Microprogrammation d'une fonction de corrélation avec sommation des échantillons sur le corrélateur d'Eiscat

Régis Gras

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE AGREE
DE GRENOBLE (C.U.E.F.A)

MEMOIRE

présenté en vue d'obtenir

le diplôme d'ingénieur

en

informatique

par

Régis GRAS

MICROPROGRAMMATION D'UNE FONCTION DE CORRÉLATION AVEC SOMMATION DES ÉCHANTILLONS SUR LE CORRÉLATEUR D'EISCAT

SOUTENU LE : 25 JUIN 1982

JURY

Président : Professeurs L. BOLLIET & F.H. RAYMOND

Membres : F. BERTIN

W. KOFMAN

Avant de commencer la présentation du sujet, je tiens à remercier Messieurs les Professeurs BOLLIET et RAYMOND qui ont bien voulu me faire l'honneur de présider ce Jury.

J'exprime également ma gratitude à Monsieur W. KOFMAN pour l'aide constante qu'il m'a apportée pour toutes les applications concernant le corrélateur EISCAT et en particulier pour la mise au point de celle que je présente aujourd'hui.

Je n'oublierais pas non plus Monsieur BERTIN qui est venu du CNET à cette occasion et qui a participé à la vérification des premiers résultats expérimentaux à Tromsö.

Enfin, j'exprime mes remerciements à Monsieur LACOUME, Directeur du CEPHAG, qui m'a facilité la préparation de ce mémoire dans le cadre du laboratoire, ainsi qu'à Madame MOREY qui a tapé ce mémoire en un temps record.

SOMMAIRE

2 - PROGRAMMATION SUR LE CORRELATEUR

2-1 Programmation du sous-programme de corrélation avec addition d'échantillons

2-1-1 Présentation des variables du programme

2-1-1-1 Les variables de l'APB-STACK

2-1-1-2 Les variables de l'APM-STACK

2-1-1-3 Autres initialisations

2-1-2 Commentaires sur l'organigramme du programme

2-1-2-1 Les PRO-insturctions

2-1-2-2 Les APB-instructions

2-2 Description du programme complet de l'application

2-3 Tests du programme

2-3-1 Tests graphiques

2-3-2 Tests arithmétiques

2-3-2-1 Addition de 2 échantillons

2-3-2-2 Addition de 3 échantillons

CONCLUSION

ANNEXES

-Structure du code intermédiaire

-Exemple de code intermédiaire

-Algorithme et code du sous-programme

-Code du programme complet

-Résultats graphiques

-Programmes de calculs numériques et résultats

BIBLIOGRAPHIE

INTRODUCTION

Le travail qui est présenté ici a pour cadre l'association scientifique EISCAT (European Incoherent Scatter).

EISCAT a été constitué par un accord international signé en décembre 1975 par l'Allemagne Féférale, la Finlande, la France, la Norvège, le Royaume Uni et la Suède.

Les installations d'EISCAT permettent à la communauté scientifique des pays membres de l'association d'étudier un certain nombre de problèmes relatifs à la haute atmosphère en zone aurorale.

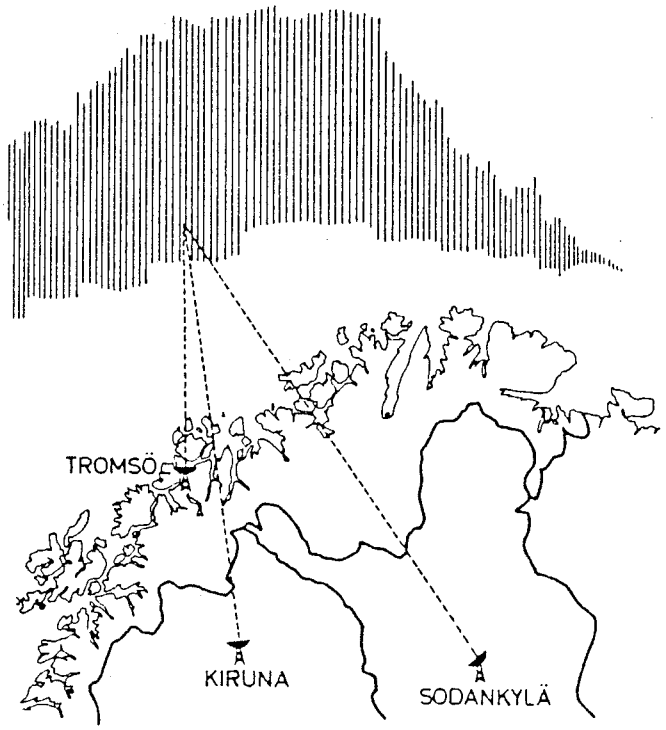
Cette étude est intéresante à plusieurs points de vues. Elle relève de la recherche fondamentale, et dans le même temps elle contribue à la connaissance de la haute atmosphère, de l'astronomie, de la physique des plasmas. En particulier, dans ce domaine, elle permet d'observer des phénomènes qu'il serait très difficile, voir impossible de reproduire en laboratoire.

Différentes applications scientifiques doivent découler de l'étude de la haute atmosphère. Par exemple, en météorologie une amélioration de la prévision du temps grâce à une meilleure connaissance de la circulation atmosphérique et de la répartition energétique ; par exemple, en propagation des ondes radio-électriques, grâce à une meilleure compréhension de l'apparition et des effets des perturbations solaires.

Pour l'étude de la haute atmosphère, la région aurorale a une importance particulière. En effet, c'est dans cette zone que les lignes du champ magnétique terrestre se referment. Ces lignes de force jouent un rôle de guide pour toutes les particules à haute énergie issues des perturbations solaires, ce qui fait qui l'ionosphère aurorale y est beaucoup plus perturbée que dans le reste du monde.

Ces raisons font des régions aurorales des lieux privilégiés pour l'étude de la haute atmosphère.

EISCAT comprend 3 stations d'observation qui sont géographiquement ainsi réparties :



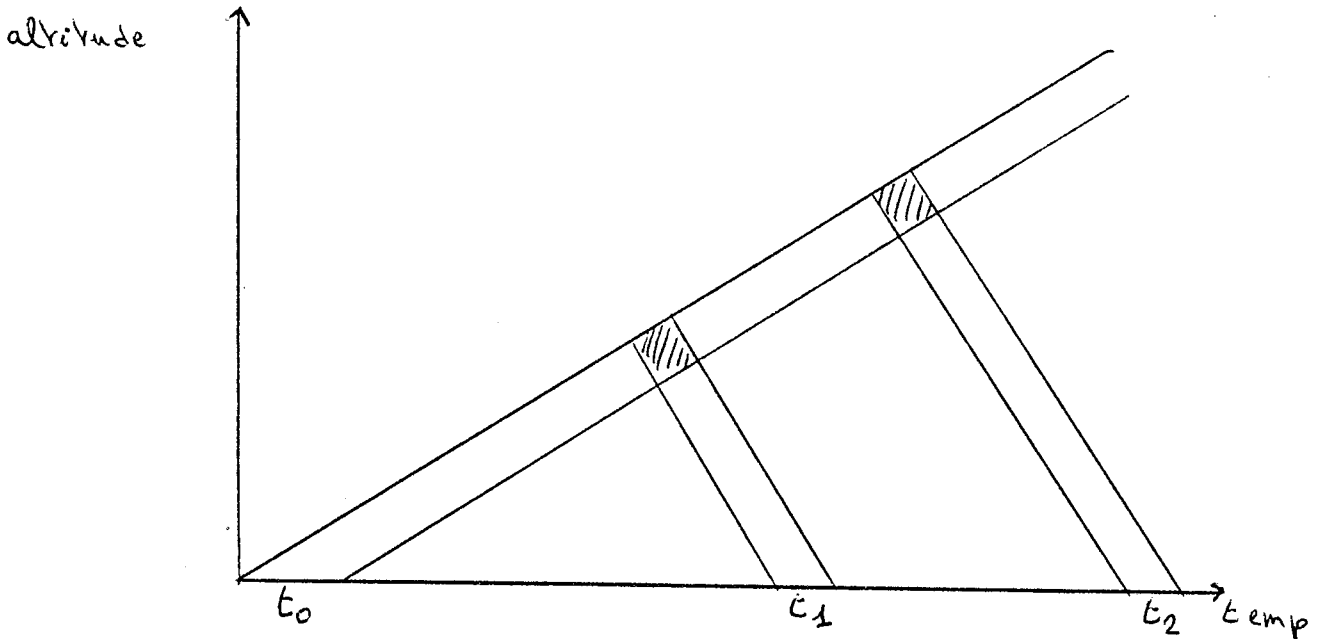
-TROMSÖ en NORVEGE
-KIRUNA en SUEDE
-SODANKYLA en FINLANDE

Principe physique des mesures effectuées à EISCAT.

L'étude des différents paramètres de l'ionosphère que l'on veut mesurer (densité électronique, température ionique, vitesse des ions, composition des ions), est basée sur la rétrodiffusion d'un signal par les électrons libres de la haute atmosphère. Ce principe physique est mis en oeuvre de la manière suivante. A un instant donné, un radar émet un pulse

(de 10 microsec. à 10 millisec.) à très haute fréquence vers l'ionosphère. Ce signal rétrodiffusé est reçu par les 3 stations de Tromsö, Kiruna et Sodankyla. Ce signal reçu au sol sera ensuite retraité numériquement pour en extraire les différents paramètres ionosphériques que l'on veut mesurer. Cependant, la masse de données reçue est telle, qu'il est procédé à une réduction de données en temps réel. Ce prétraitement en temps réel est réalisé par un corrélateur qui calcule et accumule en temps réel des fonctions d'autocorrélation sur le signal reçu. Celles-ci contiennent toute l'information portée par le signal reçu.

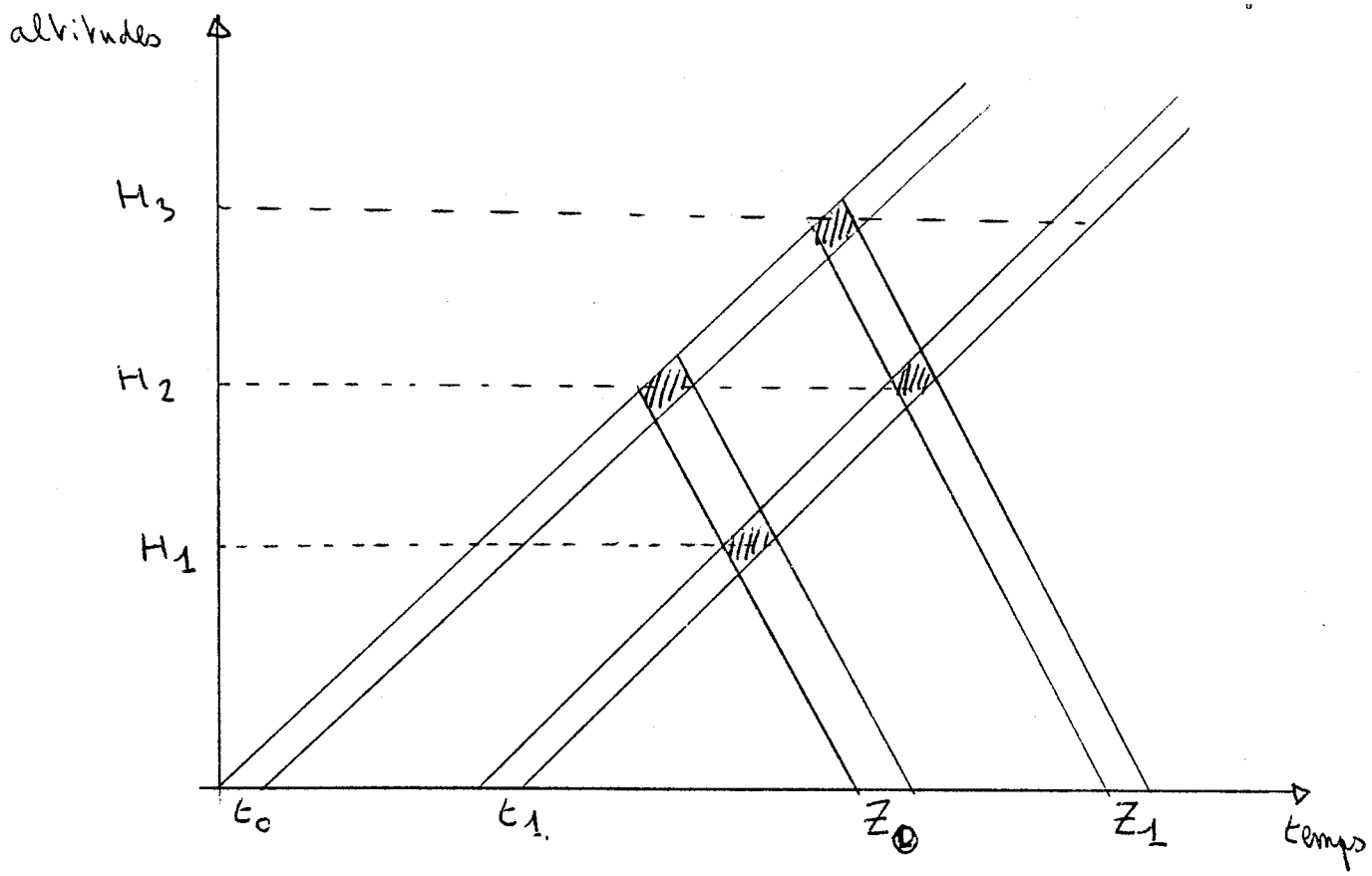
Ce principe est résumé par le schèma suivant :



soit une impulsion émise à un temps t_o. Le signal rétrodiffusé est échantilloné à des temps t_1, t_2 ... t .

Ces échantillons, reçus à des temps différents sont caractéristiques de l'altitude d'où ils proviennent.

A une succession d'impulsions, correspond une succession de réceptions provenant de différentes altitudes. Mais seuls les échantillons provenant de la même altitude sont corrélés entre eux.



Le signal reçu en Z_o provient des altitudes H_1 et H_2. De même, celui reçu en Z_1 provient des altitudes H_2 et H_3

$$Z_o = Z_o{}^{H_1} + Z_o{}^{H_2}$$
$$Z_1 = Z_1{}^{H_2} + Z_2{}^{H_3}$$

Si on calcule une fonction de corrélation sur les signaux reçus, dans le produit $Z_o\ Z_1$ on a $Z_o{}^{H_2} \cdot Z_o{}^{H_2}$ + la somme des produits mixtes qui en moyenne tendent vers zèro car il ne sont pas corrélés.

Présentation du matériel d'EISCAT

Le matériel implanté dans le cadre d'EISCAT consiste en deux systèmes de radar. Un radar tristatique UHF (ultra higth frequency) opérant à des fréquences voisines de 960 MHz et un radar monostatique VHF (very hight frequency) opérant à des fréquences voisines de 240 MHz. Ces deux systèmes sont complétement indépendants et actuellement, seul le système UHF est opérationnel.

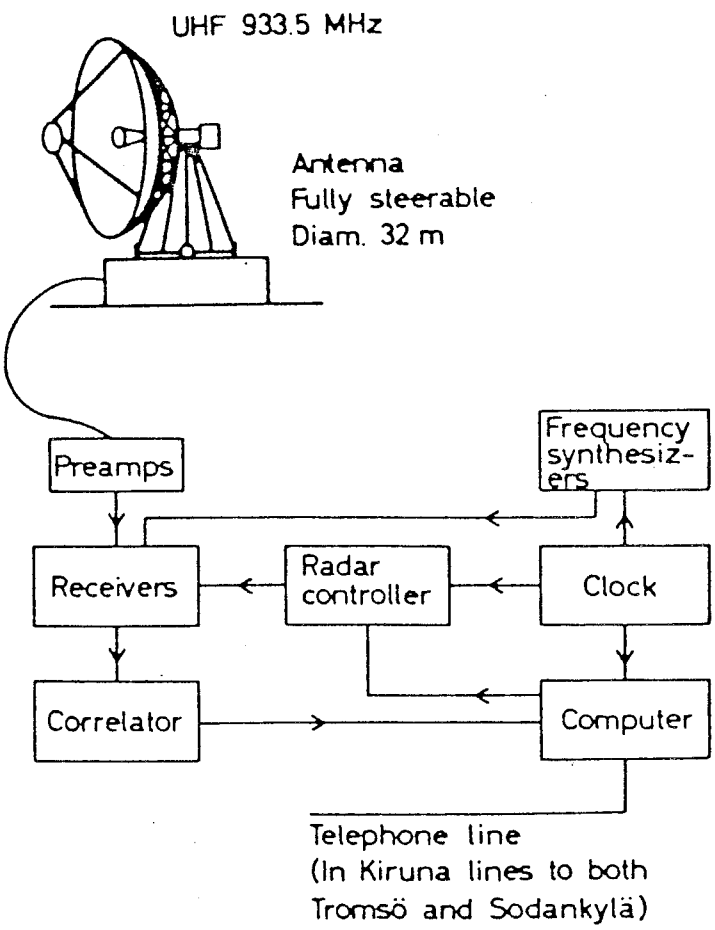
Le système VHF est implanté à Tromsö seulement, tandis que le système UHF est implanté à Tromsö (émission/réception), à Kiruna et à Sodankila (réception seule). L'intérêt du système tristatique UHF est qu'il permet d'avoir 3 mesures indépendantes du même phénomène.

Les principales caractéristiques de ces deux radars sont les suivantes :

	Système UHF	Système VHF
Fréquence	960 MHz	240 MHz
Puissance	5 MégaW crête	5 MégaW crête
transmise	150 kW moyenne	150 kW moyenne
Modulation	Pulse de 10 micros.	Pulse de 10 micros.
	à 10 milli s.	à 10 milli s.
Antenne	Paraboloïde 32 m	Antenne parabolique
	de diamètre	de 10.000 m2

Ces deux systèmes de radar sont couplés à des chaînes de réception identiques dans leur principe. Le travail présenté ici ayant été réalisé avec le système UHF, on va maintenant rentrer dans le détail de cette chaîne d'émission/réception.

La chaîne d'émission/réception UHF



Les 3 sites de Kiruna, Sodankila et Transö ont exactement la même chaîne de réception, le site de Tronsö comportant en plus l'émetteur UHF.

L'émetteur UHF a déjà été présenté. C'est un radar capable d'émettre des pulses d'une longueur comprise entre 10 micro-seconde et 10 milli-seconde. De plus, il peut émettre sur 8 fréquences différentes avec un temps de commutation de l'ordre de la micro-seconde.(La rétrodiffusion du signe dépendant de la fréquence d'émission, cela permet d'avoir des mesures indépendantes du même phénomène)

Le radar est piloté par un radar controller. Le radar controller est un processeur indépendant qui en fonction d'une horloge extérieure indique à l'émetteur quelle impulsion émettre, sur quelle fréquence, pendant quelle durée.

Le radar controller pilote également la chaîne de réception. L'altitude d'où provient le signal rétrodiffusé dépend du temps écoulé entre l'émission du signal et sa réception. Pour mesurer le signal rétrodiffusé dépuis une zone donnée, il est nécessaire de l'échantillonner à des instants précis. Les échantillonneurs de la chaîne de réception sont pilotés par le radar controller. A ce sujet, il est à noter qu'émis à très haute fréquence, le signal rétrodiffusé par les électrons libres est également reçu à très haute fréquence, ce qui ne permet pas de le traiter directement. Pour permettre un traitement numérique, il est translaté à la fréquence O. Pour celà, il est multiplié par un sinus et un cosinus à la même fréquence. On obtient alors deux composantes X et Y en quadrature, condidérées comme partie imaginaire et réelle du signal et qui contiennent l'information totale portée par le signal. Ce sont ces deux composantes en quadrature qui sont numérisées par la chaîne de réception. Une expérience consiste à répeter un certain nombre de fois (paramètre du radar controller) un schèma d'impulsion/ réception. Pour chaque cycle de réception, les échantillons numérisés (X ; Y) sont rangés sequentiellement dans un buffer par fréquence d'émission. Les données sont traitées en temps réel et accumulées par un corrélateur microprogrammable.

Le corrélateur est une machine de traitement numérique semi-spécialisée et très rapide (un micro-instruction se déroule en 250 nano sec.) :

-spécialisée, car elle est connue pour calculer des expressions de la forme $\sum_{i=0}^{N} X_i \, X_{i-j}$

-semi-spécialisée, car étant programmable, elle permet de calculer un grand nombre de fonctions ce qui introduit une certaine souplesse dans son utilisation.

A titre d'exemple, on peut vouloir calculer la puissance du signal reçu : $\sum_{i=0}^{N} X^2 + Y^2$

ou la fonction de corrélation sur des échantillons provenant de la même altitude mais à des instants différents

$$\sum_{i=0}^{N-1-i} X_i \, X_{i+1} + Y_i \, Y_{i+1}$$

(fonction de corrélation portant sur tous les points ou tronquée ...).

Pilote de la chaîne d'émission/réception, le radar controller supervise également le corrélateur temps réel. A chaque cycle de réception les données numérisées sont stockées dans un buffer. A la fin d'un cycle de réception, c'est le radar controller qui donne au corrélateur l'ordre de commencer son calcul. Ces données acquises seront alors traitées par le corrélateur et les résultats seront accumulés aux précédents. Au bout d'un certain nombre d'accumulations (paramètre du radar controller) la mémoire résultat du corrélateur sera transmise au corrélateur.

Rôle du corrélateur dans la chaîne d'émission/réception

Dans la chaîne d'émission/réception de chaque site, le calculateur a un rôle de superviseur général. Tout ce qui est programmable ou pilotable est relié au calculateur via des modules CAMAC (CAMAC est une norme internationale de liaison d'appareillage électronique).

Ainsi :

-Le calculateur charge la mémoire programmable du radar controller (programmation des émissions/réceptions à Tromsö et des réceptions seules pour les stations éloignées).

-Dans chaque site, le calculateur pilote le positionnement de l'antenne.

-C'est enfin le calculateur qui charge la mémoire-programme du corrélateur avec le code qu'il aura à exécuter. Lors d'une expérience, toutes ces opérations sont gérées en temps réel par un moniteur développé à EISCAT. Ce moniteur, EROS (Eiscat Real time Operating System), outre son travail de chargement des mémoires programme du corrélateur et du radar controller, de pointage de l'antenne, permet de plus de démarrer et d'arrêter une expérience, de démarrer et de stopper le transfert des données sur bandes magnétiques et de visualiser les fonctions calculées par le corrélateur au fur et à mesure de leur acquisition par le calculateur.

Outre son rôle de gestionnaire en temps réel du déroulement d'une expérience, le calculateur a un rôle primordial dans la préparation de la mise au point d'une expérience. En particulier il offre des facilités pour lire et vérifier les programmes du radar controller et surtout il permet d'écrire, tester, simuler et mettre au point les programmes pour le corrélateur. (On reviendra ultérieurement sur cet aspect des choses).

Synchronisation des sites

Il est clair que les radar controller sur chaque site doivent être parfaitement synchronisés pour que les chaînes de réception prennent bien au même moment les échantillons provenant de la même altitude. L'horloge programmable des calculateur étant d'une précision de 20 milliseconde, la synchronisation ne pourrait être prise en charge par les ordinateurs de chaque site.

La synchronisation est assurée par 3 horloges atomiques au Cesium qui indiquent rigoureusement la même heure sur chaque site. Cependant, ce sont les calculateurs de chaque station qui ont le rôle de supervision de l'ensemble de la chaîne d'émission/réception. En particulier, ce sont eux qui font démarrer une expérience. Or, il faut que celle-ci demarre sur les 3 sites au même moment. Les calculateurs n'étant pas capables de générer un ordre avec une telle précision, le mécanisme mis en place est le suivant. Chaque calculateur charge dans le radar controller du site l'heure du démarrage. Le radar controller étant relié à l'horloge atomique, (cf schèma de la chaîne de réception) commencer à exécuter son propre programme quand cette dernière indiquera l'heure choisie par le calculateur.

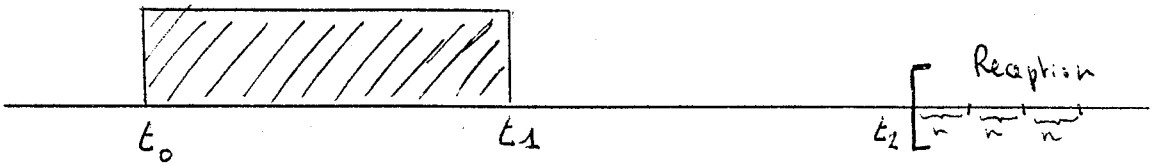
L'écart entre ces 3 horloges étant inférieur à la micro-seconde elles permettent la synchronisation des émissions/réceptions sur les 3 sites.

Liaison schéma d'impulsion-corrélateur

Une expérience est en grande partie déterminée par le schéma d'émission/réception et le programme du corrélateur qui va traiter les données reçues. Schéma d'impulsion et programme du corrélateur ne sont évidemment pas indépendants mais au contraire ce dernier dépend entièrement du schéma d'impulsion.

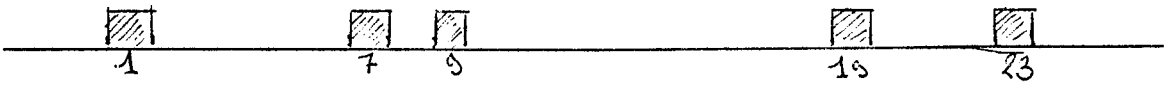
Les schémas d'impulsions varient essentiellement en fonction de l'altitude que l'on veut étudier et du site où doit avoir lieu la mesure. Le radar pouvant émettre sur différentes fréquences ou a un schéma d'impulsions différentes pour la station principale et pour les deux stations latérales. Pour bien montrer l'interdépendance du programme du corrélateur et du schéma d'impulsions prenons deux exemples.

Le single pulse. Dans cette hypothèse, le radar émet une impulsion longue sur une fréquence donnée



Au bout d'un certain nombre de micro-secondes commence l'échantillonnage des signaux reçus. On peut considérer que les échantillons proviennent de m tranches d'altitudes différentes, n échantillons provenant de la même tranche. Le corrélateur aura alors à calculer pour chaque tranche d'altitude la fonction de corrélation sur les n échantillons qui en proviennent. Ce schéma d'impulsions est bien adapté pour mesurer le milieu à haute altitude ($>$ 200 km)

Le multipulse. Dans ce schéma, on envoie des impulsions courtes et irrégulierement espacées. Dans un schéma actuellement utilisé à EISCAT on envoie 5 impulsions, et sur une échelle de temps dont l'unité serait la durée d'une impulsion, elles seraient émises aux temps = 1,7,9,19,23



Lorsque l'on commencera à numériser le signal rétrodiffusé, les échantillons séquenciels proviendront d'altitudes complétement différentes. En fait, on retrouvera dans le buffer d'échantillons la structure du schéma d'impulsions. Si on considére qu'à un pulse correspond un échantillon, les échantillons numérotés 1,7,9,19,23 viendront tous de la même altitude Ao, tandis que les échantillons numérotés 2,8,10,20,24 seront ceux qui viennent de l'altitude supérieure Al.

Le schéma d'impulsions multipulse est utilisé pour étudier le milieu à basse altitude ($<$ 200 km). Dans cette zone, le milieu varie rapidement en altitude et le schéma single pulse ne serait pas adapté.

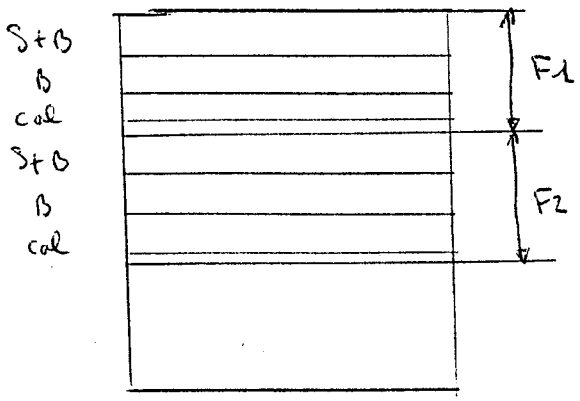
Ne serait-ce que parce que les échantillons sont rangés sequentiellement au fur et à mesure de leur numérisation, il est évident que les programmes du corrélateur qui devront traiter un schéma d'impulsions single pulse ou un schéma multipulses seront radicalement différents.

Enfin, le signal rétrodiffusé n'est pas un signal pur mais au contraire il est mélangé à un bruit. Ce bruit vient principalement de deux sources. L'une naturelle, est le bruit de fond de l'espace, l'autre est liée à la chaîne de réception elle-même. Pour arriver à isoler le signal reçu, il est nécessaire d'estimer les bruits parasites avec lesquels il est mélangé. Pour cela, tout d'abord, on prend des échantillons de bruit seul. (Echantillons pris à un moment où il n'y a plus de signal rétrodiffusé). Ensuite, pour mesurer le niveau du signal, on injecte à l'entrée de la chaîne de réception un bruit de puissance connue et on prend des échantillons de ce bruit connu à la sortie de la chaîne de réception, ce qui permettra de calibrer celle-ci.

Pour se résumer, le buffer d'entrée du corrélateur contient des échantillons

-de signal + bruit
-de bruit seul
-de calibration

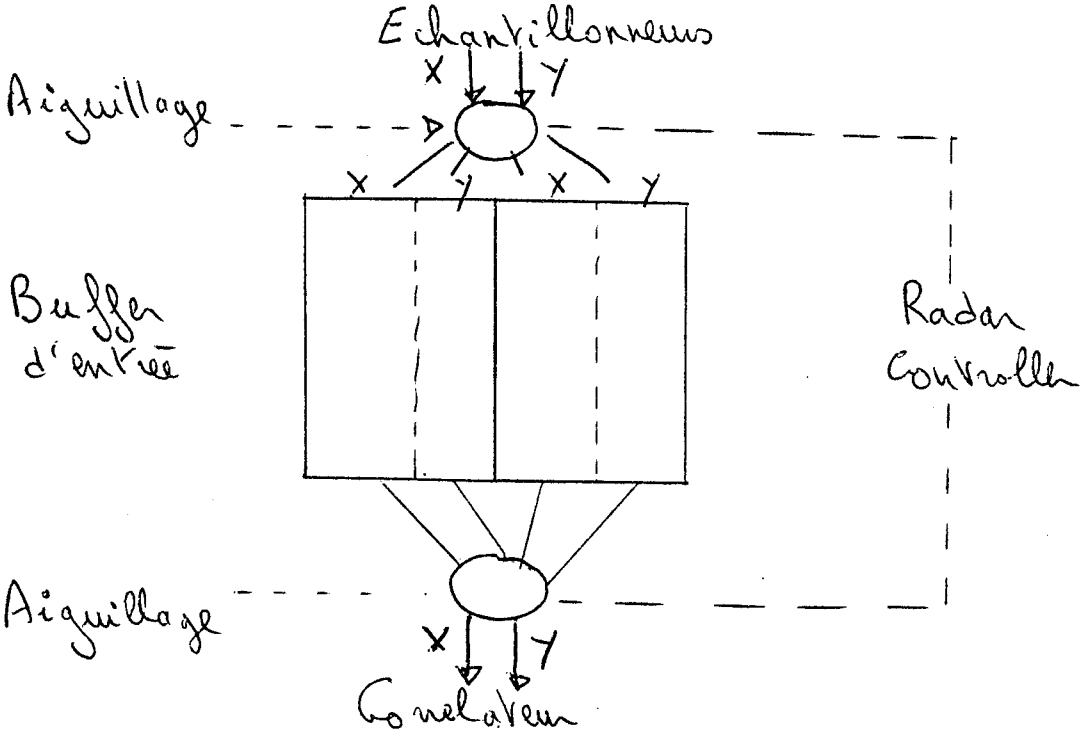
Tous ces échantillons sont rangés séquentiellement par fréquence au fur et à mesure de leur numérisation et le corrélateur traite ce buffer d'entrée avec un programme qui dépend évidemment du schéma d'émission/réception.



Note : l'adresse de début de chaque fréquence dans le buffer d'entrée est un paramètre géré par le moniteur EROS.

Comme il a été dit précédemment, une expérience consiste à répéter un certain nombre de fois le même schéma d'émission/réception.

Chaque cycle de réception donne des échantillons qui sont traités par le corrélateur, celui-ci traitant et accumulant plusieurs cycles de réception (réduction de données en temps réel). Le corrélateur doit donc pouvoir calculer sur une partie du buffer tandis que l'on continue l'acquisition des échantillons. Pour cette raison le buffer d'entrée a une structure de flip/flop, une moitié servant à la numérisation des échantillons, pendant que le corrélateur calcule sur les échantillons précédemment acquis dans l'autre moitié du buffer. La permutation entre les deux parties du buffer d'entrée est gérée par le radar controller.



Présentation du corrélateur

Pour pouvoir exposer le problème qui était posé et la solution que je lui ai apporté, il est nécessaire au préalable de présenter rapidement la structure du corrélateur. Le corrélateur d'EISCAT est un multiprocesseur microprogrammable. Il est capable en une seule micro-instruction d'adresser un échantillon dans le buffer d'entrée, de faire un calcul sur cet échantillon, d'en ranger le résultat avec ou sans accumulation dans une mémoire appelée buffer mémoiry ou buffer de résultats.

Il possède une mémoire de programme de 64 mots de 128 bits. Chaque micro-instruction se décompose en 7 sous champs qui sont eux-mêmes des instructions à chacun des 7 processeurs du corrélateur.

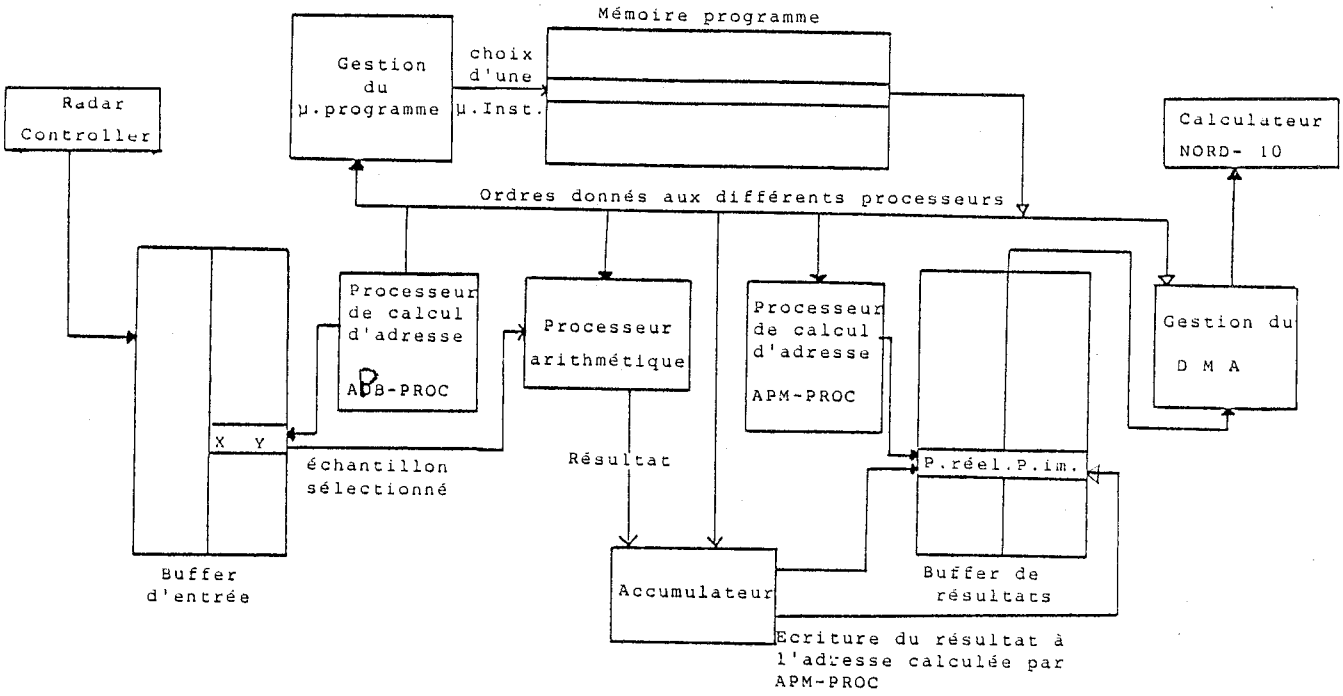
Pour le moment, les principaux processeurs qui nous intéressent sont :

-L'APB processor, processeur de calcul d'adresse qui sert à adresser un échantillon (X,Y) dans le buffer d'entrée

-L'APM processor, processeur de calcul d'adresse qui sert à adresser la result memory.

-Le processeur arithmétique qui exécute une opération arithmétique sur l'échantillon (X,Y) adressé par l'APB processeur.

-Le processeur d'accumulation qui prend le résultat du processeur arithmétique et l'écrit dans la result memory en l'accumulant ou non avec ce que cette dernière contenait déjà, à l'adresse calculée par l'APM processeur.



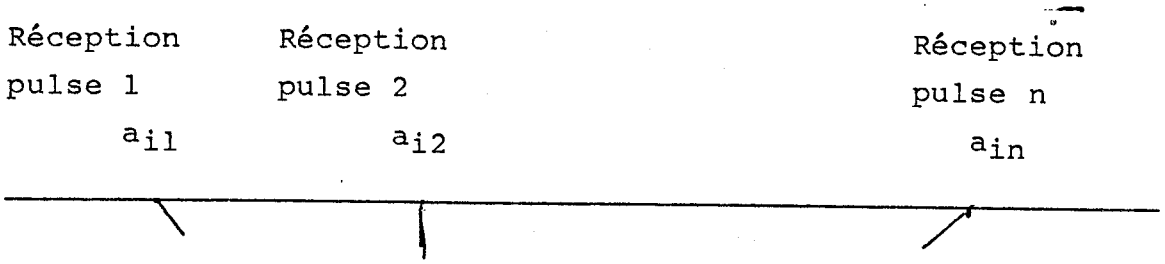
Cette présentation rapide est résumée par le schéma ci-dessus, qui est repris et détaillé dans la première partie de ce mémoire.

Présentation du problème à traiter.

Dans l'expérience que l'on se propose de réaliser, il est projeté d'additionner entre eux, soit 2 à 2, soit 3 à 3 etc... les échantillons de même nature (provenant de la même altitude et de la même fréquence d'émission), dans le but d'améliorer la qualité des mesures effectuées.

On calculera ensuite la fonction de corrélation à partir de ces nouvelles valeurs pour chaque altitude.

Appelons a_{ij} l'échantillon de l'altitude i et provenant de la rétrodiffusion du pulse de rang j. Dans le buffer d'entrée on aura la répartition d'échantillons suivante



$a_{i1} + a_{i2}$ ····· $a_{ik-1} + a_{ik}$ ····· $a_{in-1} + a_{in}$

Nouveaux échantillons provenant de l'altitude i
(cas de la sommation des points 2 à 2)

Ecartons tout de suite l'idée de modifier le sous-programme de corrélation qui existe déjà pour l'adopter à notre problème.

-En effet, on pourrait envisager tout d'abord d'additionner entre eux les échantillons dans le buffer d'entrée et d'appliquer ensuite l'algorithme de corrélation à ces nouveaux échantillons. Ce n'est pas possible, car on ne peut pas réécrire par programme dans le buffer d'entrée. (Il n'y a pas de communication entre la sortie de l'unité arithmétique et le buffer d'entrée).

-Enfin, il n'est pas envisageable non plus de vouloir faire l'addition dans la result memory et d'appliquer ensuite l'algorithme de corrélation sur ces nouvelles valeurs prises dans la result memory car il n'y a pas de communication entre cette dernière et l'entrée de l'unité arithmétique.

Le problème posé est original par rapport à toutes les applications conçues jusqu'à présent pour le corrélateur et demande pour être résolu l'écriture d'un microprogramme original. Pour résoudre le problème posé, on a décomposé la fonction de corrélation à calculer après additions d'échantillons en une série de sommes et de produits élémentaires, chose que le corrélateur sait faire.

L'algorithme proposé consiste donc à regrouper les différents produits élémentaires qui participent au calcul de la fonction de corrélation ; c'est ce qui est exposé dans la deuxième partie de ce mémoire.

PREMIERE PARTIE

PRESENTATION DU CORRELATEUR EISCAT

Cette présentation est faite, vue du côté du logiciel. Il ne sera pas question pour nous de chercher à approfondir l'aspect strictement matériel de ce corrélateur.

1 - DESCRIPTION SCHEMATIQUE

Ce corrélateur comprend un buffer d'entrée (BUFFER MEMORY) où sont rangés les échantillons à traiter et un buffer de sortie (RESULT MEMORY) où il stocke ses résultats.

Le corrélateur n'est pas isolé, mais fait partie d'un ensemble qui comprend essentiellement :

-le radar controller
-un calculateur NORD-10 de NORSK-DATA

-Le corrélateur est un multiprocesseur.
Pour différentes raisons, notamment des impértifs de vitesse, le corrélateur est un multiprocesseur dans lequel chaque instruction peut gérer 7 opérations en parallèle.

-Le corrélateur micro-programmable possède :

. une mémoire de programme de 64 mots de 128 bits
. différents champs de données.

1-1 Les buffers d'entrée et de sortie

1-1-1- La mémoire d'entrée

La mémoire d'entrée (BUFFER MEMORY) est divisée en deux parties symétriques qui peuvent être simultanément lues par le corrélateur et écrites par le radar controller.

Chaque partie de cette mémoire a des mots de 16 bits et peut avoir une capacité maximum de 64 Kmots.

C'est dans cette mémoire que sont sotckés les échantillons du signal mesuré. A chaque échantillon correspond 2 valeurs sur 8 bits.

X partie réelle de l'échantillon
Y partie imaginaire de l'échantillon.

Quand le corrélateur a fini de calculer sur la partie de la mémoire qu'il lisait, il y a permutation avec l'autre partie de la mémoire qui vient d'être écrite par le radar controller. La partie précédemment en lecture passe alors en écriture. Cette permutation est gérée par le radar controller.

1-1-2 La mémoire de sortie

La mémoire de sortie (RESULT MEMORY) a une taille maximum de 4 Kmots de 64 bits. C'est là que le corrélateur range et accumule les fonctions de corrélation qu'il calcule. Chaque mot de cette mémoire comprend :

-32 bits pour la partie réelle du résultat
-32 bits pour la partie imaginaire du résultat.

Partie réelle et partie imaginaire sont accédés ensemble par la même adresse.

1-2 Liaison avec l'extérieur

1-2-1 Liaison avec le NORD-10

Le corrélateur est relié au NORD-10 par une liaison en accès direct avec la mémoire (Direct Memory Access : DMA). Cette liaison est gérée par CAMAC (CAMAC est une norme internationale de connection électrique entre appareils.

Cette liaison sert :

-à transférer dans la mémoire du calculateur le buffer résultat

-à charger depuis le NORD-10 les mémoires programmables du corrélateur (mémoire de programme et de données : cf. 1-4-1 et 1-4-2).

1-2-2 Liaison avec le radar controller

Comme il a été dit précédemment, le radar controller gère le buffer d'entrée du corrélateur.

-remplissage d'une partie du buffer d'entrée

-permutation des deux parties de cette mémoire en fin de calcul.

De plus, c'est le radar controller qui donne au corrélateur le signal de début de calcul et qui lance le transfert DMA du buffer résultat vers le calculateur.

1-3 Le corrélateur comme multiprocesseur

Il est capable de mener 7 opérations en parallèle. Ces 7 opérations simultanées constituent un pas de calcul élémentaire.

D'une manière générale, une opération de calcul élémentaire peut se décomposer ainsi :

-choix de l'opérande dans le buffer d'entrée

-calcul arithmétique sur l'opérande sélectionné

-rangement du résultat, avec ou sans accumulation dans le buffer de sortie.

Un calcul complet, effectué par le corrélateur, va comporter un certain nombre d'opérations élémentaires. La gestion des calculs élémentaires est faite par des compteurs de boucle. Ceci correspond à 5 fonctions du corrélateur :

-calcul d'adresse sur le buffer d'entrée (APB PROCESSOR)

-unité de calcul arithmétique

-calcul d'adresse sur le buffer de sortie (APM PROCESSOR)

-accumulation des résultats dans le buffer de sortie

-gestion des compteurs de boucle (autrement dit, gestion d'un pointeur programme, cf. 2-1)

A ceci, il faut ajouter deux autres fonctions :

-gestion du transfert DMA avec le calculateur

-gestion des communications intercorrélateur (dans l'hypothèse d'un système multicorrélateur qui n'existe pas encore).

1-4 Les différents champs du corrélateur accessibles

Tout programme a besoin, pour s'exécuter :

-d'instructions

-de données

Dans ce qu'on appelle habituellement un programme, écrit en langage évolué ou en assembleur, on considère une zone où instructions et données peuvent être mélangées. Pour le corrélateur, programme et données sont physiquement rangés dans des zones différentes.

1-4-1 La mémoire de programme

Elle peut contenir 64 instructions. Chaque instruction occupe 128 bits. Une instruction est divisée en 7 champs, chaque champ étant l'instruction d'un processeur. Ces 7 champs sont exécutés simultanément.

Restriction

Sur les 64 mots de la mémoire de programme, seuls 63 sont utilisables.

-Le mot 0 de la mémoire est utilisé comme boucle d'attente de démarrage du calcul (IDLE LOOP)

-Le mot 63 est réservé à la gestion des interruptions externes (cas du système multicorrélateur) [-NON UTILISE]

1-4-2 Les mémoires de données

C'est dans ces mémoires que sont rangé les opérandes des différents processeurs.

Les principaux champs de données sont :

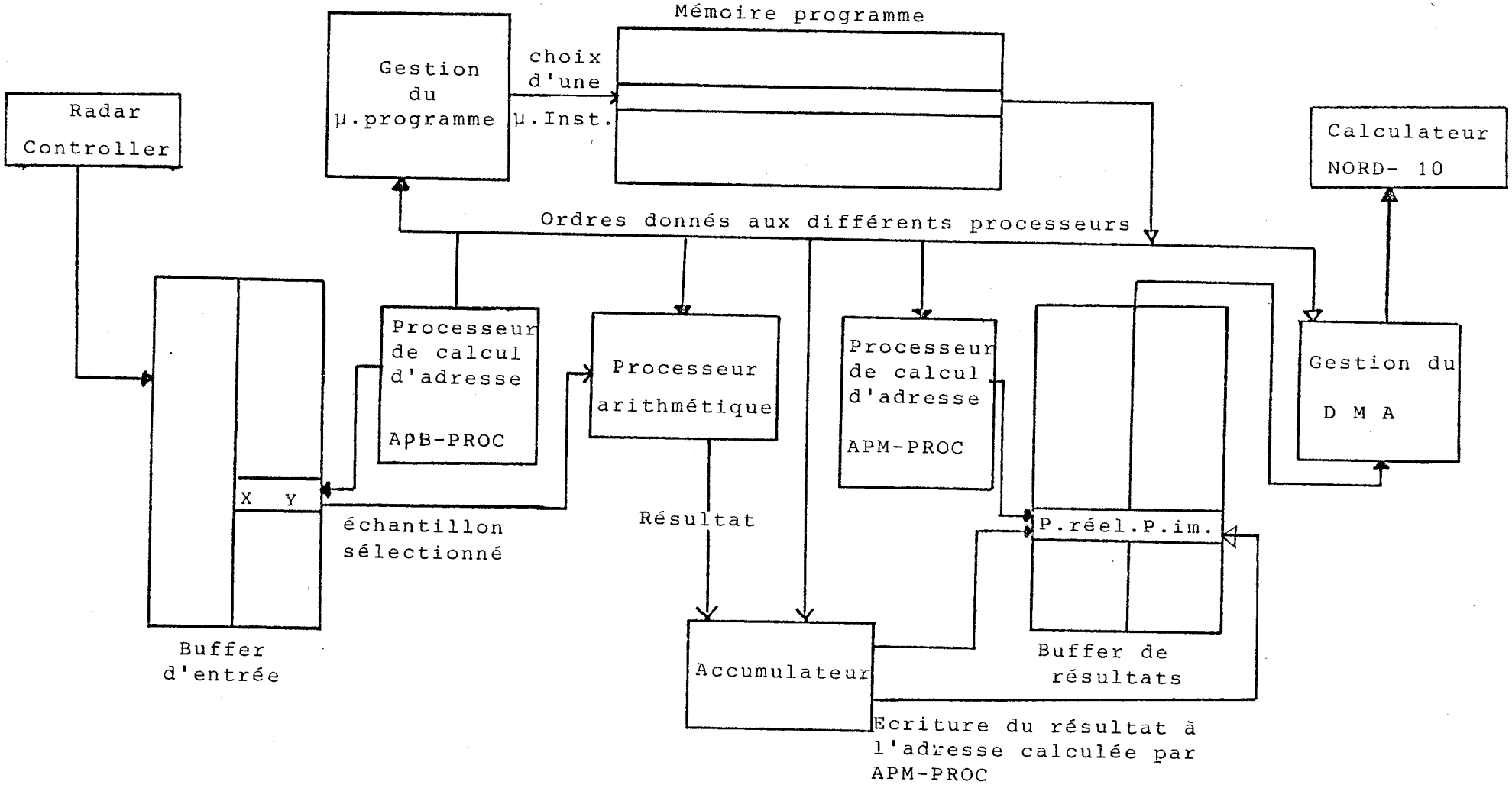
-L'APB-STACK : pile de 16 registres servant de mémoire à l'APB-processeur (processeur de gestion d'adresses du buffer d'entrée).

-L'APM-STACK : pile de 16 registres identiques aux précédents et servant à l'APM-processeur (cf 3).

-Des registres destinés à contenir les valeurs initiales des différents compteurs de boucle. Leur chargement ne se fait pas directement mais par un mécanisme décrit ultérieurement (cf 2)

-Le DATA-I-REGISTER. Ce registre doit contenir le nombre de mots de la result memory à transferer vers le calculateur.

-Le START-ADDRESS-REGISTER. Ce registre de 5 bits contient l'adresse de la première micro-instruction à exécuter. Quand le corrélateur ne travaille pas, il boucle sur la micro-instruction 0 (IDLE OOP). Quand le radar controller fait démarrer le corrélateur, le programme compteur est forcé avec la valeur de SAR.


Mémoire programme
Gestion du µ.programme
choix d'une µ.Inst.
Radar Controller
Calculateur NORD- 10
Ordres donnés aux différents processeurs
Processeur de calcul d'adresse APB-PROC
Processeur arithmétique
Processeur de calcul d'adresse APM-PROC
Gestion du D M A
X Y
échantillon sélectionné
Résultat
P.réel.P.im.
Buffer d'entrée
Accumulateur
Buffer de résultats
Ecriture du résultat à l'adresse calculée par APM-PROC

S C H E M A 1

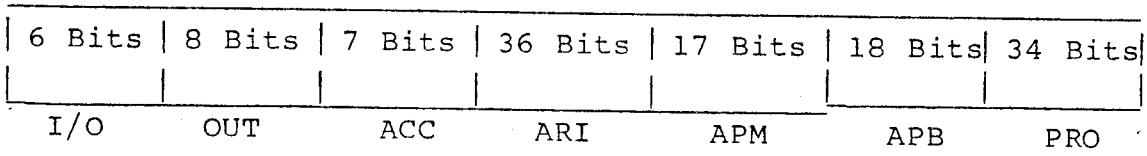
2 - FONCTIONNEMENT ET PROGRAMMATION

Les différents processeurs du corrélateur vont maintenant être décrits sous deux angles :

- leur mécanisme de fonctionnement
- leur programmation.

Chaque µ.instruction de 128 bits est divisée en 7 champs qui contrôlent chacun une fonction particulière. Lors de l'exécution d'une µ.instruction, les 7 fonctions s'exécutent en parallèle.

Ce découpage interne d'une µ.instruction est donné par le schéma 2



S C H E M A 2

PRØ-instruction : contrôle l'exécution du programme dans le corrélateur et permet le chargement des registres programmables

APB-instruction : contrôle l'adressage des données dans le buffer d'entrée (buffer memory)

APM-instruction : contrôle l'adressage des données dans le buffer de sortie (result memory)

ARI-instruction : contrôle les opérations arithmétiques effectuées sur les données

ACC-instruction : contrôle l'accumulation du résultat calculé à un instant donné avec ceux déjà rangés dans la result memory

ØUT-instruction : gère le transfert des données de la result memory vers le calculateur par DMA

I/O-instruction : contrôlerait les communications dans un système multicorrélateur (inutilisé)

(Elles ne seront pas présentées car le système multicorrélateur est abandonné).

2.1. Les PRØ-instructions

Ce champ gère :

- les compteurs de boucle
- le chargement des registres programmables
- le compteur ordinal

2.1.1. Les compteurs de boucle (schéma 3.3., et voir 2.1.2)

Pour effectuer un calcul, le corrélateur dispose de 3 compteurs de boucle de 12 bits chacun : LC1, LC2, LC3 (LØØP CØUNTER) et d'un registre temporaire LCR1A.

Les 3 compteurs de boucle ne sont pas identiques, en particulier, seul le premier, LC1 est stockable (rechargeable) à partir de LCR1A.

Dans le champ PRØ-instruction, 4 sous-champs indiquent l'opération à effectuer sur LC1, ...LC3, LCR1A.

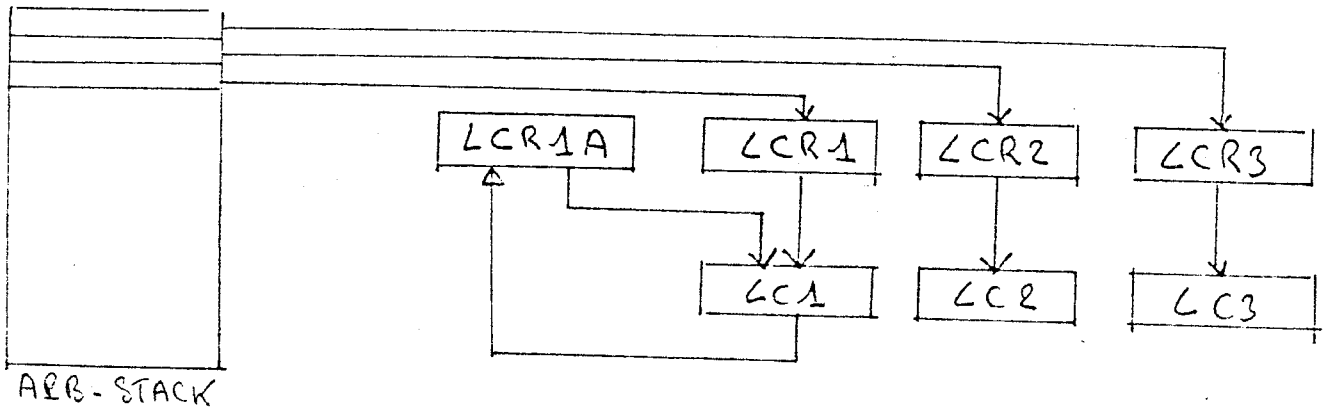
2.1.2. Mécanisme de chargement des loop-counters

Pour gérer les boucles de programme, on dispose de test à 0 sur les compteurs de boucle. La seule opération que l'on puisse faire sur un loop-counter est la décrémentation. Il est alors parfois nécessaire, en cours d'exécution, de recharger les loop-counter avec leur valeur initiale (ne serait-ce que pour gérer 2 boucles imbriquées). Ces valeurs initiales sont contenues dans les

trois "load register for loop-counter" LCR1, LCR2, LCR3.

Ces trois registres sont eux-mêmes chargeables à partir de valeurs rangées dans la mémoire de données APB-STACK (cf. 1.2.2.). Une fois ces registres chargés, leur contenu est transférable à volonté dans les loop counters.

Ce mécanisme peut être représenté par le schéma suivant :



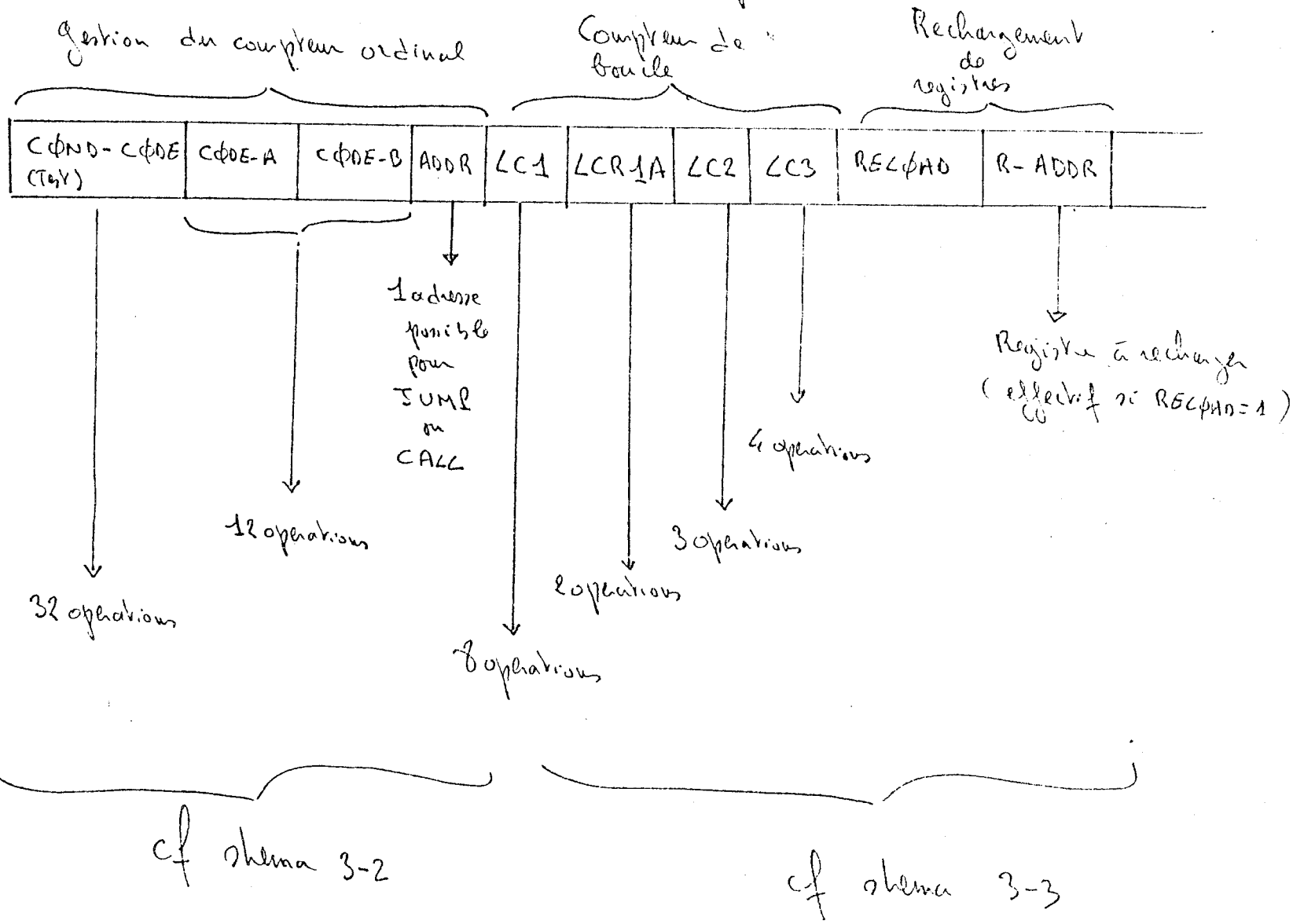
SCHEMA 4

Note : Si le chargement des 3 loop counters à partir des LCRi peut s'exécuter en une seule µ.instruction, il n'en est pas de même pour le chargement des LCRi.

Le chargement d'un LC i avec une valeur rangée dans l'APB-STACK nécessite 3 µ.instructions :

- une où le reload d'un LCRi est programmé
- 2 NOOP car le corrélateur a besoin de 2 cycles d'horloge après un reload. (Ceci est du à la structu pipeline du corrélateur).

Structure du champ programme



SCHEMA 3.1.

Gestion du compteur ordinal

```
 3   (IF LC1≠0 THEN B ELSEIF LC2≠0 THEN A OTHERWISE CONT)
 6   (IF LC3≠0 THEN B ELSEIF LC2≠0 THEN A OTHERWISE CONT)
 7   (IF LC1≠0 OR LC3≠0 THEN B ELSEIF LC2≠0 THEN A OTHERWISE CONT)
13   (IF LC1≠0 THEN B ELSEIF LC2=0 THEN A OTHERWISE CONT)
15   (IF LC1≠0 OR LC3≠0 THEN B OTHERWISE CONT)
16   (IF LC3≠0 THEN B ELSEIF LC2=0 THEN A OTHERWISE CONT)
17   (IF LC1≠0 OR LC3≠0 THEN B ELSEIF LC2=0 THEN A OTHERWISE CONT)
23   (IF LC1=0 THEN B ELSEIF LC2≠0 THEN A OTHERWISE CONT)
26   (IF LC3=0 THEN B ELSEIF LC2≠0 THEN A OTHERWISE CONT)
27   (IF LC1=0 OR LC3=0 THEN B ELSEIF LC2≠0 THEN A OTHERWISE CONT)
33   (IF LC1=0 THEN B ELSEIF LC2=0 THEN A OTHERWISE CONT)
35   (IF LC1=0 OR LC3=0 THEN B OTHERWISE CONT)
36   (IF LC3=0 THEN B ELSEIF LC2=0 THEN A  OTHERWISE CONT)
37   (IF LC1=0 OR LC3=0 THEN B ELSEIF LC2=0 THEN A  OTHERWISE CONT)
40   (USE-A)
46   (IF LC2≠0 OR LC3≠0 THEN B ELSE A)
47   (IF LC1=0 OR LC2≠0 OR LC3≠0 THEN B ELSE A)
54   (IF LC3≠0 THEN B ELSE A)
55   (IF LC1=0 OR LC3≠0 THEN B ELSE A)
56   (IF LC2=0 OR LC3≠0 THEN B ELSE A)
57   (IF LC1=0 OR LC2=0 OR LC3≠0 THEN B ELSE A)
62   (IF LC2≠0 THEN B ELSE A)
63   (IF LC1=0 OR LC2≠0 THEN B ELSE A)
66   (IF LC2≠0 OR LC3=0 THEN B ELSE A)
67   (IF LC1=0 OR LC2≠0 OR LC3=0 THEN B ELSE A)
71   (IF LC1=0 THEN B ELSE A)
72   (IF LC2=0 THEN B ELSE A)
73   (IF LC1=0 OR LC2=0 THEN B ELSE A)
74   (IF LC3=0 THEN B ELSE A)
75   (IF LC1=0 OR LC3=0 THEN B ELSE A)
76   (IF LC2=0 OR LC3=0 THEN B ELSE A)
77   (IF LC1=0 OR LC2=0 OR LC3=0 THEN B ELSE A)
```

Les différents vecteurs possibles

```
 0    PC = PC+1, POP STACK
 1    PC = RETURN ADDR., POP STACK
 2    PC = ADDR., POP STACK
 3    PC = SAR,  POP STACK
4,14  PC = PC+1
5,15  PC = RETURN ADDR.
6,16  PC = ADDR.
7,17  PC = SAR
 10   PC = PC+1, PUSH STACK
 11   PC = RETURN ADDR., PUSH STACK
 12   PC = ADDR.,  PUSH STACK
 13   PC = SAR, PUSH STACK
```

Opérations sur la pile gérées par le séquenceur

S C H E M A 3.2.

Opération sur les compteurs de boucle

0	NOOP
1	LC1 = LC1-1
2	LC1 = LCR1
3	LC1 = LCR1A
4	IF LCL = 0: LC1 = LCR1, LC2 = LC2-1, ELSE: LC1 = LC1-1
5	IF LC1 = 0 and LC3 = 0: LC1 = LCR1A, ELSE: LC1 = LC1-1
6	IF LC1 = 0: LC1 = LCR1, ELSE: LC1 = LC1-1
7	IF LC1 = 0: LC1 = LCR1A, ELSE: LC1 = LC1-1

LC1:

0	NOOP
1	LCR1A = LC1

LCR1A:

0	NOOP
1	LC2 = LC2-1
3	LC2 = LCR2

LC2:

0	NOOP
1	LC3 = LC3-1
2	LC3 = LCR3
3	IF LC3 = 0: LC3 = LCR3, ELSE: LC3 = LC3-1

LC3:

Les opérations programmables sur les 3 compteurs de boucle ne sont pas identiques pour chacun d'eux.

Opération de rechargement d'un registre

RELOAD:

0	NOOP
1	REGISTER-RELOAD

R-ADDR.:

4	RELOAD SAR	
5	"	BAR, APB
22	"	LCR1
23	"	LCR2
24	"	LCR3

L'opération de rechargement n'est effective que si le bit RECOAD est à 1

Note : Ne pas confondre LCi : = LCRi (chargement d'un compteur de boucle avec une valeur initiale) avec le chargement de valeur initiale elle-même (RELOAD).

2.1.3. Gestion du compteur ordinal

On appelle compteur ordinal ou
pointeur ordinal ou
pointeur programme ou
program counter,

l'adresse de l'instruction en cours d'exécution. Par convention, le compteur ordinal est noté PC.

Au cours de l'exécution d'un programme, le compteur ordinal prend différentes valeurs (exécution des différentes instructions).

La valeur du pointeur programme à un instant t est fonction :

- de sa valeur à l'instant t-1
- du résultat de l'instruction exécuté à l'instant t-1.

Généralement, d'un pas de programme à l'autre, le compteur ordinal est incrémenté de 1 (Pc = Pc + 1), sauf si l'instruction en cours d'exécution est un branchement (conditionnel ou non).

Exemple :

```
        I = 1                       0
10      IF (I.EQ.2) GO TO 20        1
        I = I + 1                   2
        GO TO 10                    3
20      CONTINUE                    4
```

En admettant qu'en machine, chacune de ces lignes n'occupe qu'une instruction et qu'elles soient rangées aux adresses 0,1,2,3,4, les valeurs de PC seraient :

t = 0	PC = 0
t = 1	PC = 1
t = 2	PC = 2
t = 3	PC = 3
t = 4	PC = 1
t = 5	PC = 4

Dans un langage de programmation au plus bas niveau (assembleur), on peut distinguer 2 types d'instructions :

- celles qui incrèment le compteur ordinal (après leur exécution, le calculateur passe à l'instruction suivante)
- celles qui chargent le compteur ordinal avec une autre adresse (branchement conditionnel ou non) ; c'est ce dernier type d'instruction qui permet de faire des boucles dans les programmes.

Le mécanisme de branchement conditionnel dans les programmes du corrélateur est un peu différent en ce sens que les tests et les opérations sur le compteur ordinal sont disjointes.

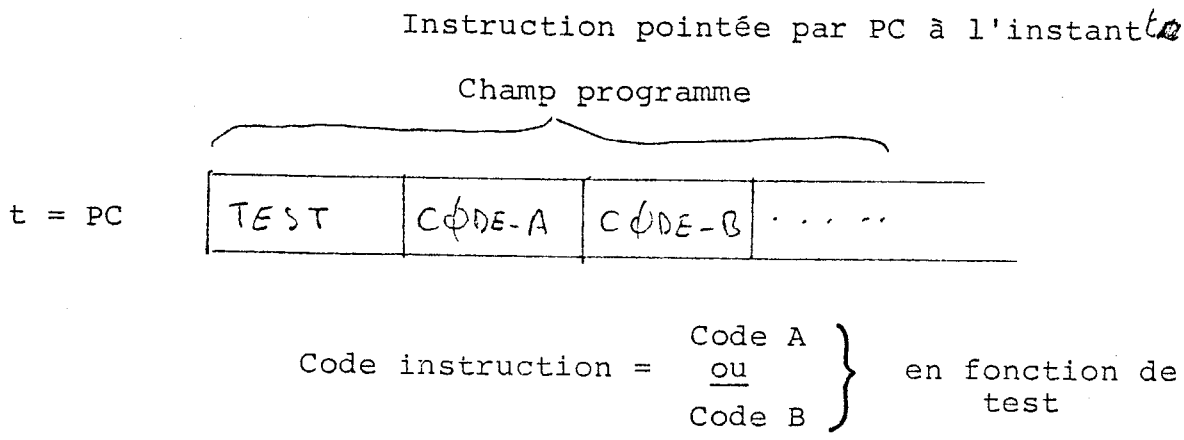
Dans le champ instruction du corrélateur, il y a 2 sous-champs appellés CODE-A et CODE-B (détail de ce champ : cf. schéma 3.1.).

Chacun d'eux reçoit le code d'une opération possible sur le compteur ordinal.

Un troisième sous-champ du champ instruction contient le code de l'un des 32 tests possibles sur les compteurs de boucle (liste des tests schéma 3.2.).

C'est le résultat du test qui détermine le choix entre le code A et le code B.

Ce mécanisme peut être résumé par le schéma 5.



t + 1 : Nouveau PC = f (code instruction)

S C H E M A 5

Programmation du compteur ordinal

Pour faciliter la programmation des boucles et pour permettre aux µ.programmes d'avoir des structures de sous-programme, le corrélateur dispose d'une pile.

Cette pile comporte 4 mémoires et est gérée selon le principe du LIFO (dernier entré, premier sorti).

A un instant donné, on ne peut référencer que la dernière valeur rentrée dans la pile. Le jeu d'instructions que l'on peut programmer :

-gère cette pile

a) sauvegarde dans la pile de la valeur du compteur ordinal + 1

(adresse de retour pour l'appel d'une subroutine)

b) destruction de la dernière adresse rentrée dans la pile

c) branchement à la dernière adresse rentrée dans la pile.(avec ou sans destruction de la pile),

La liste complète des instructions est donnée dans le schéma 3.2.

- permet d'effectuer des branchements inconditionnels ou de continuer en séquence.

Programmation de tests

Comme il a déjà été signalé, les 32 tests possibles (cf. schéma 3.2.) sont des tests sur la valeur des compteurs de boucle. On ne peut pas tester une valeur particulière sur un compteur, on ne peut que tester s'il est nul ou non nul.

Il existe deux types de tests, les simples et les doubles, qui permettent de choisir entre le CODE-A et le CODE-B.

Dans le cas des tests simples, le choix se fait de la manière suivante :

Si le test est vrai, alors utiliser le CODE-B, sinon utiliser le CODE-A.

Exemples :

IF LC1 = 0 THEN B ELSE A (code test 71)
IF LC1 = 0 OR LC2 ≠ 0 THEN B ELSE A (code 63)

Pour les tests doubles, la règle est la suivante :

> Si test 1 vrai , alors utiliser CODE-B ; sinon, si test 2 vrai , alors utiliser le CODE-A,
> sinon continuer en séquence :

Exemple :

Test 1	Test 2
IF LC3 ≠ 0 THEN B	ELSEIF LC2 ≠ 0 THEN A
OTHERWISE CONTINUE.	

Restriction :

Le champ instruction (cf. schéma 3) ne comporte qu'une seule adresse de branchement, donc, seul l'un des deux codes A ou B pourra être une instruction de branchement.

Note :

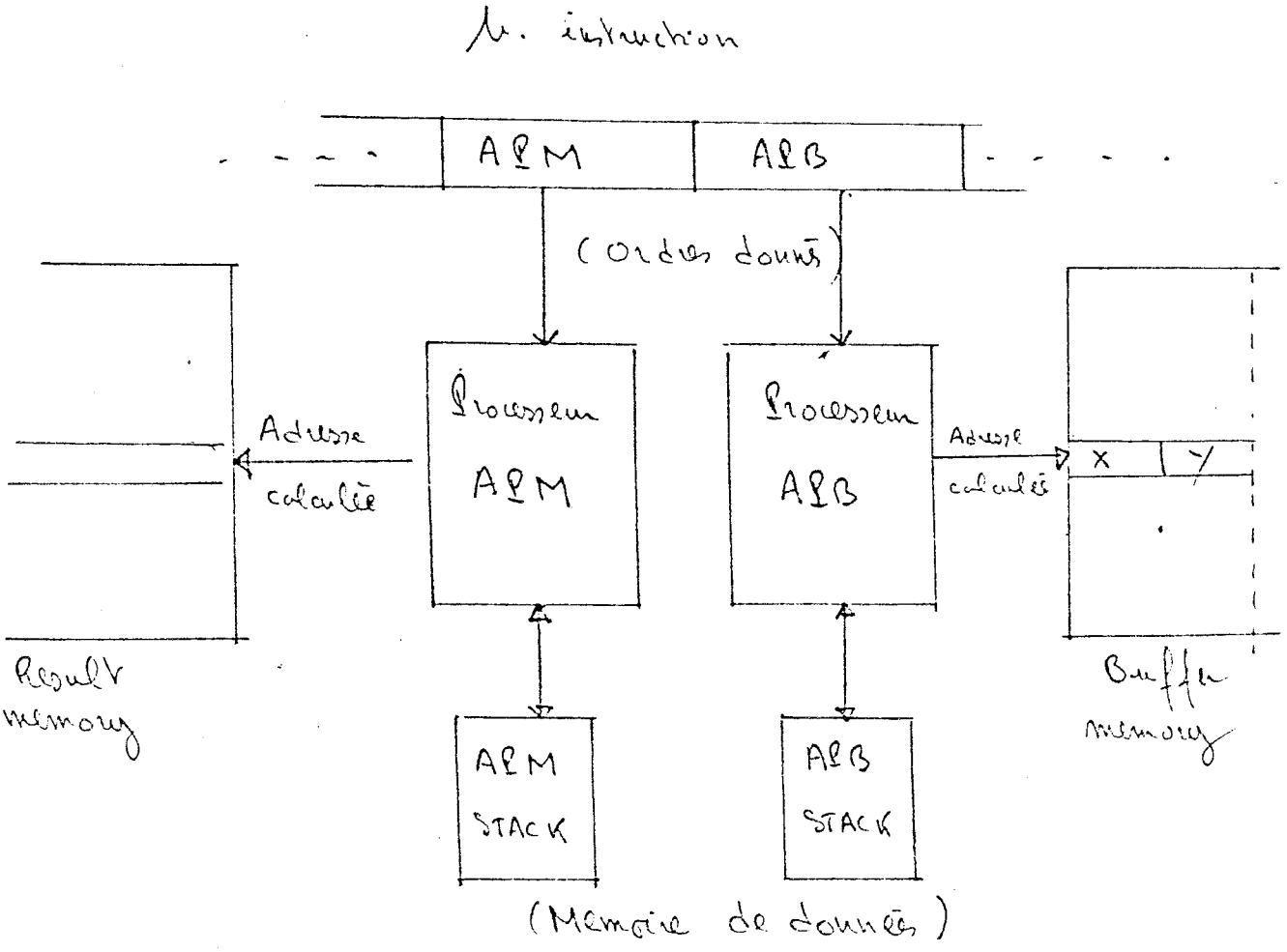
Dans le champ programme, on fait à la fois des opérations sur les compteurs de boucle et des tests sur leur valeur. L'opération se déroule en deux temps :

- d'abord le test sur la valeur du compteur
- ensuite, l'opération sur le compteur.

2.2. Les APB-APM instructions

2.2.1 - Généralités

Ce sont les instructions qui contrôlent l'adressage du buffer d'entrée et du buffer résultat. Ces instructions sont identiques, mais elles concernent deux processeurs différents, opérant en parallèle.



SCHEMA 6

Chacun des deux processeurs a sa mémoire de données de 16 registres, l'APB-STACK et l'APM-STACK.

Note

- la capacité d'adressage de l'APM est de 4 Kmots de 64 bits (32 + 32 ; partie réelle et imaginaire) et celle de l'APB de 64 Kmots de 16 bits (8 bits pour chaque échantillon).

- une adresse dans le buffer d'entrée concerne les deux échantillons X et Y.

- une adresse dans la result memory concerne la partie réelle et la partie imaginaire du résultat.

- chaque élément de la mémoire de donnée APB-STACK et APM-STACK, s'accède comme un tableau par son indice ou son adresse dans le tableau.

- Les champs instructions APM et APB ne contiennent que deux adresses d'opérande dans la mémoire de donnée. A un instant donné, on ne peut donc référencer que deux éléments de la mémoire APB-STACK ou APM-STACK.

- en plus de la mémoire de donnée, chacun des deux processeurs APM et APB disposent d'un registre programmable.

Terminologie

La terminologie utilisée dans les notes techniques d'EISCAT concernant le corrélateur est la suivante :

- R et S sont les opérandes (sélectables) dans chacune des mémoires de données
- A et B sont les adresses de ces opérandes
- La mémoire de données s'appelle RS.
  Un opérande est donc : R = RS (A) ou S = RS (B)
- Le registre programmable est le Q-REGISTER.

Restriction

Les deux opérandes ne sont pas symétriques, car, par construction, l'élément référencé par l'adresse A peut être seulement lu, tandis que celui référencé par l'adresse B peut être lu et écrit.

2.2.2. principes de fonctionnement.

Une opération d'un processeur de calcul d'adresse peut se décomposer en trois temps :

a) choix d'un opérande (ALU-SOURCE). Cet opérande peut être :

   RS (A), RS (B), Q, 0 (zéro), ou le DATA-I-REGISTER (cf out-instructions).

b) calcul sur cet opérande (ALU-FUNCTION)

   Sur les deux opérandes choisis, on peut programmer des opérations d'addition, de soustraction, ou des opérations d'un intérêt moins évident pour le calcul d'adresse comme le OU, le ET, le OU exclusif, etc..., mais qui sont liées au circuit qui réalise cette fonction.

c) utilisation du résultat

Le résultat du calcul effectué sur les opérandes sert à adresser le buffer d'entrée (APB processeur) ou celui de sortie (APM processeur).

Dans la littérature concernant le corrélateur EISCAT, cette adresse est appelée OUT.

Cependant, on distingue :

OUT qui est l'adresse dans le buffer
F qui est le résultat du calcul proprement dit.

D'une manière générale, ce résultat :

- sert d'adresse dans le buffer : OUT = F

et

- peut être conservé :

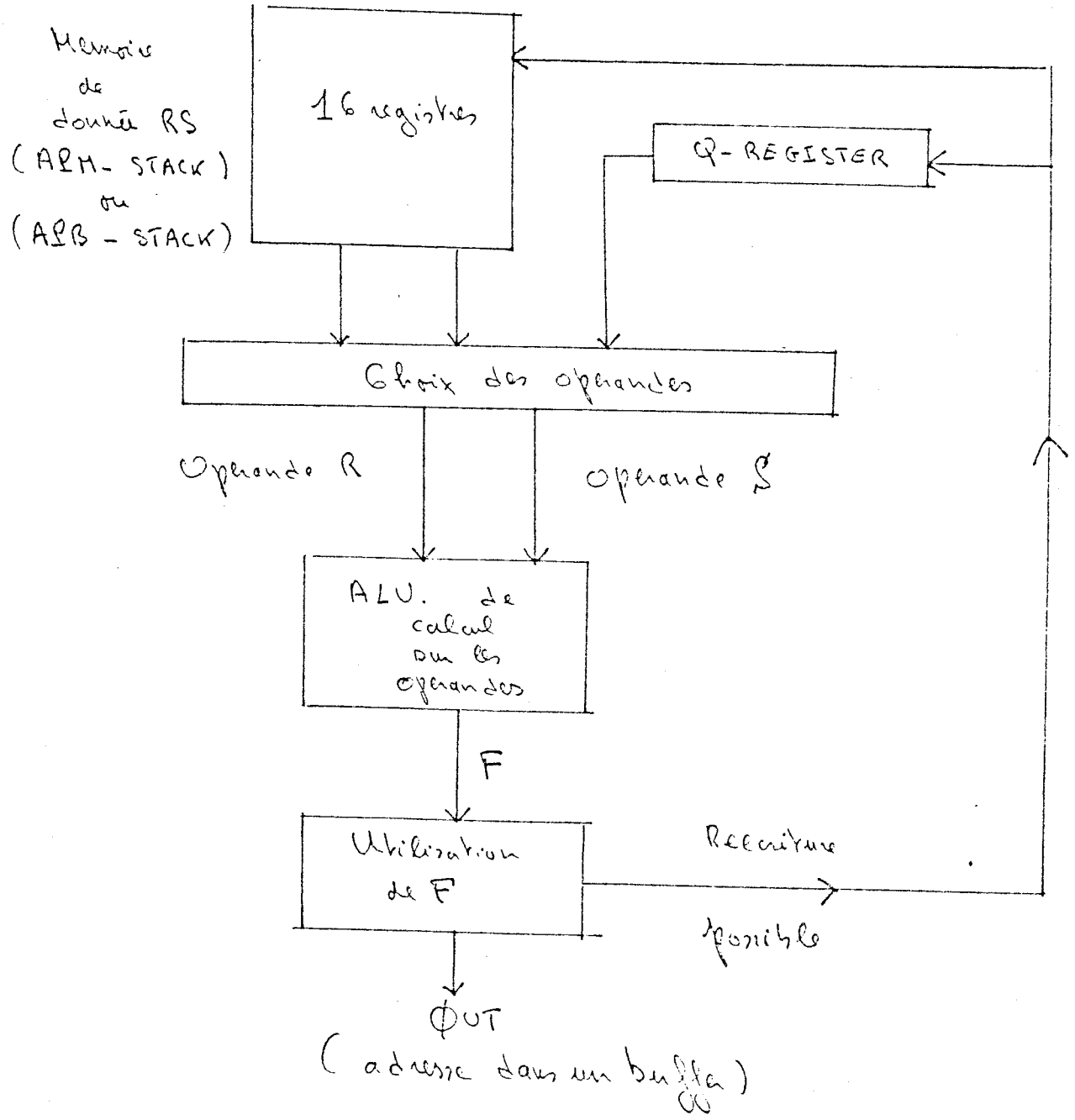
  . dans la mémoire de donnée : RS (B) = F

  ou

  . dans le registre Q : Q = F.

Cette possibilité de pouvoir stocker des résultats permet entre autre d'avoir une "adresse courante" qui sert de base au calcul de l'adresse suivante.

Le mécanisme décrit précédemment est illustré par le schéma 7.


Mémoire de donnée RS (ARM - STACK) ou (ARB - STACK)
16 registres
Q - REGISTER
Choix des opérandes
Opérande R
Opérande S
ALU. de calcul sur les opérandes
F
Utilisation de F
Réécriture possible
OUT
(adresse dans un buffer)

SCHEMA 7

2.3. - Les instructions arithmétiques

- Dans le buffer d'entrée, chaque échantillon sur 16 bits est considéré comme un échantillon complexe, la partie réelle X et la partie imaginaire Y étant chacune représentée sur 1 octet.

- L'unité arithmétique du corrélateur est prévue pour effectuer un produit complexe : (en une seule opération)

$$(a_1 + ib_1) \times (a_2 + ib_2) = (a_1a_2 - b_1b_2) + i(a_1b_2 + a_2b_1).$$

Pour effectuer ce produit le plus rapidement possible, l'unité arithmétique du corrélateur dispose de 4 multiplieurs en parallèle (cf. schéma 8)

- Chaque multiplieur a deux entrées, appellées A et B
- On peut charger les multiplieurs, soit avec un échantillon interne (provenant du buffer d'entrée), soit avec une valeur externe (pour des programmes de test du corrélateur).

Une opération arithmétique se décompose en 3 temps :

a) Choix de l'opérande pour chaque entrée de chaque multiplieur

Etant donné le multiplieur 1 (par exemple), on peut choisir pour l'entrée A :

- la valeur $X_{interne}$ ou $Y_{interne}$ (le couple X,Y étant à une adresse calculée par l'APB processeur,
- la valeur $X_{externe}$ ou $Y_{externe}$ (test du corrélateur) (pour le test, le corrélateur prend ses échantillons dans une mémoire morte),

Ce choix est à faire pour les entrées A et B de chaque multiplieur.

Note : On peut également choisir, en ce qui concerne l'entrée A des multiplieurs, de faire rentrer la valeur 1.

b) Utilisation du résultat des multiplieurs

Chaque multiplieur effectue le produit A x B. Les multiplieurs sont regroupés 2 par 2 pour donner la partie réelle et la partie imaginaire du résultat complexe.

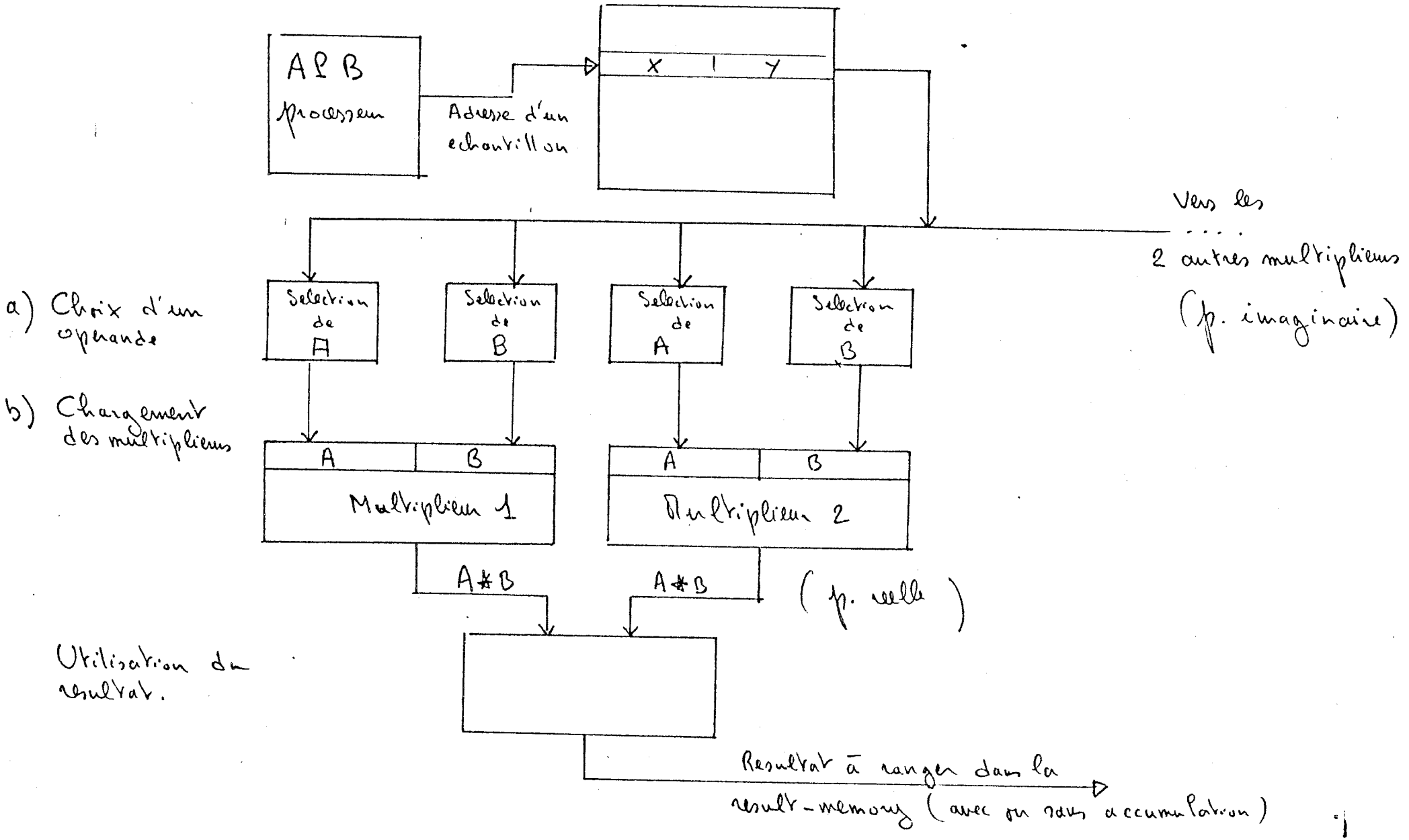
Pour chaque groupe de 2 multiplieurs, on peut faire l'opération :

Résultat : $M_1 + M_2$, $M_1 - M_2$, M_1, M_2, -1 — on ignore le résultat de l'un des multiplieurs

Ce qui sort de cette dernière opération est le résultat de l'unité arithmétique du corrélateur. Ce résultat (partie réelle et partie imaginaire) est destiné à être rangé dans la result memory à l'adresse calculée par l'APM processeur.

Note : Si, comme on l'a vu en b-, on est maître du chargement de chacune des entrées A et B des 4 multiplieurs, il doit être bien clair que l'échantillon X et Y est le même pour tous les multiplieurs.

C'est celui qui est adressé par l'APB processor, qui n'adresse qu'un seul échantillon (XY) à un instant donné dans le buffer d'entrée.


A ? B
processeur
Adresse d'un echantillon
X | Y
Vers les ....
2 autres multipliers
(p. imaginaire)
a) Choix d'un operande
b) Chargement des multipliers
Selection de A
Selection de B
Selection de A
Selection de B
A
B
Multiplieur 1
A
B
Multiplieur 2
A*B
A*B
(p. reelle)
Utilisation du resultat.
Resultat à ranger dans la result-memory (avec ou sans accumulation)
Moitié de l'unité arithmétique du coprocesseur

SCHEMA 8

2.4. Les opérations d'accumulation

2.4.1 - LES MECANISMES MIS EN SERVICE

2.4.1.1. Mécanisme 1

Soit à calculer l'expression suivante :

$$S = \sum_{i=0}^{N} X_i X_{i-1}$$

Cette expression sera calculée en N + 1 opérations.

Soit X_0X_1 le premier terme de cette somme. Ce terme sera rangé dans la result memory à une adresse donnée. Si on range le deuxième terme X_1X_2 à la même adresse, il écrasera le terme précédent.

Pour calculer correctement S, il faut l'additionner au terme précédent. C'est le mécanisme d'accumulation.

Le résultat d'un calcul de l'unité arithmétique du corrélateur peut être soit :

- rangé dans la result memory (en écrasant ce qu'il y avait au préalable)
- soit additionné avec ce qu'il y avait dans la result memory et "rerangé" à la même adresse.

(Ceci aurait pu être évité si la result memory était remise à zéro lors de l'initialisation du corrélateur, ce qui n'est jamais fait.

2.4.1.2. Mécanisme 2

Soit une expérience se déroulant en plusieurs étapes dans le temps et supposons que l'on veuille accumuler dans la result memory les résultats de chaque étape.

Par exemple :

A l'étape 0, le corrélateur a calculé $S_o = \sum_{i=0}^{N} X_{oi} X_{o\,i-1}$

A l'étape 1 " " S_1

A l'étape M " " $S_M = \sum_{i=0}^{N} X_{M\,i} X_{M\,i-1}$

Le résultat final voulu est

$$S = \sum_{j=0}^{M} Sj$$

Le mécanisme 1 est insuffisant, puisqu'à chaque étape, on écraserait le résultat de l'étape précédente.

Pour résoudre ce problème, le corrélateur possède un second dispositif d'accumulation, plus prioritaire, permettant d'inhiber le premier mécanisme décrit précédemment et de faire l'accumulation dans la result memory, même si le dispositif d'accumulation 1 indique de ne pas la faire.

Ceci correspond à deux modes de fonctionnement du corrélateur.

- on commence une expérience et le contenu de la result memory est écrasé
- on continue une expérience et les résultats sont accumulés au contenu de la result memory.

Ce mode dans lequel fonctionne le corrélateur est déterminé par le radar controler qui :

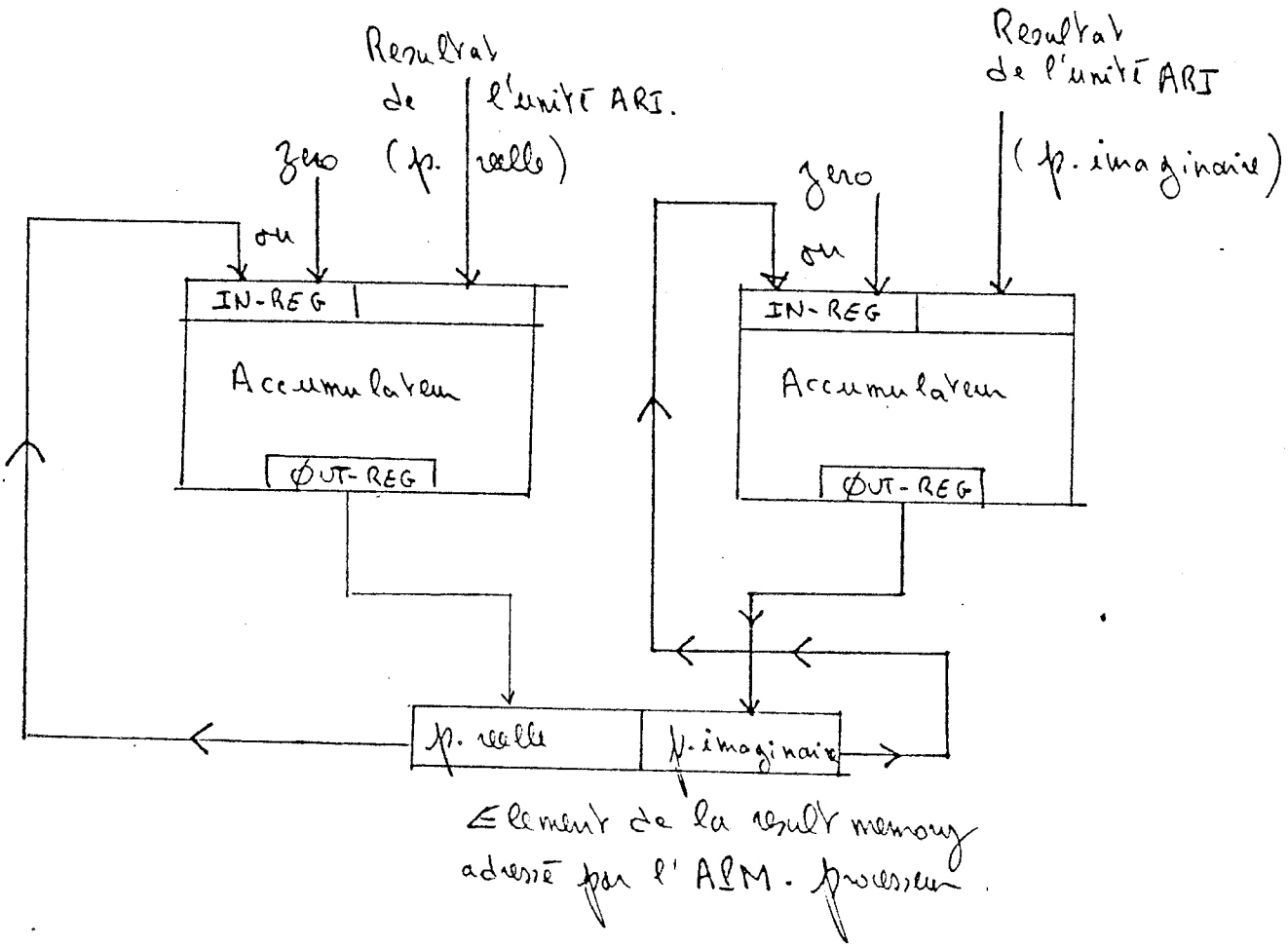
.../

- demarre une expérience avec l'ordre START-EXPERIMENT
- continue l'expérience avec l'ordre CONTINUE-EXPERIMENT.

2.4.2 - DESCRIPTION DES MECANISMES D'ACCUMULATION

2.4.2.1. Description

L'accumulation des résultats, en partie réelle et partie imaginaire, se fait dans deux accumulateurs. Ce mécanisme est décrit par le schéma 9.



SCHEMA 9

Chaque accumulateur a deux entrées. Une des entrées est toujours le résultat du calcul de l'unité arithmétique. L'accumulation se fait, ou ne se fait pas, selon que l'on charge dans l'autre entrée de l'accumulateur ce que contenait déjà la result memory, ou que l'on y charge la valeur zéro.

2.4.2.2. Terminologie

Chaque accumulateur a deux entrées et une sortie.

La sortie est OUT-REGISTER, que l'on peut écrire dans la result memory à l'adresse calculée par l'APM processeur.

Ces deux entrées sont :
- la sortie de l'unité arithmétique
- le IN-REGISTER

Le IN-REGISTER est chargé :

- soit avec le contenu de la result memory (accumulation)
- soit avec 0 (pas d'accumulation)

- le premier mécanisme d'accumulation s'appelle FF1
- le second s'appelle FF2.

2.4.2.3. Fonctionnement

Indépendament de tout mécanisme d'accumulation, les instructions d'accumulation contiennent 3 bits indiquant si oui ou non on doit :

- charger les IN/OUT REGISTER des accumulateurs
- écrire le OUT-REGISTER dans la result memory
- charger le IN-REGISTER avec le contenu de la result memory (ce chargement, s'il est demandé,

sera ou ne sera pas effectué en fonction des mécanismes FF1 et FF2).

Chaque mécanisme FF1 et FF2 est contrôlé par 2 bits

- un bit Set qui arme le mécanisme (bit à 1)
- un bit Clear qui le desactive s'il était actif (bit à 1)

(Les valeurs 0 correspondent à une NO-OPERATION).

Une fois armé, FF1 et FF2 restent actifs tant qu'une opération clear n'est pas programmée.

- Si FF2 est desarmé, la lecture de la result-memory dans le IN-REGISTER est contrôlé par FF1
- Si FF2 est armé, le contrôle de FF1 est inhibé.

Dans le cas où FF2 est désarmé :

- Si FF1 est armé, le IN-REGISTER est chargé avec le contenu de la result memory
- Sinon, le IN-REGISTER est chargé avec 0

Dans le cas contraire où FF2 est armé, il y a toujours lecture de la result memory dans le IN-REGISTER.

Pour qu'une opération d'accumulation puisse avoir lieu, il est évidemment nécessaire que ces bits soient programmés à 1.

-bit read pour charger le IN register des accumulateurs

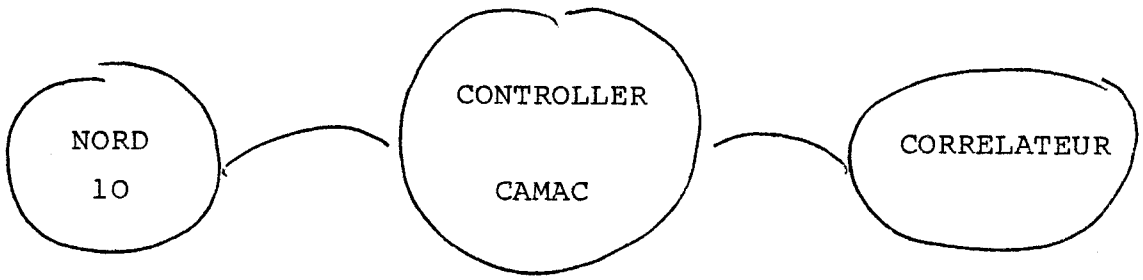
-bit write pour écrire dans la result memory

-bit validant le chargement des registres des accumulateurs. Ils sont la condition nécessaire mais non suffisante de tout mécanisme d'accumulation. Leur absence dans une micro-instruction empêche toute opération d'accumulation, quel que soit l'état des mécanismes FF1 et FF2. Ceci est très important car cela permet de n'avoir des opérations d'accumulations que lorsque l'unité arithmétique a effectivement fait un calcul et non pas après l'exécution d'une NO-OPERATION

2-5 Les OUT-Instructions

Le champ des micro-instructions permet le transfert en DMA de la result memory vers le calculateur via un contrôleur CAMAC.

2-5-1 Principe de fonctionnement

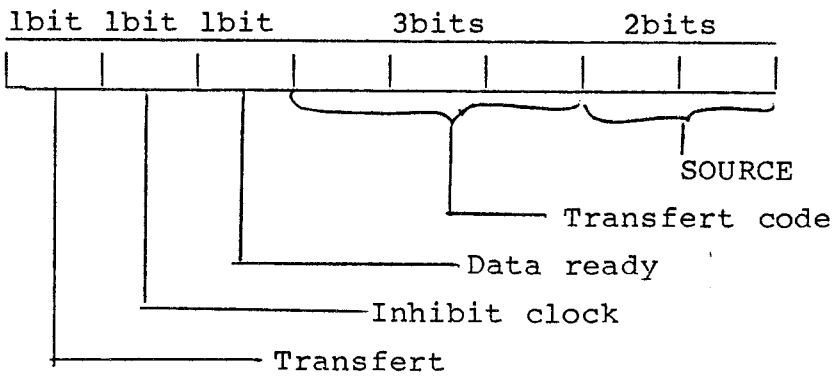


-L'échange se fait mot à mot par mot de 16 bits. La result memory comportant des mots de 64 bits, le transfert d'un mot de la result memory donne lieu à 4 échanges élémentaires.

-L'échange se fait au rythme du calculateur. L'horloge interne du corrélateur est inhibée et le transfert est rythmé par le contrôleur CAMAC.

-Le transfert se fait sur compte de mots. Le nombre de mots à transferer est contenu dans le DATA-I-REGISTER qui doit avoir été initialisé au préalable (initialisé au chargement du corrélateur, au même titre que l'APB-STACK ...).

2-5-2 Détail du champ OUT-introduction



Le champ comporte 8 bits :

-Transfert : valide l'opération de transfert ; doit être positionné à 1 pour que l'échange puisse avoir lieu.

-Inhibit clock : permet d'inhiber l'horloge interne du corrélateur

-Data-ready : envoie le signal data ready au contrôleur CAMAC. Ce n'est que quand celui-ci aura envoyé au corrélateur le signal data-received que l'échange du mot sera lancé.

-Transfert-code : indique ce que doit être envoyé au corrélateur. Cela peut être :

-le mot d'état du corrélateur

-un élément de la result memory. La result memory a des mots de 64 bits divisés en deux fois 32 bits (partie réelle et partie imaginaire). Le transfert code permet de selectionner les 16 bits à transferer (poids faible/poids fort de la partie réelle ou imaginaire).

-Source : le champ ne servirait que dans l'hypothèse d'un système multicorrélateur. Ce système aurait été un système hierarchisé comportant au maximum :

-un corrélateur maître (code 0)

-3 modules esclaves (code 1,2,3)

Le champ source préciserait alors dans quelle unité sont les valeurs à transferer au calculateur. Le système multicorrélateur n'existant pas, ce champ contient systématiquement le code 0.

2-5-1 Terminologie :

Les 32 premiers bits de la result memory sont généralement notés DATA-CHANNEL 1, les 32 suivants DATA-CHANNEL 2. Pour chacun, les 16 bits de poids-fort sont notés. MS (most significant), tandis que les 16 bits de poids faible sont notés LS (less significants)

2-5-4 Remarques :

Le champ OUT n'est pas suffisant pour gérer un transfert de données vers le calculateur. Ce transfert nécessite un programme complet. Ce programme se déroule sur 4 champs selon le schéma suivant :

PRd-instructions	APB-instructions	APM-instructions	OUT-instructios
Reload LCR3	Presenter DATA-I-REG	Q=0	NOOP
NOOP	NOOP	NOOP	NOOP
LC3 = LCR3	NOOP	Q=Q-1	
LC3 = LC3-1		Q=Q+1;OUT=Q	Transfert MS Data-ch1
		OUT = Q	Transfert LS Data-ch1
		OUT = Q	Transfert MS Data-ch2
non LC3 = 0? oui		OUT = Q	Transfert LS Data-ch2

fin

-Le transfert DMA a lieu en fin d'experience, et une experience comporte la rééxécution d'un même programme plusieurs fois (cf. mécanisme d'accumulation). Or, ce nombre de répetitions n'est pas connu du corrélateur, ce n'est pas un paramètre programmable. On ne sait donc pas, à priori, quand aura lieu le transfert des données. Dans un micro-programme, on code jamais d'instruction CALL DMA-TRANSFERT, car, dans ce cas, le transfert aurait lieu à chaque cycle de calcul.

Ce problème est résolu par le radar controller qui en fin d'expérience force le pointeur programme avec l'adresse du programme de DMA-Transfert.

En conséquence :

.Dans toute application on est obligé de prévoir dans le programme de DMA-Transfert.

.On est également obligé de le mettre à adresse fixe ! Le radar controller force le PC. toujours à la même adresse, indépendamment de l'application. L'adresse d'implantation choisie pour le DMA-Transfert est 32 décimal, au milieu de la mémoire programme, ce qui a pour inconvénient d'obliger parfois à "couper" un programme en morceaux.

3 - LES OUTILS DE DEVELOPPEMENT

En même temps que le corrélateur, les Norvégiens ont développé différents logiciels qui lui sont liés.

Ces logiciels peuvent se décomposer en trois groupes :

-des outils de développement et de mise au point des micro-programmes

-des outils de dialogue avec le corrélateur (chargement des différents champs, visualitation de la result memory)

-des programmes pour tester en ligne le bon fonctionnement du corrélateur (tests de l'unité arithmétique, des processeurs d'adressage ...)

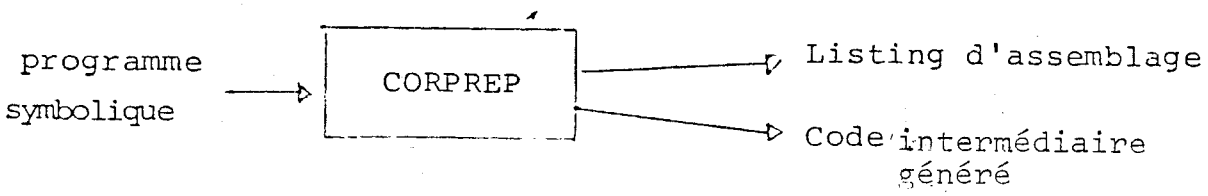
Malheureusement, le classement en différents types d'outils n'est pas aussi net que cela dans la pratique et la plupart de ces programmes sont à la fois des outils de mise au point, de test et de dialogue. Pour qui ne dispose pas du corrélateur en ligne, ils deviennent inutilement encombrants et lourds d'emploi. C'est pourquoi on a procédé au CEPHAG à une réorganisation de tous les logiciels. De tous nous en avons conservé un tel quel, un micro-assembleur et on en a reconstruit un second à partir de différents morceaux existants.

Nous allons maintenant présenter rapidement le micro-assembleur.

3.1 - Le programme CORPREP

CORPREP est un assembleur qui génère le code des µ.programmes du corrélateur.

Son fonctionnement peut être résumé par le schéma suivant



CORPREP range le code généré dans un fichier dont le nom lui est donné en paramètre.

Le symbolique qu'on lui donne à assembler a la structure suivante :

- Toutes les valeurs numériques sont exprimées en octal.
- Une µ.instruction occupe une ou plusieurs lignes, chaque µ.instruction est séparée de la suivante par la pseudo-instruction NXT

- Chaque µ.instruction est divisée en 7 champs correspondant aux 7 champs du corrélateur
- chaque champ commence par un mot clé.

Exemple :

- PRO- Pour les instructions programme
- ARI- Pour les instructions arithmétiques

- A l'intérieur de chaque champ, l'assembleur reconnaît des phrases clé, correspondant à la programmation des sous-champs.

  Chaque phrase clé est séparée de la suivante par ";"

Exemple de champ : APB-SRC = ZA ; FUNC=R+S ; DEST=F ; A = 16

APB-	Mot clé du champ correspondant aux instructions APB
SRC=ZA	SRC, mot clé du sous-champ indique la source où chercher les opérandes. ZA : le premier opérande R = 0 (zéro) le deuxième S = RS (A)
FUNC=R+S	La fonction à réaliser est la somme des opérandes
DEST=F	Le résultat de l'opération R + S sert à adresser le buffer d'entrée
A = 16	L'opérande S = RS (A) est à l'adresse 16/8 dans l'APB-STACK

Pour plus de détails, cf. la description des processeurs d'adresse et le rapport EISCAT 79/15.

Cet assembleur a l'inconvénient de faire toutes ses entrées/sorties en octal, alors que le reste des autres logiciels travaillent en décimal. Mais il a l'immense avantage de générer des NOOP (qui ne sont pas toujours des zéros) dans tous les champs qui ne sont pas programmés. Enfin, il génére en code intermédiaire, relisible à l'éditeur de texte, compatible avec tous les autres logiciels. La structure de ce code intermédiaire est donnée en annexe.

3-2 Le programme CORRTST

Ce programme est un outil de développement qui a été remodelé au CEPHAG à partir de différents modules élaborés en Norvège.

Dans son état actuel, il comprend :

-un éditeur
-un simulateur
-une fonction graphique

3-2-1 L'editeur

Beaucoup plus souple que l'assembleur, l'éditeur est actuellement l'outil de création/modification de micro-programme par excellence.

Il a deux fonctions principales :

3-2-1-1 Une fonction "d'édition de texte"

Il ne s'agit pas d'un éditeur de texte au sens usuel du terme.

Il permet de lire (commande READ) et d'écrire (commande WRITE) des micro-programmes écrits sur des fichiers en code intermédiaire. Comme ce code n'est pas facilement lisible et interprétable, l'éditeur le redécompose et présente :

-chaque micro-instruction champ à champ (PRO instruction, APB instruction ...)

-les valeurs des data fields champ à champ.

Les micro-instructions étant très longues, l'éditeur ne permet de travailler que sur un type de champ à la fois, mais permet par contre de travailler sur le même champ de plusieurs micro-instructions en même temps. De plus, chaque champ est décomposé selon les différents sous-champs qui le constitue. Par exemple, pour les PRO-instructions on aura :

condition-code code A code B adresse ....

En ce qui concerne les micro-instructions, deux commandes sont disponibles :

-commande de visualisation
LPS : champ ad_1, [ad_2]

Champ indique la nature du champ de la micro-instruction à visualiser. Ainsi, PRO correspond aux PRO-instructions, ARB aux ARB-instruction, etc... etc...

ad_1 est l'adresse de la lère micro-instruction à visualiser
ad_2 est un paramètre facultatif

-s'il est omis, l'éditeur, visualisera sur l'écran de la console le champ choisi de la micro-instruction d'adresse ad_1
exemple : LPS : APM 12

-s'il est présent, l'éditeur visualisera les champs choisis des micro-instructions de l'adresse ad_1 à l'adresse ad_2
exemple : LPS : PRO 0,10

-commande de modification :

SPS : champ ad_1, [ad_2]

champ, ad_1, ad_2 ont la même signification que précédemment.

Par exemple la modification du champ accumulation de la micro-instruction d'adresse 8 sera demandée par la requête :

SPS : ACC 8

Quand une requête de modification est faite à l'éditeur, celui-ci attend que l'on donne les valeurs qui remplacent alors, sous-champ à sous-champ les valeurs initiales des sous-champs de la micro-instruction.

On ne peut modifier selectivement un sous-champ, on est obligé de réecrire les valeurs de tout le champ.

Penible pour une modification unique (paramètre ad_2 absent). Cette particularité permet d'écrire plusieurs champs en une seule fois. (Demander une modification d'une micro-instruction qui n'existe pas, revient à la créer).

Par exemple, SPS : ARI 0,10 permettra d'écrire en une seule fois le champ instruction arithmétique des micro-instructions d'adresse 0 à 10.

Bien entendu, les mêmes sous-champs sont générés de ad_1 à ad_2, mais c'est très agréable pour générer des NO-OPERATION.

De même que l'on peut visualiser/modifier les micro-instructions, l'éditeur permet la même chose sur le data fields.

Dans ce cas, les commandes sont plus "rudimentaires"

LD permet de visualiser tous les champs de données qui sont définis (APB-STACK, APM-STACK, DATA-I-REGISSER ...)

SDF permet de les modifier.

Pour la modification, la syntaxe suit de près la structure du code intermédiaire (cf. annexe), puisque l'on est obligé de donner :

-l'adresse, la sous-adresse, la valeur.
Par exemple :
SDF
16,15,1 permet de charger la valeur 1 dans le registre 15 de l'APB-STACK.

Enfin, dans la fonction éditeur de texte, une autre commande est disponible : PPS.

PPS permet d'écrire sur l'imprimante toutes les micro-instructions d'un programme, champ par champ, ce qui permet d'en avoir une vision globale.

3-2-1-2 Une fonction "d'éditeur de lien"

Comme pour la fonction d'éditeur de texte, il ne s'agit pas d'un éditeur de lieu classique.

Sa fonction principale est de permettre des translations d'adresse. Prenons un exemple. Imaginons un micro-progrmme implanté à l'adresse 1 et qu'à la micro-instruction d'adresse 5 on ait programmé un branchement inconditionnel à la micro-instruction d'adresse 2. Le champ PRO-instruction de la micro-instruction d'adresse 5 aura les sous-champs suivants :

(Memory) location	condition code	code B	code A	adresse ...
5	56	0	6	2
	utiliser le code A inconditionnellement		JUMP à l'adresse	2

Si maintenant on veut implanter ce programme à l'adresse 11, il faudra modifier le code programme de la micro-instruction d'adresse 15 en réactualisant l'adresse de branchement qui passe de 2 à 12.

L'éditeur se charge automatiquement de ce problème de translation d'adresse, et cela de 2 manières :

-La commande READ nom [ad]

On a déjà parlé de la commande READ qui permet de lire un programme écrit en code intermédiaire sur un fichier.

Il faut maintenant préciser que cette commande comporte un paramètre facultatif [ad] qui est l'adresse d'implantation dans la mémoire programme du corrélateur. Si cette adresse d'implantation est omise, l'éditeur implante le programme à partir de la première adresse qu'il trouve dans le code intermédiaire.

Grâce à cette commande, on peut facilement construire un programme comportant plusieurs sous-programme déjà existants puisque l'on reste maître de l'adresse d'implantation.

-La commande SHIFT ad_1, ad_2, N

Elle permet de translater dans la mémoire programme, en mettant à jour les adresses de branchement, toutes les micro-instructions comprises entre les adresses ad_1 et ad_2 et cela de N positions. (N 0 ou N 0).

Cette commande est fondamentale pour l'écriture de micro-programmes. En effet, elle permet :

-de détruire une micro-instruction en l'écrasant par celle qui suit

-d'insérer une ou plusieurs micro-instructions en faisant un décalage vers les adresses croissantes de la mémoire programme.

Cette commande implantée sur la dernière version de l'éditeur en fait un véritable instrument de programme et fera tomber en désuétude le micro-assembleur. En effet, dans les versions pécédentes de l'éditeur qui n'avaient pas cette commande SHIFT, supprimer ou rajouter une instruction était impossible, à moins de réecrire toutes les suivantes ...

3-2-2 Le simulateur et la fonction graphique

3-2-2-1 Le simulateur

Comme son nom l'indique, il permet de simuler l'exécution d'un micro-programme. Cette simulation permet de visualiser le déroulement

-soit de la partie programme et adressage
- . adresse de la micro-instruction exécutée
- . valeur des différents compteurs de boucles
- . adresse calculée par les processeurs ARB et APM

-soit de la partie arithmétique
- . valeur contenue à un instant donné par les 4 multiplieurs
- . valeur contenue dans les accumulateurs
- . fonctionnement des mécanismes d'accumulation, la result memory étant en W ou RW

On peut également avoir la trace complète de l'exécution d'un programme sur l'imprimante.

Bien évidemment, ces traces de simulations, tant sur un écran que sur l'imprimante, sont facultatives, une exécution simulée pouvant durer plusieurs heures...

Dans la version originale du simulateur, le buffer d'entrée était simulé, soit par un générateur de nombres aléatoires, soit par une sinusoïde prise sur 512 points et répetée identiquement à elle-même modulo 512. Ce mode de simulations du buffer d'entrée a été modifié au CEPHAG, et dorénavent on peut :

-générer une sinusoïde sur un nombre de points variables (max : 600)

-répéter n fois le même point, ce qui revient à remplacer la sinusoïde par une fonction en escalier, ou plutôt d'avoir n sinusoïdes se chevauchant.

L'avantage d'une telle fonction, est le suivant : supposons que l'on ait redoublé 3 points. Dans le buffer d'entrée, les valeurs sont identiques par "paquet" de 3. Si maintenant on suppose que chaque point est un échantillon provenant d'une altitude différente, le calcul ne portant que sur des échantillons provenant de la même altitude, on doit avoir dans le buffer résultat 3 fonctions identiques. Ce type de génétion sert a vérifier le fonctionnement des micro-programmes.

3-2-2-2 La fonction graphique

A la fin de la simulation, la fonction graphique permet de visualiser le contenu de la result-memory.

Elle permet de tracer sur un écran :

-toute la result memory
-une partie seulement (loupe)

Elle a été modifiée au CEPHAG pour permettre de tracer sur un BENSON les courbes tracées sur l'écran.

Outre le fait que la fonction graphique permet d'avoir la valeur des différents points de la result memory (partie réelle/partie imaginaire), elle permet de detecter les erreurs grossières. Par exemple :

-si pour plusieurs altitudes les échantillons sont les mêmes, les résultats doivent être identiques,

-si en entrée on a une sinusoïde, on sait de manière théorique que la fonction de corrélation prise sur un nombre fini de points doit être une cosinusoïde modulée par un triangle.

Elle permet aussi d'apprécier directement la valeur des résultats.

DEUXIEME PARTIE

Comme il a été dit précédemment, le problème posé consiste à réaliser sur le corrélateur une fonction de corrélation calculée sur des sommes d'échantillons.

Pour des raisons qui ont été exposées en introduction, il n'est pas possible de faire une modification de microprogramme déjà existant.

Pour résoudre le problème posé, il est nécessaire d'écrire un programme original.

Remarques préliminaires

-Il est clair que l'on ne cherche à additionner entre eux que des échantillons provenant de la même altitude mais d'émissions différentes. C'est pourquoi, pour simplifier l'écriture on va raisonner pour une altitude donnée et ne prendre en compte que le numéro de l'émission.

Soit donc (a_1, a_2 ... a_n) les échantillons de la même altitude, provenant des émission 1,2 ... n

-Il est également clair, et on le supposera dans toute la suite de ce rapport, que le nombre d'émission doit être un multiple des nombres de points à additionner.

-On établira l'algorithme d'abord pour une altitude et ensuite on l'étendra à un nombre quelconque d'altitudes.

1 - CALCUL D'UNE FONCTION D'AUTOCORRELATION SUR DES SOMMES D'ECHANTILLON

Soit N échantillons que l'on désire additionner K à K.

Une somme S_i s'écrira :

$$S_i = \sum_{p=0}^{k-1} a_{ik+p} \quad , \quad i \quad [\, 0,1,2 \ldots \frac{K}{N} \,]$$

L'élément de rang j de la fonction d'autocorrélation s'écrit :

$$R(j) = \sum_{l=0}^{N/K-j} S_e * S_{e+j} \quad ; \text{ soit :}$$

$$R(j) = \sum_{l=0}^{N/K-j} \left(\sum_{p=0}^{K-1} a_{ek+p} * \sum_{p'=0}^{k-1} a_{(l+j)k+p'} \right)$$

$$R(j) = \sum_{l=0}^{N/K-j} \sum_{p=0}^{K-1} \sum_{p'=0}^{k-1} a_{ek+p} * a_{(l+j)K+p'}$$

1-1 Considerations sur l'algorithme d'autocorrélation simple

Avant d'aborder le problème de sommation des points, considérons l'algorithme de calcul de la fonction de corrélation pour une altitude et n émissions.

Cette fonction nommée R aura n points, chaque point étant :

$$R(i) = \sum_{j=1}^{n-i-1} aj \; a^*_{j+i} \quad \text{où } i \quad [0,n-1]$$

a_i est un échantillon complexe. a^*_i est son conjugué.

L'algorithme du calcul procède de la manière suivante :

Soit n+1 échantillons numérotés de 0 à n.

Soit un élément a_i. On effectue tous les produits possibles $a_i\ a_j$ avec j variant de i à n.

Une fois ces produits effectués, on recommence avec l'élément i+1 et ceci pour i variant de 0 à n.

On a donc un élément "fixe" devant lequel on fait défiler tous les autres éléments, en commençant par lui-même et jusqu'au dernier. Chacun de ces produits élémentaires est un constituant d'un point de la fonction de corélation.

$R(o) \leftarrow R(o) + a_i \times a_i$
$R(1) \leftarrow R(1) + a_i \times a_{i+1}$

$R(n-i-1) \leftarrow R(n-i-1) + a_i + a_{n-1}$
$R(n-i) \leftarrow R(n-i) + a_i + a_n$

} Pas de calcul n° i

$R(o) \leftarrow R(o) + a_{in} * a_{i+1}$
$R(1) \leftarrow R(1) + a_{i+1} * a_{i+2}$

$R(n-i-1) \leftarrow R(n-i-1) + a_{i+1} * a_n$

} Pas de calcul n° i+1

Remarque : Ce mode de calcul est lié à la structure même du corrélateur, notamment le mécanisme d'accumulation.

Représentons ce mode de calcul par des colonnes de produits. Pour notre représentation prenons comme exemple le calcul d'une fonction d'autocorrélation sur 12 points. Représenté en colonne, nous avons le tableau suivant :

Note : chaque produit élementaire est un produit complexe $a_i\ a_j^*$ dans lequel a_j^* est le conjugué de a_j

. Pour ne pas alourdir inutilement la présentation on écrira simplement $a_i\ a_j$, tout en remarquant bien que

$$a_i\ a_j^* \neq a_j\ a_i^*$$

R(0)	$a_0{*}a_0$	a_1a_1	a_2a_2	a_3a_3	a_4a_4		$a_{10}\ a_{10}$	$a_{11}a_{11}$	
R(1)	$a_0\ a_1$	a_1a_2	a_2a_3	a_3a_4	a_4a_5		$a_{10}\ a_{11}$		
R(2)	$a_0\ a_2$	a_1a_3	a_2a_4	a_3a_5	a_4a_6				
R(3)	$a_0\ a_3$	a_1a_4	a_2a_5	a_3a_6	a_4a_7				
R(4)	$a_0\ a_4$	a_1a_5	a_2a_6	a_3a_7	a_4a_8				
R(5)	$a_0\ a_5$	a_1a_6	a_2a_7	a_3a_8	a_4a_9	...			
R(6)	$a_0\ a_6$	a_1a_7	a_2a_8	a_3a_9	a_4a_{10}				
R(7)	$a_0\ a_7$	a_1a_8	a_2a_9	a_3a_{10}	a_4a_{11}				
R(8)	$a_0\ a_8$	a_1a_9	a_2a_{10}	a_3a_{11}					
R(9)	$a_0\ a_9$	a_1a_{10}	a_2a_{11}						
R(10)	a_0a_{10}	a_1a_{11}							
R(11)	a_0a_{11}								

Tableau 1

1-2 Fonction d'autocorrélation avec sommation de points

Considérons l'exemple le précédent et supposont maintenant que l'on veuille au préalable additionner les échantillons 2 à 2 (fonction R_2) ou 3 à 3 (fonction R_3)

La fonction R_2 obtenue après addition des échantillons 2 à 2 contient les 6 points suivants :

$$R(0)=(a_0+a_1)(a_0+a_1)+(a_2+a_3)(a_2+a_3)+(a_4+a_5)(a_4+a_5)$$
$$+(a_6+a_7)(a_6+a_7)+(a_8+a_9)(a_8+a_9)+(a_{10}+a_{11})(a_{10}+A_{11})$$
$$R(1)=(a_0+a_1)(a_2+a_3)+(a_2+a_3)(a_4+a_5)+(a_4+a_5)(a_6+a_7)$$
$$+(a_6+a_7)(a_8+a_9)+(a_8+a_9)(a_{10}+a_{11})$$

$$R(5)=(a_0+a_1)(a_{10}+a_{11}) \quad \text{(Cf feuille jointe)}$$

$$R(0) = (a_1+a_2+a_3)(a_1+a_2+a_3) + (a_4+a_5+a_6)(a_4+a_5+a_6) + (a_7+a_8+a_9)(a_7+a_8+a_9) + (a_{10}+a_{11}+a_{12})(a_{10}+a_{11}+a_{12})$$

$$R(1) = (a_1+a_2+a_3)(a_4+a_5+a_6) + (a_4+a_5+a_6)(a_7+a_8+a_9) + (a_7+a_8+a_9)(a_{10}+a_{11}+a_{12})$$

$$R(2) = (a_1+a_2+a_3)(a_7+a_8+a_9) + (a_4+a_5+a_6)(a_{10}+a_{11}+a_{12})$$

$$R(3) = (a_1+a_2+a_3)(a_{10}+a_{11}+a_{12})$$

$$R(0) = a_1a_1 + a_1a_2 + a_1a_3 + a_2a_1 + a_2a_2 + a_2a_3 + a_3a_1 + a_3a_2 + a_3a_3 + a_4a_4 + a_4a_5 + a_4a_6 + a_5a_4 + a_5a_5 + a_5a_6$$
$$+ a_6a_4 + a_6a_5 + a_6a_6 + a_7a_7 + a_7a_8 + a_7a_9 + a_8a_7 + a_8a_8 + a_8a_9 + a_9a_7 + a_9a_8 + a_9a_9 + a_{10}a_{10} + a_{10}a_{11} + a_{10}a_{12}$$
$$+ a_{11}a_{10} + a_{11}a_{11} + a_{11}a_{12} + a_{12}a_{10} + a_{12}a_{11} + a_{12}a_{12}$$

$$R(1) = a_1a_4 + a_1a_5 + a_1a_6 + a_2a_4 + a_2a_5 + a_2a_6 + a_3a_4 + a_3a_5 + a_3a_6 + a_4a_7 + a_4a_8 + a_4a_9 + a_5a_7 + a_5a_8 + a_5a_9$$
$$+ a_6a_7 + a_6a_8 + a_6a_9 + a_7a_{10} + a_7a_{11} + a_7a_{12} + a_8a_{10} + a_8a_{11} + a_8a_{12} + a_9a_{10} + a_9a_{11} + a_9a_{12}$$

$$R(2) = a_1a_7 + a_1a_8 + a_1a_9 + a_2a_7 + a_2a_8 + a_2a_9 + a_3a_7 + a_3a_8 + a_3a_9 + a_4a_{10} + a_4a_{11} + a_4a_{12} + a_5a_{10} + a_5a_{11} + a_5a_{12}$$
$$+ a_6a_{10} + a_6a_{11} + a_6a_{12}$$

$$R(3) = a_1a_{10} + a_1a_{11} + a_1a_{12} + a_2a_{10} + a_2a_{11} + a_2a_{12} + a_3a_{10} + a_3a_{11} + a_3a_{12}$$

$$R(0) = (a_1+a_2)(a_1+a_2) + (a_3+a_4)(a_3+a_4) + (a_5+a_6)(a_5+a_6) + (a_7+a_8)(a_7+a_8) + (a_9+a_{10})(a_9+a_{10}) + (a_{11}+a_{12})(a_{11}+a_{12})$$

$$R(1) = (a_1+a_2)(a_3+a_4) + (a_3+a_4)(a_5+a_6) + (a_5+a_6)(a_7+a_8) + (a_7+a_8)(a_9+a_{10}) + (a_9+a_{10})(a_{11}+a_{12})$$

$$R(2) = (a_1+a_2)(a_5+a_6) + (a_3+a_4)(a_7+a_8) + (a_5+a_6)(a_9+a_{10}) + (a_7+a_8)(a_{11}+a_{12})$$

$$R(3) = (a_1+a_2)(a_7+a_8) + (a_3+a_4)(a_9+a_{10}) + (a_5+a_6)(a_{11}+a_{12})$$

$$R(4) = (a_1+a_2)(a_9+a_{10}) + (a_3+a_4)(a_{11}+a_{12})$$

$$R(5) = (a_1+a_2)(a_{11}+a_{12})$$

Soit en développant :

$$R(0) = \underline{a_1a_1 + a_1a_2} + a_2a_1 + a_2a_2 + \underline{a_3a_3 + a_3a_4} + a_4a_3 + a_4a_4 + \underline{a_5a_5 + a_5a_6} + a_6a_5 + a_6a_6 + \underline{a_7a_7 + a_7a_8} + a_8a_7 + a_8a_8$$
$$+ \underline{a_9a_9 + a_9a_{10}} + a_{10}a_9 + a_{10}a_{10} + \underline{a_{11}a_{11} + a_{11}a_{12}} + a_{12}a_{11} + a_{12}a_{12}$$

$$R(1) = \underline{a_1a_3 + a_1a_4} + a_2a_3 + a_2a_4 + \underline{a_3a_5 + a_3a_6} + a_4a_5 + a_4a_6 + \underline{a_5a_7 + a_5a_8} + a_6a_7 + a_6a_8 + \underline{a_7a_9 + a_7a_{10}} + a_8a_9 + a_8a_{10}$$
$$+ \underline{a_9a_{11} + a_9a_{12}} + a_{10}a_{11} + a_{10}a_{12}$$

$$R(2) = \underline{a_1a_5 + a_1a_6} + a_2a_5 + a_2a_6 + \underline{a_3a_7 + a_3a_8} + a_4a_7 + a_4a_8 + \underline{a_5a_9 + a_5a_{10}} + a_6a_9 + a_6a_{10} + \underline{a_7a_{11} + a_7a_{12}} + a_8a_{11} + a_8a_{12}$$

$$R(3) = \underline{a_1a_7 + a_1a_8} + a_2a_7 + a_2a_8 + \underline{a_3a_9 + a_3a_{10}} + a_4a_9 + a_4a_{10} + \underline{a_5a_{11} + a_5a_{12}} + a_6a_{11} + a_6a_{12}$$

$$R(4) = \underline{a_1a_9 + a_1a_{10}} + a_2a_9 + a_2a_{10} + \underline{a_3a_{11} + a_3a_{12}} + a_4a_{11} + a_4a_{12}$$

$$R(5) = \underline{a_1a_{11} + a_1a_{12}} + a_2a_{11} + a_2a_{12}$$

La fonction de corrélation R_3 obtenue par addition des échantillons 3 à 3 comprend les 4 points suivants :

$$R(0) = (a_o+a_1+a_2)(a_o+a_1+a_2)+(a_3+a_4+a_5)(a_3+a_4+a_5) + (a_6+a_7+a_8)(a_6+a_7+a_8)+(a_9+a_{10}+a_{11})(a_9+a_{10}+a_{11})$$

$$R(1) = (a_o+a_1+a_2)(a_3+a_4+a_5)+(a_3+a_4+a_5)(a_6+a_7+a_8) + (a_6+a_7+a_8)(a_9+a_{10}+a_{11})$$

$$R(2) = (a_o+a_1+a_2)(a_6+a_7+a_8)+(a_3+a_4+a_5)(a_9+a_{10}+a_{11})$$

$$R(3) = (a_o+a_1+a_2)(a_9+a_{10}+a_{11})$$

Développons ces calculs, et comme pour la fonction de corrélation simple, représentons les en colonnes.

1-2-1 Représentation de la fonction R2

$$R(0) = a_oa_o+a_oa_1+a_2a_o+a_1a_1 + a_2a_2+a_2a_3+a_3a_2+a_3a_3 + a_4a_4+a_4a_5+a_5a_4+a_5a_5 + a_6a_6+a_6a_7+a_7a_6+a_7a_7 + a_8a_8+a_8a_9+a_9a_8+a_9a_9 + a_{10}a_{10}+a_{10}a_{11}+a_{11}a_{10}+a_{11}a_{11}$$

$$R(5) = a_oa_{10}+a_oa_{11}+a_1a_{10}+a_1a_{11}$$

Cf feuille jointe (p 67,68)

Représentons ce calcul en colonne

	a_1a_0		a_3a_2		a_5a_4		a_7a_6		$a_{11}a_{10}$
a_0 a_0	a_1a_1	a_2a_2	a_3a_3	a_4a_4	a_5a_5	a_6a_6	a_7a_7	$a_{10}a_{10}$	$a_{11}a_{11}$
a_0 a_1	a_1a_2	a_2a_3	a_3a_4	a_4a_5	a_5a_6	a_6a_7	a_7a_8	$a_{10}a_{11}$	
a_0 a_2	a_1a_3	a_2a_4	a_3a_5	a_4a_6	a_5a_7	a_6a_8	a_7a_9		
a_0 a_3	a_1a_4	a_2a_5	a_3a_6	a_4a_7	a_5a_8	a_6a_9	a_7a_{10}		
a_0 a_4	a_1a_5	a_2a_6	a_3a_7	a_4a_8	a_5a_9	a_6a_{10}	a_7a_{11}		
a_0 a_5	a_1a_6	a_2a_7	a_3a_8	a_4a_9	a_5a_{10}	a_6a_{11}			
a_0 a_6	a_1a_7	a_2a_8	a_3a_9	a_4a_{10}	a_5a_{11}				
a_0 a_7	a_1a_8	a_2a_9	a_3a_{10}	a_4a_{11}					
a_0 a_8	a_1a_9	a_2a_{10}	a_3a_{11}						
a_0 a_9	a_1a_{10}	a_2a_{11}							
a_0a_{10}	a_1a_{11}								
a_0a_{11}									

Tableau 2

1-2-2 Représentation de la fonction R3

Ces points de la fonction de corrélation R3 comprendrons les éléments suivants :

$$R(0) = a_0a_0+a_0a_1+a_0a_2+a_1a_0+a_1a_1+a_1a_2+a_2a_0+a_2a_1$$
$$+ a_2a_2+a_3a_3+a_3a_4+a_3a_5+a_4a_3+a_4a_4+a_4a_5+a_5a_3$$
$$+ a_5a_4+a_5a_5+a_6a_6+a_6a_7+a_6a_8+a_7a_6+a_7a_7+a_7a_8$$
$$+ a_8a_6+a_8a_7+a_8a_8+a_9a_9+a_9a_{10}+a_9a_{11}+a_{10}a_9+a_{10}a_{10}$$
$$+ a_{10}a_{11}+a_{11}a_9+a_{11}a_{10}+a_{11}a_{11}$$

Cf feuille jointe (p 67, 68)

$R(3)=a_0a_9+a_0a_{10}+a_0a_{11}+a_1a_9+a_1a_{10}+a_1a_{11}+a_2a_9+a_2a_{10}+a_2a_{11}$

Ce qui donne le tableau suivant

		a_2a_0			a_5a_3			a_{11} 9
	a_1a_0	a_2a_1		a_4a_3	a_5a_4		$a_{10}a_9$	$a_{11}a_{10}$
a_0 a_0	a_1a_1	a_2a_2	a_3a_3	a_4a_4	a_5a_5	a_9a_9	$a_{10}a_{10}$	$a_{11}a_{11}$
a_0 a_1	a_1a_2	a_2a_3	a_3a_4	a_4a_5	a_5a_6	a_9a_{10}	$a_{10}a_{11}$	
a_0 a_2	a_1a_3	a_2a_4	a_3a_5	a_4a_6	a_5a_7	a_9a_{11}		
a_0 a_3	a_1a_4	a_2a_5	a_3a_6	a_4a_7	a_5a_8			
a_0 a_4	a_1a_5	a_2a_6	a_3a_7	a_4a_8	a_5a_9			
a_0 a_5	a_1a_6	a_2a_7	a_3a_8	a_4a_9	a_5a_{10}			
a_0 a_6	a_1a_7	a_2a_8	a_3a_9	a_4a_{10}	a_5a_{11}			
a_0 a_7	a_1a_8	a_2a_9	a_3a_{10}	a_4a_{11}				
a_8	a_1a_9	a_2a_{10}	a_3a_{11}					
a_9	a_1a_{10}	a_2a_{11}						
a_0a_{10}	a_1a_{11}							
a_0a_{11}								

Tableau 3

On constate que tous ces tableaux sont construits de la même manière. Dans chaque colonne il y a un terme constant qui est multiplié par tous les autres échantillons possibles, pris par ordre croissant. Par convertion on appellera ce terme constant l'opérande fixe.

Si on compare les tableaux 2 et 3 avec le premier qui correspond a fonction de corrélation sur 12 points sans addition, on constate qu'à des problèmes de rangement près et de produits croisés a_ia_j ; a_ja_i, le mode de calcul de la fonction de corrélation simple doit permettre de résoudre le problème.

1-3 Présentation de l'algorithme

Soit n le nombre de point à sommer.
Soit N_e le nombre d'émission (il est bien entendu que N_e est divisible par n)

L'idée de l'algorithme est :

-de compléter les colonnes avec les éléments manquants (produits croisés) et de les traiter par "paquet" de n

-de diviser chaque colonne en groupe de n éléments.

1-3-1 Traitement par groupe de colonnes

1-3-1-1 Tout d'abord, pour additionner n échantillons on va considérer les n colonnes consécutives qui les contiennent. C'est la première organisation que l'on fait sur le tableau représentatif du calcul. Il y aura N_e/n groupes de colonnes correspondant aux N_e/n points de la nouvelle fonction de corrélation à calculer.

1-3-1-2 Dans la représentation par colonne de l'algorithme de corrélation simple, le tableau des calculs est triangulaire. La première colonne a autant d'éléments qu'il y a d'émissions et la dernière n'en a qu'un seul. D'une colonne à l'autre la différence est d'un élément. Dans chaque colonne élémentaire il y a un opérande fixe a_i devant lequel on fait défiler tous les points possibles depuis i jusqu'à n ce qui ne permet pas d'avoir les produits croisés. Pour les avoir il faut compléter les colonnes avec les points manquant. Pour compléter les colonnes la régle est la suivante.

Soit a_i l'opérande fixe, i étant le numèro de la colonne élémentaire (colonnes numérotées de 1 à N_e). Si k est le reste de la division entière de (i-1) par n, k est le nombre d'éléments à rajouter à la colonne i. Dans chaque colonne élémentaire on rajoutera les k éléments précédant l'opérande fixe.

Par exemple, pour additionner les échantillons 3 à 3, on complétera les colonnes 2 et 3 par les éléments suivants :

	$[a_1a_0]$	$\begin{bmatrix} a_2a_0 \\ a_2a_1 \end{bmatrix}$
a_0a_0	a_1a_1	$a2a_2$
a_0a_1	ala_2	
a_0a_2		

1-3-2 Traitement à l'intérieur des colonnes

Une fois que chaque colonne aura été complétée, elle contient tous les éléments, produits croisés compris, nécessaires au calcul de la fonction de corrélation avec sommation de points. Puisqu'elles ont été complétées, chaque colonne comprend maintenant un nombre d'éléments divisibles par n. Tous les n éléments consécutifs faisant partie des produits élémentaires de la fonction à calculer seront à additionner ensemble dans la result memory, et cela pour toutes les colonnes (accumulation des résultats intermédiaires).

1-3-3 Formalisation de l'algorithme

De manière plus formalisée, l'algorithme proposé est le suivant :
Soit :

NEMISS	Le nombre d'émissions
NADD	Le nombre de points à additionner
FIXE	L'adresse de l'opérande fixe
MOBILE	L'adresse de l'opérande mobile (qui défile devant l'opérande fixe)
ADR	L'adresse de rangement du produit élémentaire aij
R	Tableau contenant la partie réelle de la fonction de corrélation
IM	Tableau contenant la partie imaginaire de la fonction de corrélation
X	Tableau contenant la partie réelle des échantillons
Y	Tableau contenant la partie imaginaire des échantillons

(Toutes les variables et tous les tableaux sont entiers).

```
 N = NEMISS / NADD
DO    FOR I=1,N
      DO  FOR  J = 1, NADD
      FIXE = (I-1) * NADD + j
      MOBILE = (I-1) * NADD + 1
      ADR = 0 ; IAD = 0
      DO FOR  L = MOBILE , NEMISS
      IF (IAD.EQ.(IAD/NADD)*NADD)ADR=ADR+1
      XF = X(FIXE) ; YF = Y (FIXE)
      XM = X (L) ; YM = Y (L)
      R (ADR) = R(ADR)+XF * XM + YF * YM
      IM(ADR)= IM (ADR) - XF * YM + YF * XM
      IAD = IAD +1
    ENDDO
   ENDOO
  ENDOO
```

2 - PROGRAMMATION SUR LE CORRELATEUR

L'algorithme exposé précédemment représente l'idée de base du programme à mettre en place sur le corrélateur. Pour que le programme de calcul de la fonction de corrélation avec addition des échantillons soit général, on va lui donner une structure de sous-programme dans lequel :

-le nombre de points à additionner
-le nombre d'altitudes à traiter

sont des paramètres.

Dans la pratique, chaque émission sera faite successivement à différentes fréquences, et on acqueirra des échantillons provenant de plusieurs altitudes. De plus, à la fin de chaque cycle de réception on acqueirra des échantillons de bruit et de calibration.

Enfin, dans l'application qu'il s'agit de construire, on va calculer la moyenne et la puissance du bruit et de la calibration pour chaque fréquence. Ce calcul est fait par un sous-programme qui existe déjà, et qu'il suffit d'implanter.

Le schéma du programme complet implanté dans le corrélateur sera le suivant :

```
DEBUT    :    Attente du signal de début de calcul
  |       DO   FOR   NF = 1, NBFREQ
  |       |    CALL  ADD
  |       |    CALL  POWER
  |      ENDDO
  |      GOTO   DEBUT
  |      DMA TRANSFERT
 FIN
```

Dans lequel :

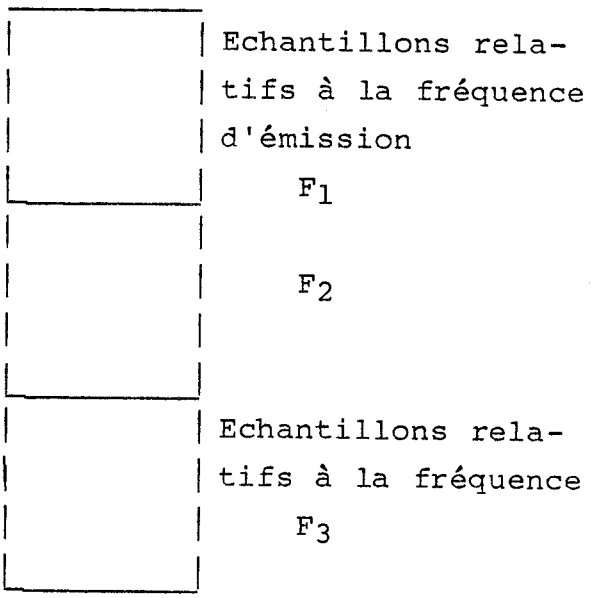
NBFREQ est le nombre de fréquences à traiter
ADD est le sous-programme de corrélation avec addition
POWER est le sous-programme de calcul de la puissance du bruit
DMATRANSFERT est le programme de transfert de la result memory vers le calculateur (il est lancé par le radar controller au bout d'un certain nombre d'expériences - ce nombre est un paramètre du programme de gestion du radar controller).

2-1 Programmation du sous-programme de corrélation avec addition d'échantillons.

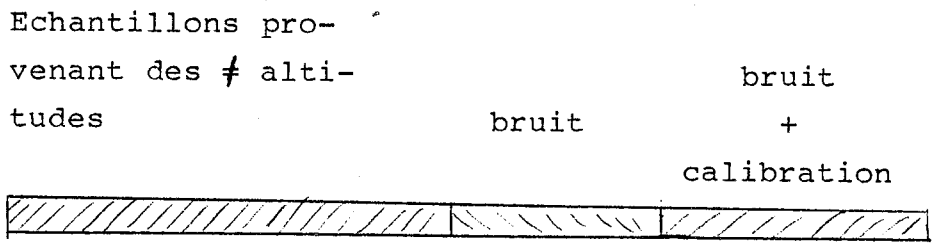
Remarques préliminaires

1 - Comme il a été dit au début de ce rapport, un micro-programme dépend de la structure du buffer d'entrée (programmation de l'APB-processeur). Les émissions sont faites à différentes fréquences. Le buffer d'entrée est partitionné en autant de zones qu'il y a de fréquences et les échantillons sont rangés sequentiellement, fréquence par fréquence.

Cette position est déterminée par le programme de gestion du radar controller. Pour notre application on imposera que les partitions soient jointives (c'est à dire qu'il n'y ait pas de trou dans le buffer d'entrée)



Pour une fréquence donnée, le rangement des échantillons de signal, bruit, bruit + calibration est identique pour chaque émission



Echantillons provenant de l'émission i

Pour chaque fréquence, les échantillons provenant des différentes émissions sont rangés sequentiellement. S'il y a eu n émissions on aura l'organisation suivante :

émission 1 émission i émission n

2 - Le programme doit être codé selon les 5 champs programmables du corrélateur (champ programme, calcul d'adresse dans le buffer d'entrée et de sortie, opérations arithmétiques, accumulation). Le transfert DMA fait l'objet d'un programme à part et le champ I/O n'est pas utilisé.

Tous ces champs sont exécutés en parallèle et il est évident qu'ils doivent être bien synchronisés. C'est pourquoi, on sera amené à programmer des NOOP pour assurer la synhronisation.

3 - Vu le mécanisme d'accumulation du corrélateur et le jeu de micro-instructions dont on dispose, l'algorithme exposé précédemment ne peut pas être codé tel quel. Comme il a déjà été signalé, la result memory du corrélateur n'est jamais remise à zéro. Pour l'initialiser, le calcul sera mené de la manière suivante :

-calcul du premier produit $a_i * a^*_j$ de chacun des points de la fonction de corrélation et écriture de chacun de ces produits dans la result memory sans accumulation

-calcul de tous les autres éléments et rangement dans la result mémory avec accumulation.

4 - On a déjà vu que le corrélateur ne possède que 3 compteurs de boucle. Si on veut tester à zéro une variable, il faut obligatoirement qu'elle soit contenue dans un de ces 3 compteurs. Or, il faut que l'on prenne en compte :

-un nombre de fréquences
-un nombre de points à additionner
-un nombre d'altitudes
-un nombre d'émissions

Pour au moins une d'entre elles, il faudra la décrementer dans l'APB-STACK (opération : RS(B)=RS (B)-1) puis la charger dans un compteur de boucle par un RELOAD et enfin tester ce compteur de boucle, ce qui est très coûteux en place dans la mémoire de programme.

De plus, le programme doit être réexécutable (CONTINUE EXPERIMENT). Il ne faut donc pas qu'une exécution détruise les paramètres de l'expérience.
Exemple :

PRO	APB
NOOP	SAUVE = NBFREQ
Boucle : Reload LCR1	SAUVE présenté sur les lignes d'APB
NOOP	NOOP
LC1 = LCR1	SAUVE = SAUVE-1
IF(LC1 = 0) THEN ...	

Cet exemple montre bien que la multiplicité des variables

-rallonge le micro-programme
-encombre l'APB-STACK (variable + valeur de sauvegarde)

Or, on ne dispose que de 63 mémoires programmables et 16 places dans l'APB-STACK.

2-1-1 Présentation des variables du programme

L'APB-STACK contient 15 paramètres et variables et l'APM-STACK n'en contient que 3.

L'APB-STACK est utilisée de la manière suivante :

RS(0) : Libre

RS(1), RS(2), RS(3) : Variables de travail

RS(4) : Sauvegarde du nombre de fréquences

RS(5) : Nombre de fréquences

RS(6) : Nombre d'émission -1

RS(7) : Nombre d'échantillons de bruit -1 et nombre d'échantillons de calibration -1

RS(8) : Increment 1 (initialisé à 1)

RS(9) : Increment 2 (sample increment + nombre de points à additionner)

RS(10) : Compteur (identique à RS(14))

RS(11) : Recalage (sample increment + nombre d'émissions -1

RS(12) : Sample increment (nombre de altitude + nombre d'échantillons de calibration)

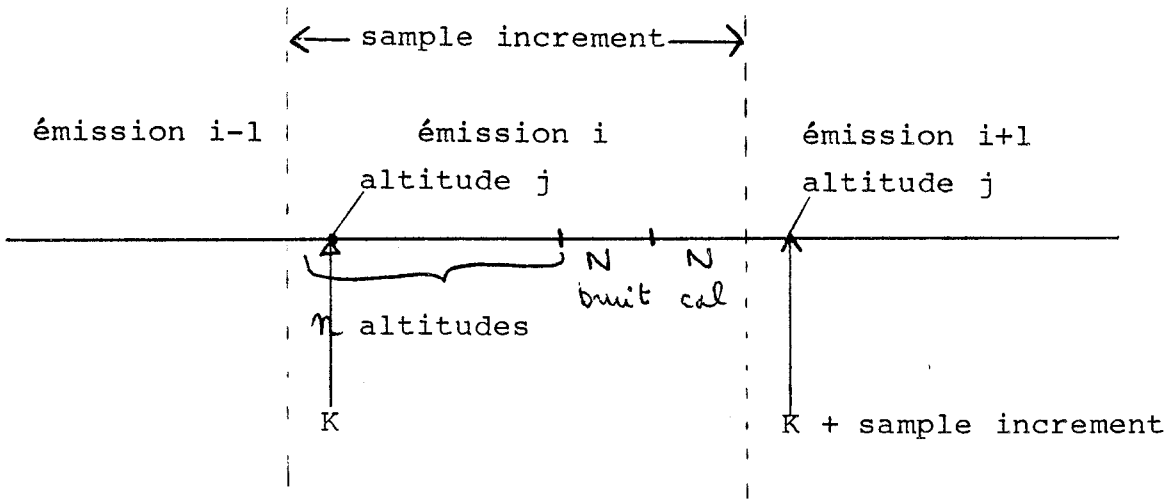
RS(13) : Nombre d'altitudes -1

RS(14) : Nombre d'émissions / Nombre de points à additionner -1

RS(15) : Nombre de points à additionner

2-1-1-1 Signification des variables de l'APB-STACK

Le sample increment est le nombre d'échantillons de toute nature contenu dans une émission. Il représente "l'encombrement" d'une émission dans le buffer memory



Si, dans le buffer memory, K est l'adresse de l'échantillon de l'altitude j, pris à la réception de l'émission i, K + sample increment est l'adresse, dans le buffer memory de l'échantillon, de l'altitude j pris à la réception de l'émission i+1

-Increment 2

L'increment 2, égal à sample increment + nombre de points à additionner, est lié au fonctionnement même de l'algorithme transposé sur le corrélateur. A cause des contraintes dues au mécanisme d'accumulation, chaque calcul élémentaire est rangé à une adresse différente dans la result memory.

Supposons que l'on veuille sommer les échantillons 3 à 3 et considérons une colonne. Soit a_i l'opérande fixe. On a les produits :

$a_i * a_{j-1}$
$a_i * a_j$
$a_i * a_{j+1}$
$a_i * a_{j+2}$
$a_i * a_{j+3}$

Les 3 produits $a_i a_j$, $a_i a_{j+1}$, $a_i a_{j+2}$ doivent tous trois être accumulés à la même adresse. L'algorithme le fera, mais en 3 temps. A chaque pas élémentaire, l'adresse de l'opérande mobile est incrémenté d'un pas de 3 (dans notre exemple).

Après avoir calculé $a_i * a_j$, le prochain calcul portera sur $a_i * a_{j+3}$. Ce mode de calcul est imposé par la nécessité d'initialiser la result memory avant de commencer l'accumulation. Mais dans l'exemple ci-dessus, les échantillons à traiter étaient consécutifs, or dans la réalité, on traite plusieurs altitudes. Dans ce cas l'increment 2 et le déplacement dans le buffer memory qui permet, à partir d'un échantillon d'une altitude de l'emission i, d'accéder à un autre échantillon de la même altitude provenant de l'émission i+k, k étant le nombre de points à additionner.

-Compteur

Le calcul s'effectue colonne par colonne. Pour chaque colonne il y a autant de passe que de points à additionner. Compteur qui est initialisé avec (nombre d'émissions /nombre de points à sommer)-1 gére la boucle de calcul élémentaire sur une colonne. Cette boucle élémentaire est répétée par colonne et sur autant de colonnes qu'il y a de points à additionner.

Si nous avons K émission et N points à additionner, toutes les K/N colonnes, la valeur initiale de compteur et décrémentée de 1, ce qui oblige de décrémenter la variable compteur dans l'APB-STACK et de programmer un "Reload" d'un loop counter.

Enfin, le sous-programme devant être réexecutable, la valeur initiale de compteur est sauvegardée dans RS(14) de l'APB-STACK et la variable compteur est rechargée avant chaque appel.

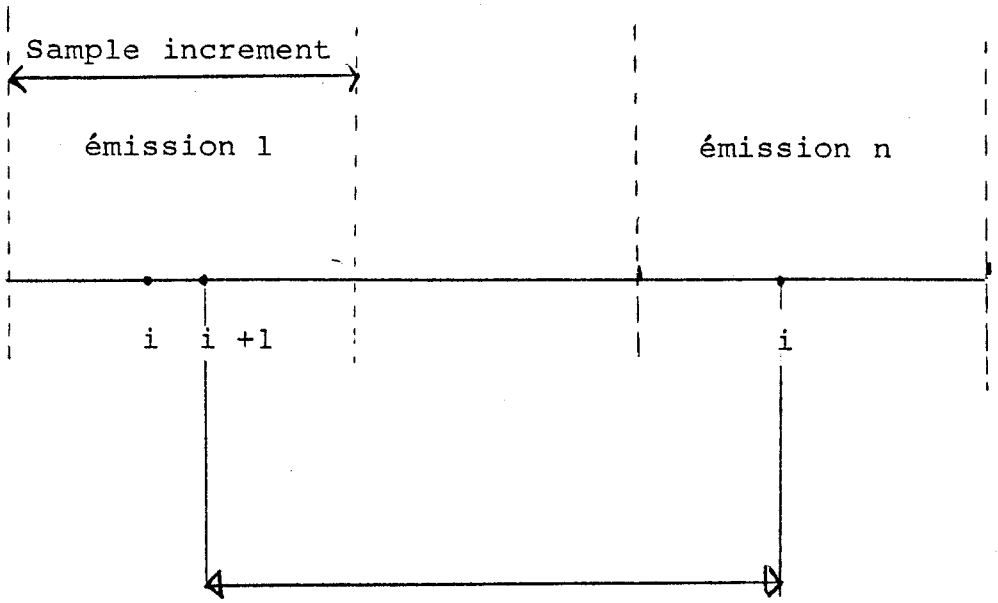
-Recalage

Le registre Q contient en permanence l'adresse de l'operande fixe. Il est incrémenté du sample increment de colonne en colonne.

Pour une altitude i, Q est initialisé avec l'adresse dans le buffer memory de l'échantillon provenant de l'altitude i pour la première émission, contient à la fin du calcul de la fonction de corrélation pour l'altitude i, l'adresse de l'échantillon de l'altitude i provenant de la dernière émission.

Le calcul devant être recommencé pour l'altitude i+1, le recalage est la distance qui sépare l'échantillon venant de l'altitude i à la dernière émission de celui provenant de l'altitude i+1 pour la première émission.

Le calcul devant être recommencé pour l'altitude i+1, le recalage est la distance qui sépare l'échantillon venant de l'altitude i à la dernière émission de celui provenant de l'altitude i+1 pour la première émission.



Recalage = Sample increment * Nb émission -1

La signification des autres variables de l'APB-STACK est évidente.

Remarques : Il faut noter que l'APB-STACK ne comporte qu'une seule place libre, ce qui a entrainé une certaine rigidité et même certains compromis.

-Il y a autant d'échantillons de bruit que d'échantillons de bruit + calibration, ce qui n'est pas une obligation physique.

-Dans l'expérience qui est décrite, la place des échantillons des différentes fréquences est figée dans le buffer d'entrée : ils doivent être consécutifs, c'est à dire que si K est l'adresse du dernier échantillon pris pour la fréquence F_1, K+1 est le premier échantillon pris pour la fréquence F_2.

La place libre restante pourra, en cas de besoin lever une de ces deux contraintes.

L'APM-STACK est utilisée de la manière suivante :

RS(0) : increment initialisé à 1
RS(13): variable de travail
RS(14): increment initialisé à 1
RS(15): rangegate increment (nombre d'émissions/nbre de points à additionner)

Toutes les autres variables de l'APM-STACK sont inutilisées.

2-1-1-2 Signification des variables de l'APM-STACK

-RS(0) : Est l'increment qui sert au programme de DMA transfert. Par convertion, ce programme utilise toujours la variable RS(0).

-RS(14): Increment utilisé pour la gestion de la result memory. Vu que l'APM-STACK n'est pas encombrée, on l'a volontairement différenciée de l'increment utilisé par le DMA transfert.

-RS(15): Initialisé avec le nombre d'émissions divisé par le nombre de points à additionner, RS(15) représente le nombre de points de la fonction de corrélation calculé pour chaque émission.

2-1-1-3 Autres unitialisations :

START-ADDRES-REGISTER : ce registre est initialisé à 1. Dès que la corrélation reçoit l'ordre de commencer le calcul, il exécute l'instruction n°1.

DATA-I-REGISTER : ce registre contient le nombre de valeurs calculées à chaque cycle. Soit :
NB fréquence * (nombre d'altitudes * rangegate increment + Nbre points power)+1
1 correspond au scan court.

	PRϕ - INSTRUCTIONS	APB - INSTRUCTIONS	ARM - INSTRUCTIONS	ARI - INSTRUCTIONS	ACC - INSTRUCTIONS	
1	DEB Reload LCR2	ϕUT = RS(14) [No comm / Nophs − 1]				1
2	NϕϕP					2
3	LC1 = LCR1 ; LC2 = LCR2	TS1 = Q		charge multiplieur H	clear FF1	3
4	NϕϕP	TS2 = TS1				4
5	LCR1A = LC1 ; LC1 = LC1 − 1 LC2 = LC2 − 1	Ø courante = TS1	Ø courante = Q	charge multiplieur B ; calcul	Read = YES ; WRITE = YES	5
6	N ϕ LC2 = LC2 − 1 ; LC2 = 0 ? ; YES	Ø courante = Ø courante + INC2	Ø courante = Ø courante + increment	charge multiplieur B ; calcul	Read = YES ; Write = YES	6
7	E1 LC2 = LCR2	TS2 = TS2 + Sample increment			Set FF1	7
8	LC2 = 0 ? YES ; N ϕ LC2 = LC2 − 1	Ø courante = TS2	Ø courante = Q	charge multiplieur B ; calcul	Read = YES ; Write = YES	8
9	N ϕ LC2 = LC2 − 1 ; LC2 = 0 ? YES	Ø courante = Ø courante + INC2	Ø courante = Ø courante + increment	charge multiplieur B ; calcul	Read = YES ; Write = YES	9
10	LC2 = LCR2 ; N ϕ LC1 = LC1 − 1 ; LC1 = 0 ? YES					10
11	LC1 = LCR1A	Q = Q + Sample increment		charge multiplieur H		11
12	LC1 = 0 ? YES ; N ϕ LC1 = LC1 − 1	TS2 = TS1				12
13	LCR1A = LC1 ; LC1 = LCR1	TS2 = TS2 − Sample increment				13
14	LC1 = LCR1 ; LC2 = LCR2	COMPTEUR = COMPTEUR − 1				14
15	LC2 = 0 ? YES ; N ϕ					15
16	Reload LCR2	ϕUT = RS(10) [COMPTEUR]				16
17	NϕϕP					17
18	LCR1A = LC1 ; LC2 = LCR2	TS1 = Q				18
19	GϕTϕ E1	TS2 = TS2 − Sample increment				19
20	LC3 = 0 ? YES RETURN ; N ϕ LC3 = LC3 − 1	COMPTEUR = RS(14)	Q = Q + remplissage increment			20
21	GϕTϕ DEB	Q = Q − RS(11) [Recule x]				21

2-1-2 Commentaires sur l'organigramme du micro-programme

Seules les PRO et les APB instructions nécessitent une explication, la signification des autres champs étant, somme toute, évidente.

2-1-2-1 Les PRO-Instructions

Tous les compteurs sont initialisés ici avec le nombre d'opérations à compter -1. Ceci est du au fait que l'on a programmé à la fois un test à 0 et une décrementation, or, le test porte sur la valeur avant décrementation.

. Le loop counter 3 LC3

Initialisé avec NBALT-1, LC3 a un rôle simple, c'est le compteur d'altitudes. Il reboucle l'algorithme pour faire calculer la fonction de corrélation pour chaque altitude.

. Le loop counter 2 LC2

Initialisé avec NBEMISS/nombre de points à additionner -1. LC2 compte les points pris pour le calcul dans chaque colonne (cf rôle de la variable compteur qui sert à "reloader" LC2 pour chaque nouveau groupe de colonnes).

. Le loop counter 1 LC1

Initialisé avec le nombre de points à additionner -1, LC1 a un double rôle. Il compte :

-le nombre de passes à faire par colonne (chaque colonne est explorée autant de fois qu'il y a de points à additionner)

-le nombre de colonnes à traiter (les colonnes sont traitées par "paquet", chaque groupe contenant autant de colonnes élémentaires qu'il y a de points à additionner).

Le rôle de LC1 compteur de passes sur une colonne est simple, par contre, pour compter les colonnes à traiter on utilise le registre LCR1A, sauvegarde de LC1. LCR1A contient le nombre de colonnes du paquet qu'il reste à traiter. A la fin du traitement d'une colonne, LC1 est rechargé avec LCR1A, puis, il est testé à zéro et décrementé ensuite. Si le test est non nul, la nouvelle valeur de LC1 est sauvegardée dans LCR1A, tandis que LC1 est rechargé avec sa valeur initiale LCR1.

. L'utilisation de la variable COMPTEUR. D'un groupe de colonnes à l'autre, il y a n éléments en moins (n étant le nombre de points à additionner). Vu que chaque colonne est parcourue n fois, d'un groupe de colonnes à l'autre, il y a par passage, un point de moins à prendre. Le nombre de points utilisés par colonne est géré par LC2. Il faut donc que d'un groupe de colonnes à l'autre, LC2 soit initialisé (reloadé) avec des valeurs décroissantes. C'est le rôle de la variable compteur qui est décrementé après le traitement d'un groupe de colonnes et avec lequel LC2 est "reloadé".

2-1-2-2 Les APB instructions

. Le registre Q contient en permanence l'adresse de l'operande fixe. A chaque changement de colonne, Q est incrementé du sample increment.

. TS1 : pour chaque groupe de colonnes, l'operande mobile est initialisé avec la même valeur : celle de l'operande fixe de la première colonne du groupe (cf régle de complément des colonnes). TS1 contient l'adresse de l'operande mobile par lequel il faut commencer le calcul pour une colonne. TS1 est initialisé avec la valeur du registre Q.

. TS2 : contient l'adresse du premier operande mobile a prendre en compte pour chaque passe de calcul dans une colonne. TS2 est initialisé avec TS1 à chaque début de calcul sur une nouvelle colonne et est incrementé du sample increment à chaque passe sur la même colonne.

. Adresse courante mobile. C'est l'adresse de l'operande mobile pendant le calcul sur une colonne. Elle est incrementée par INCREMENT 2 (sample increment * Nbre de points à additionner).

2-2 Description du programme complet de l'application

L'expérience projetée comporte :

-la corrélation des échantillons additionnés n à n, pour toutes les fréquences,

-le calcul de la puissance du bruit, également pour toutes les fréquences.

Grossièrement, le programme complet n'a donc à gérer que les appels à ces deux sous-programmes. Il devra les appeler pour toutes les fréquence émises (boucle sur les fréquences). Il devra en outre comporter le code de la subroutine de transfert DMA, l'activation de celle-ci étant faite par le radar controller.

Remarques :

-La subroutine décrite précédemment utilisant les 3 loop-counter disponibles, le programme principal utilisera, pour gérer la boucle sur les fréquences, 2 variables de l'APB-STACK, RS(4) et RS(5).

La première, RS(4) est utilisée comme valeur initiale du nombre de fréquences à traiter, la seconde RS(5) comme variable de travail.

(Rappelons que le programme devant être réexécutable, car ce n'est évidemment qu'après plusieurs accumulations que le transfert vers le calculateur aura lieu, on ne peut pas décrementer directement le nombre de fréquences).

-La subroutine de transfert DMA étant obligatoirement implantée entre les adresses 32 à 44 de la mémoire programme, on a été obligé de couper le programme principal en 2.

-Le corrélateur qui est démarré par le radar controller ne "sait pas" combien de fois le programme sera exécuté. En fin d'expériences, après n accumulations, les données sont transférées au calculateur. De la, elles sont transcrites sur bandes magnétiques en vue de leur dépouillement ultérieur. Or, pour le dépouillement des données, connaître le nombre d'accumulations est necessaire.

Bien sûr, ce renseignement peut être obtenu par différentes voies, ne serait-ce que parce que c'est le calculateur qui programme le radar controller. La solution retenue à EISCAT consiste à faire écrire par le corrélateur, en fin de la result memory, le nombre d'accumulations.

Le mecanisme mis en oeuvre est le suivant :

Chaque fois que le programme du corrélateur est terminé, avant de retourner dans la IDLE LOOP, on décremente le contenu de la première ad. libre de la result memory. Si au cours d'un cycle de calcul le corrélateur écrit N mémoires de la result memory, on fera :

mem(N+1) = mem(N+1)-1

Ainsi, en fin d'expérience, cette dernière mémoire contiendra en negatif le nombre d'accumulations.

La subroutine de corrélation avec addition de points ayant été décrite précédemment et la procédure de calcul de la puissance étant une procédure cataloguée, on se contentera de décrire rapidement l'application complète dans ses deux champs principaux : les PRO-instructions et les APB-insturctions.

Adresse mémoire	PRO-INSTRUCTIONS	APB-INSTRUCTIONS
0	IDLE ∠ØØP	IDLE ∠ØØP
1		RS(4) = RS(5) Sauvegarde du nombre de fréquence
2		Q = 0 Pointe sur le 1er échantillon
3	Reload LCR1	ØUT = RS(4) (Nb de fréquences)
4		
5	LC1 = LCR1	
6	ØUT LC1=0 OUI / NON	RS(4) = RS(4) - 1
7	Reload LCR3	ØUT = RS(13) (Nb altitudes - 1)
8		
9	Reload LCR1	ØUT = RS (15) Nb de points à additionner - 1
10	GØTØ 45	
11 ⋮ 31	Sous-programme de corrélation avec addition des échantillons SUBADD	
32 ⋮ 44	Sous-programme DMA-TRANSFERT	
45	Call SUBADD ; LC3 = LCR3	
46	Reload LCR2	ØUT = RS(7) Nb. de bruits - 1
47		
48	Reload LCR1	ØUT = RS(6) Nb d'émissions - 1
49		
50	call POWER (bruit)	Q = Q - RS(11)
51	call POWER (calibration)	
52	GØTØ 3	
53		
54	GØTØ 0	
55 ⋮ 63	Sous-programme de calcul de la puissance (POWER)	

NOTE : la mise à jour du scan count se fait dans les APM-instructions, à l'instruction 53.

2-3 Tests du programme

L'application complète comprend un sous-programme catalogué et le sous-programme de corrélation avec addition des échantillons. C'est évidemment sur le fonctionnement de cette subroutine que porte l'essentiel des tests.

2-3-1 Tests graphiques

Les premières vérifications sont faites avec la routine graphique du simulateur. On sait de manière théorique que la corrélation d'un sinus par lui-même est un cosinus multiplié par un triangle.

Si, pour deux mêmes altitudes a_i de deux émissions consécutives on met la même valeur (cf possibilité de redoubler les valeurs des échantillons dans le simulateur), la somme des deux sera toujours un sinus, mais d'amplitude double. On doit donc avoir un cosinus multiplié par un triangle comme fonction de corrélation. D'autre part, si pour tous les échantillons de chaque émission on met la même valeur, il est évident qu'en sortie on doit avoir exactement les mêmes fonctions de corrélation. Un certain nombre de tests ont été fait et tracés sur le Benson.

Ci-joint en annexe le résultat d'une simulation portant sur

1000 émissions
4 altitudes
0 bruit et 0 calibration

dans laquelle les points sont additionnés deux à deux et les échantillons représentant chaque altitude sont identiques. On obtient bien quatre courbes identiques ayant la forme attendue. Ce type de test négatif n'est pas inutile, car il permet tout de suite de détecter des erreurs d'adressage du buffer d'entrée ou de la result memory.

2-3-2 Tests arithmétiques

Comme il a déjà été dit, la meilleure manière de vérifier le fonctionnement du programme c'est de comparer les résultats obtenus avec les résultats prévus. Cependant cette dernière vérification demande à chaque fois d'écrire un programme de calcul spécifique, c'est pourquoi les premières vérifications sont faites avec la routine graphique du simulateur.

Dans le buffer d'entrée simulé, les échantillons sont de la forme

IX(I) = 100 * Sin (2PI* N(I-1)/NVAL)
IY(I) = 100 * cos (2PI* N(I-1)/NVAL)
(représentant respectivement les parties réelles et imaginaires du signal simulé)

où : N est le nombre de période de la sinusoïde
NVAL = NBPTS/NR
NBPTS étant le nombre de points contenus dans le buffer d'entrée (max 600)
NB étant le nombre de répétitions de points
I varie de 1 à NVAL

Le buffer d'entrée contient "un escalier sinusoïdal", la "largeur des marches" étant NR. D'autre part, si pour une simulation on a besoin de plus de NBPTS points, les valeurs sont répétées modulo NBPTS.

Le but des tests arithmétiques est de comparer valeur à valeur les résultats simulés avec ceux obtenus directement par un programme fortran. Deux cas ont été envisagés, l'addition de deux ou trois échantillons.

2-3-2-1 Addition de deux échantillons

Si on additionne les points deux à deux, chaque échantillon sera, en définitive, de la forme suivante :

$$IX(I) = 200 \ \text{Sin}\ (x)$$
$$IY(I) = 200 \ \cos\ (x)$$

Si on considère 128 échantillons de cette forme, la fonction de corrélation sera donnée par l'algorithme suivant :

```
DO   FOR   I = 1, 128
     DO  FOR IND = 1,129-I
         J = IND+I-1
         IR(IND)=IR(IND)+IX(I)*IX(J)+IY(I)*IY(J)
         IM(IND)=IM(IND)+IY(I)*IX(J)-IX(I)*IY(J)
     ENDDO
ENDDO
```

où IR et IM sont les parties réelles et imaginaires de la fonction de corrélation.

Ce programme donne les mêmes résultats numériques que le corrélateur traitant 256 points additionnés deux à deux (résultats numériques de la simulation en annexe).

2-3-2-2 Addition de trois échantillons

Le premier programme de test étant un peu figé, on en a réecrit un second dans lequel :

- le nombre d'émission
- le nombre de points à redoubler
- le nombre de points pris pour la génération du sinus

sont des paramètres du programme de test (programme donné en annexe). Ce nouveau programme nous a permis de vérifier que les résultats du simulateur avec les paramètres suivants :

-96 émissions
-addition des points 3 à 3
-sinusoïde gérée sur 576 points redoublés par 18

étaient identiques à ceux calculés directement avec 32 émissions avec des échantillons de la forme :

-IX(I) = 300 sin (x)
-IY(I) = 300 cos (x)

-CONCLUSION-

-:-

Les tests graphiques et numériques ayant donné satisfaction sur le simulateur, en janvier 1982, l'application complète a été testée sur le site de Tromsö avec le corrélateur EISCAT. Le programme s'est comporté comme prévu et sera utilisé par l'ensemble de la communauté EISCAT.

Cette méthode d'addition cohérente de signaux permet une amélioration du rapport signal sur bruit. Cependant, elle ne s'applique qu'à des signaux venant de la basse athmosphère dont la variation dans le temps est lente et dont la corrélation des échantillons successifs est grande.

Enfin, cette application développée pour EISCAT n'est cependant pas isolée, puisque la méthode d'addition cohérente est actuellement utilisée au Pérou au cente de Jicamarca.

- A N N E X E S -

STRUCTURE DU CODE INTERMEDIAIRE

Ce code intermédiaire généré par CORRTST et l'assembleur a pour caractéristique principale d'être relisible par un éditeur de texte. Ce code est une succession de triplets terminé par un code de fin : (0,0,0). Toutes les mémoires du corélateur qui sont accessibles par programme sont représentables dans ce code. Pour ce faire, tous les champs du corrélateur sont numérotés. Les principaux champs sont tout naturellement

-l'APB-STACK ; l'APM-STACK ; la mémoire programme ...
Le champ APB porte le n°16, le champ APM le n°17. La mémoire programme de 64 mots de 128 bits est divisée en 64 fois 8 mémoires de 16 bits. Chacun de ces 8 groupes est numéroté (de 8 à 15).

Ce numéro, appelé adresse est évidemment insuffisant. Pour repérer un élément, on a recours à une sous-adresse qui est son rang dans le champ considéré.

L'APB et l'APM comportent ainsi 16 sous-adresses, chaque groupe de la mémoire programme en comporte 64.

A chaque couple adresse/sous-adresse est associé sa valeur.

La structure d'un triplet est alors la suivante :

(adresse, sous-adresse, valeur)

Par exemple : 16,1,10 indique que le mot 1 de l'APB-STACK vaut 10.

La micro-instruction implantée à l'adresse i dans la mémoire programme sera représentée par :

8,i,V1
.....
8,i,V8

où V1, ..., V8 représentant les 128 bits de l'instruction.

Certains registres simples comme le DATA-I-Register, le STAR-ADRESS-REGISTER, n'ont évidemment pas besoin de sous-adresse. Dans ce cas, pour respecter la structure de triplet, la sous-adresse est mise à 0. Par exemple :

6,0,128 indique que le DATA-I-Register est initialisé avec la valeur 128. Un exemple de ce code intermédiaire est donné à la fin de ce mémoire.

Exemple de code intermédiaire

```
SINGLE PULSE MODIFIE 19 MARS 1981 ...
 1   0-18432
 4   0      1
 6   0    136
 8   0    111
 8   1   7748
 8   2    111
 8   3   7236
 8   4     47
 8   5   5700
 8   6   6664
 8   7   6656
 8   8     66
 8   9  30912
 8  10  30913
 8  11   6144
 8  12     66
 8  13    111
 9   0  28680
 9   1  28680
 9   2  28680
 9   3  28680
 9   4  28692
 9   5  28684
 9   6  28680
 9   7  28680
 9   8  29384
 9   9  29385
 9  10  29325
 9  11  28680
 9  12  29324
 9  13  29740
10   0      0
10   1      0
10   2      0
10   3      0
10   4   6144
10   5      0
10   6    104
10   7      0
10   8  13336
10   9  11804
10  10  10268
10  11      0
10  12   9240
10  13   7200
11   0      0
11   1  16421
11   2  16384
11   3  16423
11   4  16384
11   5  16384
11   6  16384
11   7  30720
11   8  28672
11   9  28672
11  10  28672
11  11  29696
11  12  28672
```

12	0	0
12	1	0
12	2	0
12	3	0
12	4	0
12	5	0
12	6	0
12	7	1153
12	8	1029
12	9	1029
12	10	1029
12	11	1153
12	12	1029
12	13	0
13	0	0
13	1	0
13	2	0
13	3	0
13	4	0
13	5	0
13	6	0
13	7	8447
13	8	8447
13	9	8362
13	10	8362
13	11	8447
13	12	8447
13	13	0
14	0	0
14	1	0
14	2	0
14	3	0
14	4	0
14	5	26
14	6	30
14	7	30
14	8	28
14	9	444
14	10	444
14	11	28
14	12	28
14	13	0
15	0	28671
15	1	28671
15	2	28671
15	3	28671
15	4	12287
15	5	2303
15	6	2303
15	7	245
15	8	159
15	9	-16234
15	10	-15978
15	11	2293
15	12	159
15	13	28671
16	0	20
16	1	18
16	2	8
16	3	6
16	4	0
16	8	22
16	9	220
16	10	2
16	11	2

16	13	1
16	14	1
16	15	10
17	0	1
17	13	2
17	14	1
17	15	11
0	0	0

Algorithme et code du programme

	PRϕ - INSTRUCTIONS	ARB. INSTRUCTIONS	ALM - INSTRUCTIONS	ARI - INSTRUCTIONS	ACC - INSTRUCTIONS	
1	DEB Reload LCR2	ϕUT = RS(14) [Nb cmds / Nb pas − 1]				1
2	NϕϕR					2
3	LC1 = LCR1 ; LC2 = LCR2	TS1 = Q		Charge multiplieur A	Clear FF1	3
4	NϕϕR	TS2 = TS1				4
5	LCR1A = LC1 ; LC1 = LC1 − 1 ; LC2 = LC2 − 1	Dcommande = TS1	Dcommande = Q	Charge multiplieur B ; calcul	Read = YES ; WRITE = YES	5
6	N ϕ LC2 = LC2 − 1 ; LC1 = 0 ? YES	Dcommande = Dcommande + INC2	Dcommande = Dcommande + increment	Charge multiplieur B ; calcul	Read = YES ; Write = YES	6
7	E1 LC2 = LCR2	TS2 = TS2 + Sample increment			Set FF1	7
8	LC2 = 0 ? YES ; N ϕ LC2 = LC2 − 1	Dcommande = TS2	Dcommande = Q	Charge multiplieur B ; calcul	Read = YES ; Write = YES	8
9	N ϕ LC2 = LC2 − 1 ; LC2 = 0 ? YES ; LC2 = LCR2	Dcommande = Dcommande + INC2	Dcommande = Dcommande + increment	Charge multiplieur B ; calcul	Read = YES ; Write = YES	9
10	N ϕ LC1 = LC1 − 1 ; LC1 = 0 ? YES					10
11	LC1 = LCR1A	Q = Q + Sample increment		Charge multiplieur A		11
12	LC1 = 0 ? YES ; N ϕ LC1 = LC1 − 1	TS2 = TS1				12
13	LCR1A = LC1 ; LC1 = LCR1	TS2 = TS2 − Sample increment				13
14	LC1 = LCR1 ; LC2 = LCR2	COMPTEUR = COMPTEUR − 1				14
15	LC2 = 0 ? YES ; N ϕ					15
16	Reload LCR2	ϕUT = RS(10) [COMPTEUR]				16
17	NϕϕR					17
18	LCR1A = LC1 ; LC2 = LCR2	TS1 = Q				18
19	GϕTϕ E1	TS2 = TS2 − Sample increment				19
20	LC3 = 0 ? YES RETURN ; N ϕ LC3 = LC3 − 1	COMPTEUR = RS(14)	Q = Q + Sample gate increment			20
21	GϕTϕ DEB	Q = Q − RS(11) [Recula v]				21

PROGRAM-INSTRUCTIONS DEFINED

MEM-LOC.	COND.CODE	CODE-B	CODE-A	ADDR.	LC1	LCR1A	LC2	LC3	RELOAD	R-ADDR.
0	32	6	6	0	0	0	0	0	0	0
1	32	6	4	0	0	0	0	0	1	19
2	32	6	4	0	0	0	0	0	0	0
3	32	6	4	0	2	0	3	0	0	0
4	32	6	4	0	0	0	0	0	0	0
5	32	6	4	0	1	1	1	0	0	0
6	58	4	6	6	0	0	1	0	0	0
7	32	6	4	0	0	0	3	0	0	0
8	58	6	4	10	0	0	1	0	0	0
9	58	4	6	9	0	0	1	0	0	0
10	57	4	6	7	1	0	3	0	0	0
11	32	6	4	0	3	0	0	0	0	0
12	57	6	4	14	1	0	0	0	0	0
13	32	6	6	7	2	1	0	0	0	0
14	32	6	4	0	2	0	3	0	0	0
15	58	6	4	20	0	0	0	0	0	0
16	32	6	4	0	0	0	0	0	1	19
17	32	6	4	0	0	0	0	0	0	0
18	32	6	4	0	0	1	3	0	0	0
19	32	6	6	7	0	0	0	0	0	0
20	60	1	4	0	0	0	0	1	0	0
21	32	6	6	1	0	0	0	0	0	0

APR-INSTRUCTIONS DEFINED

MEM-LOC.	ALU-SOURCE	ALU-FUNCTION	ALU-DESTIN.	A-ADDR.	B-ADDR.	SELECT
0	2	4	1	0	0	0
1	4	0	1	14	0	0
2	2	4	1	0	0	0
3	2	0	3	0	3	0
4	4	0	3	3	1	0
5	4	0	3	3	2	0
6	1	0	3	9	2	0
7	1	0	3	12	1	0
8	4	0	3	1	2	0
9	1	0	3	9	2	0
10	2	4	1	0	0	0
11	0	0	0	12	0	0
12	4	0	3	3	1	0
13	1	1	3	12	1	0
14	1	1	3	8	10	0
15	2	4	1	0	0	0
16	4	0	1	10	0	0
17	2	4	1	0	0	0
18	2	0	3	0	3	0
19	0	1	3	12	1	0
20	4	0	3	14	10	0
21	0	1	0	11	0	0

APM-INSTRUCTIONS DEFINED

MEM-LOC.	ALU-SOURCE	ALU-FUNCTION	ALU-DESTIN.	A-ADDR.	B-ADDR.
0	2	4	1	0	0
1	2	4	1	0	0
2	2	4	1	0	0
3	2	4	1	0	0
4	2	4	1	0	0
5	2	0	3	0	13
6	1	0	3	14	13
7	2	4	1	0	0
8	2	0	3	0	13
9	1	0	3	14	13
10	2	4	1	0	0
11	2	4	1	0	0
12	2	4	1	0	0
13	2	4	1	0	0
14	2	4	1	0	0
15	2	4	1	0	0
16	2	4	1	0	0
17	2	4	1	0	0
18	2	4	1	0	0
19	2	4	1	0	0
20	0	0	0	15	0
21	2	4	1	0	0

ARITHMETICAL INSTRUCTIONS DEFINED

MEM-LOC.	M1A	M1B	M2A	M2B	M3A	M3B	M4A	M4B	SM1	SM2	SM3	SM4	M12	M34
0	4	3	4	3	4	3	4	3	0	0	0	0	12	12
1	4	3	4	3	4	3	4	3	0	0	0	0	12	12
2	4	3	4	3	4	3	4	3	0	0	0	0	12	12
3	0	0	1	1	1	0	0	1	1	1	1	1	12	12
4	4	3	4	3	4	3	4	3	0	0	0	0	12	12
5	0	0	1	1	1	0	0	1	2	2	2	2	9	6
6	0	0	1	1	1	0	0	1	2	2	2	2	9	6
7	4	3	4	3	4	3	4	3	0	0	0	0	12	12
8	0	0	1	1	1	0	0	1	2	2	2	2	9	6
9	0	0	1	1	1	0	0	1	2	2	2	2	9	6
10	4	3	4	3	4	3	4	3	0	0	0	0	12	12
11	0	0	1	1	1	0	0	1	1	1	1	1	12	12
12	4	3	4	3	4	3	4	3	0	0	0	0	12	12
13	4	3	4	3	4	3	4	3	0	0	0	0	12	12
14	4	3	4	3	4	3	4	3	0	0	0	0	12	12
15	4	3	4	3	4	3	4	3	0	0	0	0	12	12
16	4	3	4	3	4	3	4	3	0	0	0	0	12	12
17	4	3	4	3	4	3	4	3	0	0	0	0	12	12
18	4	3	4	3	4	3	4	3	0	0	0	0	12	12
19	4	3	4	3	4	3	4	3	0	0	0	0	12	12
20	4	3	4	3	4	3	4	3	0	0	0	0	12	12
21	4	3	4	3	4	3	4	3	0	0	0	0	12	12

ACCUMULATOR INSTRUCTIONS DEFINED

MEM-LOC.	STROBE	I/O WRITE	READ	CLEAR1	SET1	CLEAR2	SET2
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0
4	0	0	0	0	0	0	0
5	1	1	1	0	0	0	0
6	1	1	1	0	0	0	0
7	0	0	0	0	1	0	0
8	1	1	1	0	0	0	0
9	1	1	1	0	0	0	0
10	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0

Code du programme complet

PROGRAM-INSTRUCTIONS DEFINED											OUTPUT-TRANSFER INSTRUCTIONS DEFINED						
MEM-LOC.	COND.CODE	CODE-B	CODE-A	ADDR.	LC1	LCNIA	LC2	LC3	RELOAD	B-ADDR.	MEM-LOC.	TRANSFER	INHIBIT CLOCK	DATA-READY	TRANSFER-CODE	SOURCE	
0	56	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	32	6	4	0	0	0	0	0	0	0	1	0	0	0	0	0	
2	32	6	4	0	0	0	0	0	0	0	2	0	0	0	0	0	
3	32	6	4	0	0	0	0	0	1	18	3	0	0	0	0	0	PROGRAMME
4	32	6	4	0	0	0	0	0	0	0	4	0	0	0	0	0	
5	32	6	4	0	2	0	0	0	0	0	5	0	0	0	0	0	PRINCIPAL
6	57	6	4	53	0	0	0	0	0	0	6	0	0	0	0	0	
7	32	6	4	0	0	0	0	0	1	20	7	0	0	0	0	0	
8	32	6	4	0	0	0	0	0	0	0	8	0	0	0	0	0	
9	32	6	4	0	0	0	0	0	1	18	9	0	0	0	0	0	
10	32	6	6	45	0	0	0	0	0	0	10	0	0	0	0	0	
11	32	6	4	0	0	0	0	0	1	19	11	0	0	0	0	0	
12	32	6	4	0	0	0	0	0	0	0	12	0	0	0	0	0	
13	32	6	4	0	2	0	3	0	0	0	13	0	0	0	0	0	
14	32	6	4	0	0	0	0	0	0	0	14	0	0	0	0	0	Subroutine
15	32	6	4	0	1	1	1	0	0	0	15	0	0	0	0	0	
16	58	4	6	16	0	0	1	0	0	0	16	0	0	0	0	0	
17	32	6	4	0	0	0	3	0	0	0	17	0	0	0	0	0	
18	58	6	4	20	0	0	1	0	0	0	18	0	0	0	0	0	ADDITION
19	58	4	6	19	0	0	1	0	0	0	19	0	0	0	0	0	
20	57	4	6	17	1	0	3	0	0	0	20	0	0	0	0	0	
21	32	6	4	0	3	0	0	0	0	0	21	0	0	0	0	0	
22	57	6	4	24	1	0	0	0	0	0	22	0	0	0	0	0	
23	32	6	6	17	2	1	0	0	0	0	23	0	0	0	0	0	
24	32	6	4	0	2	0	3	0	0	0	24	0	0	0	0	0	
25	58	6	4	30	0	0	0	0	0	0	25	0	0	0	0	0	
26	32	6	4	0	0	0	0	0	1	19	26	0	0	0	0	0	
27	32	6	4	0	0	0	0	0	0	0	27	0	0	0	0	0	
28	32	6	4	0	0	1	3	0	0	0	28	0	0	0	0	0	
29	32	6	6	17	0	0	0	0	0	0	29	0	0	0	0	0	
30	60	1	4	0	0	0	0	1	0	0	30	0	0	0	0	0	
31	32	6	6	11	0	0	0	0	0	0	31	0	0	0	0	0	
32	56	0	4	0	0	0	0	0	1	20	32	0	0	0	0	0	
33	56	0	4	0	0	0	0	0	0	0	33	0	0	0	0	0	
34	56	0	4	0	0	0	0	0	0	0	34	1	0	0	0	0	Subroutine
35	56	0	4	0	0	0	0	0	0	0	35	1	1	1	0	0	
36	56	0	8	0	0	0	0	2	0	0	36	1	1	1	1	0	
37	56	0	4	0	0	0	0	1	0	0	37	1	1	1	3	0	DMA
38	56	0	4	0	0	0	0	0	0	0	38	1	1	1	2	0	
39	56	0	4	0	0	0	0	0	0	0	39	1	1	1	5	0	
40	60	0	5	0	0	0	0	0	0	0	40	1	1	1	4	0	TRANSFERT
41	56	0	4	0	0	0	0	0	0	0	41	1	0	0	0	0	
42	56	0	4	0	0	0	0	0	0	0	42	1	0	0	0	0	
43	56	0	4	0	0	0	0	0	0	0	43	1	0	0	0	0	
44	56	0	6	0	0	0	0	0	0	0	44	1	0	0	0	0	
45	32	6	10	11	0	0	0	2	0	0	45	0	0	0	0	0	
46	32	6	4	0	0	0	0	0	1	19	46	0	0	0	0	0	
47	32	6	4	0	0	0	0	0	0	0	47	0	0	0	0	0	
48	32	6	4	0	0	0	0	0	1	18	48	0	0	0	0	0	PROGRAMME
49	32	6	4	0	0	0	0	0	0	0	49	0	0	0	0	0	
50	32	0	10	55	0	0	0	0	0	0	50	0	0	0	0	0	PRINCIPAL
51	32	0	10	55	0	0	0	0	0	0	51	0	0	0	0	0	
52	32	6	6	3	0	0	0	0	0	0	52	0	0	0	0	0	
53	56	0	4	0	0	0	0	0	0	0	53	0	0	0	0	0	
54	32	6	6	0	0	0	0	0	0	0	54	0	0	0	0	0	
55	32	6	4	0	0	0	0	0	0	0	55	0	0	0	0	0	
56	56	0	4	0	2	0	3	0	0	0	56	0	0	0	0	0	
57	57	6	4	61	1	0	0	0	0	0	57	0	0	0	0	0	
58	57	6	4	60	1	0	0	0	0	0	58	0	0	0	0	0	Subroutine
59	57	4	6	59	1	0	0	0	0	0	59	0	0	0	0	0	
60	56	0	4	0	0	0	0	0	0	0	60	0	0	0	0	0	CALCUL
61	58	4	6	57	2	0	1	0	0	0	61	0	0	0	0	0	
62	56	0	4	0	0	0	0	0	0	0	62	0	0	0	0	0	PUISSANCE
63	56	0	1	0	0	0	0	0	0	0	63	0	0	0	0	0	

	APB-INSTRUCTIONS DEFINED						APM-INSTRUCTIONS DEFINED						
MEM-LOC.	ALU-SOURCE	ALU-FUNCTION	ALU-DESTIN.	A-ADDR.	B-ADDR.	SELECT	MEM-LOC.	ALU-SOURCE	ALU-FUNCTION	ALU-DESTIN.	A-ADDR.	B-ADDR.	
0	7	5	1	0	0	0	0	7	5	1	0	0	
1	4	0	3	5	4	0	1	2	4	1	0	0	
2	2	4	1	0	0	0	2	2	4	1	0	0	
3	4	0	1	4	0	0	3	2	4	1	0	0	PROGRAMME
4	2	4	1	0	0	0	4	2	4	1	0	0	
5	2	4	1	0	0	0	5	2	4	1	0	0	
6	1	1	3	8	4	0	6	2	4	1	0	0	PRINCIPAL
7	4	0	1	13	0	0	7	2	4	1	0	0	
8	2	4	1	0	0	0	8	2	4	1	0	0	
9	4	0	3	15	0	0	9	2	4	1	0	0	
10	2	4	1	0	0	0	10	2	4	1	0	0	
11	4	0	1	14	0	0	11	2	4	1	0	0	
12	2	4	1	0	0	0	12	2	4	1	0	0	
13	2	0	3	0	3	0	13	2	4	1	0	0	
14	4	0	3	3	1	0	14	2	4	1	0	0	
15	4	0	3	3	2	0	15	2	0	3	0	13	
16	1	0	3	9	2	0	16	1	0	3	14	13	
17	1	0	3	12	1	0	17	2	4	1	0	0	Subroutine
18	4	0	3	1	2	0	18	2	0	3	0	13	
19	1	0	3	9	2	0	19	1	0	3	14	13	
20	2	4	1	0	0	0	20	2	4	1	0	0	
21	0	0	0	12	0	0	21	2	4	1	0	0	ADDITION
22	4	0	3	3	1	0	22	2	4	1	0	0	
23	1	1	3	12	1	0	23	2	4	1	0	0	
24	1	1	3	8	10	0	24	2	4	1	0	0	
25	2	4	1	0	0	0	25	2	4	1	0	0	
26	4	0	1	10	0	0	26	2	4	1	0	0	
27	2	4	1	0	0	0	27	2	4	1	0	0	
28	2	0	3	0	3	0	28	2	4	1	0	0	
29	0	1	3	12	1	0	29	2	4	1	0	0	
30	4	0	3	14	10	0	30	0	0	0	15	0	
31	0	1	0	11	0	0	31	2	4	1	0	0	
32	7	0	1	0	0	0	32	7	5	1	0	0	
33	7	5	1	0	0	0	33	7	5	1	0	0	
34	7	5	1	0	0	0	34	7	5	1	0	0	
35	7	5	1	0	0	0	35	7	5	0	0	0	Subroutine
36	7	5	1	0	0	0	36	0	1	0	0	0	
37	7	5	1	0	0	0	37	0	0	0	0	0	DMA
38	7	5	1	0	0	0	38	2	0	1	0	0	
39	7	5	1	0	0	0	39	2	0	1	0	0	
40	7	5	1	0	0	0	40	2	0	1	0	0	
41	7	5	1	0	0	0	41	7	5	1	0	0	TRANSFERT
42	7	5	1	0	0	0	42	7	5	1	0	0	
43	7	5	1	0	0	0	43	7	5	1	0	0	
44	7	5	1	0	0	0	44	7	5	1	0	0	
45	2	4	1	0	0	0	45	2	4	1	0	0	
46	4	0	1	7	0	0	46	2	4	1	0	0	
47	2	4	1	0	0	0	47	2	4	1	0	0	PROGRAMME
48	4	0	1	6	0	0	48	2	4	1	0	0	
49	2	4	1	0	0	0	49	2	4	1	0	0	
50	0	1	0	11	0	0	50	2	4	1	0	0	
51	7	5	1	0	0	0	51	0	0	0	14	0	PRINCIPAL
52	1	0	0	8	3	0	52	0	0	0	14	0	
53	7	5	0	0	0	0	53	2	0	1	0	0	
54	7	5	1	0	0	0	54	7	5	0	0	0	
55	1	1	3	12	11	0	55	7	5	1	0	0	
56	0	0	0	11	0	0	56	7	5	1	0	0	Subroutine
57	0	1	0	11	0	0	57	2	0	1	0	0	
58	2	0	1	0	0	0	58	7	5	1	0	0	
59	0	0	0	12	0	0	59	7	5	1	0	0	CALCUL
60	0	0	0	12	0	0	60	2	0	1	0	0	
61	2	0	3	0	3	0	61	7	5	1	0	0	PUISSANCE
62	0	1	0	11	0	0	62	2	0	1	0	0	
63	1	0	3	12	11	0	63	7	5	1	0	0	

ARITHMETICAL INSTRUCTIONS DEFINED

MEM-LOC.	M1A	M1B	M2A	M2B	M3A	M3B	M4A	M4B	SM1	SM2	SM3	SM4	M12	M34
0	0	0	0	0	0	0	0	0	0	0	0	0	15	15
1	4	3	4	3	4	3	4	3	0	0	0	0	12	12
2	4	3	4	3	4	3	4	3	0	0	0	0	12	12
3	4	3	4	3	4	3	4	3	0	0	0	0	12	12
4	4	3	4	3	4	3	4	3	0	0	0	0	12	12
5	4	3	4	3	4	3	4	3	0	0	0	0	12	12
6	4	3	4	3	4	3	4	3	0	0	0	0	12	12
7	4	3	4	3	4	3	4	3	0	0	0	0	12	12
8	4	3	4	3	4	3	4	3	0	0	0	0	12	12
9	4	3	4	3	4	3	4	3	0	0	0	0	12	12
10	4	3	4	3	4	3	4	3	0	0	0	0	12	12
11	4	3	4	3	4	3	4	3	0	0	0	0	12	12
12	4	3	4	3	4	3	4	3	0	0	0	0	12	12
13	0	0	1	1	1	0	0	1	1	1	1	1	12	12
14	4	3	4	3	4	3	4	3	0	0	0	0	12	12
15	0	0	1	1	1	0	0	1	2	2	2	2	9	6
16	0	0	1	1	1	0	0	1	2	2	2	2	9	6
17	4	3	4	3	4	3	4	3	0	0	0	0	12	12
18	0	0	1	1	1	0	0	1	2	2	2	2	9	6
19	0	0	1	1	1	0	0	1	2	2	2	2	9	6
20	4	3	4	3	4	3	4	3	0	0	0	0	12	12
21	0	0	1	1	1	0	0	1	1	1	1	1	12	12
22	4	3	4	3	4	3	4	3	0	0	0	0	12	12
23	4	3	4	3	4	3	4	3	0	0	0	0	12	12
24	4	3	4	3	4	3	4	3	0	0	0	0	12	12
25	4	3	4	3	4	3	4	3	0	0	0	0	12	12
26	4	3	4	3	4	3	4	3	0	0	0	0	12	12
27	4	3	4	3	4	3	4	3	0	0	0	0	12	12
28	4	3	4	3	4	3	4	3	0	0	0	0	12	12
29	4	3	4	3	4	3	4	3	0	0	0	0	12	12
30	4	3	4	3	4	3	4	3	0	0	0	0	12	12
31	4	3	4	3	4	3	4	3	0	0	0	0	12	12
32	0	0	0	0	0	0	0	0	0	0	0	0	15	15
33	0	0	0	0	0	0	0	0	0	0	0	0	15	15
34	0	0	0	0	0	0	0	0	0	0	0	0	15	15
35	0	0	0	0	0	0	0	0	0	0	0	0	15	15
36	0	0	0	0	0	0	0	0	0	0	0	0	15	15
37	0	0	0	0	0	0	0	0	0	0	0	0	15	15
38	0	0	0	0	0	0	0	0	0	0	0	0	15	15
39	0	0	0	0	0	0	0	0	0	0	0	0	15	15
40	0	0	0	0	0	0	0	0	0	0	0	0	15	15
41	0	0	0	0	0	0	0	0	0	0	0	0	15	15
42	0	0	0	0	0	0	0	0	0	0	0	0	15	15
43	0	0	0	0	0	0	0	0	0	0	0	0	15	15
44	0	0	0	0	0	0	0	0	0	0	0	0	15	15
45	4	3	4	3	4	3	4	3	0	0	0	0	12	12
46	4	3	4	3	4	3	4	3	0	0	0	0	12	12
47	4	3	4	3	4	3	4	3	0	0	0	0	12	12
48	4	3	4	3	4	3	4	3	0	0	0	0	12	12
49	4	3	4	3	4	3	4	3	0	0	0	0	12	12
50	4	3	4	3	4	3	4	3	0	0	0	0	12	12
51	4	3	4	3	4	3	4	3	0	0	0	0	12	12
52	4	3	4	3	4	3	4	3	0	0	0	0	12	12
53	0	0	0	0	0	0	0	0	0	0	0	0	12	12
54	0	0	0	0	0	0	0	0	0	0	0	0	15	15
55	0	0	0	0	0	0	0	0	0	0	0	0	15	15
56	0	0	0	0	0	0	0	0	0	0	0	0	15	15
57	0	0	1	1	4	0	4	1	3	3	3	3	9	9
58	0	0	0	0	0	0	0	0	3	3	3	3	6	6
59	0	0	1	1	4	0	4	1	3	3	3	3	9	9
60	0	0	1	1	4	0	4	1	3	3	3	3	9	9
61	0	0	0	0	0	0	0	0	3	3	3	3	6	6
62	0	0	0	0	0	0	0	0	3	3	3	3	6	6
63	0	0	0	0	0	0	0	0	0	0	0	0	15	15

ACCUMULATOR INSTRUCTIONS DEFINED

MEM-LOC.	STROBE	I/O WRITE	READ	CLEAR1	SET1	CLEAR2	SET2
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0
13	0	0	0	1	0	0	0
14	0	0	0	0	0	0	0
15	1	1	1	0	0	0	0
16	1	1	1	0	0	0	0
17	0	0	0	0	1	0	0
18	1	1	1	0	0	0	0
19	1	1	1	0	0	0	0
20	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0
37	0	0	0	0	0	0	0
38	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0
41	0	0	0	0	0	0	0
42	0	0	0	0	0	0	0
43	0	0	0	0	0	0	0
44	0	0	0	0	0	1	0
45	0	0	0	0	0	0	0
46	0	0	0	0	0	0	0
47	0	0	0	0	0	0	0
48	0	0	0	0	0	0	0
49	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0
51	0	0	0	0	0	0	0
52	0	0	0	0	0	0	0
53	1	1	1	0	0	0	0
54	0	0	0	0	0	0	1
55	0	0	0	0	0	0	0
56	1	0	0	1	0	0	0
57	1	0	1	0	0	0	0
58	1	0	0	0	0	0	0
59	1	0	0	0	0	0	0
60	1	1	0	0	1	0	0
61	1	0	0	0	0	0	0
62	1	1	0	0	0	0	0
63	0	0	0	0	0	0	0

PROGRAMME PRINCIPAL

Subroutine ADDITION

Subroutine DMA TRANSFERT

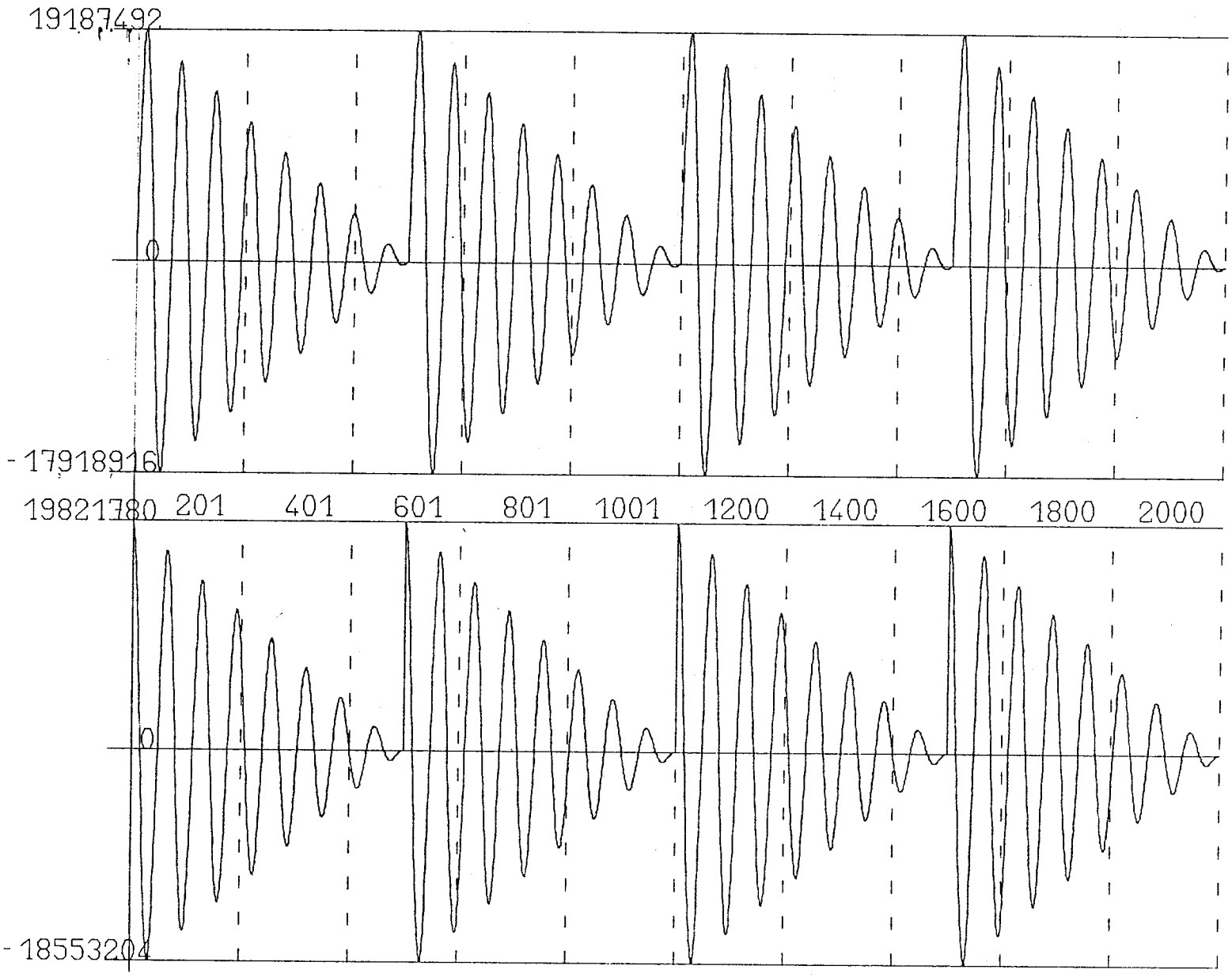
PROGRAMME PRINCIPAL

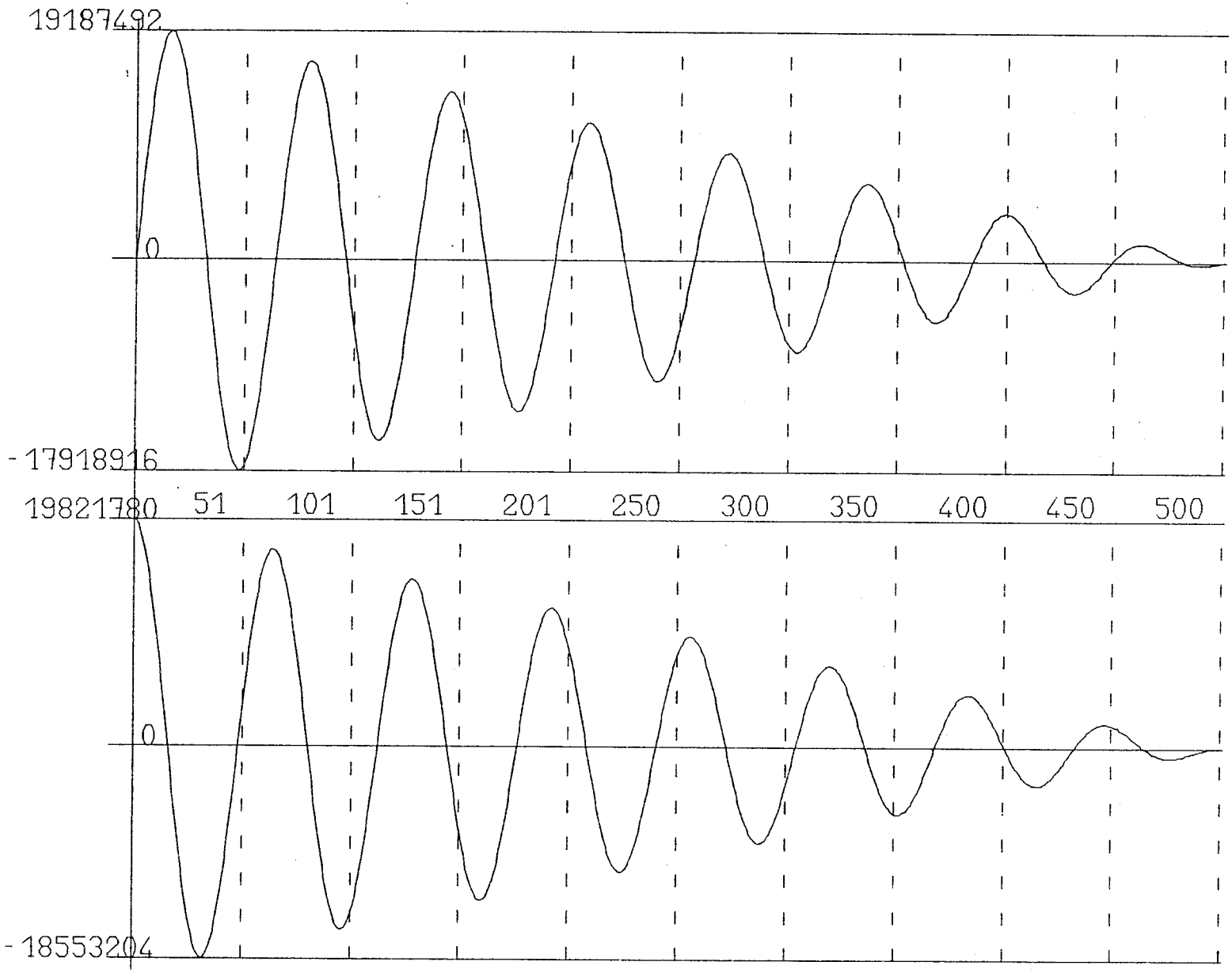
Subroutine CALCUL PUISSANCE

Résultats graphiques

Simulation d'une expérience comportant :

- 1000 émissions
- 4 altitudes
- 0 bruit et 0 calibration
- addition des échantillons 2 à 2


19187492
-17918916
0
19821780
201
401
601
801
1001
1200
1400
1600
1800
2000
0
-18553204


19187492
0
-17918916
19821780
51
101
151
201
250
300
350
400
450
500
0
-18553204

Programme de calculs numériques et résultats

Resultats numériques de la simulation sur 256 points addt. avec 2 à 2 (cf p92)

PARTIE REELLE

5079936	4943072	4619412	4124192	3479144
2711536	1852536	936640	0	-921440
-1792344	-2579984	-3255736	-3795328	-4180308
-4398688	-4444944	-4320288	-4032764	-3596224
-3030152	-2358768	-1609544	-812768	0
797568	1549352	2227216	2806744	3267360
3593660	3775904	3809952	3697504	3446116
3068256	2581160	2006000	1366552	688896
0	-673696	-1306360	-1874448	-2357752
-2739392	-3007012	-3153120	-3174960	-3074720
-2859468	-2540288	-2132168	-1653232	-1123560
-565024	0	549824	1063368	1521680
1908760	2211424	2420364	2530336	2539968
2451936	2272820	2012320	1683176	1300464
880568	441152	0	-425952	-820376
-1168912	-1459768	-1683456	-1833716	-1907552
-1904976	-1829152	-1686172	-1484352	-1234184
-947696	-637576	-317280	0	302080
577384	816144	1010776	1155488	1247068
1284768	1269984	1206368	1099524	956384
785192	594928	394584	193408	0
-178208	-334392	-463376	-561784	-627520
-660420	-661984	-634992	-583584	-512876
-428416	-336200	-242160	-151592	-69536
0	54336	91400	110608	112792
99552	73772	39200		

PARTIE IMAGINAIRE

0	983376	1913840	2756368	3480232
4059312	4473632	4710080	4762440	4631680
4326088	3860208	3254648	2535152	1731040
874704	0	-859504	-1670848	-2403600
-3031240	-3531344	-3886984	-4087296	-4127448
-4008896	-3739440	-3332240	-2805656	-2182384
-1488048	-750832	0	735632	1427856
2050832	2582248	3003376	3300336	3464512
3492456	3386112	3152792	2804272	2356664
1829616	1245056	626960	0	-611760
-1184864	-1698064	-2133256	-2475408	-2713688
-2841728	-2857464	-2763328	-2566144	-2276304
-1907672	-1476848	-1002064	-503088	0
487888	941872	1345296	1684264	1947440
2127040	2218944	2222472	2140544	1979496
1748336	1458680	1124080	759072	379216
0	-364016	-698880	-992528	-1235272
-1419472	-1540392	-1596160	-1587480	-1517760
-1392848	-1220368	-1009688	-771312	-516080
-255344	0	240144	455888	639760
786280	891504	953744	973376	952488
894976	806200	692400	560696	418544
273088	131472	0	-116272	-212896
-286992	-337288	-363536	-367096	-350592
-317496	-272192	-219552	-164432	-111704
-65776	-30096	-7600		

```
 NORD 10/100 FORTRAN COMPILER FTN-2090G      TIME 10.42     DATE 21.01.19

  1*          INTEGER FIXE,MOBILE,ADR
  2*          DOUBLE INTEGER XF,XM,YF,YM
  3*          DOUBLE INTEGER X(550),Y(550)
  4*          DOUBLE INTEGER R(128),IM(128)
  5*          REAL R1(128),IM1(128)
  6*          PI=3.14158265
  7*          OUTPUT(1)'      NOMBRE D EMISSION ?'
  8*          INPUT(1) NEMISS
  9*          OUTPUT(1)'      NOMBRE DE POINTS A REDOUBLER ?'
 10*          INPUT(1)NADD
 11*          N=NEMISS/NADD
 12*          OUTPUT(1)'     NOMBRE DE POINTS PRIS POUR LA SINUSOIDE?'
 13*          INPUT(1) NPOINT
 14*          DO FOR I=1,NPOINT/NADD
 15*          RX=(2.0*PI*FLOAT(I-1))/(NPOINT/NADD)
 16*                DO FOR J=1,NADD
 17*                IND=(I-1)*NADD+J
 18*                X(IND)=100.*SIN(RX)
 19*                Y(IND)=100.*COS(RX)
 20*                ENDDO
 21*          ENDDO
 22*          OPEN(10,FILE='L-F',ACCESS='W')
 23*          DO FOR I=1,N
 24*                DO FOR J=1,NADD
 25*                FIXE=(I-1)*NADD+J
 26*                MOBILE=(I-1)*NADD+1
 27*                      ADR=0;IAD=0
 28*                      DO FOR L=MOBILE,NEMISS
 29*                      IF(IAD .EQ. (IAD/NADD)*NADD) ADR=ADR+1
 30*                      J1=FIXE-1;J2=L-1
 31*                      IAD1=J1-(J1/NPOINT)*NPOINT+1
 32*                      IAD2=J2-(J2/NPOINT)*NPOINT+1
 33*                      XF=X(IAD1);YF=Y(IAD1)
 34*                      XM=X(IAD2);YM=Y(IAD2)
 35*                      R(ADR)=R(ADR)+XF*XM+YF*YM
 36*                      IM(ADR)=IM(ADR)-XF*YM+YF*XM
 37*                      IAD=IAD+1
 38*                      ENDDO
 39*                ENDDO
 40*          ENDDO
 41*          WRITE(10,301)
 42*          WRITE(10,300)(R(I),I=1,N)
 43*          WRITE(10,302)
 44*          WRITE(10,300)(IM(I),I=1,N)
 45*  300     FORMAT(4X,5I10)
 46*  301     FORMAT(//,10X,'PARTIE REELLE ',/,10X,10(1H-))
 47*  302     FORMAT(//,10X,'PARTIE IMAGINAIRE',/,10X,10(1H-))
 48*          DO FOR I=1,NEMISS/NADD
 49*          R1(I)=R(I)
 50*          IM1(I)=IM(I)
 51*          ENDDO
 52*          CALL TRACE(R1,N)
 53*          CALL TRACE(IM1,N)
 54*          END
```

PARTIE REELLE

2857464	2714328	2473929	2151864	1766682
1338588	887814	435168	0	-400968
-752382	-1042596	-1264014	-1411920	-1485945
-1489464	-1428732	-1313064	-1153971	-963936
-756450	-544860	-341082	-156456	0
122256	205650	248868	253782	223992
165987	88200			

PARTIE IMAGINAIRE

0	540324	1025748	1439460	1769130
2005884	2145924	2190096	2143098	2013696
1813950	1557900	1261566	941724	614448
295812	0	-261612	-479016	-645732
-758898	-817956	-825966	-788832	-714366
-612432	-493992	-369972	-251334	-147996
-67716	-17100			

-BIBLIOGRAPHIE-

-Scientific programming of the EISCAT digital correlator
(Terrance HO et Hans-Jorgen ALKER) EISCAT 79/12

-Instruction manual for EISCAT digital correlator
(Hans-Jorgen ALKER) EISCAT 79/10

-A programmable correlator module for the EISCAT radar system
(Hans-Jorgen ALKER) EISCAT 79/11

-A description of the assembly language for the EISCAT digital correlator
(Bard Willy TORUSTAD) EISCAT 79/15

-Program CORRSIM : systeme for program development and software simulation of EISCAT digital correlator
(Hans Jorgen ALKER) EISCAT 79/9

-The EISCAT correlator
(Régis GRAS) EISCAT 82/34