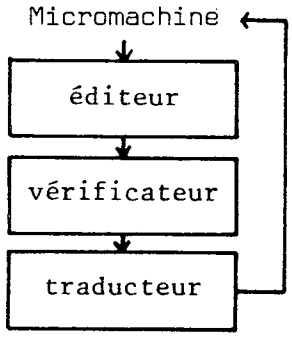
Liste fournie par le programme de traduction de la chaîne intermédiaire CASSANDRE appliqué à l'unité MICROMACHINE.

Utilisé dans nombre d'applications, ce programme de sortie est donc responsable de la présentation de la plupart des exemples qui suivent.

En comparant les exemples 5 et 3, on s'aperçoit que l'ordre d'apparition des déclarations ainsi que les commentaires ont été perdus.

L'indentation est restituée en fonction de la syntaxe.

Position du programme de traduction :



La grammaire de l'analyseur syntaxique de ce programme est donnée en ANNEXE 2.

Ont participé à la réalisation de ce programme:

Y. FANTINO

A. GUYOT

P. LIDDELL

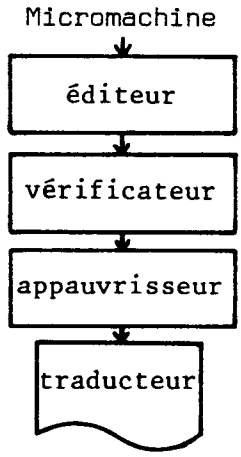
```
'UNITE' MICROMACHINE(H,ENTREE(1:16);SORTIE(1:16));
 'REGISTRE' ADRESSE(1:16),TAMPON(1:16),ARDOISE(1:16,0:7);
 'MEMOIRE' MASSE(1:16,0:255);
 'SIGNAL' BUS2(1:16),BUS1(1:16),MUI(1:12);
 'HORLOGEMERE' H;
 'EXTERNE' ADD((1:16),(1:16);(1:16)),SOUS((1:16),(1:16);(1:16));
        <H>
           'SI' -MUI(1) 'ALORS'
             ('SI' $ MUI(2:3) 'ALORS'
               ( )
               (  TAMPON<=MASSE(,$(ADRESSE)) )
               (  TAMPON<=ENTREE )
               (  TAMPON<=0000 000&MUI(4:12) ) )
           'SINON'
             ('SI' $ MUI(2:3) 'ALORS'
               (  TAMPON<=BUS1 )
               (  TAMPON<=0000 0000&BUS1(9:16) )
               (  TAMPON<=BUS1(9:16)&BUS1(1:8) )
               (  TAMPON<=BUS1(16)&BUS1(1:15) ) );
    'SI' -/.(MUI(1:3)=011) 'ALORS'
     (  <H>
           'SI' $ MUI(4:5) 'ALORS'
             ( )
             (  ARDOISE(,$(MUI(6:8)))<=BUS1 )
             (  ADRESSE<=BUS1 )
             (  MASSE(,$(ADRESSE))<=BUS1 ) );
           ADD(TAMPON,BUS2;);
           SOUS(TAMPON,BUS2;);
           BUS1:='SI' MUI(9) 'ALORS' ('SI' MUI(10) 'ALORS' TAMPON
           'SINON' BUS2) 'SINON' ('SI' MUI(10) 'ALORS' ADD(,;*)
           'SINON' SOUS(,;*));
           BUS2:=ARDOISE(,$(MUI(6:8)));
```

EXEMPLE 6

Le programme qui a fourni la liste qui suit donne pour toute description d'unité en CASSANDRE une description sémantiquement équivalente dans un sous ensemble du langage CASSANDRE.

Cet "appauvrisseur syntaxique" trouverait d'autres utilisations que la compilation de microprogramme s'il ne présentait pas l'inconvénient rédibitoire d'allonger considérablement le texte. On peut comparer les 33 lignes de l'exemple 5 avec les 154 de l'exemple 6 produits à partir du précédent.
La comparaison des textes des deux exemples est bien plus intéressante.

Position du programme d'appauvrissement :



Ont participé à la réalisation de ce programme:

A. GUYOT

```
'UNITE' MICROMACHINE(H,ENTREE(1:16);SORTIE(1:16));
 'REGISTRE' ARDOISE(1:16,0:7),TAMPON(1:16),ADRESSE(1:16);
 'MEMOIRE' MASSE(1:16,0:255);
 'SIGNAL' MUI(1:12),BUS1(1:16),BUS2(1:16);
 'HORLOGEMERE' H;
 'EXTERNE' ADD((1:16),(1:16);(1:16)),SOUS((1:16),(1:16);(1:16));
     'SI' -MUI(1).-MUI(2).MUI(3) 'ALORS'
      (  <H>
               TAMPON<=MASSE(1:16,$(ADRESSE)) );
     'SI' -MUI(1).MUI(2).-MUI(3) 'ALORS'
      (  <H>
               TAMPON<=ENTREE );
     'SI' -MUI(1).MUI(2).MUI(3) 'ALORS'
      (  <H>
               TAMPON<=0000 000&MUI(4:12) );
     'SI' MUI(1).-MUI(2).-MUI(3) 'ALORS'
      (  <H>
               TAMPON<=BUS1 );
     'SI' MUI(1).-MUI(2).MUI(3) 'ALORS'
      (  <H>
               TAMPON<=0000 0000&BUS1(9:16) );
     'SI' MUI(1).MUI(2).-MUI(3) 'ALORS'
      (  <H>
               TAMPON<=BUS1(9:16)&BUS1(1:8) );
     'SI' MUI(1).MUI(2).MUI(3) 'ALORS'
      (  <H>
               TAMPON<=BUS1(16)&BUS1(1:15) );
     'SI' MUI(1).-MUI(4).MUI(5).-MUI(6).-MUI(7).-MUI(8) 'ALORS'
      (  <H>
               ARDOISE(1:16,0)<=BUS1 );
     'SI' -MUI(2).-MUI(4).MUI(5).-MUI(6).-MUI(7).-MUI(8) 'ALORS'
      (  <H>
               ARDOISE(1:16,0)<=BUS1 );
     'SI' -MUI(3).-MUI(4).MUI(5).-MUI(6).-MUI(7).-MUI(8) 'ALORS'
      (  <H>
               ARDOISE(1:16,0)<=BUS1 );
     'SI' MUI(1).-MUI(4).MUI(5).-MUI(6).-MUI(7).MUI(8) 'ALORS'
      (  <H>
               ARDOISE(1:16,1)<=BUS1 );
     'SI' -MUI(2).-MUI(4).MUI(5).-MUI(6).-MUI(7).MUI(8) 'ALORS'
      (  <H>
               ARDOISE(1:16,1)<=BUS1 );
     'SI' -MUI(3).-MUI(4).MUI(5).-MUI(6).-MUI(7).MUI(8) 'ALORS'
      (  <H>
               ARDOISE(1:16,1)<=BUS1 );
     'SI' MUI(1).-MUI(4).MUI(5).-MUI(6).MUI(7).-MUI(8) 'ALORS'
      (  <H>
               ARDOISE(1:16,2)<=BUS1 );
     'SI' -MUI(2).-MUI(4).MUI(5).-MUI(6).MUI(7).-MUI(8) 'ALORS'
      (  <H>
               ARDOISE(1:16,2)<=BUS1 );
     'SI' -MUI(3).-MUI(4).MUI(5).-MUI(6).MUI(7).-MUI(8) 'ALORS'
```

```
  (  <H>
          ARDOISE(1:16,2)<=BUS1 );
'SI' MUI(1).-MUI(4).MUI(5).-MUI(6).MUI(7).MUI(8) 'ALORS'
 (  <H>
          ARDOISE(1:16,3)<=BUS1 );
'SI' -MUI(2).-MUI(4).MUI(5).-MUI(6).MUI(7).MUI(8) 'ALORS'
 (  <H>
          ARDOISE(1:16,3)<=BUS1 );
'SI' -MUI(3).-MUI(4).MUI(5).-MUI(6).MUI(7).MUI(8) 'ALORS'
 (  <H>
          ARDOISE(1:16,3)<=BUS1 );
'SI' MUI(1).-MUI(4).MUI(5).MUI(6).-MUI(7).-MUI(8) 'ALORS'
 (  <H>
          ARDOISE(1:16,4)<=BUS1 );
'SI' -MUI(2).-MUI(4).MUI(5).MUI(6).-MUI(7).-MUI(8) 'ALORS'
 (  <H>
          ARDOISE(1:16,4)<=BUS1 );
'SI' -MUI(3).-MUI(4).MUI(5).MUI(6).-MUI(7).-MUI(8) 'ALORS'
 (  <H>
          ARDOISE(1:16,4)<=BUS1 );
'SI' MUI(1).-MUI(4).MUI(5).MUI(6).-MUI(7).MUI(8) 'ALORS'
 (  <H>
          ARDOISE(1:16,5)<=BUS1 );
'SI' -MUI(2).-MUI(4).MUI(5).MUI(6).-MUI(7).MUI(8) 'ALORS'
 (  <H>
          ARDOISE(1:16,5)<=BUS1 );
'SI' -MUI(3).-MUI(4).MUI(5).MUI(6).-MUI(7).MUI(8) 'ALORS'
 (  <H>
          ARDOISE(1:16,5)<=BUS1 );
'SI' MUI(1).-MUI(4).MUI(5).MUI(6).MUI(7).-MUI(8) 'ALORS'
 (  <H>
          ARDOISE(1:16,6)<=BUS1 );
'SI' -MUI(2).-MUI(4).MUI(5).MUI(6).MUI(7).-MUI(8) 'ALORS'
 (  <H>
          ARDOISE(1:16,6)<=BUS1 );
'SI' -MUI(3).-MUI(4).MUI(5).MUI(6).MUI(7).-MUI(8) 'ALORS'
 (  <H>
          ARDOISE(1:16,6)<=BUS1 );
'SI' MUI(1).-MUI(4).MUI(5).MUI(6).MUI(7).MUI(8) 'ALORS'
 (  <H>
          ARDOISE(1:16,7)<=BUS1 );
'SI' -MUI(2).-MUI(4).MUI(5).MUI(6).MUI(7).MUI(8) 'ALORS'
 (  <H>
          ARDOISE(1:16,7)<=BUS1 );
'SI' -MUI(3).-MUI(4).MUI(5).MUI(6).MUI(7).MUI(8) 'ALORS'
 (  <H>
          ARDOISE(1:16,7)<=BUS1 );
'SI' MUI(1).MUI(4).-MUI(5) 'ALORS'
 (  <H>
          ADRESSE<=BUS1 );
'SI' -MUI(2).MUI(4).-MUI(5) 'ALORS'
 (  <H>
          ADRESSE<=BUS1 );
```

```
 'SI' -MUI(3).MUI(4).-MUI(5) 'ALORS'
 (  <H>
          ADRESSE<=BUS1 );
'SI' MUI(1).MUI(4).MUI(5) 'ALORS'
 (  <H>
          MASSE(1:16,$(ADRESSE))<=BUS1 );
'SI' -MUI(2).MUI(4).MUI(5) 'ALORS'
 (  <H>
          MASSE(1:16,$(ADRESSE))<=BUS1 );
'SI' -MUI(3).MUI(4).MUI(5) 'ALORS'
 (  <H>
          MASSE(1:16,$(ADRESSE))<=BUS1 );
        ADD(*,;):=TAMPON;
        ADD(,*;):=BUS2;
        SOUS(*,;):=TAMPON;
        SOUS(,*;):=BUS2;
'SI' MUI(9).MUI(10) 'ALORS'
          BUS1:=TAMPON;
'SI' MUI(9).-MUI(10) 'ALORS'
          BUS1:=BUS2;
'SI' -MUI(9).MUI(10) 'ALORS'
          BUS1:=ADD(,;*);
'SI' -MUI(9).-MUI(10) 'ALORS'
          BUS1:=SOUS(,;*);
'SI' -MUI(6).-MUI(7).-MUI(8) 'ALORS'
          BUS2:=ARDOISE(1:16,0);
'SI' -MUI(6).-MUI(7).MUI(8) 'ALORS'
          BUS2:=ARDOISE(1:16,1);
'SI' -MUI(6).MUI(7).-MUI(8) 'ALORS'
          BUS2:=ARDOISE(1:16,2);
'SI' -MUI(6).MUI(7).MUI(8) 'ALORS'
          BUS2:=ARDOISE(1:16,3);
'SI' MUI(6).-MUI(7).-MUI(8) 'ALORS'
          BUS2:=ARDOISE(1:16,4);
'SI' MUI(6).-MUI(7).MUI(8) 'ALORS'
          BUS2:=ARDOISE(1:16,5);
'SI' MUI(6).MUI(7).-MUI(8) 'ALORS'
          BUS2:=ARDOISE(1:16,6);
'SI' MUI(6).MUI(7).MUI(8) 'ALORS'
          BUS2:=ARDOISE(1:16,7);
```

EXEMPLE 7

En vue de les simuler ensemble, on a concaténé la description de la micro-machine et le microprogramme fourni par le compilateur.

Le micro-programme est la traduction de l'algorithme d'interprétation donné en exemple 2.

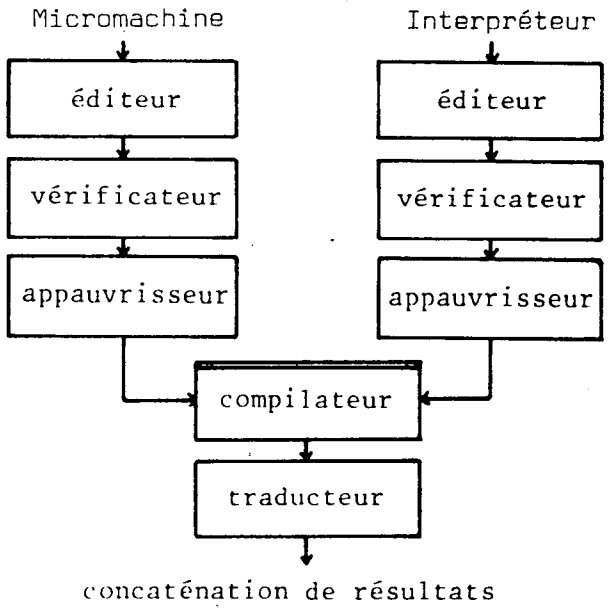
Par exemple, le pas appelé FETCH de cet interpréteur demande pour s'exécuter une séquence de trois micro-instructions repérées par le caractère ":" .

Le grand nombre de micro-instructions générées (26) s'explique par la simplicité de la micro-machine.

L'affectation des variables aux éléments de mémorisation a été faite par l'utilisateur.

Les champs de la micro-instruction auxquels le compilateur n'a pas donné de valeur, sont comblés avec la valeur zéro.

Position du compilateur de microprogrammes :



```
 'UNITE' ADD(E1(1:16),E2(1:16);S(1:16));
  'SIGNAL' RET(1:16);
              S:=E1=(E2=RET);
              RET:=*D|1|(E1.E2+E1.RET+E2.RET)&0;
'UNITE' SOUS(E1(1:16),E2(1:16);S(1:16));
'UNITE' VALN2(E(1:2);S(1:4));
              S:='SI' E(1) 'ALORS' ('SI' E(2) 'ALORS' 0001 'SINON' 001
              0) 'SINON' ('SI' E(2) 'ALORS' 0100 'SINON' 1000);
'UNITE' MICROMACHINE(H,ENTREE(1:16);SORTIE(1:16));
  'REGISTRE' ADRESSE(1:16),TAMPON(1:16),ARDOISE(1:16,0:7);
  'MEMOIRE' MASSE(1:16,0:255);
  'SIGNAL' BUS2(1:16),BUS1(1:16),MOT(1:12);
  'HORLOGEMERE' H;
  'EXTERNE' ADD((1:16),(1:16);(1:16)),SOUS((1:16),(1:16);(1:16)),VALN2
  ((1:2);(1:4));
              VALN2|1|(MOT(2:3));
         <H>
             'SI' MOT(1) 'ALORS'
              ('SI' VALN2|1|(;*) 'ALORS'
                ( )
                (  TAMPON=MASSE(,$(ADRESSE)) )
                (  TAMPON=ENTREE )
                (  TAMPON=0000 000&MOT(4:12) ) )
            'SINON'
              ('SI' VALN2|1|(;*) 'ALORS'
                (  TAMPON=BUS1 )
                (  TAMPON=0000 0000&BUS1(9:16) )
                (  TAMPON=BUS1(9:16)&BUS1(1:8) )
                (  TAMPON=BUS1(10)&BUS1(1:15) ) );
     'SI' -/.(MOT(1:3)=011) 'ALORS'
      (  VALN2|2|(MOT(6:5));
           <H>
            'SI' VALN2|2|(;*) 'ALORS'
              ( )
              (  ARDOISE(,$(MOT(6:8)))<=BUS1 )
              (  ADRESSE<=BUS1 )
              (  MASSE(,$(ADRESSE))<=BUS1 ) );
            ADD(TAMPON,BUS2;);
            SOUS(TAMPON,BUS2;);
            BUS1:='SI' MOT(9) 'ALORS' ('SI' MOT(10) 'ALORS' TAMPON
            'SINON' BUS2) 'SINON' ('SI' MOT(10) 'ALORS' ADR(,;*)
            'SINON' SOUS(,;*));
            BUS2:=ARDOISE(,$(MOT(6:8)));
FETCH:
            MOT:=0001 0001 1000;
         <H>
            'ALLERA' * ;
     :
            MOT:=0010 0000 0000;
         <H>
            'ALLERA' * ;
     :
```

```
                MUI:=0000 1011 1100;
          <H>
                'ALLERA' DECODE1 ;
DECODE1:
          <H>
             'SI' -ARDOISE(1,3).-ARDOISE(2,3) 'ALORS'
              (  'ALLERA' IMMEDIAT  );
          <H>
             'SI' -ARDOISE(1,3).ARDOISE(2,3) 'ALORS'
              (  'ALLERA' BASEE  );
          <H>
             'SI' ARDOISE(1,3).-ARDOISE(2,3) 'ALORS'
              (  'ALLERA' INDIRECT  );
IMMEDIAT:
                MUI:=1010 0011 1000;
          <H>
                'ALLERA' * ;
      :
                MUI:=0000 1100 1100;
          <H>
                'ALLERA' DECODE2 ;
BASEE:
                MUI:=1010 0011 1000;
          <H>
                'ALLERA' * ;
      :
                MUI:=1000 0101 0100;
          <H>
                'ALLERA' * ;
      :
                MUI:=0000 1100 1100;
          <H>
                'ALLERA' DECODE2 ;
INDIRECT:
                MUI:=1010 0011 1000;
          <H>
                'ALLERA' * ;
      :
                MUI:=1000 0101 0100;
          <H>
                'ALLERA' * ;
      :
                MUI:=0001 0000 1100;
          <H>
                'ALLERA' * ;
      :
                MUI:=0010 0000 0000;
          <H>
                'ALLERA' * ;
      :
                MUI:=1000 0101 0100;
          <H>
                'ALLERA' * ;
```

```
:
        MUI:=0000 1100 1100;
      <H>
        'ALLERA' DECODE2 ;
DECODE2:
      <H>
     'SI' -ARDOISE(5,3).-ARDOISE(6,3) 'ALORS'
        (  'ALLERA' CHARGE  );
      <H>
     'SI' -ARDOISE(5,3).ARDOISE(6,3) 'ALORS'
        (  'ALLERA' RANGE  );
      <H>
     'SI' ARDOISE(5,3).-ARDOISE(6,3) 'ALORS'
        (  'ALLERA' SOMME  );
      <H>
     'SI' ARDOISE(5,3).ARDOISE(6,3) 'ALORS'
        (  'ALLERA' BRANCHE  );
CHARGE:
        MUI:=0001 0100 1000;
      <H>
        'ALLERA' * ;
:
        MUI:=0010 0000 0000;
      <H>
        'ALLERA' * ;
:
        MUI:=0000 1010 1100;
      <H>
        'ALLERA' INCRE ;
RANGE:
        MUI:=0001 0100 1000;
      <H>
        'ALLERA' * ;
:
        MUI:=0001 1010 1000;
      <H>
        'ALLERA' INCRE ;
SOMME:
        MUI:=0001 0100 1000;
      <H>
        'ALLERA' * ;
:
        MUI:=0010 0000 0000;
      <H>
        'ALLERA' * ;
:
        MUI:=0000 1010 0100;
      <H>
        'ALLERA' INCRE ;
INCRE:
        MUI:=0110 0000 0001;
      <H>
        'ALLERA' * ;
```

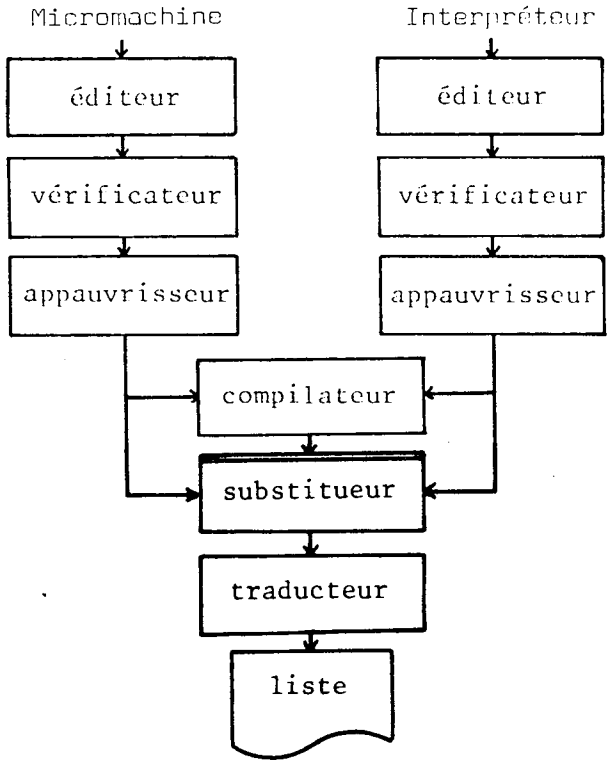
```
        :
             MOT:=0000 1001 0100;
          <H>
             'ALLEGA' FETCH ;
BRANCHE:
             MOT:=1000 0100 1000;
          <..>
             'ALLEGA' * ;
        :
             MOT:=0000 1001 1100;
          <U>
             'ALLEGA' FETCH ;
```

EXEMPLE 8

En remplaçant chaque valeur de micro-instruction de l'exemple 7 par la liste des actions élémentaires qu'elle exécute, donnée par l'exemple 6, on obtient un algorithme d'interprétation équivalent à celui de l'exemple 2.

Cette liste est la deuxième sortie du compilateur. Certaines des actions générées sont superflues.

Position du programme générant la deuxième sortie :



Ont participé à la réalisation de ce programme:

A. GUYOT

```
'UNITE' MICROMACHINE(H,ENTREE(1:16);SORTIE(1:16));
 'REGISTRE' ADRESSE(1:16),TAMPON(1:16),ARDOISE(1:16,0:7);
 'MEMOIRE' MASSE(1:16,0:255);
 'SIGNAL' BUS2(1:16),BUS1(1:16),MUI(1:12);
 'HORLOGEMERE' H;
 'EXTERNE' ADD((1:16),(1:16);(1:16)),SOUS((1:16),(1:16);(1:16));
             ADD(*,;):=TAMPON;
             ADD(,*;):=BUS2;
             SOUS(*,;):=TAMPON;
             SOUS(,*;):=BUS2;
FETCH:
        <H>
             ADRESSE<=BUS1;
             BUS1:=BUS2;
             BUS2:=ARDOISE(1:16,1);
        <H>
             'ALLERA' * ;
      :
        <H>
             TAMPON<=MASSE(1:16,$(ADRESSE));
             BUS1:=SOUS(,;*);
             BUS2:=ARDOISE(1:16,0);
        <H>
             'ALLERA' * ;
      :
        <H>
             ARDOISE(1:16,3)<=BUS1;
             BUS1:=TAMPON;
             BUS2:=ARDOISE(1:16,3);
        <H>
             'ALLERA' DECODE1 ;
DECODE1:
        <H>
           'SI' -ARDOISE(1,3).-ARDOISE(2,3) 'ALORS'
             (  'ALLERA' IMMEDIAT  );
        <H>
           'SI' -ARDOISE(1,3).ARDOISE(2,3) 'ALORS'
             (  'ALLERA' BASEE  );
        <H>
           'SI' ARDOISE(1,3).-ARDOISE(2,3) 'ALORS'
             (  'ALLERA' INDIRECT  );
IMMEDIAT:
        <H>
             TAMPON<=0000 0000&BUS1(9:16);
             BUS1:=BUS2;
             BUS2:=ARDOISE(1:16,3);
        <H>
             'ALLERA' * ;
      :
        <H>
             ARDOISE(1:16,4)<=BUS1;
             BUS1:=TAMPON;
```

```
            BUS2:=ARDOISE(1:16,4);
        <H>
            'ALLERA' DECODE2 ;
BASEE:
        <H>
            TAMPON<=0000 0000&BUS1(9:16);
            BUS1:=BUS2;
            BUS2:=ARDOISE(1:16,3);
        <H>
            'ALLERA' * ;
      :
        <H>
            TAMPON<=BUS1;
            BUS1:=ADD(,;*);
            BUS2:=ARDOISE(1:16,5);
        <H>
            'ALLERA' * ;
      :
        <H>
            ARDOISE(1:16,4)<=BUS1;
            BUS1:=TAMPON;
            BUS2:=ARDOISE(1:16,4);
        <H>
            'ALLERA' DECODE2 ;
INDIRECT:
        <H>
            TAMPON<=0000 0000&BUS1(9:16);
            BUS1:=BUS2;
            BUS2:=ARDOISE(1:16,3);
        <H>
            'ALLERA' * ;
      :
        <H>
            TAMPON<=BUS1;
            BUS1:=ADD(,;*);
            BUS2:=ARDOISE(1:16,5);
        <H>
            'ALLERA' * ;
      :
        <H>
            ADRESSE<=BUS1;
            BUS1:=TAMPON;
            BUS2:=ARDOISE(1:16,0);
        <H>
            'ALLERA' * ;
      :
        <H>
            TAMPON<=MASSE(1:16,$(ADRESSE));
            BUS1:=SOUS(,;*);
            BUS2:=ARDOISE(1:16,0);
        <H>
            'ALLERA' * ;
      :
```

```
        <H>
            TAMPON<=BUS1;
            BUS1:=ADD(,;*);
            BUS2:=ARDOISE(1:16,5);
        <H>
            'ALLERA' * ;
    :
        <H>
            ARDOISE(1:16,4)<=BUS1;
            BUS1:=TAMPON;
            BUS2:=ARDOISE(1:16,4);
        <H>
            'ALLERA' DECODE2 ;
DECODE2:
        <H>
          'SI' -ARDOISE(5,3).-ARDOISE(6,3) 'ALORS'
            (  'ALLERA' CHARGE  );
        <H>
          'SI' -ARDOISE(5,3).ARDOISE(6,3) 'ALORS'
            (  'ALLERA' RANGE  );
        <H>
          'SI' ARDOISE(5,3).-ARDOISE(6,3) 'ALORS'
            (  'ALLERA' SOMME  );
        <H>
          'SI' ARDOISE(5,3).ARDOISE(6,3) 'ALORS'
            (  'ALLERA' BRANCHE  );
CHARGE:
        <H>
            ADRESSE<=BUS1;
            BUS1:=BUS2;
            BUS2:=ARDOISE(1:16,4);
        <H>
            'ALLERA' * ;
    :
        <H>
            TAMPON<=MASSE(1:16,$(ADRESSE));
            BUS1:=SOUS(,;*);
            BUS2:=ARDOISE(1:16,0);
        <H>
            'ALLERA' * ;
    :
        <H>
            ARDOISE(1:16,2)<=BUS1;
            BUS1:=TAMPON;
            BUS2:=ARDOISE(1:16,2);
        <H>
            'ALLERA' INCRE ;
RANGE:
        <H>
            ADRESSE<=BUS1;
            BUS1:=BUS2;
            BUS2:=ARDOISE(1:16,4);
        <H>
```

```
            'ALLERA' * ;
      :
         <H>
            MASSE(1:16,$(ADRESSE))<=BUS1;
            BUS1:=BUS2;
            BUS2:=ARDOISE(1:16,2);
         <H>
            'ALLERA' INCRE ;
SOMME:
         <H>
            ADRESSE<=BUS1;
            BUS1:=BUS2;
            BUS2:=ARDOISE(1:16,4);
         <H>
            'ALLERA' * ;
      :
         <H>
            TAMPON<=MASSE(1:16,$(ADRESSE));
            BUS1:=SOUS(,;*);
            BUS2:=ARDOISE(1:16,0);
         <H>
            'ALLERA' * ;
      :
         <H>
            ARDOISE(1:16,2)<=BUS1;
            BUS1:=ADD(,;*);
            BUS2:=ARDOISE(1:16,2);
         <H>
            'ALLERA' INCRE ;
INCRE:
         <H>
            TAMPON<=0000 000&0000 0000 1;
            BUS1:=SOUS(,;*);
            BUS2:=ARDOISE(1:16,0);
         <H>
            'ALLERA' * ;
      :
         <H>
            ARDOISE(1:16,1)<=BUS1;
            BUS1:=ADD(,;*);
            BUS2:=ARDOISE(1:16,1);
         <H>
            'ALLERA' FETCH ;
BRANCHE:
         <H>
            TAMPON<=BUS1;
            BUS1:=BUS2;
            BUS2:=ARDOISE(1:16,4);
         <H>
            'ALLERA' * ;
      :
         <H>
            ARDOISE(1:16,1)<=BUS1;
```

```
      BUS1:=TAMPON;
     BUS2:=ARDOISE(1:16,1);
<H>
     'ALLERA' FETCH ;
```

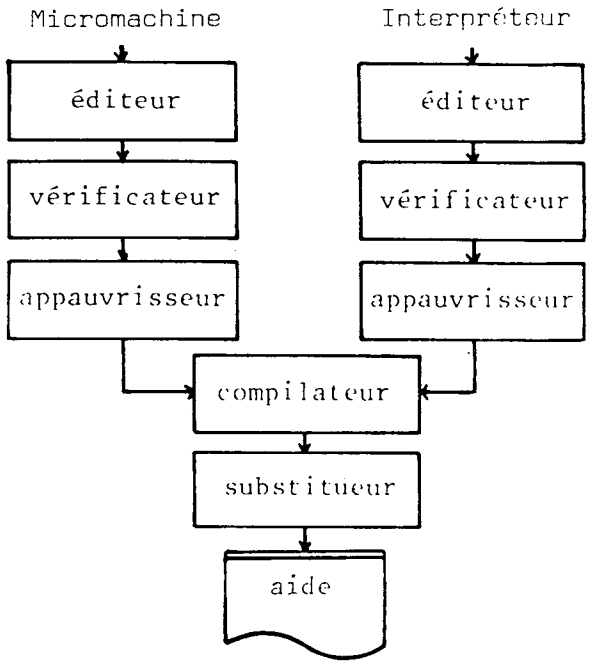
EXEMPLE 9

En appliquant à l'exemple 8 un programme d'aide à la définition de machine micro-programmée, on obtient la liste des actions élémentaires utilisées, chacune commandée par la valeur 1 de un champ de un bit de la micro-instruction MICRØCØM.

En outre, une liste des micro-instructions correspondant à chaque pas de l'algorithme est fournie.

On a ainsi, à partir de la micro-machine initiale, une proposition de structure de machine, généralement plus simple, spécifiquement adaptée à l'algorithme qu'elle doit exécuter.

Position du programme d'aide à la conception de machines :



Ont participé à la réalisation de ce programme:

A. HANCZAKSWSKI

G. MENARD

K. de POLIGNAC

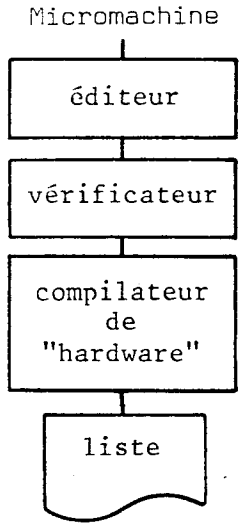
```
 'SI' MICROCOM(1) 'ALORS'   (<H> ADRESSE<=BUS1) ;
'SI' MICROCOM(4) 'ALORS'   (<H> TAMPON<=MASSE(1:16,$ADRESSE))) ;
'SI' MICROCOM(7) 'ALORS'   (<H> ARDOISE(1:16,3)<=BUS1) ;
'SI' MICROCOM(10) 'ALORS'   (<H> TAMPON<=00000000 & BUS1(9:16)) ;
'SI' MICROCOM(11) 'ALORS'   (<H> ARDOISE(1:16,4)<=BUS1) ;
'SI' MICROCOM(13) 'ALORS'   (<H> TAMPON<=BUS1) ;
'SI' MICROCOM(17) 'ALORS'   (<H> ARDOISE(1:16,2)<=BUS1) ;
'SI' MICROCOM(19) 'ALORS'   (<H> MASSE(1:16,$(ADRESSE))<=BUS1) ;
'SI' MICROCOM(20) 'ALORS'   (<H> ARDOISE(1:16,1)<=BUS1) ;
'SI' MICROCOM(21) 'ALORS'   (<H> TAMPON<=000000000 & 0000001) ;
'SI' MICROCOM(2) 'ALORS'   BUS1:=BUS2 ;
'SI' MICROCOM(3) 'ALORS'   BUS2:=ARDOISE(1:16,1) ;
'SI' MICROCOM(6) 'ALORS'   BUS2:=ARDOISE(1:16,0) ;
'SI' MICROCOM(8) 'ALORS'   BUS1:=TAMPON ;
'SI' MICROCOM(9) 'ALORS'   BUS2:=ARDOISE(1:16,3) ;
'SI' MICROCOM(12) 'ALORS'   BUS2:=ARDOISE(1:16,4) ;
'SI' MICROCOM(15) 'ALORS'   BUS2:=ARDOISE(1:16,5) ;
'SI' MICROCOM(18) 'ALORS'   BUS2:=ARDOISE(1:16,2) ;
'SI' MICROCOM(5) 'ALORS'   SOUS(,;BUS1) ;
'SI' MICROCOM(14) 'ALORS'   ADD(,;BUS1) ;
```

```
 1110000000000000000000
0001110000000000000000
0000001110000000000000
0100000110000000000000
0000001001100000000000
0100000110000000000000
0000000000011100000000
0000001100100000000000
0100000110000000000000
0000000000001110000000
1000010100000000000000
0001110000000000000000
0000000000001110000000
0000000100010001000000
1100000000010000000000
0001110000000000000000
0000000100000000011000
1100000000010000000000
0100000000000000001100
0001110000000000000000
0010000000000100000010
0000110000000000000001
0010000000000100000010
0100000000011000000000
0010000100000000000010
```

EXEMPLE 10

Un programme permet de fournir une réalisation de la machine donnée en exemple 1 par interconnexion d'éléments standards. Ces éléments de type : mémoires (MEM), registres (REG), bus (BUS), portes "et"(AND), expanseur (XPA), etc... sont listés avant la description de l'unité MICROMACHINE.

Programmes constituant le compilateur de hardware :



Ont participé à la réalisation de ce programme:

Y. FANTINO

P. LIDDELL

```
'UNITE' MEM00000(W,ADR(0:7),INFO(0:15);OUT(0:15));
  'HORLOGEMERE' W;
  'MEMOIRE' M(1:16,0:255);
      <W> M(,$(ADR))<=INFO;
          OUT:=M(,$(ADR));
'UNITE' MEM00001(W,ADR(0:2),INFO(0:15);OUT(0:15));
  'HORLOGEMERE' W;
  'MEMOIRE' M(1:16,0:7);
      <W> M(,$(ADR))<=INFO;
          OUT:=M(,$(ADR));
'UNITE' EXP00000 (E000;S );
          S:=- E000 ;
'UNITE' AND00000 (COND,E (0:3);S (0:3));
          'POUR' I=0 'A' 3
        'DEBUT'
            S (I):=COND . E(I);
        'FIN';
'UNITE' AND00001 (COND,E (0:15);S (0:15));
          'POUR' I=0 'A' 15
        'DEBUT'
            S (I):=COND . E(I);
        'FIN';
'UNITE' EXP00001 (E000(0:6),E001(0:8);S (0:15));
          S:= E000 & E001 ;
'UNITE' EXP00002 (E000(0:7),E001(0:7);S (0:15));
          S:= E000 & E001 ;
'UNITE' EXP00003 (E000,E001(0:14);S (0:15));
          S:= E000 & E001 ;
'UNITE' EXP00004 (E000(0:2);S );
          S:=-/. E000 ;
'UNITE' EXP00005 (E000(0:2),E001(0:2);S (0:2));
          S:= E000 = E001 ;
'UNITE' AND00002 (COND,E (0:1);S (0:1));
          'POUR' I=0 'A' 1
        'DEBUT'
            S (I):=COND . E(I);
        'FIN';
'UNITE' AND00003 (COND,E (0:2);S (0:2));
          'POUR' I=0 'A' 2
        'DEBUT'
            S (I):=COND . E(I);
        'FIN';
'UNITE' EXP00006 (E000(0:15),E001(0:15);S (0:15));
          S:= E000 + E001 ;
'UNITE' REG00000 (H,ENABLE(0:15),INFO(0:15);OUTPUT(0:15));
  'HORLOGEMERE' H;
  'EXTERNE' MCELL('HORLOGE',,;);
          'POUR' I=0 'A' 15
        'DEBUT'
            MCELL ||| (H,ENABLE(I),INFO(I);OUTPUT(I));
        'FIN';
'UNITE' XPA00000(IN;OUT(0:15));
```

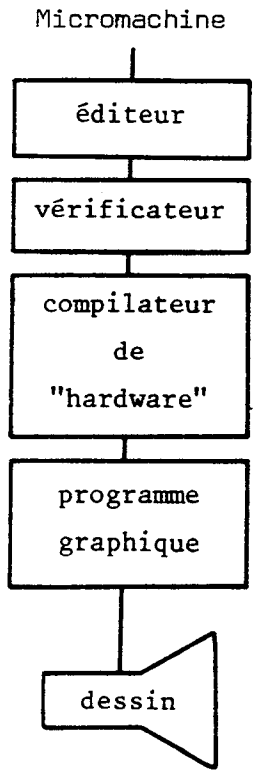
```
          'POUR' I=0 'A' 15
        'DEBUT'
           OUT(I):=IN;
        'FIN';
'UNITE' BUS00000(E000(1:16),E001(1:16),E002(1:16),E003(1:16),E004(1:16
),E005(1:16),E006(1:16);S (1:16));
          S:=E000+E001+E002+E003+E004+E005+E006;
'UNITE' BUS00001(E000(0:7),E001(0:7);S (0:7));
          S:=E000+E001;
'UNITE' BUS00002(E000(0:2),E001(0:2);S (0:2));
          S:=E000+E001;
'UNITE' BUS00003(E000(1:16),E001(1:16),E002(1:16),E003(1:16),E004(1:16
),E005(1:16),E006(1:16);S (1:16));
          S:=E000+E001+E002+E003+E004+E005+E006;
'UNITE' MICROMACHINE(H,ENTREE(1:16);SORTIE(1:16));
    'SIGNAL' ADRESSE(1:16),TAMPON(1:16);
  'SIGNAL' BUS2(1:16),BUS1(1:16),MUI(1:12);
  'SIGNAL' SGL00001(1:16),SGL00000(1:16),CND00004(0:3),CND00003(0:0),
  CND00002(0:3),CND00001(0:3),CND00000(0:0),SUN00001(0:2),SUN00000(0:7
  );
  'HORLOGEMERE' H;
  'EXTERNE' ADD((1:16),(1:16);(1:16)),SOUS((1:16),(1:16);(1:16)),
  VALNUM((1:2);(1:4));
  'EXTERNE' INV(;);
  'EXTERNE' EXP00000 (;),EXP00001 ((0:6),(0:8);(0:15)),EXP00002 ((0:7)
  ,(0:7);(0:15)),EXP00003 (,(0:14);(0:15)),EXP00004 ((0:2);),EXP00005
   ((0:2),(0:2);(0:2)),EXP00006 ((0:15),(0:15);(0:15));
  'EXTERNE' AND00000(,(0:3);(0:3)),AND00001(,(0:15);(0:15)),AND00002(,
  (0:1);(0:1)),AND00003(,(0:2);(0:2));
  'EXTERNE' XPA00000(;(0:15));
  'EXTERNE' REG00000('HORLOGE',(0:15),(0:15);(0:15));
  'EXTERNE'BUS00000((1:16),(1:16),(1:16),(1:16),(1:16),(1:16),(1:16);(
  1:16)),BUS00001((0:7),(0:7);(0:7)),BUS00002((0:2),(0:2);(0:2)),
  BUS00003((1:16),(1:16),(1:16),(1:16),(1:16),(1:16),(1:16);(1:16));
  'EXTERNE' MEM00000('HORLOGE',(0:7),(0:15);(0:15)),MEM00001('HORLOGE'
  ,(0:2),(0:15);(0:15));
          AND00001|8|(CND00004(1),BUS1(1:16););
          MEM00001|1|(H(1),AND00003|0|(CND00004(1),MUI(6:8);*),
          AND00001|8|(,;*););
          AND00001|11|(CND00004(3),BUS1(1:16););
          MEM00000|1|(H(1),AND00001|10|(CND00004(3),ADRESSE(1:16);*),
          AND00001|11|(,;*););
          EXP00006|0|(AND00001|15|(INV|3|(;*),EXP00006|2|(AND00001|17|
          (INV|2|(;*),SOUS(,;*);*),AND00001|16|(MUI(10),ADD(,;*);*);*)
          ;*),AND00001|12|(MUI(9),EXP00006|1|(AND00001|14|(INV|2|(;*),
          BUS2(1:16);*),AND00001|13|(MUI(10),TAMPON(1:16);*);*);*);
          BUS1(1:16));
          MEM00001|1|(H(1),SUN00001(0:2),;BUS2(1:16));
          VALNUM|1|(MUI(2:3););
          MEM00000|1|(H(1),SUN00000(0:7),;);
          VALNUM|2|(AND00002|0|(CND00003(0),MUI(4:5);*););
          ADD(TAMPON(1:16),BUS2(1:16););
          SOUS(TAMPON(1:16),BUS2(1:16););
```

```
 EXP00000|0|(MUI(1);CND00000(0));
AND00000|0|(CND00000(0),VALNUM|1|(;*);CND00001(0:3));
INV|1|(CND00000(0););
AND00000|1|(INV|1|(;*),VALNUM|1|(;*);CND00002(0:3));
EXP00004|0|(EXP00005|0|(MUI(1:3),011;*);CND00003(0));
AND00000|2|(CND00003(0),VALNUM|2|(;*);CND00004(0:3));
INV|2|(MUI(10););
INV|3|(MUI(9););
REG00000|0|(H(1),SGL00001(1:16),SGL00000(1:16);TAMPON(1:16))
;
REG00000|1|(H(1),XPA00000|7|(CND00004(2);*),AND00001|9|(
CND00004(2),BUS1(1:16);*);ADRESSE(1:16));
BUS00000(AND00001|0|(CND00001(1),MEM00000|1|(,,;*);*),
AND00001|2|(CND00001(2),ENTREE(1:16);*),AND00001|3|(CND00001
(3),EXP00001|0|(0000000,MUI(4:12);*);*),AND00001|4|(CND00002
(0),BUS1(1:16);*),AND00001|5|(CND00002(1),EXP00002|0|(
00000000,BUS1(9:16);*);*),AND00001|6|(CND00002(2),EXP00002|1
|(BUS1(9:16),BUS1(1:8);*);*),AND00001|7|(CND00002(3),
EXP00003|0|(BUS1(16),BUS1(1:15);*);*);SGL00000(1:16));
BUS00001(AND00001|10|(CND00004(3),ADRESSE(1:16);*),AND00001|
1|(CND00001(1),ADRESSE(1:16);*);SUN00000(0:7));
BUS00002(MUI(6:8),AND00003|0|(CND00004(1),MUI(6:8);*);
SUN00001(0:2));
BUS00003(XPA00000|0|(CND00001(1);*),XPA00000|1|(CND00001(2);
*),XPA00000|2|(CND00001(3);*),XPA00000|3|(CND00002(0);*),
XPA00000|4|(CND00002(1);*),XPA00000|5|(CND00002(2);*),
XPA00000|6|(CND00002(3);*);SGL00001(1:16));
```

EXEMPLE 11

L'interconnexion d'éléments standards peut être visualisée sur écran ou dessinée sur table traçante grace à un système conversationnel de programmes graphiques.

La hiérarchie de controle de la micromachine a été isolée et seule dessinée.

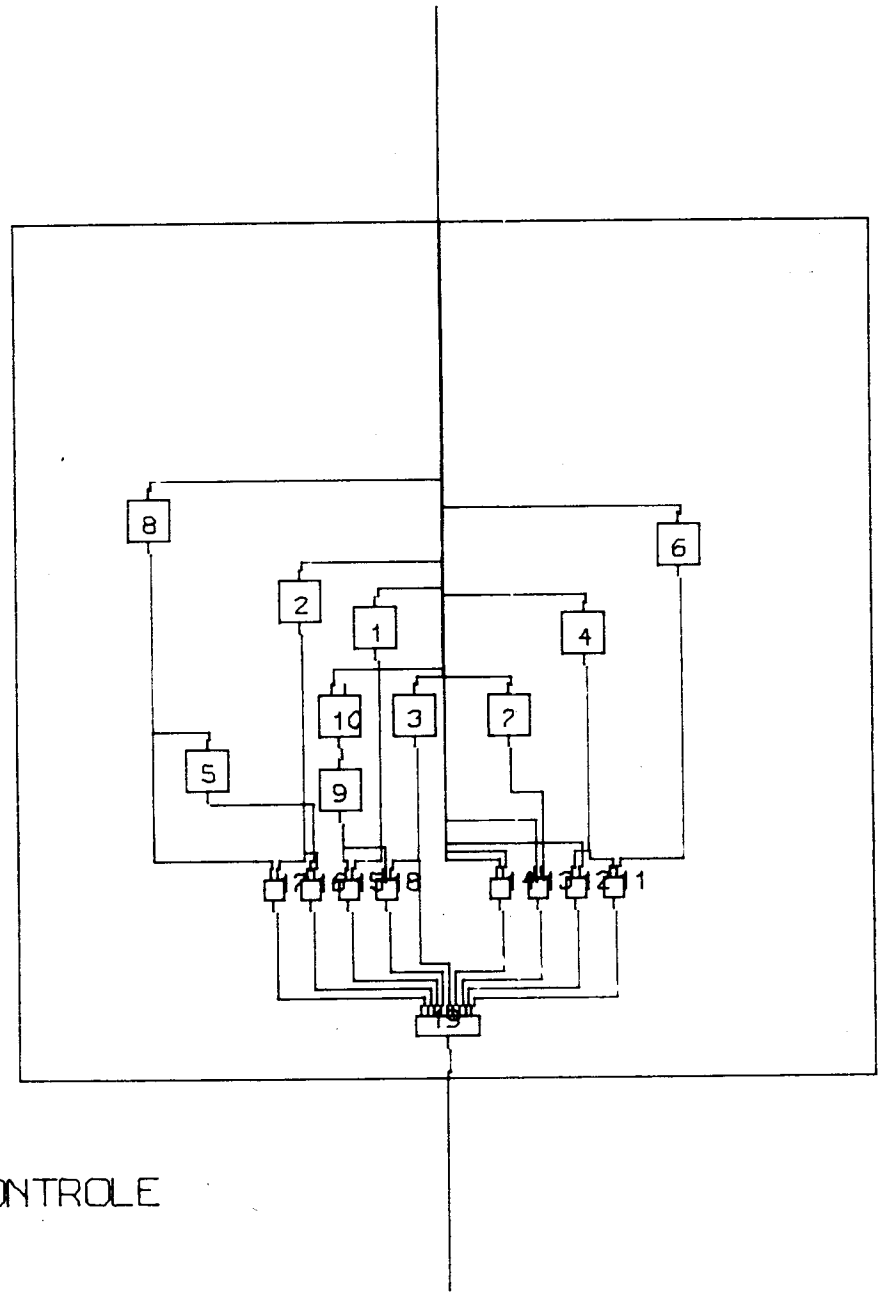


Ces programmes ont été réalisés par B. DAVID.

INSTITUT DE MATHEMATIQUES APPLIQUEES DE GRENOBLE

*** CONTROLE

1 == VALNUM2|2|
2 == VALNUM2|1|
3 == VALNUM3
4 == INV|2|
5 == INV|1|
6 == EXP00000|2|
7 == EXP00000|1|
8 == EXP00000|0|
9 == EXP00001|0|
10 == EXP00002|0|
11 == AND|5|
12 == AND|4|
13 == AND|3|
14 == AND|2|
15 == AND00000|2|
16 == AND00000|1|
17 == AND00000|0|
18 == AND00001|0|
19 == BUS00000

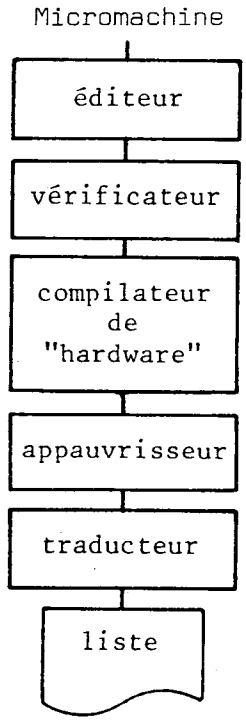


CONTROLE

EXEMPLE 12

L'interconnexion de "boîtes noires" est plus lisible sur cette liste, obtenue à partir de la précédente par l'"appauvrisseur syntaxique" et classé en ordre alphabétique par le programme de tri de CMS.

Chaine de programmes parcourue pour arriver au résultat :



Ont participé à la réalisation de ce programme:

Y. FANTINO

A. GUYOT

P. LIDDELL

```
'UNITE' MICROMACHINE(H,ENTREE(1:16);SORTIE(1:16));
 'SIGNAL' TAMPON(1:16),ADRESSE(1:16),MUI(1:12),BUS1(1:16),BUS2(1:16),
 SUN00000(0:7),SUN00001(0:2),CND00000(0:0),CND00001(0:3),CND00002(0:3
 ),CND00003(0:0),CND00004(0:3),SGL00000(1:16),SGL00001(1:16);
 'HORLOGEMERE' H;
 'EXTERNE' ADD((1:16),(1:16);(1:16)),SOUS((1:16),(1:16);(1:16)),
 VALNUM((1:2);(1:4)),INV(;),EXP00000(;),EXP00001((0:6),(0:8);(0:15)),
 EXP00002((0:7),(0:7);(0:15)),EXP00003(,(0:14);(0:15)),EXP00004((0:2)
 ;),EXP00005((0:2),(0:2);(0:2)),EXP00006((0:15),(0:15);(0:15)),
 AND00000(,(0:3);(0:3)),AND00001(,(0:15);(0:15)),AND00002(,(0:1);(0:1
 )),AND00003(,(0:2);(0:2)),XPA00000(;(0:15)),REG00000('HORLOGE',(0:15
 ),(0:15);(0:15)),BUS00000((1:16),(1:16),(1:16),(1:16),(1:16),(1:16),
 (1:16);(1:16)),BUS00001((0:7),(0:7);(0:7)),BUS00002((0:2),(0:2);(0:2
 )),BUS00003((1:16),(1:16),(1:16),(1:16),(1:16),(1:16),(1:16);(1:16)
 ),MEM00000('HORLOGE',(0:7),(0:15);(0:15)),MEM00001('HORLOGE',(0:2),(
 0:15);(0:15));
              ADD(*,;):=TAMPON(1:16);
              ADD(,*;):=BUS2(1:16);
              ADRESSE(1:16):=REG00000|1|(,,;*);
              AND00000|0|(*,;):=CND00000(0);
              AND00000|0|(,*;):=VALNUM|1|(;*);
              AND00000|1|(*,;):=INV|1|(;*);
              AND00000|1|(,*;):=VALNUM|1|(;*);
              AND00000|2|(*,;):=CND00003(0);
              AND00000|2|(,*;):=VALNUM|2|(;*);
              AND00001|0|(*,;):=CND00001(1);
              AND00001|0|(,*;):=MEM00000|1|(,,;*);
              AND00001|1|(*,;):=CND00001(1);
              AND00001|1|(,*;):=ADRESSE(1:16);
              AND00001|10|(*,;):=CND00004(3);
              AND00001|10|(*,;):=CND00004(3);
              AND00001|10|(,*;):=ADRESSE(1:16);
              AND00001|10|(,*;):=ADRESSE(1:16);
              AND00001|11|(*,;):=CND00004(3);
              AND00001|11|(,*;):=BUS1(1:16);
              AND00001|12|(*,;):=MUI(9);
              AND00001|12|(,*;):=EXP00006|1|(,;*);
              AND00001|13|(*,;):=MUI(10);
              AND00001|13|(,*;):=TAMPON(1:16);
              AND00001|14|(*,;):=INV|2|(;*);
              AND00001|14|(,*;):=BUS2(1:16);
              AND00001|15|(*,;):=INV|3|(;*);
              AND00001|15|(,*;):=EXP00006|2|(,;*);
              AND00001|16|(*,;):=MUI(10);
              AND00001|16|(,*;):=ADD(,;*);
              AND00001|17|(*,;):=INV|2|(;*);
              AND00001|17|(,*;):=SOUS(,;*);
              AND00001|2|(*,;):=CND00001(2);
              AND00001|2|(,*;):=ENTREE(1:16);
              AND00001|3|(*,;):=CND00001(3);
              AND00001|3|(,*;):=EXP00001|0|(,;*);
              AND00001|4|(*,;):=CND00002(0);
```

```
 AND00001|4|(,*;):=BUS1(1:16);
AND00001|5|(*,;):=CND00002(1);
AND00001|5|(,*;):=EXP00002|0|(,;*);
AND00001|6|(*,;):=CND00002(2);
AND00001|6|(,*;):=EXP00002|1|(,;*);
AND00001|7|(*,;):=CND00002(3);
AND00001|7|(,*;):=EXP00003|0|(,;*);
AND00001|8|(*,;):=CND00004(1);
AND00001|8|(,*;):=BUS1(1:16);
AND00001|9|(*,;):=CND00004(2);
AND00001|9|(,*;):=BUS1(1:16);
AND00002|0|(*,;):=CND00003(0);
AND00002|0|(,*;):=MUI(4:5);
AND00003|0|(*,;):=CND00004(1);
AND00003|0|(*,;):=CND00004(1);
AND00003|0|(,*;):=MUI(6:8);
AND00003|0|(,*;):=MUI(6:8);
BUS00000(*,,,,,;):=AND00001|0|(,;*);
BUS00000(,*,,,,;):=AND00001|2|(,;*);
BUS00000(,,*,,,;):=AND00001|3|(,;*);
BUS00000(,,,*,,;):=AND00001|4|(,;*);
BUS00000(,,,,*,;):=AND00001|5|(,;*);
BUS00000(,,,,,*;):=AND00001|6|(,;*);
BUS00000(,,,,,,*;):=AND00001|7|(,;*);
BUS00001(*,;):=AND00001|10|(,;*);
BUS00001(,*;):=AND00001|1|(,;*);
BUS00002(*,;):=MUI(6:8);
BUS00002(,*;):=AND00003|0|(,;*);
BUS00003(*,,,,,,;):=XPA00000|0|(;*);
BUS00003(,*,,,,,;):=XPA00000|1|(;*);
BUS00003(,,*,,,,;):=XPA00000|2|(;*);
BUS00003(,,,*,,,;):=XPA00000|3|(;*);
BUS00003(,,,,*,,;):=XPA00000|4|(;*);
BUS00003(,,,,,*,;):=XPA00000|5|(;*);
BUS00003(,,,,,,*;):=XPA00000|6|(;*);
BUS1(1:16):=EXP00006|0|(,;*);
BUS2(1:16):=MEM00001|1|(,,;*);
CND00000(0):=EXP00000|0|(;*);
CND00001(0:3):=AND00000|0|(,;*);
CND00002(0:3):=AND00000|1|(,;*);
CND00003(0):=EXP00004|0|(;*);
CND00004(0:3):=AND00000|2|(,;*);
EXP00000|0|(*;):=MUI(1);
EXP00001|0|(*,;):=0000 000;
EXP00001|0|(,*;):=MUI(4:12);
EXP00002|0|(*,;):=0000 0000;
EXP00002|0|(,*;):=BUS1(9:16);
EXP00002|1|(*,;):=BUS1(9:16);
EXP00002|1|(,*;):=BUS1(1:8);
EXP00003|0|(*,;):=BUS1(16);
EXP00003|0|(,*;):=BUS1(1:15);
EXP00004|0|(*;):=EXP00005|0|(,;*);
EXP00005|0|(*,;):=MUI(1:3);
```

```
 EXP00005|0|(,*;):=011;
EXP00006|0|(*,;):=AND00001|15|(,;*);
EXP00006|0|(,*;):=AND00001|12|(,;*);
EXP00006|1|(*,;):=AND00001|14|(,;*);
EXP00006|1|(,*;):=AND00001|13|(,;*);
EXP00006|2|(*,;):=AND00001|17|(,;*);
EXP00006|2|(,*;):=AND00001|16|(,;*);
INV|1|(*;):=CND00000(0);
INV|2|(*;):=MUI(10);
INV|3|(*;):=MUI(9);
MEM00000|1|(,*,;):=AND00001|10|(,;*);
MEM00000|1|(,*,;):=SUN00000(0:7);
MEM00000|1|(,,*;):=AND00001|11|(,;*);
MEM00000|1|(H(1),,;);
MEM00000|1|(H(1),,;);
MEM00001|1|(,*,;):=AND00003|0|(,;*);
MEM00001|1|(,*,;):=SUN00001(0:2);
MEM00001|1|(,,*;):=AND00001|8|(,;*);
MEM00001|1|(H(1),,;);
MEM00001|1|(H(1),,;);
REG00000|0|(,*,;):=SGL00001(1:16);
REG00000|0|(,,*;):=SGL00000(1:16);
REG00000|0|(H(1),,;);
REG00000|1|(,*,;):=XPA00000|7|(;*);
REG00000|1|(,,*;):=AND00001|9|(,;*);
REG00000|1|(H(1),,;);
SGL00000(1:16):=BUS00000(,,,,,,;*);
SGL00001(1:16):=BUS00003(,,,,,,;*);
SOUS(*,;):=TAMPON(1:16);
SOUS(,*;):=BUS2(1:16);
SUN00000(0:7):=BUS00001(,;*);
SUN00001(0:2):=BUS00002(,;*);
TAMPON(1:16):=REG00000|0|(,,;*);
VALNUM|1|(*;):=MUI(2:3);
VALNUM|2|(*;):=AND00002|0|(,;*);
XPA00000|0|(*;):=CND00001(1);
XPA00000|1|(*;):=CND00001(2);
XPA00000|2|(*;):=CND00001(3);
XPA00000|3|(*;):=CND00002(0);
XPA00000|4|(*;):=CND00002(1);
XPA00000|5|(*;):=CND00002(2);
XPA00000|6|(*;):=CND00002(3);
XPA00000|7|(*;):=CND00004(2);
```

EXEMPLE 13

Le micro-programme fourni en exemple 7 est testé.

Le jeu d'essais est un macro-programme faisant la somme des entiers d'un tableau.

Un programme d'édition donne la valeur de la micro-instruction en cours en binaire et celle des registres de mémoire locale qui varient en hexadécimal.

Ce programme a été interrompu au début de la deuxième boucle.

Ont participé à la réalisation de ce programme:

F. ANCEAU
Y. BRESSY
J. DOUSSY
A. GUYOT
G. MENARD
F. PERRON

```
simul micromac
.test (demande d'exécution d'un programme de test pré-enregistré)
R H=1
RH ARDOISE(,1)=0010
RH ARDOISE(,5)=0010
RH MASSE(,16:25)=4007:4808:4407:8007:3809:3409:4C00:0010:0001:0020
RH MASSE(,32:34)=0005:0010:0015
IMPCOM OFF
                PROGRAMME DE TEST
   DEBUT        LA      INDEX
                ADD     UN
                STA     INDEX
                LA I    INDEX
                ADD I   SOMME
                STA I   SOMME
  INDEX         DATA    10
  UN            DATA    1
  SOMME         DATA    20

 ETAT       MICROINSTRUCTION        MEM.LOCALE   COMP ACCU INCT ADRE BASE
FETCH
            MUI 000 10 001 1000     ARDOISE 0000 0010 0000 0000 0000 0010 0000 0000

            MUI 001 00 000 0000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----

            MUI 000 01 011 1100     ARDOISE ---- ---- ---- 4007 ---- ---- ---- ----
DECODE1
            MUI --- -- --- ----     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
BASEE
            MUI 101 00 011 1000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----

            MUI 100 00 101 0100     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----

            MUI 000 01 100 1100     ARDOISE ---- ---- ---- ---- 0017 ---- ---- ----
DECODE2
            MUI --- -- --- ----     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
CHARGE
            MUI 000 10 100 1000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----

            MUI 001 00 000 0000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----

            MUI 000 01 010 1100     ARDOISE ---- ---- 0010 ---- ---- ---- ---- ----
INCRE
            MUI 011 00 000 0001     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----

            MUI 000 01 001 0100     ARDOISE ---- 0011 ---- ---- ---- ---- ---- ----
FETCH
            MUI 000 10 001 1000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----

            MUI 001 00 000 0000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----

            MUI 000 01 011 1100     ARDOISE ---- ---- ---- 4808 ---- ---- ---- ----
```

SIMULATION

```
DECODE1
        MUI --- -- --- ----     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
BASEE
        MUI 101 00 011 1000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
        MUI 100 00 101 0100     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
        MUI 000 01 100 1100     ARDOISE ---- ---- ---- ---- 0018 ---- ---- ----
DECODE2
        MUI --- -- --- ----     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
SOMME
        MUI 000 10 100 1000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
        MUI 001 00 000 0000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
        MUI 000 01 010 0100     ARDOISE ---- ---- 0011 ---- ---- ---- ---- ----
INCRE
        MUI 011 00 000 0001     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
        MUI 000 01 001 0100     ARDOISE ---- 0012 ---- ---- ---- ---- ---- ----
FETCH
        MUI 000 10 001 1000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
        MUI 001 00 000 0000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
        MUI 000 01 011 1100     ARDOISE ---- ---- ---- 4407 ---- ---- ---- ----
DECODE1
        MUI --- -- --- ----     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
BASEE
        MUI 101 00 011 1000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
        MUI 100 00 101 0100     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
        MUI 000 01 100 1100     ARDOISE ---- ---- ---- ---- 0017 ---- ---- ----
DECODE2
        MUI --- -- --- ----     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
RANGE
        MUI 000 10 100 1000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
        MUI 000 11 010 1000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
INCRE
        MUI 011 00 000 0001     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
        MUI 000 01 001 0100     ARDOISE ---- 0013 ---- ---- ---- ---- ---- ----
FETCH
        MUI 000 10 001 1000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
        MUI 001 00 000 0000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
        MUI 000 01 011 1100     ARDOISE ---- ---- ---- 8007 ---- ---- ---- ----
DECODE1
        MUI --- -- --- ----     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
```

SIMULATION

```
INDIRECT
          MUI 101 00 011 1000     APPOISE ---- ---- ---- ---- ---- ---- ---- ----
          MUI 100 00 101 0100     APPOISE ---- ---- ---- ---- ---- ---- ---- ----
          MUI 000 10 000 1100     APPOISE ---- ---- ---- ---- ---- ---- ---- ----
          MUI 001 00 000 0000     APPOISE ---- ---- ---- ---- ---- ---- ---- ----
          MUI 100 00 101 0100     APPOISE ---- ---- ---- ---- ---- ---- ---- ----
          MUI 000 01 100 1100     APPOISE ---- ---- ---- ---- 0021 ---- ---- ----
DECODE2
          MUI --- -- --- ----     APPOISE ---- ---- ---- ---- ---- ---- ---- ----
CHARGE
          MUI 000 10 100 1000     APPOISE ---- ---- ---- ---- ---- ---- ---- ----
          MUI 001 00 000 0000     APPOISE ---- ---- ---- ---- ---- ---- ---- ----

          MUI 000 01 010 1100     APPOISE ---- ---- 0010 ---- ---- ---- ---- ----
INCRE
          MUI 011 00 000 0001     APPOISE ---- ---- ---- ---- ---- ---- ---- ----
          MUI 000 01 001 0100     APPOISE ---- 0014 ---- ---- ---- ---- ---- ----
FETCH
          MUI 000 10 001 1000     APPOISE ---- ---- ---- ---- ---- ---- ---- ----
          MUI 001 00 000 0000     APPOISE ---- ---- ---- ---- ---- ---- ---- ----
          MUI 000 01 011 1100     APPOISE ---- ---- ---- 8809 ---- ---- ---- ----
DECODE1
          MUI --- -- --- ----     APPOISE ---- ---- ---- ---- ---- ---- ---- ----
INDIRECT
          MUI 101 00 011 1000     APPOISE ---- ---- ---- ---- ---- ---- ---- ----
          MUI 100 00 101 0100     APPOISE ---- ---- ---- ---- ---- ---- ---- ----
          MUI 000 10 000 1100     APPOISE ---- ---- ---- ---- ---- ---- ---- ----
          MUI 001 00 000 0000     APPOISE ---- ---- ---- ---- ---- ---- ---- ----
          MUI 100 00 101 0100     APPOISE ---- ---- ---- ---- ---- ---- ---- ----
          MUI 000 01 100 1100     APPOISE ---- ---- ---- ---- 0030 ---- ---- ----
DECODE2
          MUI --- -- --- ----     APPOISE ---- ---- ---- ---- ---- ---- ---- ----
SOMME
          MUI 000 10 100 1000     APPOISE ---- ---- ---- ---- ---- ---- ---- ----
          MUI 001 00 000 0000     APPOISE ---- ---- ---- ---- ---- ---- ---- ----
```

SIMULATION

```
        MUI 000 01 010 0100     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
INCRE
        MUI 011 00 000 0001     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----

        MUI 000 01 001 0100     ARDOISE ---- 0015 ---- ---- ---- ---- ---- ----
FETCH
        MUI 000 10 001 1000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----

        MUI 001 00 000 0000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----

        MUI 000 01 011 1100     ARDOISE ---- ---- ---- 8409 ---- ---- ---- ----
DECODE1
        MUI --- -- --- ----     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
INDIRECT
        MUI 101 00 011 1000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----

        MUI 100 00 101 0100     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----

        MUI 000 10 000 1100     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----

        MUI 001 00 000 0000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----

        MUI 100 00 101 0100     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----

        MUI 000 01 100 1100     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
DECODE2
        MUI --- -- --- ----     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
RANGE
        MUI 000 10 100 1000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----

        MUI 000 11 010 1000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
INCRE
        MUI 011 00 000 0001     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----

        MUI 000 01 001 0100     ARDOISE ---- 0016 ---- ---- ---- ---- ---- ----
FETCH
        MUI 000 10 001 1000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----

        MUI 001 00 000 0000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----

        MUI 000 01 011 1100     ARDOISE ---- ---- ---- 4C00 ---- ---- ---- ----
DECODE1
        MUI --- -- --- ----     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
BASEE
        MUI 101 00 011 1000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----

        MUI 100 00 101 0100     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----

        MUI 000 01 100 1100     ARDOISE ---- ---- ---- ---- 0010 ---- ---- ----
DECODE2
        MUI --- -- --- ----     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
```

SIMULATION

```
BRANCHE
        MUI 100 00 100 1000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
        MUI 000 01 001 1100     ARDOISE ---- 0010 ---- ---- ---- ---- ---- ----
FETCH
        MUI 000 10 001 1000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
        MUI 001 00 000 0000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
        MUI 000 01 011 1100     ARDOISE ---- ---- ---- 4007 ---- ---- ---- ----
DECODE1
        MUI --- -- --- ----     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
BASEE
        MUI 101 00 011 1000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
        MUI 100 00 101 0100     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
        MUI 000 01 100 1100     ARDOISE ---- ---- ---- ---- 0017 ---- ---- ----
DECODE2
        MUI --- -- --- ----     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
CHARGE
        MUI 000 10 100 1000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
        MUI 001 00 000 0000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
        MUI 000 01 010 1100     ARDOISE ---- ---- 0011 ---- ---- ---- ---- ----
INCRE
        MUI 011 00 000 0001     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
        MUI 000 01 001 0100     ARDOISE ---- 0011 ---- ---- ---- ---- ---- ----
FETCH
        MUI 000 10 001 1000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
        MUI 001 00 000 0000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
        MUI 000 01 011 1100     ARDOISE ---- ---- ---- 4808 ---- ---- ---- ----
DECODE1
        MUI --- -- --- ----     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
BASEE
        MUI 101 00 011 1000     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
        MUI 100 00 101 0100     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
        MUI 000 01 100 1100     ARDOISE ---- ---- ---- ---- 0018 ---- ---- ----
DECODE2
        MUI --- -- --- ----     ARDOISE ---- ---- ---- ---- ---- ---- ---- ----
SOMME
```

simulation interrompue

ANALYSEUR

SYNTAXE ET SEMANTIQUES DU LANGAGE CASSANDRE.

Une dizaine au moins de suiveurs syntaxiques sont utilisés dans le système CASSANDRE, compte non tenu des diverses versions de ce système. Chacun des programmes présenté en annexe 1 contient un. Les analyseurs s'obtiennent grâce au générateur (compiler-compiler) de Monsieur GRIFFITHS.

Nous avons choisi de donner comme exemple d'utilisation le Traducteur car sa grammaire (sur-grammaire du langage CASSANDRE) est l'une des plus simples et les fonctions sémantiques commandant la présentation et la restitution des symboliques, peu nombreuses et faciles à comprendre.

Dans le programme Appauvrisseur, un analyseur semblable, pilote la contruction d'une arborescence syntaxique, reprise par un transducteur. Dans le Compilateur de microprogramme, deux analyseurs construisent des hiérarchies sémantiques respectivement à partir de la micro-machine et du microprogramme.

L'analyseur du Substitueur est du type "leurré". Des fonctions sémantiques modifient le texte en cours d'analyse et induisent les décisions du suiveur syntaxique.

```
INPUT SYNTAX=

130     <LIU>           <DEJ> &LIGNE '83'

131     <DEJ>           'IDUD' &LIGNE &COL56 <LIST> &LIGNE <LISTETA>
                        'IDUD'
                        'IDUD' &LIGNE &COL56 <LIST>
                        'IDUD' &LIGNE <LISTETA>

132     <LISTETA>       <LETAT>
                        <LETAT> &LIGNE <LISTETA>

133     <LETAT>         'VET' &TEXE &SORE 'DPT' &SORE 'PV'
                        'VET' &TEXE &SORE 'DPT' &LIGNE &COL56 <LIST>
                        'VET' &TEXE &SORE 'DPT' &LIGNE &SORE 'DEBUT' &LIGNE <LISTETA> &LIGNE &SORE 'FIN'

134     <LIST>          <FLIST> &SORE 'PV'
                        <FLIST> &SORE 'PV' &LIGNE &COL56 <LIST>
                        .

135     <FLIST>         &BLANK <BB>
                        &SORE 'POUR' &BLANK 'IDENT' &TEXE <BOUCL> &BLANK &SORE 'A' &BLANK <EXPA> *TAB* &LIGNE &COL56 &S
ORE 'DEBUT' &LIGNE &COL56 <SLIST> &SORE 'FIN' *BACK*

136     <BOUCL>         &SORE 'PAS' <EXPA>
                        &BLANK &SORE 'DEPUIS' &BLANK <EXPA>
                        &SORE 'PAS' <EXPA> &BLANK &SORE 'PAS' &BLANK <EXPA>
                        &BLANK &SORE 'DEPUIS' &BLANK <EXPA> &BLANK &SORE 'PAS' &BLANK <EXPA>

137     <SLIST>         <FLIST> &SORE 'PV' &LIGNE &COL56 <SLIST>
                        ()
                        &SORE <SLISTA>

138     <SLISTA>        'STA' &BLANK <EXPB> &BLANK &SORE 'ALORS' *TAB* &LIGNE &COL56 &SORE 'DEBUT' &LIGNE &COL56 <LIST>
&LIGNE &COL56 &SORE 'FIN' &SORE 'PV' *BACK* &LIGNE &COL56 <SLIST>
                        'STA' &BLANK <EXPB> &BLANK &SORE 'ALORS' *TAB* &LIGNE &COL56 &SORE 'DEBUT' &LIGNE &COL56 <LIST>
&LIGNE &COL56 &SORE 'FIN' &BLANK &SORE 'SINON' &LIGNE &COL56 &SORE 'FIN' &LIGNE &COL56 <LIST> &LIGNE &COL56 &SORE 'FIN
' &SORE 'PV' *BACK* &LIGNE &COL56 <SLIST>

139     <BB>            &SORE <OUC>
                        &BLANK <BBE>
                        &BLANK &SORE 'FAIRE' &BLANK <EXPDS>
                        &BLANK &SORE 'TNE' <EXPH> &SORE 'SUP' &LIGNE &COL116 <AFG>

140     <BBC>           'SI' &BLANK <AEX> &BLANK &SORE 'ALORS' *TAB* &LIGNE &COL56 <BRETBR> *BACK*
                        'SI' &BLANK <AEX> &BLANK &SORE 'ALORS' *TAB* &LIGNE &COL56 <BRETBR> &LIGNE &COL56 &SORE 'SINON'
&LIGNE &COL56 <BRTBR> *BACK*

141     <BRETBR>        &BLANK <BBE>
                        &BLANK &SORE <TBR>

142     <BRTBR>         &BLANK <BB>
                        &BLANK &SORE <TBR>

143     <TBR>           'BC' &BLANK &SORE 'BC'
                        'BC' &BLANK &SORE 'BC' &LIGNE &COL56 &SORE <TBR>
                        'BC' <LIB> &BLANK &SORE 'BC'
                        'BC' <LIB> &BLANK &SORE 'BC' &LIGNE &COL56 &SORE <TBR>
```

```
144     <LBR>           <FLIST>
                        <FLIST> &SORE 'PV' &LIGNE &COL78 <LBR>

145     <BPE>           <TERE> &SORE 'BRANCH' <EXP>
                        <SU> <BSU>
                        <SU> <BSU> &SORE 'BRANCH' <EXP>

146     <AFG>           ()
                        <FAFG> &SORE 'VIRG' &LIGNE &COL116 <AFG>
                        <FAFG>

147     <FAFG>          &BLANK2 <AF>
                        &SORE 'POUR' &BLANK 'IDENT' &TEXE <BOUCL> &BLANK &SORE 'A' &BLANK <EXPA> *TAB* &LIGNE &COL116 &
SORE 'DEBUT' <SIF> &SORE 'FIN' *BACK*

148     <SAF>           &LIGNE &COL116 <FAFG> <SAF>
                        &LIGNE &COL116 &SORE <SAFSI> <SAF>
                        &BLANK &SORE 'VIRG' <SAF>
                        &LIGNE &COL116

149     <SAFSI>         'STA' &BLANK <EXPB> &BLANK &SORE 'ALORS' *TAB* &LIGNE &COL116 &SORE 'DEBUT' &LIGNE &COL116 <AFG
> &LIGNE &COL116 &SORE 'FIN' *BACK*
                        'STA' &BLANK <EXPB> &BLANK &SORE 'ALORS' *TAB* &LIGNE &COL116 &SORE 'DEBUT' &LIGNE &COL116 <AFG
> &LIGNE &COL116 &SORE 'FIN' &BLANK &SORE 'SINON' &LIGNE &COL116 &SORE 'DEBUT' &LIGNE &COL116 <AFG> &LIGNE &COL116 &SOR
E 'FIN' *BACK*

150     <AF>            &SORE <AFC>
                        &BLANK2 <AFE>

151     <AFC>           'ST' &BLANK <AEX> &BLANK &SORE 'ALORS' *TAB* &LIGNE &COL126 <AFETAF> *BACK*
                        'ST' &BLANK <AEX> &BLANK &SORE 'ALORS' *TAB* &LIGNE &COL126 <AFETAF> &LIGNE &COL116 &SORE 'SINO
N' &LIGNE &COL126 <AFCTAF> *BACK*

152     <AFETAF>        &BLANK3 <AFF>
                        &SORE <TAF>

153     <AFCTAF>        &BLANK &SORE <AFC>
                        &BLANK <AFF>
                        &SORE <TAF>

154     <TAF>           'PG' &BLANK &SORE 'PD'
                        'PG' &BLANK &SORE 'PD' &LIGNE &COL126 &SORE <TAF>
                        'PG' <LAF> &BLANK &SORE 'PD'
                        'PG' <LAF> &BLANK &SORE 'PD' &LIGNE &COL126 &SORE <TAF>

155     <LAF>           <AF>
                        <AF> &SORE 'VIRG' &LIGNE &COL136 <LAF>

156     <AFF>           &SORE <ALLE>
                        <TERE> &SORE 'BRANCH' <EXP>
                        <TERE> &SORE 'ELG' <EXPD>
                        <SU> <BSU>
                        <SU> <BSU> &SORE 'BRANCH' <EXP>
                        <JDET> &SORE 'ELG' <EXPDS>

157     <ALLE>          'ALLERA' &BLANK <EXPDS> &BLANK
                        'ALLERA' &BLANK <EXPDS> &BLANK &SORE 'DE' &BLANK <SU> <BSU>
```

ANNEXE 2-4

```
158     <EXPDS>         &SORE 'ET' &BLANC <AEX> &BLANC &SORE 'ALORS' &BLANC <VIDET> &BLANC &SORE 'SINON' &BLANC <EXPDS>
                        <VIDET>
                        &SORE 'ET' &BLANC <AEX> &BLANC &SORE 'ALORS' &BLANC &SORE 'PD' <EXPDS> &SORE 'PD' &BLANC &SORE
'SINON' &BLANC <EXPDS>

159     <VIDET>         &SORE 'SU'
                        <IDET>
                        'VET' &TEXT
                        'VETM' &TEXT
                        'VETX' &TEXT
                        &SORE 'ETAT' &BLANC &SORE 'IDF' &BLANC <SU> <BSU>

16      <IDET>          'IDET' &TEXT
                        'IDET' &TEXT <DIM>

161     <EXP>           &SORE 'ET' &BLANC <AEX> &BLANC &SORE 'ALORS' &BLANC <TP> &BLANC &SORE 'SINON' &BLANC <EXP>
                        <TP>

162     <TP>            <PP> &SORE <SIG> <TP>
                        <PP>

163     <PP>            <TER>
                        &SORE <SIG> <PP>
                        &SORE <SIG> <MACRO> <PP>
                        &SORE <SIG> <MACRO> <MACRO> <PP>

164     <TER>           <TER>
                        &SORE 'ERR' <DIM>
                        &SORE 'INDF' <DIM>
                        <CTRC>
                        <SU> <BSU>
                        &SORE 'PC' <EXP> &SORE 'PD'

165     <SIG>           <CORREL>
                        <CONCRED>
                        <OPR>
                        <ASIG>

166     <CONCRED>       'CONC'
                        'REDUC'

167     <OPR>           'PLUS'
                        'POINT'
                        'MOINS'

168     <ASIG>          'DIG'
                        'ROTG'
                        'TILDA'
                        'TAU'
                        'UNSOUP'
                        'DERIV'

169     <CORREL>        'SUP'
                        'INF'
                        'SUPG'
                        'INFG'
                        'EG'
```

```
                    'DIE'

7   <TERE>          'IOSE' &TEXE
                    'IOSE' &TEXE <DIM>
                    'IOS' &TEXE
                    'IOS' &TEXE <DIM>
                    'IOR' &TEXE
                    'IOR' &TEXE <DIM>
                    'IOH' &TEXE
                    'IOH' &TEXE <DIM>
                    'IOHM' &TEXE
                    'IOHM' &TEXE <DIM>
                    'IOM' &TEXE
                    'IOM' &TEXE <DIM>

71  <AEX>           <EXP>
                    &SORE 'VALNUM' &BLANE <EXP>
                    &SORE 'CC' <EXPBE> &SORE 'CC'

72  <EXPBE>         <PRBE>
                    <PRBE> &SORE <SIG> <EXPBE>

73  <PRBE>          &SORE 'PG' <EXPBE> &SORE 'PD'
                    <EXPOS> &SORE <SIG> <EXPOS>

74  <DIM>           &SORE 'PG' <LIND> &SORE 'PD'

75  <LIND>          <LIND1>
                    <IND> &SORE 'DRT' <TAD> <LIND1>
                    <IND> <LIND1>

76  <LIND1>         ()
                    &SORE 'VIRG' <LIND>

77  <IND>           <EXPA>
                    &SORE 'VALNUM' &SORE 'PG' <EXP> &SORE 'PD'

78  <CTBJ>          <BIT> <CTBCA>
                    <BIT> <BIT> <CTBCA>
                    <BIT> <BIT> <BIT> <CTBCA>
                    <BIT> <BIT> <BIT> <BIT> <CTBUA>
                    <BIT> <BIT> <BIT> <BIT> &BLANE <LTB>

79  <CTBJA>         ()
                    &SORE 'DRT' <CTBD>

8   <BIT>           &SORE 'NL'
                    &SORE 'NIL'
                    &SORE 'DNIL'

81  <EXPA>          &SORE 'SIA' &BLANE <EXPB> &BLANE &SORE 'ALORS' &BLANE <TPA> &BLANE &SORE 'SINON' &BLANE <EXPA>
                    <TPA>

82  <TPA>           &SORE 'MOINS' <TERA>
                    <TPA> &SORE 'PLUS' <TPA>
                    <TPA> &SORE 'MOINS' <TPA>
                    <TPA>
```

ANNEXE 2-6

```
182     <EPA>           <TERA>
                        <TERA> &SOR& 'POINT' <EPA>
                        <TERA> &SOR& 'POINT' &SOR& 'REDUC' <TERA>

184     <TERA>          'IDENT' &TEX&
                        'ENT' &SORENT&
                        &SOR& 'PG' <EXPA> &SOR& 'PD'

185     <EXPA>          <PPA>
                        <PPA> &SOR& <CONCRET> <EXPB>

186     <PPB>           &SOR& 'CC' <EXPA> &SOR& 'CC'
                        <TPA> &SOR& <OPREL> <TPA>

187     <SU>            'IDUE' &TEX&
                        'IDUE' &TEX& &SOR& 'CC' <EXPA> &SOR& 'CC'

188     <BSU>           &SOR& 'PG' <LP> &SOR& 'PV' <LP> &SOR& 'PD'

189     <LP>            <LP1>
                        <EXP> <LP1>
                        &SOR& 'BSU' <LP1>
                        &SOR& 'BSU' 'ENT' &SORENT& <LP1>

190     <LP1>           ()
                        &SOR& 'VIRG' <LP>

191     <MACRO>         &SOR& 'CC' <EXPA> &SOR& 'CC'

192     <EXPH>          &SOR& 'SI' &BLANC <AEX> &BLANC &SOR& 'ALORS' &BLANC <TPH> &BLANC &SOR& 'SINON' &BLANC <EXPH>
                        <TPH>

193     <TPH>           <PPH> &SOR& 'PLUS' <TPH>
                        <PPH>

194     <PPH>           <EPH> &SOR& 'CONC' <PPH>
                        <EPH>

195     <EPH>           &SOR& 'PG' <EXPH> &SOR& 'PD'
                        &SOR& 'TAU' <MACRO> <EPH>
                        &SOR& 'REDUC' &SOR& 'PLUS' <EPH>
                        &SOR& 'DERIV' <TER>
                        <SU> <BSU>
                        <TERE>
```

Pour lire cette grammaire il faut savoir que:

- < α > désigne un symbole non terminal.

- La fonction sémantique &SOR& et la fonction sémantique &TEX& commandent respectivement l'impression du mot-clé ' α ' qui le suit ou du symbolique ' α ' qui la précède dans l'analyseur. Les autres fonctions commandent l'indentation. Ce sont: &LIGN& qui fait passer à la ligne, &COLn& qui fait passer à la $n^{ième}$ colonne, &BLANn& qui saute n caractères, *TAB* et *BACK* qui règlent la tabulation respectivement trois colonnes plus à droite ou à gauche.