Réalisation d'outils logiciels pour la mise en œuvre de microprocesseurs dans la conduite automatique de procédés complexes

Jean-Paul Eynard

▶ **To cite this version:**

Jean-Paul Eynard. Réalisation d'outils logiciels pour la mise en œuvre de microprocesseurs dans la conduite automatique de procédés complexes. Performance et fiabilité [cs.PF]. 1979. dumas-00334254

HAL Id: dumas-00334254

https://dumas.ccsd.cnrs.fr/dumas-00334254

Submitted on 24 Oct 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

— MEMOIRE D'INGENIEUR —

REALISATION D'OUTILS LOGICIELS POUR LA MISE EN ŒUVRE DE MICROPROCESSEURS DANS LA CONDUITE AUTOMATIQUE DE PROCEDES COMPLEXES

Jean-Paul EYNARD

Janvier 1979

ATELIER DE MICROINFORMATIQUE DE GRENOBLE

Je remercie Mr. le Professeur Louis BOLLIET qui a bien voulu m'accueillir au sein de l'Atelier de Microinformatique de Grenoble afin d'y préparer ce mémoire dans d'excellentes conditions.

Il y a un an les domaines des microprocesseurs et de l'automatique étaient nouveaux pour moi, aussi n'ai-je pu y progresser et y réaliser ce travail que grâce aux conseils et à l'aide de Raymond BOUTTAZ, responsable technique de l'Atelier de Microinformatique, de Zdenek BINDER et Daniel REY, chercheurs au Laboratoire d'Automatique de Grenoble.

Grenoble, janvier 1979

- TABLE DES MATIERES -

DEUXIEME PARTIE

Développement d'algorithmes de commande automatique multivariable structurables en temps réel.

II.1 Présentation générale des problèmes d'automatisme

II.1.1 Quelques définitions
II.1.2 Commande par poursuite d'objectifs selon un critère quadratique
II.1.3 Algorithmes de base

II.2 Le procédé pilote du L.A.G.
Le projet IMPACT

II.2.1 Structure modulaire de la commande
II.2.2 Un algorithme standard
II.2.3 Implantation matérielle

II.3 Développement du logiciel pour un centre de décision

II.3.1 Une organisation modulaire
II.3.2 Structuration de l'algorithme standard
II.3.3 Le moniteur de communication
II.3.4 Fonctionnement général d'un centre de décision

II.4 Tests et exploitation

II.4.1 Optique particulière de réalisation
II.4.2 Tests sur l'outil de développement
II.4.3 Les implantations futures

CONCLUSION

- AVANT-PROPOS -

L'ampleur des calculs nécessaires à la commande automatique de procédés complexes a freiné les tentatives de décentralisation de cette commande, décentralisation pourtant nécessaire à son évolution et à sa fiabilité. Pour un système multivariable les différents niveaux de la conduite utilisent le calcul matriciel, et les résolutions des différentes équations se situaient obligatoirement sur un minicalculateur suffisamment puissant, et unique dans la commande.

L'avènement des microprocesseurs, et leur faible coût, va permettre cette décentralisation en répartissant les fonctions de la commande sur un réseau de microcalculateurs. Toutefois on connait les contraintes imposées actuellement : peu de mémoire disponible, donc programmation en langage d'assemblage, obligation d'utiliser des dispositifs arithmétiques extérieurs au microprocesseur, en résumé absence des facilités offertes par l'emploi des langages de hauts niveaux implantés sur les minicalculateurs.

Or un des problèmes les plus élémentaires qu'un de ces microcalculateurs aura à résoudre au niveau d'une unité locale s'exprime par les équations suivantes :

- évolution d'un modèle local

$$x(i+1) = Ax(i) + Bu(i)$$
$$y(i) = Cx(i)$$

-calcul de la commande dans son expression la plus simple

$$u(i) = -Lx(i) + M1zi - M2zo$$

où A,B,C,L,M1,M2 sont des matrices et x,y,u,zi,zo sont des vecteurs.

Quand on a procédé à une approche des possibilités logicielles des microprocesseurs 8 bits actuellement sur le marché, en vue de l'écriture et de la mise au point de tels algorithmes en langage d'assemblage, une nécessité s'impose : celle de procéder au développement préalable d'un logiciel général de manipulation de matrices et d'utilisation de dispositifs arithmétiques, minimisant l'occupation mémoire mais présentant toutefois une bonne facilité d'emploi.

C'est à l'élaboration d'un tel logiciel pour le microprocesseur INTEL 8080 que nous nous sommes attachés dans un premier temps et dont la description constitue la première partie de ce mémoire.

Dans un deuxième temps l'utilisation de ce logiciel a trouvé sa place en permettant une mise au point aisée des programmes de commandes multivariables. Les algorithmes développés, structurables en temps réel, prendront place dans les divers centres de décision du système de commande décentralisée coordonnée mis au point par le Laboratoire d'Automatique de Grenoble dans le cadre du projet IMPACT.

D'une manière générale nous avons tenté de développer des logiciels standards pouvant être utilisés pour d'autres applications, essayant ainsi d'échapper à la spécialisation sur une application particulière et de permettre une évolution future d'une partie du matériel sans remise en cause totale du logiciel. Ces qualités sont obtenues principalement en constituant des blocs fonctionnels indépendants communiquant selon des protocoles élaborés précisément.

PREMIERE PARTIE

* * *

REALISATION D'OPERATEURS MATRICIELS POUR LES MICROPROCESSEURS INTEL-8080/8085

1.1 ASPECT LOGICIEL DES MICROPROCESSEURS

Notre propos n'est pas de faire une présentation détaillée mais plutôt de dégager les aspects intéressant ou limitatifs dans le cadre de l'étude présente : la manipulation de matrices. Le schéma-bloc et la liste des instructions qu'on trouvera en annexe A amènent les remarques suivantes.

Le jeu d'instructions :

Il comprend cinq types différents d'instructions

- Transfert de données
- Arithmétique
- Logique
- Branchements
- Contrôle de la pile, des entrées/sorties et des bits d'état

et permet trois types d'adressage :

- Immédiat (l'instruction contient la donnée)
- Direct (l'instruction contient l'adresse de la donnée ou spécifie un registre contenant cette donnée)
- Indirect (l'instruction spécifie la paire de registres qui contient l'adresse de la donnée, la plupart de ces instructions n'admettant que la paire HL)

L'accumulateur (A).

Il existe un seul accumulateur de 8 bits habilité à traiter les opérations arithmétiques et logiques, à véhiculer les informations dans les opérations d'entrée/sortie, et à bénéficier d'instructions spéciales comme le chargement ou le déchargement direct en mémoire à une adresse spécifiée dans l'instruction. Il n'existe pas de multiplication ni de division.

Les registres généraux (B,C,D,E,H,L)

Ils sont surtout utilisés par paire (BC,DE,HL) , possibilité extrèmement intéressante dans le cas présent puisqu'elle permet de stocker des adresses et donc d'utiliser les paires comme pointeurs dans les matrices (Cette possibilité n'existe pas par exemple avec le microprocesseur MOTOROLA 6800 qui ne possède qu'un registre index, mais qui avec ses deux accumulateurs offre par contre des possibilités arithmétiques et logiques plus grandes).
Les paires de registres peuvent etre affectées d'une valeur immédiate, incrémentées ou décrémentées de 1 mais n'ont accès à aucune autre opération arithmétique ou logique. Toutefois la paire HL joue un rôle privilégié très important. Outre la possibilité :

- d'ajouter à son contenu le contenu des autres paires
- de transférer son contenu dans le pointeur de pile ou le compteur ordinal
- de chargement ou de déchargement direct en mémoire

elle est la seule utilisable dans la plupart des instructions arithmétiques ou de transfert à adressage indirect. Elle joue donc parfois le rôle d'accumulateur double.

La pile

C'est une pile logicielle et elle doit être implantée en mémoire sous la responsabilité du programmeur par initialisation du pointeur de pile (SP).
On n'est donc pas limité par la taille de cette pile (si ce n'est par la mémoire disponible) , aussi on l'utilisera commodément pour le passage des paramètres et même pour définir des espaces de travail pour les programmes.

I.2 LES ARITHMETIQUES A POINT FLOTTANT

Il est évident que les nombres traités au cours de la commande de procédés complexes ne puissent se satisfaire d'une représentation entière sur 8 bits. Aussi faudra-t-il dans la majorité des cas utiliser une arithmétique à point flottant.

Nous avons pu disposer de trois arithmétiques :

- L'arithmétique programmée FPAL-INTEL
- L'arithmétique programmée FP.AIMI développée pour les tests à l'Atelier de Microinformatique
- Le circuit en un seul boitier AM9511 (Advanced Micro Devices).

I.2.1 FPAL-INTEL

La représentation d'un nombre se fait sur 32 bits qu'on numérote de droite à gauche :

- bit 31 : signe de la mantisse (1 négatif, 0 positif)
- bits 30 à 23 : exposant = exposant réel + 127
- bits 22 à 0 : mantisse normalisée, le bit à 1 étant implicite

La valeur du nombre est donnée par la formule :

(1. + .mantisse) * 2**(exposant - 127)

Ce qui permet une amplitude décimale de 8.43 10**(-37) à 3.37 10**38. Les opérations arithmétiques utilisent un espace de travail de 18 mots appelé FPR (Floating Point Record) composé principalement d'un accumulateur flottant où le nombre est stocké sous forme étendue (FACC) , d'une zone de calcul et d'une zone de traitement des erreurs ou anomalies rencontrées.

La séquence d'appel des opérations arithmétiques se décompose en trois étapes :

1- Chargement du 1er opérande dans l'accumulateur flottant
(fonction FLOAD)

2- Activation de l'opération désirée avec indication de l'adresse du 2ème opérande. Le résultat est placé dans FACC.

3- Déchargement de FACC dans le nombre résultat (fonction FSTOR).

Dans chacune de ces étapes la paire de registres BE doit contenir l'adresse du dernier octet du FPR, la

paire DE l'adresse du 4ème octet de l'opérande flottant concerné.

Une routine d'erreur permet de traiter les anomalies détectées suivant l'opération activée et à tout moment on peut obtenir un état des erreurs rencontrées au cours du traitement.

Les temps d'exécution indicatifs donnés sont de :

0.68 ms pour l'addition
1.50 ms pour la multiplication
3.65 ms pour la division.

FPAL est livrée sur 3 k-Octets de ROM à implanter à une adresse indiquée au moment de la commande.
Pour plus d'informations on se reportera d'une part en annexe B et d'autre part aux brochures décrivant le produit.

I.2.2 FP.ATMI

Cette arithmétique répond aux spécifications de FPAL-INTEL. Elle a été programmée initialement pour les tests sur l'outil de développement. Toutefois elle présente l'avantage d'un encombrement mémoire moins important (mais en conséquence, ses performances en temps d'exécution et traitement des erreurs sont dégradées). La possibilité d'implanter sur PROM uniquement les opérations voulues, aux adresses voulues, augmente encore le gain de mémoire et de ce fait cette arithmétique a pris place dans des applications réelles.

I.2.3 AM9511

Le circuit AM9511 se présente sous la forme d'un boitier de 24 broches. Les fonctions réalisées sont multiples : arithmétique flottante, entière sur 16 et 32 bits, fonctions trigonomètriques, racine carrée, logarithmes et exponentiation. Nous nous intéressons uniquement à l'arithmétique à point flottant.

La représentation d'un nombre sur 32 bits est la suivante :

- bit 31 : signe de la mantisse (1 négatif, 0 positif)
- bits 30 à 24 : exposant en complément à 2
- bits 23 à 0 : mantisse normalisée, le bit 23 étant obligatoirement à 1

La valeur du nombre est donnée par la formule :

(0. + .mantisse) * 2**(exposant)

Ce qui permet la représentation des nombres décimaux de 2.7 10**(-20) à 9.2 10**18 .

Pour effectuer une opération il faut :

1- transmettre le ler opérande au circuit (4 transferts de 8 bits)
2- transmettre le 2ème opérande
3- transmettre un mot de commande afin d'activer l'opération désirée
4- extraire le résultat du circuit

Un registre d'état peut être accédé qui indique les anomalies rencontrées au cours du traitement.

Les transmissions se font par le bus des données du microprocesseur à l'aide d'instructions d'entrée/sortie ou simplement par des instructions de transfert de données suivant le montage choisi.

(Voir en annexe C une présentation succinte d'un montage simple de l'AM9511)

I.2.4 Remarques

De l'étude de ces différentes arithmétiques programmées et cablées se dégagent des analogies :

- La représentation des nombres se fait sur 32 bits
- L'exécution d'une opération nécessite 3 étapes principales (si l'on regroupe les étapes 2 et 3 de l'AM9511)
- Avec FPAL on adresse le dernier octet des nombres. Or on s'appercoit qu'il est commode d'employer la même méthode avec le mode de transmission de données de l'AM9511.

Ainsi on peut d'ores et déjà penser qu'il est possible de définir une méthode unique de communication entre opérateurs matriciels et dispositifs arithmétiques.

I.3 REGLES ET METHODES DE REALISATION DES OPERATEURS

I.3.1 LES REGLES DE BASE

Les opérateurs sont conçus suivant 4 règles principales :

Règle 1 : Représentation des matrices en mémoire

Elles doivent être rangées colonne par colonne par analogie avec le langage FORTRAN, ou les calculettes programmables, chaque élément étant représenté sur 4 octets.

Règle 2 : Communication avec le dispositif arithmétique

Elle se fait suivant les spécifications de FPAL-INTEL, c'est à dire comme nous l'avons vu, en trois étapes :

1- Chargement du 1er opérande dans l'accumulateur flottant
(FLOAD)
2- Activation de l'opération choisie
(FADD,FMUL....)
3- Récupération du résultat
(FSTOR)

Dans chacune de ces étapes les paires de registres BC et DE contiennent respectivement l'adresse du 18ème octet de l'espace FPR (zone de travail de l'arithmétique) et l'adresse du 4ème octet de l'opérande considéré.

(Voir annexe B un exemple d'appel).

La zone FPR est systématiquement réservée dans la pile au dessus des zones de travail propres aux opérateurs.

Bien entendu certaines opérations ne nécessitent pas de FLOAD si le 1er opérande se trouve déjà dans l'accumulateur flottant, et d'autres pas de FSTOR si elles ne représentent qu'un calcul intermédiaire.

Tout dispositif arithmétique peut s'accomoder de cette règle si l'on dispose de l'interface adéquat entre lui et l'opérateur matriciel. Pour l'AM9511 par exemple un interface a été programmé comportant toutes les fonctions nécéssaires :

FLOAD : transmet vers le circuit les 4 octets du 1er opérande pointé par DE

Fxxxx : transmet le 2ème opérande adressé par DE. Puis envoie le mot de commande correspondant à l'opération désirée

FSTOR : récupère le résultat et range les 4 octets en mémoire à l'adresse contenue dans DE

(Voir annexe C un listing des fonctions)

Dans le cas présent la zone FPR adressée par la paire BC n'est pas utilisée. Toutefois elle pourrait éventuellement servir dans des programmes d'interface ayant à faire des conversions de format entre la représentation en mémoire et la représentation demandée par le dispositif arithmétique.

Ainsi le format des nombres en mémoire (sur 32 bits) et le dispositif arithmétique employé sont transparents pour les opérateurs matriciels.

Règle 3 : Passage des paramètres par le programme appelant

Les paramètres sont stockés dans la pile avant l'appel de l'opérateur. Il s'agit de une à trois adresses de matrices (adresse du 1er octet de la 1ère colonne) et de deux dimensions, ou de trois dans le cas des opérateurs produit.

Les opérateurs sont de 4 types.

Type 1 : 1 matrice opérande, soit MR

Type 2 : 2 matrices opérandes, soit MR matrice résultat et M2 matrice origine

Type 3 : 3 matrices opérandes, soit MR matrice résultat, M2 matrice opérande 2 et M1 matrice opérande 1

Type M : identique au type 3 mais pour la multiplication

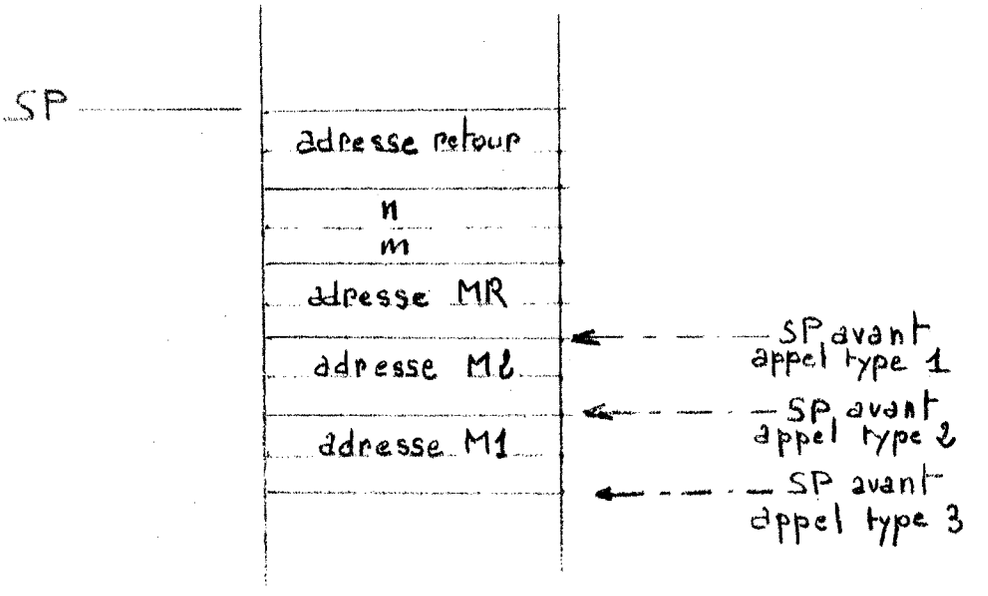
Si les dimensions des matrices sont telles que :

m = nombre de lignes
n = nombre de colonnes

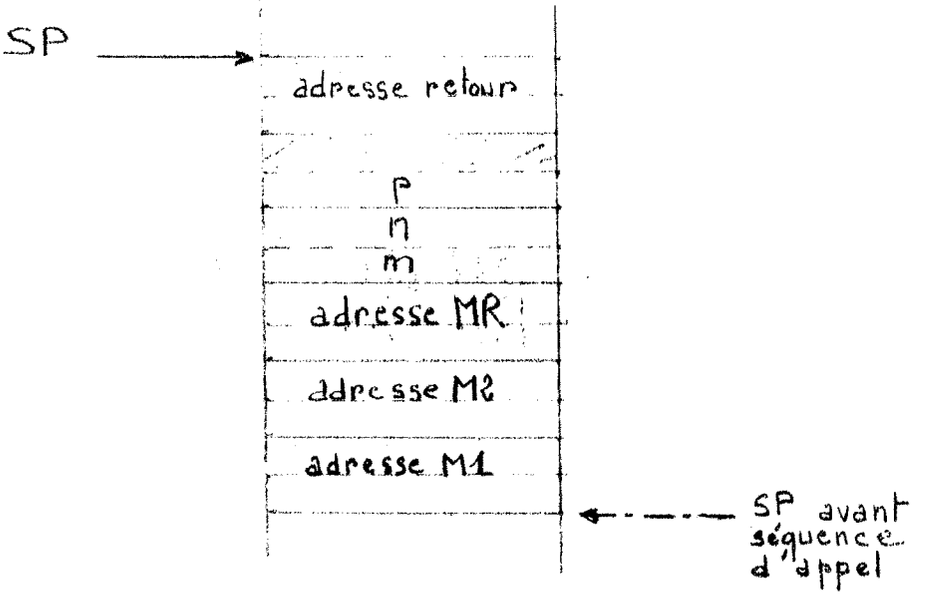
et pour la multiplication :

p = nombre de colonnes de la matrice opérande 2

alors au moment du débranchement vers l'opérateur la pile doit se présenter ainsi pour les types 1, 2 et 3 :



Et pour la multiplication (type M) :



L'octet hachuré, rempli initialement par le programme appelant pour spécifier une des trois options du produit, est utilisé dans la version actuelle par l'opérateur lui-même comme explicité au paragraphe I.4.

Règle 4 : Espace de travail des opérateurs.

Il se situe dans la pile au dessus de l'adresse de retour. Il est constitué :

- d'une zone de sauvegarde des registres
- d'une zone de stockage des valeurs courantes des indices et des pointeurs dans les matrices. Sa taille est variable suivant les opérateurs.
- de 18 mots destinés à l'arithmétique flottante (FPR) .

Le pointeur de pile (SP) adresse le 1er octet libre au-dessus de cet espace et ne pointe jamais plus bas dans la pile jusqu'au moment du retour vers le programme appelant.

Les opérateurs n'altèrent jamais ni la zone FPR de la pile, ni la zone de passage des paramètres qui sont des zones "appartenant" à d'autres programmes (exception faite pour la multiplication qui utilise le dernier octet de la zone paramètre) . De même ils sauvegardent systématiquement l'état des regitres.

I.3.2 APPERCU SUR LES METHODES DE PROGRAMMATION

Comme nous l'avons vu la paire de registres BC doit contenir l'adresse de la fin de la zone FPR au moment de l'appel des fonctions arithmétiques, aussi ces registres sont-ils initialisés à cette valeur et restent inchangés durant tout le traitement de l'opérateur.

Les pointeurs des éléments dans les différentes matrices sont stockés en espace de travail dans la pile. Afin de les incrémenter on y accède par un adressage de type Base + Déplacement de la manière suivante :

- SP sert de base (sa valeur reste en effet constante pour l'opérateur)
- HL est initialisé avec un certain déplacement par rapport au haut de la pile
- Le contenu de SP est ajouté au contenu de HL
- Les deux octets de la pile adressés ainsi par HL contiennent le pointeur désiré. Ces deux octets sont chargés dans la paire DE pour être incrémentés ou pour l'appel des fonctions arithmétiques.

Les ressources du 8080 ne permettent guère d'autres méthodes d'évolution des indices des matrices. Aussi le temps passé pour la gestion des pointeurs n'est-il pas négligeable par rapport aux traitements purement arithmétiques, surtout si un dispositif cablé rapide est utilisé.

Le pointeur de pile adressant toujours un mot situé au-dessus des espaces de travail, les opérateurs sont :

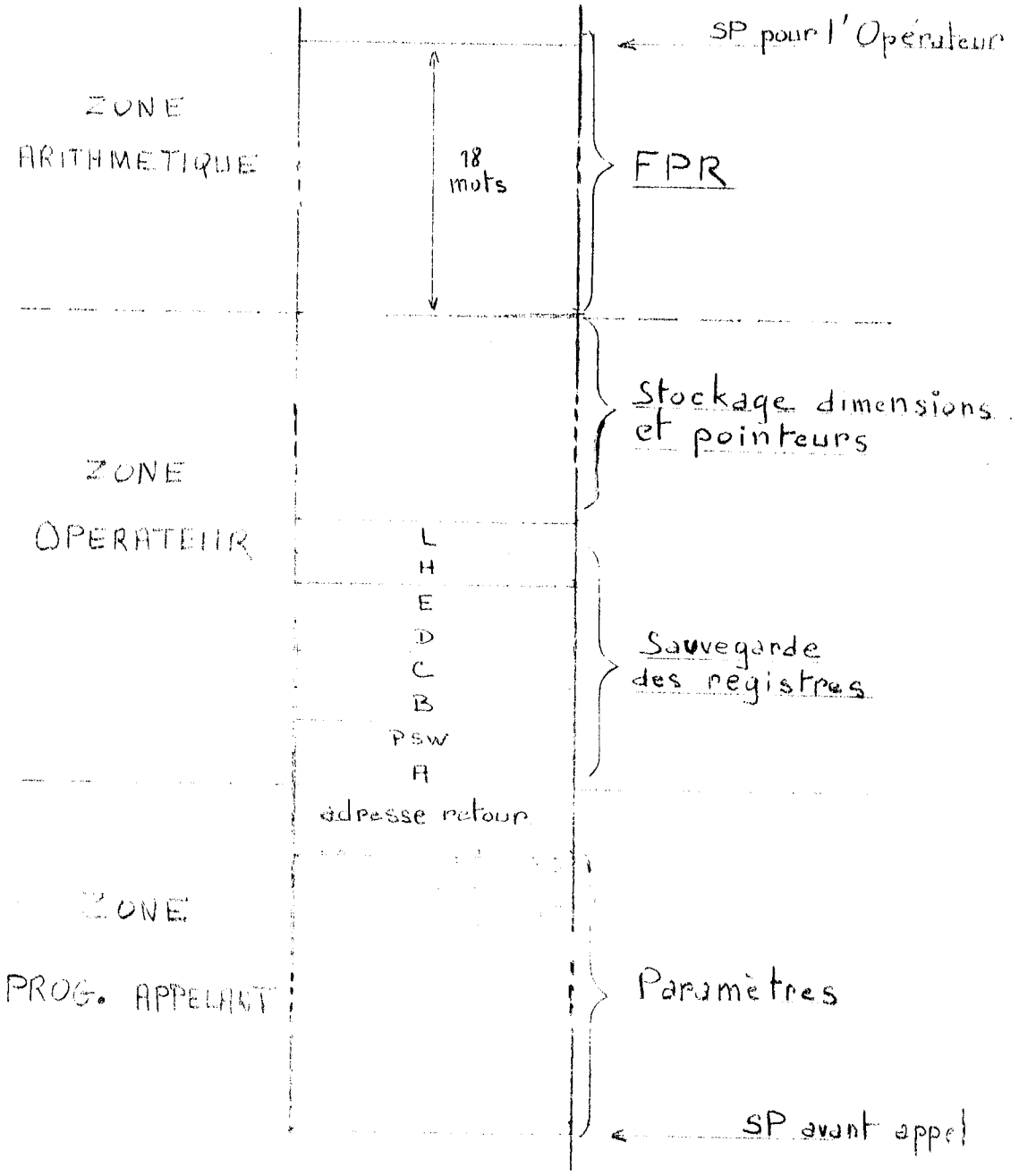
- interruptibles , l'exécution d'un autre programme pouvant être lancé s'il respecte les conventions d'utilisation de la pile

- réentrants , un programme activé après une interruption pouvant utiliser à son tour l'opérateur interrompu.

Ceci sous réserve de la sauvegarde des registres par la routine de traitement des interruptions et d'une arithmétique flottante possédant les mêmes qualités que les opérateurs, ce qui est le cas pour celles citées plus haut.

Enfin les opérateurs ne font aucune vérification sur la validité des paramètres qui leur sont fournis, et les conséquences des erreurs à ce niveau sont imprévisibles.

– Schéma général de l'utilisation de la pile –



I.4 NOTES D'UTILISATION DES OPERATEURS

I.4.1 TABLEAU RECAPITULATIF

On rappelle les 4 types d'opérateurs :

Type 1 : 1 opérande , MR = .opérateur.MR

Type 2 : 2 opérandes , MR = .opérateur.M2

Type 3 : 3 opérandes , MR = M1.opérateur.M2

Type M : Produits , MR = (MR ±) M1 * M2
(3 dimensions)
Il existe 3 opérateurs produits comme indiqué dans le tableau ci-dessous. Initialement l'opérateur choisi était sélectionné à l'aide d'un indicateur placé dans le dernier octet des paramètres. Il est apparu par la suite plus commode pour l'utilisateur de faire appel à 3 opérateurs différents. Toutefois cet octet est positionné automatiquement par les opérateurs produits constituant ainsi l'exception à la règle de non ingérance d'un programme dans une zone de la pile ne lui étant pas affectée.

Nom symb.	Fonction	Type	Occupation pile	Appel arithmétique
FMRAZ	MR = [0]	1	34	oui
FMNEG	MR = -MR		34	oui
FMUNI	MR = [I]		38	oui
FMTRP	MR = [M2]^T	2	20	non
FMMOV	MR = M2		18	non
FMADD	MR = M1 + M2	3	44	oui
FMSUB	MR = M1 - M2		44	oui
FMPRD	MR = M1 * M2	M	50	oui
FMADPR	MR = MR + M1 * M2		50	oui
FMSUPR	MR = MR - M1 * M2		50	oui

I.4.2 EXEMPLE D'APPEL

La séquence d'instructions suivante appelle le 1er opérateur produit :

```
LXI   H,0
DAD   SP        save SP dans HL
LXI   B,M1
PUSH  B         adresse matrice M1 dans la pile
LXI   B,M2
PUSH  B         adresse matrice M2 dans la pile
LXI   B,MR
PUSH  B         adresse matrice MR dans la pile
LXI   D,0402H
PUSH  D         m=4 et n=2 dans la pile
MVI   D,03H
PUSH  D         p=3 dans la pile (dernier octet non init.)
CALL  FMPRD
SPHL            resto. pointeur de pile
```

I.4.3 PERFORMANCES

Les temps sont donnés ici à titre indicatif. Il proviennent de tests réalisés sur l'émulateur 8080 de l'outil de développement Tektronix-8002. L'arithmétique utilisée est celle développée pour les tests sur cet outil. Tous les éléments des matrices mises en jeu étaient de valeur moyenne différente de zéro. Ces temps comprennent la séquence d'appel de l'opérateur.

En considérant le nombre d'opérations elémentaires mises en jeu par l'opérateur et à l'examen des résultats, on peut dégager une loi empirique (du moins pour les plages de dimensions testées) :

- Pour le produit : temps ms. = 12.5*m*n*p

- Pour l'addition : temps ms. = 2.2*m*n

- Temps d'exécution des opérateurs -

m	n	p	Produit	Addition
2	2	1	50 ms.	10
		2	99	
3	3	1	112	21
		3	335	
4	4	1	199	34
		4	793	
5	5	1	311	57
		5	1550	
6	6	1	447	82
		6	2684	
7	7	1	610	112
		7	4272	
8	8	1	798	144
		8	6384	
9	9	1	1010	183
		9	9106	
10	10	1	1252	226
		10	12514	

I.5 LOGICIEL D'AIDE A LA MISE AU POINT DES PROGRAMMES

L'expérience montre qu'un temps très important est consacré à la mise au point des programmes écrits en langage d'assemblage. Pour les applications sur microprocesseur , un outil de développement permettant par exemple de dérouler les instructions pas à pas, de placer des points d'arrêt et d'examiner des mots en mémoire réduit considérablement ce temps. Toutefois dans le cas présent, la manipulation des données aussi bien que la vérification des tests par examen direct de la mémoire sont extrèmement malaisés du fait que la valeur d'un nombre flottant sur 4 octets est loin d'être immédiate pour l'observateur.

Aussi pour permettre d'entrer aisément des données flottantes en mémoire et de visualiser les résultats des calculs, des routines de conversions de nombres décimaux sous forme de chaines de caractères ASCII en binaire interne, et inversement ont été développées. Ces routines seront ensuite intégrées dans des opérateurs de saisie et de visualisation de matrices.

I.5.1 ROUTINES DE CONVERSION DES NOMBRES FLOTTANTS

Elles sont réalisées de manière modulaire afin de faciliter les modifications pour passer d'un format à un autre.

Elles ont été programmées dans le sens d'une minimisation de l'occupation mémoire dans l'optique d'une implantation sur application réelle : la version la plus élaborée occupe environ 2,5 K-octets de mémoire et nécessite 90 octets de RAM; une version restreignant la syntaxe d'entrée en ASCII tient sur 2 K. Dans ce but aucune table de conversion n'est utilisée, les conversions des mantisses se faisant par recalculs successifs des puissances de 2 ou de 10 nécessaires, de même pour la décomposition des exposants. Cette méthode pénalise sévèrement les performances de temps des routines, mais ceci n'a que peu d'importance puisque ces temps restent minimes devant les délais de saisie ou d'affichage sur les dispositifs d'entrée/sortie.

Notons qu'un développement de routines identiques pour le microprocesseur MOTOROLA 6800, entrepris à titre de comparaison sur Exorciser, mais dans l'optique opposée d'un emploi systèmatique de tables de conversions, montre que l'obtention d'une précision identique nécessite environ 5 k de mémoire.

FPCVB
(Conversion d'un décimal ASCII en binaire)

- En entrée : le nombre décimal doit être sous forme d'une chaîne de caractères ASCII pointée par la paire HL. Cette chaîne de caractères est analysée jusqu'à la rencontre du caractère "Retour chariot" (OD en héxadécimal).

Syntaxe d'entrée : (graphes en annexe D)

$$\left[\left\{\begin{matrix}+\\-\end{matrix}\right\}\right]\left\{\begin{matrix}n\,[n\cdots n][\,.n[n\cdots n]]\\ .n[n\cdots n]\end{matrix}\right\}[E[\pm]n[n]]$$

Pour la version restreinte :

$$\left\{\begin{matrix}+\\-\end{matrix}\right\}\quad n.n[n\cdots n]\,[E\{\pm\}nn] \qquad (0\leq n\leq 9)$$

En sortie : La paire HL contient l'adresse du dernier octet du nombre flottant obtenu. Le bit de retenue (CY) est positionné à 1 si une erreur a été détectée.

Structure de la routine : elle se compose de 6 modules.

FPASYN : Analyse syntaxique

FPMEFD : Mise sous format DCB (Décimal Codé Binaire) avec recherche du meilleur cadrage pour l'optimisation de la précision et la diminution maximale de l'exposant.

FPCVBM : Conversion de la mantisse par divisions successives par 2 de la partie entière et multiplications par 2 de la partie fractionnaire.

FPCVBE : Conversion de l'exposant décimal en une suite de puissances de 2. Le premier terme de la suite représente l'exposant binaire final. On doit alors procéder à une suite d'additions de la mantisse décalée à droite autant de fois que de puissances de 2 successives rencontrées. la mantisse finale est ensuite normalisée.

FPMEFB : Mise sous format interne définitif. C'est cette routine qu'il faut modifier quand on change de format interne.

FPSPG : ensemble de sous-programmes généraux utilisés aussi par la routine de conversion inverse (décalages 4 ou 8 bits de chaînes d'octets, transferts de chaines, arithmétique en DCB etc.).

FPCVD
(Conversion d'un binaire en décimal ASCII)

En entrée : la paire HL doit contenir l'adresse du dernier octet du nombre binaire à convertir.

En sortie : HL contient l'adresse d'une chaîne de caractères terminée par le caractère retour chariot (0D héxadécimal). Format de sortie :

{ [±] INF
  0.0
  [±] n.nnnnnn [E [±] nn] } (= nombre infini)

- Structure de la routine :

FPCVDM : Conversion de la mantisse par calculs successifs en DCB des 24 puissances de 2 nécessaires avec additions des puissances concernées. La précision est augmentée en diminuant le plus possible les exposants positifs par décalages gauches de la mantisse.

FPCVDE : Conversion en DCB de l'exposant binaire en une suite de puissances de 10. Le premier terme de la suite représente l'exposant décimal. Son coefficient est appliqué à la mantisse. On procède ensuite aux additions successives de la mantisse, décalée de 4 bits à chaque pas, et multipliée par le coefficient correspondant.

FPMEFC : Mise sous format externe.

Actuellement la bibliothèque de modules de conversions en service se présente comme suit :

Conversion en binaire

Version normale FPAL-INTEL	Version réduite FPAL-INTEL	Version normale AM9511
FPASYN	FPASYNL	FPASYN
FPMEFD	FPMEFDL	FPMEFD
FPCVBM	idem	idem
FPCVBE	idem	idem
FPMEFB	idem	AMMEFB
FPSPG	idem	idem

Conversion en chaine ASCII

Version FPAL	Version AM9511
FPCVDM	AMCVDM
FPCVDE	idem
FPMEFD	idem
FPSPG	idem

1.5.2 OPERATEURS MATRICIELS DE DIALOGUE

Par analogie aux opérateurs matriciels utilisant un dispositif arithmétique, deux opérateurs utilisant les routines de conversion en conjonction avec des routines d'Entrée/Sortie spécifiques au matériel utilisé ont été développés : FMCONI et FMCONO, opérateurs de type 1, permettant respectivement de saisir (CONsole Input) et d'afficher (CONsole Output) des matrices.

L'appel de ces opérateurs obéit aux règles générales d'appel des opérateurs de type 1.

Les communications entre opérateurs et routines externes se font comme suit.

FMCONI : pour chaque élément de la matrice, et ceci colonne par colonne, l'opérateur appelle une routine d'entrée (SPCONI) qui lui fournit en retour une chaine de caractères pointée par HL. Cette chaine est soumise à son tour à la routine de conversion en binaire (FPCVB) suivant les règles décrites plus haut. Si aucune erreur n'a été détectée, le nombre ainsi converti est rangé en mémoire dans l'espace alloué à la matrice.

FMCONO : chaque élément binaire de la matrice donnée en paramètre est converti à l'aide de la routine de conversion en décimal ASCII (FPCVD) suivant les règles déjà citées. La chaine de caractères ainsi obtenue, pointée par HL, est passée à la routine de sortie (SPCONO) qui se charge de l'affichage.

En cours de programme on peut ainsi aisément, par simples appels d'opérateurs, placer des matrices en mémoire à partir du clavier par exemple (ou d'une disquette si l'on dispose de la routine SPCONI adéquate) , et afficher tous les calculs intermédiaires d'un algorithme. L'emploi de tels outils a permis de diminuer considérablement les temps de tests des programmes, qu'il s'agisse des opérateurs ou des algorithmes les utilisant.

(On trouvera en annexe D des exemples de saisie et d'affichage de matrices).

I.6 LOGICIEL D'AIDE A LA PROGRAMMATION.

Les séquences d'appel des opérateurs matriciels nécessitent en moyenne une dizaine d'instructions. Outre que la répétition de cette séquence dans un programme est fastidieuse elle est la source d'erreurs d'écriture qui entrainent une augmentation des temps de tests car aucune vérification n'est faite quant à la validité des paramètres par les opérateurs eux-mêmes.

Les erreurs les plus courantes portent sur la non concordance entre le nombre de paramètres fournis et le type de l'opérateur appelé, ou la non correspondance des dimensions passées et celles utilisées en fait pour définir la matrice.

Un des moyens de palier à cela est l'utilisation de macro-instructions si l'on dispose d'un macro-assembleur. Un autre moyen est le développement d'assembleurs acceptant une syntaxe particulière de manipulation de matrices et d'appel d'opérateurs.

I.6.1 EXEMPLE DE MACRO-INSTRUCTIONS.

La règle principale est de définir des symboles, soit par équivalence, soit comme étant des adresses en mémoire contenant des dimensions de matrices. Alors seuls les noms symboliques des matrices seront évoqués dans les appels d'opérateurs, les dimensions étant automatiquement retrouvées par le macro-assembleur. Ceci permet en outre de faire dans certains cas des vérifications sur la concordance des dimensions des différentes matrices entrant en jeu.
De même on se donne le moyen de vérifier dans la macro-instruction la concordance entre le nombre de paramètres fournis et le type d'opérateur appelé.

Un peut distinguer trois modes de dimensionnement des matrices dans les programmes :

1- Les dimensions des matrices sont connues au moment de l'écriture des programmes (Dimensions statiques)

2- Les dimensions sont affectées par le programme au cours de son exécution (dimensions dynamiques)

3- Toutes les dimensions des matrices mises en jeu font partie d'un ensemble restreint de valeurs défini par ailleurs au moment de l'exécution.

Nous avons défini trois niveaux de macro-instructions , chaque niveau prenant en compte les différents modes de définition des matrices :

- Déclaration de dimensions
- Réservation d'espace mémoire
- Appel des opérateurs matriciels

Les développements de ces trois niveaux sont donnés ci-dessous avec comme illustration les macro-instructions utilisées sur le système Tektronix-8002.

Déclaration de dimensions : MXDIM matr, m, (,n)

où matr est le nom symbolique contenant l'adresse de la matrice, m et n sont des entiers représentant le nombre de lignes et le nombre de colonnes de la matrice (n=1 par défaut).

La macro génère les deux équivalences suivantes :
- matr.1 EQU m
- matr.2 EQU n

Cette macro est utilisée dans le mode 1 de dimensionnement statique. Outre l'avantage de définir clairement les dimensions d'une matrice, les équivalences générées vont déclencher les tests de validité sur les dimensions dans la macro-instruction d'appel des opérateurs. Pour cette raison elle doit être placée pour l'assemblage avant la première utilisation de la matrice.

Réservation d'espace mémoire :

```
MXRES     matr,maxm (,maxn)
MXRES.    matr,m     (,n)
MXRES..   matr
```

La macro MXRES.. utilisant les dimensions matr.1 et matr.2 ne peut être utilisée qu'en mode 1 et seulement si la matrice a été définie par MXDIM dans le même assemblage, sinon il faut utiliser MXRES. . Ces deux macros génèrent le code de réservation mémoire pour les éléments de la matrice :

```
soit,   matr   BLOCK   m*n*4
soit,   matr   BLOCK   matr.1*matr.2*4
```

MXRES assure cette même fonction, mais en plus elle réserve deux octets de mémoire qui contiendront les dimensions dynamiques des matrices affectées par programme. Ces dimensions sont initialisées au moment de l'assemblage avec les valeurs maximales fournies, soit la génération suivante :

```
matr$   BYTE    maxn
        BYTE    maxm
matr    BLOCK   maxm*maxn*4
```

MXRES est utilisée spécialement en mode 2 , les dimensions devant être stockées par le programme dans

les deux octets adressés par le symbole matr$.

Appel des opérateurs :

En mode 1 et 2 on utilise la même macro-instruction qui reconnaît le mode utilisé en testant le dimensionnement préalable des matrices :

- FMCALL opérateur, (matra,) (matrb,) matrr

Cette macro génère la séquence d'appel correspondant au type de l'opérateur spécifié. S'il y a une discordance entre le type de l'opérateur et le nombre de matrices paramètres un message est imprimé. Un test est fait sur l'existence des symboles créés par la macro MXDIM (de type matr.1 et matr.2). Si ces symboles existent ils sont utilisés comme facteurs immédiats dans le stockage des dimensions dans la pile, et de plus des tests sont faits sur la concordance des dimensions des différentes matrices paramètres. Si une anomalie est rencontrée un message est imprimé.
Si ces symboles n'existent pas les dimensions sont extraites des mots mémoire adressés par le symbole de la forme matr$.
Le pointeur de pile est restauré à sa valeur précédent la séquence d'appel.

Dans le mode 3 les dimensions des matrices représentent un nombre restreint de valeurs (4 par exemple pour l'application décrite en partie II). Aussi pou éviter l'initialisation par programme de toutes les zones de type matr$, avec des valeurs souvent identiques, les dimensions sont fournies à la macro-instruction d'appel:

- MDCALL opér., (matra,) (matrb,) matrr, m, n (,p)

Cette macro, comme la précédente, génère la séquence d'appel suivant l'opérateur utilisé, détecte les anomalies sur le nombre de paramètres, mais ne fait évidemment aucune vérification sur les dimensions.

I.6.2 UN ASSEMBLEUR CROISE SPECIALISE

Le prototype d'un assembleur croisé acceptant une syntaxe particulière de manipulation de matrices a été mis au point à l'aide du générateur d'assembleurs "GAGE" développé à l'Atelier de Microinformatique par Y. BEKKERS et P. FONTANILLE.

La particularité de ce produit est qu'il offre la possibilité de modeler aisément les analyseurs des assembleurs grâce à un générateur d'analyseurs syntaxique et lexicographique (GASEL) ,construisant des tables d'analyse à partir d'un langage de description. Les programmes écrits en FORTRAN sont facilement portables.

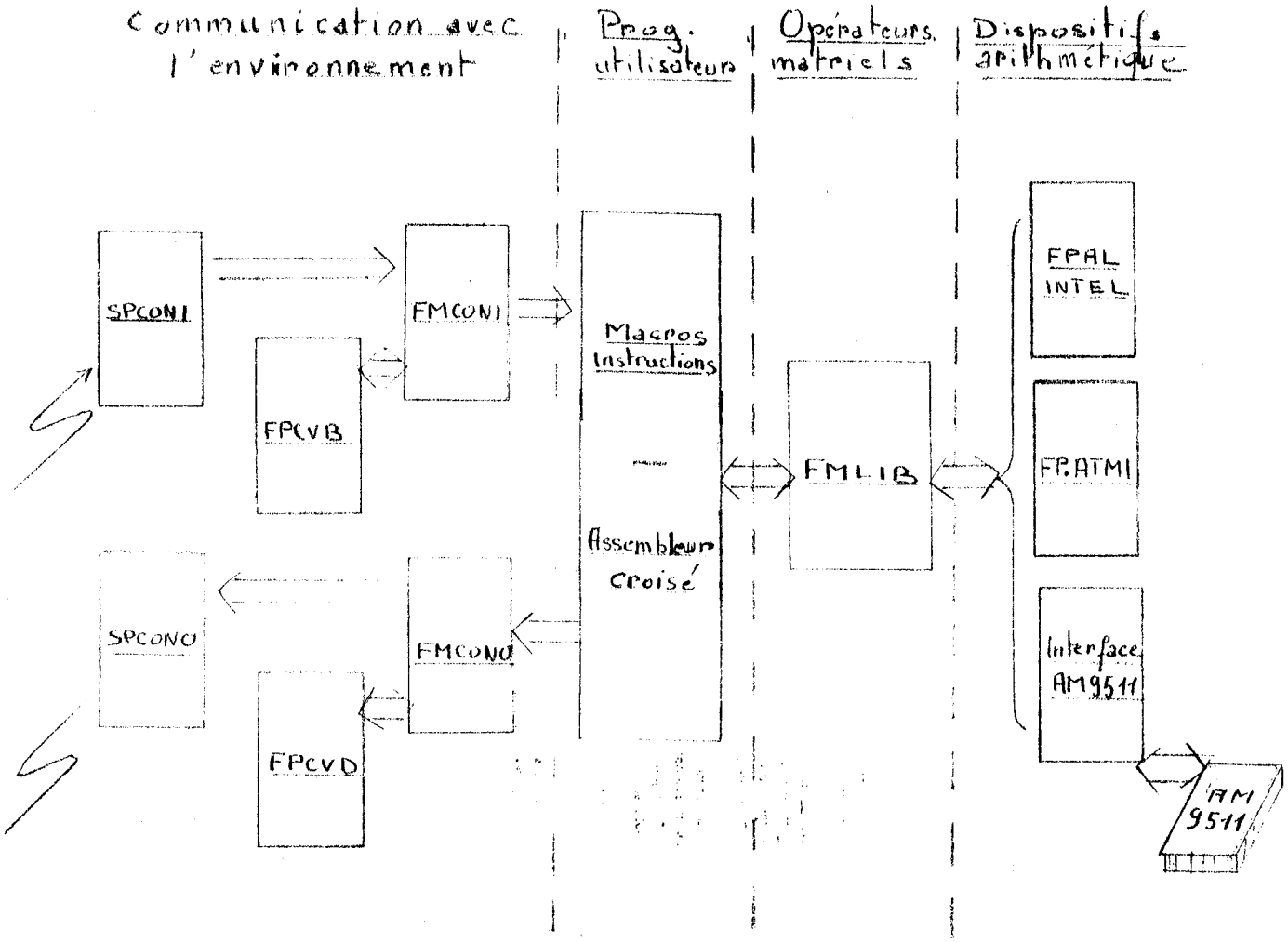
L'assembleur prototype généré comporte une syntaxe de dimensionnement statique des matrices (les dimensions étant rangées dans la table des symboles), et une syntaxe d'appel des opérateurs claire et "parlante" utilisant la classification par type des opérateurs.

Cette approche est intéressante, car s'il est souvent impossible d'utiliser des langages évolués standards générant au mieux deux fois plus de code qu'un langage d'assemblage, le développement d'outils de ce type, peu gourmands en espaces mémoire et portables, peuvent permettre de simplifier beaucoup la tâche du programmeur dans des domaines spécialisés.

(On trouvera en annexe E des exemples d'utilisation de ces 2 outils)

I.7 RECAPITULATIF DES OUTILS LOGICIELS DISPONIBLES

Le schéma ci-dessous visualise par fonctions les outils logiciels disponibles à ce stade.



DEUXIEME PARTIE

* * *

DEVELOPPEMENT D'ALGORITHMES DE COMMANDE AUTOMATIQUE MULTIVARIABLE STRUCTURABLES EN TEMPS REELS

II.1 PRESENTATION GENERALE DES PROBLEMES D'AUTOMATISME

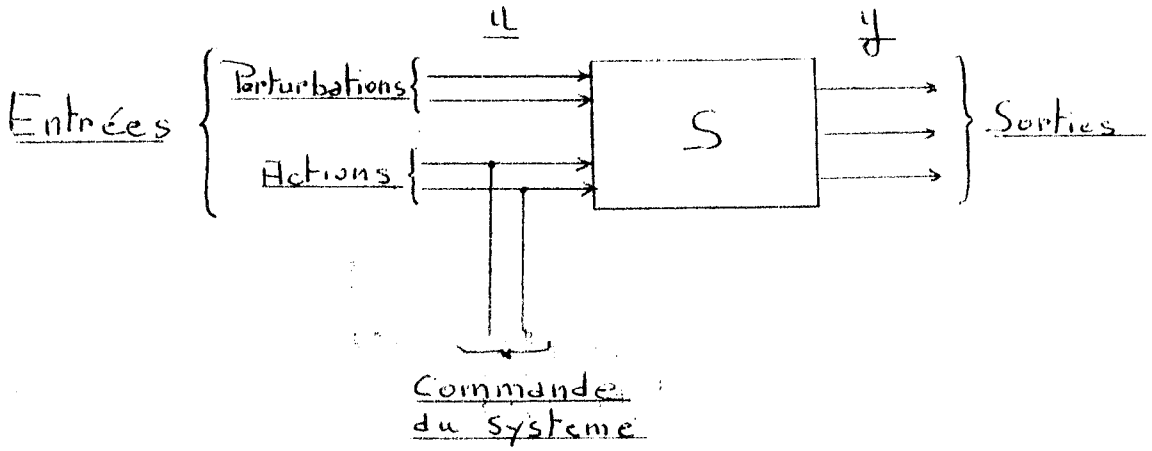
II.1.1 QUELQUES DEFINITIONS

Un système peut être schématisé par une "boite noire" à laquelle on applique des entrées et qui fournit des sorties. Les entrées sont de deux types :

- Les perturbations (ou contraintes) indépendantes de la volonté du décideur

- les actions décidées et appliquées au système et qui en constituent véritablement la commande.

Assurer la conduite du système c'est calculer et appliquer les commandes adéquates afin d'obtenir les sorties désirées.



Le calcul de la commande utilise une représentation mathématique du procédé : son modèle, obtenu soit à partir des lois du procédé (modèle de connaissance) soit par identification à partir d'une suite de mesures sur les Entrées/Sorties (modèle de représentation). D'une manière générale les systèmes linéaires sont modélisés suivant des systèmes d'équations différentielles reliant les entrées et les sorties.

La représentation d'état permet de représenter à tout instant les états du procédé. Ces états peuvent ou non avoir une signification physique. Dans le cas d'une schématisation de type analogique du procédé , conçue à partir des équations différentielles, ils représentent les sorties des intégrateurs.

Les méthodes de commande par calculateurs numériques imposent une discrétisation de la représentation qui permette de travailler sur des variables considérées comme constantes pendant des périodes d'échantillonnages d'une durée T. Alors à chaque pas d'échantillonnage i le procédé est représenté par le modèle :

$$x(i+1) = Ax(i) + B\,u(i)$$
$$y(i) = Cx(i)$$

où

x(n) est le vecteur d'états
u(r) le vecteur des commandes
y(m) le vecteur des sorties
A(n,n) est la matrice d'état
B(n,r) la matrice d'entrée
C(m,n) la matrice de sortie

Du fait de leur linéarité ces modèles ne représentent le procédé d'une manière satisfaisante qu'autour d'un point de fonctionnement , c'est à dire seulement dans une certaine plage de valeurs des entrées/sorties. Quand les écarts deviennent trop importants il faut changer de modèle.

II.1.2 COMMANDE PAR POURSUITE D'OBJECTIFS SELON UN CRITERE QUADRATIQUE

Reprenons le modèle d'un système linéaire défini précédemment :

$$x(i+1) = Ax(i) + B\,u(i)$$
$$y(i) = Cx(i)$$

Si zi représente l'objectif à poursuivre pour les entrées et zo l'objectif à poursuivre pour les sorties, le problème est de calculer à chaque pas d'échantillonnage la commande u(i) qui assure le mieux cette poursuite. C'est à dire qui minimise un critère quadratique exprimant les écarts entre objectifs et variables d'entrée/sortie :

$$J = \sum_{i=1}^{N} ei^{T}(i)\,R\,ei(i) + eo^{T}(i+1)\,Q\,eo(i+1)$$

où ei = zi - u
eo = zo - y
R et Q sont des matrices de pondération entre les différents écarts.

La solution du problème est fonction de la classe d'évolution des objectifs. Dans les deux cas où :

- zi et zo sont constants
- zo est sortie d'un systéme linéaire d'entrée zi (N-->infini)

la commande s'écrit

$$u(i) = -Lx(i) + n(i)$$
$$u(i) = \text{bouclage} + \text{anticipation}$$

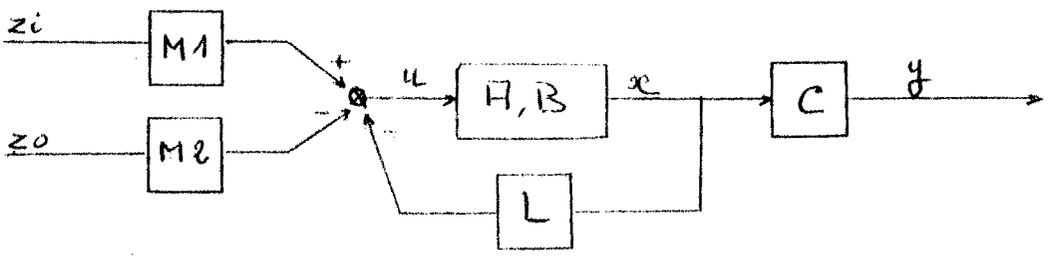
la partie anticipation étant fonction des objectifs à l'instant d'échantillonnage i.

II.1.3 ALGORITHMES DE BASE

Dans le cas d'<u>objectifs constants</u> la commande s'écrit :

$$u(i) = -Lx(i) + M1zi - M2zo$$

et l'on peut en donner une représentation schématique :



Nous donnons à titre indicatif les méthodes d'obtention des matrices à partir des matrices du modèle (A,B,C).

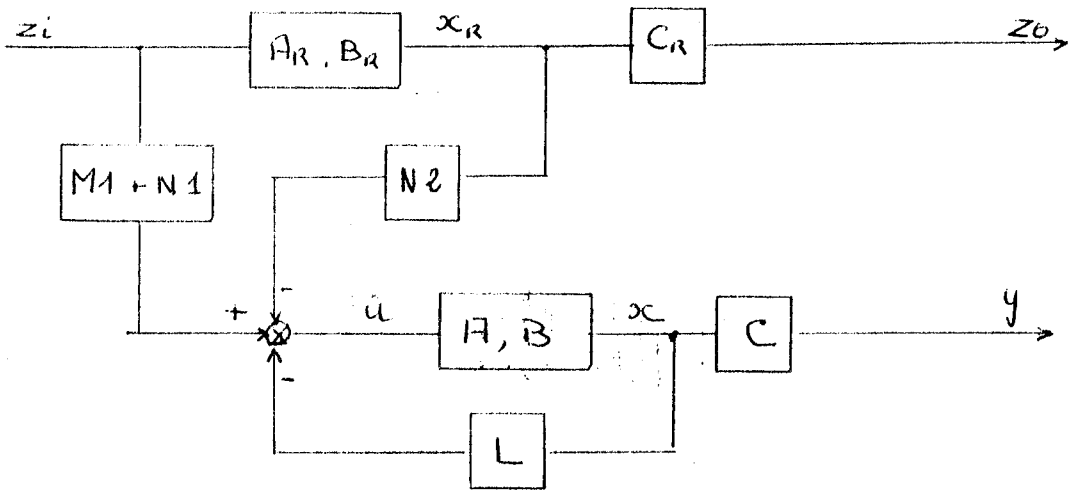
$$L = SB^T KA$$
$$S = (R + B^T K B)^{-1}$$
K étant solution de l'équation de RICATTI:
$$K = A^T KA - A^T KBSB^T KA + C^T QC$$
$$M1 = SR^T (G^T - I)^{-1} L^T R + SR$$
$$M2 = SB^T (G^T - I)^{-1} C^T Q$$
$$G = A - BL$$

Dans le cas de la poursuite d'un modèle de référence, c'est à dire où zi reste constant et zo est donné par :

$$x_R(i+1) = A_R x_R(i) + B_R zi$$
$$zo\ (i) = C_R x_R(i)$$

la commande s'écrit :

$$u(i) = -Lx(i) + M1zi + N1zi - N2x_R(i)$$



Pour les deux types d'algorithmes précédents s'il existe des contraintes sur les entrées et les sorties telles que d'une manière générale :

$$u_{min}(i) \leq u(i) \leq u_{max}(i)$$
$$y_{min}(i) \leq y(i) \leq y_{max}(i)$$

il y a apparition d'un terme correctif sur les variables contraintes si celles-ci venaient à sortir de leur limites. Ces termes correctifs s'obtiennent par un déplacement des objectifs sur ces variables :

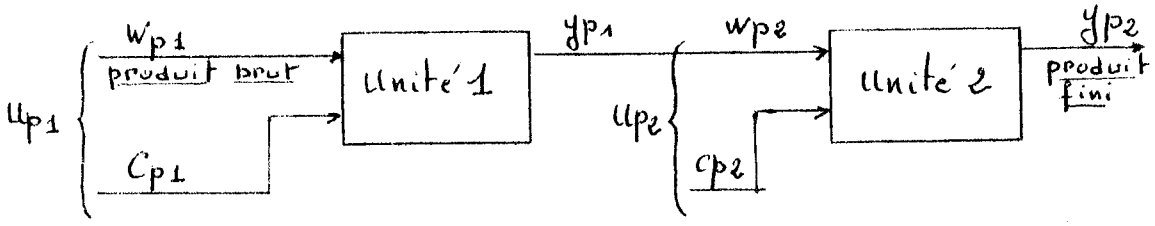
$$u(\cdot) = u_0(\cdot) + M1.\Delta zi(\cdot) + M2.\Delta zo(\cdot)$$

avec $u_0(\cdot)$,commande sans contraintes
$\Delta zi, \Delta zo$,variations calculées des objectifs correspondant aux variables contraintes afin de les maintenir dans leurs limites de validité.

II.2 LE PROCEDE PILOTE DU L.A.G. LE PROJET IMPACT

II.2.1 STRUCTURE MODULAIRE DE LA COMMANDE

Le procédé pilote est constitué de deux colonnes de distillation qui pour le développement présent seront toujours montées en série. Ces deux unités fournissent un produit fini (sortie de la deuxième unité = yp2) à partir d'un produit brut (entrée contrainte de la première unité = wp1). La sortie de la première unité (yp1) constitue une entrée interaction pour la deuxième (wp2). De plus chaque unité possède sa commande propre (cp1 et cp2) :



Et si l'on considère le procédé globalement, il apparaît comme un système d'entrée up = (wp1, cp1, cp2) et de sortie yp = yp2.

On donne un exemple de la composition de ces vecteurs :

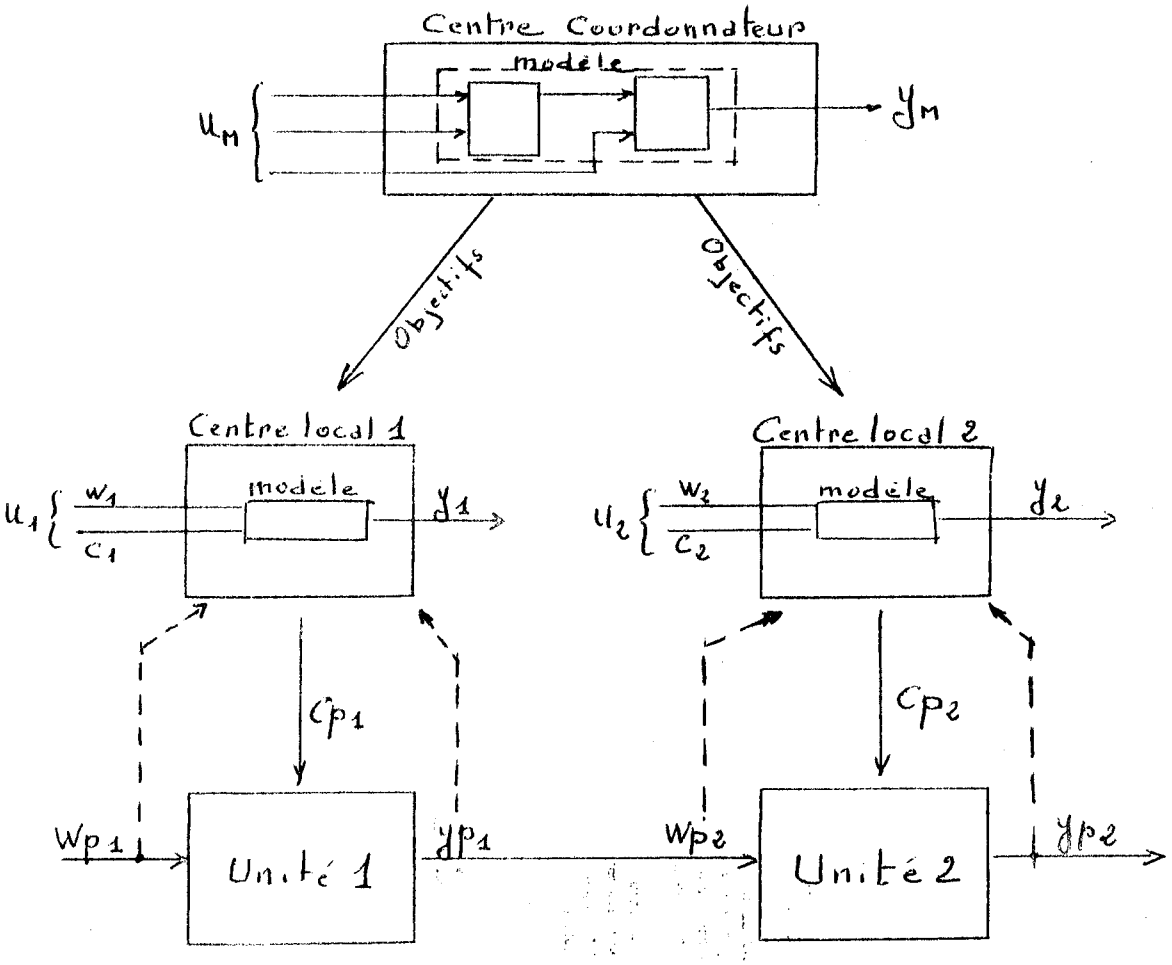
wp = concentration et débit du produit en entrée
yp = concentration et débit du produit en sortie
cp = puissance de chauffe et reflux

La commande appelée commande décentralisée coordonnée va être structurée suivant trois niveaux :

- un niveau d'instrumentation chargé de l'acquisition , du traitement et de la communication des mesures et jouant un rôle de régulation locale

- un niveau de commande locale multivariable possédant un modèle du centre qu'il commande. Les objectifs poursuivis sont alloués par le niveau supérieur. La commande tient compte éventuellement des interactions provenant des centres de même niveau. Elle acquiert les mesures élaborées par le niveau d'instrumentation et lui renvoie la commande calculée.

- un niveau coordonnateur possédant un modèle simplifié du procédé global obtenu par agrégation des modèles locaux, de telle sorte que toutes les variables d'entrée/sortie des niveaux locaux y apparaissent.



II.2.2 UN ALGORITHME STANDARD DE COMMANDE MULTIVARIABLE

Dans la structure de commande telle qu'elle vient d'être définie, on peut généraliser les fonctions à remplir par chaque centre de décision :

1- Poursuite des objectifs alloués par le niveau supérieur

2- Respect des interactions apportées par les centres voisins de même niveau

3- correction des écarts constatés entre le calcul effectué par le centre et les informations de retour communiquées par le niveau inférieur

4- Elaboration de directives à transmettre aux centres du niveau inférieur directement dépendants.

Il est évident que selon sa situation dans la structure de commande chaque centre remplit différemment ces fonctions, ou même n'a pas à en remplir certaines. Toutefois il reste possible de ne définir qu'un seul algorithme façonnable par une fonction indépendante de structuration de l'algorithme. Cette possibilité nécessite une allocation dynamique des tailles des matrices et un positionnement d'aiguillages adéquats pendant l'exécution des programmes , c'est à dire en temps réel.

On appellera un tel algorithme "l'algorithme standard de commande multivariable". Les tâches qu'il remplit dérivent des algorithmes de base développés en II.1, et sont décrites ci-dessous sans démonstration supplémentaires :

1- Poursuite d'objectifs

- modèle $x = Ax + Bu$
  $y = Cx$

- objectifs constants (pour le centre coordonnateur ou un centre de commande isolé)

  $u = -Lx + M1zi - M2zo$

- objectifs obtenus par poursuite d'un modèle de référence (centre local poursuivant son image dans le coordonnateur)

  $u = -Lx + Mx_M - Nu_B$
  où x_M = états du modèle coordonnateur
  u_B = (zi, zo) objectifs du centre coordonnateur

2- Respect des interactions

Elles sont représentées dans notre cas par les relations :

w1 = wp1
w2 = y1

et s'expriment par le terme correctif suivant sur les objectifs :

Dzwi = w - ya (ya = y1 ou wp1)
Dzi = (MwDzi , 0)T
u = u - M1Dzi

3- Correction des écarts

Le correcteur utilise un modèle de l'unité construit avec les matrices A,B,C et qui s'écrit :

xc = Axc + Buc
yc = Cxc

La commande correctrice uc poursuit comme objectif l'écart e constaté entre les sorties du procédé et les sorties calculées, avec des objectifs d'entrée nuls :

e = y + yc - yp
uc = -Lxc - M2e

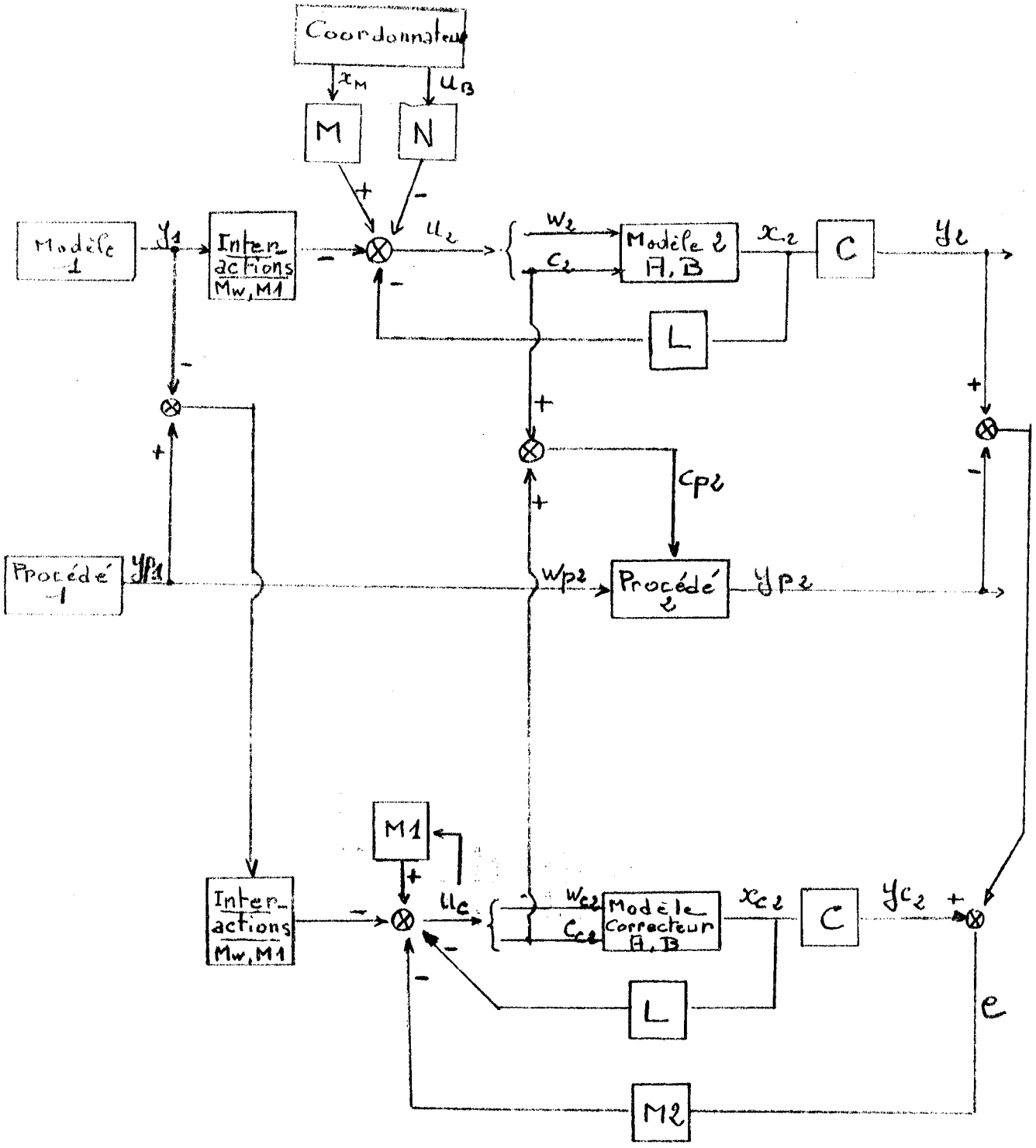
Mais on peut fixer comme objectif d'entrée à la commande correctrice sa valeur au pas précédent (cas d'une erreur statique nulle) :

uc = -Lxc + Mluc -M2e
pas i pas i-1

Enfin cette commande correctrice doit satisfaire à la contrainte imposée par l'écart constaté sur les sorties du centre voisin précédent. cette contrainte est calculée comme pour la commande u avec ya = ypa - ya.

La figure ci-dessous représente le schéma-bloc de l'algorithme le plus complexe appliqué au centre local commandant l'unité 2.

- Schéma-bloc de l'algorithme standard -

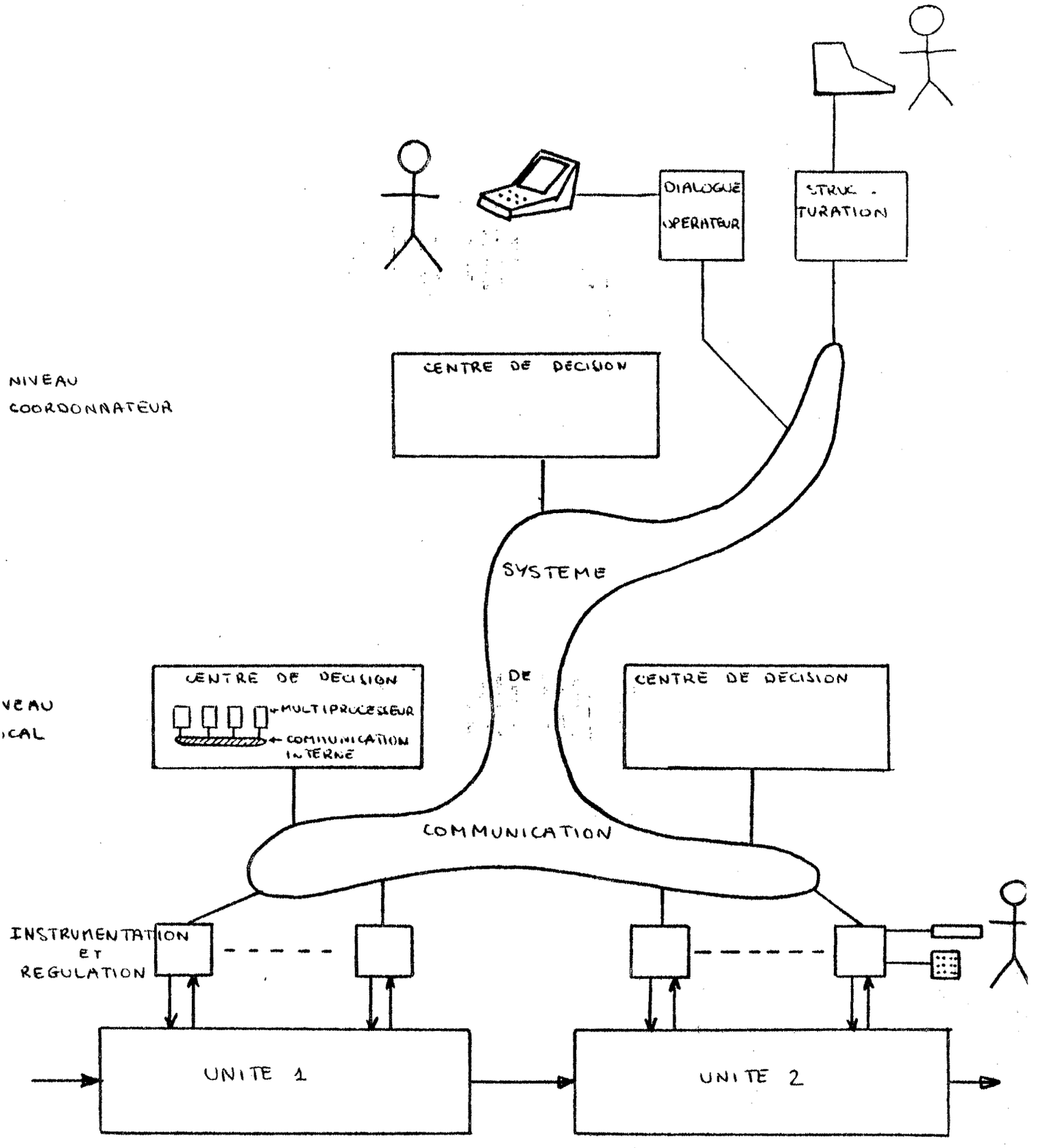


II.2.3 IMPLANTATION MATERIELLE

L'implantation matérielle se fera sur l'image de la structure de commande définie plus haut. Les centres locaux et le centre coordonnateur implantés sur microcalculateurs assurent la commande en temps réel, alors que le dialogue opérateur, la structuration du système général et le calcul des différentes matrices nécessaires sont assurés par un minicalculateur.

Les différents centres sont reliés par un système de communication qui assure la synchronisation de l'ensemble. De ce fait chaque centre de décision devra remplir deux fonctions distinctes : une fonction de communication et une fonction algorithmique de calcul.

- Organisation du réseau IMPACT -



II.3 DEVELOPPEMENT DU LOGICIEL POUR UN CENTRE DE DECISION TYPE

II.3.1 UNE ORGANISATION MODULAIRE

On a adopté une conception modulaire des centres de décision enfin d'en assurer des implantations diverses aisées dans la structure de commande et permettant une évolution indépendante de chaque module.

On définit quatre modules :

1- Un moniteur de communication avec l'environnement, permettant en particulier la définition des fonctions algorithmiques à remplir.

2- L'ossature de l'algorithme standard.

3- Les opérateurs matriciels.

4- Le dipositif arithmétique.

Indépendamment de la forme particulière qu'ils peuvent prendre les modules communiquent entre eux conformément aux protocoles établis. Ces méthodes ont été illustrées en partie I au cours de la description des opérateurs matriciels et des dispositifs arithmétiques. L'algorithme standard est présenté en II.2 et il nous reste à décrire la méthode employée pour sa structuration par le module de communication et d'une manière générale la description du protocole régissant les interactions entre ces deux modules.

II.3.2 STRUCTURATION DE L'ALGORITHME STANDARD

Cette structuration est obtenue par des aiguillages dans l'algorithme. Ces aiguillages sont fonction d'un mot de flags (MODFLAG) positionnés par le module de communication. L'algorithme standard permet ainsi la commande pour les organisations suivantes du système de conduite (dans l'état actuel des programmes; il est évident qu' en situant différemment les aiguillages ont obtient d'autres solutions) :

1- deux centres de décisions locaux pouvant communiquer entre eux mais non controlés par un coordonnateur
   1-1 centre 1 (MODFLAG = 0000xxxx)
       cas aussi d'une unité isolée
   1-2 centre 2 (MODFLAG = 0100xxxx)

2- Commande à 2 niveaux avec 1 coordonnateur et 2 centres locaux
   2-1 centre 1 (MODFLAG = 1000xxxx)
   2-2 centre 2 (MODFLAG = 1100xxxx)

2-3 centre coordonateur (MODFLAG = 1111**0000**)

Un détermine de même les méthodes suivantes d'application du correcteur :

3-1 pas de correcteur (MODFLAG = xxxx0000)
3-2 correcteur (MODFLAG = xxxx0001)
3-3 correcteur avec erreur statique nulle (MODFLAG = xxxx0011)

La figure ci-dessous schématise cette structuration par aiguillages.

L'algorithme qui a besoin de données, établies en temps réel par ailleurs, les acquiert et transmet ses résultats par débranchement vers des routines du module de communications :

- ACQSORT , acquisition des sorties mesurées sur le procédé
- ACQENTR , acquisition des entrées mesurées sur le procédé
- ACQCENTl , acquisition d'informations provenant du centre précédent de même niveau
- ACQCOORD , acquisition d'informations provenant du centre coordonnateur
- COMCMD , envoi de la commande calculée vers le procédé.

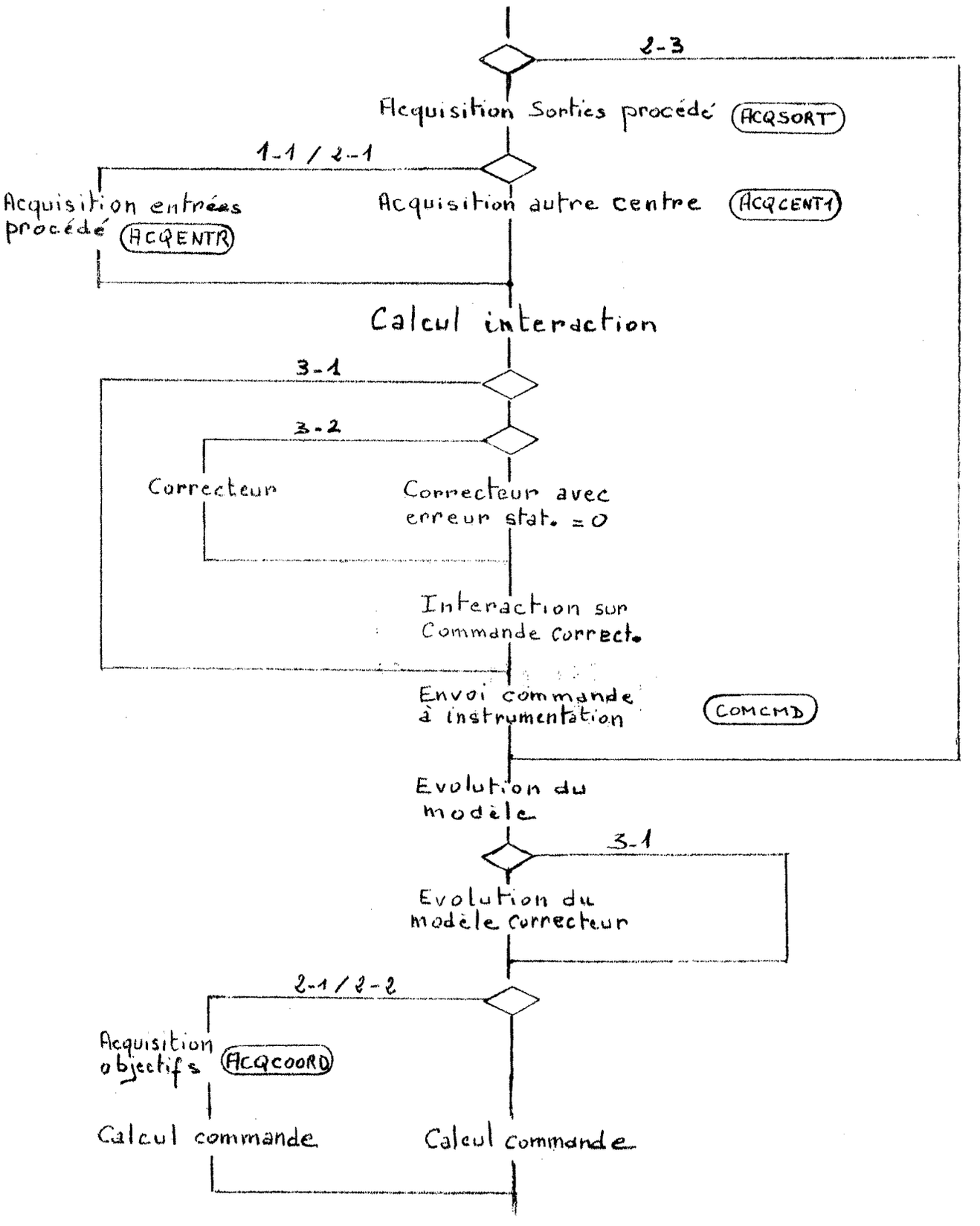
On peut donc établir les dispositifs de dialogue entre l'algorithme et le moniteur de communication :

- MODFLAG
- Les routines citées ci-dessus
- Une mémoire RAM commune contenant les données

- Structuration de l'algorithme standard -

(com.) : dialogue par débranchement vers le module de communication

1-1 1-2
2-1 2-2 2-3
3-1 3-2 3-3 : aiguillages décrits plus haut



II.3.3 LE MONITEUR DE COMMUNICATION

Ses rôles principaux sont de participer à la structuration générale du système de conduite par dialogue avec le réseau (type d'algorithme, réception des matrices....) et d'assurer la synchronisation du centre (échanges entre différents niveaux, tops d'échantillonnage...).

La partie dialogue avec l'environnement est divisée en tâches dont la liste minimale est donnée ci-dessous :

- Centre concerné (local 1 ou 2, coordonnateur...)
- Dimensions du modèle (nombre d'entrées, de commandes, de sorties, d'états; toutes les matrices seront dimensionnées par ces valeurs)
- Réception des matrices du modèle (A,B,C)
- Réception des matrices de bouclage et d'anticipation (L, M1, M2, Mw)
- Mode d'application du correcteur
- Réception des objectifs et validation à un instant donné
- Acquisition uo et yo du procédé, c'est à dire le point de fonctionnement du procédé à l'instant initial, tous les calculs étant faits par différence avec ce point.
- Remise à zéro des états du modèle
- Remise à zéro des états du modèle correcteur
- Fin du dialogue, c'est à dire branchement vers une routine ATTN d'attente d'un top d'échantillonnage ou de toute autre demande externe.

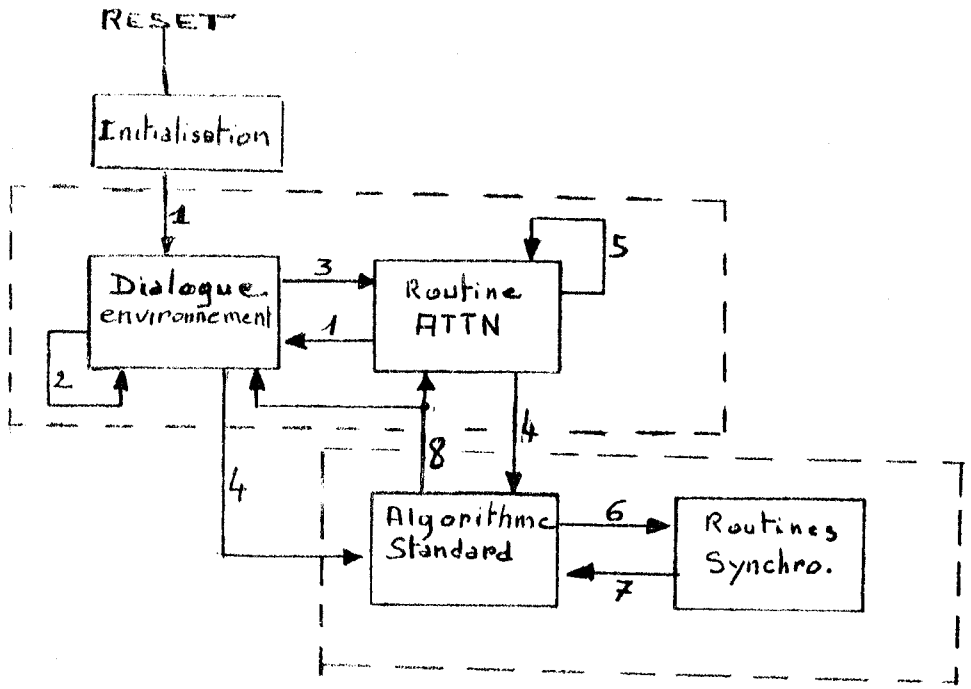
La partie synchronisation du centre, en plus de son initialisation matérielle (RESET) qui démarre le dialogue, se compose des routines de communication avec les autres centres du système de conduite :

- ACQENTR
- ACQSORT
- ACQCOORD
- ACQCENT1
- COMCMD
- COMRES , envoi d'informations sur le réseau

et de la routine ATTN , boucle d'attente des tops d'échantillonnage et des autres interventions externes (interruptions matérielles, consultation des dispositifs d'entrées/sorties, tests d'indicateurs en mémoire etc.)

II.3.4 FONCTIONNEMENT GENERAL D'UN CENTRE DE DECISION

Le fonctionnement d'un centre peut être représenté par l'automate schématisé ci-dessous :



Evènements

1 : Initialisation du dialogue
2 : Activation d'une tâche dialogue
3 : Fin du dialogue
4 : Top d'échantillonnage
5 : Pas d'interruption
6 : Demande ressource de synchro.
7 : Fin traitement synchro.
8 : Fin algorithme

De ce point de vue on remarque que le centre est constitué de deux programmes :

- Un programme de priorité basse interruptible par l'évènement "top d'échantillonnage", constitué de la partie dialogue avec l'environnement et de la boucle d'attente du moniteur

- Un programme de priorité haute non interruptible, constitué de l'algorithme standard de calcul de la commande, de la partie communication inter-niveaux du moniteur, des opérateurs matriciels et de l'arithmétique.

Il apparaît de plus que deux conditions sont nécessaires au bon fonctionnement du centre :

1- Le délai nécessaire au déroulement de l'algorithme doit être obligatoirement inférieur à la durée séparant deux tops d'échantillonnage (Attention aux temps de transfert d'informations demandées par les routines de synchronisation)

2- Les parties du dialogue pouvant etre activées en temps réel et modifiant des données utilisés par l'algorithme de commande doivent masquer l'interruption "top d'échantillonnage", quitte à faire sauter un pas de calcul au centre (par exemple la validation de nouveaux objectifs).

II.4 TESTS ET EXPLOITATION

II.4.1 UNE OPTIQUE PARTICULIERE DE REALISATION

Pour le premier centre réalisé avec le microprocesseur INTEL 8080 on définit plus précisément deux fonctions du moniteur :

- Les tops d'échantillonnage sont donnés par interruption externe sur le microprocesseur (signal sur la broche INTE), se traduisant par un débranchement à l'adresse héxadécimale 0038 obtenu par envoi de l'instruction RST 7 sur le bus des données.

- La routine ATTN boucle sur le test de l'état de l'interface série avec la console opérateur (PCI 8251). Si un caractère est présent en entrée on se débranche vers la partie dialogue opérateur.

Ces dispositions permettent de réaliser un centre de commande isolé dialoguant avec un opérateur par l'intermédiaire de la console et synchronisé avec un dispositif de mesure par un horloge temps réel et une communication série.

II.4.2 TESTS SUR L'OUTIL DE DEVELOPPEMENT

Les points remarquables des tests sur l'outil de développement portent sur :

- La mise au point d'un moniteur de tests comportant un surcroit de tâches de dialogue avec l'opérateur, permettant ainsi l'affichage des différents vecteurs et matrices afin de surveiller les évolutions des variables à chaque pas. Cette partie du moniteur activable à partir de la boucle ATTN s'est révélée très utile pour la vérification des calculs effectués par l'algorithme.

- La simulation des communications inter - niveaux par visualisation ou saisie de matrices à la console (routines du type ACQxxx et COMxxx constituées d'appels des opérateurs FMCONI et FMCONO).

- L'utilisation d'un modèle élaborant les mesures.
Ce modèle est identique au modèle de la commande avec les variables du procédé :

$$xp = Axp + B(up + uf)$$
$$yp = Cxp$$

Le vecteur uf est un vecteur de perturbation constante sur la commande élaborée par le centre de décision.

L'affichage des vecteurs u, up, y, yp et des objectifs

donnés (zi et zo) permet de vérifier graphiquement l'évolution du procédé vers le point de fonctionnement désiré et ceci dans tous les cas de configuration de l'algorithme (à l'aide de MODFLAG).

Sur l'émulateur 8080 de l'outil de développement Tektronix 8002, l'exécution de l'algorithme le plus complexe (MODFLAG = 1100 0011) prend environ 2 secondes (temps d'entrées/sorties non compris).

II.4.3 LES IMPLANTATIONS FUTURES

Les implantations prévues dans la structure du réseau IMPACT se feront sur des cartes microcalculateurs construites autour de l'unité centrale constituée du microprocesseur 8080 et de ses circuits d'horloge et de contrôle.

Ces cartes sont dotées de plus d'un interface de communication en série (PCI 8251) et de deux interfaces de communication en parallèle programmables (PPI 8255) offrant six ports d'entrée/sortie. La mémoire RAM est implantée par modules de 256 octets jusqu'à 2 K-octets, et la mémoire ROM par modules de 1 K-octets jusqu'à 4 K-octets.

Ceci présente une configuration minimale extensible.

Sur cette configuration nous essayons de définir une implantation standard des programmes afin d'atteindre un but recherché tout au long de ce travail , c'est à dire la constitution d'un produit standard adaptable par un minimum d'interventions :

- ROM 1 (0000 à 03FF) : partie du moniteur (ou totalité selon sa taille) assurant l'initialisation du système, la reconnaissance des tâches de dialogue à activer, les routines de synchronisation (ACQxxx, COMxxx, ATTN)

- ROM 2 (0400 à 07FF) : Algorithme standard

- ROM 3 (0800 à 0BFF) : Opérateurs matriciels

- ROM 4 (0C00 à 0FFF) : arithmétique FP.ATMI ou interface AM9511

Sur la carte extension prennent place :

- Les tâches de dialogue non implantées en ROM 1.

- Les routines particulières à l'implantation, telles que les routines de conversion pour un dialogue à la console opérateur.

En ce qui concerne les données, il faut définir des emplacements en RAM aussi universels que possible. Nous

proposons :

- MODFLAG en 3C00, adresse du début de la mémoire RAM

- Initialisation de la Pile à 43FF, adresse du dernier octet de la RAM sur la carte maîtresse

- Dimensions et matrices sur la RAM en extension afin de disposer d'une place suffisante pour une définition générale des données acceptable dans toute la structure de commande.

Ainsi on dispose d'une carte de base pour les centres de décision. Alors pour changer d'arithmétique, par exemple, il suffit de changer la ROM 4. Suivant le système de communication utilisé on dispose de ROM 1 différentes. Ceci sans réédition de liens entre les différents modules , ni reprogrammation des PROM non modifiées, pour peu que les adresses concernées par les protocoles régissant les interactions entre modules restent inchangées.

Ceci donne la possibilité de conserver une carte de secours en cas de panne sur un des centres du système de commande.

- CONCLUSION -

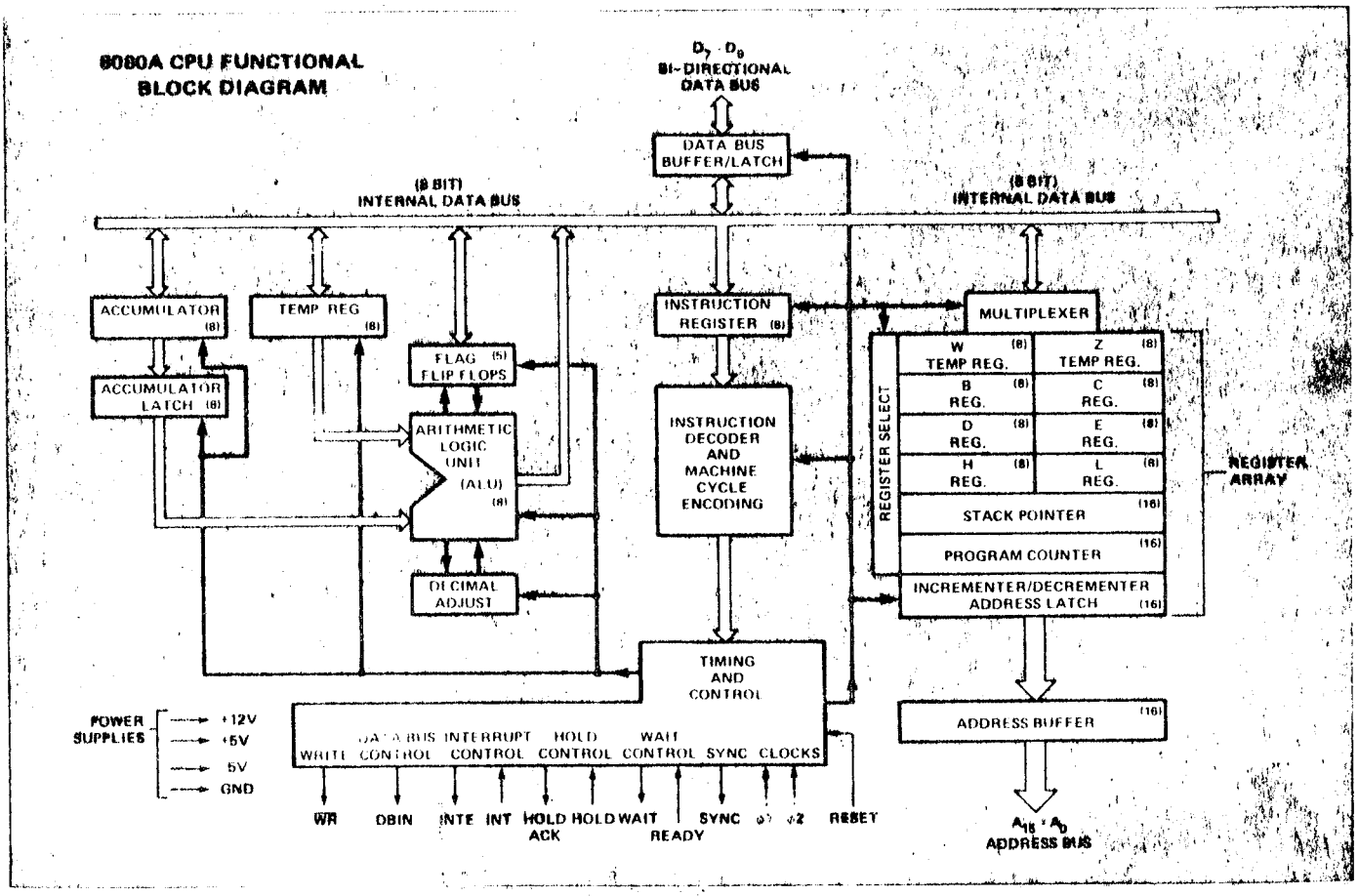
La stratégie de mise au point progressive et ascendante choisie par le L.A.G. pour la réalisation d'un système de commande décentralisée coordonnée illustre bien la nécessité de la conception modulaire du logiciel.

L'aspect à la fois standard et évolutif du produit obtenu doit permettre son implantation diverse sans trop d'efforts supplémentaires, en particulier son intégration dans les différents supports matériels prévus pour le réseau de communication.

La mise au point de ce logiciel nous a apporté d'autre part la confirmation du bien-fondé de la constitution préalable d'outils généraux d'aide au développement. Nous espérons que ces outils , comme c'est déjà le cas pour les routines de conversion , trouverons des utilisations futures.

* * *

- ANNEXE A -


8080A CPU FUNCTIONAL BLOCK DIAGRAM
D7 - D0 BI-DIRECTIONAL DATA BUS
DATA BUS BUFFER/LATCH
(8 BIT) INTERNAL DATA BUS
(8 BIT) INTERNAL DATA BUS
ACCUMULATOR (8)
TEMP REG (8)
FLAG (5) FLIP FLOPS
ACCUMULATOR LATCH (8)
ARITHMETIC LOGIC UNIT (ALU) (8)
DECIMAL ADJUST
INSTRUCTION REGISTER (8)
INSTRUCTION DECODER AND MACHINE CYCLE ENCODING
MULTIPLEXER
REGISTER SELECT
W (8) TEMP REG.
Z (8) TEMP REG.
B (8) REG.
C (8) REG.
D (8) REG.
E (8) REG.
H (8) REG.
L (8) REG.
STACK POINTER (16)
PROGRAM COUNTER (16)
INCREMENTER/DECREMENTER ADDRESS LATCH (16)
REGISTER ARRAY
TIMING AND CONTROL
ADDRESS BUFFER (16)
POWER SUPPLIES
+12V
+5V
-5V
GND
DATA BUS WRITE CONTROL
INTERRUPT CONTROL
HOLD CONTROL
WAIT CONTROL
SYNC
CLOCKS
WR
DBIN
INTE
INT
HOLD ACK
HOLD
WAIT
READY
SYNC
Φ1
Φ2
RESET
A15 - A0 ADDRESS BUS

INTEL 8080

ISTRUCTION SET

ummary of Processor Instructions

Mnemonic	Description	Instruction Code[1] D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Clock[2] Cycles
OV r1,r2	Move register to register	0	1	D	D	D	S	S	S	5
OV M,r	Move register to memory	0	1	1	1	0	S	S	S	7
OV r,M	Move memory to register	0	1	D	D	D	1	1	0	7
LT	Halt	0	1	1	1	0	1	1	0	7
VI r	Move immediate register	0	0	D	D	D	1	1	0	7
VI M	Move immediate memory	0	0	1	1	0	1	1	0	10
NR r	Increment register	0	0	D	D	D	1	0	0	5
CR r	Decrement register	0	0	D	D	D	1	0	1	5
NR M	Increment memory	0	0	1	1	0	1	0	0	10
CR M	Decrement memory	0	0	1	1	0	1	0	1	10
DD r	Add register to A	1	0	0	0	0	S	S	S	4
DC r	Add register to A with carry	1	0	0	0	1	S	S	S	4
UB r	Subtract register from A	1	0	0	1	0	S	S	S	4
BB r	Subtract register from A with borrow	1	0	0	1	1	S	S	S	4
NA r	And register with A	1	0	1	0	0	S	S	S	4
RA r	Exclusive Or register with A	1	0	1	0	1	S	S	S	4
RA r	Or register with A	1	0	1	1	0	S	S	S	4
MP r	Compare register with A	1	0	1	1	1	S	S	S	4
DD M	Add memory to A	1	0	0	0	0	1	1	0	7
DC M	Add memory to A with carry	1	0	0	0	1	1	1	0	7
UB M	Subtract memory from A	1	0	0	1	0	1	1	0	7
BB M	Subtract memory from A with borrow	1	0	0	1	1	1	1	0	7
NA M	And memory with A	1	0	1	0	0	1	1	0	7
RA M	Exclusive Or memory with A	1	0	1	0	1	1	1	0	7
RA M	Or memory with A	1	0	1	1	0	1	1	0	7
MP M	Compare memory with A	1	0	1	1	1	1	1	0	7
DI	Add immediate to A	1	1	0	0	0	1	1	0	7
CI	Add immediate to A with carry	1	1	0	0	1	1	1	0	7
UI	Subtract immediate from A	1	1	0	1	0	1	1	0	7
BI	Subtract immediate from A with borrow	1	1	0	1	1	1	1	0	7
NI	And immediate with A	1	1	1	0	0	1	1	0	7
RI	Exclusive Or immediate with A	1	1	1	0	1	1	1	0	7
RI	Or immediate with A	1	1	1	1	0	1	1	0	7
PI	Compare immediate with A	1	1	1	1	1	1	1	0	7
LC	Rotate A left	0	0	0	0	0	1	1	1	4
RC	Rotate A right	0	0	0	0	1	1	1	1	4
AL	Rotate A left through carry	0	0	0	1	0	1	1	1	4
AR	Rotate A right through carry	0	0	0	1	1	1	1	1	4
MP	Jump unconditional	1	1	0	0	0	0	1	1	10
C	Jump on carry	1	1	0	1	1	0	1	0	10
NC	Jump on no carry	1	1	0	1	0	0	1	0	10
Z	Jump on zero	1	1	0	0	1	0	1	0	10
NZ	Jump on no zero	1	1	0	0	0	0	1	0	10
P	Jump on positive	1	1	1	1	0	0	1	0	10
M	Jump on minus	1	1	1	1	1	0	1	0	10
PE	Jump on parity even	1	1	1	0	1	0	1	0	10
PO	Jump on parity odd	1	1	1	0	0	0	1	0	10
ALL	Call unconditional	1	1	0	0	1	1	0	1	17
C	Call on carry	1	1	0	1	1	1	0	0	11/17
NC	Call on no carry	1	1	0	1	0	1	0	0	11/17
Z	Call on zero	1	1	0	0	1	1	0	0	11/17
NZ	Call on no zero	1	1	0	0	0	1	0	0	11/17
P	Call on positive	1	1	1	1	0	1	0	0	11/17
M	Call on minus	1	1	1	1	1	1	0	0	11/17
PE	Call on parity even	1	1	1	0	1	1	0	0	11/17
PO	Call on parity odd	1	1	1	0	0	1	0	0	11/17
ET	Return	1	1	0	0	1	0	0	1	10
C	Return on carry	1	1	0	1	1	0	0	0	5/11
NC	Return on no carry	1	1	0	1	0	0	0	0	5/11

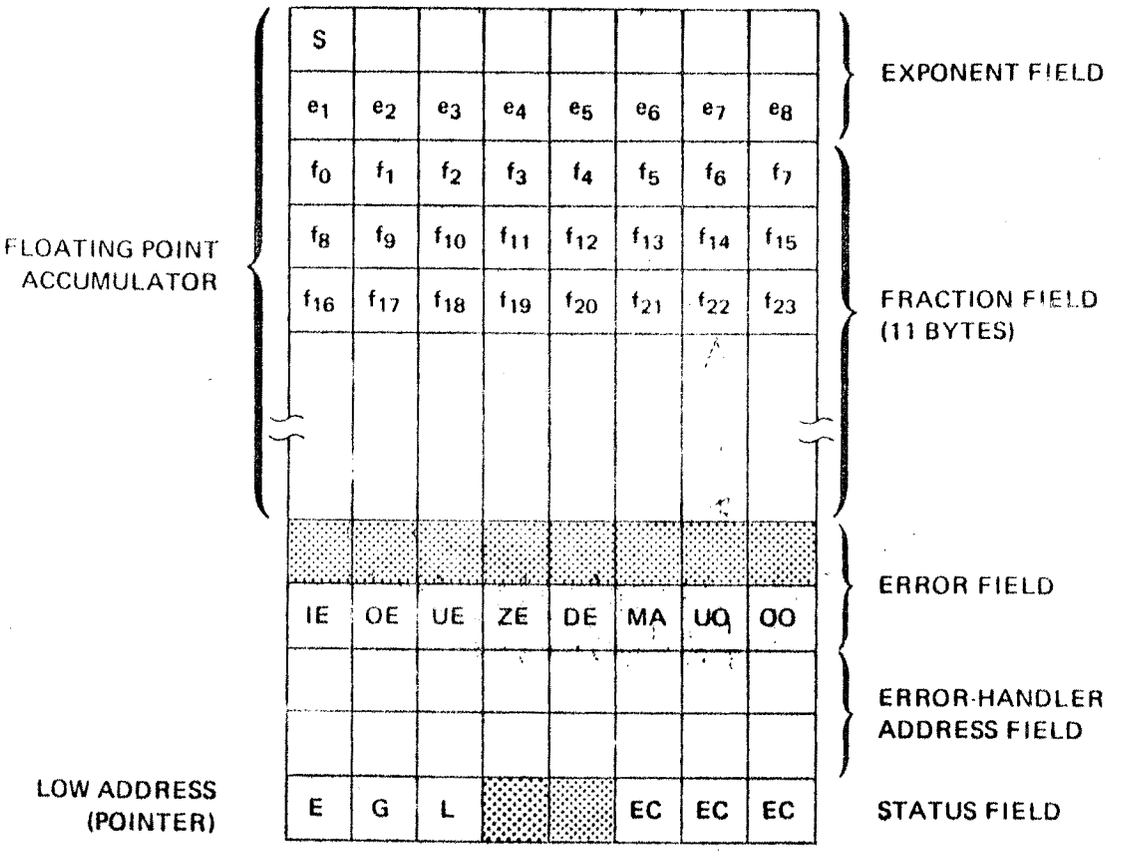
Mnemonic	Description	Instruction Code[1] D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Clock[2] Cycles
RZ	Return on zero	1	1	0	0	1	0	0	0	5/11
RNZ	Return on no zero	1	1	0	0	0	0	0	0	5/11
RP	Return on positive	1	1	1	1	0	0	0	0	5/11
RM	Return on minus	1	1	1	1	1	0	0	0	5/11
RPE	Return on parity even	1	1	1	0	1	0	0	0	5/11
RPO	Return on parity odd	1	1	1	0	0	0	0	0	5/11
RST	Restart	1	1	A	A	A	1	1	1	11
IN	Input	1	1	0	1	1	0	1	1	10
OUT	Output	1	1	0	1	0	0	1	1	10
XI B	Load immediate register Pair B & C	0	0	0	0	0	0	0	1	10
XI D	Load immediate register Pair D & E	0	0	0	1	0	0	0	1	10
XI H	Load immediate register Pair H & L	0	0	1	0	0	0	0	1	10
XI SP	Load immediate stack pointer	0	0	1	1	0	0	0	1	10
PUSH B	Push register Pair B & C on stack	1	1	0	0	0	1	0	1	11
PUSH D	Push register Pair D & E on stack	1	1	0	1	0	[illegible]	[illegible]	[illegible]	11
PUSH H	Push register Pair H & L on stack	1	1	1	0	0	1	0	1	11
PUSH PSW	Push A and Flags on stack	1	1	1	1	0	1	0	1	11
POP B	Pop register pair B & C off stack	1	1	0	0	0	0	0	1	10
POP D	Pop register pair D & E off stack	1	1	0	1	0	0	0	1	10
POP H	Pop register pair H & L off stack	1	1	1	0	0	0	0	1	10
POP PSW	Pop A and Flags off stack	1	1	1	1	0	0	0	1	10
STA	Store A direct	0	0	1	1	0	0	1	0	13
LDA	Load A direct	0	0	1	1	1	0	1	0	13
XCHG	Exchange D & E, H & L Registers	1	1	1	0	1	0	1	1	4
XTHL	Exchange top of stack, H & L	1	1	1	0	0	0	1	1	18
SPHL	H & L to stack pointer	1	1	1	1	1	0	0	1	5
PCHL	H & L to program counter	1	1	1	0	1	0	0	1	5
DAD B	Add B & C to H & L	0	0	0	0	1	0	0	1	10
DAD D	Add D & E to H & L	0	0	0	1	1	0	0	1	10
DAD H	Add H & L to H & L	0	0	1	0	1	0	0	1	10
DAD SP	Add stack pointer to H & L	0	0	1	1	1	0	0	1	10
STAX B	Store A indirect	0	0	0	0	0	0	1	0	7
STAX D	Store A indirect	0	0	0	1	0	0	1	0	7
LDAX B	Load A indirect	0	0	0	0	1	0	1	0	7
LDAX D	Load A indirect	0	0	0	1	1	0	1	0	7
INX B	Increment B & C registers	0	0	0	0	0	0	1	1	5
INX D	Increment D & E registers	0	0	0	1	0	0	1	1	5
INX H	Increment H & L registers	0	0	1	0	0	0	1	1	5
INX SP	Increment stack pointer	0	0	1	1	0	0	1	1	5
DCX B	Decrement B & C	0	0	0	0	1	0	1	1	5
DCX D	Decrement D & E	0	0	0	1	1	0	1	1	5
DCX H	Decrement H & L	0	0	1	0	1	0	1	1	5
DCX SP	Decrement stack pointer	0	0	1	1	1	0	1	1	5
CMA	Complement A	0	0	1	0	1	1	1	1	4
STC	Set carry	0	0	1	1	0	1	1	1	4
CMC	Complement carry	0	0	1	1	1	1	1	1	4
DAA	Decimal adjust A	0	0	1	0	0	1	1	1	4
SHLD	Store H & L direct	0	0	1	0	0	0	1	0	16
LHLD	Load H & L direct	0	0	1	0	1	0	1	0	16
EI	Enable Interrupts	1	1	1	1	1	0	1	1	4
DI	Disable interrupt	1	1	1	1	0	0	1	1	4
NOP	No operation	0	0	0	0	0	0	0	0	4

- ANNEXE B -

Tableau des fonctions FPAL

FPAL Subroutine	B,C Addresses:	D,E Addresses:	Result Stored At:	Function
FABS	FPR	---	FAC	|FAC| ← FAC
FADD	FPR	MEM	FAC	FAC ← FAC + MEM
FCLR	FPR	---	FAC	FAC ← 0
FCMPR	FPR	MEM	REG A	FAC $\gtrless$ MEM
FDIV	FPR	MEM	FAC	FAC ← FAC/MEM
FERROR	FPR	---	REGS H,L	REGS H,L ← ERROR
FIXSD	FPR	MEM	MEM	$MEM_{int} \leftarrow FPR_{fp}$
FLOAD	FPR	MEM	FAC	FAC ← MEM
FLTDS	FPR	MEM	FAC	$FAC_{fp} \leftarrow MEM_{int}$
FMUL	FPR	MEM	FAC	FAC ← FAC · MEM
FNEG	FPR	---	FAC	0 ← 0 otherwise, change sign of FAC
FRESET	B(0)=Error Handler Bit C=Error Field Initialization	User Error Handler	FPR	ERROR ← B,C ERR HAND ADDR ← D,E
FSET	B(0)=Error Handler Bit C=Error Field Initialization	User Error Handler	FPR	FAC ← 0 ERROR ← B,C ERR HAND ADDR ← D,E STATUS ← 0
FSTAT	FPR	---	REG A	REG A ← STATUS
FSTOR	FPR	MEM	MEM	MEM ← FAC
FSUB	FPR	MEM	FAC	FAC ← FAC - MEM
FZTST	FPR	---	REG A	FAC $\lessgtr$ 0

Espace de travail de l'arithmétique



- Exemples d'appels des fonctions arithmétiques -

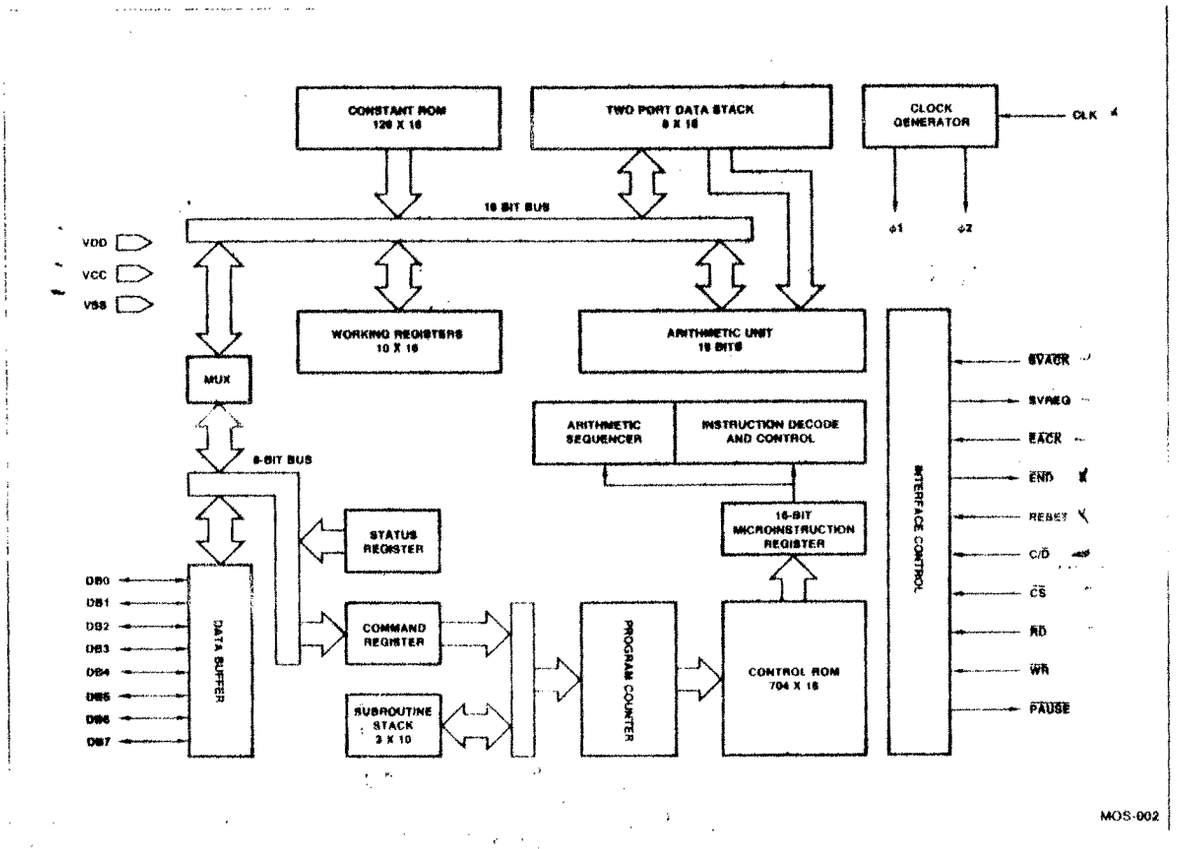
La séquence d'instructions suivantes calcule la formule :

$$IP = (A1*B1 + A2*B2 + A3*B3)/C1$$

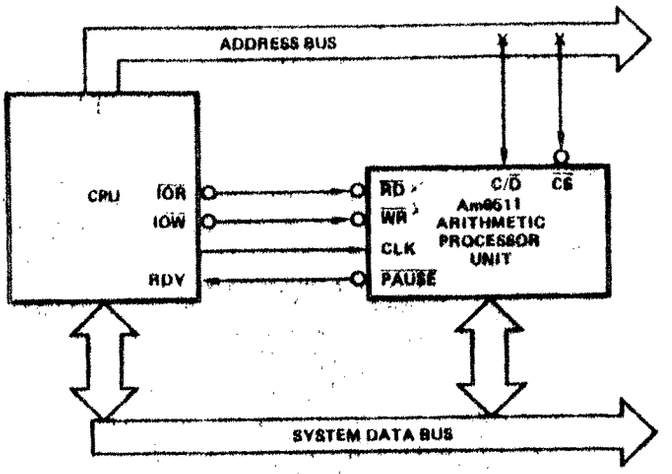
```
LXI     B,FPR       ;B,C POINTS AT THE FPR
PUSH    B
LXI     B,0         ;DEFAULT ERROR HANDLER TO BE USED
CALL    FSET        ;FPR IS INITIALIZED
LXI     B,FPR       ;POINTERS TO FPR AND A1 ARE LOADED
LXI     D,A1
CALL    FLOAD       ;A1 IS LOADED INTO THE FAC
LXI     D,B1        ;POINTER TO B1 IS LOADED
CALL    FMUL        ;A1*B1 IS FORMED IN THE FAC
LXI     D,IP        ;POINTER TO IP IS LOADED
CALL    FSTOR       ;A1*B1 STORED IN LOCATION ADDRESSED
                    ;BY IP
LXI     D,A2
CALL    FLOAD       ;A2 IS LOADED INTO THE FAC
LXI     D,B2
CALL    FMUL        ;A2*B2 IS FORMED IN THE FAC
LXI     D,IP
CALL    FADD        ;A1*B1 + A2*B2 IS FORMED IN THE FAC
CALL    FSTOR       ;A1*B1 + A2*B2 IS STORED IN IP
LXI     D,A3
CALL    FLOAD       ;A3 IS LOADED INTO THE FAC
LXI     D,B3
CALL    FMUL        ;A3*B3 IS FORMED IN THE FAC
LXI     D,IP
CALL    FADD        ;A1*B1 + A2*B2 + A3*B3 IS FORMED
                    ;IN THE FAC
LXI     D,C1
CALL    FDIV        ;(A1*B1 + A2*B2 + A3*B3)/C1 IS
                    ;FORMED IN THE FAC
LXI     D,IP
CALL    FSTOR       ;(A1*B1 + A2*B2 + A3*B3)/C1 IS
                    ;STORED IN IP
```

- ANNEXE C -

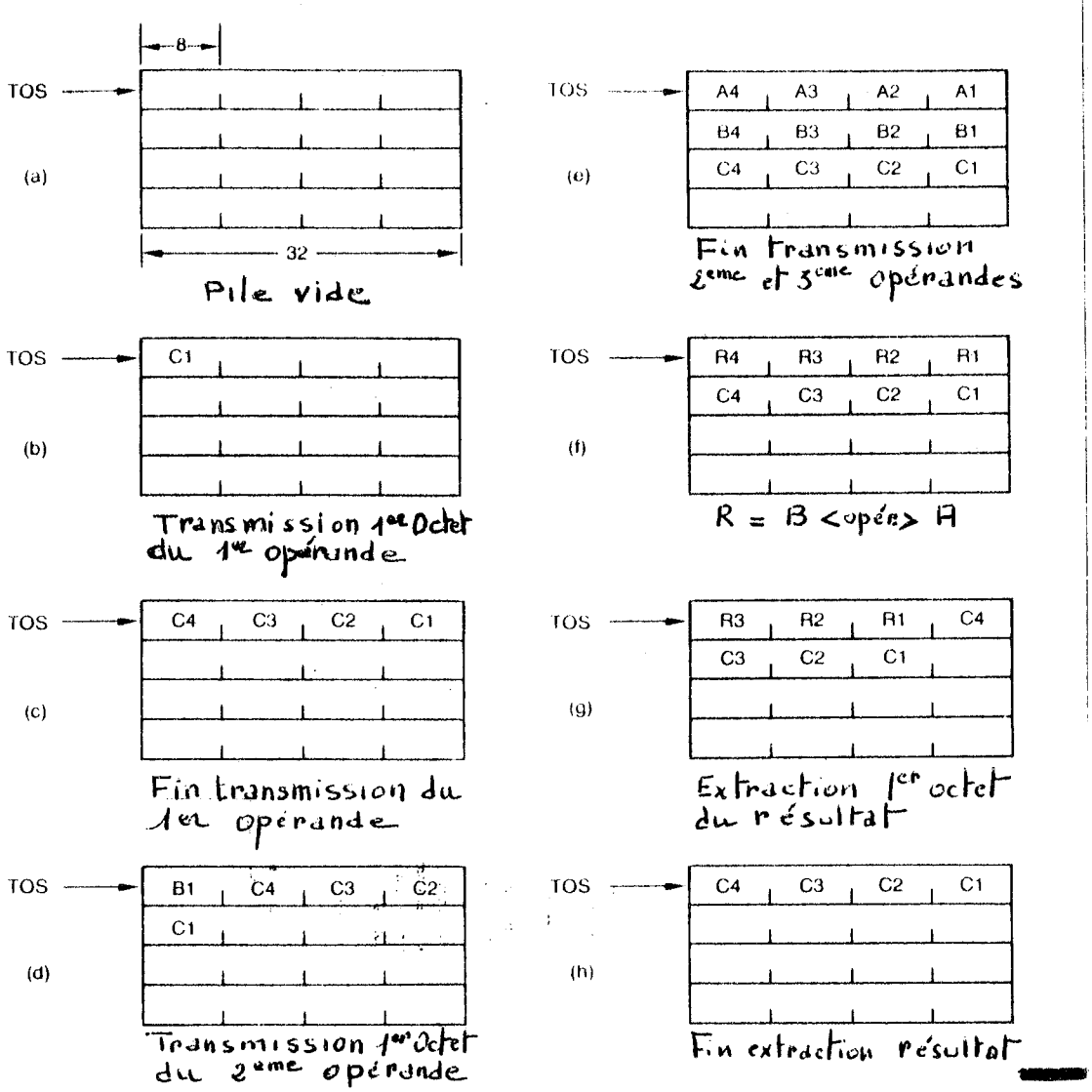
Architecture interne de l'AM9511



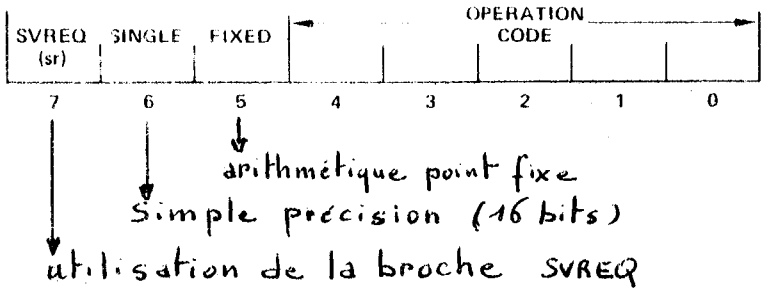
Exemple d'un montage simple de l'AM9511



Mode d'utilisation de la pile interne de l'AM9511



Format du mot de commande

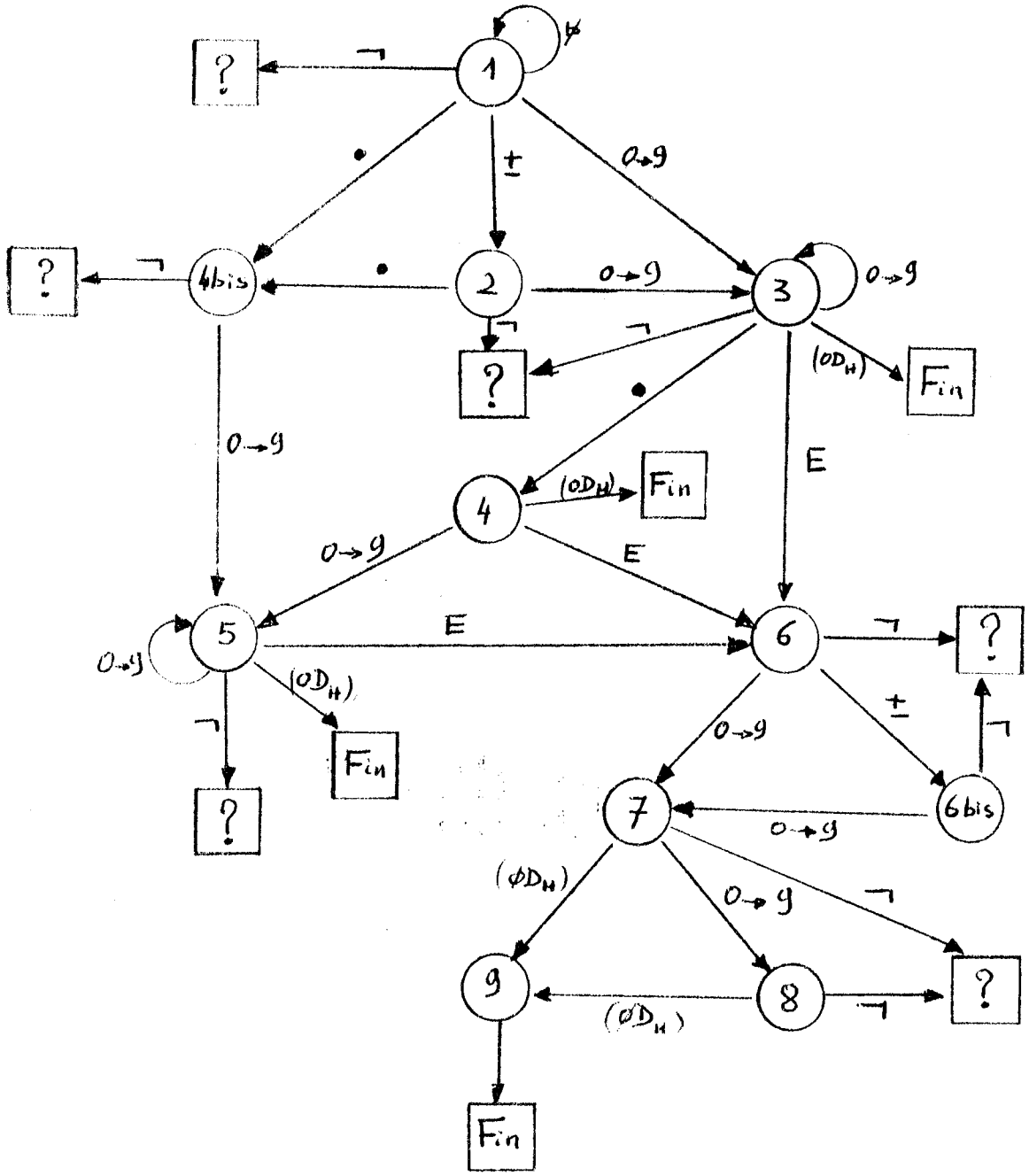


- Exemples de routines d'interface -

```
Tektronix 8080/8085 ASM V3.0                                    Page    1

00001                           NAME    AM9511
00002                   ;
00003                   ;*****************************************
00004                   ;    INTERFACE ENTRE LE CIRCUIT AM9511    *
00005                   ;      ET LES OPERATEURS MATRICIELS       *
00006                   ;*****************************************
00007                   ;
00008                           GLOBAL  AMREAD,AMWRITE,AMCMD,AMSTAT
00009                   ;
00010                   ;

00015                           SECTION FLOAD
00016 0000 F5                   PUSH    PSW
00017 0001 C5                   PUSH    B
00018 0002 D5                   PUSH    D
00019 0003 0604                 MVI     B,4
00020      0005     >  FLOAD1   EQU     $
00021 0005 1A                   LDAX    D
00022 0006 320000   >           STA     AMWRITE
00023 0009 1B                   DCX     D
00024 000A 05                   DCR     B
00025 000B C20500   >           JNZ     FLOAD1
00026 000E D1                   POP     D
00027 000F C1                   POP     B
00028 0010 F1                   POP     PSW
00029 0011 C9                   RET
00030                   ;
00031                   ;
00032                           SECTION FSTOR
00033 0000 F5                   PUSH    PSW
00034 0001 C5                   PUSH    B
00035 0002 1B                   DCX     D
00036 0003 1B                   DCX     D
00037 0004 1B                   DCX     D
00038 0005 1B                   DCX     D
00039 0006 0604                 MVI     B,4
00040      0008     >  FSTOR1   EQU     $
00041 0008 13                   INX     D
00042 0009 3A0000   >           LDA     AMREAD
00043 000C 12                   STAX    D
00044 000D 05                   DCR     B
00045 000E C20800   >           JNZ     FSTOR1
00046 0011 C1                   POP     B
00047 0012 F1                   POP     PSW
00048 0013 C9                   RET

00077                   ;
00078                   ;
00079                           SECTION FADD
00080 0000 CD0000   >           CALL    FLOAD
00081 0003 F5                   PUSH    PSW
00082 0004 3E10                 MVI     A,10H
00083 0006 320000   >           STA     AMCMD
00084 0009 F1                   POP     PSW
00085 000A C9                   RET
```

- ANNEXE D -

Graphe d'analyse d'un nombre décimal flottant sous forme d'une chaine de caracteres ASCII



Actions sémantiques de l'analyseur syntaxique

2 : Stockage signe

3 : Rangement chiffre sous forme DCB dans la partie entière après rotation 1/2 octet à gauche de la zone

Si plus de 7 chiffres : - rangement dans partie fractionnaire
- +1 sur exposant

5 : Rangement chiffre sous forme DCB dans la partie fractionnaire.
Si plus de 24 chiffre, arrêt du stockage

6, 6bis : Stockage Signe exposant

7 : Stockage 1er chiffre exposant en DCB

8 : Stockage 2ème chiffre exposant

9 : Test validité exposant
Transformation en binaire complément à 2

Fin : Fin de l'analyse avec bit CY = 0

? : Fin de l'analyse avec bit CY = 1

Exemple d'utilisation des opérateurs de dialogue

```
ATELIER DE MICROINFORMATIQUE DE GRENOBLE
----------------------------------------
*BIBLIOTHEQUE DE PROCEDURES MATRICIELLES*
*** MICROPROCESSEURS INTEL 8080-8085 ***

              * ADDITION *
DIMENSIONS : XXX XXX ?
002 004

MATRICE 1
DIM 002 * 004
COL 001
123
3
COL 002
3E-8
123456789Ø123456789
COL 003
-3E+8
1
COL 004
Ø
Ø
FIN

MATRICE 2
DIM 002 * 004
COL 001
Ø
1
COL 002
Ø
1
COL 003
123
6
COL 004
123456
Ø
FIN

MATRICE RESULTAT
DIM 002 * 004
COL 001
+1.230000 E+02
+4.000000
COL 002
+3.000000 E-08
+1.234568 E+18
COL 003
-2.999999 E+08
+7.000000
COL 004
+1.234560 E+05
Ø.Ø
FIN
```

- ANNEXE E -

Exemples d'emploi de macro-instructions

Dimensionnement, Réservation mémoire, Appel opérateur dans le cas de matrices statiques

```
00050                                  MXDIM   MAT1,3

00052                                  MXDIM   MAT2,1,2

00054                                  MXDIM   MATR,3,2


00056                                  MXRES.. MAT1
      0000 000C      + MAT1            BLOCK   MAT1.1*MAT1.2*4

00058                                  MXRES.. MAT2
      000C 0008      + MAT2            BLOCK   MAT2.1*MAT2.2*4

00060                                  MXRES.. MATR
      0014 0018      + MATR            BLOCK   MATR.1*MATR.2*4


00062                                  FMCALL  ADPR,MAT1,MAT2,MATR
      002C 210000   >+                LXI     H,MAT1
      002F E5                         PUSH    H
      0030 210C00   >+                LXI     H,MAT2
      0033 E5                         PUSH    H
      0034 211400   >+                LXI     H,MATR
      0037 E5                         PUSH    H
      0038 2603      +                MVI     H,MAT1.1
      003A 2E01      +                MVI     L,MAT1.2
      003C E5                         PUSH    H
      003D 2602      +                MVI     H,MAT2.2
      003F E5                         PUSH    H
      0040 21F6FF                     LXI     H,-10
      0043 CD0000   >+                CALL    FMADPR
      0046 39                         DAD     SP
      0047 F9                         SPHL
```

Réservation mémoire pour matrices dimensionnées dynamiquement

```
00051                                  MXRES   MAT1,4,4
      0000 04        + MAT1$           BYTE    4
      0001 04        +                 BYTE    4
      0002 0010      + MAT1            BLOCK   4*4*4

00053                                  MXRES   MAT2,4,3
      0042 03        + MAT2$           BYTE    3
      0043 04        +                 BYTE    4
      0044 0030      + MAT2            BLOCK   4*3*4

00055                                  MXRES   MATR,4,3
      0074 03        + MATR$           BYTE    3
      0075 04        +                 BYTE    4
      0076 0030      + MATR            BLOCK   4*3*4
```

TSVP

Appels opérateurs pour les matrices déclarées page précédente

type 1

```
00057                                  FMCALL  NEG,MAT1
      00A6 210200  >+                  LXI     H,MAT1
      00A9 E5                          PUSH    H
      00AA 2A0000  >+                  LHLD    MAT1$
      00AD E5                          PUSH    H
      00AE 21FCFF   +                  LXI     H,-00002*2
      00B1 CD0000  >+                  CALL    FMNEG
      00B4 39                          DAD     SP
      00B5 F9                          SPHL
```

type 2

```
00059                                  FMCALL  TRP,MAT2,MATR
      00B6 214400  >+                  LXI     H,MAT2
      00B9 E5                          PUSH    H
      00BA 217600  >+                  LXI     H,MATR
      00BD E5                          PUSH    H
      00BE 2A4200  >+                  LHLD    MAT2$
      00C1 E5                          PUSH    H
      00C2 21FAFF   +                  LXI     H,-00003*2
      00C5 CD0000  >+                  CALL    FMTRP
      00C8 39                          DAD     SP
      00C9 F9                          SPHL
```

type 3

```
00061                                  FMCALL  ADD,MAT2,MAT2,MATR
      00CA 214400  >+                  LXI     H,MAT2
      00CD E5                          PUSH    H
      00CE 214400  >+                  LXI     H,MAT2
      00D1 E5                          PUSH    H
      00D2 217600  >+                  LXI     H,MATR
      00D5 E5                          PUSH    H
      00D6 2A4200  >+                  LHLD    MAT2$
      00D9 E5                          PUSH    H
      00DA 21F8FF   +                  LXI     H,-00004*2
      00DD CD0000  >+                  CALL    FMADD
      00E0 39                          DAD     SP
Tektronix  8080/8085 ASM V3.0
      00E1 F9                          SPHL
```

type M

```
00063                                  FMCALL  PRD,MAT1,MAT2,MATR
      00E2 210200  >+                  LXI     H,MAT1
      00E5 E5                          PUSH    H
      00E6 214400  >+                  LXI     H,MAT2
      00E9 E5                          PUSH    H
      00EA 217600  >+                  LXI     H,MATR
      00ED E5                          PUSH    H
      00EE 2A0000  >+                  LHLD    MAT1$
      00F1 E5                          PUSH    H
      00F2 3A4200  >+                  LDA     MAT2$
      00F5 67                          MOV     H,A
      00F6 E5                          PUSH    H
      00F7 21F6FF                      LXI     H,-10
      00FA CD0000  >+                  CALL    FMPRD
      00FD 39                          DAD     SP
      00FE F9                          SPHL
```

Exemple d'utilisation de l'assembleur croisé spécialisé généré par GAGE -

TEST DES MATRICES

```
LIGNE  PROGRAMME SOURCE

    2          TYPE     CONS
    3          EXTERN   MATL,MATM1,MATM2
    4          EXTERN   VECTJ,VECTUC,VECTJJ,VECTJP
    5          EXTERN   VECTX,VECTZI,VECTZJ


    7  DIM     MATL(4,6)
    8  DIM     MATM1(4,4)
    9  DIM     MATM2(4,2)
   10  DIM     VECTU(4)
   11  DIM     VECTUC(4)
   12  DIM     VECTJJ(4)
   13  DIM     VECTJP(4)
   14  DIM     VECTX(6)
   15  DIM     VECTZI(4)
   16  DIM     VECTZJ(2)


   18          OPM      VECTJ=MATM1.PRD.VECTZI

   19          OPM      VECTJ=MATL.SUPR.VECTX

   20          OPM      VECTJ=MATM2.SUPR.VECTZJ

   21          OP3      VECTJP=VECTU.ADD.VECTUC

   22          OP3      VECTJP=VECTJP.ADD.VECTJJ

   23          OP1      .CONJ.VECTJP

   24          OP2      VECTJ.MOVE.VECTJJ

   25          OPM,KEEP VECTJ=MATM1.PRD.VECTZI

   26          CALL     FMCOND
   27          LXI      H,-10
   28          DAD      SP
   29          SPHL
   30          END
```

- BIBLIOGRAPHIE -

1- Microprocesseurs, Microinformatique

"Les microprocesseurs"
R. ZAKS, P. LE BEUX (SYBEX 1978)

"Du microprocesseur au microordinateur"
H. LILEN (Editions RADIO 1978)

"Logiciel de développement pour microprocesseurs - CAGE : générateur d'assembleurs"
Y. BEKKERS, P. FONTANILLE
Atelier de microinformatique de Grenoble (1978)

2- Automatique

"Commande et régulation par calculateur numérique"
C. FOULARD, S. GENTIL, J.P. SANDAZ
(Editions Eyrolles 1977)

"Sur la commande décentralisée coordonnée. Application à un procédé pilote de distillation"
D. REY, Thèse de Docteur-Ingénieur INPG (novembre 1978)

"Méthodologie de l'automatisation progressive par l'informatique répartie"
Z. BINDER, Rapport d'activité L.A.G. 1977 (no 77-36)

"Coordinated decentralized control of a complex distillation pilot plant"
Z. BINDER, A. JANEX, B. MONNIER, D. REY
(IFAC 1977)

"Problems of instrumentation set in the control of complex systems"
Z. BINDER (VII Congress IMEKO , LONDON 1976)

A MICROPROCESSOR CONTROLLER FOR DECENTRALISED HIERARCHICAL CONTROL

L. Barrero *, Z. Binder*, J.P. Eynard**, J. Rey*

* Laboratoire d'Automatique de Grenoble
** Atelier de Microinformatique
Institut National Polytechnique de Grenoble, BP 46,
38402 - Saint-Martin-d'Hères, FRANCE

Abstract. The practical implementation of control systems for complex processes necessitates a clear demarkation of the responsabilities, the hardware and the software involved. The paper presents an analysis of the structure of control systems and shows how this knowledge could be exploited for constructing a microprocessor network of decision centres. The software design details of such a controller is given. The experience gained in this venture is detailed.

Keywords : Microprocessors; Real Time ; Process Control.

INTRODUCTION

The increasing complexity of control theoretic methods developed recently, render most of them inaccessible to the user. It has therefore become desirable to find out solutions which could be implemented in practice. In the case of hierarchical control, we could be lead to quasi disjoint problems using decomposition of algorithms. Thus the concept of a computer network could be envisaged so as to provide microprocessor based control with the capabilities of large sized machines. The simplification thus introduced, would result in a coherence with consequences both on the hardware and the software.

In this context we have spent a considerable effort to analyse the organisation and role of industrial process control systems. The control of industrial processes is generally based on a multilevel hierarchical organisation. The different levels are characterised by differences in the responsabilities and in the hardware and software involved. The diversity in the means adopted is justified by the control objectives forming in general a coherent assembly. This structuration in the different decision centres forms distinct entities composed of hardware and control algorithm software. The modularity and standardisation of these decision centres facilitates implementation and adds to their reliability. The degree of automation depends on the manner in which the centre is constructed about a microprocessor and the application software implanted.

CONTROL SYSTEM ORGANISATION

The hierarchical organisation of control systems is the result of the historical evolution of process control and management concepts. The appearance of powerful computers has strengthened the tendency towards centralisation of decision making, microprocessors, on the contrary, offer means of decentralisation. These two different tendencies are envisaged in the "Coordinated Decentralised Control" CODECO approach developed at the "Laboratoire d'Automatique de Grenoble" [3]. The organisation of this approach is schematised in fig. 1. The advantage of this organisation lies in the possibility of a progressive implementation and the safety it offers in the case of degraded functioning.

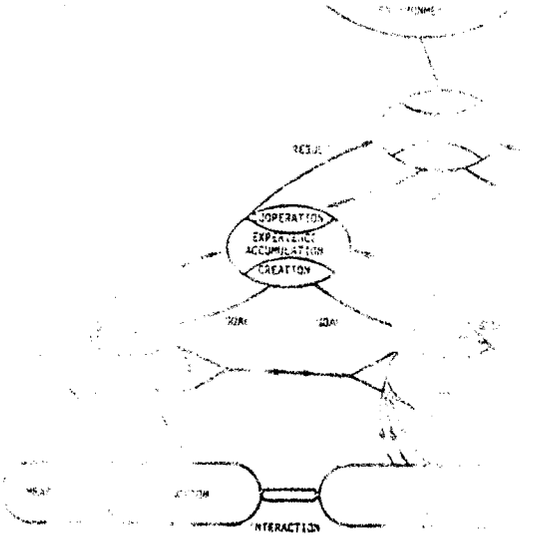


Fig. CODECO System Organisation

Moreover, the structure of the may be correlated with the architecture of the physical system and transformations in case of system modification. The control system is formed by a multilevel network of

centres, linking the process and the environment. Information exchanges take place between the decision centres of the same level (interaction) and centres situated at hierarchically different levels.

Let us suppose that the aim of this network of decision centres is to interpret objectives (even if they are contradictory) of the environment for the physical process with a view to attaining them. In other words, it ensures the satisfaction of the objectives of the environment by the process in the sense of a predefined criterion. In this manner, the functions of a control system may be defined as well as those of its component centres.

A fundamental function of each centre is the continuous conservation and amelioration of operational experience. It could be noted that centres situated at a higher level in the hierarchy conserve the experience (usually expressed by a model) of a larger area although with lesser details. This information serves to define a strategy for satisfying the objectives with respect to the environment in a cooperative sense. The strategy obtained and the evolution of the model subsequently serve to create objectives for centres situated generally at lower levels in the hierarchy.

The basic properties which we have just defined may be adapted according to specific conditions of the criterion and the relative position of the centre in the overall control system. Moreover, certain centres are multiplied in order to ensure proper functioning under different operating conditions (start up, transition, critical operation), or to ensure the change of structure of the physical system and of the control system.

TRACKING CONTROL ALGORITHMS

In order to represent the relations between the decision centers and their models and to express the functioning, we propose a tracking formulation [2]. In this sense, the functioning of a control system may be defined as a successive tracking of hierarchically superior centres by lower level centres. The "distance" between these centre, or their models represented often as mathematical criterion expresses an index of tracking performance. The reduction of the distance may be obtained by modifying control inputs parameters or by changing these models

The relation between models (or the systems they represent) is translated by the relation between their variables, in particular, their external variables (input - output variables). One thus arrives at a situation where variable objectives are tracked by variables of the system or of the model. In Fig. 2 we present an example of the tracking of model variables of a higher level centre by two centres situated at a lower level. This is the essential function of the decision centres which we term cooperation with environment.

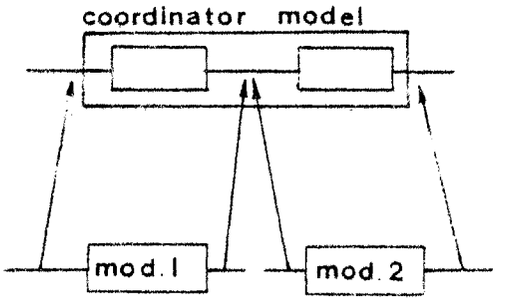


Fig. 2. Model Tracking Strategy

As an example, let us study the algorithmic structure of a decision centre for local control. The model for which the tracking algorithm is used, is linear. The quadratic criterion expresses the input-output tracking error. This permits defining an optimal trajectory for minimising the deviation between the input variables u (and the output variables y) with respect to their corresponding objective variables Z_i and Z_o. The solution of the above problem, [3] is represented by the upper part of figure 3. [6].

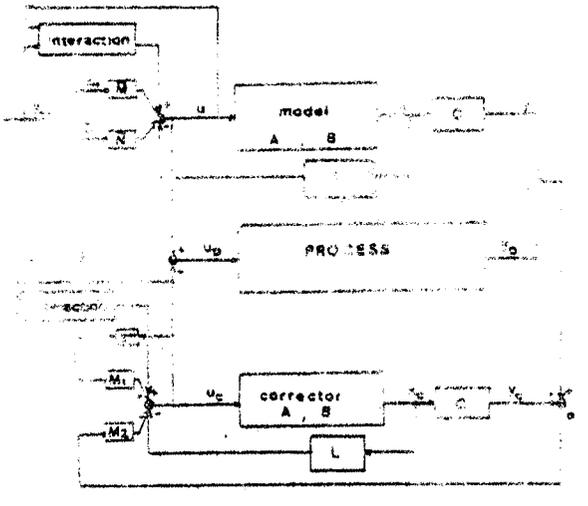


Fig. 3. Structure of the Decision Centre

For a decision centre connected to the process, the predictive control law thus computed is applied to the process. Its value might be modified so as to obey constraints if any or to take into account interaction values if there is any connection with other processes. As a result of perturbations or in case of modelling errors, the evolution of the process could differ from that predicted by the model. The error between the output of the model and those of the process could be reduced using a corrector shown in the lower half of fig. 2. Its structure [5] is similar to the tracking algorithm described earlier. The objectives Z_{ic} , Z_{oc} are defined by a superpositionning of the corrector model variables and the errors to be reduced. The corrective action thus obtained is added to the action u_p to be applied the process.

Other decision centres, have a similar structure but do not utilise a corrector module which is replaced by algorithms for improving models.

SOFTWARE DEVELOPMENT FOR DECENTRALISED MULTIVARIABLE CONTROL

Software for handling the above mentioned algorithms has been developed based on two guidelines.

In the first place we have considered it essential to have facilities for real time restructuration of the decision centre algorithms [1]. This choice was transformed into the design of a system of decision centres based on fairly powerful microprocessors which could handle an interpretor for the real time restructuration commands. This real time interpretation had the role of generating a code which would permit the following functions :

- definition of data used,
- communication of data,
- construction of control algorithm using matrix operations represented as sub programmes.

Thus the equation of the linear model

$$x(i+1) = A\,x(i) + B\,u(i)$$

would be described to the interpretor as

```
* , A , x , z     (z = A x)
* , B , u , y     (y = B u)
+ , y , z , x     (x = y + z)
```

The algorithms presented in fig. 3 have been implanted on an Intelec 8 microcomputer and have been tested to control a process which was simulated on an analogue computer.

More recently, a second guideline was set and the software was produced at the "Atelier de Microinformatique de Grenoble". The primary options retained were :

- operational reliability in an industrial environment. Hence the choice of microprocessor boards with programmes on passive memories, a better maintenability and thus a less cumbersome means of providing back up centres.
- Need for the evolution of the software to keep pace with the evolution of the hardware. Modular design of programmes.
- Construction of standard multivariable control algorithms restructurable in real time. Hence an algorithm made up of a firm framework with specific functions activated or not by pointers. (e.g. corrector module).

It is this software which shall be studied in detail shortly.

Modular Conception of Software

A modular organisation of the software is imperative to simplify its practical implantation and to ensure its future evolution. We define four independant modules :

- communication with the environment,
- definition of the framework of standard control algorithms,
- matrix operators,
- arithmetic library.

We define thereafter, a protocol for the dialogue between these modules so as to render one module transparent to the other. Thus the modification of any module would not affect the others in any way. This feature is highly interesting in the case of microprocessors where the multitude of tools available render it difficult to transport software. This transportation is done in most of the cases by using passive memories containing a fixed module.

Floating point library : The control of complex systems necessitates floating point arithmetic. A representation on 4 bytes (sign, exponent, mantissa), generally termed single precision, permits a field of representation of decimal numbers from 10^{-20} to 10^{+18} with storage of the 7 significant digits of the mantissa. The device could either be programmed or hardwired. In all cases it would be necessary to define a 3 stage protocol :

- Acquire first operand (LOAD function),
- Acquire second operand and the type of operation desired (OPER function).
- Conservation of the resultant (STORE function).

We have actually three arithmetic devices, two of which are programmed arithmetic (FPA of INTEL and our FP.ATMI) and the third being hardware (AM 9511 of Advanced Micro Devices). Each one of this is based on an different type of internal representation.

Matrix operators : Control calculations are a series of matrix operations. The consitution of a set of generalised independant matrix operations of a particular algorithm is necessary for the dynamic structuration of the control system and for its future evolution.

Similarly the independance of these operators with respect to the arithmetic device is ensured by the protocol of dialogue described earlier and each operation on the elements of the matrix is activated by the following sequence :

LOAD address of first operand
OPER address of second operand
STORE address of resultant.

The standard algorithm communicates "parameters" to matrix operations (viz. address of the matrices concerned dimension of these matrices specification of the desired operations), to a memory unit specified by the protocol. Generally, it is convenient to use the stack in microprocessors where this is available.

The operators are classified into the following categories :

Type 1 - One matrix operand and two dimensions
operand = < operator > operand

Type 2 - Two operands, two dimensions
operand 2 = < operator > operand 1

Type 3 - Three operands, two dimensions
operand 3 = operand 2 < operator > operand 1

Type M - Three operands, three dimensions Matrix Products.

The complete set of operators for the microprocessor INTEL 8080 (product, sum, difference, transfer, transposition, reset to zero, change of sign, identity matrix) occupies less than 1 K byte of memory. The matrices are arranged column by column in the memory, as in the case of FORTRAN.

The standard algorithm : It is constituted by a sequence of calls to matrix operators and to branching out to the communication module. We can thus achieve the functions necessary for most of the computations in control theory, as presented in fig. 2 and fig. 3. It is well understood that all the decision centres would not need all the functions : Optional sequences are preceded by a word of flags (MODFLAG) whose bits are positionned by the structuration function of the communication module. For example, MODFLAG) permits to tailor the algorithm for the coordinator centre, for a local centre or for an isolated centre, or permits to incorporate the corrector or not etc... With the microprocessor INTEL 8080, the algorithms in their most comprehensive form occupy 1K byte of memory. They take about 2 seconds for execution for each sampling step with a model of 4 states, 4 inputs and 4 outputs, (the communication time with the environment not included).

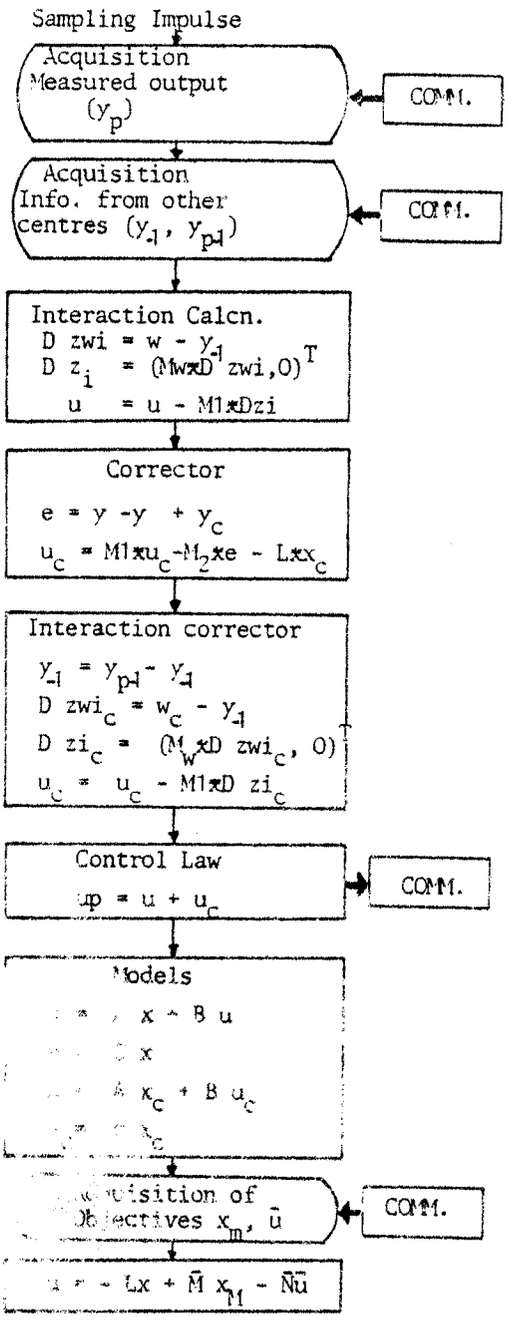


Fig. 4. Flow Chart of the Decision Centre Algorithm.

already suggested, the functions specific to each installation are activated by positioning pointers depending upon flags manipulated by the communications module.

We recall the three main parts of the computation :
- Evolution of the model and calculation of the control law by tracking of objectives defined by higher levels.

- Respecting of interaction with other centres of the same level, thus leading to constraints.

- Correction of directives to lower levels, using information provided by the lower levels.

Method of Design of the Software

A modular concept should be adopted and dialogue protocols between the different modules should be defined precisely. At first sight, this conception could seem to bloat up programmes and an additional time and effort for the first debugging operations. But, the future evolution of the software would be considerably reduced, especially in the follow ing cases :
- change of the arithmetic library,
- availability of hardware matrix operators,
- change of communication network, leading to a change of the communication module only.

One could thus construct a modular evolution of the software accompanied by a modular evolution of the hardware and that of the control structure.

CONCLUSION

The importance of progressive implantation of control systems has lead us to define an organisation consisting of a network of decision centres. The realisation of these centres based on microprocessors requires certain transformations of the algorithms and the software. The practical constraints posed by microprocessors implie certain simplification of the algorithms. Thus, modularity of algorithms is necessary for facilitating interchange of the basic elements of the microprocessor. We have thus resorted to the use of a tracking approach for the decision centre algorithms. These algorithms, having been tested in advance on a real life process, have proved to be robust and efficient. Similar tests on microprocessors have been satisfactory in as much as execution time and memory requirement are concerned. It must be noted that the software developed by us has a modular structure and can be easily used for constructing other control algorithms.

In this context we have taken up research on the synthesis of a multiprocessor centre with the basic algorithms presented in this paper. The hardware and software thus conceived lends itself very well for new methods using multiple models simultaneously.

REFERENCES

[1] Barrero, L., B. Caillot, P. Ladet, M. Silva-Suarez (1977). Les microprocesseurs dans la commande décentralisée des procédés industriels complexes. Symposium MIMI'77 - Zurich 6/77.

[2] Binder, Z., O. Badr, R. Perret (1975). Tracking approach to the control problems. Preprint of 6th World Congress IFAC, part IC-37.3 - IFAC - ISA Pitsburg 75.

[3] Binder, Z. (1977). Sur l'organisation et la conduite des systèmes complexes. Thèse Doct.-es-Sciences, Grenoble.

[4] Deschizeaux, P., P. Ladet (1977). Links definition and management in structuration of real time applications. IFAC/IFIP Workshop on Real Time Programing, Eindoven 6/77.

[5] Rey, D. (1978). Sur la commande décentralisée coordonnée. Application à un procédé pilote de distillation. Thèse Doct. Ing., Grenoble.

[6] Rey, D., A. Franco, Z. Binder (1979). Generalised tracking algorithm in the control of industrial processes. IFAC/IFIP Symposium SOCOCO 79, Prague 6/79.

[7] Takahashi, Y., M.J. Rabins, D.M. Auslander (1970). Control and dynamic systems. Addison Wesley Publishing Company.

[8] Zaks, R., P. Lebeux (1977). "Les Microprocesseurs". Ed. SYBEX 77.

=:=:=:=:=:=:=:=:=

The communication module : It is intended for the following three roles :

- Structuration of the standard algorithm by dialogue with the pole responsible for the structuration. i.e. positioning of the bits of MOD

- Synchronisation of the centre : the routine ATTN helps recognise the synchronisation signals. (e.g. synchronisation bip of the real time clock).

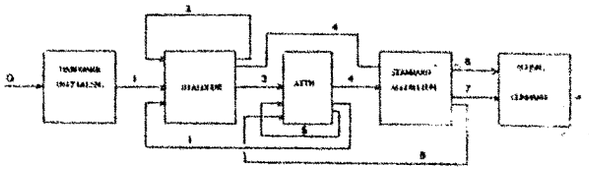
- Reception and dispatch of information to other control centres as defined by the following

ACQUENTR	acquisition of input measurements.
ACQSOR	Output measurements acquisition.
ACQCORD	Information from coordinator.
ACQCENT 1	Information from a neighbouring centre of the same level.
COMCMD	Dispatch orders to instrument level.
COMCENT 1	Dispatch information to neighbours.

The address of these routines, the data stored in memory and the FLAG word constitutes the dialogue protocol with the standard algorithm.

Functional Organisation of a Decision Centre

Each decision centre should fulfil the two functions
i) definition of the centre by dialogue with the environment and ii) computation of the control law by executing the algorithms. The graph of fig. 5 illustrates the operational organisation adopted for the centres of the control system.



Events

0 RESET hardware
1 Initialise dialogue
2 Activate dialogue task
3 End of dialogue
4 Sampling bip. (algorithm starts)
5 WAIT loop
6 Inter - centre communication
7 End of Inter centre communication
8 End of algorithm.

Fig. 5. Operational State Diagram of the Decision Centre.

From this point of view it is observed that the centre is constituted of two programmes :

- a low priority programme, interruptible by the "bip" from the real time clock. It is composed of dialogue with the environment and a wait loop of the communication module.

- A uninterruptible top priority programme composed of the standard algorithm, the matrix operators, arithmetic devices and the inter - centre communication part of the communication module.

It is important to note that certain parts of the dialogue should mask the interruption during the period of their execution e.g. the validation of new objectives must not miss the calculation step.

CONCEPTION OF A MICROPROCESSOR BASED CONTROL SYSTEM

The experience gained during the realisation of the software presented above leads to the following observations on the use of microprocessors in the control of complex systems and on the methods which seem useful to employ if these ventures have to be successful.

Functional Organisation of Control.

The control system is composed of decision centres. If the quadratic optimisation part is calculated off-line by a mini computer and the real time part (i.e. the evolution of the module, the feedback loops, anticipation terms and correction terms) could be implanted on a microprocessor based centre.

Thus each centre fulfils two functions : an environment dependant function and computation function based on a standard algorithm.

The communication function : has three principal roles. They are :

i) structuration of the algorithm by activation of specific computations. This helps modifying and real time restructuring of control algorithms distributed over different centres. This structuration itself is composed of acquisition of matrices necessary to the calculations, the dimensions of these matrices being dynamically determined. This role is fulfilled by communicating with the unit responsible for structuration of the control.
ii) With other centres of the same hierarchical level, the respecting of interactions is achieved. With a higher level, the objectives to be followed are obtained. Directives are transmitted to a lower level.

iii) Synchronisation, particularly detection of sampling impulses for triggering the computations.

These roles could be achieved depending on the implantation of the control. One could quote for example their triggering either by external interrupts or by requests through input/output devices or by consulting dedicated memory words which would be positionned by special transparent devices (e.g. DMA).

The computation function : The algorithm utilised is a standard algorithm. As we have