N° d'ordre : 741

THESE

Présentée à

L'UNIVERSITÉ DES SCIENCES ET TECHNIQUES DE LILLE

Pour obtenir le titre de

DOCTEUR DE TROISIEME CYCLE

(INFORMATIQUE)

Par

Claude SAMUELSON

ETUDE ET DEFINITION D'UNE ARCHITECTURE MULTI-PROCESSEURS ADAPTEE AU TRAITEMENT DE LA PAROLE

Soutenue le 21 décembre 1978, devant la commission d'examen

MM.	BACCHUS	Président
	CARREZ	Examinateur
	MERCIER	Examinateur
	STEPHAN	Examinateur
	CORDONNIER	Rapporteur

UNIVERSITE DES SCIENCES
ET TECHNIQUES DE LILLE

01.02.1976

DOYENS HONORAIRES de l'Ancienne Faculté des Sciences

MM. R. DEFRETIN, H. LEFEBVRE, M. PARREAU.

PROFESSEURS HONORAIRES des Anciennes Facultés de Droit et Sciences Economiques, des Sciences et des Lettres

M. ARNOULT, Mme BEAUJEU, MM. BROCHARD, CHAPPELON, CHAUDRON, CORDONNIER, CORSIN, DEHEUVELS, DEHORS, DION, FAUVEL, FLEURY, P. GERMAIN, HEIM DE BALSAC, HOCQUETTE, KAMPE DE FERIET, KOUGANOFF, LAMOTTE, LASSERRE, LELONG, Mme LELONG, MM. LHOMME, LIEBAERT, MARTINOT-LAGARDE, MAZET, MICHEL, NORMANT, PEREZ, ROIG, ROSEAU, ROUBINE, ROUELLE, SAVART, WATERLOT, WIEMAN, ZAMANSKI.

PRESIDENTS HONORAIRES DE L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

MM. R. DEFRETIN, M. PARREAU.

PRESIDENT DE L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

M. J. LOMBARD.

PROFESSEURS TITULAIRES

M.	BACCHUS Pierre	Astronomie
M.	BEAUFILS Jean-Pierre	Chimie Physique
M.	BECART Maurice	Physique Atomique et Moléculaire
M.	BILLARD Jean	Physique du Solide
M.	BIAYS Pierre	Géographie
M.	BONNEMAN Pierre	Chimie Appliquée
M.	BONNOT Ernest	Biologie Végétale
M.	BONTE Antoine	Géologie Appliquée
M.	BOUGHON Pierre	Algèbre
M.	BOURIQUET Robert	Biologie Végétale
M.	CELET Paul	Géologie Générale
M.	CONSTANT Eugène	Electronique
M.	DECUYPER Marcel	Géométrie
M.	DELATTRE Charles	Géologie Générale
M.	DELHAYE Michel	Chimie Physique
M.	DERCOURT Michel	Géologie Générale
M.	DURCHON Maurice	Biologie Expérimentale
M.	FAURE Robert	Mécanique
M.	FOURET René	Physique du Solide
M.	GABILLARD Robert	Electronique
M.	GLACET Charles	Chimie Organique
M.	GONTIER Gérard	Mécanique
M.	GRUSON Laurent	Algèbre
M.	GUILLAUME Jean	Microbiologie
M.	HEUBEL Joseph	Chimie Minérale
M.	LABLACHE-COMBIER Alain	Chimie Organique
M.	LANSRAUX Guy	Physique Atomique et Moléculaire
M.	LAVEINE Jean-Pierre	Paléontologie
M.	LEBRUN André	Electronique
M.	LEHMANN Daniel	Géométrie

.../...

Mme	LENOBLE Jacqueline	Physique Atomique et Moléculaire
M.	LINDER Robert	Biologie et Physiologie Végétales
M.	LOMBARD Jacques	Sociologie
M.	LOUCHEUX Claude	Chimie Physique
M.	LUCQUIN Michel	Chimie Physique
M.	MAILLET Pierre	Sciences Economiques
M.	MONTARIOL Frédéric	Chimie Appliquée
M.	MONTREUIL Jean	Biochimie
M.	PARREAU Michel	Analyse
M.	POUZET Pierre	Analyse Numérique
M.	PROUVOST Jean	Minéralogie
M.	SALMER Georges	Electronique
M.	SCHILTZ René	Physique Atomique et Moléculaire
Mme	SCHWARTZ Marie-Hélène	Géométrie
M.	SEGUIER Guy	Electrotechnique
M.	TILLIEU Jacques	Physique Théorique
M.	TRIDOT Gabriel	Chimie Appliquée
M.	VIDAL Pierre	Automatique
M.	VIVIER Emile	Biologie Cellulaire
M.	WERTHEIMER Raymond	Physique Atomique et Moléculaire
M.	ZEYTOUNIAN Radyadour	Mécanique

PROFESSEURS SANS CHAIRE

M.	BELLET Jean	Physique Atomique et Moléculaire
M.	BODARD Marcel	Biologie Végétale
M.	BOILLET Pierre	Physique Atomique et Moléculaire
M.	BOILLY Bénoni	Biologie Animale
M.	BRIDOUX Michel	Chimie Physique
M.	CAPURON Alfred	Biologie Animale
M.	CORTOIS Jean	Physique Nucléaire et Corpusculaire
M.	DEBOURSE Jean-Pierre	Gestion des entreprises
M.	DEPREZ Gilbert	Physique Théorique
M.	DEVRAINNE Pierre	Chimie Minérale
M.	GOUDMAND Pierre	Chimie Physique
M.	GUILBAULT Pierre	Physiologie Animale
M.	LACOSTE Louis	Biologie Végétale
Mme	LEHMANN Josiane	Analyse
M.	LENTACKER Firmin	Géographie
M.	LOUAGE Francis	Electronique
Mle	MARQUET Simone	Probabilités
M.	MIGEON Michel	Chimie Physique
M.	MONTEL Marc	Physique du Solide
M.	PANET Marius	Electrotechnique
M.	RACZY Ladislas	Electronique
M.	ROUSSEAU Jean-Paul	Physiologie Animale
M.	SLIWA Henri	Chimie Organique

MAITRES DE CONFERENCES (et chargés d'Enseignement)

M.	ADAM Michel	Sciences Economiques
M.	ANTOINE Philippe	Analyse
M.	BART André	Biologie Animale
M.	BEGUIN Paul	Mécanique
M.	BKOUCHE Rudolphe	Algèbre
M.	BONNELLE Jean-Pierre	Chimie
M.	BONNEMAIN Jean-Louis	Biologie Végétale
M.	BOSCQ Denis	Probabilités
M.	BREZINSKI Claude	Analyse Numérique
M.	BRUYELLE Pierre	Géographie

.../...

M. CARREZ Christian	Informatique
M. CORDONNIER Vincent	Informatique
M. COQUERY Jean-Marie	Psycho-Physiologie
Mle DACHARRY Monique	Géographie
M. DEBENEST Jean	Sciences Economiques
M. DEBRABANT Pierre	Géologie Appliquée
M. DE PARIS Jean-Claude	Mathématiques
M. DHAINAUT André	Biologie Animale
M. DELAUNAY Jean-Claude	Sciences Economiques
M. DERIEUX Jean-Claude	Microbiologie
M. DOUKHAN Jean-Claude	Physique du Solide
M. DUBOIS Henri	Physique
M. DYMENT Arthur	Mécanique
M. ESCAIG Bertrand	Physique du Solide
Me EVRARD Micheline	Chimie Appliquée
M. FONTAINE Jacques-Marie	Electronique
M. FOURNET Bernard	Biochimie
M. FORELICH Daniel	Chimie Physique
M. GAMBLIN André	Géographie
M. GOBLOT Rémi	Algèbre
M. GOSSELIN Gabriel	Sociologie
M. GRANELLE Jean-Jacques	Sciences Economiques
M. GUILLAUME Henri	Sciences Economiques
M. HECTOR Joseph	Géométrie
M. JACOB Gérard	Informatique
M. JOURMEL Gérard	Physique Atomique et Moléculaire
Mle KOSMAN Yvette	Géométrie
M. KREMBEL Jean	Biochimie
M. LAURENT François	Automatique
Mle LEGRAND Denise	Algèbre
Mle LEGRAND Solange	Algèbre
M. LEROY Jean-Marie	Chimie Appliquée
M. LEROY Yves	Electronique
M. LHENAFF René	Géographie
M. LOCQUENEUX Robert	Physique Théorique
M. LOUCHET Pierre	Sciences de l'Education
M. MACKE Bruno	Physique
M. MAHIEU Jean-Marie	Physique Atomique et Moléculaire
Me N'GUYEN VAN CHI Régine	Géographie
M. MAIZIERES Christian	Automatique
M. MALAUSSENA Jean-Louis	Sciences Economiques
M. MESSELYN Jean	Physique Atomique et Moléculaire
M. MONTUELLE Bernard	Biologique Appliquée
M. NICOLE Jacques	Chimie Appliquée
M. PAQUET Jacques	Géologie Générale
M. PARSY Fernand	Mécanique
M. PECQUE Marcel	Chimie Physique
M. PERROT Pierre	Chimie Appliquée
M. PERTUZON Emile	Physiologie Animale
M. PONSOLLE Louis	Chimie Physique
M. POVY Lucien	Automatique
M. RICHARD Alain	Biologie
M. ROGALSKI Marc	Analyse
M. ROY Jean-Claude	Psycho-Physiologie
M. SIMON Michel	Sociologie
M. SOMME Jean	Géographie
Mle SPIK Geneviève	Biochimie
M. STANKIEWICZ François	Sciences Economiques
M. STEEN Jean-Pierre	Informatique

.../...

M.	THERY Pierre	Electronique
M.	TOULOTTE Jean-Marc	Automatique
M.	TREANTON Jean-René	Sociologie
M.	VANDORPE Bernard	Chimie Minérale
M.	VILLETTE Michel	Mécanique
M.	WALLART Francis	Chimie
M.	WERNIER Georges	Informatique
M.	WATERLOT Michel	Géologie Générale
Mme	ZINN-JUSTIN Nicole	Algèbre

ETUDE ET DEFINITION D'UNE. ARCHITECTURE MULTI-PROCESSEURS ADAPTEE AU TRAITEMENT DE LA PAROLE .

claude
SAMUELSON

Le travail de recherche présenté dans ce mémoire a été effectué au C.N.E.T. à Lannion.

Je tiens à exprimer tous mes remerciements,

à la direction scientifique du C.N.E.T.,et en particulier à M. LE MEZEC,adjoint au directeur scientifique, pour m'avoir permis de préparer cette thèse de troisième cycle au C.N.E.T. ;

à tous les membres du département "Services spéciaux du Téléphone" (SST),dirigé par M. STEPHAN , pour l'accueil qu'ils m'ont réservé et l'aide qu'ils m'ont apportéeau cours de l'élaboration de ce travail de recherche ;

à M. EL MALLAWANY (s'il m'entend de ses pyramides);

à M. CORDONNIER ,professeur à l'université de LILLE ,pour avoir accepté de diriger cette thèse , et enfin ,
aux membres du jury,pour l'intérêt qu'ils ont bien voulu porter à mon travail .

P L A N

INTRODUCTION

INTRODUCTION

La "philosophie" du concepteur de système s'est considérablement modifiée depuis quelques années. Cela est dû, en grande partie, aux progrès foudroyants de la micro-électronique, liés à la baisse non moins spectaculaire du coût des composants (sur le plan complexité/prix).

Depuis les débuts de l'informatique, le "cerveau" de l'ordinateur (processeur central + mémoire centrale) était ce qui coûtait le plus cher. Tout le logiciel reposait donc sur l'utilisation la plus rentable du "cerveau". Le concepteur se pliait à cette contrainte : il adaptait l'application à réaliser au matériel existant, il se conformait aux spécifications de la machine ; en un mot, il servait l'ordinateur.

L'apparition récente du microprocesseur monolithique et bon marché change (ou devrait changer) la méthodologie du concepteur.

Ce dernier va suivre la démarche suivante :

- définition précise de l'application à réaliser ;
- détermination de l'architecture répondant le mieux au problème ;
- conception et réalisation d'une machine adaptée.

...

La machine devient donc un moyen, un outil et non plus une fin en soi.

C'est dans cette nouvelle optique que nous avons tenté de travailler.

L'objectif posé au départ était de concevoir un système capable "d'analyser" le signal vocal, à des fins de reconnaissance et de codage de la parole, en temps réel.

L'analyse de la parole a pour but l'extraction d'un certain nombre de paramètres significatifs décrivant le signal acoustique et son évolution dans le temps. Cette "compression" de l'information permet de manipuler et traiter un volume restreint de données.

Les méthodes de prédiction linéaire, appliquées au traitement du signal vocal, ont la faveur des chercheurs dans le domaine depuis une dizaine d'années. Elles offrent en effet deux caractéristiques intéressantes :

- elles se fondent sur l'utilisation de relations mathématiques assez simples.
- elles permettent de travailler sur des informations numériques (en l'occurence, des échantillons de parole numérisée).

Ces caractéristiques font qu'elles se prêtent bien au traitement informatique. Markel et Gray (1976) ont décrit en détail l'utilisation de la prédiction linéaire pour le traitement de la parole.

Le traitement numérique du signal (qu'il soit vocal ou non) est une conséquence du développement de "calculateurs numériques" fonctionnant à grande vitesse et permettant ainsi le temps réel. Dans de nombreux domaines, il supplante peu à peu le traitement analogique (cf. GOLD 1977).

Ces diverses considérations nous ont conduit à étudier la conception d'un analyseur numérique de parole par prédiction linéaire fonctionnant en temps réel. Cependant, nous n'avons pas cherché à faire une réalisation spécifique à cette simple application. Nous nous sommes placés sur le plan plus général du traitement du signal et des traitements de type vectoriel.

La présente étude s'articule en 2 grandes parties.

Dans la première partie, l'application à traiter est décrite et analysée. La seconde partie porte sur la réalisation proprement dite. Un accent particulier a été mis sur la phase d'analyse de l'application car c'est elle qui oriente pour une grande part la conception architecturale du système à réaliser.

Le chapitre un de la première partie présente tout d'abord un certain nombre de généralités sur la parole. L'appareil phonatoire humain et le principe de production de la parole y sont succintement décrits.

Fant (1960) et Flanagan (1965) ont présentés des modèles simulant le comportement du conduit vocal. Plus récemment, Atel et Hannauer (1971) ont développé un modèle linéaire de production de parole, moyennant certaines hypothèses. Ce modèle est présenté du point de vue analogique et numérique. Enfin, un certain nombre de système de codage de parole sont décrits (Vocodeurs à canaux, à prédiction linéaire ...).

Dans le chapitre deux, nous abordons l'étude de la prédiction linéaire. La théorie de la prédiction linéaire est décrite sur le plan général de l'analyse de séries temporelles. En effet, ses applications ne se limitent pas à la communication parlée, mais elles touchent également des domaines divers, tels que les stat., la neurophysique et la géophysique (cf. MAKHOUL 1975).

Un second sous-chapitre présente les différents modules nécessaires à une analyse complète du signal à l'aide de la prédiction linéaire.

Un convertisseur analogique/numérique discrétise le signal. Il fournit au système d'analyse des échantillons de parole. L'analyse est du type "global", c'est-à-dire qu'elle porte sur un ensemble d'échantillons. Avant leur sortie de l'analyseur, les paramètres extraits sont codés en vue de leur transmission ou d'un traitement ultérieur.

L'étude de l'application permet, dans le chapitre trois, de déterminer certaines spécifications du système à réaliser.

Les décompositions fonctionnelles de l'application sont tout d'abord mises en évidence : des traitements de type "série" et "parallèle" apparaissent avec leurs liaisons éventuelles. Nous indiquons ensuite les paramètres qui définissent les conditions d'analyse. Par exemple, la fréquence d'échantillonnage, la cadence d'analyse, l'intervalle d'analyse, le nombre de bits alloués au codage, etc... L'analyse d'activité du système permet d'évaluer le volume d'opérations à effectuer en temps réel. Elle est présentée dans ce chapitre.

Dans l'optique d'une réalisation performante et la plus simple possible, nous avons effectué une étude statistique mesurant l'effet d'une arithmétique tronquée sur la précision des résultats. Enfin, dans la même optique, nous avons dégagé un certain nombre de primitives vectorielles.

La deuxième partie se décompose également en 3 chapitres.

Dans le premier chapitre, nous sommes amenés à étudier certaines machines existantes, ainsi que des architectures de machines convenant aux types de traitements à effectuer.

Une machine composée de plusieurs unités de traitements est nécessaire. Grâce à l'utilisation de mémoires intégrées à grande capacité et de micro-processeurs, il est possible de définir une machine multi-processeurs, relativement souple d'emploi, gardant comme critère fondamental une grande rapidité de calcul.

Le chapitre deux décrit la structure matérielle de la machine multi-processeurs en cours de réalisation. Les différentes unités composant la machine sont détaillées. Le système est prévu pour comporter de 1 à 8 processeurs identiques, communiquant entre eux grâce à une mémoire commune. Chaque processeur est en fait un ordinateur à part entière, possédant son unité centrale, sa mémoire locale et son système d'entrée/sortie.

Le chapitre trois, enfin, étudie le logiciel de support de la machine.

Le premier sous-chapitre est consacré à la présentation de la programmation de la machine. Les spécifications d'un langage de haut niveau pour l'utilisateur y sont définies. A un niveau plus élémentaire un langage d'assemblage est décrit, ainsi que la composition d'une micro-instruction.

La liaison entre le niveau supérieur et le niveau élémentaire se fait grâce à un langage intermédiaire, dans lequel chaque instruction correspond à un appel de micro-programme.

Dans le second sous-chapitre, une réflexion sur les problèmes liés aux synchronisations et communications dans le système multi-processeurs est amorcée.

Nous tentons de décrire une solution adaptée à l'application.

PREMIERE PARTIE

L'APPLICATION

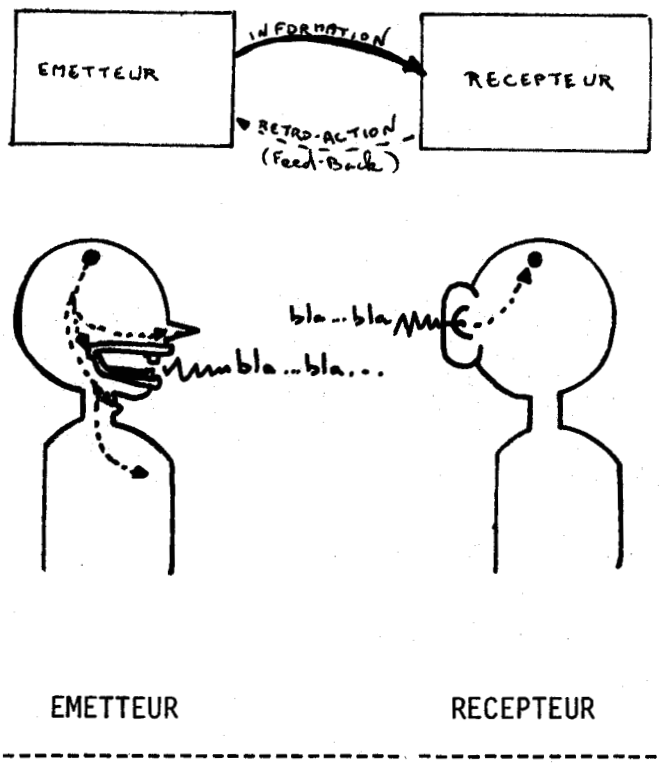
CHAPITRE UN - LA PAROLE ET SON ENVIRONNEMENT

...

CHAPITRE UN - LA PAROLE ET SON ENVIRONNEMENT

I.1/ GENERALITES SUR LA PAROLE

La parole est un moyen privilégié de communication entre les êtres humains. Le dessin ci-dessous représente de manière schématique la manière dont une personne transmet une information ("parle") à une autre.



La source d'information se situe dans le cerveau de l'émetteur, à ce niveau, c'est une "pensée". Cette pensée est traduite toujours au niveau du cerveau en unités linguistiques, puis en commandes neuro-motrices des différents muscles mis en jeu pour parler.

...

La contraction de ces différents muscles provoque la production des sons désirés. Si ce qui se passe au niveau du cerveau n'est pas encore totalement compris, le système de phonation, lui est à peu près connu. Nous le décrirons par la suite.

La transmission de l'information se fait à travers l'air qui sépare l'émetteur du récepteur. L'homme a inventé d'autres moyens de transmissions, comme le téléphone ou la radiophonie. Au niveau de la transmission, il y a presque toujours une perte d'informations. Cette perte peut être due aux bruits parasites environnants, à l'éloignement des 2 personnes, à la limitation de la bande passante du téléphone ou à toute autre perturbation. L'émetteur a pu également mal interpréter sa pensée (mauvaise prononciation).

C'est donc un message en général tronqué qui parvient aux oreilles du récepteur.

L'activité du système auditif du récepteur va de l'analyse du signal reçu au niveau des oreilles jusqu'à la compréhension du message au niveau du cerveau.

Les schémas 1 et 2 résument respectivement les processus d'émission et de réception du message.

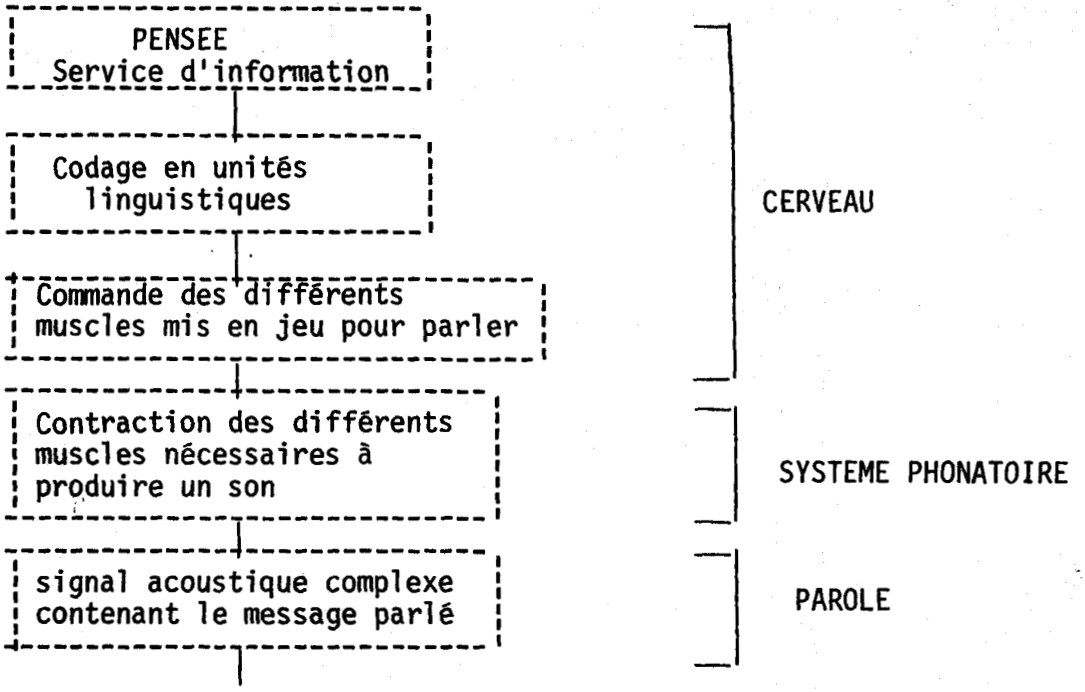


Schéma 1 - Processus d'émission

...

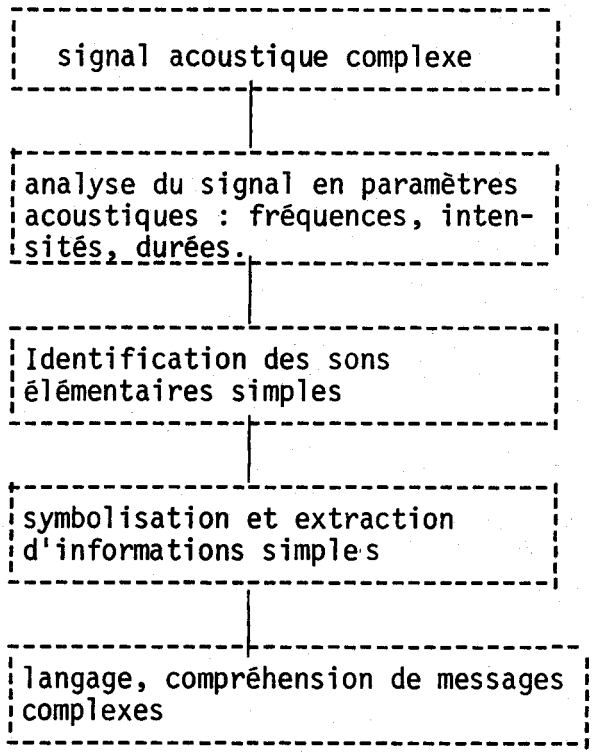


Schéma 2 : processus de réception

Le récepteur fait appel aux règles du langage afin de saisir la signification du message. Par ailleurs, il pallie à la mauvaise réception du message (distorsions, ambiguités, parasites) en exploitant les contraintes phonétiques et linguistiques propres au langage ainsi que les redondances du message émis.

Ces redondances apparaissent à 2 niveaux :

- le contenu du message peut être redondant : "je monte en haut", "je descend en bas"...

- le signal acoustique de parole présente également une certaine redondance. En effet, seule une partie de l'information acoustique qui frappe l'oreille est nécessaire à la compréhension d'un mot.

...

Nous verrons que l'objectif principal de l'analyse de la parole est d'éliminer les redondances inhérentes au signal acoustique. Il s'agit donc de décrire ce signal en fonction de paramètres "pertinents" qui le caractérisent. Si cette analyse permet de réduire l'information à traiter, elle présente un danger certain : la moindre perte d'information non redondante sera catastrophique pour la compréhension du message.

Avant d'analyser le signal vocal, il est nécessaire d'étudier la manière dont il est produit par l'appareil phonatoire et la manière dont il est perçu par l'appareil auditif.

Historiquement, on s'est d'abord intéressé à la manière dont il était produit. L'appareil phonatoire était en effet plus facile à observer que l'appareil auditif. C'est ainsi que dès le 18ème siècle, des "modèles mécaniques" de production de parole sont apparus. Les moyens électroniques et informatiques actuels ont permis de créer des modèles plus simples simulant le comportement du conduit vocal.

Le paragraphe suivant décrira l'appareil phonatoire, ainsi qu'un modèle simple de production de parole. Si les "modèles d'oreilles" ne sont pas étudiés ici, il faut savoir qu'ils se développent depuis peu, et qu'ils représentent un moyen supplémentaire d'analyse du signal vocal (cf. CAELEN 1974, Haton et Perennou 1978).

...

I.2 / LA PRODUCTION DE LA PAROLE

La phonation n'est en fait qu'une spécialisation de divers organes dont la fonction essentielle est soit de nous permettre de respirer,soit de nous permettre de boire et manger.

I.2.1. L'appareil phonatoire humain (Cf. figures 1 et 2)

L'énergie nécessaire à la production d'un son est fournie par le souffle respiratoire. C'est l'ensemble des sons élémentaires (ou phonèmes) prononcés qui constituent le message parlé. La source d'excitation essentielle est située dans le larynx,au niveau des cordes vocales.

La source d'énergie : l'organe respiratoire .

La contraction des muscles thoraciques et du diaphragme provoque l'expulsion de l'air des poumons. Cet air arrive sous pression (via la trachée) à l'entrée du conduit vocal (qui va des cordes vocales aux lèvres).

L'oscillateur : les cordes vocales .

Les cordes vocales se comportent de manière différente suivant le type de son prononcé :

- Lors de la génération d'un son dit "voisé",comme les voyelles, l'air des poumons va provoquer un phénomène d'accolements / décollements des cordes (Figure 3.1). Ce phénomène est quasi-périodique. La fréquence de vibration des cordes ,ou fréquence fondamentale,varie suivant le phonème prononcé et l'individu qui l'a prononcé. Ainsi,une voix de femme se caractérise par une fréquence fondamentale (notée Fo) plus élevée que celle d'un homme.

-Lors de la génération d'un son "non-voisé",comme certaines consonnes,les cordes vocales laissent passer l'air des poumons (Figure 3.2). C'est alors la turbulence provoquée par le passage de l'air dans une constriction au niveau des cavités bucco-nasales qui va produire le son.

Le résonateur : le conduit vocal .

Le conduit vocal se compose du conduit oral,et du conduit nasal.

Le conduit oral comprend la cavité pharyngale et la cavité buccale (lèvres dents,palais).C'est un tube cylindrique de section variable (de 0 à 20 cm^2), et de longueur à peu près égale à 17 centimètres .

La cavité nasale est branchée en dérivation sur le conduit oral,mais n'intervient que lors de la prononciation de sons "nasalisés",comme /n/.

Les sons élémentaires utilisés en français sont décrits en annexe I . Outre la distinction voisé/non-voisé, plusieurs classes de phonèmes apparaissent:

- Les voyelles orales : La "luette" située au niveau du velum empêche l'air de passer dans le conduit nasal. Les sons prononcés sont donc sans nasalité.
- Les voyelles nasales : Elles mettent en jeu la résonance de la cavité nasale. Le degré de couplage entre la cavité buccale et la cavité nasale détermine la nasalité plus ou moins importante d'un son. Le rayonnement du son provient alors des lèvres et des narines.
- Les semi-voyelles et les consonnes liquides ont un comportement assez voisin des voyelles orales.
- Les consonnes plosives : Elles sont générées en retenant l'air dans la cavité buccale,et en le laissant échapper brusquement.

Certaines sont sourdes(non-voisées) : /p/,/t/,/k/ ;les autres sont"sonores" (voisées) : /b/,/d/,/g/ .

- Les consonnes nasales : C'est le conduit nasal qui est principalement mis à contribution lors de la production de ce type de son.
- Les consonnes fricatives : Pour ces consonnes,il n'y a pas contribution de la cavité nasale . Les fricatives voisées (/v/,/z/,/ʒ/) correspondent à une double excitation (cordes vocales + constriction).

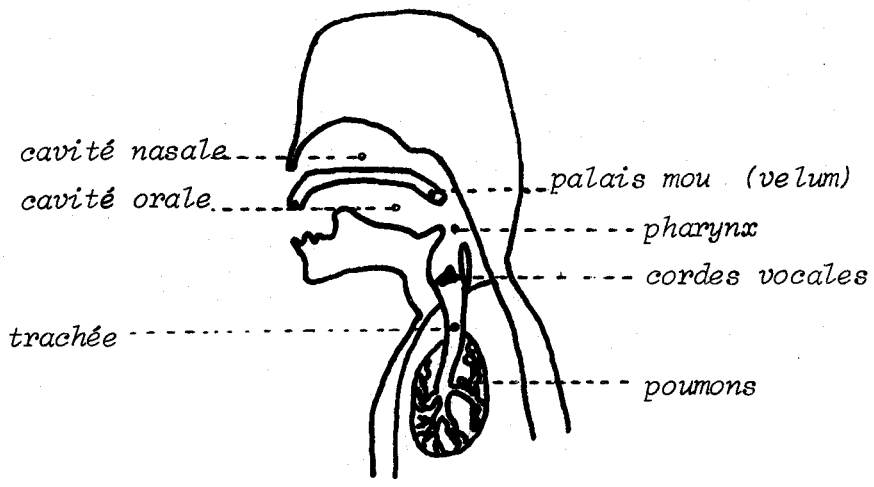


figure 1 . L'appareil phonatoire .

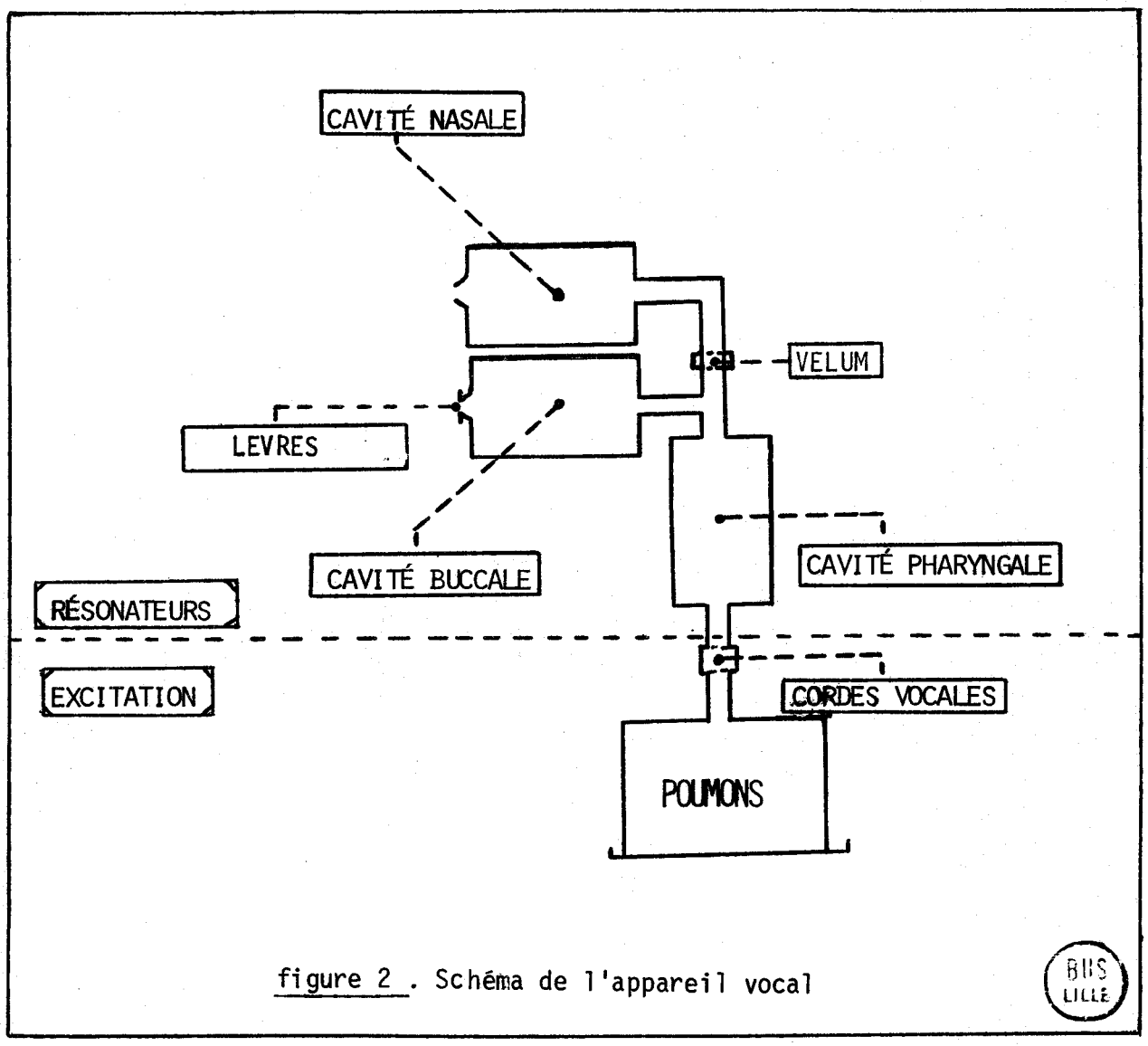


figure 2 . Schéma de l'appareil vocal

I.2.2. Modèle de production de parole

Un modèle de production de parole doit pouvoir simuler le fonctionnement de l'appareil phonatoire humain. Le schéma de la figure 4 décrit un modèle général de production. Il reprend les différents éléments de la figure 2 .

L'appareil vocal produit un signal dont la structure est complexe, aussi est-il nécessaire de définir des modèles relativement simples pour simuler son comportement.

Nous avons vu que les sons produits étaient le résultat de l'excitation acoustique du conduit vocal. Cette excitation prend deux aspects :

-Dans le cas des sons voisés, c'est une suite de pulsations quasi-périodiques (de période Po = 1 / Fo),générées par les cordes vocales. Un GENERATEUR D'IMPULSIONS pourra simuler la vibration des cordes vocales.

-Dans le cas des sons non-voisés, c'est la turbulence de l'air passant dans une constriction du conduit vocal. Une SOURCE DE BRUIT pourra permettre de générer ce type de sons.

Un modèle simple de production de parole ne peut s'envisager que si les hypothèses suivantes sont posées:

- Sur un intervalle de temps relativement court (10 à 20 millisecondes), on peut considérer que la forme du conduit vocal ne varie pas . Les variations du conduit vocal seront alors représentées par une suite de configurations stationnaires.

- L'appareil vocal se comporte comme une ligne de transmission possédant une fonction de transfert dans le domaine complexe z . Cette fonction de transfert peut s'exprimer par un filtre possédant des pôles et des zéros.

Fant (1960) et Flanagan (1965) ont développé des modèles de production de parole à partir de ces hypothèses.

Atal et Hanauer (1971) ont développé un modèle linéaire où les contributions de la source (les cordes vocales), du conduit vocal et du rayonnement aux lèvres pouvaient être représentées par un filtre récursif tout-pôle. Pour cela,les hypothèses suivantes étaient formulées:

1) Le conduit vocal se compose d'un ensemble de "p" tubes acoustiques cylindriques successifs,de même longueur,mais de sections différentes(cf. figure 5) .

2) La longueur de chaque tube est suffisamment petite en comparaison de la longueur d'onde du signal pour permettre de considérer la propagation du son dans un tube comme une onde plane.

3) Les tubes successifs sont rigides.

4) Les hypothèses relatives aux équations de propagation des ondes sont valides.

5) Le modèle est linéaire et indépendant de l'effet de la source.

6) Les effets du conduit nasal peuvent être négligés.

Le modèle de production de parole qui en résulte est présenté dans les figures 6 et 7.

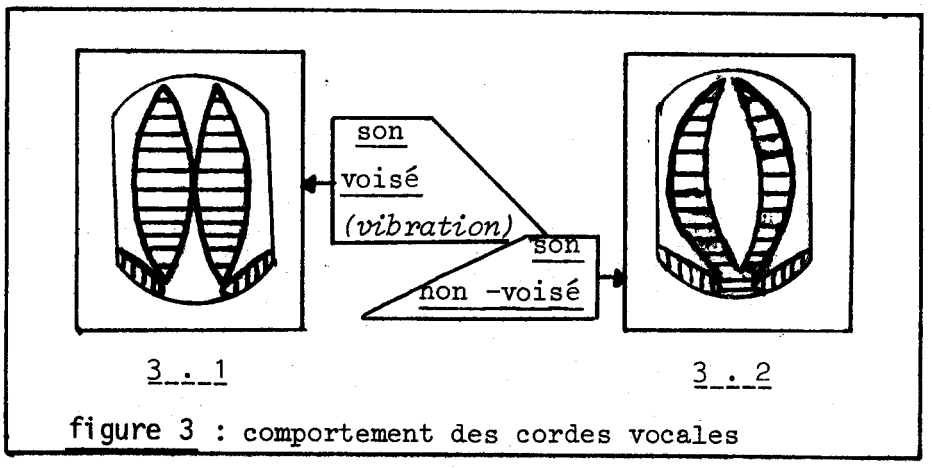


figure 3 : comportement des cordes vocales

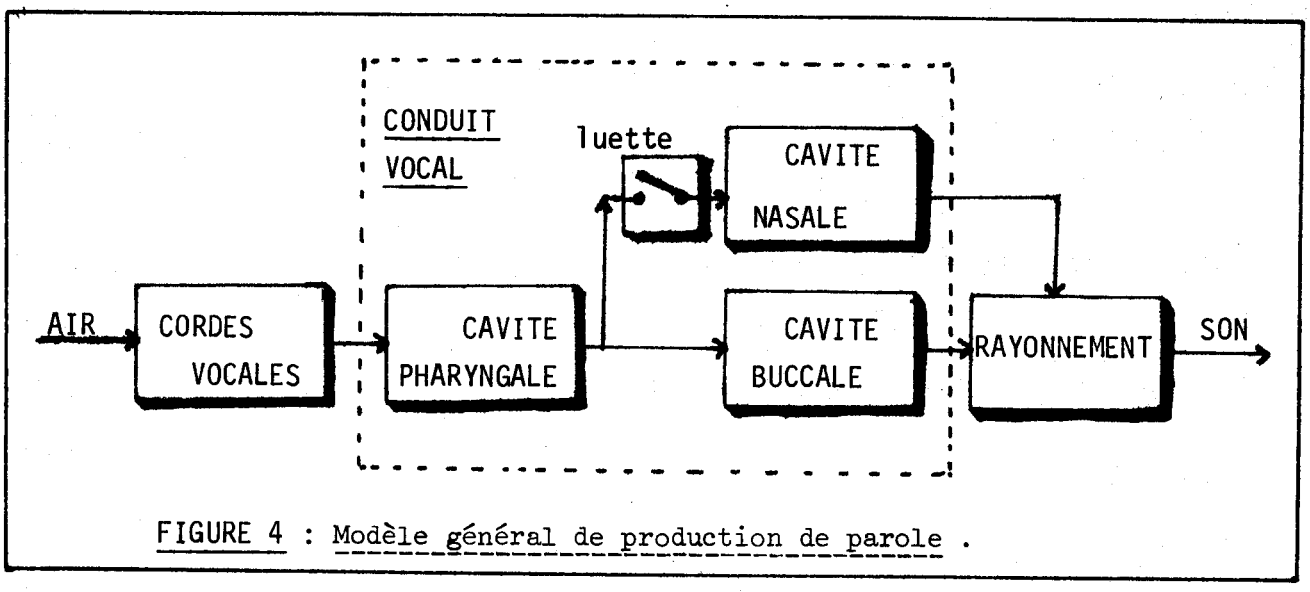


FIGURE 4 : Modèle général de production de parole .

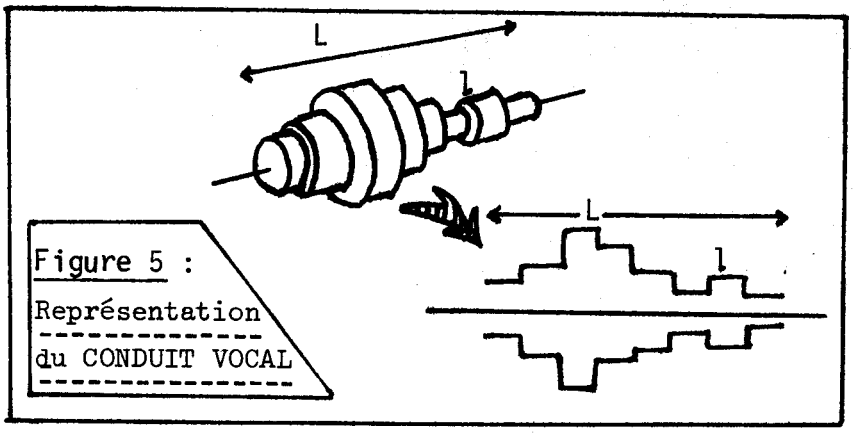


Figure 5 :
Représentation
du CONDUIT VOCAL

figure 6 : MODELE ANALOGIQUE DE PRODUCTION DE PAROLE

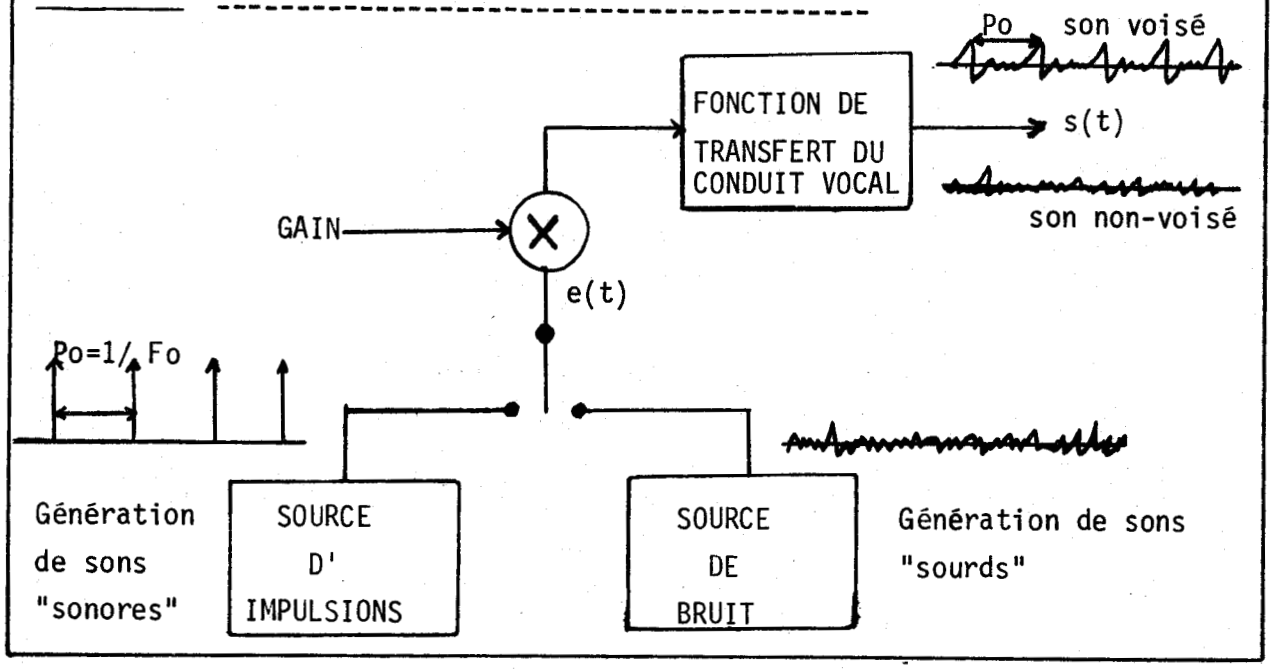
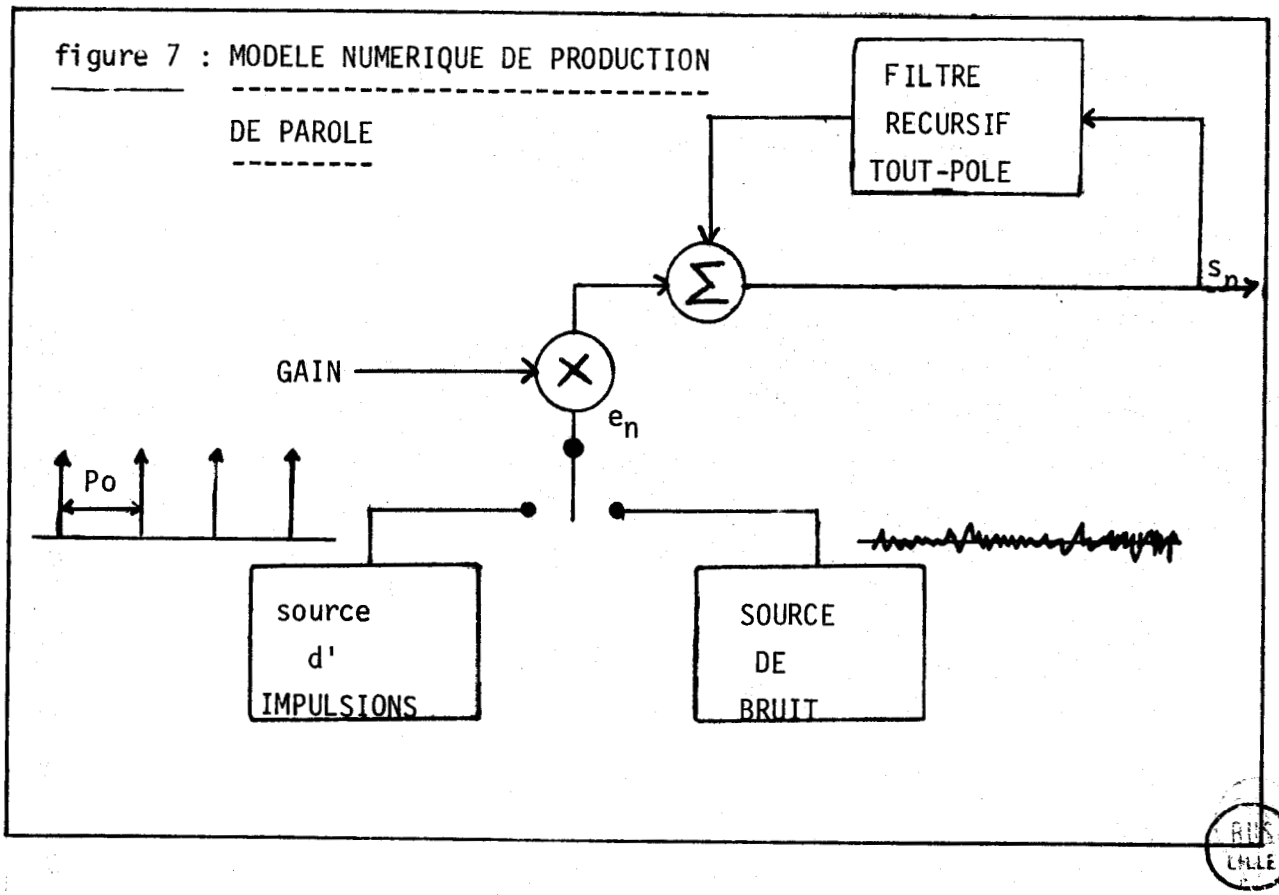


figure 7 : MODELE NUMERIQUE DE PRODUCTION DE PAROLE



I.3 / LE CODAGE DU SIGNAL DE PAROLE

Le codage du signal de parole a pour but l'élimination des redondances inhérentes à ce signal. Un nombre restreint de paramètres peut décrire le signal vocal, permettant ainsi une transmission à des débits peu importants, et le traitement d'une masse de données plus faible.

Le codage de la parole comporte deux phases distinctes:

- Une phase d'analyse, où les caractéristiques du conduit vocal sont extraites, puis codées...

- Une phase où un synthétiseur décode ensuite l'information, et la restitue.

Ces deux phases sont utilisables séparément :

- L'analyse peut être suivie d'un système de reconnaissance de la parole...(*figure a*)

- Le synthétiseur peut être précédé d'un système de génération de synthèse de la parole...(*figure b*)

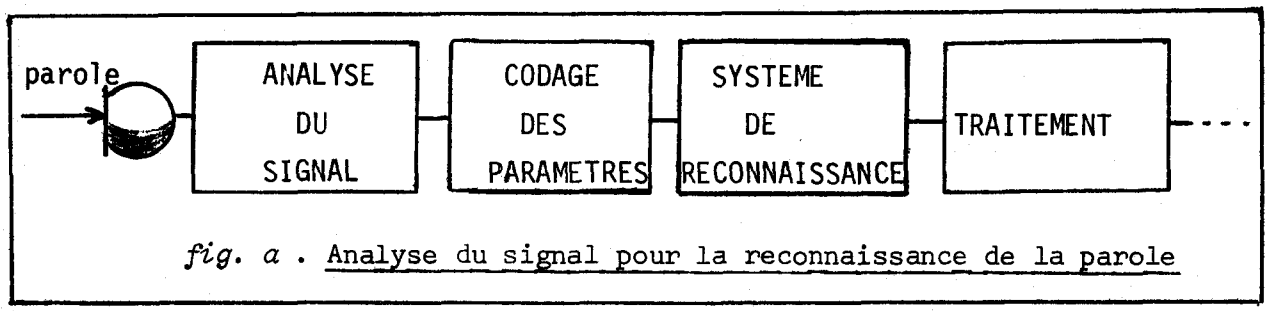


fig. a . Analyse du signal pour la reconnaissance de la parole

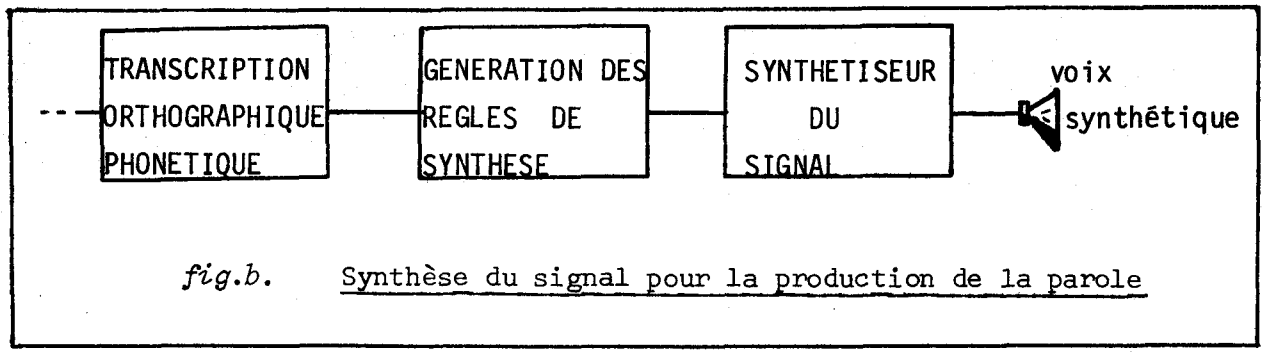


fig.b. Synthèse du signal pour la production de la parole

Différents systèmes de codage ont été étudiés,et,pour certains,réalisés. Les uns font appel à des méthodes spectrales d'analyse (Vocodeurs à canaux, à formants,à bande de base...); les autres font appel à des méthodes temporelles d'analyse (Vocodeurs à prédiction linéaire ...). Nous allons présenter succintement les principaux systèmes :

-Le vocodeur à canaux *(cf. GOLD & RADER 1967)* : Le vocodeur à canaux (fig. 1) est actuellement le plus utilisé pour les codages de parole à faible débit. Le mot " *vocodeur* " vient d'une contraction anglaise de "VOice CODER" .

Le vocodeur à canaux permet d'approcher le spectre à court terme du signal. La première version de ce type de codeur a été présentée en 1939 par Dudley.

La partie analyse se compose d'un banc de filtres (jusqu'à une vingtaine) permettant le découpage du signal en bandes de fréquences contigües. L'énergie du signal est calculée dans chaque bande de fréquences. Un canal supplémentaire sert à l'extraction de la fréquence fondamentale F_0. Les informations obtenues (énergie dans chaque canal + F_0) sont codées,puis transmises,soit à un système de reconnaissance,soit à la partie "synthétiseur" du vocodeur afin de restituer la parole.

La qualité de la parole de synthèse dépend du nombre de canaux,de l'étendue de la bande des fréquences analysées,et du nombre de bits utilisés pour coder les paramètres.

-Le vocodeur à bande de base *(cf. EL MALLAWANY & al 1973)* : Le vocodeur à bande de base est utilisé uniquement dans des applications d'analyse /Synthèse de la parole. La partie analyse se compose,comme pour un vocodeur à canaux,d'un ensemble de filtres passe-bande . Cependant, les basses fréquences (inférieures à 800 Hz),sont transmises intégralement au synthétiseur. C'est en effet dans cette bande de fréquence que se situe la fréquence fondamentale. A l'aide de traitements non-linéaires portant sur la bande de base,on obtient un signal avec des harmoniques haute fréquence pour exciter les canaux hautes-fréquences.

-Le vocodeur à formants : Le conduit vocal peut être caractérisé par ses résonances principales ou " FORMANTS " . Dans la bande téléphonique,on trouve en moyenne trois formants .

Le vocodeur à formants " *CIPHON*" ,étudié par *Thomson-Csf*,est un exemple de ce type de vocodeurs (cf. *BOURGENOT & DECHAUX 1975*) .

Le débit de transmission du vocodeur à formants a l'avantage d'être très faible (de l'ordre de 1200 bits par seconde). Seules les caractéristiques des formants et la fréquence fondamentale sont transmises. Il en résulte toutefois une qualité de synthèse médiocre,mais des applications de reconnaissance de parole sont envisageables. Il existe des vocodeurs à Formants dont le débit est supérieur, et qui permettent d'obtenir une bonne voix de synthèse.

-Le système CEPSTRUM *(cf. SCHAFER & RABINER 1970)* : Ce système permet de coder de manière dite "homomorphique" le signal vocal. Il demande une définition spectrale poussée. Grâce à deux transformées de Fourier successives,on sépare le signal source (périodique) et le signal apériodique dû aux évolutions articulatoires relativement lentes du conduit vocal.

Cette méthode nécessite cependant de nombreux calculs,et une application en temps réel est possible,mais délicate à mettre en oeuvre, du moins dans le cas de l'analyse + synthèse .

-Le vocodeur à prédiction linéaire *(cf. figure2)* : Les méthodes de prédiction linéaire appliquées au traitement du signal vocal sont de plus en plus utilisées depuis une dixaine d'années. Dans les vocodeurs précédemment décrits,les données acoustiques étaient traduites sous forme spectrale. Les caractéristiques de quasi-périodicité et de non stationnarité du signal vocal y étaient cependant mal représentées. Par contre une analyse temporelle du signal de parole par un *modèle* de conduit vocal permet de tenir compte des données articulatoires(cf. I.2).

Atal et Hanauer (1971) ont montré qu'il était possible d'approcher la forme du conduit vocal à un instant donné grâce à un modèle composé de p tubes cylindriques successifs de longueurs égales,mais de sections différentes. Cette modélisation permet de définir une "fonction de transfert" du conduit vocal, constituée par un filtre récursif tout-pôle.

Différents modèles de conduits vocaux ont été étudiés. L'intérêt du modèle à prédiction linéaire tient au fait qu'il permet d'estimer directement les paramètres du filtre à partir de la forme acoustique du signal vocal.

Markel et Gray ont décrit récemment tous les aspects de la prédiction linéaire appliquée à la parole (*"LINEAR PREDICTION OF SPEECH " 1976)* .

La *figure 2* décrit un vocodeur à base de prédiction linéaire. La phase d'analyse permet :

-L'extraction des "p" coefficients du filtre modèle à partir du signal vocal numérisé;

-Le calcul du gain du système,afin d'avoir un niveau de sortie de la parole de synthèse comparable au niveau d'entrée;

-La détection de la fréquence fondamentale F_0,définissant l'excitation périodique ou non du filtre de synthèse.

Après codage,les paramètres trouvés sont transmis avec un débit de l'ordre de 5 kbits par seconde à la partie synthétiseur . Le synthétiseur permet la restitution approchée du signal original.

Par la suite,c'est la partie "analyse" de ce vocodeur qui nous intéressera plus particulièrement en vue d'une réalisation matérielle.

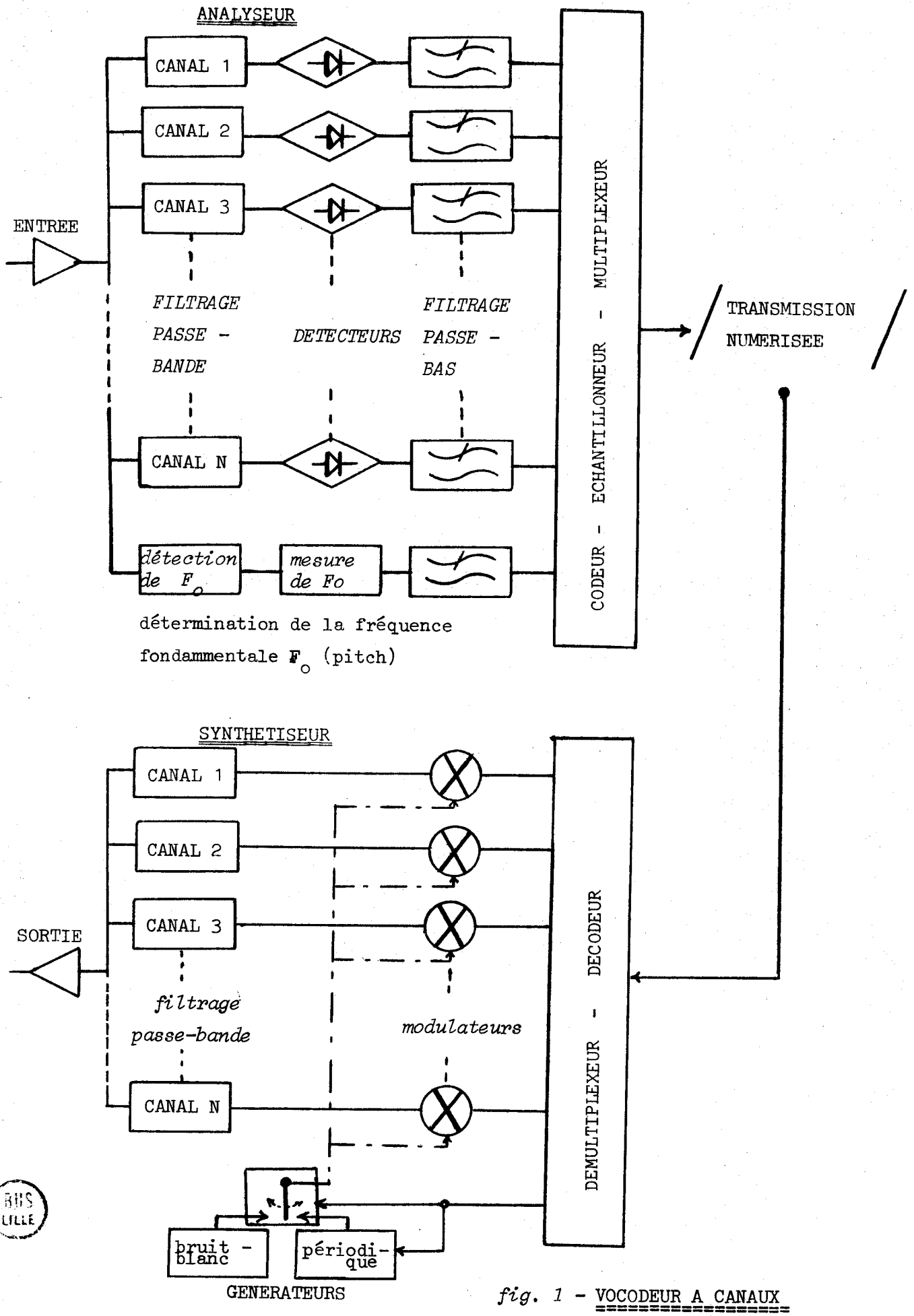


fig. 1 - VOCODEUR A CANAUX

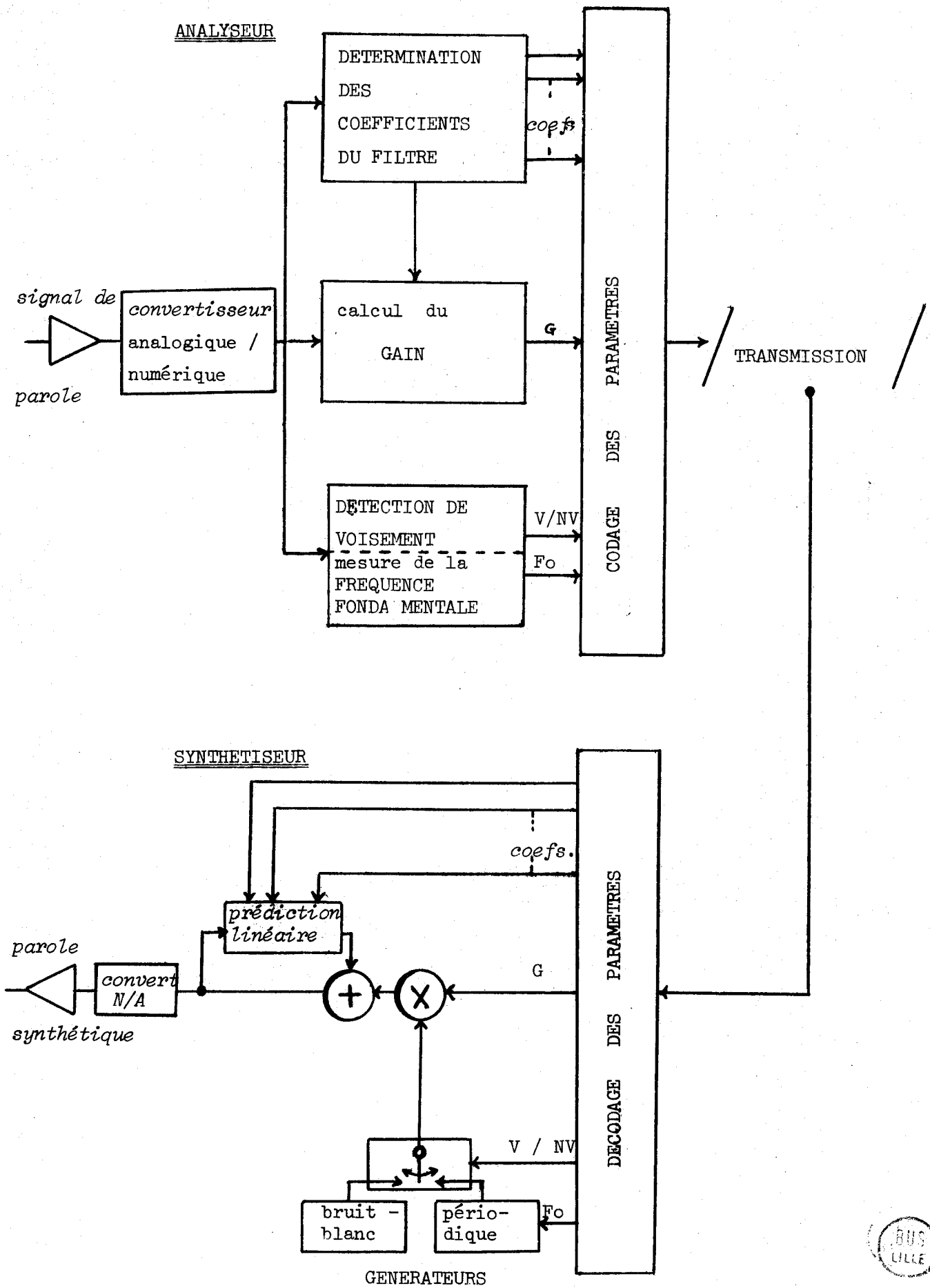


fig.2- VOCODEUR A PREDICTION LINEAIRE

CHAPITRE DEUX

L' ANALYSE PAR PREDICTION LINEAIRE

II. 1 / LA PREDICTION LINEAIRE

...

II. 2 / SCHEMA GENERAL D'UNE ANALYSE

CHAPITRE DEUX : L'ANALYSE PAR PREDICTION LINEAIRE

II.1. LA PREDICTION LINEAIRE

II.1.1 / APPROCHE

Pour analyser un signal suivant un modèle temporel, il faut d'abord l'échantillonner . D'un signal continu s(t) , on obtiendra un signal discret s(nT) , où T représente la période d'échantillonnage du signal et où n est une variable entière .

La fréquence d'échantillonnage du signal f_e sera définie par :

$$f_e = 1 / T$$

Il s'agit maintenant de trouver un modèle paramétré simple pour représenter le signal .

l'équation (1) donne un des modèles les plus utilisés :

$$(1) \quad \begin{cases} s(nT) = G \cdot \sum_{m=0}^{q} b_m \, e(\overline{n-m}\ T) + \sum_{i=1}^{p} a_i \cdot s(\overline{n-i}\ T) \\ b_0 = 1 \end{cases}$$

Le signal discret s(nT) est la sortie d'un système dont l'entrée e(nT) est inconnue .

Paramètres du système:

a_i ; i variant de 1 à p .

b_m ; m variant de 1 à q .

G : gain du système .

En notant de manière simplifiée s(nT) par s_n et e(nT) par e_n , l'équation (1) se réécrira :

$$(2) \quad \begin{cases} s_n = G \cdot \sum_{m=o}^{q} b_m \cdot e_{n-m} + \sum_{i=1}^{p} a_i \cdot s_{n-i} \\ b_o = 1 \end{cases}$$

L'échantillon s_n est donc prédit grâce à une combinaison linéaire des p sorties et des q entrées précédentes . D'où la dénomination :

"Prédiction Linéaire" .

Dans le domaine des fréquences, l'équation (2) peut se représenter en prenant sa transformée en z :

$$(3) \qquad \boxed{S(z) = H(z) \cdot E(z)}$$

avec : $S(z) = \sum_{+\infty}^{-\infty} s_n \, z^{-n}$ (transformée en z de s_n)

$E(z) = \sum_{+\infty}^{-\infty} e_n \, z^{-n}$ (transformée en z de e_n)

La fonction de transfert $H(z) = \frac{S(z)}{E(z)}$ représentera le modèle général "pôle-zéro" . On déduit H(z) des équations (2) et (3) :

$$(4) \qquad \boxed{H(z) = G \cdot \frac{(1 + \sum_{n=1}^{q} b_m \cdot z^{-m})}{(1 - \sum_{i=1}^{p} a_i \cdot z^{-i})}}$$

Les racines du polynôme du numérateur sont les "zéros" du filtre modèle .

Les racines du polynôme du dénominateur sont les "pôles" du filtre modèle.

Deux cas particuliers intéressants peuvent se présenter :

- Le cas où les a_i sont nuls ,pour i variant de 1 à p . Le dénominateur sera alors égal à 1, et le filtre défini par sa fonction de transfert H(z) sera "tout-zéro" .

- Le cas où les b_m sont nuls , pour m variant de 1 à q . Le numérateur sera alors égal à G ,et le filtre sera dit : "tout-pôle"

La plupart des applications porte sur des filtres "tout-pôle",dits " filtres autorégressifs ". C'est ce type de filtres qui va être étudié plus en détail .

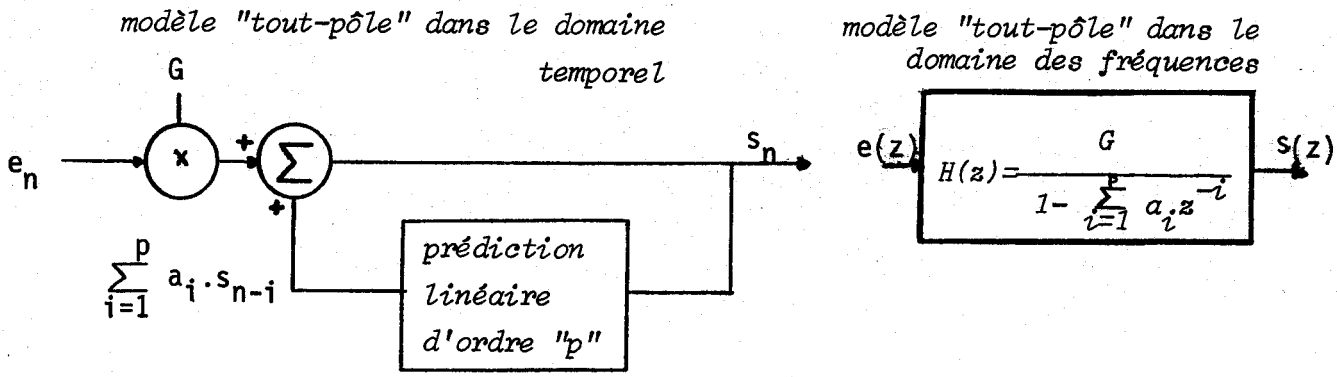
II.1.2 / DEFINITION

Si $b_m = 0$ pour m variant de 1 à q , les équations (2) et (4) deviennent respectivement :

$$(5) \qquad s_n = G \cdot e_n + \sum_{i=1}^{p} a_i \cdot s_{n-i}$$

$$(6) \qquad H(z) = \frac{G}{1 - \sum_{i=1}^{p} a_i \cdot z^{-i}}$$

Les Schémas ci-dessous représentent ces deux équations :



Définition de la prédiction linéaire : Un échantillon de signal"s_n" peut être déduit d'une combinaison linéaire des"p"échantillons le précédant immédiatement et d'une entrée e_n (équation (5)) .

II.1.3 / RESOLUTION : Les moindres carrés .

Le problème consiste donc à calculer à partir de l'équation (5) le facteur de gain G, ainsi que les paramètres a_i , i variant de 1 à p .
Les a_i sont appelés les " coefficients de prédiction " du filtre .

Pour la résolution d'un tel système,on supposera le signal d'entrée e_n inconnu. Soit $\hat{s}_n$ la valeur approchée de s_n, l'équation (5) devient :

(7) $$\hat{s}_n = \sum_{i=1}^{p} a_i \cdot s_{n-i}$$

L'erreur de prédiction e'_n sera égale à $G \cdot e_n$, puisque :

(8) $$s_n = \hat{s}_n + G \cdot e_n$$

Il est possible d'exprimer l'erreur de prédiction en fonction de l'échantillon de signal s_n et de sa valeur approchée :

(9) $$e'_n = s_n - \hat{s}_n = s_n - \sum_{i=1}^{p} a_i \cdot s_{n-i}$$

Il s'agit de touver les "a_i" qui minimisent l'erreur de prédiction e'_n pour l'ensemble des échantillons du signal .

Soit E,l' "erreur quadratique totale de prédiction" :

$$E = \sum_{n} e'^{2}_{n} .$$

En remplaçant e'_n par sa valeur définie dans l'équation (9),on aura :

(10) $$E = \sum_{n} (s_n - \hat{s}_n)^2 = \sum_{n} (s_n - \sum_{i=1}^{p} a_i \cdot s_{n-i})^2$$

Les valeurs a_k, k variant de 1 à p, minimisant E au sens des moindres carrés, s'obtiennent en annulant la dérivée partielle de E par rapport à chaque a_k :

$$\frac{\delta E}{\delta a_k} = 0 \quad ; \text{ pour k variant de 1 à p}$$

Soit :

$$(11) \qquad \boxed{\frac{\delta E}{\delta a_k} = \sum_n 2 \left(s_n - \sum_{i=1}^{p} a_i \cdot s_{n-i} \right) \cdot \left(- s_{n-k} \right) = 0}$$

Ce qui donne le système d'équations :

$$(12) \qquad \boxed{\sum_{i=1}^{p} a_i \cdot \sum_n s_{n-i} \cdot s_{n-k} = \sum_n s_n \cdot s_{n-k}}$$

k variant de 1 à p

Ceci est un système linéaire de "p" équations simultanées à "p" inconnues. La résolution de ce système permettra d'obtenir les coefficients a_i du filtre . Deux méthodes d'approche ont été largement utilisées pour cela : La première est fondée sur l'autocorrélation du signal;la seconde sur la covariance du signal.

II.1.3.1 / Méthode d'autocorrélation

Deux hypothèses d'approche sont nécessaires :

Hypothèse 1 . *Nous supposerons que l'écart quadratique total E, défini dans l'équation (10) , est minimisé sur une durée infinie (n variant de $-\infty$ à $+\infty$) . L'équation (12) devient :*

$$(13) \qquad \boxed{\sum_{i=1}^{p} a_i \sum_{-\infty}^{+\infty} s_n \cdot s_{n+|k-i|} = \sum_{n=-\infty}^{+\infty} s_n \cdot s_{n+|k|} \quad ;}$$

k variant de 1 à p

Soit R_k *la fonction d'autocorrélation du signal :*

(14) $$\boxed{R_k = \sum_{-\infty}^{+\infty} s_n \cdot s_{n+|k|}}$$

L'équation (13) peut alors s'exprimer par :

(15) $$\boxed{\sum_{i=1}^{p} a_i \cdot R_{|k-i|} = R_k} \; ; \; \underline{k \text{ variant de } 1 \text{ à } p} \; .$$

Remarque : $\underline{R_{-k} = R_k}$

Hypothèse 2 . *Le signal n'est intéressant (ou n'est connu) que sur un intervalle fini . Aussi, une fonction de fenêtre lui sera appliquée afin d'obtenir un signal* s'_n *,nul à l'extérieur d'un intervalle de N échantillons :*

$$\begin{cases} s'_n = s_n \cdot f_n \; , & \text{pour } n \text{ variant de } 0 \text{ à } N-1 \; ; \\ s'_n = 0 \; , & \text{pour } n \text{ négatif ou supérieur à } N-1 \; ; \end{cases}$$

f_n *représente la fonction de fenêtre .*

La fonction d'autocorrélation R_k *,définie dans l'équation (14) devient avec la nouvelle hypothèse :*

(16) $$\boxed{R_k = \sum_{n=0}^{N-1-|k|} s'_n \cdot s'_{n+|k|}} \; , \; \underline{\text{avec } k \text{ positif ou nul}} \; ;$$

L'erreur de prédiction e'_n *est minimisée sur l'intervalle* $[0 \; , \; N-1+p]$

II.1.3.2 / Méthode de covariance

Une hypothèse d'approche est également nécessaire :

Hypothèse . *Contrairement à la méthode d'autocorrélation, nous supposerons que l'erreur quadratique totale E ,définie dans l'équation (10) , sera minimisée sur un intervalle fini correspondant à N échantillons :*

$$\text{Echantillons} \left[s_n \right] = \left[s_0, s_1, \cdots, s_{N-1} \right]$$

La fonction de covariance du signal s'écrira :

$$(17) \qquad \boxed{c_{ik} = \sum_{n=p}^{N-1} s'_{n-k} \cdot s'_{n-i}}$$

Le système d'équations défini en (12) deviendra :

$$(18) \qquad \boxed{\sum_{i=1}^{p} a_i \cdot c_{ki} = c_{k0}} \qquad \text{, pour } k \text{ variant de 1 à } p \text{ ;}$$

Remarque : $c_{ik} = c_{ki}$

L'erreur de prédiction e'_n *sera minimisée sur l'intervalle* $[p , N-1]$.

II.14 / Calcul des coefficients du filtre

En résumé, les deux méthodes d'approche sont définies dans les deux ensembles qui suivent :

METHODE D'AUTOCORRELATION

Résoudre pour k variant de 1 à p :

(19.1) $$\sum_{i=1}^{p} a_i \cdot R_{|k-i|} = R_k$$

Avec la fonction d'autocorrélation :

(19.2) $$R_k = \sum_{n=0}^{N-1-|k|} s'_n \cdot s'_{n+|k|}$$

où k est positif ou nul

Et l'erreur de prédiction, pour n variant de 0 à N-1+p :

(19.3) $$e'_n = s'_n - \sum_{i=1}^{p} a_i \cdot s'_{n-i}$$

METHODE DE COVARIANCE

Résoudre pour k variant de 1 à p :

(20.1) $$\sum_{i=1}^{p} a_i \cdot c_{ki} = c_{k0}$$

Avec la fonction de covariance :

(20.2) $$c_{ik} = \sum_{n=p}^{N-1} s'_{n-k} \cdot s'_{n-i}$$

Et l'erreur de prédiction, pour n variant de p à N-1 :

(20.3) $$e'_n = s'_n - \sum_{i=1}^{p} a_i \cdot s'_{n-i}$$

Soit à résoudre les systèmes linéaires de "p"équations à "p" inconnues défini s par (19.1) et (20.1) . Parmi les différents algorithmes de résolution proposés (Gauss,Gauss-Jordan ,Jacobi ...) ,le plus adapté est celui qui utilise la méthode dite de la "racine carrée"*(cf. KUNZ 1957)*. En effet,les matrices des coefficients $R_{|k-i|}$ et c_{ki} sont des matrices de covariance dans les deux cas . Or une matrice de covariance est symétrique et définie positive (en général), d'où la méthode employée (nécessitant moins de calculs et de place mémoire qu'une autre) .

II.L4.1 / Calcul des coefficients de corrélation

Le système (19.1) peut s'écrire sous sa forme matricielle :

$$\boxed{R \,.\, A = r}$$

Ou :

$$\begin{bmatrix} R_0 & R_1 & R_2 & \cdots & R_{p-1} \\ R_1 & R_0 & R_1 & \cdots & R_{p-2} \\ \vdots & \vdots & \vdots & & \vdots \\ R_{p-1} & R_{p-2} & R_{p-3} & \cdots & R_0 \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix} = \begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_p \end{bmatrix}$$

Remarque : *La matrice de corrélation R ,de dimension p par p , est une matrice symétrique de type "Toeplitz" .*

L'expression de la fonction d'autocorrélation (19.2) permet de calculer les coefficients d'autocorrélation : $\{R_0, R_1, R_2, R_3, \ldots, R_p\}$. La procédure de calcul est la suivante :

P1	COEFFICIENTS D'AUTOCORRELATION

```
pour k variant de 0 à p faire:
début
    R(k) = 0.
    pour n variant de 0 à N-1-k faire:
    début
        R(k) = R(k) + s(n)*s(n+k)
    fin
fin
```

II.1.4.2 / Calcul des coefficients de covariance

Le système (20.1) s'écrit sous sa forme matricielle :

$$\boxed{C \cdot A = c}$$

Soit :

$$\begin{bmatrix} c_{11} & c_{12} & c_{13} & \cdots & c_{1\rho} \\ c_{12} & c_{22} & c_{23} & \cdots & c_{2\rho} \\ \vdots & & & & \vdots \\ c_{1\rho} & c_{2\rho} & c_{3\rho} & \cdots & c_{\rho\rho} \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_\rho \end{bmatrix} = \begin{bmatrix} c_{10} \\ c_{20} \\ \vdots \\ c_{\rho 0} \end{bmatrix}$$

Remarque : *La matrice de covariance C est symétrique* ($c_{ik} = c_{ki}$).

La fonction de covariance (20.2) permet le calcul des coefficients de covariance : $\{c_{00}, c_{10}, c_{11}, \ldots, c_{pp}\}$. La procédure P2 décrit ce calcul :

P 2 — CALCUL DES COEFFICIENTS DE COVARIANCE .

```
pour i variant de 0 à p faire :
début
    pour k variant de i à p faire :
    début
        c(i,k) = 0.
        pour n variant de p à N-1 faire :
        début
            c(i,k) = c(i,k) + s(n-i) * s(n-k)
        fin
    fin
fin
```

```
comment : coefficients symétriques ;
pour i variant de 1 à p-1 faire :
début
    pour k variant de i+1 à p faire:
    début
        c(k,i) = c(i,k)
    fin
fin
```

II.1.4.3 / L'erreur minimale de prédiction .

A partir de l'erreur quadratique totale E définie en (10),et de sa minimisation au sens des moindres carrés (12), il est possible de déterminer l'erreur quadratique minimale de prédiction :E_p

$$(21) \qquad E_p = \sum_n s_n^2 - \sum_{i=1}^{p} a_i \cdot \sum_n s_n \cdot s_{n-i}$$

En exprimant E_p en fonction de R_k (19.2) , on aura :

$$(21.1) \qquad E_p = R_o - \sum_{i=1}^{p} a_i \cdot R_i$$

De même,en fonction de c_{ik} (20.2),E_p s'exprimera par :

$$(21.2) \qquad E_p = c_{oo} - \sum_{i=1}^{p} a_i \cdot c_{oi}$$

II.1.4.4 / L'algorithme de Durbin .

Cet algorithme récursif permet de calculer à partir des coefficients d'autocorrélation R_k , les coefficients a_k du filtre. La méthode est dite d' " *autocorrélation rapide* " . Elle nécessite encore moins de temps de calcul et de place mémoire que la résolution par la méthode de la " *racine carrée* " .

L'algorithme est le suivant :

$E_o = R_o$

pour i variant de 1 à p faire :

début

$$k_i = - \left(R_i - \sum_{j=1}^{i-1} a_j^{(i-1)} R_{i-j} \right) / E_{i-1}$$

$$a_i^{(i)} = k_i$$

$$a_j^{(i)} = a_j^{(i-1)} + k_i \cdot a_{i-j}^{(i-1)} \qquad 1 \leqslant j \leqslant i-1$$

$$E_i = (1 - k_i^2) \; E_{i-1}$$

fin

p : *nombre de coefficients du filtre.*

$\{a_i\}$; $1 \leqslant i \leqslant p$: *coefficients du filtre .*

E_i : *erreur de prédiction à l'ordre "i" .*

k_i : coefficients de corrélation partielle *ou coefficients de réflexion .*

$\{R_k\}$; $0 \leqslant k \leqslant p$: *coefficients d'autocorrélation .*

II.1.4.5 / Les coefficients de réflexion .

Les coefficients "k_i",qui apparaissent dans l'algorithme de Durbin comme résultats intermédiaires , sont appelés les "*coefficients de corrélation partielle*" ou "*coefficients de réflexion*" .

Dans les applications de la prédiction linéaire à l'analyse du signal vocal,ces coefficients sont très importants. En effet,si le conduit vocal est modélisé par un tube acoustique ayant p sections (voir I.2 et figure 1),les "k_i",i variant de 1 à p,représentent les coefficients de réflexion entre deux sections consécutives :

$$k_i = \frac{A_{i+1} - A_i}{A_{i+1} + A_i} \quad ; \quad i \text{ variant de 1 à p-1} \qquad (22.1)$$

A_{i+1} : aire de la section "i+1" du conduit vocal

A_i : aire de la section "i" du conduit vocal

Il est possible d'exprimer,à partir de (22.1),les rapports d'aires de deux sections consécutives en fonction des "k_i" :

$$\frac{A_{i+1}}{A_i} = \frac{1+ k_i}{1- k_i} \qquad (22.2)$$

Une condition nécessaire et suffisante pour que le filtre modèle soit stable est que k_i,pour i variant de 1 à p ,soit compris entre -1 et +1 :

$$|k_i| < 1 \quad ; \quad 1 \leqslant i \leqslant p \qquad (22.3)$$

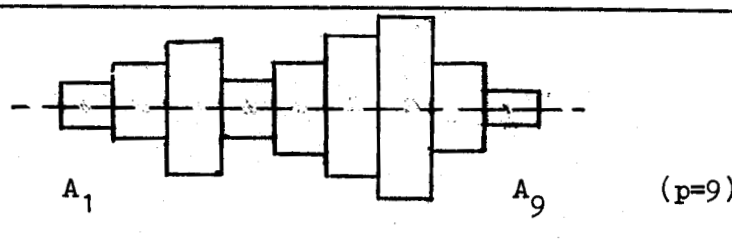


figure 1. *Modèle du conduit vocal* .

II.1.5 / Calcul du GAIN .

Le gain G du filtre modèle : $H(z) = \dfrac{G}{1 - \sum_{i=1}^{p} a_i z^{-i}}$ est obtenu par la conservation de l'énergie totale entre le signal et la réponse impulsionnelle H(z). Le gain doit satisfaire :

(23) $\boxed{G^2 = E_p}$

Le carré du gain est égal à l'erreur quadratique totale minimale de prédiction E_p ,définie en III.4.3 par les équations (21),(21.1) et (21.2).

(23.1) $\boxed{G^2 = R_0 - \sum_{i=1}^{p} a_i R_i}$ *Méthode d'autocorrélation*

(23.2) $\boxed{G^2 = c_{oo} - \sum_{i=1}^{p} a_i c_{oi}}$ *Méthode de covariance*

II.1.6 / Avantages/Inconvénients/Variantes des différentes méthodes

II.1.6.1 / Méthode d'autocorrélation .

Si du point de vue théorique, la stabilité du filtre tout-pôle H(z) est garantie dans la méthode d'autocorrélation, en fait, ce n'est pas le cas si les calculs sont faits sur ordinateur,en virgule fixe,la précision étant alors limitée.

De plus,le passage du signal par une fenêtre réduit la résolution spectrale ,mais est nécessaire,puisque le signal n'est pas stationnaire sur un temps long.

Les coefficients de corrélation partielle "k_i" (coeffs. de réflexion) ont permis de définir une nouvelle approche de la prédiction linéaire.
La formulation d'ITAKURA & SAITO (1971) permet en effet de calculer les "k_i" sans passer par les coefficients "a_i" du filtre. Un coefficient "k_i" représente alors le coefficient de corrélation entre l'erreur de prédiction précédente et l'erreur de prédiction suivante :

$$(24) \qquad k_i = -\frac{\sum_{n=1}^{N+i} X_{i-1}(n) \cdot Y_{i-1}(n)}{\left[\sum_{n=1}^{N+i} (X_{i-1}(n))^2 \cdot \sum_{n=1}^{N+i} (Y_{i-1}(n))^2\right]^{1/2}}$$

~ pour i variant de 1 à p

avec :
- $Y_o(n) = X_o(n-1)$
- N = nombre d'échantillons de parole pour une analyse.
- p = nombre de coefficients de réflexion.

Cette structure de calcul des "k_i" est représentée sur la figure 2 .
La stabilité du filtre peut être assurée par cette méthode,sans avoir besoin d'une fenêtre,et avec peu de sensibilité aux calculs en virgule fixe (cf. Samuelson & al. 1977). Cependant,l'inconvénient majeur de cette structure en"treillis " pour le calcul des coefficients est le nombre quatre fois plus important de calculs par rapport à la méthode classique d'autocorrélation.

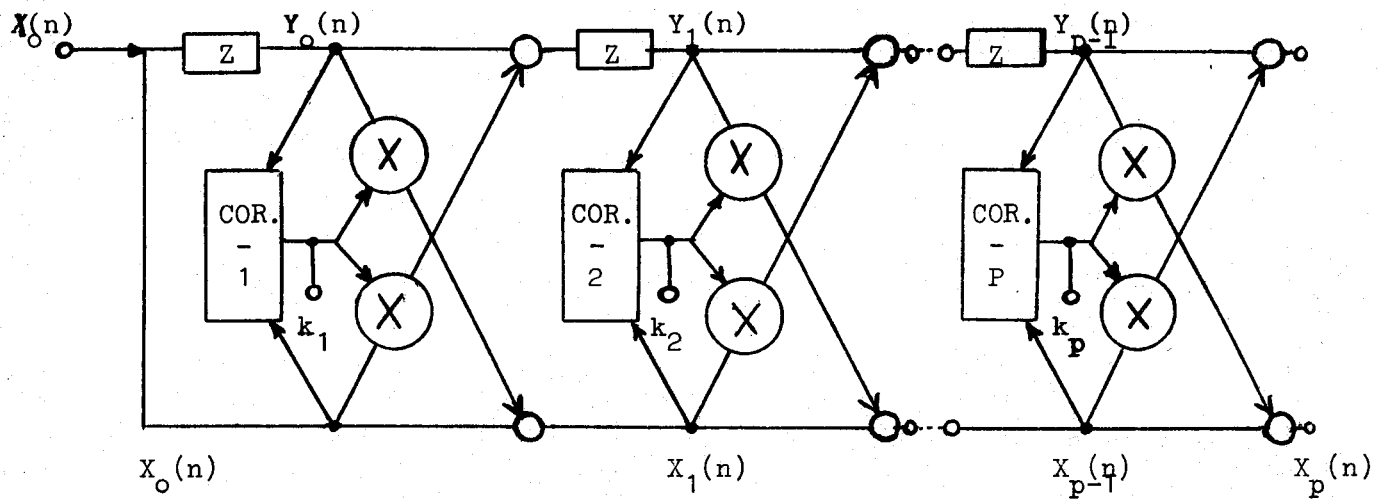


figure 2. STRUCTURE EN TREILLIS POUR LE CALCUL DES "k_i"

Remarque : *Bien que la formulation d'Itakura et Saïto ait été développée pour la méthode d'autocorrélation, elle peut également s'appliquer à la méthode de covariance.*

LE ROUX (1976) définit un nouvel algorithme pour la détermination des coefficients de corrélation partielle, plus rapide que la méthode classique, et plus stable. Bien que dérivé de celui de Durbin, l'algorithme de Le Roux évite le calcul des coefficients "a_i" du filtre, par contre, il détermine des "coefficients d'intercorrélation " .

II.1.6.2 / Méthode de covariance .

La méthode de covariance a l'avantage de ne pas nécessiter de fenêtre, par contre elle n'assure pas la stabilité du filtre, même dans les calculs en virgule flottante.

II.1.6.3 / Nouvelles méthodes / Comparaisons.

De nouvelles méthodes, utilisant la structure de filtre en treillis, et assurant la stabilité du filtre tout-pôle, ont été présentées par MAKHOUL (1977). La formulation d'Itakura et Saïto représente d'ailleurs un cas particulier à ces méthodes. Les coefficients de réflexion sont calculés directement à partir de la covariance du signal d'entrée.

Le tableau I résume le nombre d'opérations (multiplications et divisions) nécessaires aux principales méthodes de prédiction linéaire .

NOMBRE DE DIVISIONS ET DE MULTIPLICATIONS	AUTOCORRELATION	COVARIANCE	ITAKURA/SAITO
METHODES CLASSIQUES	$pN + p^2$	$pN+\frac{1}{6}p^3+\frac{3}{2}p^2$	5 p.N
NOUVELLES METHODES EN TREILLIS	$pN+\frac{1}{6}p^3+\frac{3}{2}p^2$	$pN+\frac{1}{2}p^3+2p^2$	5 p.N

TABLEAU I . *(d'après Makhoul 1977)*

p est le nombre de coefficients du filtre.
N est le nombre d'échantillons de parole analysés.

Par exemple, pour *p = 10* et *N = 160* (analyse sur 20 millisecondes & échantillonnage à 8 kHz) , le nombre de mutiplications et de divisions nécessaire à une analyse par prédiction linéaire est indiqué dans le tableau II .

méthodes	autocorrélation	covariance	Itakura/Saïto
classiques	~1700	~1900	~8000
nouvelles	~1900	~2300	~8000

TABLEAU II .

Le tableau III résume les propriétés des différentes méthodes de prédiction linéaire.

PROPRIETE	*METHODES DE PREDICTION LINEAIRE*			
	AUTOCOR.	*COVAR.*	*ITAKURA /SAITO*	*NOUVELLE METHODE*
FENETRE	*nécessaire*	*pas de fenêtre*	*non nécessaire*	*pas de fenêtre*
STABILITE	*théorique garantie*	*non garantie*	*peut être garantie*	*peut être garantie*
STABILITE AVEC CALCULS EN VIRGULE FIXE	*non garantie*	*non garantie*	*peut être garantie*	*peut être garantie*
EFFICACITE DES CALCULS	*bonne*	*bonne*	*mauvaise (calculs nombreux)*	*bonne*
QUANTIFICATION DES COEFFS."Ki"EN COURS DE RECURSION	*non possible*	*non possible*	*possible*	*possible*
NOMBRE D'ECHANTILLONS ANALYSES	N	PEUVENT POUR UNE	ETRE REDUITS A ~0.7 N MEME RESOLUTION	

TABLEAU III . *Comparaison des différentes méthodes* .
(d'après Makhoul 1977)

II.2. SCHEMA GENERAL D'UNE ANALYSE

L'étude théorique de la "prédiction linéaire" a permis de dégager les principaux algorithmes de résolution. Il est maintenant nécessaire de décrire les divers modules permettant une analyse complète du signal vocal par la méthode de prédiction linéaire.

Les principales étapes d'une analyse en temps réel sont les suivantes :

- discrétisation du signal vocal,
- prétraitement du signal,
- traitement du signal,
- détermination de la fréquence fondamentale Fo,
- codage des paramètres obtenus en vue de leur transmission ou d'un traitement ultérieur.

Le schéma général d'une analyse est donné figure 1.

Nous allons étudier plus en détail chacun des modules représentés.

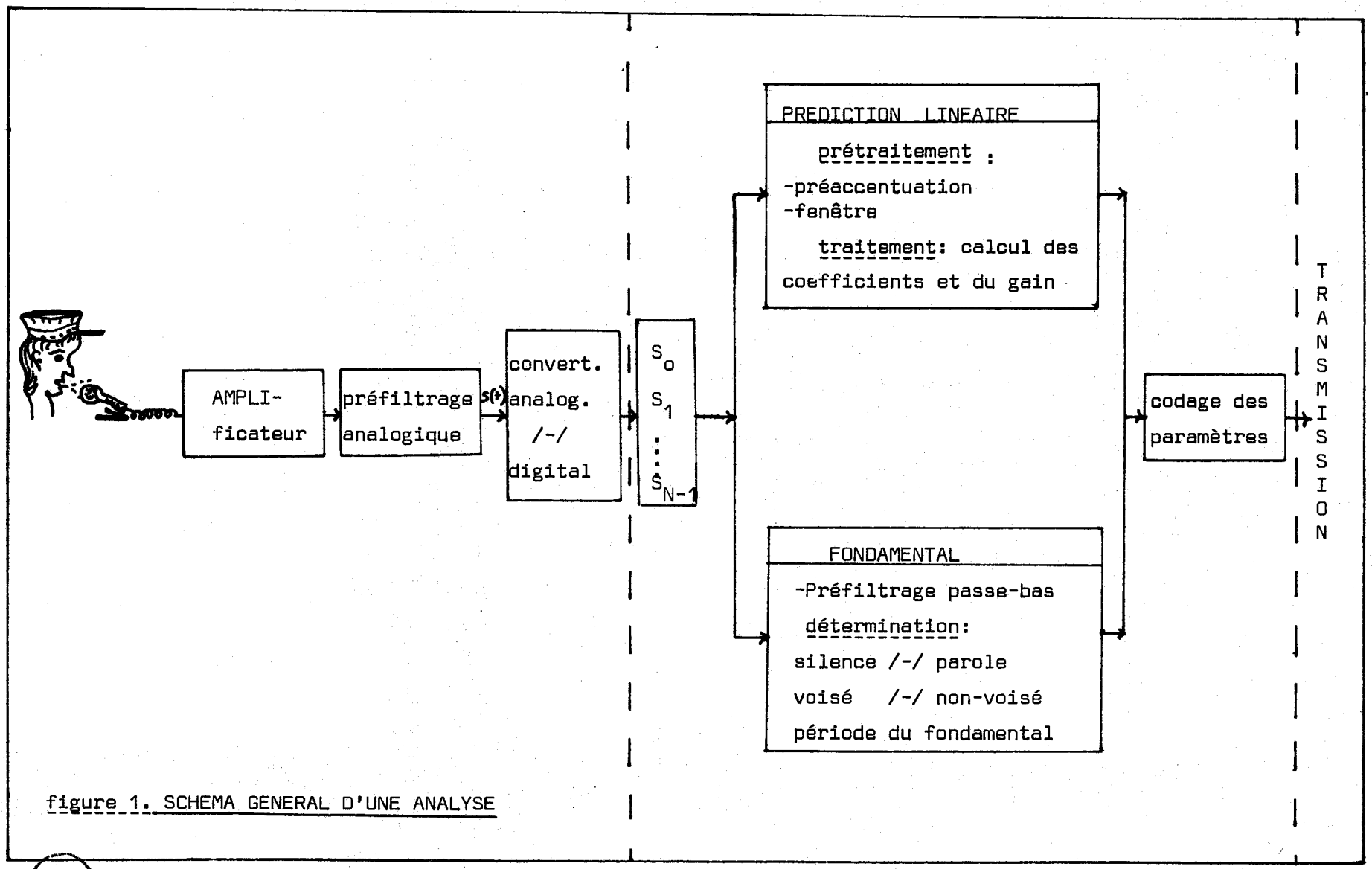


figure 1. SCHEMA GENERAL D'UNE ANALYSE

II.2.1 / Entrée de l'analyseur

Le signal analogique capté par le microphone est d'abord amplifié. Il est ensuite préfiltré et enfin échantillonné dans un convertisseur analogique/numérique.

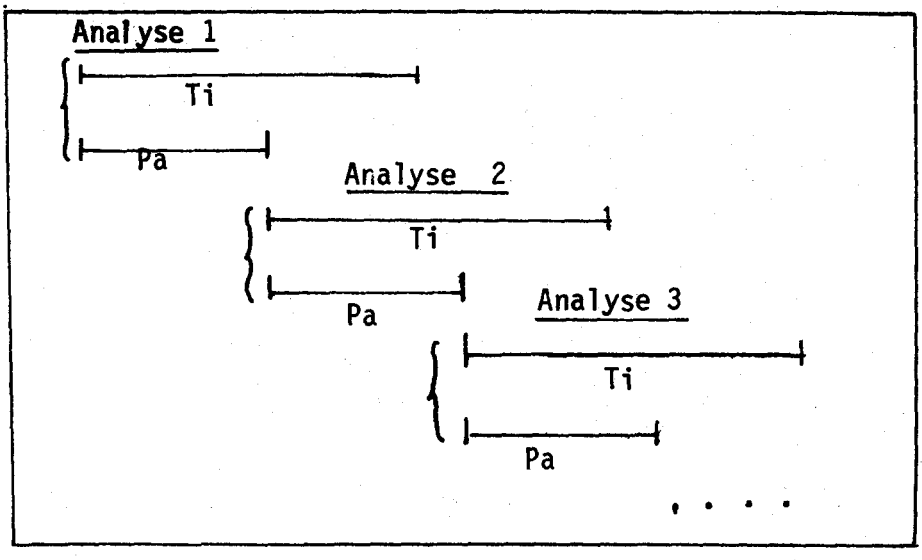
Le préfiltrage permet d'éviter les phénomènes de repliement du spectre . Si Fe est la fréquence d'échantillonnage, un filtre passe-bas affaiblissant les fréquences au-delà de $F_e/2$ sera nécessaire.

Pour la parole, les fréquences d'échantillonnage courantes vont de 8 à 15 kHz. Le convertisseur analogique/numérique dont on se sert discrétise le signal à la fréquence désirée et fournit des échantillons numérisés sur 12 bits. La valeur d'un échantillon peut donc varier entre - 2047 et + 2047.

La partie qui nous intéresse pour l'analyse se situe après la numérisation du signal. L'INTERVALLE D'ANALYSE, de durée Ti et la PERIODE D'ECHANTILLONNAGE Te (Te = 1/Fe) déterminent le nombre d'échantillons à analyser.

$$N = Ti / Te$$

La cadence d'analyse, dont la durée Pa est appelée "PERIODE D'ADAPTATION", est différente de l'intervalle d'analyse. Une nouvelle analyse est faite, toutes les Pa secondes (Pa est de l'ordre de 13 millisecondes), sur un bloc de N échantillons de parole correspondant à un intervalle de durée Ti (Ti peut varier de 20 à 30 millisecondes).



II.2.2 / Le prétraitement du signal

II.2.2.1. Le filtre de préaccentuation

Si l'on désire mieux approcher l'enveloppe spectrale dans les hautes fréquences, il est nécessaire d'appliquer un filtre de préaccentuation aux N échantillons de parole (So, S1,... Sn-1) avant l'analyse. On observe, en effet, que la pente du spectre d'un son voisé décroît d'environ - 6 dB par octave. Ceci est dû aux effets de la source d'excitation du conduit vocal (- 12 dB par octave) et du rayonnement aux lèvres (+ 6 dB par octave). Pour compenser ces effets et rendre le spectre plus plat, il convient d'appliquer au signal un filtre de la forme :

$$P(z) = 1 - m \cdot z^{-1}$$

(cf. Markel and Gray 1976)

Pour les sons non-voisés, ce filtre aura pour effet d'amplifier les hautes fréquences présentes, ce qui ne présente aucun inconvénient majeur.

La valeur exacte de "m" est fonction de l'autocorrélation du signal.

Soit R_k la fonction d'autocorrélation (cf. II.1.4 équation (19.2))

$$R_k = \sum_{n=0}^{N-1-|k|} S_n \cdot S_{n+|k|}$$

avec k positif ou nul

$$m = R_1 / R_0 = \sum_{n=0}^{N-2} S_n \cdot S_{n+1} \Big/ \sum_{n=0}^{N-1} S_n^2$$

La valeur de m est proche de 1 pour un son voisé, et pour un son non voisé, elle tend vers zéro .

Si m est calculé à chaque analyse, il faut transmettre sa valeur parmi les paramètres de sortie du système. Aussi, sur le plan pratique a-t-on choisi pour simplifier une préaccentuation conditionnelle avec un facteur de

préaccentuation fixe.

La valeur exacte de m est calculée à chaque analyse :

- si m est supérieur ou égal à 0.6, alors
  b = facteur de préaccentuation fixe.

- si m est inférieur à 0.6, alors
  b = 0

Il n'y aura préaccentuation qu'au delà d'un seuil (0.6). Le facteur "b" de préaccentuation sera dans ce cas proche de l'unité (b compris entre 0.90 et 0.95).

Cette méthode permet de ne transmettre qu'un seul bit en sortie (1 : préaccentuation ; o : pas de préaccentuation). Il permet en outre de ne pas préaccentuer un signal correspondant à un son non-voisé.

Application de la préaccentuation au signal

$$\begin{cases} s'_o = s_o \\ s'_n = s_n - (s_{n-1} \cdot b) \end{cases}$$

pour n variant de 1 à N - 1.

Sur les figures 2 (a) et 2 (b), on peut voir l'influence de la préaccentuation sur l'enveloppe spectrale d'un son voisé.

II.2.2.2. La fenêtre

Une fenêtre d'analyse implicite est déjà appliquée, puisqu'on ne traite que N échantillons à la fois. La méthode d'autocorrélation (cf. II.1) nécessite une fenêtre d'analyse. Si aucune fenêtre explicite n'est appliquée, des discontinuités vont apparaître entre les valeurs extrêmes de l'intervalle d'analyse et les valeurs nulles à l'extérieur de l'intervalle. Une distorsion spectrale en résultera. Plusieurs types de fenêtres ont été étudiées pour leur application à la prédiction linéaire (cf. Markel 1971) . La plus utilisée est la fenêtre de HAMMING,car elle est simple et suffisante.

Les figures 3 (a), 3 (b) et 3 (c) montrent l'effet de la fenêtre sur le signal.

Application de la fenêtre

$$S'n = Sn \cdot Fn$$

pour n variant de 0 à N - 1

Si c'est la fenêtre de Hamming qui est utilisée, on aura :

$$Fn = 0.54 - 0.46 \cos(2 \pi n / N)$$

Les figures 4 (a) et 4 (b) montrent l'effet de la préaccentuation et de la fenêtre de Hamming sur le signal correspondant à la prononciation de la lettre /a/.

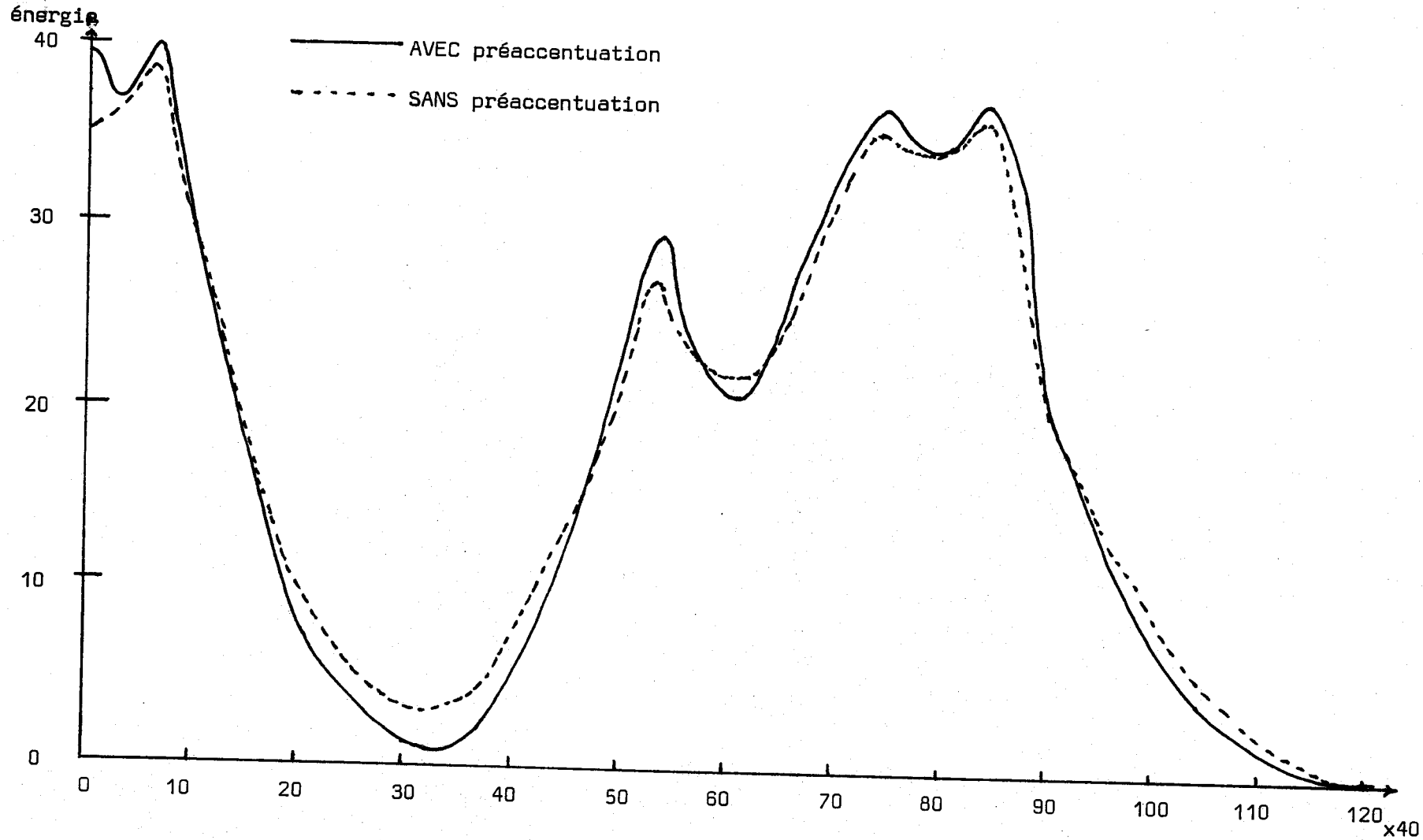


figure 2 (a) . Enveloppe spectrale d'un /i/ .

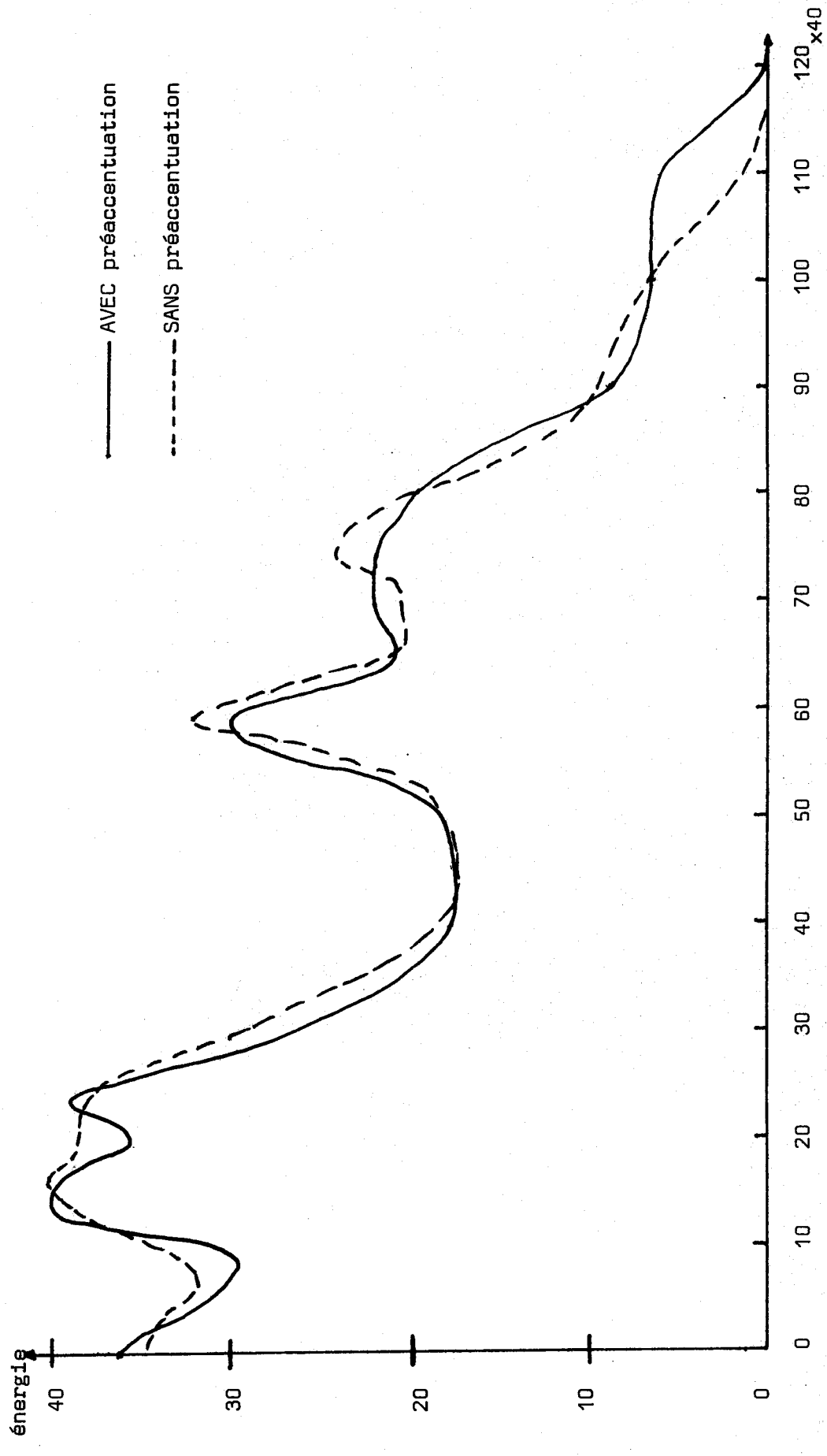


figure 2 (b) . Enveloppe spectrale d'un /e/ .

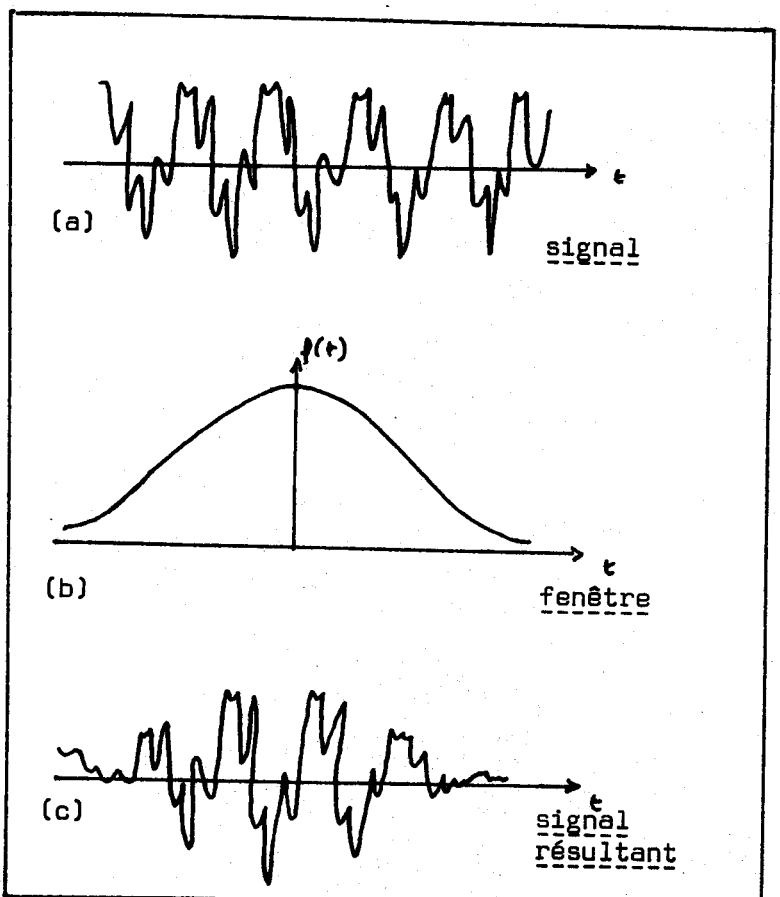


figure "3" . EFFET DE LA FENETRE DE HAMMING

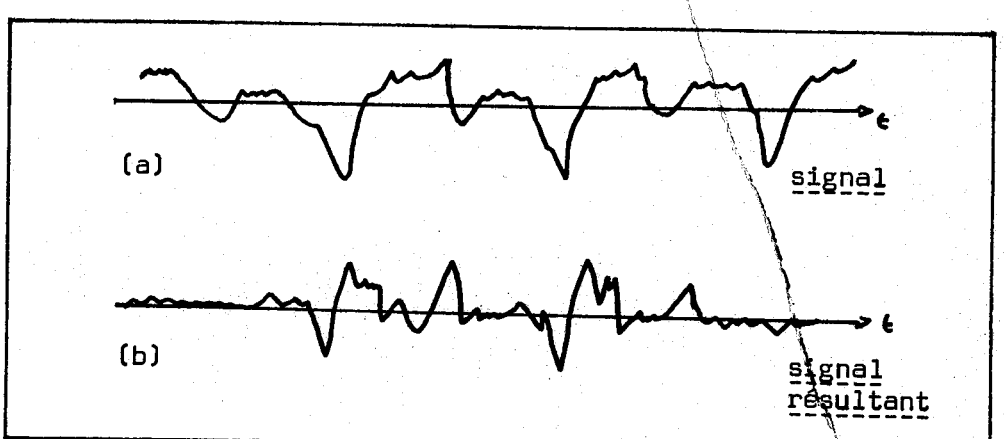


figure "4" . Voyelle /a/ . Préaccentuation + Fenêtre .

II.2.3 / L' Analyse

II.2.3.1. Calcul de la fonction de transfert du conduit vocal

Le signal ayant été prétraité, on lui applique maintenant un des algorithmes de prédiction linéaire décrits dans II.1.

Les paramètres de sortie choisis pour la transmission sont les coefficients de réflexion (K1, K2, ... Kp), de préférence aux coefficients du filtre prédicteur (a1, a2, ... ap). Ceci pour plusieurs raisons :

1) les "ai" peuvent présenter un filtre instable sous l'effet de la quantification , alors que les "Ki" (pour i variant de 1 à p) permettent d'assurer la stabilité du filtre (les "Ki" ont des valeurs comprises entre - 1 et 1), en dépit de la troncature des valeurs .

2) les "Ki" sont les plus adaptés au codage pour des transmissions à faibles débits. Cela résulte du 1), de la faible dispersion de leurs valeurs (seuls les premiers coefficients demandent un codage précis), de la possibilité de les coder de manière efficace .

3) à partir des "Ki", on peut retrouver les "ai" assez facilement.

4) l'algorithme, de prédiction d'Itakura et Saïto, permet d'obtenir directement les "Ki".

II.2.3.2. Le calcul du gain

Le gain du système peut être calculé en même temps que les "Ki". Nous avons vu (II.1.5.) que le carré du gain était égal à l'erreur quadratique minimale de prédiction Ep.

$$G^2 = Ep$$

II.2.4 / L'extraction de la fréquence fondamentale

La fréquence fondamentale (Fo) correspond à la fréquence de vibration des cordes vocales lors de la prononciation de son voisé, comme les voyelles. C'est un paramètre important pour l'étude du signal vocal. Plus les cordes vocales vibrent rapidement, plus la fréquence fondamentale est élevée (cas des voix féminines).

Quatre phases principales sont nécessaires pour déterminer Fo :

- préfiltrage du signal vocal,
- détermination silence/parole,
- décision son voisé ou non-voisé,
- dans le cas d'un son voisé : détermination de Po (Po : 1/Fo : période du fondamental).

Le préfiltrage du signal vocal : un filtre passe-bas, coupant les fréquences supérieures à 900 ou 1000 Hz, permet d'éliminer les pointes non significatives du signal (la fréquence fondamentale est située dans la bande 50- 500 Hz). Un filtre numérique d'ordre 2 (type Butterworth) est suffisant pour réaliser ce préfiltrage.

Détermination silence/parole : Le calcul de l'énergie du signal filtré permet de déterminer si aucun son n'a été prononcé. L'énergie est donnée par la formule suivante :

$$E^2 = \frac{\sum_{i=0}^{N-1} s_i^2}{N}$$

où E est la valeur efficace du signal,
$(So, \ldots S_{N-1})$ les échantillons de parole.
N est le nombre d'échantillons .

Décision voisé/non-voisé : Une première décision peut être prise en calculant le nombre de passages par zéro du signal. Un nombre élevé de passages par zéro dénote que le son prononcé est selon toute vraisemblance non voisé (une consonne non voisée présente en effet des "turbulences" quasi-aléatoires, alors qu'un son voisé se caractérise par une certaine périodicité).

La seconde décision sera faite parallèlement à la détection de la période du fondamental, grâce à un seuil de voisement.

Détermination de la période Po du fondamental : De nombreuses méthodes ont été mises au point pour extraire Po (NOLL 1967, ATAL et HANAUER 1971, MARKEL 1973, GOLD et RABINER 1969, ROSS et al 1974, MILLER 1976).

Nous nous sommes intéressés plus particulièrement au détecteur de Fondamental décrit par DUBNOWSKI, SCHAFER et RABINER en 1976. Le schéma fonctionnel du détecteur est décrit figure 5. L'avantage de la méthode est sa relative simplicité et son indépendance totale du calcul des coefficients du filtre. Les deux traitements (extraction du fondamental et calcul des "Ki") peuvent donc se dérouler en parallèle, ce qui n'est pas négligeable si l'on vise une application en temps réel.

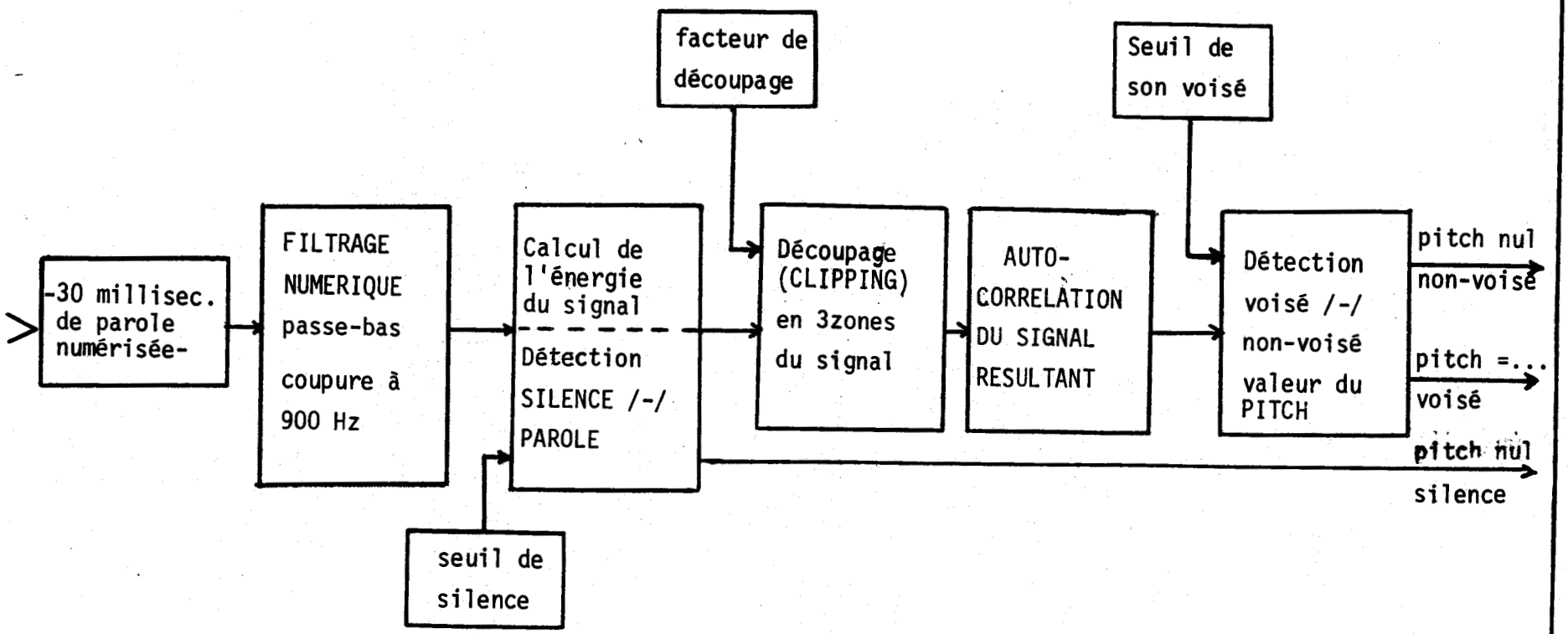


figure 5. DÉTECTION DE LA PERIODE DU FONDAMENTAL . (PITCH)

II.2.5 / Le codage des paramètres

Quatre ensembles de paramètres doivent être codés en vue de leur transmission.

Ce sont :

1) les "p" coefficients de réflexion (K1, K2, ... Kp)
2) le facteur de gain G
3) les indications relatives au fondamental : silence/parole, voisé/non voisé, période Po
4) l'indicateur de préaccentuation.

Le codage des coefficients de réflexion :

Un codage par les logarithmes des rapports d'aire du conduit vocal a été choisi (cf. VISWANATHAN et MAKHOUL 1975).

Le paramètre à transmettre correspondant à un Ki sera :

$$Kod_i = ((\log_2 Bi - MIN) / (MAX-MIN))* 2^{nbits}$$

où $Bi = A_i/A_{i+1} = (1 + Ki) / (1- Ki)$

= rapport d'aire entre 2 sections i et i + 1 du conduit vocal.

MIN = limite inférieure des variations de Log_2 Bi

MAX = limite supérieure des variations de log_2 Bi

nbits = nombre de bits alloués au codage d'un coefficient.

Kod_i sera toujours positif et inférieur ou égal à $2^{nbits} - 1$. La valeur absolue de Ki doit être inférieure à 1.

Il est à noter que le nombre de bits alloués au codage peut être différent pour chaque coefficient.

Le codage du gain G

Le gain est codé logarithmiquement. Il faut évidemment définir une limite inférieure non nulle pour G.

La période du fondamental

Po peut être codé logarithmiquement, mais on préfère souvent transmettre directement sa valeur, la valeur "zéro" pouvant indiquer un silence ou un son non-voisé.

La préaccentuation

Un bit suffit pour indiquer s'il y a eu ou non préaccentuation du signal.

II.2.6 / La sortie de l'analyseur

Les paramètres codés ou non seront transmis soit au synthétiseur dans le cas d'un vocodeur à prédiction linéaire, soit à la chaine de reconnaissance de parole pour les traitements ultérieurs.

Les paramètres codés occuperont de 60 à 70 bits par intervalle d'analyse .

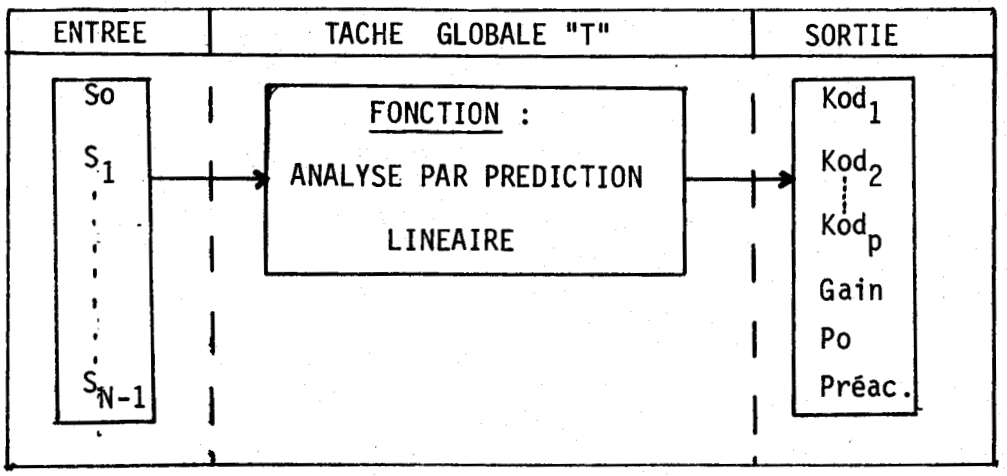
CHAPITRE TROIS

SPECIFICATIONS DU SYSTEME A REALISER

CHAPITRE TROIS : SPECIFICATIONS DU SYSTEME A REALISER

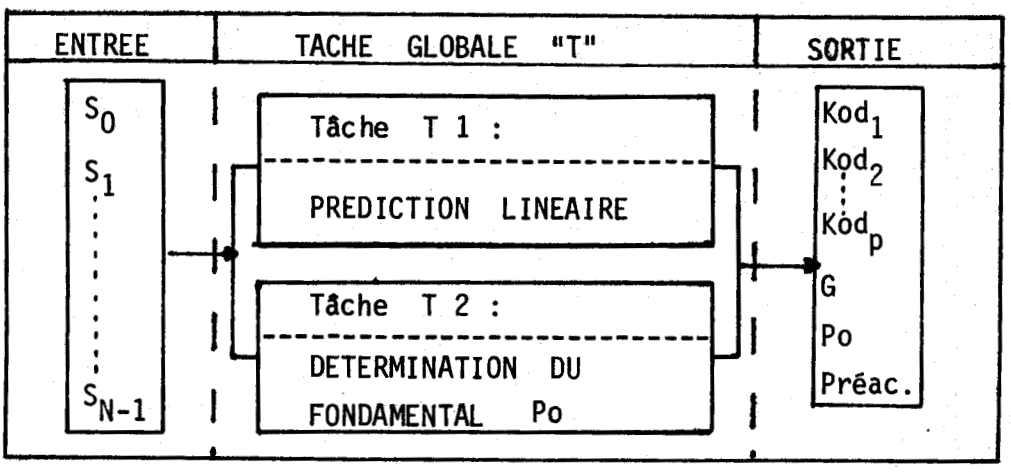
III.1. DECOMPOSITIONS FONCTIONNELLES DE L'APPLICATION

D'un point de vue global, le processus qui traite la tâche à effectuer se caractérise par un flux d'entrée (un vecteur à N composantes) , et un flux de sortie (les paramètres de sortie occupant "q" bits) :



Une première décomposition permet d'éclater la tâche globale en deux sous-tâches indépendantes.

Ces deux sous-tâches peuvent se dérouler en parallèle,de manière indépendante et simultanée,sous réserve de se synchroniser en début et fin de traitement.



Chacune des sous-tâches peut elle-même être décomposée de manière plus fine:

= La tâche T 1 : "traitement par prédiction linéaire du signal."

Cette tâche comprend :

- t_1 : Le prétraitement du signal
  - tt_1 : La préaccentuation
  - tt_2 : L'application de la fenêtre
- t_2 : Le traitement du signal
  - tt_3 : Le calcul des coefficients de réflexion
  - tt_4 : Le calcul du gain
- t_3 : Le codage des paramètres

= La tâche T 2 : " Détermination de la période Po du fondamental."

Cette tâche comprend :

- t'_1 : Le préfiltrage numérique du signal
- t'_2 : La détermination du fondamental
- t'_3 : Le codage éventuel de Po

Cette nouvelle décomposition met en évidence une structure de type "macro-pipe-line" :

. Le flux d'information est unidirectionnel

. Il se caractérise par le passage de données d'un élément du "pipe-line" au suivant.

. Chaque élément du pipeline correspond à une tâche spécialisée.

Les tâches spécialisées,dans T 1 et T 2 , communiquent entre elles suivant le schéma classique du " *Producteur / Consommateur* " :

Processus i	Processus i+1
producteur	consommateur
1) *Produire l'information*	1) *Prélever l'information*
2) *Déposer l'information*	2) *Traiter l'information*
3) *Retour en 1)*	3) *Retour en 1)*

Le processus "i" est le producteur ; le processus "i+1" est le consommateur .

Remarque :-*Des "sémaphores de synchronisation" (DIJKSTRA 1965) sont nécessaires pour coordonner l'exécution des tâches :*

Le producteur ne peut déposer l'information à transmettre que si le consommateur a prélevé l'information précédente. DE même,le consommateur ne peut prélever l'information que si le producteur l'a déposée.

Il est possible d'affiner encore les tâches spécialisées,mais les décompositions deviennent alors trop dépendantes de l'algorithme choisi.

En résumé,l'application se caractérise par des décompositions de deux types:

- Décomposition de type *Parallèle* :

Si F est la fonction " Analyse par prédiction linéaire ", E,l'entrée du système,et S la sortie, nous avons :

$$S = F(E) = \{F1(E) \text{ et } F2(E)\}$$

-Décomposition de type *Série* :

$$\begin{cases} F1(E) = f3(f2(f1(E))) \\ F2(E) = f'3(f'2(f'1(E))) \end{cases}$$

Le flux d'information est unidirectionnel et se compose de données de type vectoriel.

Les figures 1 et 2 représentent de deux manières différentes le découpage de l'application.

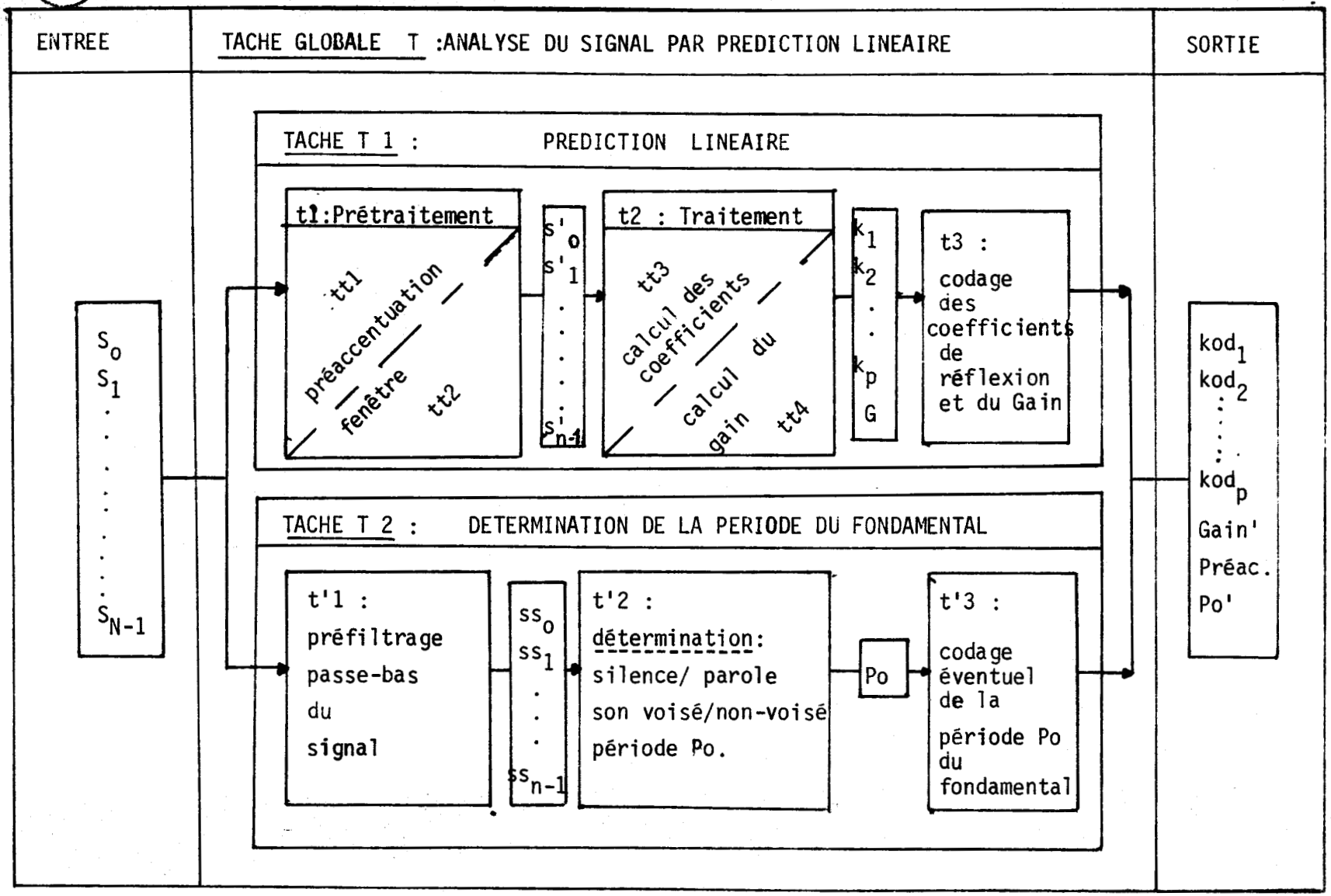


Figure 1 . DECOMPOSITIONS SUCCESSIVES ET PARALLELES DE L'APPLICATION

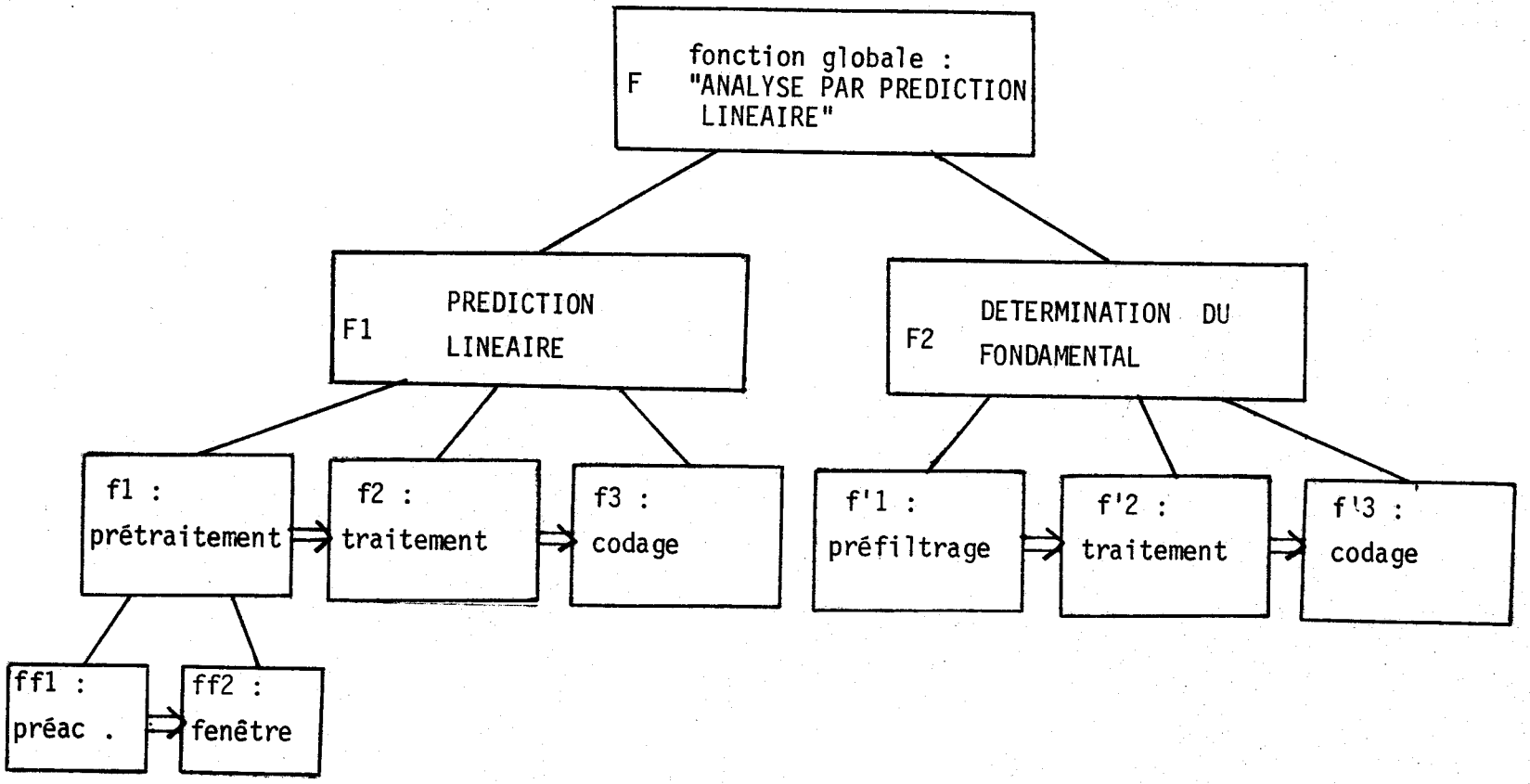


Figure 2 . DECOUPAGE FONCTIONNEL DE L'APPLICATION .

⇒ liens "série"

III.2. Choix et contraintes dans l'application

Le choix des conditions d'analyse et les contraintes qui en découlent sont très importants pour définir le comportement dynamique de l'application.

= Choix de la fréquence d'échantillonnage F_e :

Une fréquence d'échantillonnage élevée permet une représentation proche du signal vocal continu. Cependant,nous avons vu (II.2.1) que le nombre d'échantillons à analyser dépendait de la période d'échantillonnage "T_e" ($T_e = 1 / F_e$) : Plus T_e est court,plus le nombre d'échantillons à analyser est élevé,ce qui a pour conséquence d'augmenter le nombre de calculs à effectuer.

Le choix d'une fréquence d'échantillonnage de l'ordre de 8 à 12 kHz est un moyen terme satisfaisant. Les caractéristiques principales du signal vocal sont apparentes,même pour les voix de femmes ou d'enfants,et l'échantillonnage reste faible. Les fréquences d'échantillonnage extrêmes pour la parole vont de 3 à 32 kHz environ.

= Choix de l'intervalle d'analyse T_i :

Le choix de l'intervalle doit tenir compte de deux facteurs :

- la durée de l'intervalle.
- la position de l'intervalle.

La durée de l'intervalle doit être telle que le mouvement du conduit vocal soit négligeable. Ceci afin d'éviter un lissage trop important de l'évolution des caractéristiques du conduit vocal . Cette durée sera de l'ordre de 10 à 30 millisecondes.

La position de l'intervalle détermine le type d'analyse :

a) l'analyse en synchronisme avec la période Po du fondamental. L'intervalle se situe alors entre deux impulsions d'excitation du conduit vocal. Cependant cette méthode d'analyse pose des problèmes :

. La détermination du fondamental doit précéder l'analyse.
. Les intervalles d'analyse ont des durées variables.

Pour une analyse en temps réel,nous préfèrerons donc le second type d'analyse:

b) L'analyse asynchrone :

La durée de l'intervalle d'analyse et sa position ne dépendent plus des impulsions d'excitation du conduit vocal. L'analyse est faite périodiquement sur un intervalle de durée fixe.

= Choix du nombre "p" de coefficients (ordre du filtre prédicteur)

Il s'agit de déterminer le nombre minimum "p" de coefficients nécessaires pour modéliser les caractéristiques significatives du conduit vocal.

Atal et Hanauer (1971) ont montré le rapport entre la longueur du conduit vocal (des cordes vocales aux lèvres) et l'ordre du filtre prédicteur . La"mémoire" du filtre modèle doit être égale à deux fois le temps requis pour la propagation du son depuis les cordes vocales jusqu'aux lèvres :

$$T = 2 \cdot L / C$$

où : | L est la longueur du conduit vocal (17,5 cm environ)
| C est la vitesse de propagation du son (353 m / s)

Ce qui donne pour "T " environ 1 milliseconde .

Pour une période d'échantillonnage T_e de 100 microsecondes (F_e= 10 kHz), le nombre "p" de coefficients sera :

$$p = \frac{T}{T_e} = \frac{10^{-3}}{10^{-4}} = 10$$

Cependant,il faut tenir compte de l'effet de la source d'excitation et du rayonnement aux lèvres :

. Si aucune préaccentuation n'est faite pour corriger ces effets, deux pôles supplémentaires sont nécessaires au filtre:

| p = 12 pour F_e = 10 kHz
| p = 10 pour F_e = 8 kHz

. Si une préaccentuation conditionnelle fixe est appliquée au signal,un seul coefficient supplémentaire est suffisant:

$$\begin{cases} p = 11 \text{ pour } F_e = 10 \text{ kHz} \\ p = 9 \text{ pour } F_e = 8 \text{ kHz} \end{cases}$$

= Choix de la 'période d'adaptation' P_a :

La forme du conduit vocal variant continuellement dans le temps, il est nécessaire de recalculer les "p" coefficients périodiquement, afin de les remettre à jour.

Une nouvelle analyse sera donc faîte toutes les 13 millisecondes environ.Dans cet intervalle de temps,la forme du conduit vocal peut être considérée stationnaire.

= Choix de l'algorithme de prédiction :

On veut pouvoir traiter n'importe quel algorithme de prédiction en temps réel. Aussi va-t-on se baser sur celui qui demande le plus de calculs (Itakura et Saïto 1971) pour définir les contraintes .

= CONCLUSION : Les paramètres et contraintes de l'application.

1) Le convertisseur analogique / numérique fournit des échantillons de parole codés sur 12 bits (bit de signe compris) .

2) La fréquence d'échantillonnage sera de 8,10 ou 12 kHz .

3) L'intervalle d'analyse aura une durée de 20 à 30 millisecondes. Pendant cette durée,l'évolution du conduit vocal peut être considérée comme stationnaire.

4) Le nombre d'échantillons à analyser (N) se déduit de la période d'échantillonnage T_e et de la durée de l'intervalle d'analyse T_i:

;si T_e = 100 microsecondes (F_e = 10 kHz)
et T_i = 20 millisecondes,alors :

$$N = T_i / T_e = 200$$

;si T_e = 125 microsecondes (F_e = 8 kHz)
et T_i = 20 millisecondes , alors :

N = 160

5) La période d'adaptation P_a (cadence d'extraction des paramètres) est de 13 millisecondes.

6) Le nombre "p" de coefficients du filtre est fonction de la fréquence d'échantillonnage et de l'existence d'une préaccentuation .
p varie de 9 à 12 environ.

7) Le nombre de bits alloués au codage des sorties de l'analyseur est de 65 bits environ .

8) Le débit de l'analyseur : 65 bits toutes les 13 millisecondes =

5 kbits / seconde

III.3. Considérations générales sur la machine à réaliser

III.3.1 / Analyse d'activité du système

En conclusion des considérations précédentes, la machine doit traiter une analyse par prédiction linéaire en temps réel, c'est à dire en 13 millisecondes au maximum.

L'application, décomposée en sous-tâches successives ou parallèles, a été simulée afin de définir l'activité de chacun des modules.

Le tableau I indique, pour chacun des modules de l'application (exception faite de la détermination du fondamental), le nombre d'opérations à effectuer par analyse.

Ce tableau permet de faire les remarques suivantes :

a) Le nombre de multiplications est très élevé et est du même ordre que le nombre d'additions / soustractions.

Il est donc probable que, quelle que soit l'architecture envisagée, il sera nécessaire de traiter à la même vitesse les additions et les multiplications. Ceci n'est pas le cas des machines classiques où il faut environ quatre fois plus de temps pour traiter une multiplication.

b) Les calculs les plus nombreux sont faits dans le module qui extrait les coefficients de réflexion et le gain : Ce module représente un " goulot d'étranglement " dans la chaîne de traitement .

c) Une machine classique, de puissance moyenne , ne peut tenir le temps réel. Une machine de "grosse" puissance pourrait éventuellement convenir, mais le coût serait alors disproportionné par rapport aux performances nécessitées par l'application.

TABLEAU I

NOMBRE D'OPERATIONS PAR ANALYSE

(sans la détermination du fondamental)

conditions d'analyse :

- Algorithme de prédiction d'Itakura et Saïto
- Fréquence d'échantillonnage : 8 kHz .
- Intervalle d'analyse :20 millisecondes.
- Nombre d'échantillons :160
- Période d'adaptation :13 millisecondes.
- Nombre de coefficients : 9

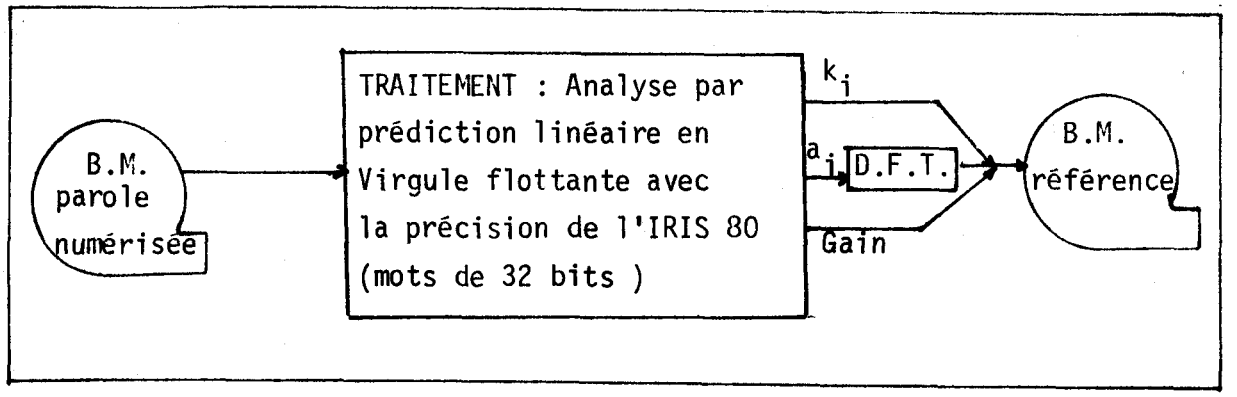
modules / nombre d'opérations	PRETRAITEMENT		TRAITEMENT	TOTAL	
	Préaccent.	Fenêtre	coeffs. + Gain	13 ms.	soit par SECONDE
ADDITIONS et SOUSTRACTIONS	480	0	≈ 7500	≈8000	≈ 615 000
MULTI-PLICATIONS	460	160	≈ 7400	≈8100	≈625 000
DIVISIONS	1	0	10	11	≈ 1 000
RACINES CARREES	0	0	19	19	≈2 000

III.3.2 / Comportement de l'application en virgule fixe.

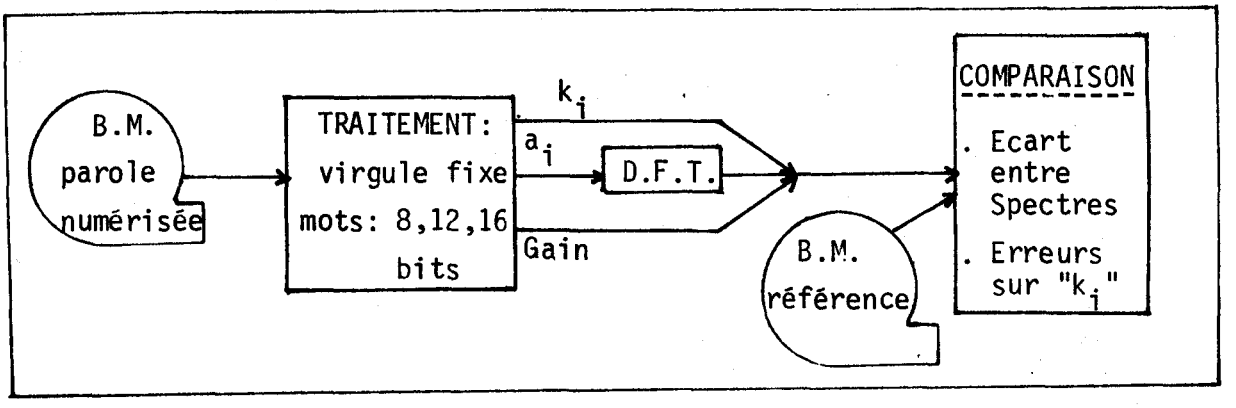
Dans l'optique d'une réalisation performante et la plus simple possible sans dégradation pour le système,nous avons étudié les erreurs introduites par l'utilisation de mots-machine de longueur finie. Ces erreurs affectent plus ou moins les résultats, et il est possible en particulier de quantifier ces erreurs en terme de déviations du spectre du signal vocal.

Une simulation complète de l'application a été faite,un paramètre définissant la longueur des mots-machine(Samuelson et al 1977).

Partant d'une référence (les calculs effectués en virgule flottante sur 32 bits), il s'agissait de la comparer aux résultats obtenus avec des longueurs de mots allant de 8 à 16 bits ,les calculs étant effectués en virgule fixe simulée (voir les deux schémas ci-dessous).



REFERENCE



COMPARAISON

Lors de cette simulation,il est tout de suite apparu que si l'on voulait réaliser des calculs en virgule fixe,il était nécessaire d'appliquer au signal un facteur d'échelle :

Ajustement du signal .

Pour obtenir un maximum de précision lors des calculs en virgule fixe, il est nécessaire que le signal soit amplifié s'il est faible, et atténué s'il est trop fort.

Les échantillons de parole nécessaires à une analyse seront donc ajustés par un facteur d'échelle 2^M,tel que :

$$2^{nbits-3} \leqslant |\hat{s}_n| * 2^M < 2^{nbits-2}$$

où: nbits = nombre de bits du calculateur virgule fixe simulé.

$|\hat{s}_n|$ = échantillon de parole d'amplitude maximale dans l'intervalle d'analyse {0 , N-1 }

Remarque : Cet ajustement du signal va se faire une première fois en début, et une seconde fois en fin de prétraitement,après la préaccentuation et l'application de la fenêtre.

La détermination et l'application du facteur d'échelle sont décrites dans la séquence suivante :

: Recherche du maximum en valeur absolue parmi les N échantillons de parole à analyser :

$$MAXI = |\hat{s}_n| = \max(|s_n|) \quad \text{pour n variant de 0 à N-1}$$

: Détermination de M pour que (1) soit vérifié .

: Ajustement des échantillons :

$$s_n = s_n * 2^M \quad \text{pour n variant de 0 à N-1}$$

Les résultats de l'étude sur les problèmes occasionnés par des longueurs de mots finies et des calculs effectués en virgule fixe ont été obtenus à partir d'un corpus de phrases prononcées par dix locuteurs différents.

Ces phrases ont été numérisées,puis analysées par la méthode de prédiction linéaire d'Itakura et Saïto.

Les tests effectués ont porté sur :

. L'erreur quadratique moyenne (EQM)

. La variance de l'erreur sur les coefficients de réflexion.

L'Erreur Quadratique Moyenne (cf. Markel & Gray 1974 b) a permis de mettre en évidence les écarts entre spectres à court terme du signal : Les spectres de référence ont été comparés aux spectres obtenus par des calculs de coefficients faits en virgule fixe.

L'erreur s'exprime par :

$$EQM = (1 / N_f) \cdot \sum_{j=1}^{N_f} E_j$$

où : N_f est le nombre d'analyses

$$E_j = (1 / NECHT) \cdot \sum_{k=1}^{NECHT} (BB)^2$$

où : NECHT = 256 = Nombre de points de la Transformée De Fourier Discrète (DFT),appliquée au signal.

$$BB = 10 \log_{10} \left[(|A_k|^2 / G^2) \cdot (\overline{G}^2 / |\overline{A_k}|^2) \right]$$

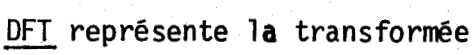
$(|A_k|^2 / G^2)$: valeur de référence virgule flottante

$(\overline{G}^2 / |\overline{A_k}|^2)$: valeur calculée virgule fixe

où : G est le gain de référence

$\overline{G}$ est le gain calculé en virgule fixe

$A_k = DFT(1, a_1, a_2, \ldots, a_p, 0, 0, \ldots, 0)$



où : DFT représente la transformée de Fourier Discrète

$a_1, \ldots, a_p$ sont les coefficients du filtre

p est le nombre de coefficients.

Remarques:

La DFT s'applique à un signal numérisé et permet de calculer les raies du spectre.

Les coefficients du filtre (les "a_i") permettent d'obtenir le spectre lissé du signal grâce à une DFT .

Les écarts entre spectres obtenus sont représentés figure 1 .

La simulation a été faite pour des mots de 8, 12 et 16 bits,avec et sans préaccentuation

La variance de l'erreur engendrée par la troncature des coefficients de réflexion "k_i" est donnée par la formule suivante :

$$VE(k_i) = \left[(1/N_f) \cdot \sum_{j=1}^{N_f} (\hat{k}_i - k_i)^2\right] - \left[(1/N_f) \cdot \sum_{j=1}^{N_f} (\hat{k}_i - k_i)\right]^2$$

où: N_f est le nombre d'analyses faites .

$\hat{k}_i$ représente la valeur du i-ème coefficient de réflexion calculé en virgule fixe à la j-ème analyse.

k_i représente la valeur du i-ème coefficient calculé avec la précision maximale à la j-ème analyse.

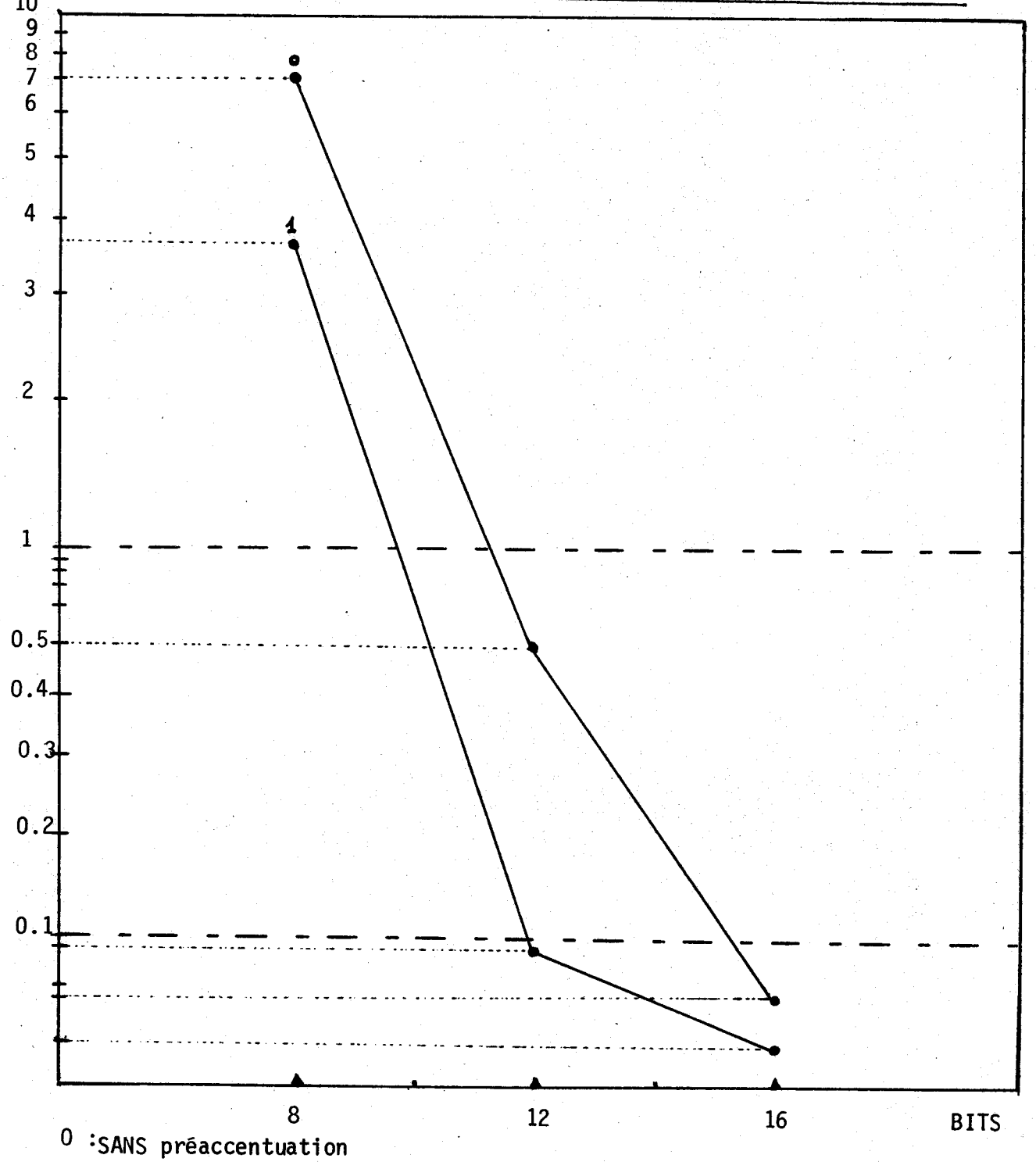
Cette formule est l'application directe de l'expression générale de la variance : $v(x) = \sum_i (f_i . x_i)^2 - (\sum_j f_i . x_j)^2$

Les figures 2 et 3 donnent la variance de l'erreur ,respectivement pour "k_1" et "k_2" , les deux coefficients les plus sensibles aux effets de troncatures.

Le nombre d'analyses N_f a été de plus de 1500 pour chaque configuration (8,12 et 16 bits).

En conclusion de cette étude,il ressort des figures 1,2 et 3 qu'une machine effectuant des calculs en virgule fixe, avec des mots-mémoire d'une longueur de 16 bits , offre une précision suffisante pour l'application. Par ailleurs,cette longueur de 16 bits représente une valeur normalisée couramment employée. La mise en oeuvre matérielle sera donc facilitée par ce choix.

figure 1 : Erreur Quadratique Moyenne en fonction du nombre de bits



0 :SANS préaccentuation

1: AVEC préaccentuation

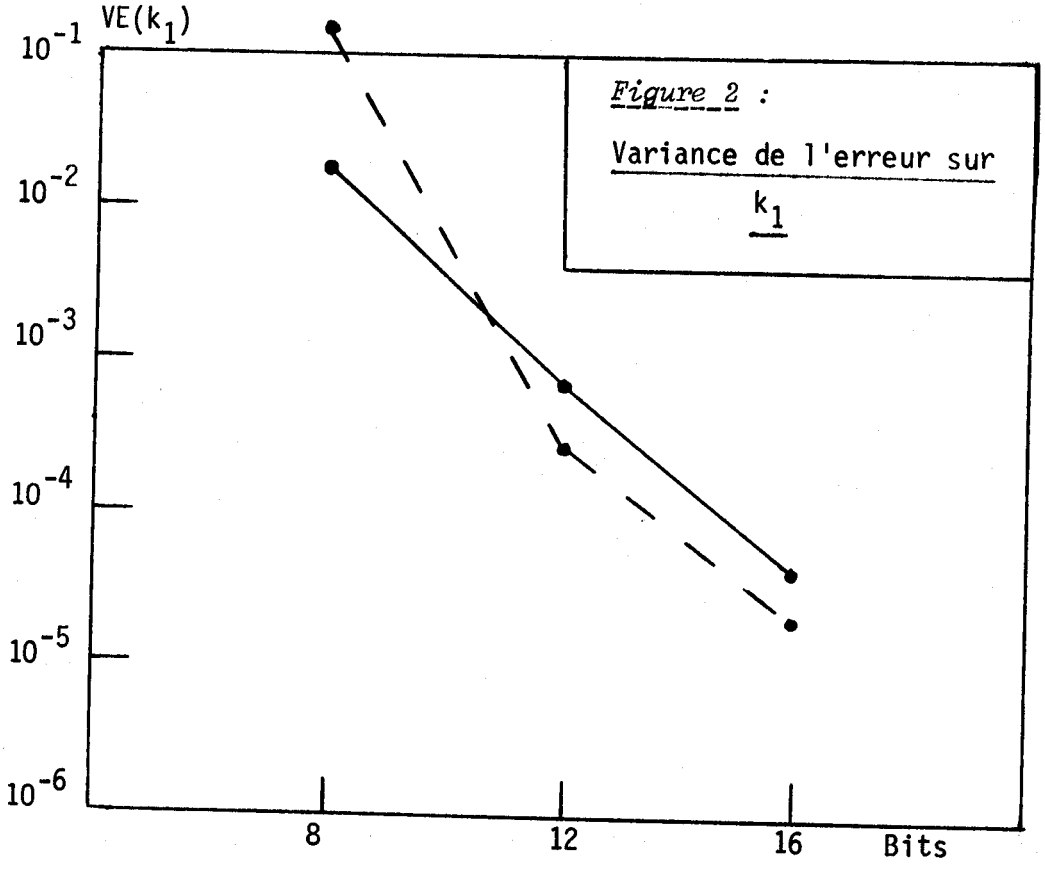


Figure 2 :
Variance de l'erreur sur k_1

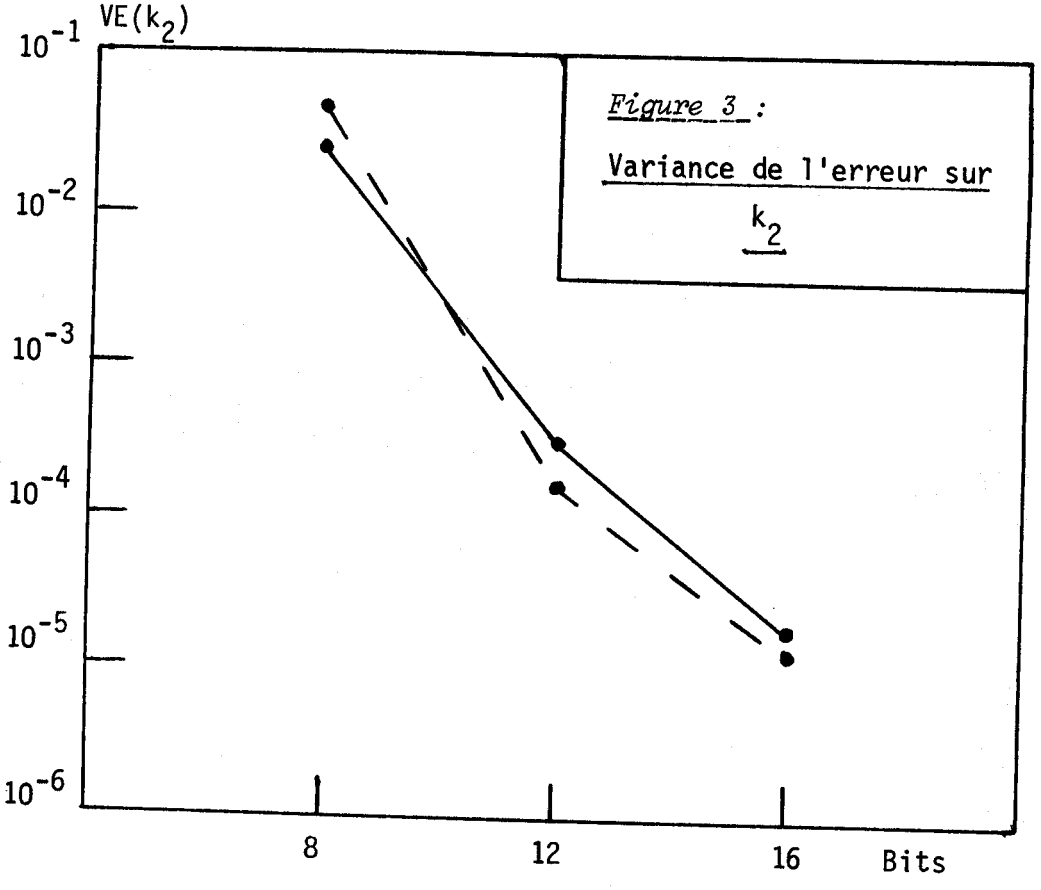


Figure 3 :
Variance de l'erreur sur k_2

— — AVEC préaccentuation
——— SANS préaccentuation

III.3.3 / Primitives vectorielles

L'observation des algorithmes de l'application,et l'enchainement des divers modules (cf. III.1) incitent à concevoir une machine adaptée au TRAITEMENT VECTORIEL .

Des primitives pour le traitement des vecteurs ont été dégagées. Elles restent suffisament générales pour pouvoir servir à d'autres applications où la manipulation de données vectorielles est nécessaire.

Les différents modules de l'application s'expriment alors par un grand nombre d'appels à ces primitives, ce qui simplifie leur écriture de manière notable. De plus,ces primitives sont nécessaires si l'on souhaite obtenir les performances recherchées.

Dans l'optique d'une réalisation spécialisée,il sera possible d'inclure ces primitives aux autres instructions.

Les pages suivantes présentent la liste des primitives vectorielles nécessaires à l'application. La fonction réalisée lors de l'appel à chacune de ces primitives est décrite sous forme d'algorithme.

LISTE DES PRIMITIVES VECTORIELLES

DEVEC — Décalage d'éléments d'un vecteur .

paramètres :

X	: Vecteur 1
N1	: nombre d'éléments de X
Y	: Vecteur 2
N2	: nombre d'éléments de Y
P	: Nombre de décalages.

fonction réalisée :

pour i *variant de* 1 *à* P *faire* :
    Y(i) = 0
pour j *variant de* P+1 *à* N1+P *faire* :
    Y(j) = X(j-P)

remarques :

. N2 *sup. ou égal à* N1+P
. P *positif*

VAVEC — Initialisation d'un vecteur à une valeur .

paramètres:

X	: Vecteur
N	: nombre d'éléments de X
Val	: Valeur d'initialisation

fonction réalisée:

pour i *variant de* 1 *à* N1 *faire* :
    X(i) = VAL

remarque :

Cas particulier : remise à zéro d'un vecteur
Val = 0

PROSC — Produit scalaire de deux vecteurs .

paramètres :

X	: Vecteur 1
N1	: Nombre d'éléments du vecteurX
Y	: Vecteur 2
N2	: nombre d'éléments de Y
PSC	: Résultat

fonction réalisée :

```
N = min (N1,N2)
PSC = 0
pour i variant de 1 à N faire :
    | PSC = PSC + X(i) * Y(i)
```

MAXM — Rang du terme de valeur maximale dans un vecteur .

paramètres:

X	: Vecteur
N	: nombre d'éléments de X
Rg	: Rang du maximum.

fonction réalisée :

```
Rg = 1
pour i variant de 2 à N faire :
    |Rg = si X(i) sup. à X(Rg) alors i
                              sinon Rg
```

MAXMA — Rang du terme maximal en valeur absolue dans un vecteur

paramètres : mêmes paramètres que"MAXM " :X,N,Rg

fonction réalisée :

```
Rg = 1
pour i variant de 2 à N faire :
    |Rg = si abs(X(i)) sup à abs(X(Rg))
         alors i ,sinon Rg
```

SOTAB — Somme de deux tableaux terme à terme .

paramètres :

X	: tableau 1
N1	: nombre d'éléments du tableau X
Y	: tableau 2
N2	: nombre d'éléments du tableau Y
Z	: tableau somme
N3	: nombre d'éléments de Z

fonction réalisée :

```
N = min(N1,N2)
pour i variant de 1 à N faire :
    Z(i) = X(i) + Y(i)
```

remarques :

. N3 *sup. ou égal* min(N1,N2)

. Pour passer un minimum de paramètres, et simplifier la procédure, N1 = N2 = N3

MUTAB — Produit terme à terme de deux tableaux .

paramètres :

X	: tableau 1
N1	: nombre d'éléments de X
Y	: tableau 2
N2	: nombre d'éléments de Y
Z	: tableau produit
N3	: nombre d'éléments de Z

fonction réalisée :

```
N = min (N1,N2)
pour i variant de 1 à N faire :
    Z(i) = X(i) * Y(i)
```

remarques :

. Mêmes remarques que pour la primitive "SOTAB"

P2VEC

Multiplication/Division des éléments d'un vecteur par des puissances de"2" . (Décalage de R bits des éléments)

paramètres :

X	: Vecteur
N	: nombre d'éléments du vecteur
R	: puissance de "2"

fonction réalisée :

pour i *variant de* 1 *à* N *faire* :

$X(i) = X(i) * 2^{R}$

remarque :

.Si R est négatif,les éléments du vecteur sont divisés par des puissances de "2" .

SCAVE

Produit d'un scalaire par un vecteur .

paramètres :

X	: Vecteur 1
N1	: nombre d'éléments de X
Y	: vecteur résultant
N2	: nombre d'éléments de Y
SCA	: Scalaire

fonction réalisée :

pour i *variant de* 1 *à*N1 *faire* :

$Y(i) = X(i) * SCA$

remarques :

. N2 *sup. ou égal* N1

. Cas général : N1 = N2

III.3.4 / Premières conclusions sur la machine à réaliser .

1) Les décompositions de l'application ont permis de mettre en évidence des traitements parallèles et série.

2) L'observation du volume de calculs à effectuer indique qu'une machine d'architecture classique et de taille moyenne ne pourra traiter l'application en temps réel. Un multiplieur rapide devra de toute manière être incorporé à l'unité arithmétique et logique.

3) Les études statistiques,portant sur l'influence d'une arithmétique tronquée sur les résultats d'analyse,ont permis de déduire qu'une machine effectuant des calculs en virgule fixe ,sur 16 bits ,possédait une précision suffisante.

4) L'observation des types de traitements à effectuer incitent à concevoir une machine orientée vers le calcul vectoriel. Des primitives ont été définies dans ce but. Les traitements étant répétitifs,le fait d'appeler des primitives spécialisées,fréquemment utilisées, permet d'améliorer les performances du système.

DEUXIEME PARTIE

L A R E A L I S A T I O N

CHAPITRE UN

DEFINITION DE L'ARCHITECTURE DE LA MACHINE

...

CHAPITRE UN - DEFINITION DE L'ARCHITECTURE DE LA MACHINE

I.1/ INVENTAIRE - GENERALITES

L'application est maintenant définie de manière précise. Les besoins et les contraintes pour réaliser cette application ont été dégagés. La contrainte la plus "critique" est le temps réel. Elle oblige à effectuer un volume très important de calculs en un temps très court.

Il reste à trouver ou concevoir une machine qui obéit à cette contrainte.

. Dans le premier cas, il s'agit de chercher, parmi les machines existantes, celles qui répondent aux besoins de l'application.

. Dans le second cas, il faut concevoir et réaliser une machine adaptée aux besoins de l'application.

Deux types de machines correspondent au premier cas :

- des machines spécialisées

Elles ont été conçues pour l'application ou une classe d'applications voisines. Nous allons décrire particulièrement certaines de ces machines, mises au point par le "Lincoln Laboratory" au M.I.T. D'autres machines de type voisin existent ou sont en cours de réalisation. Cependant, aucune ne possède la souplesse désirée (modification des paramètres, changement d'algorithme).

...

- des machines plus générales

L'application peut y être implantée. Ce sont en particulier les processeurs "tableaux" et les processeurs "pipe-line". Cependant, peu de ces machines sont disponibles ou exploitables de manière rationnelle pour l'application.

De plus, ce type de machines n'est pas entièrement satisfaisant, car il ne correspond pas à la tendance actuelle face à la conception de systèmes informatiques.

Concevoir et réaliser un système adapté aux besoins représente une évolution irréversible, surtout pour les applications "temps réel".

Trois types de réalisation sont possibles :

- la réalisation d'un système spécialisé, cablé, donc figé : une telle machine n'est pas intéressante dans notre cas. Nous désirons un système "paramétrable". De plus, les progrès constants de la technologie risquent de périmer rapidement la machine.

- la réalisation d'un système spécialisé, mais micro-programmé.

- la réalisation d'un système plus général, adapté à l'application, mais permettant de traiter d'autres applications du même type.

Ces deux derniers points sont les plus intéressants. Nous les développeront au moment du choix de l'architecture du système à réaliser.

...

I.1.1. Les systèmes spécialisés

Un certain nombre de machines spécialisées à été construit pour répondre au problème de traitement en temps réel du signal numérisé.

Plusieurs de ces machines ont été conçues pour les besoins de la communication parlée. Cependant, certaines sont adaptées au traitement du signal de manière plus générale (traitement d'images, de signaux radar...).

Les caractéristiques nécessaires à de telles machines sont :

- une grande vitesse,
- une facilité de programmation,
- une souplesse d'emploi,
- un environnement spécifique du traitement de signal en temps réel.

Nous allons nous intéresser plus particulièrement aux systèmes spécialisés qui permettent d'effectuer l'analyse du signal vocal par prédiction linéaire en temps réel.

Le "lincoln laboratory" au M.I.T. a réalisé 3 machines pour les codages de parole à faible et moyen débit :

- le FDP (Fast Digital Processor),
- le DVT (Digital Voice Terminal),
- le LPCM (Linear Predictive Coding Microprocessor).

...

LE "FDP" (cf. GOLD, LE BOW, MC HUGH, RADER 1971)

Mis au point vers 1970, le FDP représente un premier pas vers le traitement en temps réel du signal vocal. Cependant, son manque de flexibilité et de facilité de programmation a conduit les chercheurs du "lincoln laboratory" à concevoir le "Digital Voice Terminal".

LE "DVT" (cf. BLANKENSHIP, HOFSTETTER et AL 1975)

Le "Digital Voice Terminal" possède les caractéristiques suivantes :

- temps de cycle très rapide : 55 nanosecondes,
- facilité de programmation,
- machine peu encombrante,
- un système périphérique spécifique a été conçu autour de la machine pour permettre le traitement du signal vocal en temps réel (cf. fig.1)

Architecture du DVT

Le DVT est constitué essentiellement d'un processeur effectuant des calculs en virgule fixe, et travaillant sur 16 bits en complément à 2.

Il comporte plusieurs sous-ensemble (voir figure 2) :

- une mémoire de type "RAM" (Random Access Memory) qui est utilisée pour les données du programme et les constantes.
- une autre mémoire du type "RAM" pour les programmes.
- une mémoire de type "ROM" (Read only Memory) contenant les micro-codes de la machine.
- un ensemble de registres actifs : (A, X, P, B).
- une unité arithmétique et logique (ALU) qui comprend un multiplieur séparé.

...

- un système d'entrée/sortie pour communiquer avec l'extérieur.

La mémoire-programme possède sa propre commande de chargement de programme, permettant ainsi l'entrée des programmes (sous forme bin aire). Toutes les 55 nanosecondes, la mémoire-programme fournit une nouvelle instruction au registre d'instruction. Le décodage de l'instruction est réalisé dans la mémoire des micro-codes, permettant ainsi la génération des micro-commandes nécessaires au fonctionnement du système.

L'unité arithmétique et logique (cf. figure 3) comporte un multiplieur rapide qui permet d'effectuer un produit 16 bits x 16 bits, avec un résultat sur 32 bits en 220 nanosecondes (soit 4 cycles machine).

Le DVT utilise le "pipe-line par instruction" afin d'augmenter la rapidité du système. Un triple recouvrement (recherche, lecture, exécution) permet d'effectuer l'instruction en 1 seul cycle machine (55 nanosecondes) au lieu des 3 cycles normalement nécessaires (165 nanosecondes). Ce type de recouvrement (voir figure 4) oblige à prendre certaines précautions dans le séquencement des instructions (particulièrement dans le cas des instructions de saut).

Plusieurs algorithmes de codage de parole à faible et moyen débit ont été implantés dans le DVT, dont le "vocodeur" à prédiction linéaire (VLPC). L'algorithme de prédiction linéaire choisi est celui de Markel (cf. Figure 5), mais il a été adapté.

L'implantation du VLPC dans le Digital Voice Terminal a permis aux chercheurs du "Lincoln Laboratory" de faciliter le développement d'un matériel plus spécialisé : le LPCM, plus simple et meilleur marché que le DVT.

...

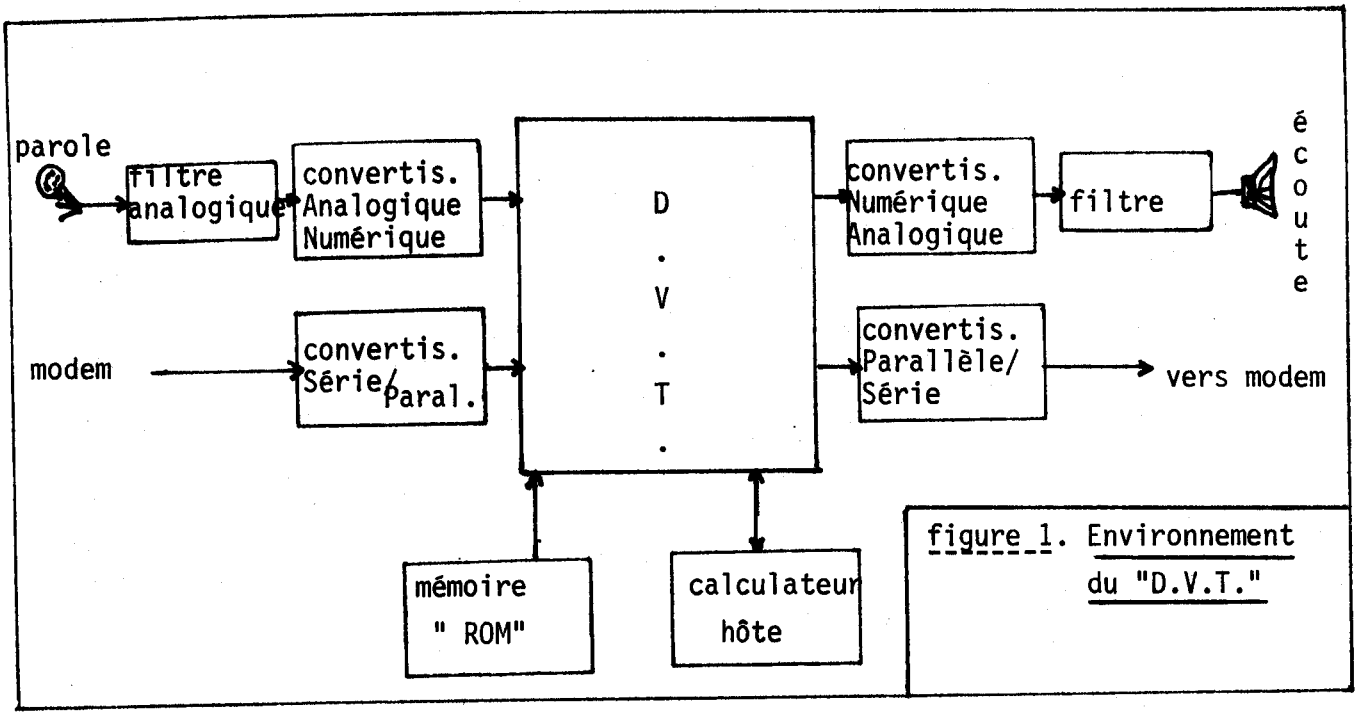


figure 1. Environnement du "D.V.T."

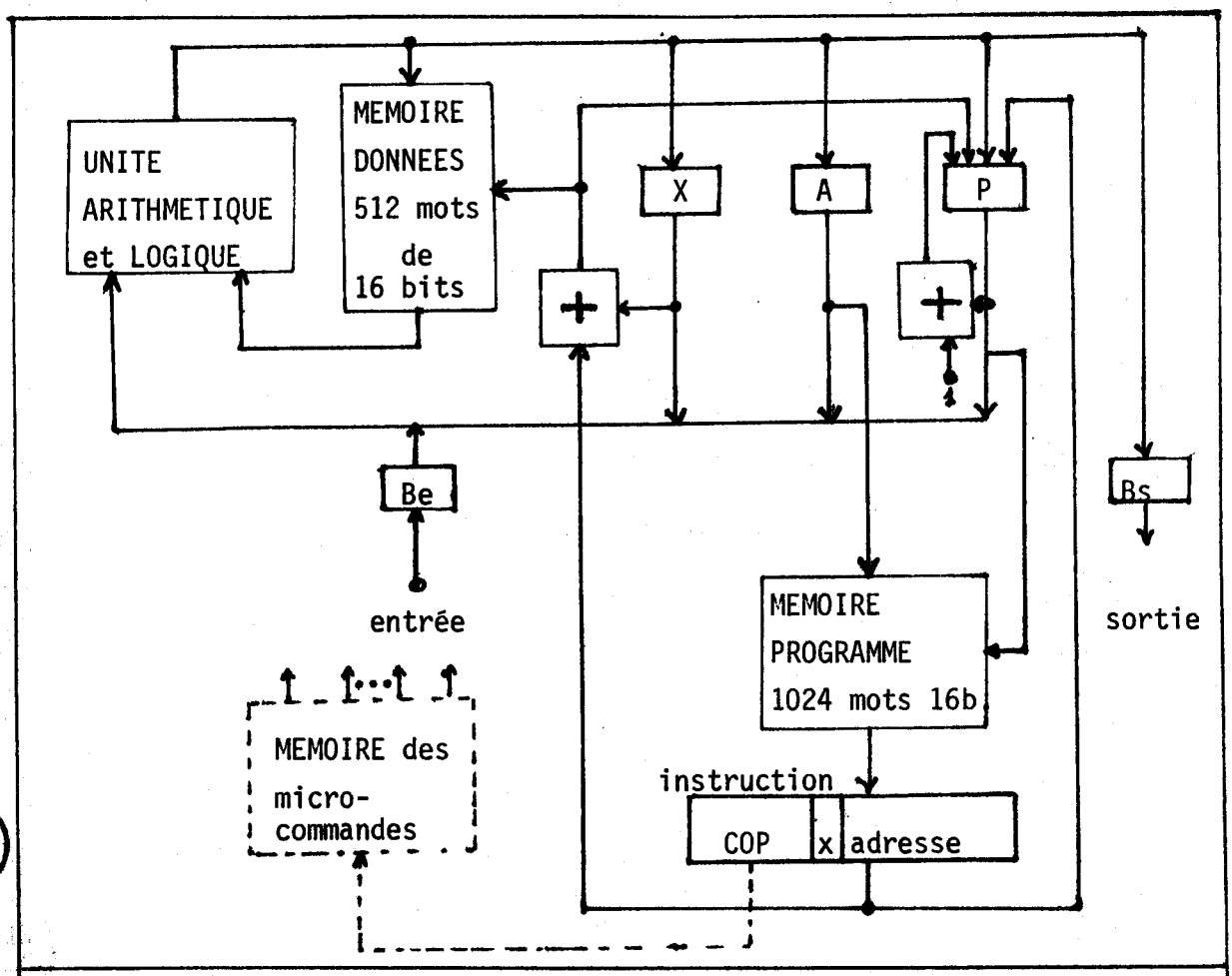


figure 2 . Structure simplifiée du "Digital Voice Terminal"

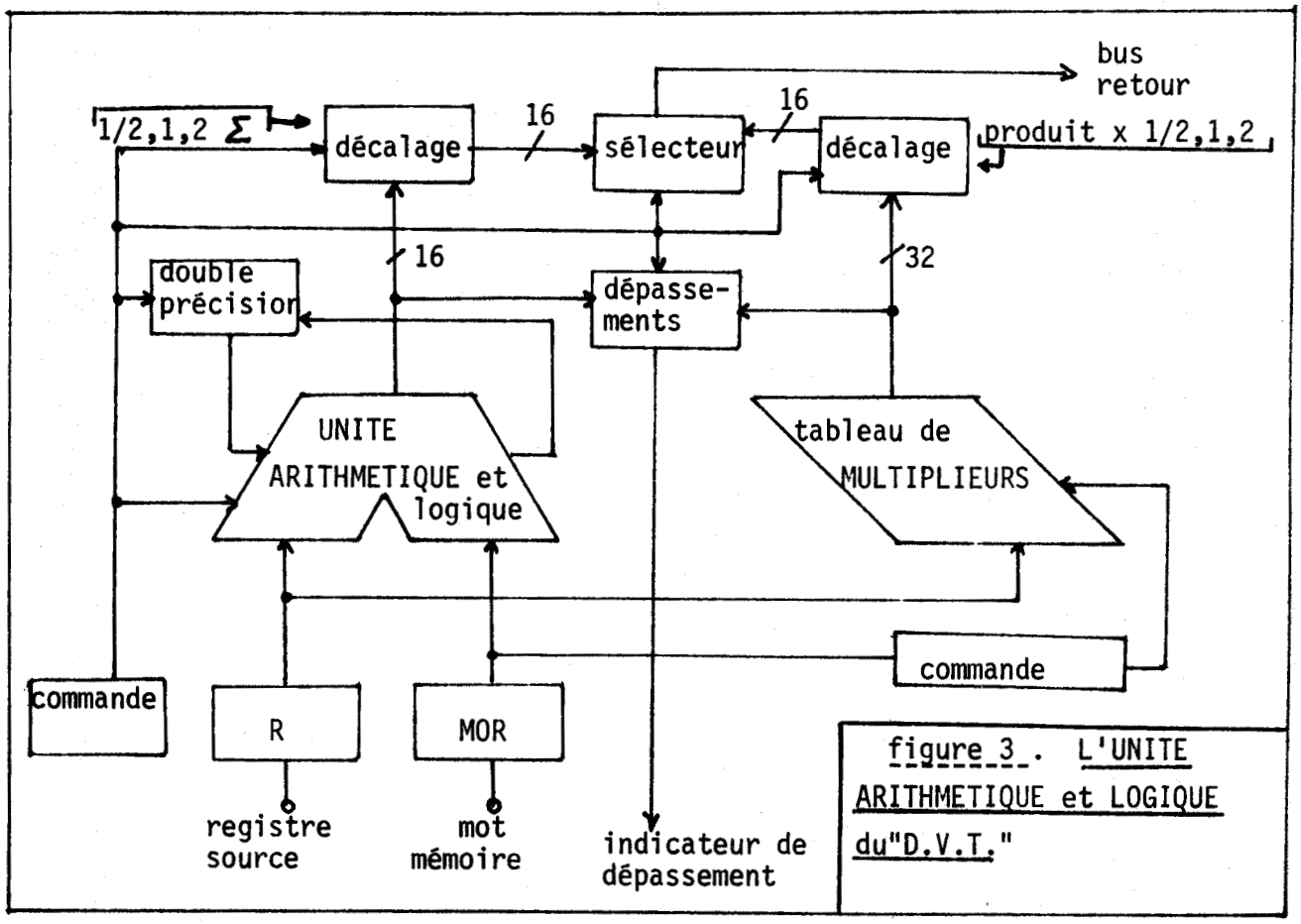


figure 3 . L'UNITE ARITHMETIQUE et LOGIQUE du"D.V.T."

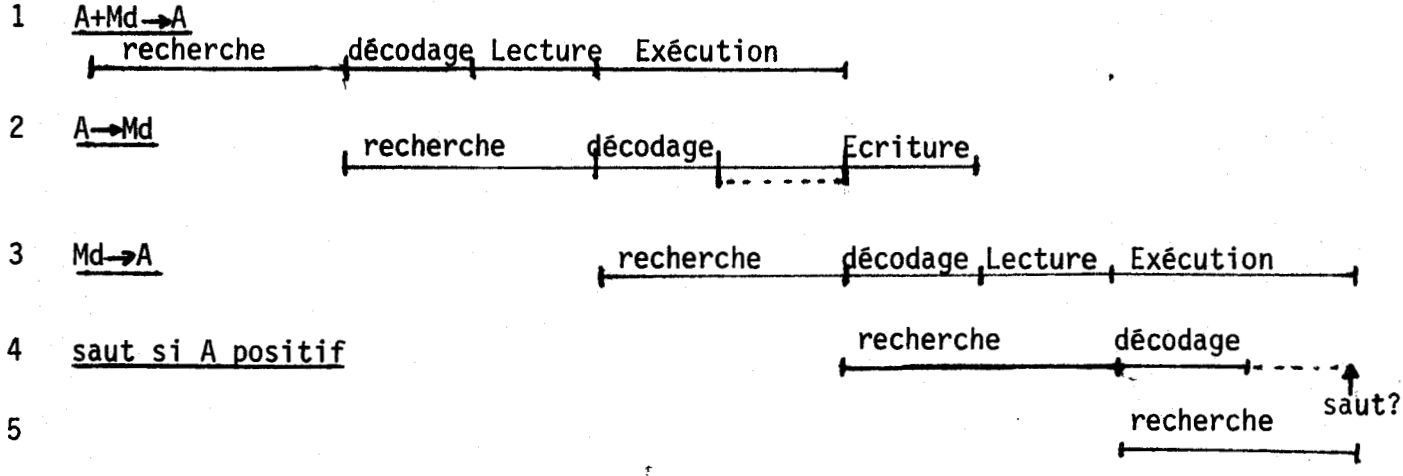


figure 4 . Comportement temporel du "pipe-line" .

55 nanosecondes

A :registre accumulateur

Md:adresse en mémoire donnée

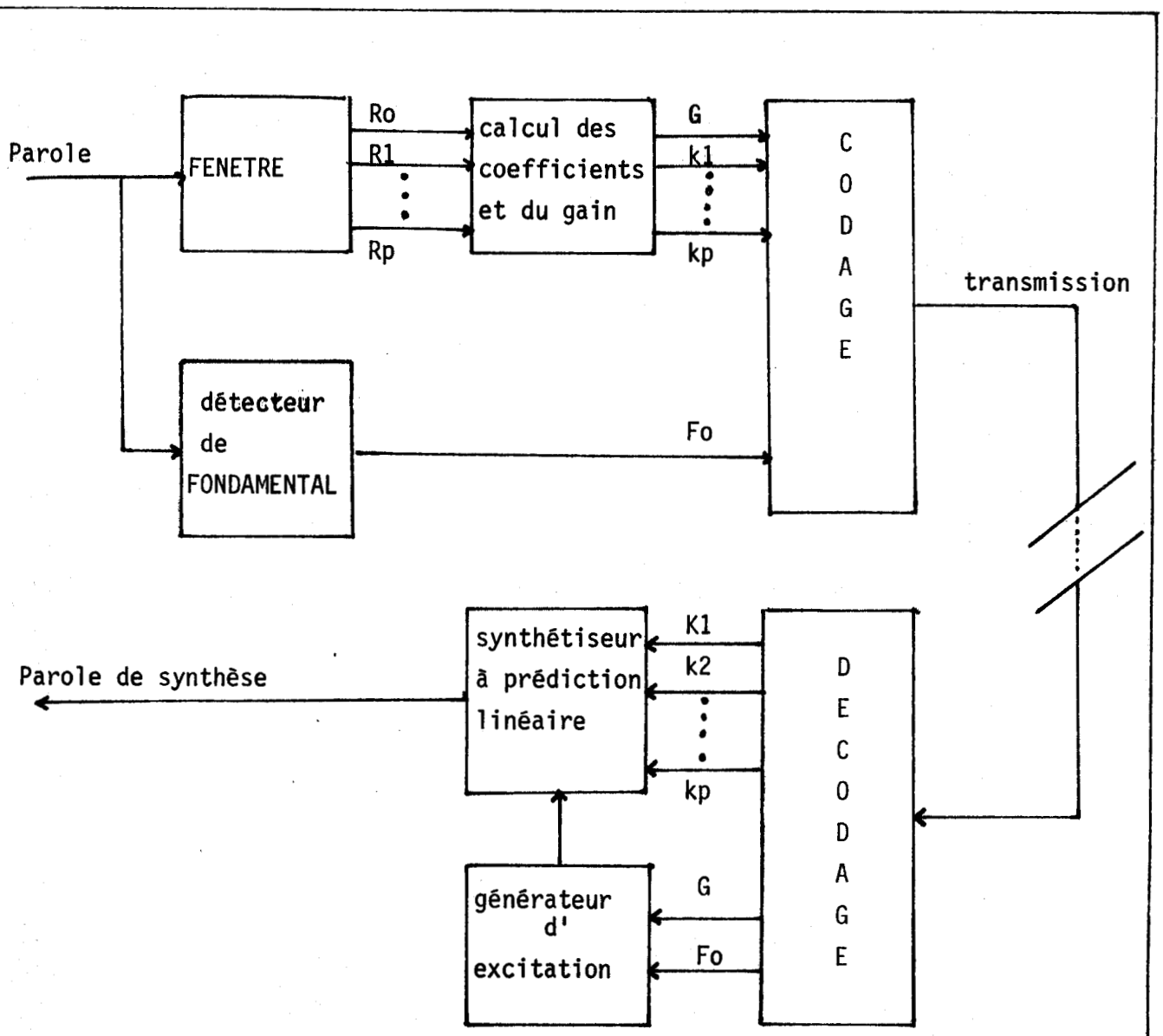


figure 5 . ALGORITHME DU VOCODER A PREDICTION LINEAIRE

Ro,R1,...Rp = coefficients d'autocorrélation
k1,k2,...kp = coefficients de réflexion
G = gain
Fo = fondamental

LE "LPCM" (cf. Hofstetter, Tierney, Wheeler 1977)

Une réalisation à base de micro-processeurs d'un vocodeur à prédiction linéaire a donc été faite.

Le but de cette réalisation était de fournir un produit compact, pas cher, donc facilement commercialisable. De ce fait, les possibilités de la machine ont été volontairement réduites.

Les caractéristiques de la machine sont les suivantes :

- temps de cycle de 150 nanosecondes (600 nanosecondes pour une multiplication).
- calculs en virgule fixe sur 16 bits en complément à 2. Le schéma général de la machine est présenté figure 6.

L'unité arithmétique et logique est constituée d'un processeur central et d'un multiplieur.

- le processeur central se compose de 4 micro-processeurs en tranches de 4 bits (type AMD 2901).
- le multiplieur comporte 8 éléments multiplieurs de 4 x 2 bits. Une multiplication 16 bits x 16 bits avec résultat sur 32 bits demandera 4 cycles machine.

Lors de l'exécution d'une multiplication, le processeur central reste libre pour d'autres tâches.

La mémoire-données se compose de 2 parties :

- une mémoire non volatile (ROM) de 512 mots de 16 bits. Cette mémoire contient les différentes tables nécessaires à l'algorithme de prédiction linéaire.
- une mémoire volatile (RAM) de 1 K mots de 16 bits. Les données du programme y sont rangées.

La mémoire-programme comporte 1 K mots de 48 bits. Elle contient le programme sous forme de micro-instructions.

...

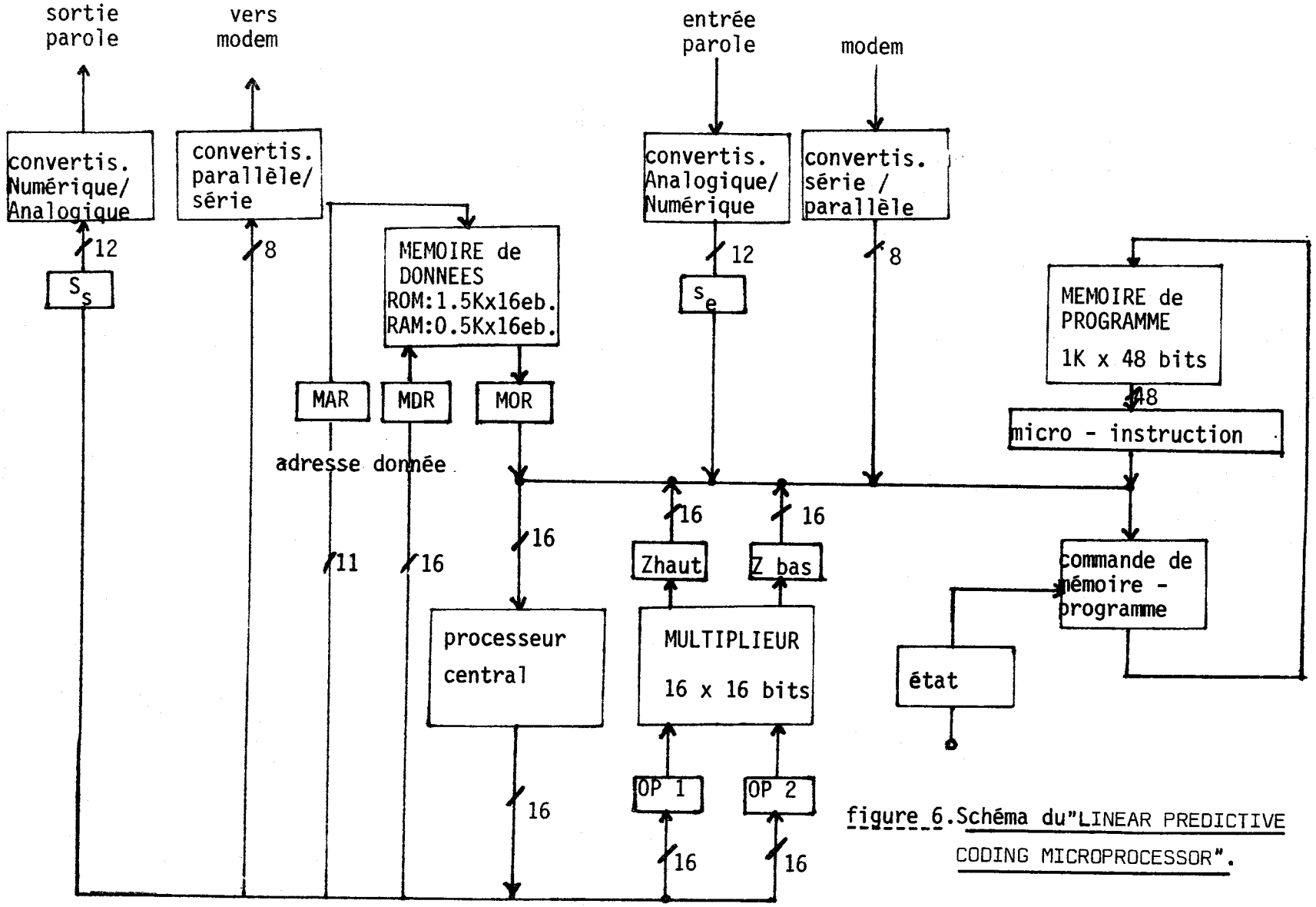


figure 6. Schéma du "LINEAR PREDICTIVE CODING MICROPROCESSOR".

L'algorithme de prédiction linéaire est traité de la même façon que dans le "Digital Voice Terminal" (figure 5).

CONCLUSION SUR LES SYSTEMES SPECIALISES

Les machines étudiées ont été conçues pour le codage de la parole et non à des fins de reconnaissance.

Le DVT reste une machine d'étude, constituant le premier pas vers une réalisation plus spécialisée.

Le LPCM est cette machine spécialisée. Il sera sans doute commercialisé d'ici peu, à un prix relativement accessible (1 000 à 2 000 $) et sans doute après une intégration plus poussée. Cependant, ses limitations (3 débits de codage, impossibilité de changer l'algorithme de prédiction, impossibilité de changer les paramètres du système...) font qu'il ne correspond pas exactement à nos besoins.

Le codage par prédiction linéaire a été implanté sur des machines adaptées au traitement du signal (cf. MJ KNUDSEN 1975). Cependant, ce type de machines ne convient pas à tous les algorithmes de prédiction, car elles ne sont pas suffisamment rapides.

Les machines spécialisées ne donnent pas entière satisfaction. Il faut se demander s'il n'existe pas de machine non-spécialisée pouvant répondre à notre problème.

I.1.2. Les architectures de machines

Les limites des machines d'architecture classique sont dues en grande partie au caractère séquentiel des traitements. Nous allons passer en revue différentes architectures permettant de tirer partie du parallélisme et du caractère "pipe-line" de certains traitements. Nous allons voir également si de telles architectures sont adaptées à notre application.

Flynn (1966) a défini une classification en 4 catégories des architectures de machines :

1° L'architecture "SISD" (Single Instruction, Single Data)

C'est la machine classique monoprocesseur : il n'y a qu'un flux de données et un flux d'instructions.

Une telle architecture nous l'avons vu ne permet pas de traiter efficacement et en temps réel notre application. C'est donc une architecture d'un autre type qu'il faut chercher.

2° L'architecture "SIMD" (Single Instruction Multiple Data)

Le flux d'instructions est unique, le flux de données est multiple.

Le schéma général d'une machine d'architecture "SIMD" est donné figure 7.

Trois types de machines "SIMD" se sont développés :

- les "processeurs tableaux" (figure 8) : c'est le principe d'ILLIAC IV (cf. Barnes et al 1968).

...

- les processeurs associatifs : la machine "STARAN" fait partie de ce type de processeurs (cf. Sayre 1976).

- les processeurs ensembles (machine "PEPE")

Ce sont actuellement les "processeurs tableaux" qui se développent le mieux. En effet, si ILLIAC IV était une machine encombrante et peu utilisable, d'autres machines sont apparues depuis, permettant des communications plus complexes entre les différents processeurs (c'est le cas de la machine "MAP" adaptée au traitement de tableaux).

Certaines caractéristiques de ces machines sont intéressantes pour notre application.

En particulier :

- les calculs peuvent être décrits par des instructions de type vectoriel ;

- un grand nombre d'opérations pourront se dérouler simultanément sur des données différentes ;

- les opérandes manipulés simultanément peuvent être recherchés parallèlement.

D'une manière plus générale, le traitement du signal fait partie des applications adaptées à ces machines.

3° L'architecture "MISD" (Multiple Instruction Single Data)

Ce type d'architecture est basé sur le principe du travail à la chaîne. Le résultat d'une instruction ou d'une phase d'instruction doit être obtenu avant d'effectuer la nouvelle instruction ou la phase suivante de l'instruction. Les machines "pipe-line" font partie de cette catégorie.

Les décompositions "série" mises en évidence dans notre application correspondent à ce type de traitements.

...

Une architecture de type "MISD" peut donc être envisagée pour réaliser une machine adaptée à l'application.

4° L'architecture "MIMD" (Multiple Instructions Multiple Data)

Ce type d'architecture (figure 9) représente le véritable parallèlisme : des tâches indépendantes sont effectuées sur des ensembles distincts de données parallèlement.

Un système de synchronisations permet de gérer l'ordre d'exécution des tâches, ainsi que les conflits divers qui se présentent dans un tel système (partage de ressources...).

Un certain nombre de systèmes multi-processeurs a été réalisé en prenant pour base une architecture "MIMD".

Pour notre application, une telle architecture a l'avantage de permettre d'utiliser au mieux les décompositions tant parallèles que "série" mises en évidence. C'est donc vers ce type d'architecture que nous allons nous orienter. Cependant, l'application étant précise, ce n'est pas une machine générale "MIMD" que nous allons réaliser. En effet, les tâches sont connues, le type de communications est défini, aussi est-il inutile et même peu concevable de mettre au point un système trop général.

...

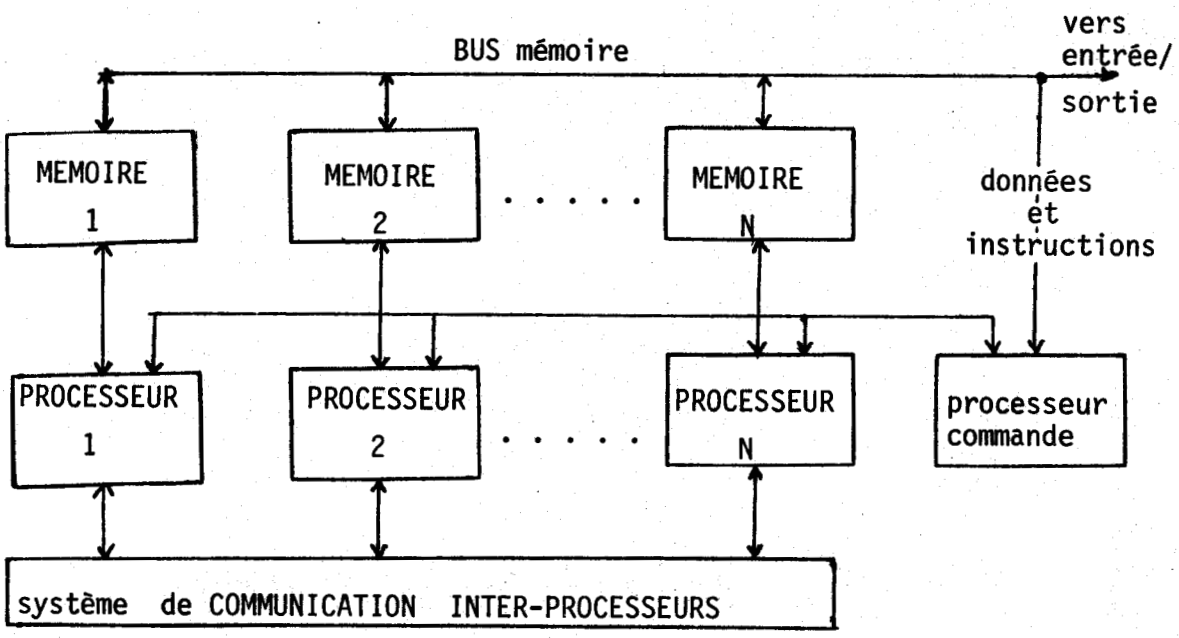


FIGURE 7. Schéma général d'une machine à architecture de type "SIMD".

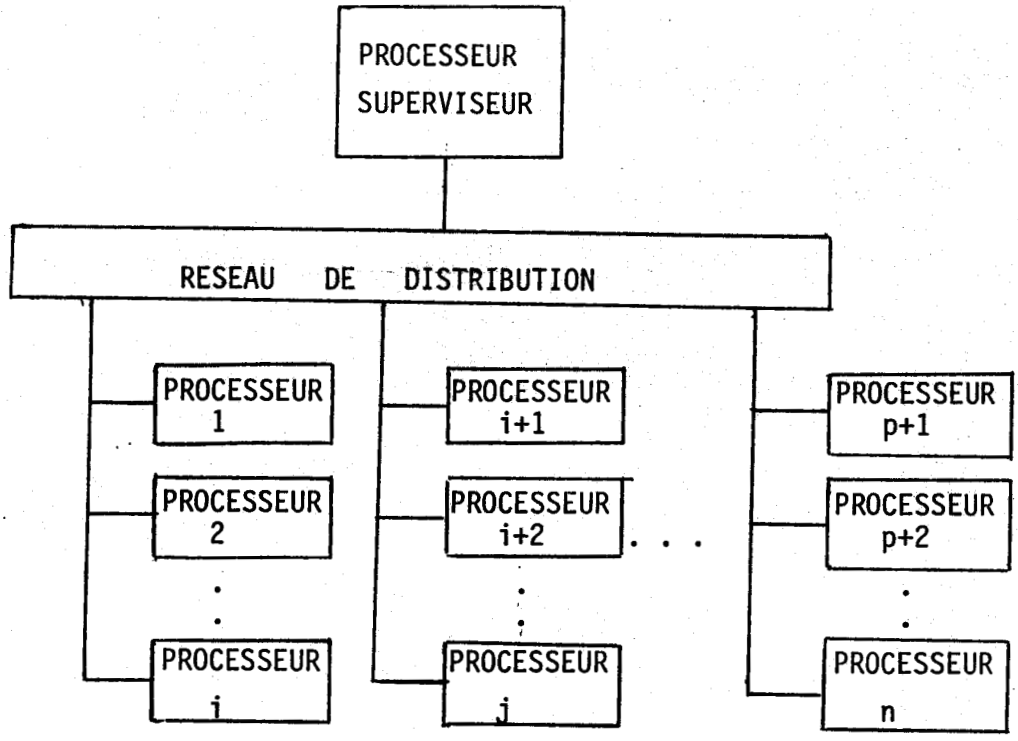


Figure 8. Schéma simplifié d'un"processeur tableaux "

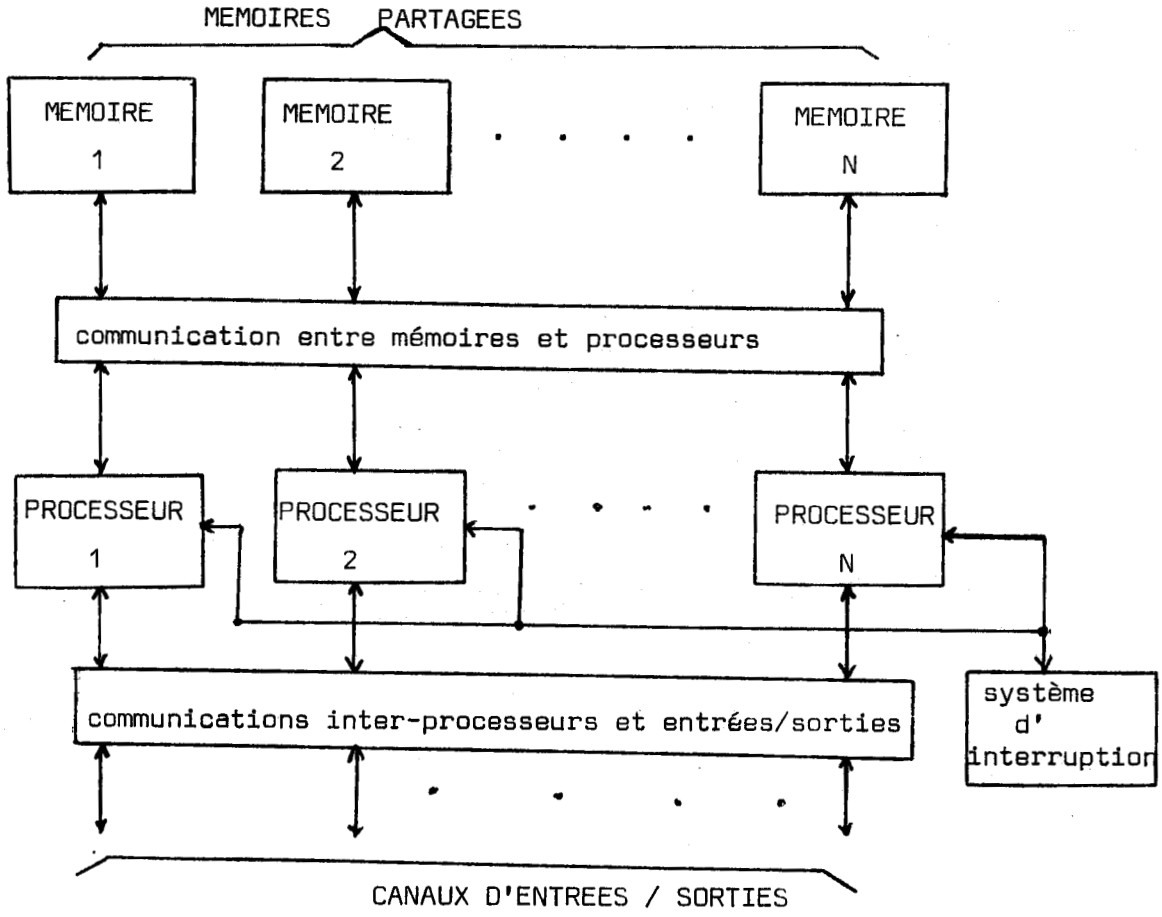


Figure 9 . Schéma général d'une architecture de type "MIMD" .

I.2/ LES SYSTEMES MULTI-MICROPROCESSEURS

Nous avons passé en revue un certain nombre d'architectures de machines utilisant plusieurs processeurs. Cependant, la réalisation de telles machines serait restée du domaine expérimental, s'il n'y avait eu les progrès récents de la technologie, liés à la diminution du coût des composants. Grâce aux techniques d'intégration à grande échelle (LSI), la complexité des composants double à peu près tous les deux ans. Par contre, le coût des composants, quelle que soit leur complexité, se situe aux environs de 100 francs lorsque le volume des ventes est suffisant. Les courbes 1, 2 et 3 décrivent ces phénomènes.

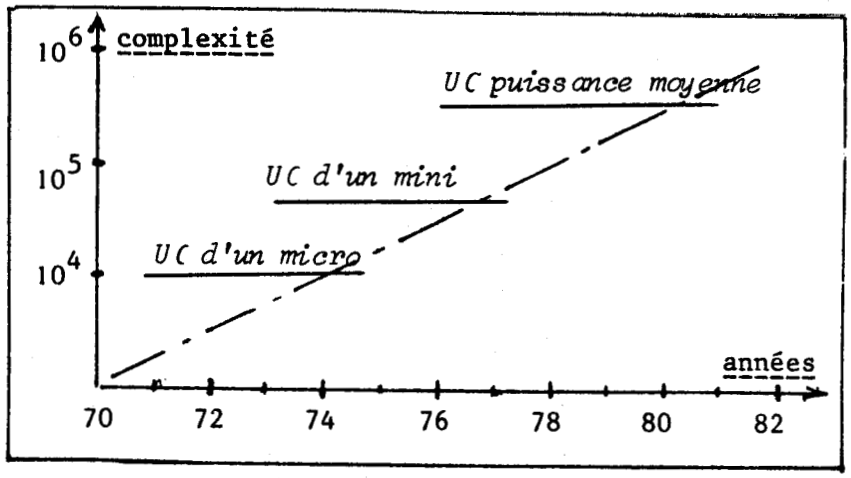
Cette "démocratisation" de l'informatique offre de nouvelles perspectives au concepteur, notamment dans le cadre des systèmes multi-processeurs.

Le microprocesseur est une unité de traitement classique, mais miniaturisée. Il est bon marché, performant et peu encombrant ; aussi représente-t'il un outil informatique de choix pour un grand nombre d'applications. La tendance actuelle vise donc à multiplier le nombre de microprocesseurs dans un système, quitte à les sous-employer, afin de diminuer le coût du logiciel, qui lui reste très élevé. L'utilisation des microprocesseurs et des mémoires à grande capacité permet maintenant la réalisation de machines "multi-processeurs". Nous allons dégager les problèmes qui se posent lors de la conception de ces machines, puis décrire quels sont les types d'organisations possibles.

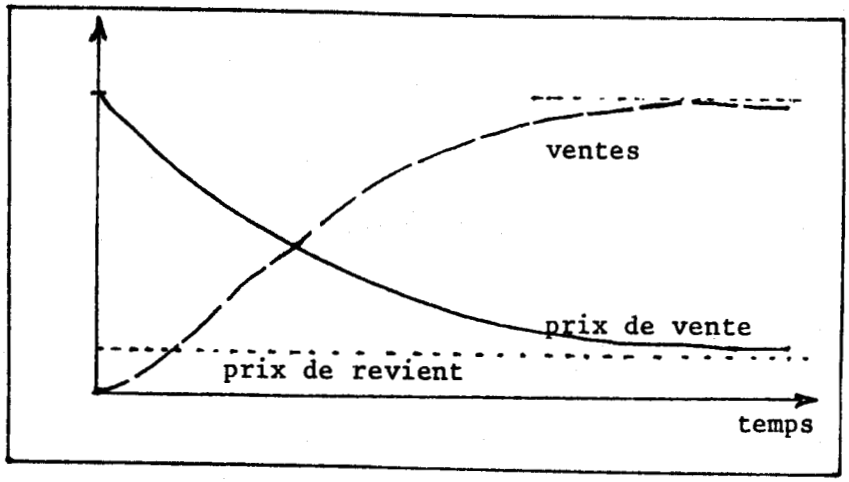
I.2.1. Problèmes de conception d'une machine multi-microprocesseurs

Par rapport aux systèmes conventionnels, des problèmes spécifiques se posent lors de la conception de systèmes multi-microprocesseurs. La plupart de ces problèmes sont d'ailleurs indépendants des microprocesseurs eux-mêmes.

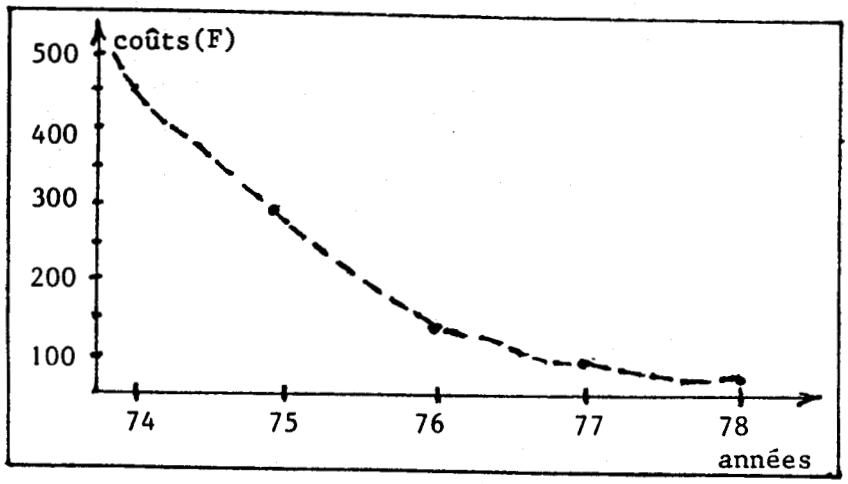
...



COURBE 1 : Evolution de la complexité des composants



COURBE 2 : Prix d'un composant / volume des ventes



COURBE 3 : Evolution du prix d'un microprocesseur

1) Problème de décomposition des tâches

Une application qui fonctionne en séquence, sur mono-processeur, doit pouvoir se décomposer en tâches susceptibles d'être traitées en pa rallèle. Il est donc nécessaire d'identifier les types de parallélismes existants dans l'application puis de subdiviser le traitement global en sous-tâches.

Ce travail de décomposition a été abordé dans le cas de l'analyse par prédiction linéaire. Des traitements parallèles et successifs ont été mis en évidence.

Il faut noter que dans certains cas, il est possible d'effectuer le découpage automatique d'une fonction, grâce à un compilateur adapté ou à des systèmes d'exécution spécialisés.

2) Problème de langage de programmation

Il faut choisir un langage évolué et adapté, qui permette à l'utilisateur de programmer son application de la manière la plus simple possible.

Ainsi, en plus des instructions usuelles d'un langage évolué, il est nécessaire de dégager des PRIMITIVES permettant la réalisation de fonctions spécifiques. Le langage sera interprété pour être assimilable par la machine. Des primitives de type vectoriel ont été définies pour notre application. D'autres primitives seront nécessaires, comme par exemple des primitives de synchronisation.

3) Problème de spécialisation des processeurs

Les processeurs doivent-ils être spécialisés dans certains types de travaux ou doivent-ils pouvoir exécuter n'importe quelle tâche (processeurs banalisés) ? Nous reviendrons sur ce problème qui détermin e l'organisation de la machine.

...

4) Problème de gestion système

Pour notre application, le modèle à réaliser est relativement rigide.

Ainsi, la gestion de l'allocation des processeurs physiques aux différentes tâches et l'ordonnancement des processeurs pour réaliser un traitement parallèle ou pipe-line ne sera pas quelconque.

5) Problème de protocole

Il faut établir un protocole de communication entre la mémoire commune et les processeurs, ainsi qu'entre les différents processeurs.

Les synchronisations et les échanges dans l'application portent sur des données connues ; celà facilite donc l'établissement d'un protocole.

6) Problème d'interférences entre tâches

Pour obtenir les meilleurs résultats du système, il est nécessaire de minimiser la dépendance des tâches entre elles, donc de limiter les communications et les accès à des ressources partagées.

7) Problème de survivance à la panne

Si un élément est défaillant, il faut pouvoir l'identifier, sauvegarder les tâches et les réassigner. Nous verrons par la suite que plus le degré de spécialisation des processeurs est élevé, plus le système a des chances de "s'écrouler" en cas de panne des processeurs.

A ce propos, il faut noter que ce ne sont pas les micro-processeurs qui tombent en panne le plus souvent : ce sont les organes d'entrée /sortie, les interconnexions internes ou les mémoires.

...

Des problèmes plus spécifiquement structuraux se posent également :

- choix de la méthode d'interconnexions des bus de liaisons.
- stratégie d'assignation des interruptions.
- outils d'évaluation et de contrôle pour la mise au point et le développement du système.

Le problème de spécialisation des processeurs conduit à étudier deux classes de systèmes multiprocesseurs :

- les systèmes symétriques où les processeurs sont banalisés.
- les systèmes asymétriques où les processeurs sont spécialisés.

Nous verrons les organisations de machines qui découlent de ces classes, ainsi que la manière dont les problèmes posés sont résolus dans chacun des cas.

I.2.2. Les systèmes multi-processeurs symétriques

Une fédération de n processeurs identiques se partagent le travail à effectuer.

Ce travail se compose d'un certain nombre de tâches, chaque tâche constituant un processus différent.

Les processeurs physiques constituent les ressources nécessaires à l'exécution desdits processus.

Dans un système symétrique, chaque processeur peut exécuter n'importe lequel des processus. Ce sont surtout des applications de type général qui peuvent être traitées de cette manière. Les processeurs étant équivalents, ils sont en concurrence pour traiter les processus en attente d'être exécutés. Ceci élimine l'attente nécessaire dans les systèmes asymétriques, où le processus ne peut s'exécuter que lorsque son processeur spécialisé est libre.

La modularité est un autre avantage de cette classe de systèmes. Si un processeur tombe en panne, ceci ne remet pas en cause le fonctionnement du reste du système. Il y aura seulement une perte globale de performance, puisqu'une unité de traitement manquera.

En contrepartie, les systèmes symétriques obligent à développer un matériel important, puisque chaque processeur doit pouvoir tout faire. De plus, les problèmes d'allocations des processeurs aux processus exigent un logiciel conséquent.

Organisation d'un système multi-processeur symétrique (figure 10)

N processeurs équivalents sont reliés à une mémoire commune. Les processeurs partagent donc cette mémoire, ainsi que les périphériques. Une mémoire locale minimale est nécessaire pour chaque processeur. Elle contient les éléments permettant de prendre en compte les interruptions locales de processeur.

GESTION DES RESSOURCES

a) les processeurs

Deux mécanismes sont possibles pour gérer l'allocation des processeurs aux processus :

- l'allocation centralisée,
- l'allocation distribuée.

Dans le premier cas, un allocateur global a décidé des affectations. L'algorithme d'allocation est traité par le processeur exécutant le processus qui a émis la requête. Le déroulement de l'algorithme peut remettre en cause l'allocation d'autres processeurs : un mécanisme doit alors permettre à n'importe quel processeur de prévenir nominativement n'importe quel autre processeur : c'est une réquisition naturelle.

...

Dans le second cas, chaque processeur a un algorithme propre d'allocation : c'est un mécanisme d'auto-allocation des processeurs.

Les processeurs libres vont explorer systématiquement la liste des processus candidats, en extraire le plus prioritaire ou le premier de la liste, et l'exécuter.

b) la mémoire commune

La mémoire commune est partagée par les différents processeurs. Il existe deux manières possibles d'allouer la mémoire à un processeur :

- le partage synchrone de la mémoire par multiplexage dans le temps des différents accès.

- le partage asynchrone de la mémoire : chaque processeur va se porter candidat à l'accès à la mémoire.

Un circuit de priorité peut par exemple autoriser l'un des processeurs à accéder à la mémoire.

PROBLEMES DE SYNCHRONISATIONS

L'accès aux ressources partagées doit être protégé.

Par exemple, deux processeurs ne doivent pas accéder en même temps à la liste des processus candidats.

Une des méthodes possibles consiste à employer le "Test and Set" (verrous d'accès permettant l'exclusion mutuelle sur les ressources "critiques").

...

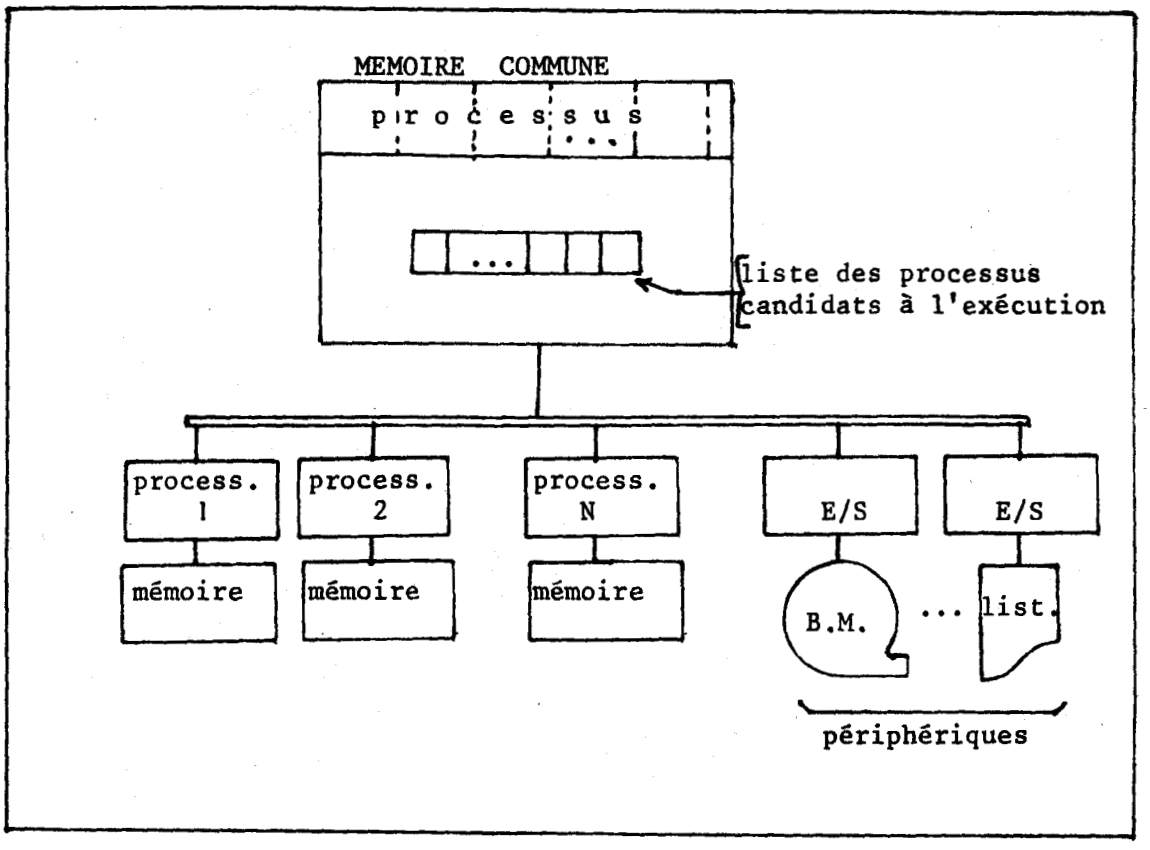


figure 10 . système multi-processeurs symétrique

MEMOIRE COMMUNE : Elle contient :

.Les processus utilisateurs

.Les processus système

.La liste des processus candidats à l'exécution

.L'allocateur de processeur

MEMOIRES LOCALES : Elles contiennent :

.Les mécanismes d'interruptions locales.

.Les mécanismes d'interprétation des interruptions externes.

I.2.3. Les systèmes multiprocesseurs asymétriques

Les processeurs ont des fonctions de traitement spécialisées.

Cette classe de systèmes est utilisée pour des applications précises. Le type et la fréquence d'occurence d'un processus sont connus à l'avance. Le temps d'exécution de chaque processus et la place mémoire occupée sont à peu près déterminés. Les processeurs sont spécialisés pour traiter un type particulier de tâches.

Dans le cadre de la "prédiction linéaire", un processeur peut par exemple être affecté à la détermination du fondamental, un autre à l'application de la fenêtre de Hamming, etc...

Le mécanisme global d'allocation des processeurs aux processus peut être supprimé. En effet, la décision d'assignation n'a pas besoin d'être faite en temps réel par le système, ce qui simplifie les problèmes logiciels qui en découlent. Par contre, c'est l'utilisateur qui doit répartir les processus dans les différents processeurs, de manière "équitable". C'est-à-dire, que la charge de travail dans les différents processeurs doit être à peu près équivalente.

Cette notion de spécialisations peut conduire à rendre les périphériques "intelligents" c'est-à-dire à associer un processeur à chaque périphérique.

ORGANISATION D'UN SYSTEME MULTI-PROCESSEUR ASYMETRIQUE (figure 11)

Les mémoires locales prennent de l'importance, la mémoire commune ne servant en limite que pour échanger des messages entre les processeurs.

...

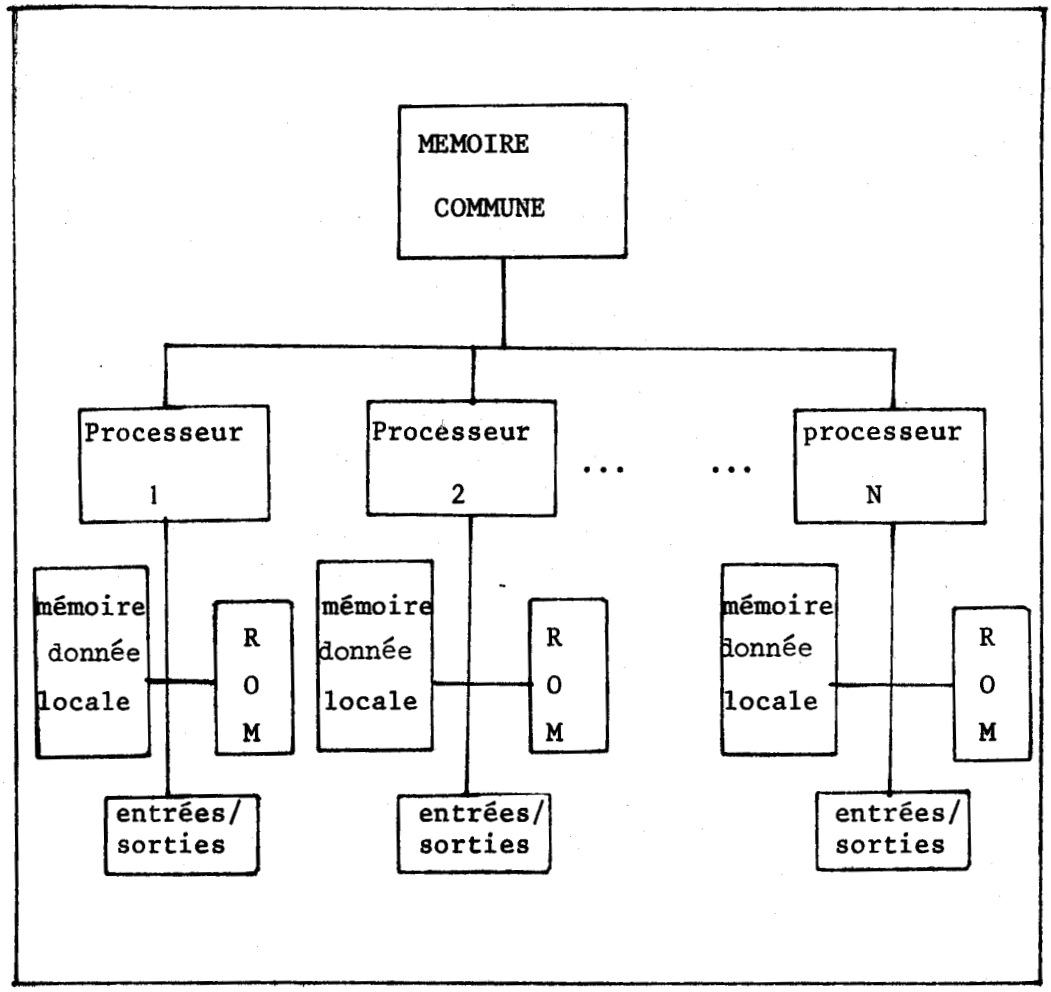


figure 11. Système multi-microprocesseurs asymétrique

MEMOIRE COMMUNE : Elle contient les messages entre processeurs.

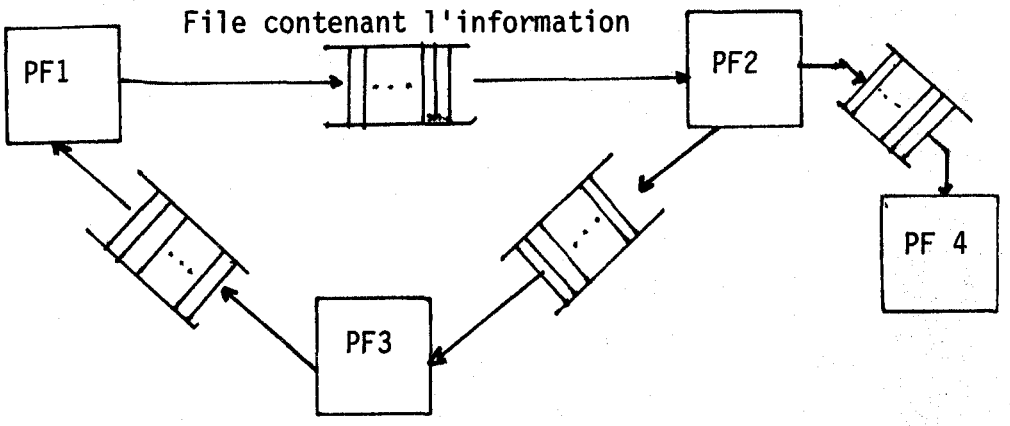
MEMOIRES -DONNEE LOCALES : Elles contiennent:
.Les données locales
. Les mécanismes d'interruptions locales

MEMOIRES - PROGRAMME : Elles contiennent les programmes spécialisés à effectuer par chaque processeur

Chaque processeur "fonctionnel", c'est-à-dire traitant une fonction spécialisée constituera en fait un ordinateur à part entière, possédant ses instructions, ses données et ses périphériques.

Problèmes de communication

La mémoire commune servira de "boîte aux lettres" entre les différents processeurs fonctionnels. Si elle ne sert que de moyen de communication, elle peut être remplacée par des files d'attente associées à chaque processeur :



Problèmes de synchronisation

Dans le cas de l'utilisation d'une mémoire commune pour les communications, un mécanisme de verrou est nécessaire pour assurer l'indivisibilité des modifications des informations mémorisées.

...

I.3/ PROPOSITION D'ARCHITECTURE

I.3.1. Compromis universalité/spécialisation

Le choix de concevoir une machine découle des remarques suivantes :

- les machines spécialisées existantes sont limitées dans leur utilisation et dans leurs possibilités de modifications.

- les machines de conception classique ne conviennent pas.

- les machines "générales" favorisant les traitements de tableaux ou les traitements de type "pipe-line" (machines MAP, PROPAL, MICRAL...) peuvent convenir, mais sont soit trop chères, soit non disponibles dans des délais raisonnables, soit enfin non adaptées spécifiquement à l'application.

- les progrès récents de la micro-informatique incitent le concepteur à définir une machine en fonction de ses besoins. Les techniques LSI (Large Scale Integration) fournissent en effet des outils performants et bon marché permettant de concevoir des architectures originales.

Ayant choisi de réaliser une machine, le concepteur doit définir son niveau de spécificité :

- il peut concevoir une machine générale, dont l'utilisation n'est pas précisément définie et dans lequel l'application est programmée.

- il peut concevoir une machine micro-programmée.

- il peut concevoir une machine câblée, figée pour l'application.

La première voie n'a pas d'intérêt si l'on vise les performances nécessitées par notre application.

...

L'application est précise, une architecture câblée semble donc plus adaptée si l'on désire un maximum de rapidité et de rendement du système. Cependant, toute spécialisation se caractérise par un compromis sur les possibilités de la machine. Or la machine à réaliser est destinée principalement à l'étude. Elle doit donc pouvoir évoluer (algorithmes différents, conditions d'analyses différentes...).

En dehors de ces considérations, un système permettant de mettre au point en temps réel des algorithmes appartenant à des classes voisines d'applications serait appréciable. La machine doit donc s'adapter à d'autres applications, comme le traitement du signal et le calcul des filtres numériques. Enfin, une extension de la machine devrait permettre d'implanter le système de reconnaissance de la parole "KEAL" actuellement en cours de développement au CNET. La machine traiterait alors la parole en temps réel, de la phase d'analyse du signal jusqu'à la phase de compréhension du message.

C'est donc un choix d'architecture orienté vers des applications spécifiques qui a été retenu.

La réalisation d'une machine multi-processeurs, à base de micro-processeurs représente le moyen le plus intéressant pour traiter notre application.

La microprogrammation permet de conserver la souplesse du logiciel et les performances d'un système câblé.

...

I.3.2. Le système multi-microprocesseurs

Nous avons défini précédemment deux classes de systèmes multi-processeurs :

- les systèmes multi-processeurs symétriques où les processeurs sont banalisés.

- les systèmes multi-processeurs asymétriques où les processeurs sont spécialisés.

AVANTAGES/INCONVENIENTS DES DEUX CLASSES DE SYSTEMES

Avantages des systèmes symétriques

- ils présentent une structure souple et aux possibilités diverses, les applications sont de type général.
- les processeurs étant banalisés, l'ensemble est modulaire.
- la réalisation matérielle est relativement simple.
- en cas de panne d'un processeur, le système ne s'écroule pas.

Inconvénients

- les programmes et les données sont centralisés en mémoire commune, les accès à cette mémoire sont donc très nombreux.
- le logiciel, pour gérer l'ensemble et en particulier l'allocation des processeurs aux processus, est complexe à mettre en oeuvre.
- le caractère général de ce type de systèmes nuit aux performances lors de l'implantation d'applications précises.

...

Avantages des systèmes asymétriques

- la spécialisation des processeurs permet d'obtenir les meilleurs résultats pour des applications précises.
- la décentralisation du matériel permet une meilleure autonomie des différents processeurs.
- les programmes et les données propres à chaque processeur sont en mémoire locale, les accès à la mémoire commune sont limités aux transmissions de messages.
- le logiciel pour gérer l'ensemble est simplifié.

Inconvénients

- la multiplication du matériel complique la réalisation.
- la spécialisation des processeurs diminue la souplesse de l'ensemble.
- en cas de panne d'un processeur, le système s'écroule.

LE CHOIX D'ARCHITECTURE

Puisque nous désirions nous placer dans une optique plus vaste que la simple réalisation de l'analyseur à prédiction linéaire, nous avons opté pour la machine multi-microprocesseurs semi-spécialisée.

Il est intéressant de conserver la "modularité" du système à processeurs banalisés et la "spécificité" du système à processeurs spécialisés.

Les processeurs sont donc équivalents, mais l'utilisateur a cependant la possibilité de spécialiser un ou plusieurs processeurs.

Le chapitre suivant décrit la structure matérielle de la machine qui découle de l'architecture choisie.

...

CHAPITRE DEUX

STRUCTURE MATERIELLE DE LA MACHINE

...

CHAPITRE DEUX - STRUCTURE MATERIELLE DE LA MACHINE

II.1/ STRUCTURE GENERALE DE LA MACHINE

La machine a une structure de type "multi-processeurs".

Elle comporte de 1 à 8 processeurs identiques, une mémoire commune, ainsi qu'un coupleur pour connecter un calculateur hôte.

Les différents processeurs sont équivalents, mais adaptés au traitement vectoriel, grâce à des micro-programmes spécialisés.

Un multiplieur accumulateur rapide, incorporé à chaque processeur, favorise les applications de type "traitement du signal", "calcul de filtres numériques" et "calcul de F.F.T." (Fast Fourier Transform) en temps réel.

Les mots-machine ont une longueur de 16 bits et les calculs sont effectués en virgule fixe.

La vitesse globale de la machine varie suivant la charge de travail des différents processeurs.

Chaque processeur est en fait un ordinateur à part entière, possédant son unité centrale, sa mémoire locale et son système d'entrées-sorties propre.

Le temps de cycle d'un processeur est de 200 nanosecondes. C'est donc le temps nécessaire à effectuer une opération quelconque (même une multiplication grâce au multiplieur rapide).

...

La figure 1 présente la structure générale de la machine.

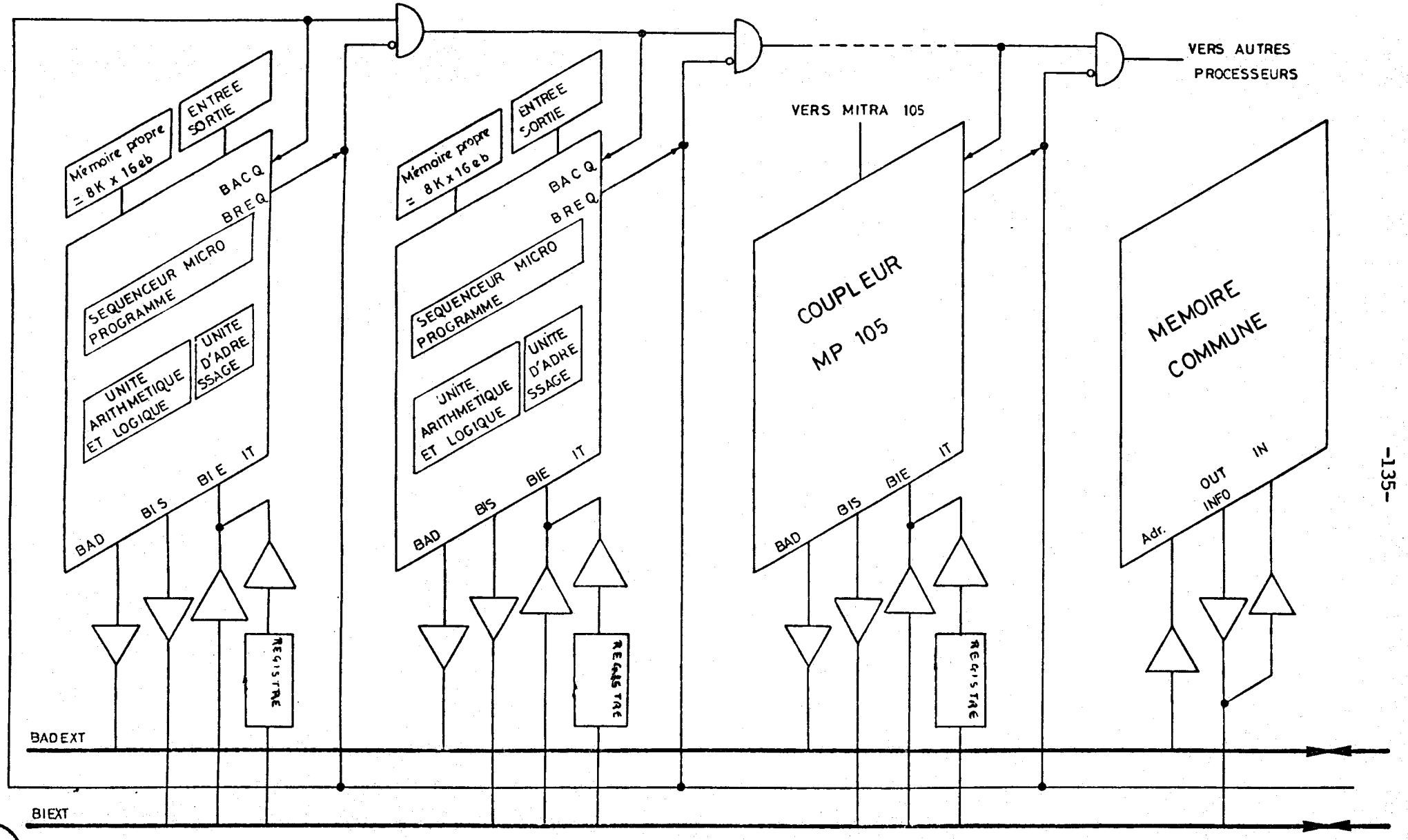
Les n processeurs dialoguent entre eux et avec la mémoire commune, grâce à des bus communs, asynchrones (bus commun d'adresse et bus commun de données). L'accès aux bus communs se fait par un assemblage en chaîne des différents processeurs (daisy chain). Sur la figure 1, c'est le processeur de priorité "0" qui est le plus prioritaire. Il fait une demande d'accès aux bus communs (BREQ), s'il y est autorisé (BACQ), il bloque l'accès pour les autres processeurs. La "daisy chain" présente l'avantage d'être simple à réaliser du point de vue technologique. Par contre, ce type d'assemblage est dangereux si les processeurs les plus prioritaires demandent souvent les bus communs, et donc empêchent les autres processeurs d'y accéder.

La mémoire commune peut contenir des données et des procédures globales, mais elle sert principalement de "boîte aux lettres". Un processeur, lorsqu'il a accès aux bus communs, peut envoyer un message dans la mémoire commune. Ce message sera lu par le processeur destinataire lorsque ce dernier aura lui-même accès aux bus communs. La communication inter-processeurs se fait donc par l'intermédiaire de la mémoire commune, mais il existe également la possibilité pour un processeur d'envoyer directement une interruption à l'un quelconque des autres processeurs.

La figure 2 décrit de manière simplifiée la structure d'un processeur. Les différentes unités communiquent grâce à 3 bus :

- le bus d'entrée d'information (BIE)
- le bus de sortie d'information (BIS)
- le bus d'adresse (BAD).

...



BACQ : Autorisation d'accés aux bus communs
BREQ : Demande d accés aux bus communs
BADEXT: Bus adresse commun
BIEXT : Bus donnée commun

Figure 1. STRUCTURE GENERALE.

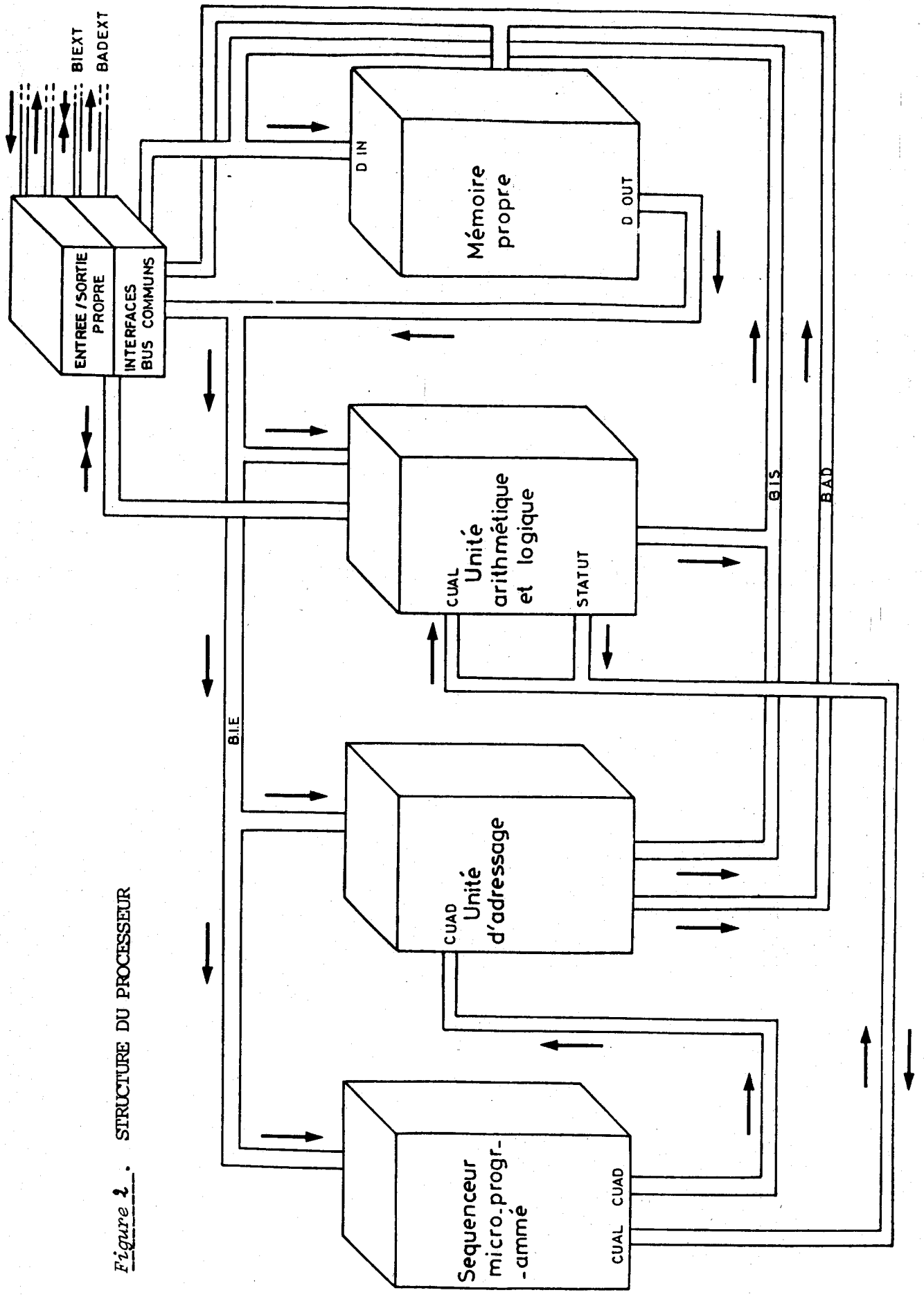


Figure 2 . STRUCTURE DU PROCESSEUR

II.2 / L'UNITE CENTRALE D'UN PROCESSEUR

Les unités de séquencement, d'adressage, et l'unité arithmétique et logique sont séparées.

II.2.1. L'unité de séquencement (US)

L'unité de séquencement (figure 3) permet de commander l'unité d'adressage et l'unité arithmétique et logique.

Elle comprend :

- un registre d'instruction,
- un séquenceur microprogrammé (AM 2910),
- une mémoire de microprogrammes,
- un registre de micro-instruction .

Une instruction, précédemment extraite de la mémoire propre du processeur, (ou éventuellement de la mémoire commune), est chargée dans le registre d'instruction.

L'instruction comporte 2 parties :

- un code opératoire (sur 12 bits) qui est en fait une adresse de branchement à un des microprogrammes de la mémoire de microprogrammes.

- l'adresse d'un des 16 registres de l'unité d'adressage (sur 4 bits).

Lorsque l'instruction est chargée, le micro-programme correspondant s'exécute.

Le séquenceur AM 2910 gère l'enchaînement des micro-instructions qui composent le micro-programme.

...

La micro-instruction à exécuter est rangée dans le registre de micro-instructions. Son exécution provoque la commande simultanée du séquenceur lui-même, de l'unité d'adressage et de l'unité arithmétique et logique.

Il faut noter que la partie "commande du séquenceur" de la micro-instruction permet de sélectionner la micro-instruction suivante à exécuter. C'est un recouvrement de type "pipe-line à 1 niveau".

Caractéristiques du "AM 2910" (figure 4)

- il commande et contrôle le séquencement des microinstructions à exécuter qui sont rangées en mémoire de micro-programmes.

- il permet d'accéder à n'importe quelle microinstruction lorsque la mémoire de micro-programmes comporte 4 K mots. Un mot a ici la longueur d'une microinstruction (64 bits). Il travaille donc sur des adresses de 12 bits (adresses de 0 à 4095).

- outre les possibilités d'accès séquentiel, il réalise les branchements conditionnels (la condition étant définie par le positionnement préalable de drapeaux) ou inconditionnels.

- une pile de type "LIFO" (last in, first out) sert dans 2 cas :

  - pour mémoriser les adresses de retour dans le cas d'appels à des "sous-microprogrammes". La pile comportant 5 mots de 12 bits, il est donc possible d'avoir 5 "sous-microprogrammes" imbriqués.

  - pour mémoriser les adresses de retour de boucles jusqu'à 5 boucles imbriquées. Le registre/compteur sert dans le cas des boucles de "décompteur" d'itérations (quand il arrive à zéro : fin de boucle).

...

- un multiplexeur sert à sélectionner l'adresse de la microinstruction suivante à exécuter. Cette adresse peut avoir 4 origines.

  - le registre d'adresse-microprogramme (mPC) contenant généralement l'adresse de la micro-instruction qui suit séquentiellement la précédente.

  - le registre-compteur mémorisant une adresse lors de l'exécution d'une micro-instruction précédente.

  - une adresse externe "D" correspondant à une entrée directe. Cette adresse est fournie soit par le registre de micro-instruction,soit par le champ adresse du registre d'instruction.

  - le premier élément de la pile (le dernier entré) qui fournit une adresse de retour.

- 16 instructions différentes de commande, liées à des conditions, sont possibles. Le décodeur d'instructions est incorporé au boîtier.

Le"contrôleur de microprogrammes" AM 2910 est un circuit récent fabri-qué par AMD . Il a l'avantage d'adresser une mémoire de microprogrammes de 4 K mots,ce qui est suffisant pour la plupart des applications . L'utilisation des anciens circuits (il aurait fallu 3 boitiers AM2909 pour adresser 4 K mots) ne se justifie que s'il est nécessaire d'avoir une mémoire de microprogrammes plus importante .

...

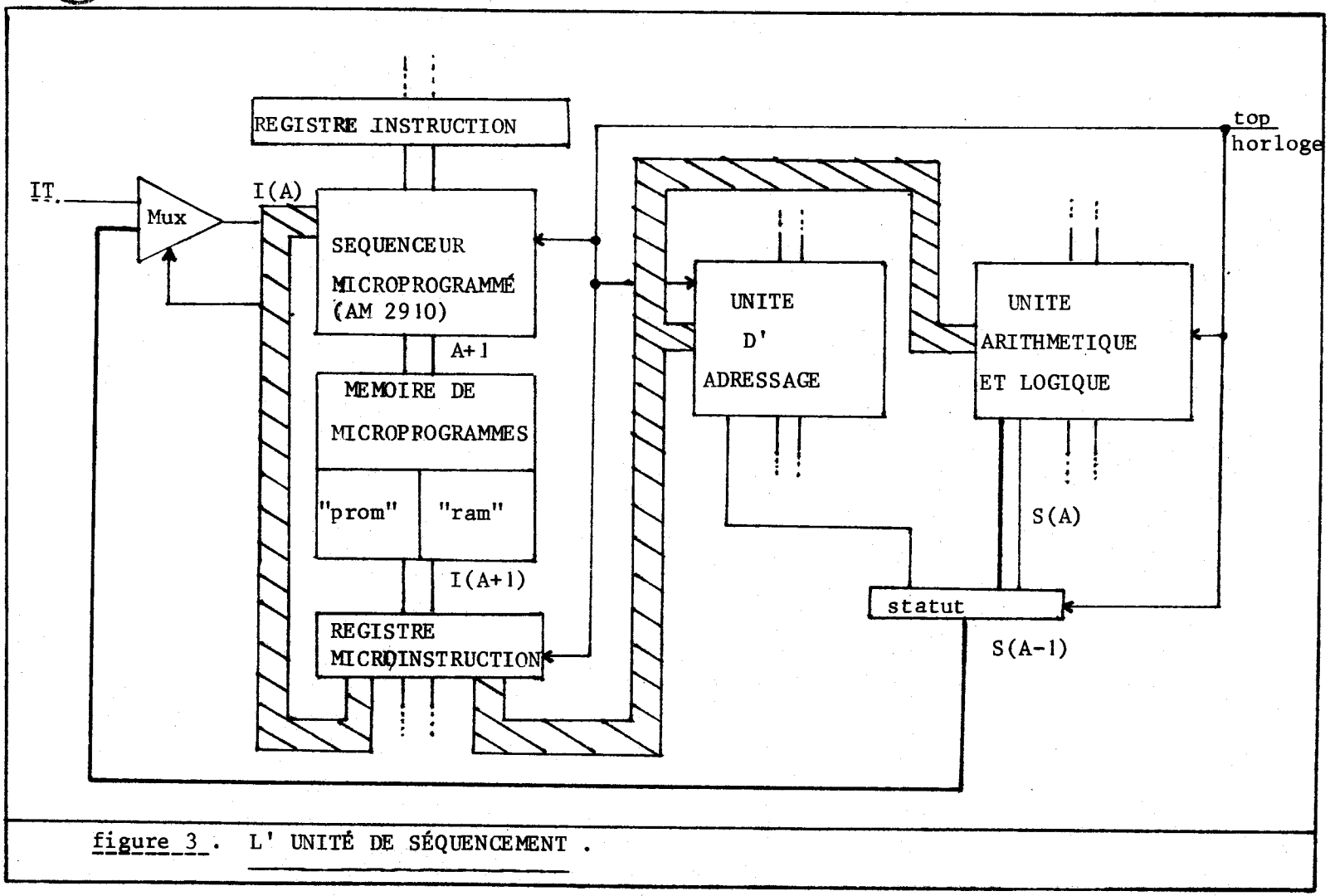


figure 3 . L' UNITÉ DE SÉQUENCEMENT .

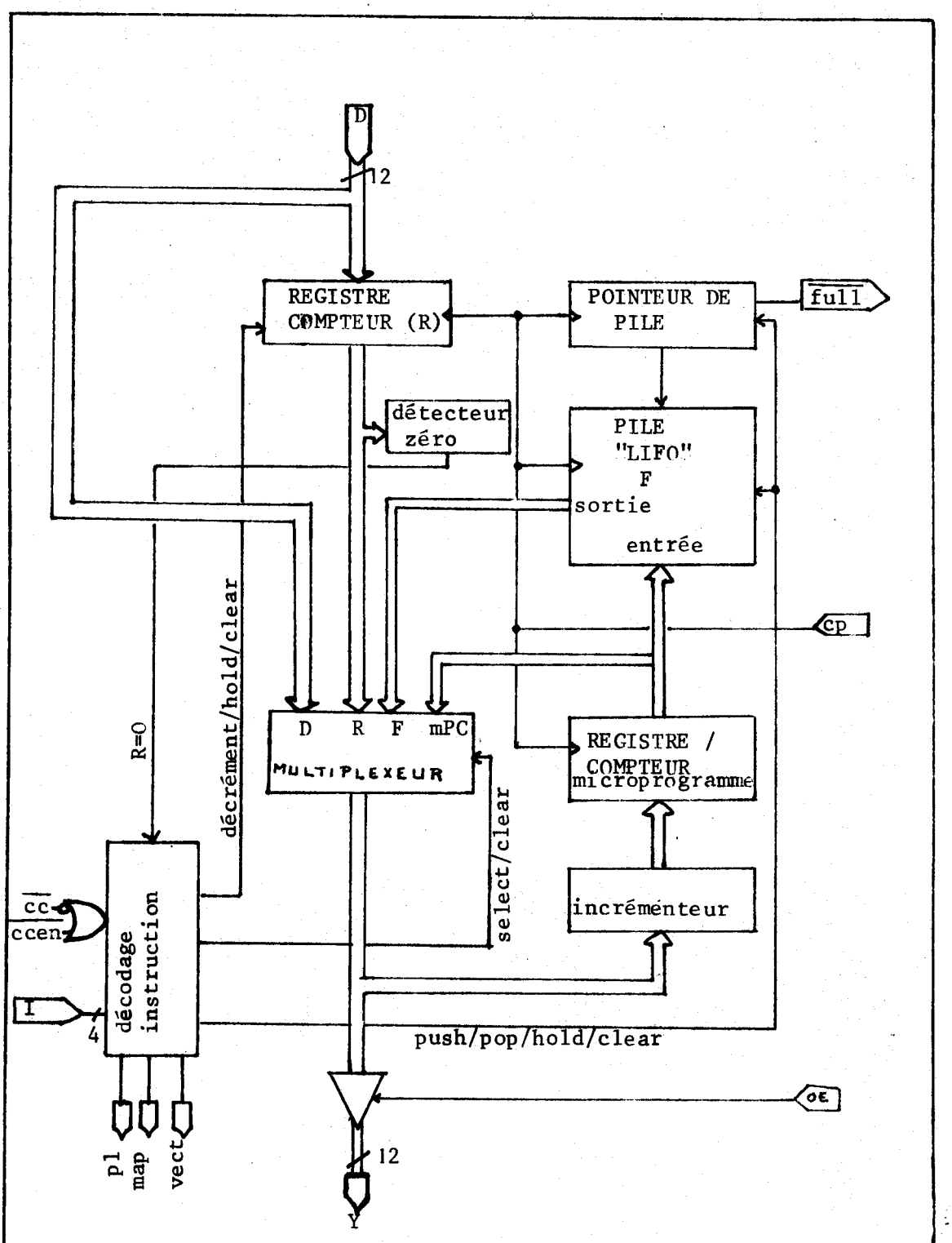


figure 4. CONTROLEUR DE MICROPROGRAMME "AM 2910"

La mémoire de micro-programme de 4 K mots de 64 bits se divise en 2 parties :

- une mémoire "PROM" non volatile : elle contient les micro-programmes de base, communs à toutes les applications : ces micro-programmes seront donc dupliqués dans les mémoires "PROM" de tous les processeurs.

- une mémoire "RAM" volatile : elle contient des micro-programmes spécifiques à une application ou à une tâche. Elle sera donc chargée avant le début de l'exécution de cette application ou de cette tâche.

Abrévations utilisées sur la figure 4

D	Entrée directe d'une adresse dans le registre/compteur ou dans le multiplexeur
I	Sélectionne une des 16 instructions de commande possible.
$\overline{CC}$ $\overline{CCEN}$	Code condition Code condition
$\overline{OE}$	Commande 3 états de la sortie Y
CP	Impulsion d'horloge
Y	Adresse sélectionnée en mémoire de micro-programme
$\overline{FULL}$	Indication que la pile est pleine
$\overline{PL}$ $\overline{MAP}$ $\overline{VECT}$	Permettent de sélectionner la source de l'entrée directe D

...

II.2.2. L'unité d'adressage (UAD)

L'unité d'adressage est bâtie autour de microprocesseurs en tranches (4 fois 4 bits) de type AMD 2901.

Un registre d'adresse (RAD) permet de mémoriser l'information qui est envoyée à l'unité d'adressage à travers le bus information d'entrée (BIE).

L'AMD 2901 (figure 5) se compose principalement d'une unité arithmétique et logique associée à un ensemble de 16 registres adressables.

L'un des 16 registres est utilisé comme compteur ordinal, les autres servent de registres de base, d'indexation, de pointeurs...

Seule une partie des fonctions possibles grâce aux AMD 2901 a été gardée .

En résumé, 3 types d'opérations sont utilisés :

- les opérations de chargement portant sur le registre d'adresse.

Par exemple, chargement du contenu de RAD dans un registre interne AR_i...

- les opérations arithmétiques portant sur les registres internes.

Par exemple, incrément de 1 du contenu d'un registre AR_i et rangement du résultat dans AR_i...

- les opérations de masquage entre 2 registres internes ou entre le registre d'adresse et un registre interne (utilisation du "et" logique). Une partie de la figure 6 décrit l'unité d'adressage.

...

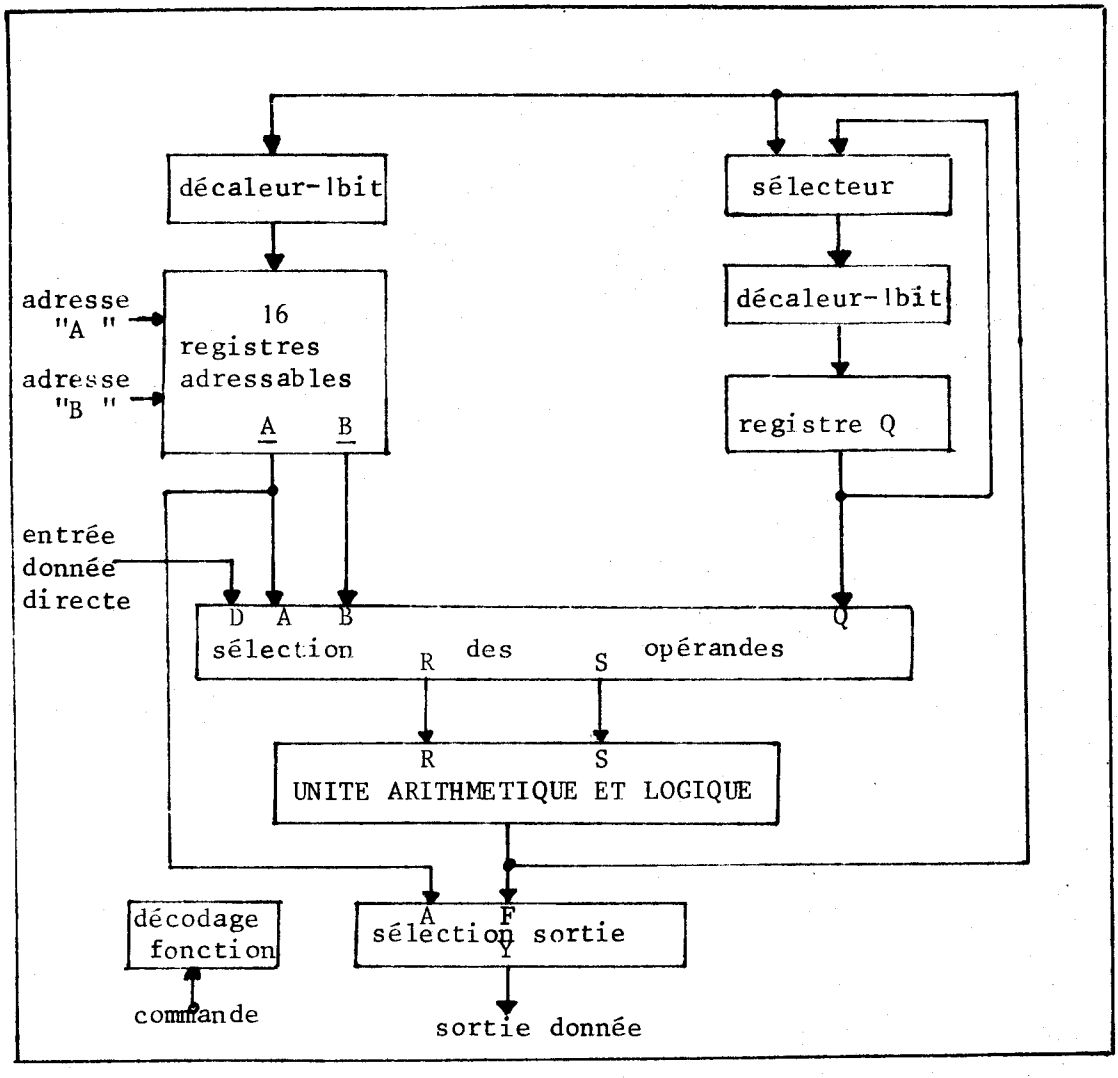


figure 5. Schéma simplifié du AM 2901 .

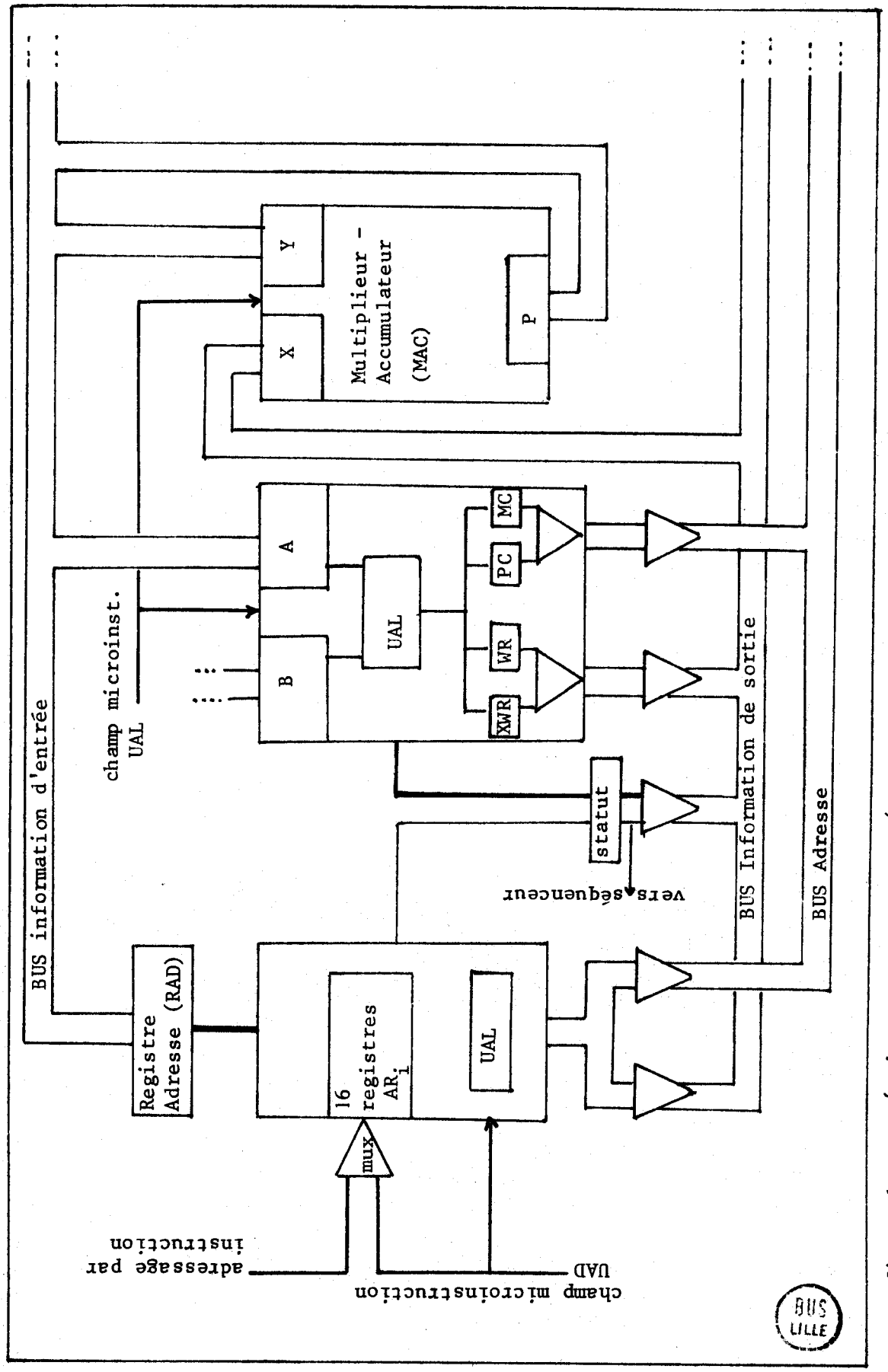


figure 6 . L'UNITÉ D'ADRESSACE ET UNITÉ ARITHMETIQUE ET LOGIQUE .

II.2.3. L'unité de traitement

L'unité de traitement (figure 6) comporte 2 éléments distincts :

- une unité arithmétique et logique (UAL). Elle permet d'effectuer toutes les opérations élémentaires (additions, soustractions, opérations logiques).

- un multiplieur-accumulateur (MAC). Il permet d'effectuer un produit suivi d'une addition

II.2.3.1. L'unité arithmétique et logique

Comme dans l'unité d'adressage, ce sont des microprocesseurs en tranches qui ont été utilisés. Quatre "74LS481" montés en cascade constituent l'unité arithmétique et logique. Contrairement aux "AM2901", ces microprocesseurs ne possèdent pas de pile de registres internes. Cependant, la mémoire locale associée au processeur ayant un temps d'accès de l'ordre de 50 nanosecondes, l'utilisation de registres internes n'est pas justifiée. Un microprocesseur (voir figure 7) ne contiendra donc que les registres indispensables :

- un accumulateur (WR)
- un registre d'extension (X WR)
- un compteur programme (PC)
- un compteur mémoire (MC)

Les fonctions principales réalisées dans l'UAL sont :

- chargement d'un opérande par le "port" A dans l'accumulateur WR.

- addition, soustraction entre deux opérandes (l'un des opérandes a été en général chargé dans l'accumulateur auparavant).

...

- fonctions logiques entre 2 opérandes :

"ou", "non ou","non ou exclusif" (les fonctions "ET" et "NON ET" sont possibles en choisissant les opérandes).

- les comparaisons entre 2 opérandes :

le résultat de la comparaison des 2 opérandes positionne des drapeaux (non défini, arithmétiquement supérieur, logiquement supérieur, égal...).

- les décalages logiques et arithmétiques :

les décalages logiques et arithmétiques, à droite ou à gauche, les décalages circulaires sont possibles. Ils peuvent être faits en simple précision (sur WR par exemple) ou en double précision (sur XWR et WR par exemple).

Le compteur mémoire (MC) ne sert pas. Par contre, le compteur-programme (PC) sert à la sauvegarde et à la restitution des registres de l'unité arithmétique et logique, de l'unité d'adressage, ou du registre de "statut".

Le "74LS481" est un microprocesseur en tranches dont la structure est très complexe. Ses possibilités arithmétiques sont importantes,en particulier,il est possible de préprogrammer des multiplications et des divisions,et de faire toutes sortes de décalages.

...

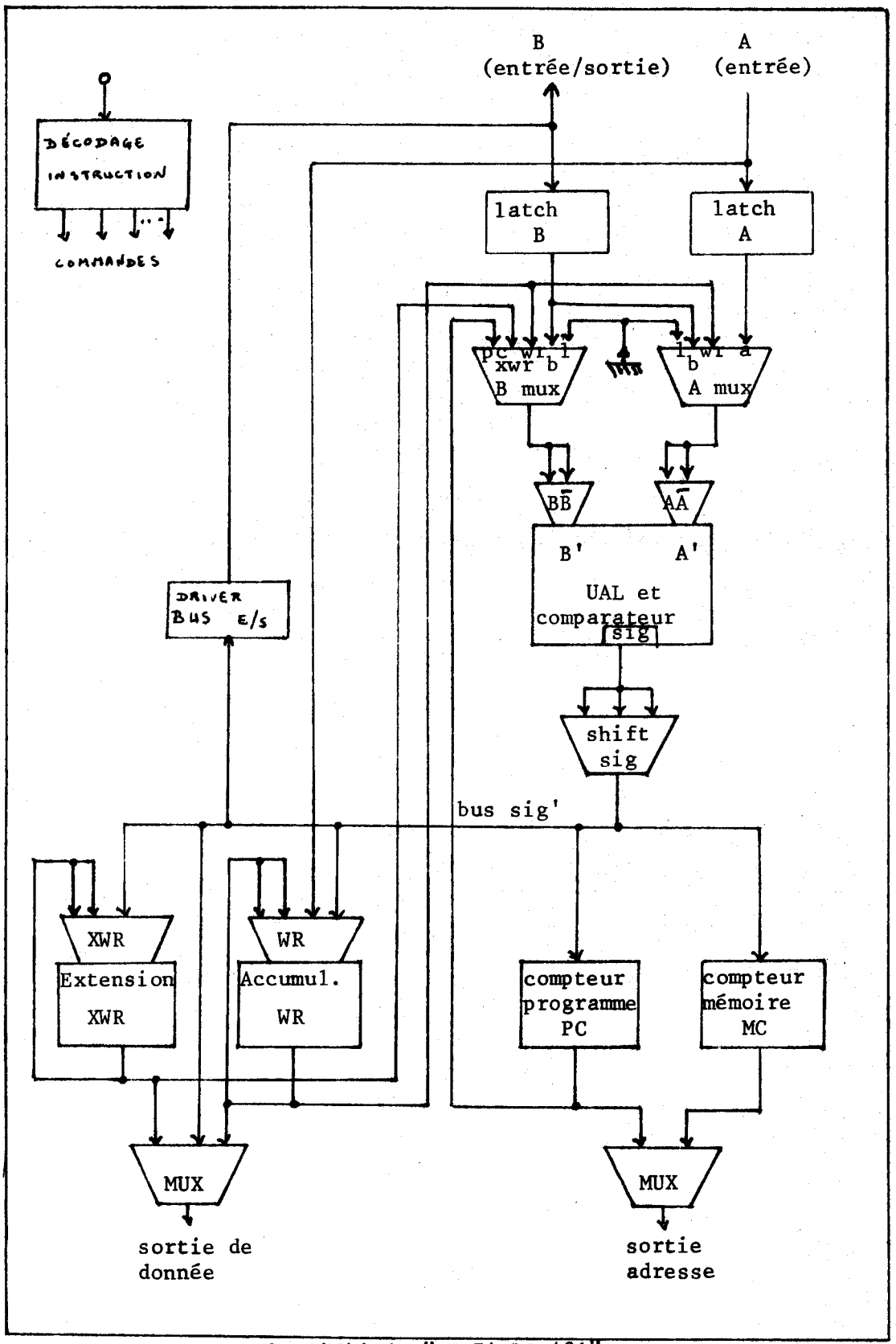


figure 7. Schéma simplifié du "SN 74 LS 481" .

II.2.3.2. Le multiplieur-accumulateur

Le circuit "TDC 1010J" de chez "TRW" permet d'effectuer des multiplications 16 bits par 16 bits et de cumuler le résultat des produits successifs sur 35 bits (32 bits + extension). Une multiplication suivie d'une addition prend 115 nanosecondes. Le schéma fonctionnel de ce circuit LSI est donné figure 8.

Les nombreux produits-scalaires de vecteurs qui figurent dans les algorithmes de prédiction linéaire seront traités très rapidement grâce au multiplieur-accumulateur.

Les commandes qui peuvent être prises en compte par le multiplieur sont les suivantes :

- commande de "complément à 2"

Les opérandes en entrée sont des nombres sur 16 bits en complément à 2.

- commande d'"arrondi"

Si l'on ne désire sortir que les bits de poids forts, le résultat est arrondi et non tronqué.

- commande d'"accumulation"

Le contenu du registre-accumulateur P est augmenté du résultat du dernier produit calculé. Si la commande n'est pas faite, le résultat du produit est simplement rangé dans P.

- commande de "soustraction"

Si les commandes d'accumulation et de soustraction sont toutes deux faîtes, le contenu de P est soustrait au résultat du produit.

...

- commande de "préchargement"

Cette commande permet d'initialiser le contenu du registre P.

Cependant, cette initialisation ne se fait que si les 3 commandes qui suivent sont également faites.

- commandes 3 états

Ces commandes permettent, lorsque la commande précédente n'est pas faite, de sortir le registre P.

Commande 1 : sortie de "l'extension" (bits 34 à 32 de P)
Commande 2 : sortie des "16 bits de poids forts" (bits 31 à 16 de P)
Commande 3 : sortie des "16 bits de poids faibles" (bits 15 à 0 de P)

Les "ports" X et Y permettent de faire entrer les 2 opérandes de 16 bits.

Les "ports" bidirectionnels XTP et MSP permettent dans un sens de précharger les bits de poids forts et l'extension du registre P et dans l'autre sens de sortir le contenu de l'extension et des bits de poids forts de P.

Le port "Y" est également bidirectionnel et a 3 fonctions : il permet l'entrée d'une opérande, le préchargement des bits de poids faibles de P, et la sortie des bits de poids faibles de P.

...

remarque :

Le multiplieur-accumulateur est optionnel. En effet,son utilisation n'est justifiée que dans des applications où le nombre de multiplications est élevé (c'est le cas de l'analyse par prédiction linéaire) .

La suppression du multiplieur ne remet pas en cause la structure et le fonctionnement du processeur.

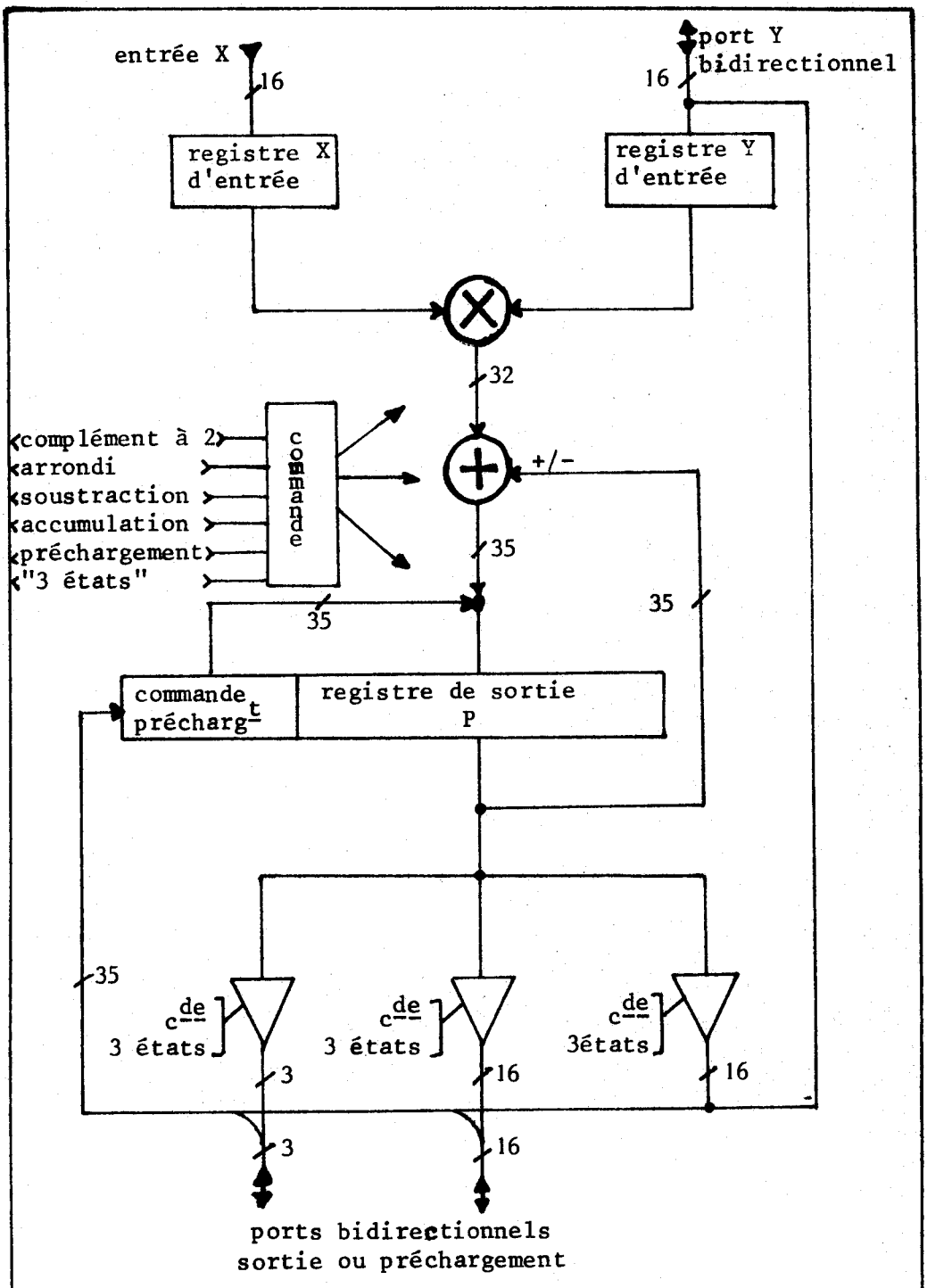


figure 8 . SCHEMA FONCTIONNEL DU TDC 1010 J .

II.3/ LES MEMOIRES

La machine possède :

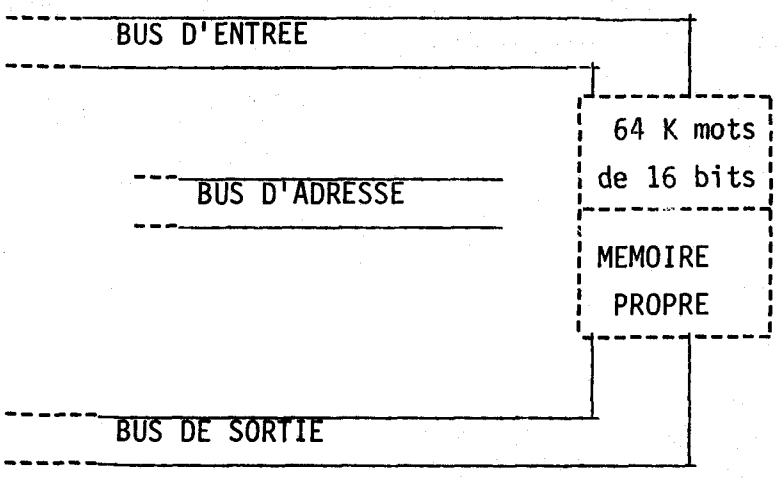
- une mémoire commune,
- des mémoires propres associées à chaque processeur.

II.3.1. La mémoire commune

- son temps d'accès est de 350 nanosecondes.
- sa capacité maximale est de 16 pages de 64 K mots.

II.3.2. Les mémoires propres

Chaque mémoire se compose d'une série de boitiers de 4 K mots de 16 bits. Sa capacité minimale est donc de 4 K mots de 16 bits, sa capacité maximale de 64 K mots. Son temps d'accès est de 50 nanosecondes.



...

II.4/ LES ENTREES/SORTIES ET LES INTERFACES BUS COMMUNS

II.4.1. L'unité d'entrées/sorties

L'unité d'entrées/sorties permet d'échanger de l'information entre un processeur et un ou plusieurs périphériques. Le "port" B (voir figure 6 et 7) est bidirectionnel. Il y a donc possibilité actuellement d'associer 16 lignes de commandes bidirectionnelles à un interface parallèle de 16 bits en émission et réception.

II.4.2. L'unité de dialogue "bus communs" (figures 9 et 10)

Cette unité permet de mettre en liaison les bus communs avec les bus internes des processeurs. Il y a donc une unité de ce type par processeur.

Il existe 3 types d'interfaces :

- l'interface 1 permet le dialogue entre la mémoire commune et un processeur.

- l'interface 2 permet le téléchargement depuis l'extérieur de la mémoire propre d'un processeur ; elle permet également de tester cette mémoire.

- l'interface 3 permet le dialogue direct interprocesseurs (interruptions par exemple).

...

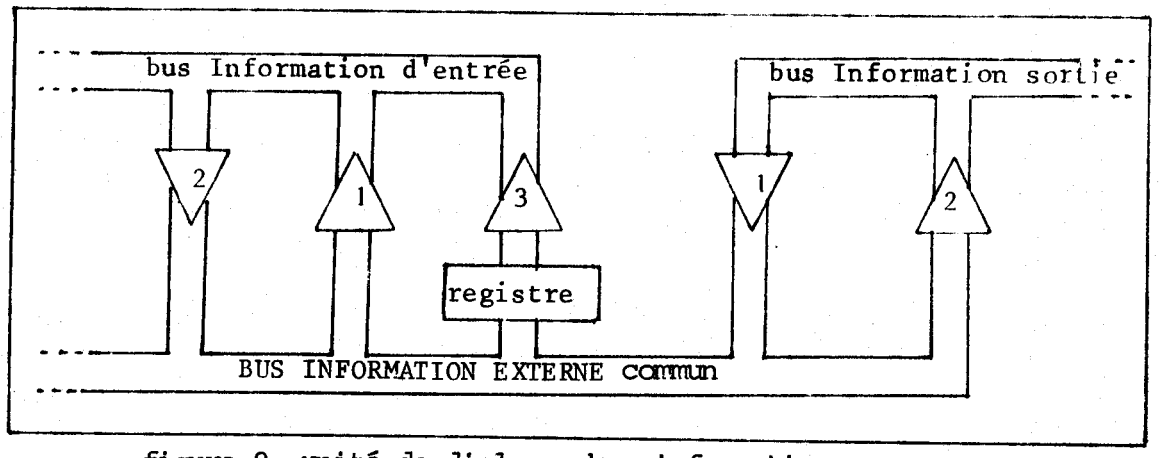


figure 9. unité de dialogue bus information

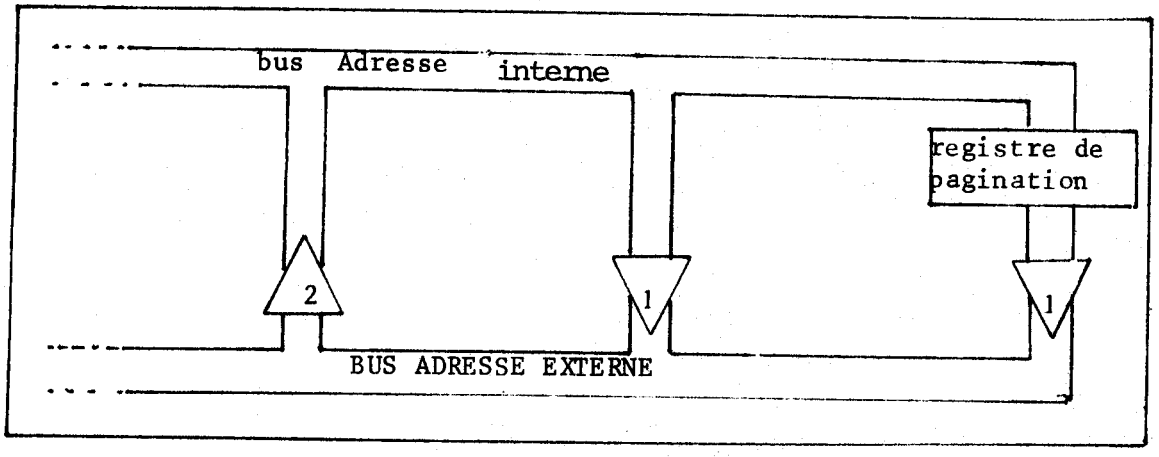


figure 10. unité de dialogue bus adresse

1 : interface 1.

2 : interface 2.

3 : interface 3.

II.5 / FONCTIONNEMENT D'UN PROCESSEUR (figure 11)

Le programme à exécuter et les données locales sont en mémoire propre du processeur. Les données globales sont en mémoire commune.

Un micro-programme particulier permet de charger une instruction dans le registre d'instruction . Les 12 bits de poids forts du registre d'instruction définissent une adresse en mémoire de micro-programme.

Cette adresse permet de sélectionner la première micro-instruction du micro-programme traitant l'instruction .

La micro-instruction est chargée dans le registre de micro-instruction. Son exécution provoque la commande parallèle des différentes unités (le séquenceur lui-même,l'unité d'adressage,l'unité de traitement , la gestion des bus). C'est un pipe-line à un niveau . La commande sur le séquenceur permet par exemple de sélectionner la micro-instruction suivante de micro-programme.

L'ensemble des unités du processeur fonctionne de manière synchrone . Le temps de cycle pour l'exécution d'une micro-instruction est de 200 nano-secondes.

Une interruption,vue du micro-programme,est en fait une condition qui est vérifiée ou non : C'est le code-condition (cc) qui agit sur le séquenceur.

L'adressage de la mémoire propre et de la mémoire commune se fait sur 16 bits. Ceci permet d'accéder à 64 K mots de mémoire. Un registre de pagination est associé à la mémoire commune. Un bit supplémentaire dans la gestion des bus indique si c'est à la mémoire propre ou à la mémoire commune qu'on veut accéder.

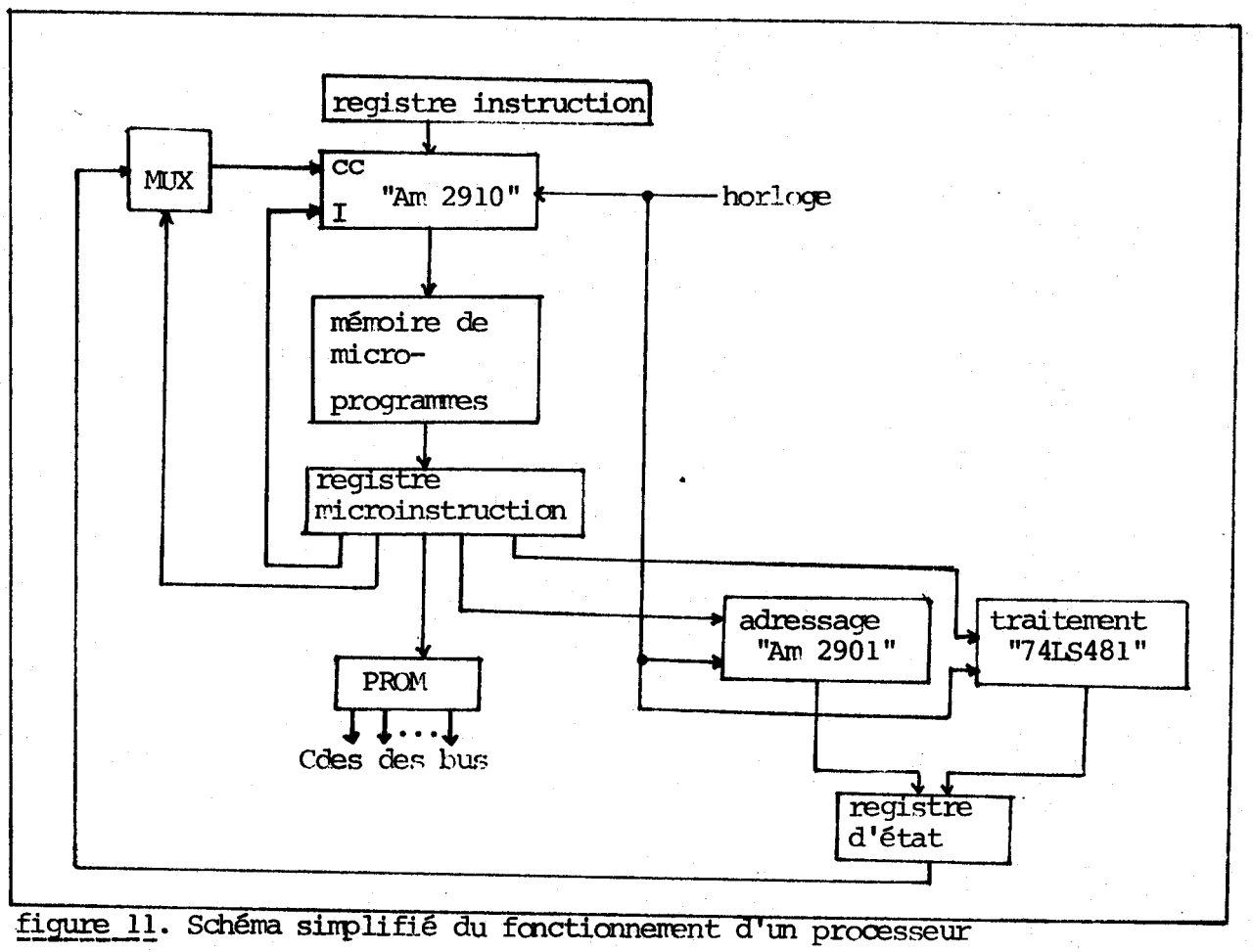


figure 11. Schéma simplifié du fonctionnement d'un processeur

CHAPITRE TROIS

ETUDE DU LOGICIEL DE SUPPORT

CHAPITRE TROIS - ETUDE DU LOGICIEL DE SUPPORT

III.1/ LA PROGRAMMATION DE LA MACHINE

La structure matérielle de la machine étant définie, il s'agit maintenant d'offrir à l'utilisateur la possibilité d'y programmer son application.

La définition d'un langage de haut niveau est nécessaire pour faciliter le travail de l'utilisateur. Or, la machine n'interprète que les commandes définies par le jeu de microinstructions. Le but est donc de passer du langage de haut niveau au niveau élémentaire compris par la machine. Après avoir défini les spécifications du langage évolué pour l'utilisateur puis celle des microinstructions, nous essaierons d'établir l'interface entre les deux niveaux.

III.1.1. Spécifications d'un langage de haut niveau pour l'utilisateur

Il ne s'agissait pas de réinventer un langage évolué. Bien que le choix définitif de ce langage ne soit pas encore établi, nous en avons retenu trois : "FORTRAN", "LTR" et "PASCAL".

Les algorithmes de l'application sont actuellement écrits en "FORTRAN". L'intérêt d'utiliser ce langage est donc d'éviter la réécriture des programmes. Seules quelques modifications seront nécessaires (suppression des réels, décomposition en modules peu dépendants, ajout des synchronisations...). Cependant, "FORTRAN" n'est pas adapté au temps réel ; de plus, ce n'est pas un langage "structuré".

"LTR" est par définition un "langage temps réel". Il est mieux adapté à notre problème. D'autre part, l'utilisateur a la possibilité de découper son application en "processus" qui se synchronisent sur événements. La déclaration de données globales (qui pourront être implantées en mémoire commune) est également possible.

D'autres particularités sont intéressantes dans ce langage : le traitement de listes, de chaînes de caractères, de chaînes de bits...

...

"PASCAL" est un langage structuré qui permet de mettre en évidence les aspects parallèles existant dans un traitement. Il tend à être de plus en plus utilisé, c'est donc son côté universel qui le favorise par rapport à "LTR".

Chacun des langages cités a ses avantages et ses inconvénients. Cependant, certaines caractéristiques communes nous permettent d'établir les spécifications générales du langage de haut niveau. D'autre part, des restrictions dues aux caractéristiques de la machine vont apparaître (pas de déclarations de réels, longueur des mots limitée à 16 bits...). Enfin, à côté des instructions "classiques", des primitives spécialisées vont être ajoutées au langage (primitives vectorielles par exemple), ceci afin d'accroître la vitesse d'exécution des applications.

III.1.1.1. Les instructions classiques

Nous allons recenser rapidement les types d'instructions que l'on retrouve quelque soit le langage choisi.

DECLARATIONS — Les types de variables autorisés sont :
- les entiers (simple ou double précision).
- les tableaux d'entiers (simples ou double précision).
- les booléens.
- les tableaux de booléens.

INSTRUCTIONS D'AFFECTATION — Elles permettent d'affecter à une variable le résultat d'une expression arithmétique ou d'une expression booléenne.

L'expression arithmétique comporte une suite de constantes, de variables et/ou de valeurs de fonctions arithmétiques séparées par des opérateurs arithmétiques.

Les opérateurs arithmétiques sont les 4 opérations classiques (+, -, *, /) et l'exponentiation.

...

L'expression booléenne comporte une suite de constantes, de variables et/ou de valeurs de fonctions booléennes séparées par des opérateurs booléens. Les opérandes peuvent également, dans certains cas, être des expressions arithmétiques.

Il existe 2 types d'opérateurs :
- les opérateurs de comparaison ($<$, $>$, $\leq$, $\geq$, =, $\neq$).
- les opérateurs logiques (non, et, ou, ou exclusif).

LES RUPTURES DE SEQUENCE

Les ruptures de séquence inconditionnelles (GØ, TØ) sont à proscrire dans le cas d'une programmation structurée. Elles sont cependant inévitables en FORTRAN.

Les ruptures de séquence conditionnelles sont de la forme :

```
si  < expression booléenne >
alors
début
    < séquence d'instructions 1 >
fin
sinon
début
    < séquence d'instructions 2 >
fin
```

INSTRUCTIONS D'ITERATION

Deux formes d'instructions d'itérations sont intéressantes. Leur forme générale est :

```
pour  < variable >  variant de  < expression arithmétique >  à
  < expression arithmétique >  par pas de  < expression arithmétique >
    faire  < séquence d'instructions >
```

```
  tant que  < expression booléenne >  faire  < séquence
                                               d'instructions >
```

...

LES PROCEDURES — Il en existe 2 sortes : les fonctions qui ne renvoient qu'un résultat et les sous-programmes classiques.

INSTRUCTIONS D'ENTREE/SORTIE — Des ordres de lecture et d'écriture sont nécessaires pour communiquer avec les périphériques.
Ces spécifications générales nous permettent de voir qu'il est possible de définir un langage intermédiaire proche du langage de haut niveau, mais indépendant de sa structure et sa syntaxe propre. Nous y reviendrons lors de la définition de l'interface langage évolué/langage machine.

III.1.1.2. Les primitives spécialisées

En plus des instructions classiques, l'utilisateur a la possibilité d'appeler des primitives spécialisées comme par exemple, le calcul d'un produit scalaire de deux vecteurs. Le but essentiel de la définition de telles primitives est d'améliorer les performances lors de l'exécution des applications.

En effet si à chaque primitive correspond un micro-programme spécialisé, il en résultera un gain de temps lors de l'exécution en temps réel d'une application.

D'autre part, une économie de place peut résulter de la rationalisation de l'écriture de ces primitives, en fonction des possibilités de la machine.

Enfin, certaines primitives sont impossibles ou trop lourdes à écrire en langage évolué (cas de la manipulation de bits en FORTRAN par exemple).

...

Des primitives vectorielles ont été définies pour notre application (voir partie I, chapitre 3). D'autres primitives spécialisées ont pu être dégagées, principalement à partir des algorithmes de reconnaissance de la parole.

Nous allons recenser les différentes primitives spécialisées. A chacune d'elles correspondra un micro-programme propre.

1) primitives vectorielles déjà définies

DEVEC (X,N1,Y,N2,P)	Décalage d'éléments d'un vecteur
VAVEC (X,N,VAL)	Initialisation d'un vecteur à une valeur
PROSC (X,N1,Y,N2,PSC)	Produit scalaire de 2 vecteurs
MAXM (X,N,RG)	Rang du terme maximal dans un vecteur
SOTAB (X,N1,Y,N2,Z,N3)	Somme de 2 tableaux terme à terme
MUTAB (X,N1,Y,N2,Z,N3)	Produit de 2 tableaux terme à terme
P2VEC (X,N,R)	Décalage de "R" bits de tous les éléments d'un vecteur
SCAVE (X,N1,Y,N2,SCA)	Produit d'un scalaire par un vecteur

2) primitives vectorielles supplémentaires

NINM (X,N,RG)	Rang du terme minimum dans un vecteur
NINMA (X,N,RG)	Rang du terme minimum en valeur absolue dans un vecteur
TRIC (X,N1,Y,N2)	Tri par ordre croissant des éléments d'un vecteur
TRICA (X,N1,Y,N2)	Tri par ordre croissant de la valeur absolue des éléments
TRID (X,N1,Y,N2)	Tri par ordre décroissant
TRIDA (X,N1,Y,N2)	Tri par ordre décroissant de la valeur absolue des éléments.
MOYN (X,N,MOY)	Moyenne des éléments.

...

3) primitives matricielles

PROMA — Produit matriciel

Paramètres

X	matrice 1
N1	Nombre de lignes de X
N2	Nombre de colonnes de X et de lignes de Y
Y	Matrice 2
N3	Nombre de colonnes de Y
Ƶ	Matrice produit
N4	Nombre de lignes de Ƶ
N5	NOmbre de colonnes de Ƶ

COIMA — Matrice de coincidence

Paramètres

X	vecteur 1
N1	nombre d'éléments de X
Y	vecteur 2
N2	Nombre d'éléments de Y
Ƶ	matrice de coincidence
VAL1	valeur identité
VAL2	valeur différence

4) traitement de listes

Des primitives pour le traitement des listes sont très utiles au système de reconnaissance de la parole.

Primitives portant sur l'ensemble de la liste :

CREATION d'une liste NL
DESTRUCTION d'une liste NL

Primitives portant sur les éléments de la liste :

AJOUT d'un nouvel élément en queue de liste
MODIFICATION d'un élément s'il répond à une certaine condition
RECHERCHE d'un élément
SUPPRESSION d'un élément
INSERTION d'un élément

Remarques : le langage "LTR" permet de définir et contrôler des ensembles chaînés comme les listes.

5) Fonctions sur les mots-machine

- racine carrée d'un nombre,
- valeur absolue d'un nombre,
- décalages arithmétiques et logiques d'un mot-machine,
- extraction d'un champ de bits dans un mot.

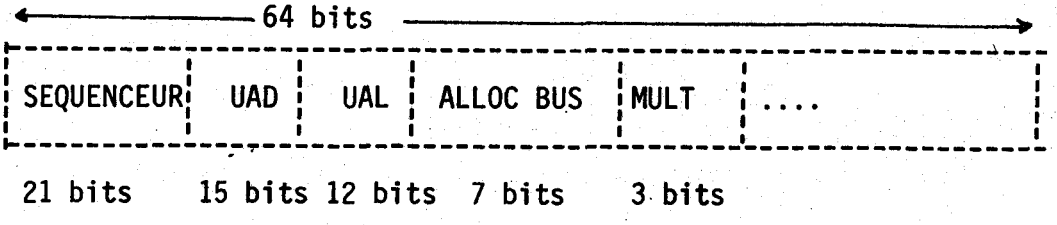
...

III.1.2. Les microinstructions

Ayant défini le niveau "supérieur", nous allons nous placer au niveau le plus élémentaire de la machine où la réalisation technologique ne permet d'interpréter qu'un langage d'instructions très simples. Par exemple, un microprocesseur de type AMD 2901 permet de réaliser des opérations comme le "et", le "ou" ou bien des décalages sur des ensembles de 4 bits.

La mise en oeuvre du "mécano" que constitue un processeur de la machine doit se faire grâce à une suite de commandes portant sur les différents éléments. Un jeu de microinstructions, permettant de générer ces diverses commandes, a donc été défini. Le format et le code de la plupart des commandes n'est pas arbitraire, puisqu'il dépend des spécifications mêmes de l'élément commandé. Ainsi, le "contrôleur de microprogrammes" AM 2910 possède un certain nombre de commandes codifiées qui doivent apparaître dans les microinstructions.

Une microinstruction comporte un certain nombre de champs distincts, chaque champ intervenant sur un élément différent du processeur.



Nous allons détailler chacun de ces champs et donner des exemples d'utilisation.

...

1) le champ "séquenceur"

Il permet de commander l'unité de séquencement du processeur.

Il comprend :

- un "code-commande" occupant 4 bits et permettant d'indiquer le type d'opération à effectuer dans le séquenceur (sauts conditionnels ou inconditionnels...).

- un "code-conditions" occupant 5 bits et permettant, par exemple, de définir la condition qui doit être vérifiée pour effectuer un saut conditionnel.

- une valeur d'adresse occupant 12 bits (valeur de 0 à 4095) et indiquant l'adresse de branchement lors d'un saut.

Exemple :

0111	00101	000001000000
CCM	CCD	ETIQ

CCM : "code commande": saut conditionnel...

CCD : "code condition" : ... si le résultat obtenu dans l'unité arithmétique et logique est différent de zéro...

ETIQ : ... saut à l'adresse microprogramme 64 si la condition est vérifiée.

...

2) le champ "UAD"

Il permet de commander l'unité d'adressage.

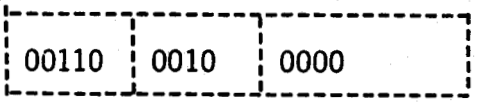
Il comprend principalement :

- un "code-commande" de 5 bits indiquant le type d'opérations à effectuer dans l'unité d'adressage (chargements de registres d'adresses, chargements avec incréments, indexations, ...).

- 2 zones de 4 bits permettent d'indiquer sur quel(s) registre(s) porte la commande. Il existe 16 registres.

Exemples :

(a)

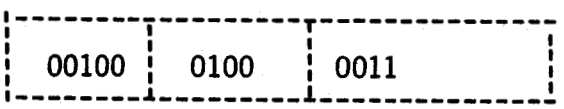


CCM : "code-commande" : charger le contenu du registre-adresse...

OP1 : ... dans le registre 2 de l'unité d'adressage (0010)

(ADR) ⟶ (registre 2)

(b)



CCM : charger dans le registre OP1, le contenu du registre d'adresse indexé par le contenu du registre OP2

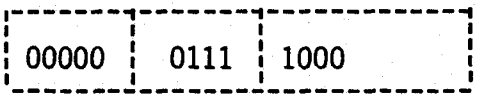
OP1 : registre 4.

OP2 : registre 3.

...

(ADR) + (registre 3) ⟶ (registre 4)

(c)



CCM OP1 OP2

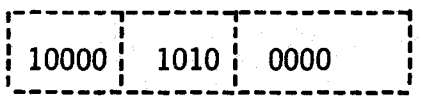
CCM : additionner les contenus des registres OP1 et OP2 et mettre le résultat dans OP1.

OP1 : registre 7.

OP2 : registre 8.

(registre 7) + (registre 8) ⟶ (registre 7)

(d)



CCM OP1 OP2

CCM : réaliser un "et logique" entre les registres OP1 et OP2, mettre le résultat dans OP1.

OP1 : registre 10.

OP2 : registre 0.

(registre 10) ∧ (registre 0) ⟶ (registre 10)

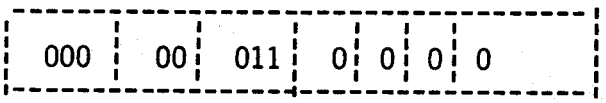
...

3) le champ "UAL"

Il permet de commander l'unité arithmétique et logique du processeur. Il comporte 12 bits. Les zones qui le composent ont des significations et des longueurs différentes suivant le type d'opérations à réaliser.

Exemples :

(a) Additions et soustractions



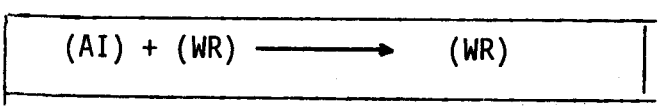
OPD : registre où sera rangé le résultat (WR).

OP1 : registre contenant l'opérande 1 (AI)

OP2 : registre contenant l'opérande 2 (WR)

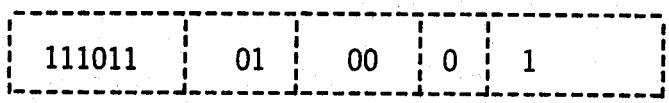
CP : les 2 bits indiquent respectivement s'il faut compléter les opérandes 1 et 2.

R : report.



...

(b) Décalages



CCM : "code-commande" de décalage.

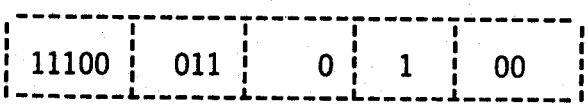
OP : registre à décaler (WR).

TD : type de décalage (arithmétique).

SD : sens de décalage (droite).

Décalage arithmétique à droite du contenu du registre WR

(c) Comparaisons



CCM : "code-commande" de comparaison.

OP1, OP2 : registres à comparer (WR et AI dans l'ensemble).

SC : sens comparaison (OP1 comparé à OP2).

Les résultats des comparaisons positionnent des drapeaux.

...

4) le Champ "ALLOC BUS"

Il permet de commander de préciser l'utilisation des bus internes du processeur et comporte 7 bits.

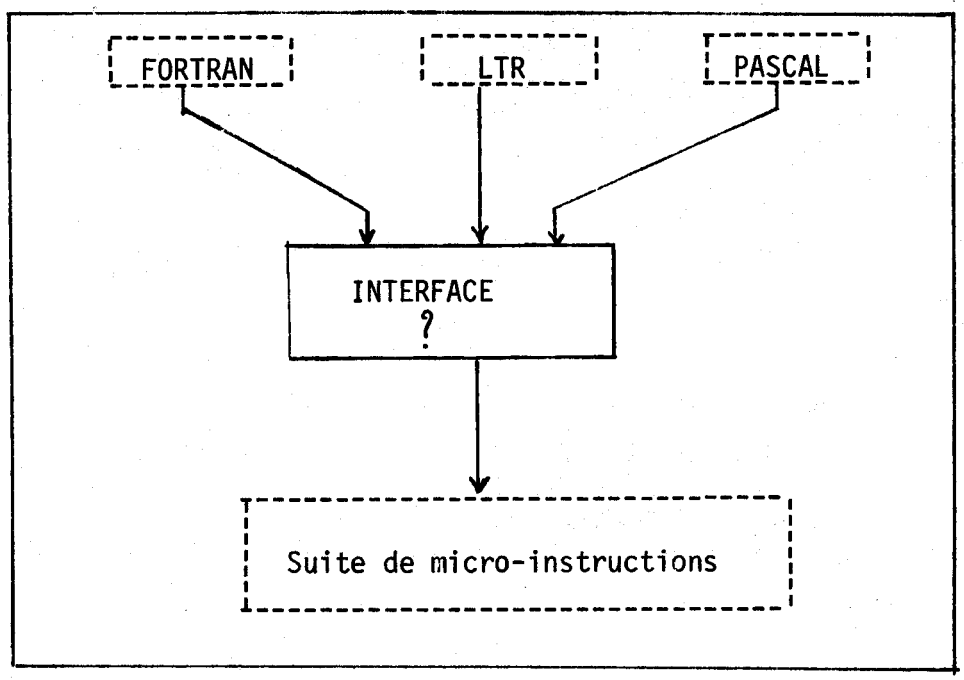
5) le champ "MULT"

Il permet de commander le multiplieur rapide du processeur.

III.1.3. L'interface : langage de haut niveau/langage machine

Un programme utilisateur, écrit dans un langage de haut niveau, doit se traduire au niveau de la machine par une suite de microinstructions.

Il s'agit donc de définir les niveaux intermédiaires pour atteindre ce but.



Pour que le système soit au maximum indépendant du langage évolué choisi, on se définit un langage intermédiaire.

III.1.3.1. Le langage intermédiaire

Un compilateur va traiter le programme écrit en langage de haut niveau et génèrera le langage intermédiaire.

A une instruction du langage évolué correspondra une ou plusieurs instructions en langage intermédiaire.

Cette phase de compilation et de traduction n'a pas besoin de se faire sur la machine. En effet, seule l'exécution d'une application en temps réel nous intéresse. La phase d'implantation de l'application peut se faire en différé et sur une machine classique (cf. figure 1).

. . .

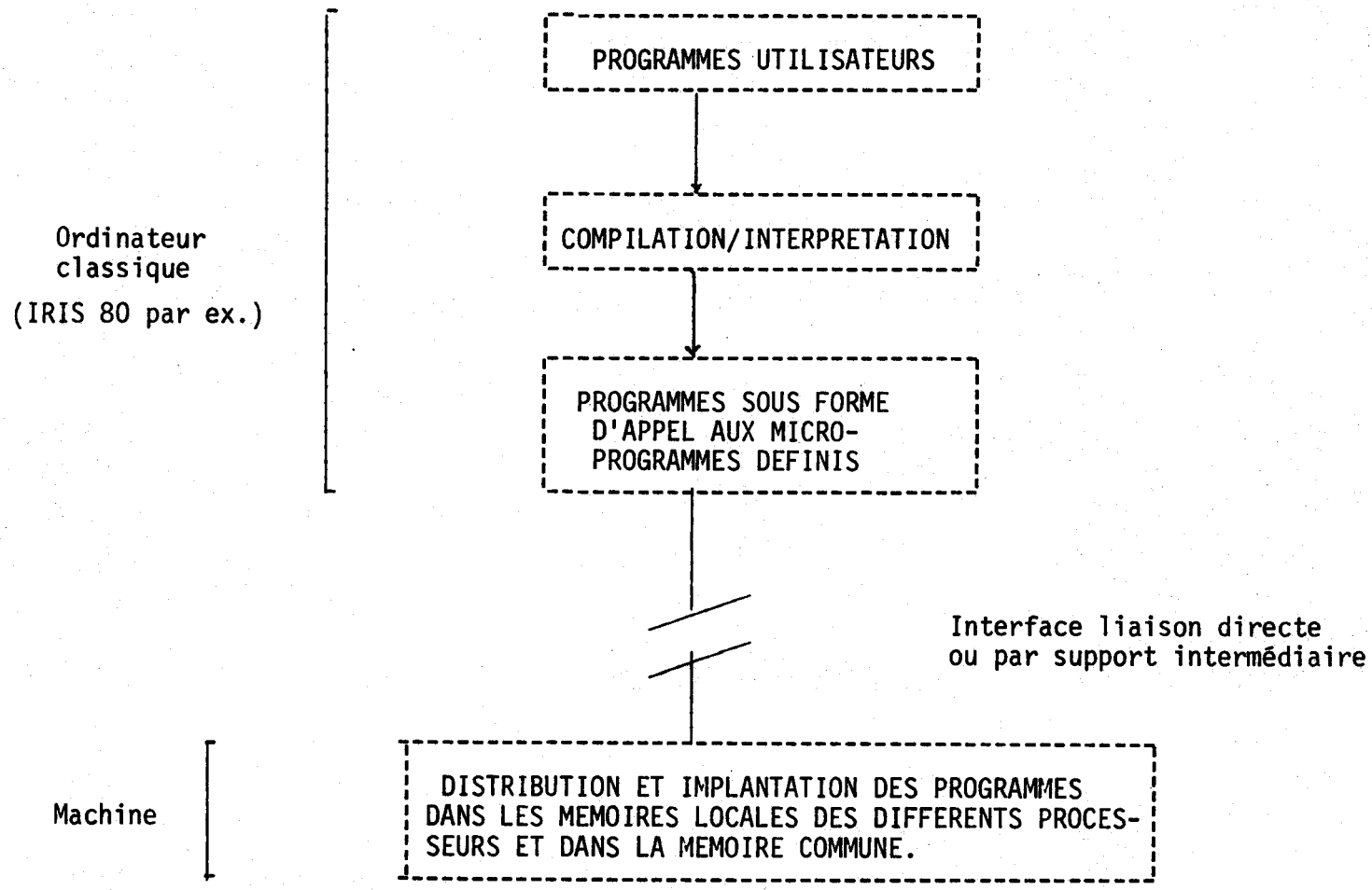
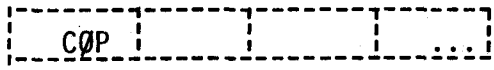


Figure 1 : phase d'implantation de l'application

...

Une instruction du langage intermédiaire est en fait un appel à un micro-programme défini dans les mémoires des micro-programmes de la machine.

Un programme, tel qu'il apparaît en mémoire propre d'un processeur, se compose d'une suite d'appels à des micro-programmes. Chaque instruction comporte un code opération permettant le branchement au micro-programme désiré et une suite de paramètres ou une indication d'adresses où sont rangés les paramètres.



paramètres

Illustrons par 2 exemples ce que nous venons d'exposer :

- soit une instruction en langage évolué de la forme

< variable > = < expression arithmétique >

Le compilateur va traduire cette instruction en 2 instructions intermédiaires.

1. Code opération : calcul d'une expression arithmétique.

   Paramètres : adresse où est rangée l'expression
   adresse résultat expression.

2. Code opération : affectation

   Paramètres : adresse résultat expression
   adresse de la variable.

Au niveau de la machine, l'instruction 1 correspondra à un branchement au micro-programme qui traite les expressions arithmétiques avec les paramètres indiqués ; l'instruction 2 correspondra à un branchement au micro-programme qui traite les instructions d'affectations.

...

- soit l'instruction conditionnelle de la forme :

si < expression booléenne > alors < séquence 1 > sinon < séquence 2 >

Elle sera traduite par :

1. Code opération : calcul expression booléenne.
   Paramètres : adresse début de l'expression
   adresse résultat de l'évaluation.

2. Code opération : instruction conditionnelle
   Paramètres : adresse résultat évaluation
   adresse début séquence 1
   adresse début séquence 2

III.1.3.2. Les microprogrammes

Un microprogramme se compose d'une suite de micro-instructions. L'exécution de cette suite d'instructions réalisera une fonction. Les fonctions sont soit du type général (ruptures de séquence conditionnelles ou inconditionnelles, traitement des boucles, calcul d'expressions...), soit de type spécialisé (micro-programme traitant les produits scalaires, les sommes de vecteurs).

Il est possible qu'à l'intérieur d'un micro-programme, il soit fait appel à un autre micro-programme.

La mémoire de micro-programmes comporte une partie non volatile et une partie volatile (voir chapitre II). Dans la mémoire non volatile, on trouvera les micro-programmes communs à toutes les applications.

Dans la mémoire volatile, l'utilisateur aura la possibilité d'inclure des micro-programmes spécifiques à son application ou à une classe de tâches de son application. Le chargement de ces micro-programmes se fera lors de la phase d'implantation de l'application dans la machine par un micro-programme spécialisé en mémoire non volatile.

...

Pour écrire et mettre au point les micro-programmes, un langage d'assemblage a été défini. Lorsqu'un micro-programme écrit en assembleur est au point, il est traduit en code-machine et rangé dans les mémoires de micro-programmes des différents processeurs.

III.1.3.3. Le langage d'assemblage

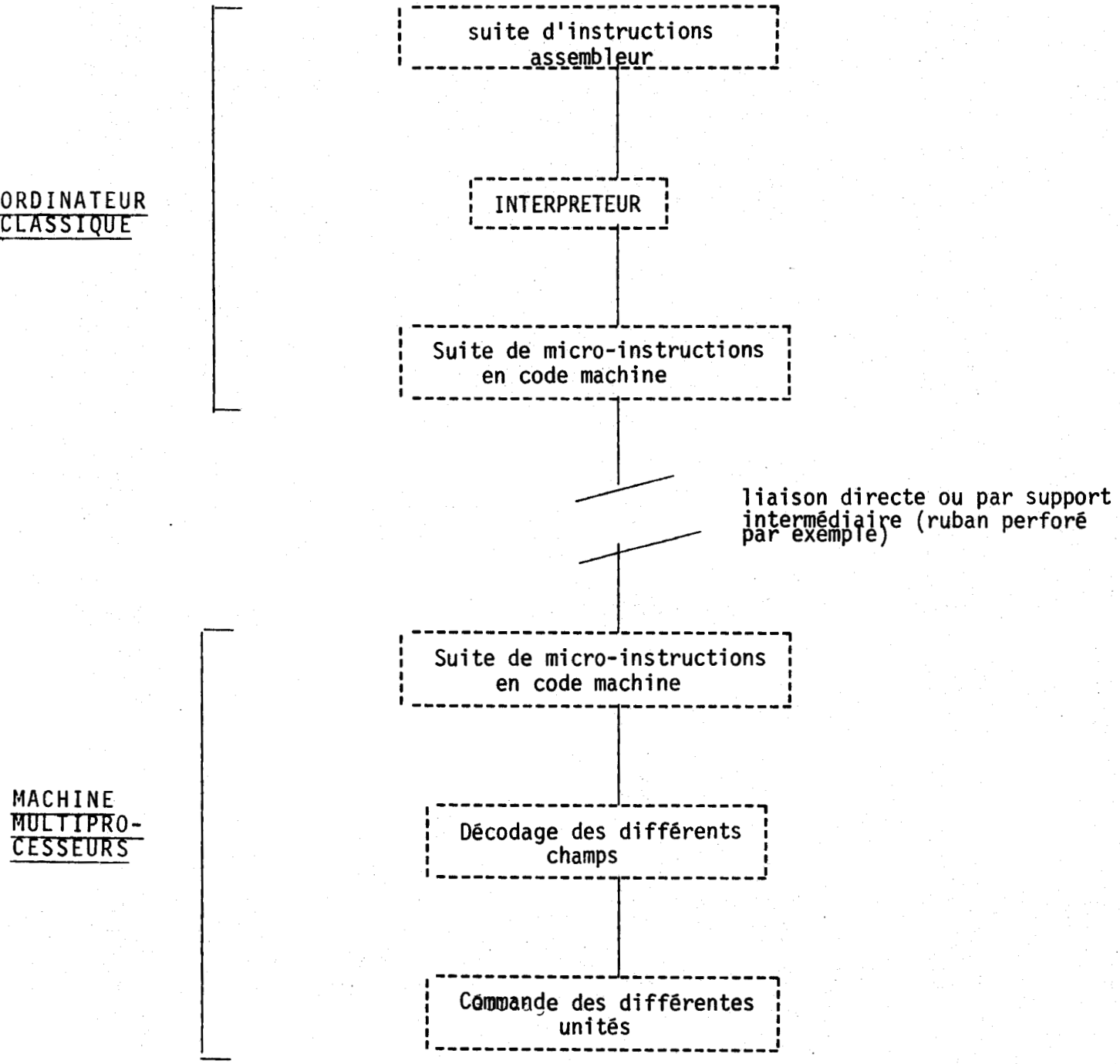
Il est évidemment exclu de programmer la machine en alignant une série de bits. Un langage d'assemblage, composé de codes mnémoniques, est nécessaire pour générer les micro-instructions.

Cet assemblage ne se fait pas sur la machine. On procède donc à un "assemblage croisé" (cross-assembler).

L'assembleur doit comporter :

- une désignation alphanumérique des codes opératoires.
- une désignation symbolique des données.
- des déclarations de données et de réservations mémoire.
- éventuellement, des pseudo-opérations traduites en appels de procédures standards.

Un interpréteur va traduire l'assembleur en code assimilable par la machine. (voir schéma page suivante).



L'interpréteur a été écrit en "Métasymbol" sur IRIS 80.

...

Chaque ligne d'assembleur correspond à un champ de micro-instruction. Quatre à cinq lignes d'assembleur décriront donc une micro-instruction (la cinquième ligne servant dans le cas où le multiplieur rapide doit être utilisé). Nous allons décrire par quelques exemples l'utilisation de l'assembleur.

1) description du champ séquenceur en assembleur

Forme générale :

```
Etiquette Ø CCM, CCD Ø ETIQ
```

Exemple :

```
JRP, NZC  64
```

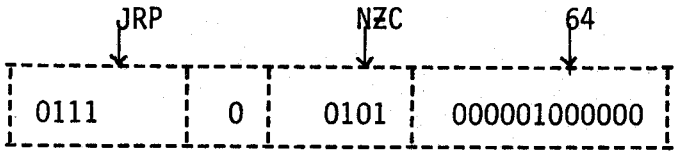
saut à la microinstruction d'adresse 64, si le résultat obtenu dans l'unité arithmétique et logique est différent de "zéro".

Cette instruction assembleur est interprétée (transformation du code mnémonique en code-machine) et génère la suite de bits du champ séquenceur.

JRP : code-commande de saut conditionnel.

NZC : code-condition si résultat UAL ≠ 0

64 : adresse de branchement



...

2) description du champ UAD en assembleur

Forme générale :

CCM ß ØP1,(ØP2)

Exemples :

a)

LD	AR2	
00110	0010	0000

"charger le contenu du registre-adresse dans le registre 2 de l'unité d'adressage".

b)

LDX	AR4 ,	AR3
00100	0100	0011

"charger le contenu du registre-adresse, indexé par le contenu du registre 3 dans le registre 4".

c)

ADD	AR7 ,	AR8
00000	0111	1000

"Additionner les contenus des registres 7 et 8 de l'unité d'adressage, mettre le résultat dans le registre 7.

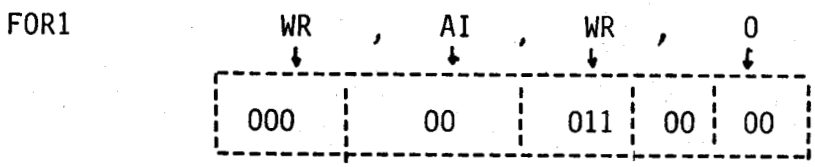
...

3) description du champ "UAL" en assembleur

Il n'existe pas de forme générale, mais 14 formes différentes suivant l'opération à effectuer.

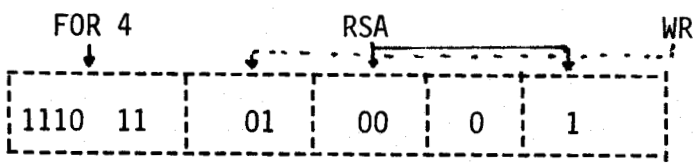
Exemples :

a) additions et soustractions



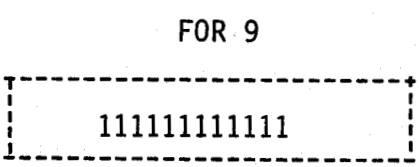
"additionner les contenus des registres AI et WR, mettre le résultat des WR".

b) décalages



"décalage arithmétique droite du contenu du registre WR".

c) pas d'opération dans l'unité d'adressage



...

Le langage d'assemblage a été entièrement défini pour toutes les commandes et opérations possibles.

Il est analysé syntaxiquement et traduit en code-machine.

...

III.2 / ETUDE DES COMMUNICATIONS ET SYNCHRONISATIONS

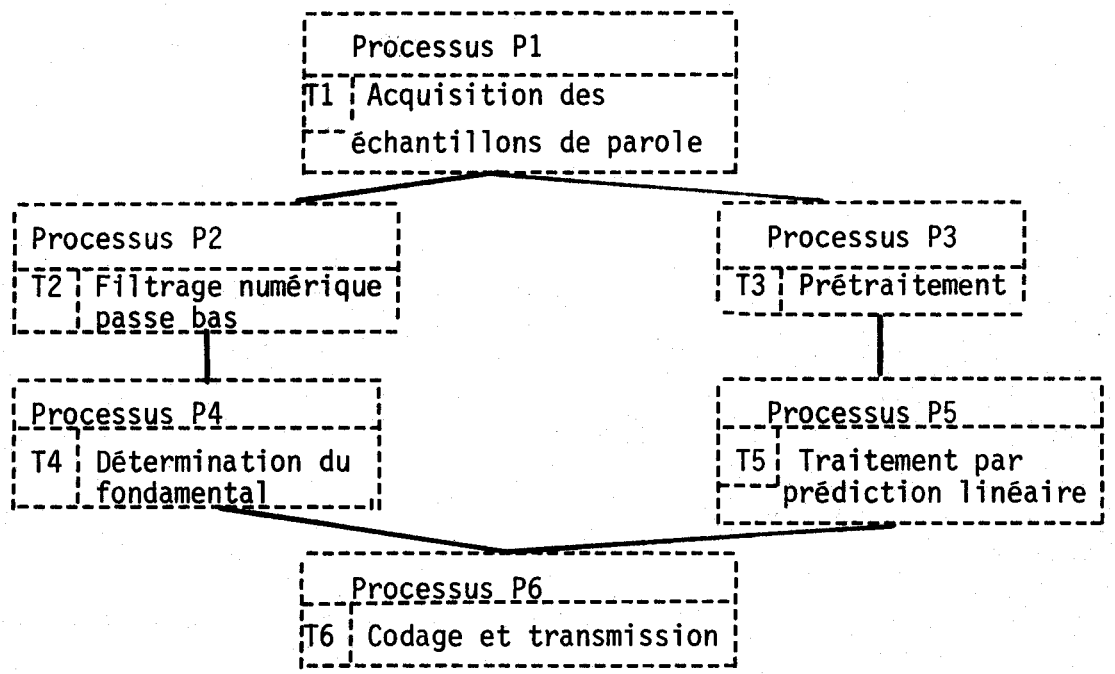
Les problèmes de communications et de synchronisations entre processeurs sont très importants dans un système multi-processeurs.

Notre but n'est pas de définir un système d'exploitation trop général et par là même trop complexe à gérer. Les propriétés de l'application à réaliser permettent de limiter les types de communications et de synchronisations.

III.2.1. Graphe descriptif de l'application

Quelle que soit l'application à traiter, il est possible de la représenter par un graphe où chaque noeud représente une tâche à effectuer.

Pour l'analyseur à prédiction linéaire, les décompositions de type "série" et de type "parallèle" apparaissent dans le graphe descriptif.



L'information qui est transmise d'un noeud du graphe au noeud suivant a un format connu. Ainsi T3 transmet à T4 un ensemble d'échantillons de parole prétraités.

La tâche globale a un caractère répétitif : l'analyse est faite périodiquement sur un ensemble d'échantillons de parole différent, et ceci tant qu'il y a de l'information à traiter. Ceci entraîne :

- a) le lancement sur évènement d'une analyse (sur horloge ou arrivée d'échantillons).
- b) la récupération de l'information traitée avec un débit égal en moyenne.
- c) la nécessité de marquer les flots de données échangés.

La chronologie des lancements d'analyses doit être respectée en sortie.

Le fonctionnement de l'ensemble peut être assimilé à une chaîne de montage : un processus Pj qui exécute une tâche répétitive Tj reçoit l'information d'un processus Pi, la traite, puis l'envoie vers le processus suivant Pr.

Tant que la chaîne est "alimentée", chaque processus boucle sur la tâche à effectuer.

On peut remarquer sur l'exemple de graphe donné précédemment que P1 a deux destinataires P2 et P3. P1 devra donc envoyer des informations aux deux successeurs. Par contre, P6 a deux "expéditeurs". Le processus P6 ne pourra exécuter la tâche T6 que lorsque deux prédécesseurs auront fourni leurs messages respectifs.

Nous reviendrons sur ces problèmes de synchronisations et de formats de messages.

Ayant défini un certain nombre de processus, il s'agit maintenant de leur allouer les processeurs physiques.

III.2.2. Allocation des processeurs aux processus

L'utilisateur connaît le nombre de processeurs dont il dispose. C'est à lui de définir au mieux l'implantation des processus pour avoir les meilleures performances en fonction des possibilités de la machine. C'est le caractère "temps réel" et la nécessité pour l'utilisateur d'optimiser la charge répartie qui conduisent à ce choix.

Une table d'implantation, en mémoire commune, indiquera le "potentiel dynamique" du système. Chaque processeur peut traiter un ou plusieurs processus. Les programmes correspondants à ce ou ces processus seront chargés en mémoire locale du ou des processus concerné (s) lors de la mise en place de l'application. Là encore apparaît le caractère spécifique du système pour lequel, compte tenu des performances visées, on ne peut admettre de perdre du temps pour charger un processus dans le site où il s'exécutera.

Certains cas extrêmes d'implantations peuvent se présenter :

a) il n'existe qu'un seul processeur. L'application, constituée de n processus s'exécute alors de la manière séquentielle sur l'unique processeur.

La table d'implantation est la suivante :

Processus	P1	P2... Pn
Processeurs PS	x	x ... x

Cette implantation est classique et ne présente pas d'intérêt pour notre application, les performances n'étant pas suffisantes.

b) une tâche quelconque est reproduite à m exemplaires dans les m processeurs. Un demande pour obtenir l'exécution d'un processus pouvant être adressé à plusieurs sites différents. Le tableau d'implantation est "plein".

Une telle implantation nécessite une gestion très complexe des allocations. En outre, chaque processeur aura en mémoire propre tous les programmes de l'application : les mémoires propres devront être de grande capacité. Par contre, la défaillance d'un processeur ne remettra pas en cause le bon déroulement d'une application, à condition qu'il soit possible de la détecter et de retirer le processeur en panne du tableau d'implantation.

c) il existe un processeur spécialisé par processus à effectuer. Ceci est possible si le nombre de processus n'est pas trop important.

Processus / Processeurs	P1	P2	Pn
PS1	1	0	0
PS2	0	1	0
PSn	0	0	1

Il y a dans ce cas n processus et n processeurs. Cette implantation a l'avantage de simplifier au maximum les problèmes d'allocation des processeurs aux processus, puisque cette allocation est fixe. Par contre, si les processus n'ont pas des temps d'exécution du même ordre, il en résulte un sous-emploi de certains processeurs. Dans le cas d'un traitement de type "pipe-line", la vitesse de l'ensemble va dépendre du processus le plus lent à s'exécuter.

Enfin, la panne d'un processeur est fatale au bon fonctionnement du sytème (contrairement au cas précédent).

Ce passage en revue de certains cas extrêmes nous permet de définir une implantation intermédiaire adaptée à notre application

Certains processeurs peuvent être spécialisés dans l'exécution d'une ou plusieurs tâches. Réciproquement, plusieurs processeurs peuvent être alloués à un même processus. Le choix de la meilleure implantation est laissée à l'utilisateur qui seul connaît la charge de chaque phase.

Par exemple, nous disposons de cinq processeurs pour traiter l'application d'analyse par prédiction linéaire, la répartition des différents processus est la suivante :

Processus / Processeurs	P1 Acquisition	P2 Filtrage	P3 Pré traitement	P4 Fonda-mental	P5 Traitement	P6 Codage/ transmission
PS1	X					X
PS2		X	X			
PS3					X	
PS4					X	
PS5				X		

Le processeur PS1 peut être alloué soit au processus P1, soit au processus P6. La mémoire propre du processeur PS1 contient les programmes correspondant aux deux processus.

Les processeurs PS3 et PS4 peuvent être alloués au processus P5 uniquement. Les programmes correspondants sont dupliqués dans les mémoires propres de PS3 et PS4. Le processus P5 exécute la tâche : "traitement par prédiction linéaire", c'est-à-dire, la phase de l'application qui nécessite le plus grand nombre de calcul. L'allocation possible de deux

processeurs permet d'éviter un "goulot d'étranglement" au niveau de ce processus lors de l'exécution. Cependant, un problème de synchronisation se pose. En effet, le temps d'exécution du processus est variable : si une occurence du processus P5 est lancée à l'instant t dans le processeur PS3 et l'occurence suivante à l'instant t' dans le processeur PS4, il est possible que le traitement dans PS3 se termine après le traitement dans PS4. Il faudra tenir compte de l'ordre dans lequel les occurences du processus ont été lancées :

Processus 5 : "traitement par prédiction linéaire"

occurence i. échantillons de parole : $s_1, \ldots, s_n$

Processus 3 t ———————— $t + \Delta t$

occurence i + t échantillons de parole : $s'_1, \ldots, s'_n$

Processus 4 t' ———————— $t' + \Delta t'$

L'information à transmettre au processus suivant (processus 6) est celle qui a été définie par l'occurence i du processus 5.

Ces considérations nous conduisent à étudier les moyens de communiquer l'information.

III.2.3. Les communications

Les communications se font par l'intermédiaire d'une boîte aux lettres en mémoire commune.

Chaque processus peut laisser un message à destination de n'importe quel autre (en fait, à destination de ses successeurs dans le graphe) et prendre les messages qui lui sont envoyés. Les messages comportent une adresse où trouver l'information en mémoire commune et le nombre d'éléments (de mots) d'information. Un indicateur donne l'état de la boîte aux lettres du processus considéré.

Chaque processus possède sa propre boîte aux lettres.

Le "centre d'échange des messages" prend la forme suivante pour notre application.

Processus	Processus émetteurs	Etat de la boîte	Adresse de début d'information	Nombre d'éléments d'information
P1		Occupée	ADR 1	200
P2	P1	Occupée	ADR 2	200
P3	P1	Vide		
P4	P2	Occupée	ADR 3	200
P5	P3	Vide		
P6	P4	Vide		
	P5	Occupée	ADR 4	12

L'état de la boîte est pris arbitrairement à un instant donné de l'exécution de l'application.

Le processus P6 possède deux cases distinctes dans sa boîte. En effet, il peut recevoir des messages de deux processus P4 et P5. P6 doit recevoir les deux messages pour pouvoir s'exécuter.

P4 détermine le fondamental, P5 calcule les coefficients de réflexion et le gain, P6 code et transmet l'ensemble.

III.2.4. Les synchronisations

Un premier moyen pour gérer le déroulement d'une application est de créer un "superviseur". Le superviseur contient le graphe de l'application à traiter.

Chaque processus, lorsqu'il termine une tâche, va faire appel au superviseur pour gérer la suite du traitement.

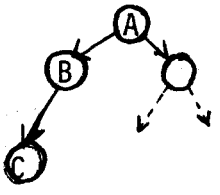
Le superviseur permet de décharger l'utilisateur de la tâche de gestion de son application. Cependant, une perte de temps trop importante résulterait d'une telle solution, surtout si, comme c'est le cas, nous visons une application temps réel. D'autre part, l'utilisateur connaît son application : il peut définir les liens de synchronisation les mieux adaptés pour son cas.

Le second moyen pour organiser le déroulement d'une application est une solution "programmée".

Dans le cas extrême, tous les processus sont actifs et bouclent en attente d'informations à traiter. Ce cas est à rejeter, car il suppose des accès trop nombreux et trop fréquents à la mémoire commune. En effet, un processus "sans travail" va constamment tester en mémoire commune l'état de sa boîte aux lettres.

Un cas plus réaliste, mais relativement difficile à mettre en oeuvre, consiste à lancer en fin de processus, le processus successeur et également le processus qui suit le successeur.

Soit l'exemple du graphe suivant :



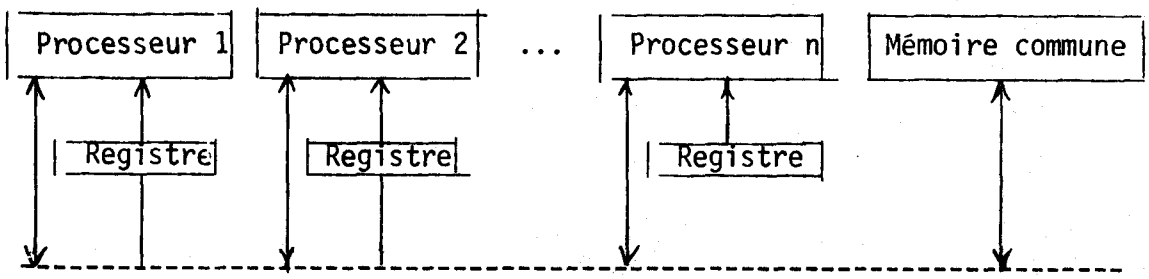
Lorsque A se termine, il lance B et lance C en attente

Un troisième moyen, dicté par certaines particularités de notre application, permet de définir une solution "locale" de gestion. C'est ce dernier moyen qui a été choisi. Nous allons le décrire en détail.

III.2.5. Travail d'un processeur

Lorsqu'un processus est alloué à un processeur, il le reste pour toute la durée de son exécution. En effet, il n'existe aucune interruption lorsque le processeur est "au travail". Par contre, lorsque le processeur est "au repos", il doit être possible de lui envoyer une information indiquant le travail à faire.

Dans le schéma général de la machine, repris de manière simplifiée dans la figure ci-dessous, on peut voir la présence de registres au niveau de chaque processeur. Le processeur au repos se mettra en attente active d'évènement sur le registre. Cette boucle sur le registre évite des accès nombreux à la mémoire commune.



C'est un des processeurs "actifs" qui va envoyer à tous les processeurs au repos cette information dans le registre. En l'occurence, ce sera l'identité du processus à prendre en charge. Le processeur, lorsqu'il est réveillé, suit trois phases de fonctionnement.

Dans une première phase, il acquiert l'information à traiter, s'il est concerné et si aucun autre processeur ne l'a devancé. Un mécanisme de "test and set" est nécessaire lors de l'accès concurrent de plusieurs processeurs à une boîte aux lettres.

Dans une seconde phase, il exécute le travail.

Dans une troisième phase, enfin, il transmet l'information traitée dans la boîte aux lettres du processus successeur. Il se remet ensuite au repos.

L'algorithme qui suit décrit de manière plus détaillée chacune des phases.

1. TEST : registre = o ?
   si oui : repos, retour en 1
   si non : le registre contient l'identité du processus à prendre.
   en charge : Pi, continuer en séquence.

2. Consultation de la table d'implantation.

3. TEST : le processeur peut-il prendre en charge le processus Pi ?
   (Possède-t-il les programmes correspondants à la tâche à effectuer en mémoire) ?
   si non : il n'est pas concerné ; registre = o ; retour en 1
   si oui : accès concurrent des processeurs concernés ; continuer en séquence.

4. Blocage de l'accès à la boîte aux lettres du processus Pi.

5. TEST : état de la boîte = vide ?
   si oui : un autre processeur a déjà pris le travail en charge ; débloquer l'accès à la boîte ; registre = o ; retour en 1.
   si non : prendre le message (adresse information + nombre d'éléments ; boîte = vide ; continuer en séquence).

6. Débloquer l'accès à la boîte aux lettres du processus Pi.

7. Exécution du travail : traitement du processus Pi.

8. L'information traitée à transmettre est mise en mémoire commune à l'adresse "ADR", le nombre d'éléments d'information est "NEL".

9. Recherche du (ou des) processus successeur Pj.

10. Blocage accès boîte aux lettres Pj.

11. TEST : boîte aux lettres vide ?
    si non : attendre qu'elle soit vide : débloquer accès boîte
    retour en 10.
    si oui : continuer en séquence.

12. Déposer information : ADR, NEL.

13. Boîte = pleine.

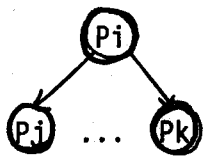
14. Débloquer accès boîte.

15. Réveil des processeurs : registre des processeurs = identité du processus Pj.

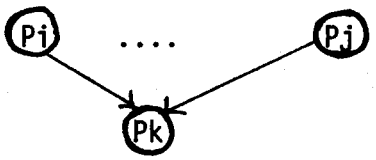
16. TEST : boîte aux lettres processus Pj = vide ?
    non : le processus n'a pas encore été pris en charge ;
    retour en 15.
    oui : le processus a été pris en charge, le processeur peut
    se mettre en repos ; retour en 1.

Cet algorithme ne tient pas compte d'un certain nombre d'éléments.

Dans le cas où il existe plusieurs successeurs à un processus Pi (f1), il faut alors transmettre l'information aux différents successeurs et le processeur doit attendre que chacun des processus successeurs ait été pris en charge pour se mettre au repos.



f1



f2

Le cas où plusieurs processus émetteurs doivent se synchroniser pour fournir une information au processus successeur (f2) n'a pas été traité. Nous avons vu lors de l'étude des communications (III.2.3.) que des cases supplémentaires étaient ajoutés dans une boîte aux lettres dans le cas où il existait plusieurs processus "émetteurs". Le processus Pk (f2) ne sera déclenché que si tous les messages lui sont parvenus.

Le problème, déjà posé, de l'ordre d'occurence des processus n'est pas résolu. Dans le cas de notre application, il faut respecter le principe du "premier entré, premier sorti". La première série d'échantillons de parole analysés doit bien entendu être la première à être transmise. L'association d'un numéro d'ordre à l'occurence d'un processus doit permettre de résoudre le problème.

Enfin, un processeur actif peut lui-même prendre en charge le processus successeur qu'il a sélectionné et donc se mettre en concurrence avec d'autres processeurs pour l'accès à la boîte aux lettres du processus concerné, sans s'être mis au repos entre temps.

L'algorithme fait ressortir la nécessité d'une exclusion mutuelle sur l'accès aux boîtes aux lettres. En effet, sans protection, plusieurs processeurs peuvent tester et modifier l'état de la boîte en même temps.

L'emploi d'une instruction spéciale de "test and set" (TAS) permet de protéger l'accès et la modification des boîtes.

Cette étude des communications et des synchronisations dans le système multi-processeurs n'est que l'ébauche d'une étude plus complète qui est en cours de développement. Nous avons tenté de cerner le problème et d'apporter une solution qui réponde au cahier des charges spécifiques de notre application. L'étude des mécanismes de synchronisation et de communication ébauchée ici a permis d'obtenir l'assurance que la configuration proposée et les compléments nécessaires pour supporter ces aspects sont effectivement adaptés. Ils permettent de garantir à l'utilisateur qu'il pourra utiliser à son gré un outil de communication et des primitives de synchronisation répondant à son problème.

Cette garantie constitue pour nous la dernière étape dans l'étude de l'architecture.

CONCLUSION

CONCLUSION

CONSIDERATIONS SUR L'ETUDE EFFECTUEE

Nous avons présenté dans ce mémoire les principales étapes conduisant au développement d'un système orienté vers l'analyse de la parole.

Pour cela, nous avons tenté de suivre les différentes phases d'une méthodologie générale s'appliquant à la conception de toute architecture orientée vers une application.

Notre travail a d'abord consisté à analyser l'application. Ainsi, l'observation des algorithmes de prédiction linéaire et de leur comportement dynamique en cours d'exécution a permis de dégager les spécifications générales du système à réaliser. Dans cette phase d'analyse, il est rapidement apparu que la contrainte "temps réel" représentait la principale difficulté à résoudre. En effet, un volume très important de calculs devait s'effectuer dans un laps de temps très court. C'est surtout cet aspect qui nous a orienté vers un système spécialisé.

La décomposition de l'application en modules faiblement dépendants ou indépendants , la mise en évidence des possibilités de parallélisme et l'étude des caractéristiques du flot d'information ont également contribué à la définition d'une architecture adaptée à nos besoins.

Notre choix s'est porté sur un système multiprocesseurs à base de microprocesseurs.

La machine définie possède un certain nombre de caractéristiques particulières :

. Elle utilise des microprocesseurs en tranches;

. Les différents processeurs qui la compose sont équivalents;

. Les processeurs grâce à des micro-programmes spécifiques, sont spécialisés pour le traitement de primitives vectorielles.

. Dans chaque processeur il est possible d'ajouter un multiplieur-accumulateur rapide.

. L'unité d'adressage et l'unité de traitement sont séparées.

Toutes ces caractéristiques visent principalement le même but: améliorer les performances du système.

Les problèmes qui se posent lors de la mise en oeuvre de systèmes multi-processeurs sont très complexes. Aussi, nous sommes-nous efforcés de les restreindre en simplifiant le logiciel de support, notamment les possibilités de communications dans le système, en fonction du cahier des charges de notre application.

APPLICATIONS / EXTENSIONS

L'étude présentée a permis d'examiner les problèmes posés par la conception d'une architecture adaptée. L'étude a d'abord porté sur l'application et nous a conduit à la définition d'un système de base adapté à l'analyse de la parole par prédiction linéaire.

Nous nous sommes ensuite placé sur un plan plus général . En effet, l'architecture choisie est suffisamment souple pour permettre de traiter des applications voisines ou complémentaires de l'application de base.

De nombreuses applications, comportant des tâches parallélisables et nécessitant un volume important de calculs, peuvent être traitées par la machine.

En particulier,les applications temps réel de traitement du signal s'adaptent bien à l'architecture de la machine (Calculs de filtres numériques,de F.F.T. ...) .

Dans le prolongement de l'analyseur à prédiction linéaire,il est possible d'implanter dans la machine les algorithmes de "synthèse" ,afin de réaliser un vocodeur à prédiction linéaire complet.

Enfin,une extension de la machine (par augmentation du nombre de processeurs) permettrait d'implanter le système de reconnaissance de la parole continue,en cours de développement au C.N.E.T. . Ce système fonctionne actuellement sur un ordinateur de type "10070" en "pseudo" temps réel;c'est à dire que les délais entre l'acquisition d'un message et sa compréhension sont relativement longs.la nécessité d'une machine adaptée permettant vraiment le temps réel,s'impose donc à plus ou moins long terme. Lors de la conception de la machine multi-processeurs,nous n'avons pas perdu de vue cette possibilité d'extension .

Le processus de reconnaissance de la parole peut se décomposer en un certain nombre de modules indépendants qui communiquent entre eux suivant un mode "pipe-line".

Les principaux modules sont actuellement:

.Un analyseur phonétique,qui permet de passer du signal vocal codé (par prédiction linéaire ou par un vocodeur à canaux) à sa représentation phonétique.

.Un analyseur lexical,permettant de comparer les mots phonétiques définis avec le vocabulaire existant;

.Un analyseur syntaxique,qui recherche la phrase syntaxiquement correcte parmi les séquences de mots possibles.

. Eventuellement,un analyseur sémantique .

Il est envisageable de spécialiser un processeur pour chaque module. L'ébauche de système de communications présenté dans le dernier chapître restera valable.

ETAT DE DEVELOPPEMENT

Un processeur de la machine est en cours de réalisation . Lorsqu'il sera au point,il suffira de le dupliquer à "n" exemplaires.

Le langage d'assemblage,permettant la génération des micro-instructions, a été entièrement défini.

Un jeu de micro-programmes de base est en cours de définition. Ces microprogrammes seront implantés dans les mémoires de micro-programmes de chaque processeur.

Les réflexions sur le langage de haut niveau et sur la formalisation du système de communications se poursuivent actuellement.

"L'analyseur à prédiction linéaire" sera la première application implantée dans la machine. Ceci va permettre de vérifier le bon fonctionnement du système avant de s'engager dans l'implantation d'autres applications.

BIBLIOGRAPHIE

B I B L I O G R A P H I E

ANNEES		
1957	KUNTZ KS	"Numerical analysis" Mac Graw-Hill New-York
1960	FANT G	"Acoustic theory of speech production" Mouton and Co the Hague
1965	FLANAGAN JL	"Speech analysis synthesis and perception" Academic New-York p. 119
1966	FLYNN MJ	"Very high speed computing systems" Proceedings of IEEE vol. 54 p. 1901 - 1909 (December)
1967	GOLD B. RADER CM	"The Channel Vocoder" IEEE trans. AU-15 p. 148 (December)
	NOLL AM	"Cepstral pitch determination" JASA Vol. 41 p. 293-309
1968	BARNES G et AL	"The Illiac IV computer" IEEE trans. C-17 p. 746-757 (August)
1969	GOLD B	"Parrallel processing techniques for estimating pitch periods of speech..." JASA Vol. 46 p. 442-448
1970	SCHAFFER RW RABINER LR	"System for automatic analysis of voiced speech" JASA Vol. 47 n° 2 p. 634
	GRAHAM WR	"The parallel and the pipe-lines computers" DATAMATION p. 68-71 (April)
1971	ATAL BS HANAUER SL	"Speech analysis and synthesis by Linear Prediction of the speech wave" JASA Vol. 50 n° 2 p. 637-655
	ITAKURA F	"Digital filtering techniques for speech analysis and synthesis" 7 th congress on acoustics Budapest 25 C 1 p. 637-655
	MARKEL JD	"FFT pruning" IEEE trans. AU-19 p. 305-311

...

1971	GOLD B and Al	"The FDP, a fast programmable Signal processor" IEEE trans. C-20 p. 33-38 (January)
	TIEN CHI CHEN	"Parallelism, Pipelining and Computer efficiency" COMPUTER DESIGN p. 69-74 (January)
1972	MARKEL JD	"The sift algorithm for fondamental frequency estimation" IEEE trans. AU-20 p. 367-377 (December)
	FLYNN MJ	"Some computer organizations and their effectiveness" IEEE trans. C-21 p. 948-960 (September)
1973	EL MALLAWANY I et Al	"Méthode de codage en vue de l'analyse et de la synthèse de la parole" CONGRES AFCET RENNES p. 61
	MARKEL JD GRAY AH	"On autocorrelation equations as applied to speech analysis" IEEE trans. AU-21 p. 69-79 (April)
	ROSENFIELD JL VILLANI RD	"An approach to multiprocessing at the level of very small tasks" IEEE trans. C-22 p. 149-153 (February)
1974	ANCEAU F	"Contribution à l'étude des systèmes hiérarchisés de ressources dans l'architecture des machines informatiques" Thèse, Université Scientifique et Médicale GRENOBLE
	MARKEL JD GRAY AH	"A linear prediction vocoder simulation based upon the autocorrelation method" IEEE trans. ASSP-22 p. 124-134 (April)
	MARKEL JD GRAY AH	"Fixed point truncation arithmetic implementation of a linear prédiction autocorrelation vocoder" IEEE trans. ASSP-22 p. 273-282 (August)
	ROSS et Al	"Average magnitude difference function pitch extraction" IEEE trans. ASSP-22 p. 353-362
	CAELEN G	"Un modèle mathématique de cochlée et son application à l'analyse du signal vocal" Thèse de Docteur-Ingénieur TOULOUSE
	REYLING G	"Performance and control of multiple microprocessors systems" COMPUTER DESIGN p. 81-86 (March)
	DENNIS JB MISUNAS DP	"A computer architecture for highly parallel signal processing" Proceedings of ACM SAN DIEGO p. 402-409 (November)

...

1975	EL MALLAWANY I	"Contribution aux recherches sur la Communication parlée" Thèse du Docteur-Ingénieur GRENOBLE
	BOURGENOT DECHAUX C	"Codage de la parole à faible débit : le vocodeur Ciphon" Revue technique Thomson-CSF Vol. 7 n° 4 (Décembre)
	MILLER RL	"Pitch detection by data reduction" IEEE trans. ASSP-23 p. 72-79
	MAKHOUL JI	"Linear prediction : A Tutorial Review" Proceedings of IEEE Vol. 63 p. 561-580
	VISWANATHAN R MAKHOUL JI	"Quantization properties of transmissions parameters in linear predictive systems" IEEE trans. ASSP-23 n° 3 p. 309-321
	RABINER LR GOLD B	"Theory and application of digital signal processing" Englewood Cliffs, Prentice Hall 1975
	BLANKENSHIP PE et Al	"The lincoln digital voice terminal system" Technical note MIT (August)
	KNUDSEN M.	"Real-Time linear predictive coding of speech on the SPS 41 triple microprocessor..." IEEE trans. ASSP-23 p. 140-145 (February)
	DE MORI R et Al	"Special purpose computer for signal processing" IEEE trans. C-24 p. 1202-1211 (December)
1976	DUBNOWSKI JJ SCHAFER RW RABINER LR	"A real-time digital Hardware pitch detector" IEEE trans ASSP-24 p. 1-8
	LE ROUX J	"Optimisation du calcul des coefficients de corrélation partielle" 7ème journée d'étude sur la parole NANCY (Mai)
	MARKEL JD GRAY AH	"Linear prediction of speech" Communication and cybernetics 12 Springer-Verlag
	BAER JL	"Multiprocessing systems" IEEE trans. C-25 p. 1271-1277 (December)
	BOWRA JW TORNG HC	"The modeling and design of multiple fonction unit processors" IEEE trans. C-25 p. 210-221 (March)
	SAYRE G	"Staran : an associative approach to multiprocessor architecture" RAIRO vol. 10 n° 5 p. 59-76 (mai)
	MAZARE G	"MCS : a symetric multi-microprocessor system" 2d symposium on micro-architecture EUROMICRO conference (October)

...

1977	EL MALLAWANY I GILLET F SAMUELSON C	"Vocoder à codage prédictif linéaire" Fiche technique CNET DAS/SST/2
	MAKHOUL JI	"Stable efficient lattice method for linear prediction" IEEE trans. ASSP-25 p. 423-428
	GOLD B	"Digital speech networks" proceedings of IEEE vol. 65 p. 1636-1658
	MORF M et Al	"Efficient solution of covariance equations for linear prediction" IEEE trans. ASSP-25 p. 429-433
	WEISSBERGER AJ	"Analysis of multiple microprocessor system architecture" COMPUTER DESIGN p. 151-163 (June)
	JACKSON H	"Synchronization problems in multiprocessor systems" EUROMICRO CONFERENCE session F2 p. 158-167
	CAPRANI O JENSEN KH OUGAARD U	"Microprocessor connected to a common memory" EUROMICRO CONFERENCE Session F2 p. 175-181
	NOVIC N COUTURIER GW	"Inter processor communication in systems with distributed control" Proceedings of IEEE vol. 65 p. 1323-1329 (September)
	RUSSO PM	"Interprocessor communication for multi-microcomputer systems" COMPUTER p. 67-76 (April)
1978	HATON JP PERENNOU G	"Reconnaissance automatique de la parole" 8e école d'été informatique de l'AFCET NAMUR (juillet)
	FULLER SH et al	"Multiprocessors : an overview and working example" proceeding of IEEE vol. 66 p. 216-228 (February)
	JONES A et Al	"Programming issues raised by a multiprocessor" Proceeding of IEE Vol. 66 p. 229-237 (February)
	MORRIS LR MUDGE JC	"Speed enhancement of digital signal processing software via microprogramming a general purpose minicomputer" IEEE trans. ASSP-26 p. 135-140 (April)
	ANCEAU F	"Principes et mise en oeuvre des microprocesseurs" ENSIMAG GRENOBLE (janvier)

...

1978/79	SAMUELSON C GILLET F	"Etude d'une machine multi-processeurs orientée vers l'analyse et la reconnaissance de la parole" article pour la 1ère conférence européenne sur le calcul parallèle et distribué. TOULOUSE A paraître en février 1979.
	SAMUELSON C	"Analyseur numérique de la parole par prédiction linéaire" RECHERCHES ACOUSTIQUES CNET Vol. 5 1978 (à paraître)

NOTICES TECHNIQUES DE CONSTRUCTEURS :

AMD - TEXAS INSTRUMENTS - TRW

ANNEXES

LISTE DES PHONEMES DU FRANCAIS (suite).

TYPE	REPRESENTATION	MOT-CLE	VOISEMENT
Consonnes Plosives	*p*	pour	non-voisé
	t	tête	non-voisé
	k	carrée	non-voisé
	b	baril	voisé
	d	de	voisé
	g	grêle	voisé
Consonnes Nasales	*m*	merci	voisé
	n	nous	voisé
	ɲ	gagnons	voisé
Consonnes fricatives	*v*	va	voisé
	z	jaser	voisé
	ʒ	jeudi	voisé
	f	faute	non-voisé
	s	simuler	non-voisé
	ʃ	chaton	non-voisé
Consonnes liquides	*r*	rude	voisé
	l	lutte	voisé

ANNEXE I .

LISTE DES PHONEMES DU FRANCAIS .

TYPE	REPRESENTATION	MOT-CLE	VOISEMENT
Voyelles Orales	*a*	mâle	voisé
	a	gratte	voisé
	ɛ	mais	voisé
	e	péril	voisé
	ø	nœud	voisé
	œ	sœur	voisé
	ə	le	voisé
	i	pi	voisé
	ɔ	colle	voisé
	o	pôle	voisé
	y	vu	voisé
	u	poule	voisé
Voyelles nasales	ã	maman	voisé
	ɛ̃	main	voisé
	œ̃	brun	voisé
	ɔ̃	long	voisé
Semi - Voyelles	*j*	maillot	voisé
	ɥ	puit	voisé
	w	ouistiti	voisé