N d'ordre : 207

THESE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

pour obtenir le titre de

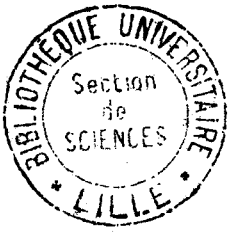
DOCTEUR INGENIEUR

par

Michel MARQUETTE

Ingénieur IDN

SUR UN SYSTEME DE SIMULATION HYBRIDE



Soutenue le 28 Juin 1977, devant le Jury d'examen :

M.	C.	MAIZIERES	Président
M.	J.C.	GENTINA	Rapporteur
M.	F.	LAURENT	Examinateur
M.	P.	BORNE	Examinateur
M.	R.	BOSSUT	Invité
M.	G.	ROY	Invité

Avant propos

Le travail que nous présentons dans ce mémoire a été effectué au Laboratoire de Systématique de l'Université des Sciences et des Techniques de Lille I.

Nous sommes particulièrement sensible au grand honneur que Monsieur Maizières nous a fait en acceptant de présider notre Jury de Thèse. Qu'il trouve ici l'expression de notre profonde reconnaissance.

C'est un agréable devoir pour nous d'exprimer notre profonde gratitude à Monsieur Jean Claude Gentina, Maitre Assistant et chargé de cours à l'Institut Industriel du Nord qui a bien voulu nous aider et nous guider durant l'élaboration de notre thèse. Qu'il trouve ici l'expression de notre très vive reconnaissance pour l'enseignement qu'il a su nous dispenser, l'intérêt qu'il a porté à nos travaux et les conseils éclairés qu'il nous a prodigués.

Nous voulons exprimer notre profonde gratitude à Monsieur Laurent, Maître de Conférences à l'Université de Lille I et Directeur du Laboratoire de Systématique qui a bien voulu nous accueillir au sein de son équipe. Nous lui sommes reconnaissant pour l'enseignement qu'il nous a dispensé et l'intérêt qu'il nous a porté au cours de nos recherches.

Nous tenons à exprimer notre très vive sympathie à Monsieur Borne, Maitre Assistant à Lille I et chargé de cours à l'Institut Industriel du Nord dont le concours a été très précieux. Nous le remercions d'avoir accepté de participer à notre Jury.

Monsieur le Professeur Bossut, Directeur de L'Institut Industriel du Nord, nous a , à tout moment, apporté son soutien et plus particulièrement en acceptant de participer à notre Jury de Thèse. Nous tenons à ce qu'il reçoive ici le témoignage de notre très respectueuse reconnaissance ainsi que l'expression de notre sincère dévouement.

Nous sommes particulièrement heureux que Monsieur Roy, Chef de service à la SINTRA-EF ait bien voulu participer à notre Jury de Thèse, en nous apportant ainsi l'appréciation, très utile du milieu Industriel. Nous l'en remercions très sincèrement.

Le concours des élèves ingénieur de l'Institut Industriel qui dans le cadre de projets d'option ont testés et utilisés les résultats de cette thèse a été particulièrement précieux. Nous témoignons à ces élèves notre reconnaissance et leur exprimons toute notre sympathie.

Nous ne saurions terminer cet avant propos sans exprimer notre reconnaissance à tous les chercheurs de l'équipe du Laboratoire de Systématique pour l'aide précieuse qu'ils nous ont apportée à la fois sur le plan scientifique et sur le plan des contacts amicaux.

A chacun nous adressons nos remerciements les plus vifs.

INTRODUCTION GENERALE

L'étude des systèmes complexes ne peut généralement être totalement appréhendée sur le plan théorique et nécessite de ce fait le recours aux méthodes expérimentales. La simulation peut donc être considérée comme un outil de base permettant de résoudre ce type de problème.

Le degré de complexité d'un système apparait essentiellement au niveau de ses fonctions de base ou découle tout simplement de ses caractéristiques de taille et de structure. Il convient toutefois de présiser qu'une simulation ne peut être envisagée que dans l'hypothèse où il est possible de définir un modèle de référence semblable au processus. Les lois de la similitude jouent donc un rôle fondamental en simulation.

Les moyens techniques concernant la simulation des systèmes peuvent être classés différemment selon d'une part la nature des informations traitées et d'autre part la nature des modèles utilisables.

Il peut s'agir tout d'abord de la réalisation d'une maquette vérifiant les lois de la similitude. Cette technique est utilisée fréquemment dans le domaine de la mécanique des fluides.

Si l'on dispose d'un modèle à caractère mathématique on peut être conduit à mettre en oeuvre des simulations électroniques ; nous en distinguerons essentiellement deux classes.

L'ordinateur a tout d'abord été considéré comme l'outil le plus puissant, il a pleinement justifié ce point de vue par ses nombreuses applications au traitement des informations discrètes, booléennes ou codées (numériques). Toutefois sa mise en oeuvre devient délicate lorsqu'il s'agit de traiter une information continue par nature.

Dans ce cas, les notions d'échelle des temps et de dynamique temporelle d'un processus ne sont pas définis de manière naturelle ou implicite. Il est alors préférable de recourir à l'utilisation d'une autre classe de calculateurs : les calculateurs à courant continu, aussi appelés calculateurs analogiques.

Pour ce type de calculateur, l'information est traitée en continu selon un mode parallèle. Toute variable évolue donc implicitement par rapport au temps, et il est généralement possible de régler "l'échelle des temps" dans le sens soit d'une accélération soit d'un ralentissement de sa dynamique /1/, /2/, /3/.

L'évolution de ce second type de calculateur a pu s'effectuer en deux étapes. Dans une première phase, l'adjonction aux opérateurs analogiques d'opérateurs booléens et hybrides a permis d'accroître le champ d'utilisation de ces matériels. La technique de programmation par cablage semble dans ce cas plus délicate du fait du nombre important des opérateurs de base disponibles et de la nature à la fois booléenne et continue des informations traitées. Dans un deuxième temps, le couplage d'un calculateur à opérateurs logiques et analogiques cablés et d'un calculateur digital a permis de résoudre ces difficultés tout en offrant la possibilité de simulations à la fois plus importantes et plus complexes. /4/, /5/, /6/, /7/.

L'ensemble de traitement ainsi obtenu, appelé calculateur hybride de type II, présente donc deux avantages principaux .

En effet aux instruction à exécution séquentielle des calculateurs numériques il associe la possibilité d'opérations spatiales et temporelles selon un mode parallèle.

De plus, la possibilité de traitements simultanés accroit la vitesse d'exécution des calculs.

L'étude d'un couplage à hautes performances conduit cependant à résoudre un nombre important de problèmes tant sur le plan du matériel que du logiciel.

C'est le cas notamment pour les adaptations d'informations généralement réalisées à partir d'interfaces de codage et de décodage. De nombreuses difficultés apparaissent également au niveau de la gestion des échanges entre les deux calculateurs.

Le premier problème est résolu dans le sens analogique numérique par la mise en oeuvre de convertisseurs analogiques digitaux (ADC) fortement multiplexés. Le transfert inverse s'effectue à partir de convertisseurs numériques analogiques (DAC) et d'atténuateurs numériques (DAM). Ces éléments ont actuellement des caractéristiques de fonctionnement trés satisfaisantes en précision comme en fréquence.

La classe des calculateurs digitaux "couplables" correspond à notre sens aux mini-calculateurs industriels en temps réel. De tels systèmes sont particulièrement adaptés à la gestion temps réels d'échanges avec leur environnement. Les produits actuellement disponibles sur le marché des "mini" ont donc généralement une aptitude implicite au couplage.

Malheureusement les logiciels fournis par les constructeurs ne peuvent être directement utilisés en simulation si ce n'est au niveau de langage peu évolués tels les assembleurs.

En effet, les moniteurs temps réel classiques ont pour but essentiel de gérer d'une part les tâches correspondant au matériel et au logiciel et d'autre part de piloter les entrées-sorties des systèmes, en particulier les périphériques informatiques. De tels moniteurs assurent généralement la gestion des événements et des fichiers. L'ensemble des fonctions moniteurs est géré sur un fichier informatique qu'il est possible de configurer et éventuellement de compléter. Dans le contexte d'une application au calcul hybride, il convient donc d'introduire une bibliothèque de fonctions spéciales permettant de contrôler les opérateurs analogiques

et booléens ainsi que les transferts bidirectionnels d'informations.

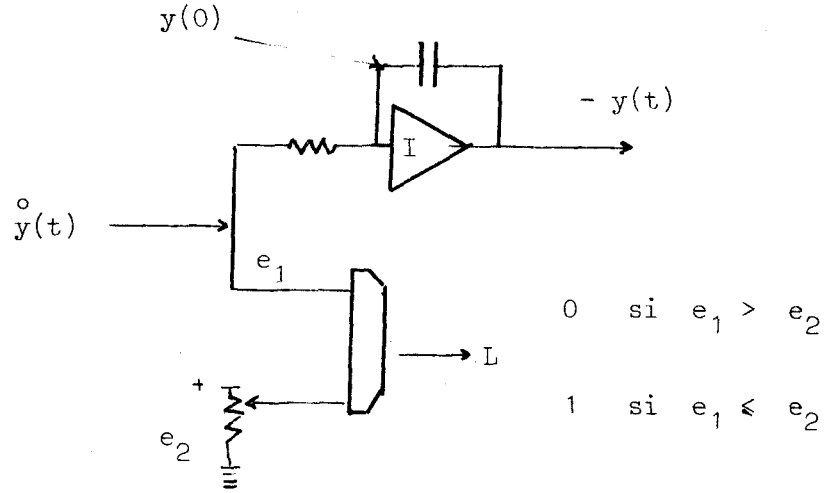
L'utilisation d'un logiciel déjà existant permet toutefois de multiplier les traitement à caractères informatiques.
Ainsi, l'utilisation de langages évolués tels le Fortran, le PL, ou même le Basic, amène une réduction des difficultés inhérentes à la conception et à la mise au point des programmes. En contrepartie, les temps de traitement et le volume mémoire nécessaires à l'exécution de tels systèmes sont souvent trés importants. Les contraintes qu'imposent la simulation par calculateur hybride supposent au contraire des traitements rapides ou en temps réel. Pour ces raisons, les constructeurs de calculateurs hybrides ou parfois les utilisateurs de ces mêmes matériels ont été amenés à étudier et à mettre au point des logiciels dits de simulation . /8/,/9/,/10/,/11/,/12/, /13/,/14/.

C'est ainsi que nous aborderons dans ce mémoire l'étude puis la description d'un moniteur de simulation hybride associé à un langage de simulation./15/. Le logiciel que nous présenterons a volontairement été orienté dans le sens d'une application relative aux matériels dont nous disposerons
Annexes 3
Afin de préciser davantage le "cahier des charges" d'un tel moniteur il convient d'illustrer par un exemple simple les problèmes spécifiques du calcul hybride. Le problème que nous proposons d'aborder à cet effet concerne la recherche de l'extremum d'une fonction dont on ne connait que la dérivée temporelle à tout instant.

Notons $y(t)$ une fonction d'évolution continue par rapport au temps. L'annulation de la dérivée $\overset{\circ}{y} = \frac{dy}{dt}$ de cette fonction est une condition nécessaire d'existence d'un extremum de y. A partir des données respectives de la condition initiale $y(0)$ et du signal $\dot{y}(t)$, il est donc possible d'élaborer instantanément $y(t)$ ainsi qu'une information booléenne $L \in [0,1]$

sur le signe de la dérivée $\dot{y}(t)$.

Il vient le schéma analogique suivant



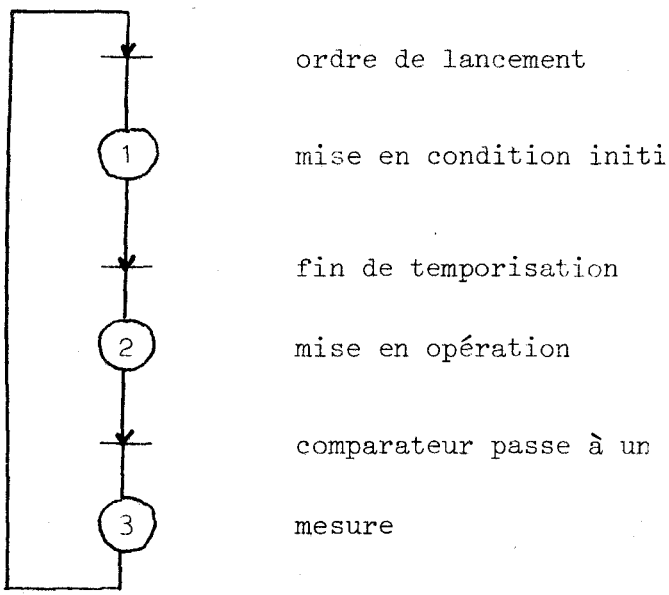
Figue 1

La séquence de calcul réalisée consiste tout d'abord à mettre en conditions initiales l'intégrateur I pendant un temps suffisant pour qu'il puisse atteindre son état d'équilibre.

Dans une deuxième étape, l'intégrateur I_1 est mis en "mode calcul" La fonction f(t) évolue alors en fonction du temps.

La troisième étape débute lorsque le comparateur détecte le passage par zéro de la dérivée $\dot{y}$.
Les intégrateurs sont alors mis en mode mémoire et la mesure de y est effectuée.
Ce traitement est donc obtenu par l'enchainement séquentiel de plusieurs

phases. Chacune réalise une ou plusieurs opérations élémentaires spécifiques du calcul hybride, telles les commandes d'intégrateurs ou les prises de mesure. Le déroulement de chaque séquence est assujetti à autant de contraintes qu'il y a de phase : un ordre opérateur pour le lancement du calcul, la fin d'une temporisation pour la deuxième phase et pour le troisième le passage à un du comparateur. Une phase ne peut être atteinte que lorsque la contrainte qui la conditionne est réalisée. Des outils tels que les graphes de Petri ou de Girard permettent en général la description de ces séquences. (figure 2) /15/,/16/,/17/,/18/,/19/,/20/,/21/,/23/.



Dans une même phase les opérations hybrides élémentaires s'enchainent séquentiellement. Par exemple pour la troisième phase la prise de mesure succède à la mise en mémoire des intégrateurs. Par contre au passage d'une phase à l'autre l'exécution des opérations s'effectue au synchronisme de l'événement correspondant à la réalisation de la contrainte.
Nous distinguerons donc deux types d'évolutions transitoires selon qu'il s'agit d'exécutions enchainées purement séquentielles ou synchronisées par événements. Le moniteur hybride doit donc satisfaire à ces deux contraintes. Si la première semble naturelle puisque les instructions d'un calculateur

numérique sont exécutées selon un mode séquentiel, la réalisation du mode "synchone" doit faire l'objet d'une étude particulière. Il convient également de préciser que le moniteur doit assurer la gestion des échanges entre les deux parties analogique et hybride de cet ensemble hybride. En plus des fonctions propres à la machine hybride, le moniteur doit reprendre les fonctions de gestion de tâches, des fichiers et périphériques informatiques déjà existantes.

C'est dans cet esprit que nous avons développé le logiciel hybride et nous proposons d'exposer ce travail selon une présentation en trois parties principales.

Dans un premier chapitre, nous définissons les solutions adoptées pour résoudre à la fois les problèmes relatifs à la synchronisation des échanges et à la définition des fonctions moniteur. Les caractéristiques du moniteur hybride sont ensuite précisées.

Un deuxième chapitre traite du langage hybride et du compilateur qui lui est associé.

Afin de préciser l'aspect pratique d'un tel logiciel au niveau de la mise en oeuvre, nous traiterons dans un dernier chapitre trois exemples de simulation hybride.

CHAPITRE I

DESCRIPTION GENERALE DU MONITEUR HYBRIDE

INTRODUCTION

Le moniteur hybride peut être considéré comme un système d'exploitation dont les buts essentiels concernent à la fois la gestion de l'ensemble des deux calculateurs et l'exécution des programmes hybrides. Ces programmes sont décrits sous forme de listes instructions.
Au même titre que les informations circulant entre les parties analogiques et logiques, ces listes peuvent être considérées comme une source de données pour le programme de gestion. Le programme de gestion assure ainsi la mise en oeuvre des opérations dites "hybrides".

Avant d'aborder la description de ce système il convient - dans un premier temps - de définir par l'analyse des notions fondamentales du calcul hybride le concept d'instruction hybride.

Les contraintes inhérentes au choix des matériels utilisés sont mises en évidence dans une seconde étape.

Il sera alors possible de définir les fonctions du programme de gestion et d'en préciser le mode de réalisation.

A toutes fins utiles, la liste des opérations élémentaires est donnée en annexe technique.

I - NOTIONS DE BASE

I.1 - Opérations et instructions hybrides

Le choix d'un support matériel adapté à la résolution d'un problème de simulation peut conduire à utiliser un ensemble de calcul hybride. Ce matériel est programmé à deux niveaux , par cablage d'opérateurs logiques ou analogiques, ou par écriture d'une liste d'instructions hybrides en mémoire vive du calculateur. Ces instructions sont spécifiques des techniques hybrides utilisées. Elles doivent dans tous les cas conduire à une écriture et à une mise au point aisée des programmes.

Une première classe d'instructions assure la gestion des échanges entre les deux calculateurs. Elle contient :

i) Les commandes d'intégrateurs, des track and store, d'interrupteurs électroniques qui sont de nature booléenne.

ii) Les sorties tout ou rien ou analogiques.

iii) Les opérations effectuées par la chaîne de mesure qui sont de natures numérique ou analogique.

Certaines opérations ne sont pas spécifiques du calcul hybride. Elles en facilitent toutefois l'utilisation au niveau de la programmation du système. Nous proposons de les regrouper en deux classes suivant la nature de leurs applications.

D'une part des opérations à caractère purement informatique permettent de gérer les fichiers et les périphériques. Les plus couramment

utilisés sont le télétype de service, le lecteur, le perforateur rapide et le disque. L'implantation des fichiers, aussi bien en mémoire centrale que sur disque doit être transparente pour l'utilisateur.

D'autre part des opérations logiques contrôlent le déroulement des séquences et opérations.

Cette dernière classe contient par exemple les instructions de rupture de séquence conditionnelle ou inconditionnelle et les instructions de gestion des boucles.

Une opération hybride étant le plus souvent associée, par définition, à un ou plusieurs paramètres ou opérandes, nous dirons globalement qu'il s'agit d'une instruction hybride.

Symboliquement cette instruction pourra s'écrire sous forme de phrase selon une syntaxe adaptée.

I.2 - Synchronisation des opérations hybrides.

2.1 - Evénements

Par définition un traitement hybride peut se dérouler à la fois sur le calculateur numérique et sur la calculatrice analogique selon un mode parallèle.

Le calculateur digital a pour charge d'exécuter séquentiellement des instructions machine. L'évolution temporelle du traitement ne dépend que du temps d'exécution de chacune des instructions. Le temps nécessaire à la résolution d'un problème et donc directement lié au volume des programmes utilisés.

Contrairement à ce qui se passe sur le calculateur numérique, toutes les variables présentes sur la calculatrice analogique sont par définition des fonctions continues du temps. Ceci est dû au fait que l'intégrateur par rapport au temps y est une des opérations de base.

L'intéraction des opérations numériques et analogiques conduit à choisir une échelle de temps de référence. A priori il existe deux possibilités : soit utiliser une échelle de temps associée à la calculatrice analogique, soit synchroniser la logique sur une base de temps.

Afin de pouvoir faire un choix entre ces deux méthodes il est nécessaire d'étudier de manière plus précise la nature des intéractions entre variables analogiques et variables logiques.

Sans restreindre la généralité de l'étude nous envisagerons, à titre d'illustration, le cas de la simulation d'un processus.
Nous supposerons le processus décrit à partir d'un système d'équations différentielles. La simulation de ce modèle peut être envisagée par cablage de la partie analogique.

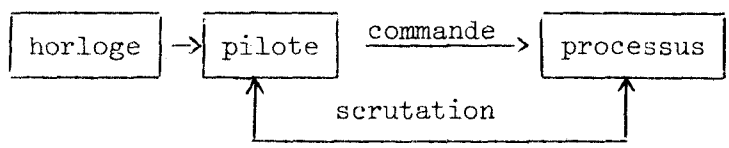
Le calculateur numérique éventuellement complété par la partie logique de la calculatrice analogique a pour fonction essentielle de piloter le processus par des commandes logiques et des sorties analogiques.

Deux méthodes peuvent être envisagées pour réaliser un retour d'information du processus vers le pilote :

Le première, inspirée des techniques d'échantillonnage consiste à faire une scrutation périodique des variables d'état du processus.

La seconde a pour but de minimiser le volume des échanges. Elle consiste à éffectuer le retour d'information lorsqu'une condition externe réalisée au niveau du processus est réalisée. Cette condition permet de définir l'instant ou l'échange d'information est utile.

La première méthode s'adapte trés bien au traitement des problèmes échantillonnés. Dans ce cas il convient de synchroniser, sur une base de temps, la logique au pilote



La scrutation étant à l'initiative du pilote la structure obtenue est du type "boucle ouverte".

Dans le but de conserver l'aspect continu des variables, la fréquence devra satisfaire aux contraintes usuelles exprimées par Shannon [1]. Ceci n'est donc concevable que pour des systèmes à évolution lente et à partir d'un échantillonnage trés rapide.

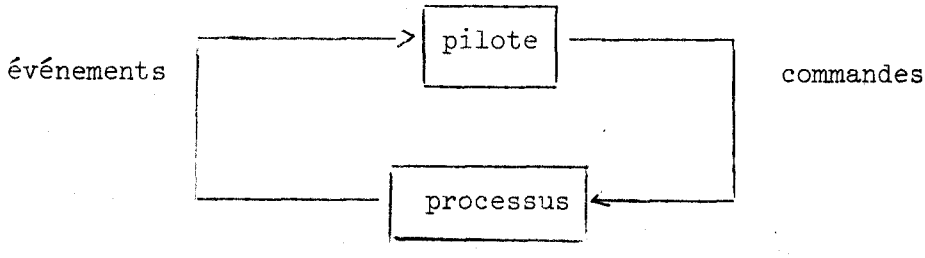
La seconde méthode permet de pallier ces inconvénients. Par contre elle nécessite de savoir à quel moment il convient de demander au pilote une action particulière. Cet instant est atteint lorsque le processus passe par un état de référence choisi à priori par l'opérateur.

Le pilote ne disposant pas instantanément des informations nécessaires c'est au niveau du processus que sera définie la référence ; lorsque cette dernière est atteinte le processus prévient le pilote par l'émission d'un signal. Celui-ci est de nature booléenne et par définition nous l'appellerons "événement".

Dés la réception de l'événement le pilote lui associe un code qui servira à l'identifier dans la suite du traitement.

Les événements représentent donc l'ensemble des informations transmises au pilote à l'initiative seule du processus.

La structure obtenue est alors du type "boucle fermée" en ce sens que le processus peut à certains moments décider d'une exécution particulière au niveau de l'ensemble. La décision est ainsi partagée entre le pilote et le processus suivant le schéma ci-dessous :



Il convient de noter que le traitement des informations liées à un événement peut être alors envisagé sans difficulté.

Pour implémenter cette méthode nous avons choisi un mini calculateur industriel et utilisé un logiciel temps réel. Cet ensemble de traitement entraine généralement pour le pilote des durées de traitement faibles en regard de l'évolution en temps du processus. En conséquence le pilote se

trouve fréquemment en attente d'événement. Pour cette raison l'échelle de temps sera celle du processus.

Avant de donner une description plus détaillée de la synchronisation par événement il est nécessaire de définir les différents modes d'exécution des instructions.

2.2 - Macro-instructions

Une phase de calcul réalise l'exécution d'un groupe d'opérations hybrides élémentaires. Elle est décrite par une "macro-instruction". Celle-ci par définition est un ensemble d'instructions élémentaires dont la mise en oeuvre globale est conditionnée par la réalisation d'un événement sur le processus.

La première instruction joue un rôle trés particulier. En effet elle ne peut être mise en oeuvre qu'à partir d'un ordre de traitement. Les suivantes sont alors exécutées séquentiellement.
Par définition nous dirons que le mode de la première instruction est du type "déclenché" et celui des suivantes du type "séquentiel".

2.3 - Mode déclenché

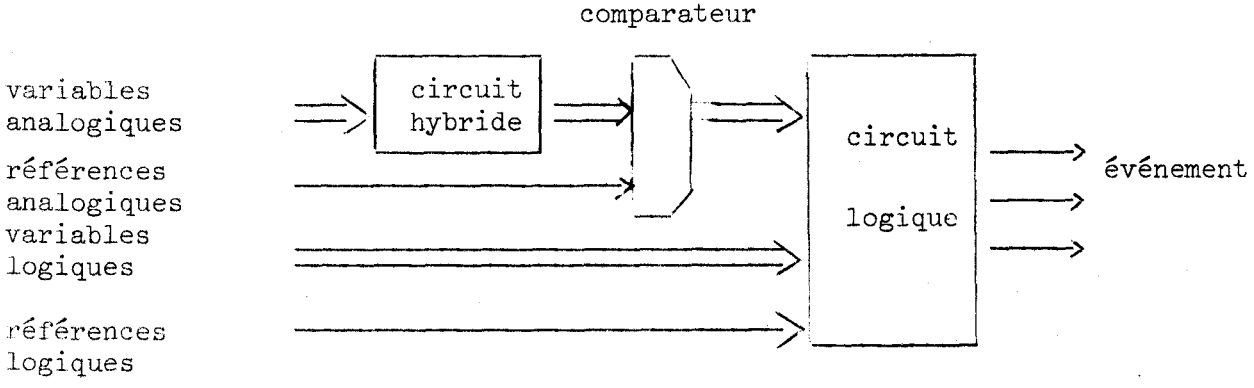
D'une façon générale un état de référence peut être représenté sous la forme d'un ensemble de valeurs prises par des variables logiques, analogiques ou une combinaison des deux.

Pour une variable logique l'état de référence correspond au niveau associé à la valeur zéro ou un d'une constante booléenne. L'événement est alors généré à partir d'un comparateur qui peut être par exemple une simple porte ET.

Dans le cas des variables analogiques les références sont également des constantes. Il est alors nécessaire d'utiliser des comparateurs analogiques de manière à coder sous forme booléenne l'événement correspondant.

La combinaison des variables analogiques peut se faire soit à travers un circuit hybride en amont des comparateurs soit sur les variables logiques qui en sont issues.

La combinaison de plusieurs variables booléennes se fait par un circuit logique dont les sorties génèrent les événements. La structure globale d'un détecteur de passage par un état de référence est illustrée par le schéma suivant :



L'expérérience a montré qu'en général le nombre de variables réellement utilisées est trés faible. En effet les références les plus fréquemments utilisées sont l'état d'un interrupteur pour le lancement manuel de l'exécution d'un ou plusieurs macro instructions, les compteurs branchés sur l'horloge temps réel pour les différentes temporisations, l'état d'un comparateur. Dans chacun de ces exemples n'intervient qu'une seule variable carac-

téristique de composants simples tels les interrupteurs, les compteurs ou les comparateurs. Afin de transférer l'information dite "événement" du calculateur analogique au calculateur numérique, nous avons été amené à utiliser au mieux la structure temps réel du calculateur digital et en particulier le système prioritaire. Dans ce sens, à chaque événement est associé soit une interruption du matériel (hard), soit - pour des raisons de facilités d'emploi - des interruptions dues au logiciel.
De manière à ce qu'une instruction ne soit déclenchée que par l'événement précisé par l'utilisateur il est nécessaire d'une part de repérer au niveau de l'instruction cet événement et d'autre part d'identifier clairement toutes les interruptions pouvant survenir. Dans ce but, la première instruction de chaque macro instruction contient le code correspondant à l'événement conditionnant son exécution. Ce code est appelé par définition"code de déclenchement". De plus à chaque événement est associé un code. Dans le pratique celui-ci est déterminé par l'analyse de l'origine de l'interruption associé à l'événement. Le programme de gestion a pour objectif essentiel la comparaison du code de déclenchement de l'instruction en cours aux codes des interruptions effectives ou en attente. En cas de concordance du code l'instruction est exécutée. Sinon l'événement est ignoré. On réalise de ce fait une filtrage des événements parasites.

Une instruction ne peut donc être déclenchée que par un seul événement. Il est toutefois possible de composer plusieurs événements dans le but d'exécuter une seule instruction : Ce problème se résoud facilement en cablant un circuit supplémentaire de composition des variables logiques.

D'un autre point de vue, il peut être beaucoup plus intéressant de générer plusieures tâches différentes même indépendantes à partir d'un événement unique ; c'est ce que nous proposons d'exposer dans le paragraphe suivant.

I.3 - Instructions et programmes hybrides

3.1 - Instructions hybrides

Une instruction comporte un certain nombre d'éléments caractéristiques de la tâche à accomplir.

Il convient de définir d'une part son mode d'exécution et d'autre part le traitement à éffectuer.
Le mode peut être soit du type séquentiel, soit du type déclenché. Dans ce dernier cas il faut préciser le code de déclenchement de la macro-instruction.

Le traitement est défini par un code opération qui donne accés à la bibliothèque des sous-programmes qui réalisent l'exécution des opérations hybrides élémentaires. La description du traitement est complétée si nécessaire par un ensemble de paramètres.

3.2 - Programmes hybrides

D'une façon générale les macro-instructions relatives à la résolution d'un problème sont regroupées en listes ou "programmes hybrides".

Afin de faciliter l'analyse et le traitement des problèmes hybrides nous envisagerons la possibilité pour un même événement de mettre en oeuvre plusieurs tâches éventuellement indépendantes.

Afin d'illustrer ce point de vue citons l'exemple du programme de protection de la calculatrice analogique en cas de saturation d'amplificateurs. L'événement relatif à la détection de la saturation est connecté sur

un appel prioritaire, et met en oeuvre l'opération de mise en gel de tous les intégrateurs. Ce programme est totalement indépendant des programmes relatifs au traitement du problème étudié.

De cette façon, il est possible d'exécuter plusieurs programmes en parallèle et de façon quasi-simultannée. Cette méthode permet de réaliser par logiciel la fonction de multi processeur. Compte-tenue des besoins usuels relatifs aux traitements hybrides le nombre de ces programmes est égal, au maximum, à quatre.

La priorité des traitements correspond à l'ordre dans lequel ont été écrits ces programmes.

I.4 - Structure globale du programme de gestion

La structure du programme de gestion découle de l'analyse séquentielle des tâches qu'il doit accomplir.

Tout d'abord les interruptions effectives sont codées par programmation aprés reconnaissance de leur origine.

Le programme de gestion compare alors les codes des interruptions aux codes de déclenchement. En cas de concordance il y a réalisation de la ou des macro-instructions et exécution des opérations hybrides qu'elles décrivent.

Les instructions des programmes hybrides constituent ainsi une deuxième source d'informations pour le programme de gestionau même titre que les événements.

Afin de simplifier la présentation de ce premier organigramme, nous n'avons pas décrit la fonction de multi processeur. Le schéma de la figure I.1 décrit globalement les phases principales du programme de gestion. Nous proposons de reprendre l'étude détaillée de chacun de ces éléments.

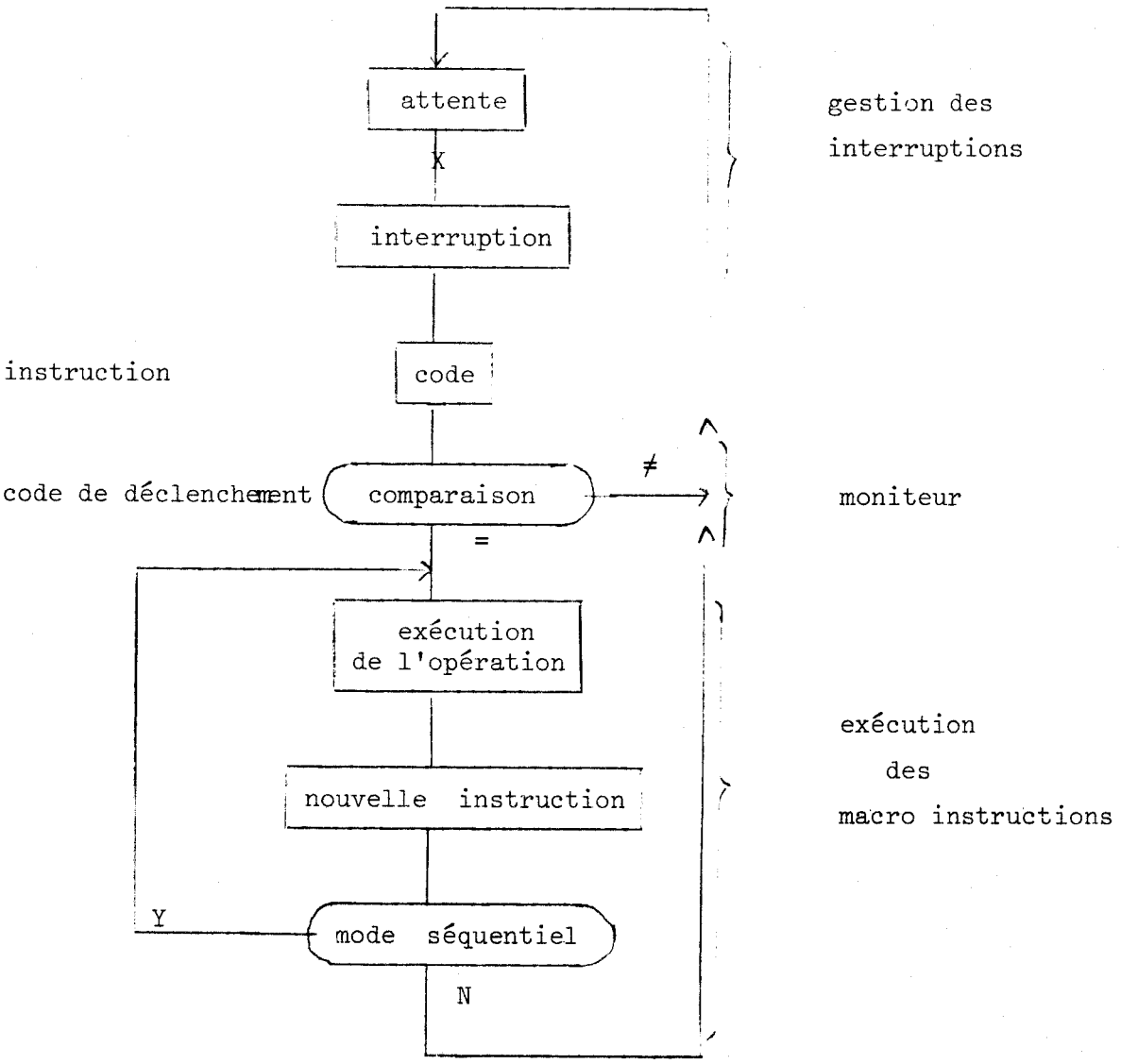


Figure I.3 - Structure globale du moniteur

II - GESTION DES INTERRUPTIONS

II.1 - Hiérarchie des traitements

La structure temps réel du calculateur autorise une programmation suivant plusieurs niveaux de priorité.

Les tâches "hard" qui traitent les interruptions dues au matériel sont les plus prioritaires. Elles sont classées selon une hiérarchie cablée sur le calculateur. Chacune peut à tout moment si elle n'est pas masquée interrompre une tâche de priorité plus basse.

Les tâches qui ne sont pas engagées par interruption prioritaire du matériel sont globalement d'un niveau de priorité inférieur . Elles possèdent aussi une hiérarchie programmée correspondant à l'importance du traitement qu'elles effectuent.

Le programme de gestion assurant le contrôle des interruptions est le plus prioritaire et de ce fait réside en mémoire vive. Il correspond au "foreground".

Vient ensuite un ensemble de tâches, correspondant au middleground, à la disposition de l'utilisateur. Elles sont généralement utilisées pour des calculs, l'exploitation des résultats, ou la préparation de données. Elles sont activées par celle de plus bas niveau ou IDLE.

Elles ne doivent pas être confondues avec les sous programmes de calcul sur lesquels peut se brancher temporairement le programme de gestion.

La taille mémoire étant réduite, les informations volumineuses et les tâches peu importantes d'utilisation rare ne résident pas en mémoire centrale. Elles sont rappelées du disque au moment de leur utilisation en mode canal.

Un travail de fond peut être exécuté pendant les temps morts, en particulier en phase d'attente d'interruption. Il est dit de niveau "back ground".

Du fait des hiérarchies, une tâche peut à tout moment être interrompue par une tâche plus prioritaire. Ce traitement de la tâche interrompu doit ensuite être repris. Dans ce cas il est nécessaire de sauvegarder les éléments caractéristiques de la tâche interrompue momentanément. Ces éléments ou contextes sont les contenus des registres du calculateur, à savoir l'accumulateur (A) et son extension (B), l'index (X) et le tampon (T). L'état (E) est fixé en initialisation et n'est modifié que localement.

En résumé - Les tâches peuvent être regroupées suivant leur priorité en quatre groupes :

- Traitement des interruptions
- Programme de gestion (foreground)
- Programme de calcul (middleground)
- Programme de fond (background)

Nous proposons d'analyser dans un premier temps les raisons qui nous ont amené à ne pas envisager de sauvegarder le contexte du foreground et dans un deuxième temps les traitements relatifs aux interruptions. La description des autres programmes fait l'objet des paragraphes suivants.

II.2 - File d'attente

Pour des raisons d'encombrement il n'est pas judicieux de réserver une zone mémoire par interruption pour la sauvegarde du contexte. L'utilisation d'une seule zone de sauvegarde impose le masquage de tous les niveaux de priorité durant le traitement des interruptions pour éviter l'écrasement d'un contexte par un autre. Dans ces conditions il y a risque de voir apparaitre

simultanément, sur un même niveau de priorité, plusieurs interruptions. Le dernier appel étant le seul mémorisé, les premiers sont alors perdus puisqu'ils ne peuvent pas être traités à temps.

Il est donc nécessaire de réduire au minimum la durée de ces traitements. Dans ce sens chaque appel engendre le codage de l'interruption, suivi de sa mémorisation dans une file d'attente.
Dans ce cas le niveau relatif de priorité de l'interruption dans la hiérarchie est perdu. Par contre compte-tenu de la minimisation du temps de traitement la probabilité de voir apparaitre plusieurs appels quasi simultanés est faible.

Ces raisons expliquent le choix qui a été fait dans le sens d'un compromis entre le temps de traitement et le volume de mémoire utilisée.

Afin de pouvoir tout de même utiliser partiellement l'intérêt que présente la hiérarchie cablée, certaines interruptions sont réservées à des branchements directs à des sous programmes effectuant des traitements spécifiques.

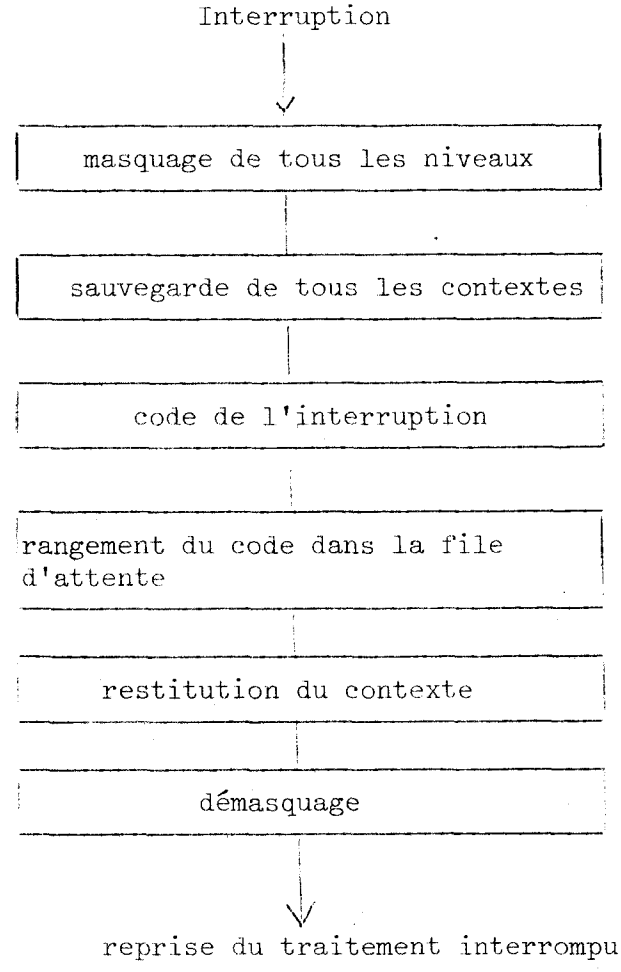
II.3 - Traitements des interruptions

Le traitement d'une interruption aboutit d'une manière générale à la génération d'un code. Les exceptions concernent d'une part l'horloge temps réel qui donne lieu à une gestion préalable de compteurs et d'autre part les interruptions qui aboutissent aux branchements à des sous programmes spécifiques.

3.1 - Traitement général (figure II.1)

Ce traitement est déduit de la nécessité d'une file d'attente. Aprés avoir masqué tous les niveaux et sauvegardé le contexte l'interruption

est codée, le code obtenu est ensuite rangé dans la file d'attente. La reprise du traitement interrompu se fait aprés restitution du contexte et démasquage.



Traitement général d'une interruption

Figure II.1

3.2 - Traitement de l'horloge temps réel (figure II.2)

L'utilisation principale de l'horloge temps réel a été orientée dans le sens de la génération de temporisation . Celles-ci sont réalisées par des compteurs incrémentés à chaque interruption d'horloge. Seul l'événement de passage par zéro du compteur entraine le codage d'un événement.

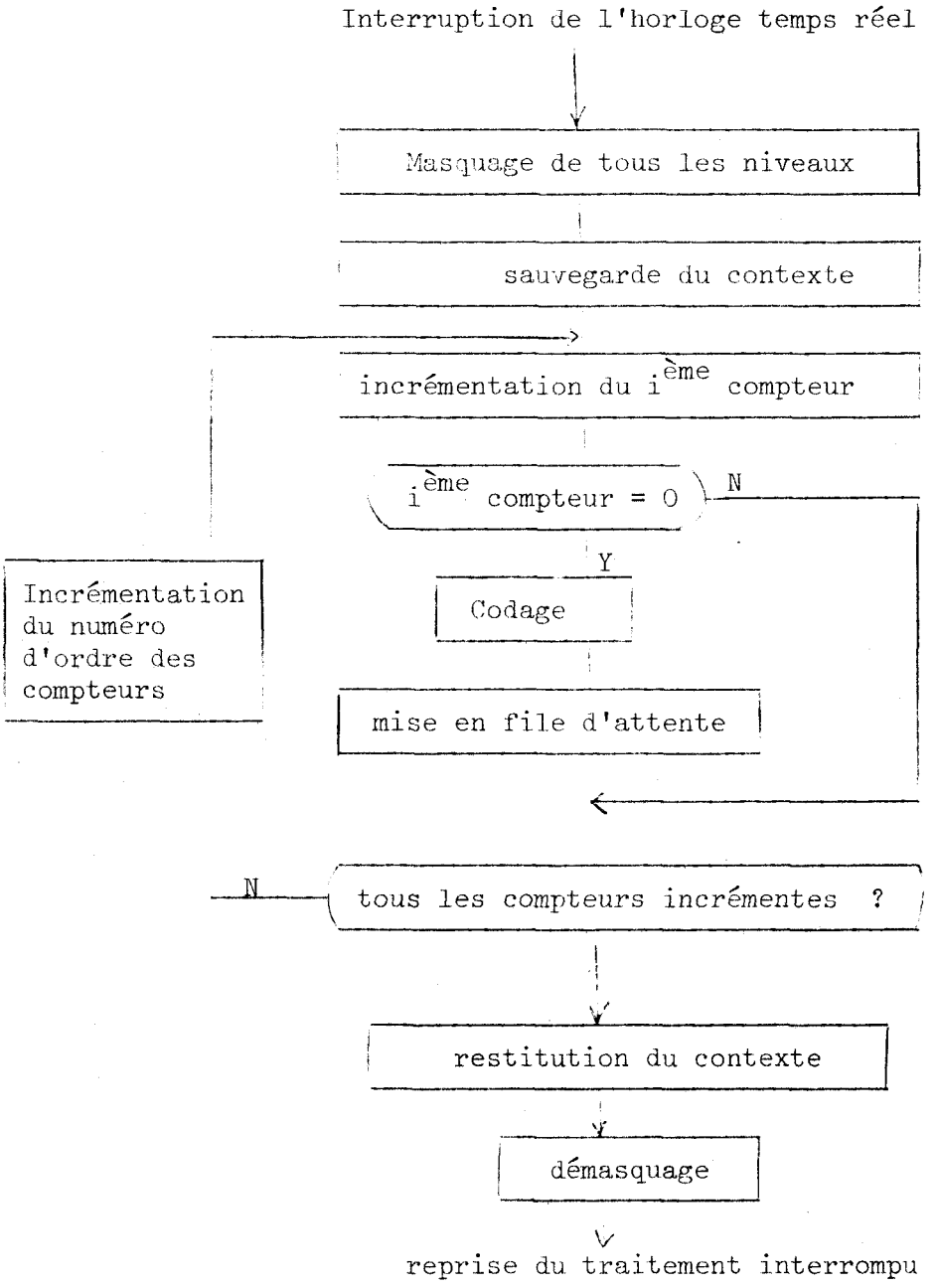


Figure II.2 - Traitement des interruptions de l'horloge

Chaque interruption d'horloge provoque donc le masquage de tous les niveaux puis la sauvegarde du contexte et l'incrémentation des compteurs associés à l'horloge.L'annulation de l'un d'eux entraine le codage de cet événement suivi du rangement en file d'attente du code. Lorsque tous les compteurs ont été incrémentés le traitement se termine par la restauration du contexte et le démasquage.

3.3 - Traitements spécifiques (Figure II.3)

Ces traitements permettent d'effectuer sur interruption des opérations non standardisées sous un niveau de priorité maximum.C'est à l'utilisateur de définir ces opérations. La séquence réalisée, consiste, aprés masquage de tous les niveaux et sauvegarde du contexte, à appeler un sous programme spécifique.

Un exemple usuel en contrôle de processus permet d'illustrer ce point de vue. Envisageons le cas d'une régulation en température d'un processus peu stable. En cas de défaillance de la commande ou de dépassement des limites de sécurité il faut pouvoir alerter immédiatement un programme spécifique de sécurité. Celui-ci est activé par un capteur de température.

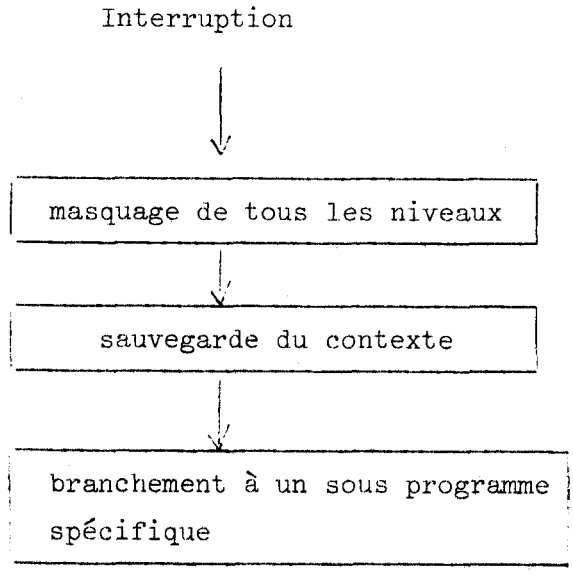


Figure II.3 - Traitements spécifiques

III - INSTRUCTIONS ET PROGRAMMES HYBRIDES.

III.1 - Formats des instructions

Ces formats ont été choisis en fonction des critères suivants :

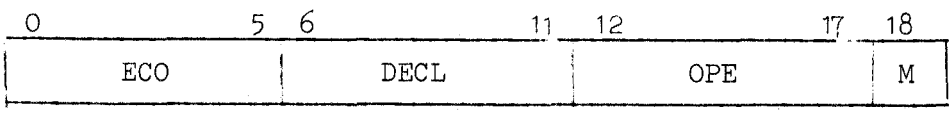
- Tout d'abord la taille d'un mot mémoire est de 19 bits.

- Il faut également tenir compte du volume et du nombre des informations nécessaires à la description complète d'une instruction hybride.

Notons enfin, qu'il faut utiliser un bit d'indicateur de mode séquentiel ou déclenché.
Les codes de déclenchement au nombre de 64 utilisent 6 bits. De même, le code opérations est limité à 6 bits codant 64 opérations différentes. Il reste alors 6 bits pour le codage des paramètres pour le même mot mémoire.

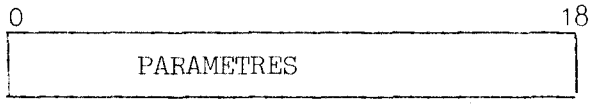
Le cadrage de ces informations a été choisi de façon à en faciliter l'écriture et la lecture dans la version sans compilateur du système . A chaque chiffre octal ne correspond qu'une seule donnée.

Format choisi :



OPE : code opération
DECL : code de déclenchement
M : mode de déclenchement
ECO : extension du code opération ou paramètre de type court

Les paramètres utilisant plus de six bits sont codés sur un mot complet



Une instruction peut donc éventuellement se composer de plusieurs mots mémoire.

III.2 - Programmes

Un programme hybride est constitué d'une suite de macro-instructions, elle-mêmes constituées d'une séquence d'instructions hybrides.
Un pointeur repère à tout moment l'instruction à prendre en compte.
Les macro-instructions sont exécutées lorsque les deux conditions suivantes sont réalisées.

1) Le pointeur adresse une macro-instruction dont l'exécution doit être déclenchée au synchronisme d'un événement.

2) Le niveau de priorité activé correspond au code de déclenchement précisé dans la première instruction de la macro.

Lorsque ces deux conditions sont satisfaites il y a exécution séquentielle des instructions en mode séquentiel de la macro-instruction.

Le pointeur est soit incrémenté aprés l'exécution de chaque instruction, soit déplacé par les instructions de rupture ou de branchement.

La macro-instruction est terminée lorsque le pointeur adresse une instruction en mode déclenché.

IV - PROGRAMME DE GESTION

IV.1 - Description

Le programme de gestion a accés par lecture aux codes des événements stockés dans la file d'attente. Il a pour mission de les exploiter. Du point de vue fonctionnel trois boucles réalisent les différentes fonctions du programme de gestion. L'organigramme de la figure IV.1 met en évidence l'imbrication de ces boucles.

La première traite les macro-instructions à l'intérieur d'un programme hybride. C'est dans cette boucle que sont exécutées les opérations hybrides.

Pour un événement donné la deuxième boucle analyse, pour chacun des quatre programmes, le déclenchement des macro-instructions. Elle procède par comparaison du code de l'événement stocké en file d'attente à celui de déclenchement codé dans l'instruction. Elle réalise ainsi la fonction de multiprocesseur.

La troisième boucle gère la file d'attente. Elle y puise les codes d'événements jusqu'à épuisement.

En fin d'exécution, la file d'attente étant vide, le programme de gestion enchaine sur l'exécution d'un programme de fond.

IV.2 - Interruption de traitement

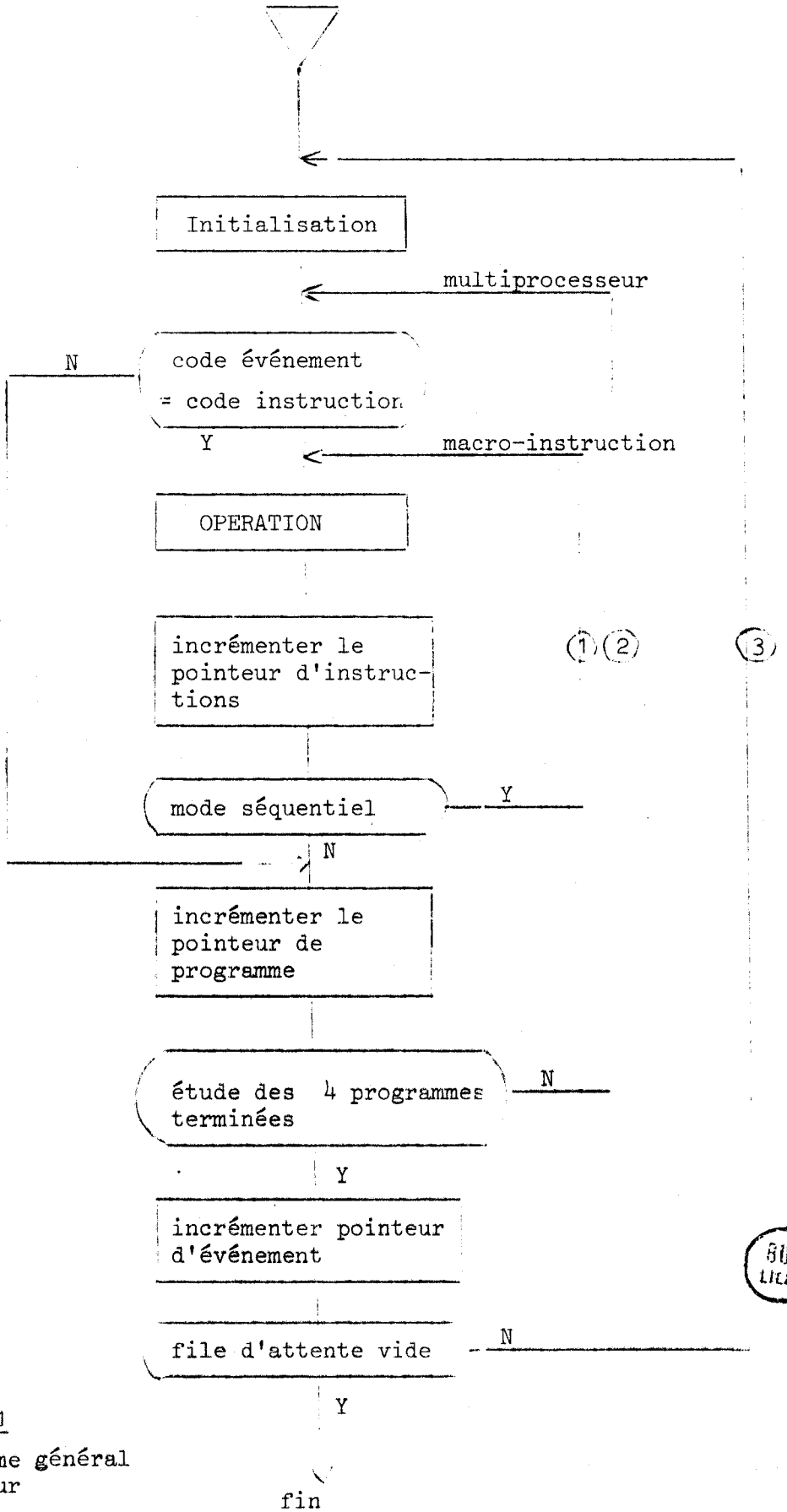
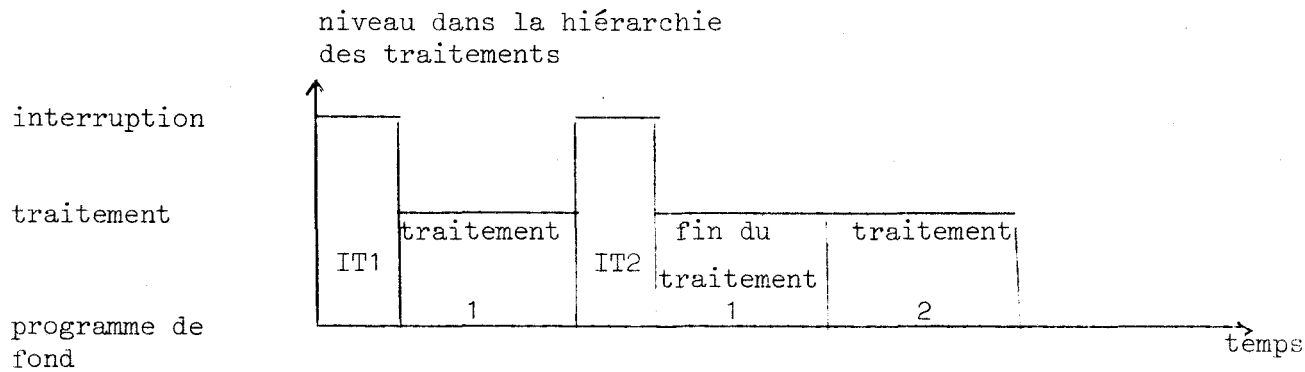


Figure IV.1

Organigramme général du moniteur

Un événement peut interrompre soit le programme de gestion, soit le programme de fond. Le premier cas ne pose pas de difficulté puisqu'avant d'atteindre le programme de fond, le système de gestion analyse la file d'attente et reste actif tant que celle-ci n'est pas vide.

L'enchainement des tâches peut dans ce cas être décrit schématiquement sur la figure suivante :



Le programme de fond par contre ne fait pas de scrutation de la file d'attente. Pour éviter que les événements qui l'interrompent restent inactifs, il est nécessaire de modifier légérement le traitement. Les modifications portent sur les accés au moniteur. La figure IV.2 en expose le principe. Lorsque le programme de fond est interrompu, son contexte est sauvegardé dans des mémoires spécifiques

En sortie le retour peut se faire soit vers le moniteur pour le traitement

d'une nouvelle interruption si la file d'attente n'est pas vide soit, au contraire, vers le programme de fond. Dans le deuxième cas il y a restitution préalable du contexte du programme de fond.

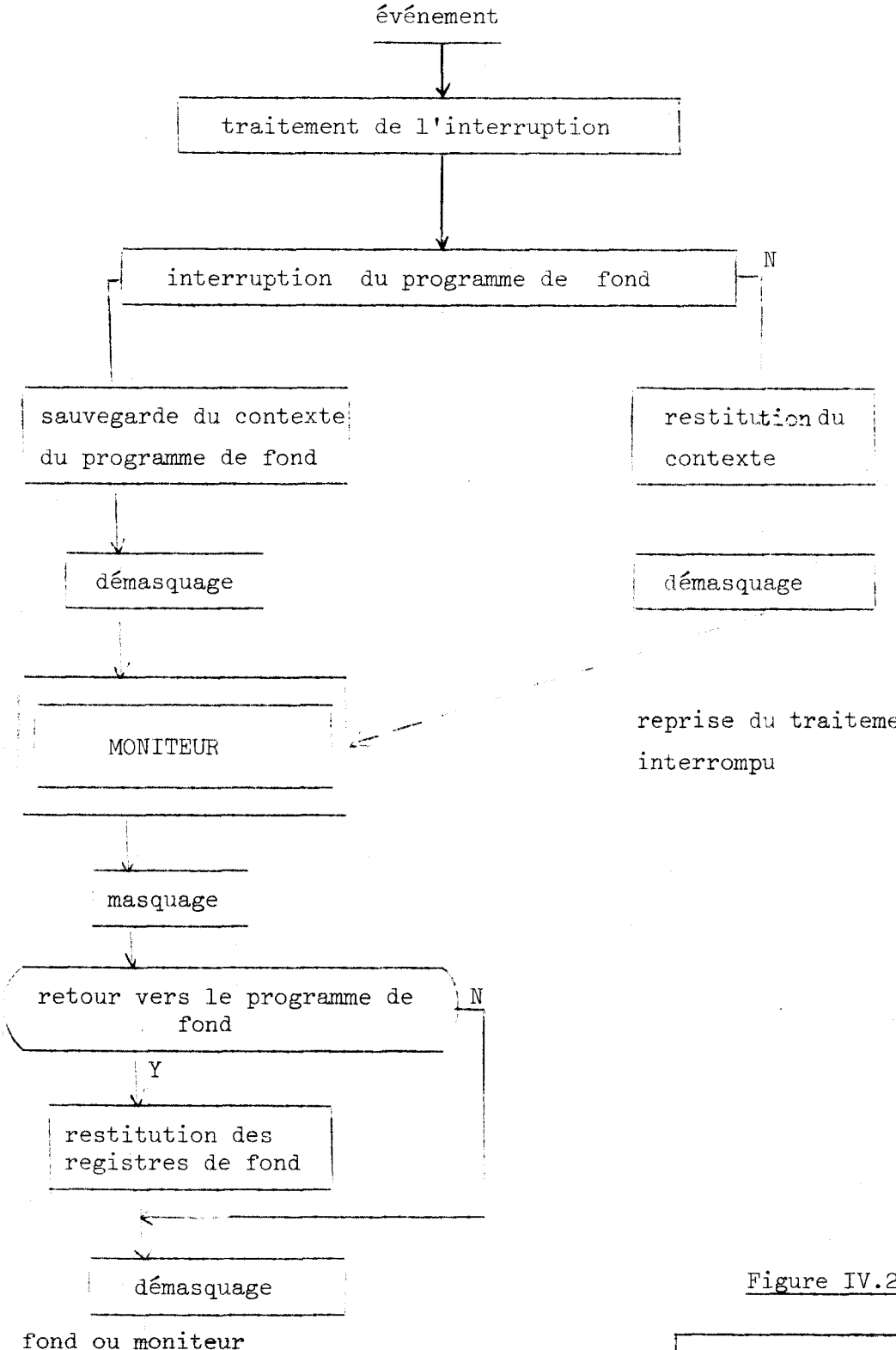


Figure IV.2

IV.3 - Opération

Lorsque le moniteur a autorisé le déclenchement d'une macro-instruction il assure l'exécution des opérations hybrides qui y sont décrites.
Celles-ci, au nombre de 64 maximum, sont regroupées dans une bibliothèque. Afin de pouvoir être facilement modifiée, intégrée ou supprimée, chacune forme un module indépendant dont l'accés se fait par une procédure banalisée.

Dans l'introduction nous les avions classées suivant leur nature en trois ensembles. Le premier correspondait aux opérations purement hybrides, le second était relatif à celles qui permettaient de contrôler les séquences de programmes et nous avions mis dans le troisième les opérations de gestion de fichier et de périphériques informatiques.

Nous proposons d'illustrer maintenant ces ensembles d'opérations.

IV.3.1 - Opérations de contrôle des éléments hybrides

Suivant le sens de l'échange ces opérations peuvent être divisées en deux classes.

Des commandes d'opérateurs hybrides, en particulier les intégrateurs, les interrupteurs électroniques ou les "track and store", et les sorties d'informations numériques ou analogiques, sont issues du calculateur numérique vers le calculateur analogique. Les programmes réalisant ces opérations sont extrêment courts. Ils ne comportent généralement qu'une dizaine d'instructions. Ils effectuent la reconnaissance des opérandes ou adresses, et la sortie numérique correspondante.

Dans l'autre sens, outre les entrées numériques dont les programmes de réalisation s'apparentent aux précédents, les opérations existantes sont relatives à la chaines de mesure. Elles effectuent les adressages et les échanges en mode simple, répétitif ou synchrone.

Les entrées et sorties analogiques s'effectuent souvent de façon répétitive. Pour en faciliter l'utilisation il est possible d'utiliser comme paramètre d'échange à la place d'une valeur ou d'une adresse pointant sur cette valeur, l'adresse d'une table. La gestion de cette table se fait alors automatiquement aprés chaque échange. Les informations étant stockées sur disque leur volume peut atteindre 32 K mots.

Les échanges calculateurs/disque se font par blocs en mode canal.

IV.3.2 - Opérations de contrôle de séquence

Ces opérations ont pour buts essentiels de réaliser des branchements conditionnels ou non et permettent en particulier la gestion d'opérations répétitives programmées sous forme de boucles.

Les ruptures sont obtenues par repositionnement des pointeurs des programmes.

Pour illustrer leurs utilisations possibles nous citerons deux exemples.

A l'intérieur d'un même programme elles permettent de réaliser des boucles. Elles sont alors équivalentes aux opérations DO du FORTRAN ou du PL.

Lordqu' un programme déplace le pointeur d'un autre programme il peut alors rendre, provisoirement, son exécution impossible. Ceci permet de réaliser l'activation ou l'inhibition des tâches.

Les conditions utilisent des paramètres initialisables et incrémentables. Les conditions sont remplies lors de leur annulation.

IV.3.3 - Gestion des périphériques informatiques

Les opérations portant sur ces périphériques sont les entrées de données et les listages de valeurs. Pour en faciliter l'exploitation, le codage de ces informations est fait en milli-volt pour les valeurs et en octal pour les variables booléennes ou logiques.

Les drivers de périphériques ainsi que la gestion des tables étant également utilisés par le compilateur nous avons regroupé leur présentation au chapitre II.6.

La liste des opérations et leur description détaillée est donnée en annexe technique.

CONCLUSION

Nous avons présenté dans ce chapitre les caractéristiques du moniteur hybride.
L'un des objectif essentiel de ce logiciel concerne la simplification de mise en oeuvre d'un système couplé analogique digital tout en conservant le caractère temps réel des traitements engagés sur les différentes parties du calculateur. Pour ces raisons, tout en assurant la gestion hiérarchisée des tâches, ce logiciel a été doté d'une fonction multiprocesseur. Dans le même sens, le concept d'événement a permis de résoudre les problèmes relatifs à la synchronisation des échanges d'informations entre les différentes parties du calculateur tout en minimisant, au niveau du processeur digital, les temps de traitement.

Du point de vue de l'utilisateur, la résolution d'un problème peut être envisagée selon trois techniques de programmation différentes.

Tout d'abord, si l'on suppose qu'une partie importante du processus étudié peut être représentée selon un modèle continu,on peut envisager une programmation en continu à partir d'un cablage analogique de l'équation d'état correspondante. Les variables d'état du processus sont alors directement accessibles et visualisables en continu permettant ainsi d'appréhender concrétement un phénomène. La mise au point du programme est relativement aisée, il faut cependant noter que le domaine de variation des variables est limité. Ce qui implique certaines contraintes d'échelle et de précision.
En second lieu, la programmation des opérateurs booléens peut être cablée sur le panneau logique du calculateur. Le nombre des opérateurs disponibles est limité, cependant cette possibilité de cablage permet trés souvent de simplifier considérablement la programmation du processeur digital.
Le troisième niveau est constitué par les programmes hybrides dont l'exécution est assurée par le moniteur. Les différentes opérations réalisées font appel

à une bibliothèque de sous programmes facilement modifiables ou adaptables au matériel utilisé. Elles sont décrites par des instructions indépendantes du langage utilisé sur le calculateur numérique. Une connaissance de base de la calculatrice analogique suffit donc à l'exploitation de l'ensemble analogique digital.

CHAPITRE II

LE COMPILATEUR

INTRODUCTION

La structure du moniteur hybride a été définie de manière à réaliser simplement les fonctions essentielles du calcul hybride de type II. Ces opérations de base, sont codées numériquement sur un format imposé. C'est pourquoi nous nous proposons de présenter dans ce chapitre les différents moyens qui permettent d'une part de faciliter l'écriture des programmes et d'autre part d'en simplifier la mise en oeuvre.

Plusieurs points de vue doivent être ainsi envisagés.
Notons tout d'abord que les instructions sont codées en octal et malgré le soin apporté à la définition du codage des informations dans les formats choisis, ce type de codage est difficile à exploiter, tant à l'écriture, qu'à la lecture.

D'autre part au niveau de la programmation hybride certaines données dont délicates à manipuler. En effet la fonction multiprocesseur est décrite par plusieurs programmes hybrides distincts qu'il est nécessaire de délimiter clairement. De plus certains paramètres peuvent être transmis de l'un à l'autre.
Cette transmission nécessite de repérer les paramètres par leur adresse et donc d'avoir accés à des informations d'un type purement informatique.
Le même genre de problème se pose à l'intérieur d'un programme hybride où existent des ruptures de séquence. Les points de branchement peuvent être repérés soit par un déplacement relatif par rapport au point de rupture soit par l'adresse du point de branchement. Le volume de mot nécessaire à la description d'une instruction varie d'une instruction à l'autre et par conséquent le calcul des déplacements relatifs est délicat.
Le branchement sur adresse présente les mêmes inconvénients que ceux présentés par la transmission de paramètre à savoir : la nécessité d'une connaissance informatique du calculateur numérique.

Afin de pallier ces inconvénients nous avons été amenés à introduire

une écriture symbolique et un compilateur associé qui traduit les codes symboliques en codes binaires assimilables par le moniteur.
Nous adopterons pour ce chapitre une présentation en trois parties.
De manière à structurer l'ensemble des programmes et en particulier à faciliter le repérage des paramètres communs nous proposons dans une première partie d'introduire le concept de directive. Dans le même sens, l'utilisation des étiquettes évite de préciser de manière absolue les adresses des opérandes ou des points de branchement.

Dans un second volet nous définirons le symbolisme utilisé dans la représentation des opérations hybrides proprement dite.

En dernier lieu, nous serons naturellement amenés à décrire le fonctionnement du compilateur, et plus précisément le principe du codage binaire des instructions symboliques.

I - ORGANISATION GENERALE DU COMPILATEUR

La fonction multiprocesseur, dont nous avons défini l'utilité particulière au premier chapitre, nous conduit à distinguer deux niveaux dans l'analyse des sources d'informations du compilateur. Dans une première étape nous présenterons la structure d'un ensemble de programmes hybrides dont l'exécution doit être parallèle ou quasi simultanée. Nous étudierons ensuite l'organisation interne de ces programmes.

I.1 - Structure des programmes à exécution parallèle.

Les deux contraintes relatives à la fois à l'indépendance et à la simultanéité de certains traitements nous ont amenés à réaliser cette fonction "multiprocesseur" par l'écriture de plusieurs programmes hybrides. Dans ce sens chaque programme correspond à un traitement particulier et peut être indépendant des autres.

Afin d'en simplifier l'écriture nous avons utilisés des délimiteurs de début et de fin de programme.
L'indépendance des traitements ne concerne en fait que l'exécution ; un programme hybride aura donc la possibilité d'utiliser des paramètres définis dans d'autres programmes ou encore de transmettre des informations vers les sous programmes de calcul.

Afin de s'affranchir des contraintes informatiques ces transmissions se feront grâce à des pointeurs de données repérés par des étiquettes.

Nous définirons globalement les délimiteurs de programmes et l'étiquetage des pointeurs de données comme étant des pseudo-instructions. Ces pseudo-instructions ne génèrent en fait aucun code binaire ; elles sont utilisées comme indicateur par le compilateur.

Elles sont repérées par le symbole * et sont définies par les syntaxes suivantes :

* DEB et * FIN sont les délimiteurs
* EXT la déclaration de pointeur.

Des commentaires peuvent aussi être introduits en tout point d'un programme par la pseudo-instruction *COM.

I.2 - Structure interne d'un programme

L'exécution d'un programme hybride s'effectue généralement séquentiellement. Cependant il est trés utile de définir au niveau des macro-instructions, des instructions de rupture de séquence conditionnelles ou inconditionnelles. A priori les branchements vers une macro-instruction peuvent se faire soit vers une adresse, soit par déplacement relatif. Dans la mesure où les instructions hybrides utilisent des formats et des volumes de mots différents le calcul des déplacements relatifs est délicat. Pour cette raison le branchement sur adresse a donc été retenu. Afin d'en simplifier l'utilisation les points de branchements sont repérés par des étiquettes. Le compilateurs devra donc associer les adresses absolues à ces étiquettes.

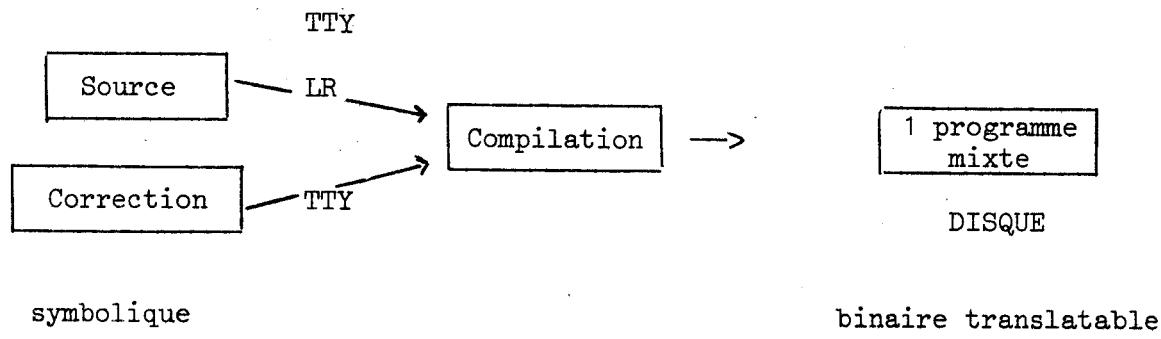
Au cours de la compilation, il peut arriver que certaines étiquettes définies à postériori ne puissent être précisées au niveau de l'instruction analysée.

Dans ce sens l'utilisation d'un disque permet de générer un fichier partiellement compilé pour lequel certains symboles n'ont pu être chargés au niveau des instructions. Dans ces conditions il est nécessaire d'effectuer un second passage de ce fichier sous le contrôle du compilateur pour résoudre ces symboles. Afin de limiter le nombre d'échanges avec le disque le deuxième passage est associé au chargement en mémoire vive des programmes opérationnels.

Par définition nous dirons que le fichier intermédiaire est écrit en binaire translatable et que les programmes opérationnels le sont en code binaire.

Le traitement en deux phases peut être illustré par le schéma de la figure I.2.

1° Passage - Phase de compilateur



2° Passage - Phase d'assemblage - chargement

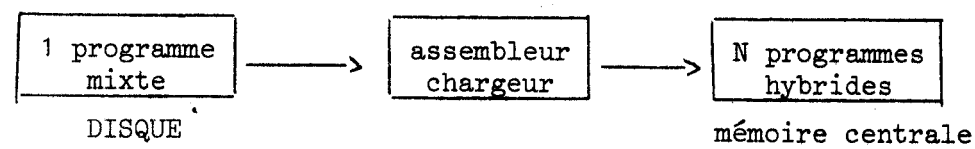


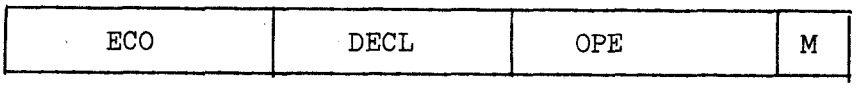
Figure I.2 - Phase de la compilation

I.3 - Code binaire

Le compilateur a pour objet essentiel la génération de codes binaires selon un format assimilable par le moniteur hybride. L'instruction est codée sur un ensemble de mots. Le premier définit d'une part le code opération sur six bits associé à un paramètre du type opérande immédiat sur six bits, d'autre part le mode de déclenchement et dans l'affirmation d'un déclenchement le code associé. Les autres mots sont optionnels.ils servent à définir les autres paramètres lorsqu'ils existent ou lorsqu'ils nécessitent plus de six bits.

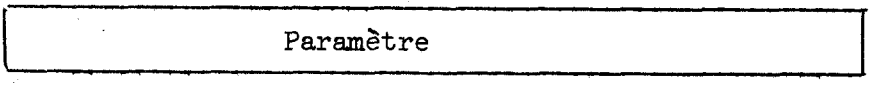
Les formats des mots ainsi obtenus sont illustrés par le schéma suivant :

Instruction



ECO : paramètre immédiat de type court
DECL : code de déclenchement
OPE : code de l'opération
M : mode de déclenchement

Paramètre



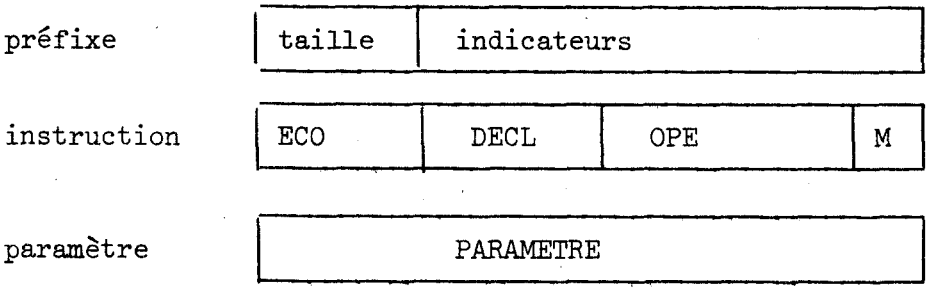
Paramètre supplémentaire ou de type long

I.4 - Code binaire translatable : préfixe

Le binaire translatable reprend les formats précédents. Les paramètres repérés par une étiquette sont définis par le code de l'étiquette.

Afin de préciser si un paramètre correspond à une valeur réelle ou à une étiquette il est donc nécessaire d'utiliser des indicateurs de présence de symboles non résolus. A cette fin chaque groupe de mots représentant une instruction est précédé par un mot supplémentaire contenant ces indicateurs.
Par définition nous l'appélerons préfixe. De plus pour que l'assembleur-chargeur puisse calculer les adresses d'implantation de chaque programmes hybride le préfixe contient une information relative à la taille de l'instruction.

Pour le traitement d'une instruction nous disposons donc des éléments suivants :



Il est à noter que du fait de la présence d'un préfixe aucun bit d'un paramètre n'est réservé à la fonction indicateur. Par conséquent tous les paramètres peuvent prendre toutes valeurs cadrées sur 19 bits.

II - ECRITURE SYMBOLIQUE DES PROGRAMMES

Le choix de la plupart des instructions hybrides est déduit du langage utilisé dans la programmation des fonctions logiques et analogiques.
Les opérandes immédiats des opérations correspondent aux adresses effectives des opérateurs cablés sur les panneaux de programmations logique et analo-

gique.

Les ruptures de séquences et les transmissions de paramètres utilisent systématiquement des étiquettes.

Enfin les opérations de gestion de fichiers et de périphériques ont été définies de façon à ce qu'aucune connaissance informatique préalable ne soit nécessaire à leur utilisation.

II.1 - Syntaxe de instructions hybrides

Une instruction se présente comme une phrase constituée :

- d'une part d'un en-tête comprenant éventuellement une étiquette et un code de déclenchement

- d'autre part d'un corps avec le code de l'opération et ses paramètres.

Elle se termine par un retour chariot. Le mode est défini implicitement par la présence ou l'absence du code de déclenchement.
Le code symbolique d'écriture d'une instruction hybride est donc défini de la manière suivante

⌈(ETI) ⌈DEC⌉ : ⌉OPE ⌈PARAM⌉ ,⌈PARAM⌉ RC

ETI : étiquette constituée de trois caractères alphanumériques (ET1), (010)...

DEC : code de déclenchement de l'instruction : V0, E0, H1...

OPE : code opération formé de 3 caractères : MIC, EAS,...

PARAM : codes des paramètres. Leur nombre dépend de l'opération. Il ne peut excéder quatre.

⌈OU⌉ peut être omis dans l'écriture d'une instruction.

II.2 - Symbolisme des opérations hybrides

Nous proposons d'illustrer sur un exemple de programme hybride le symbolisme utilisé. La liste des opérations est donnée en annexe n°1. L'exemple proposé concerne l'exploration paramétrique à partir d'une gestion numérique, des solutions d'un système différentiel. Le programme réalisé permet de gérer l'évolution des paramètres selon une technique itérative simple. Ces derniers, au nombre de N sont supposés enregistrés dans une table.

La séquence réalisée consiste tout d'abord à initialiser ce processus en fonction de la valeur d'un paramètre, puis à résoudre son évolution jusqu'à ce que la variable de sortie atteigne une valeur limite. La séquence sera répétée N fois. La figure II.2 explicite les différentes phases et, le listage II.2 b donne l'écriture du programme hybride associé.

La première instruction, étiquetée (010) est déclenchée par un appel opérateur. Elle a pour objet d'initialiser le paramètre de décompte du nombre de boucles á effectuer.

Les instructions suivantes réalisent la boucle proprement dites.

Tout d'abord les intégrateurs commandés en 2A1 sont mis en conditions initiales avec la valeur prise dans la table de paramètre dont le pointeur est automatiquement incrémenté. Une temporisation assurant la prise effective des conditions initiales charge à une valeur négative le paramètre

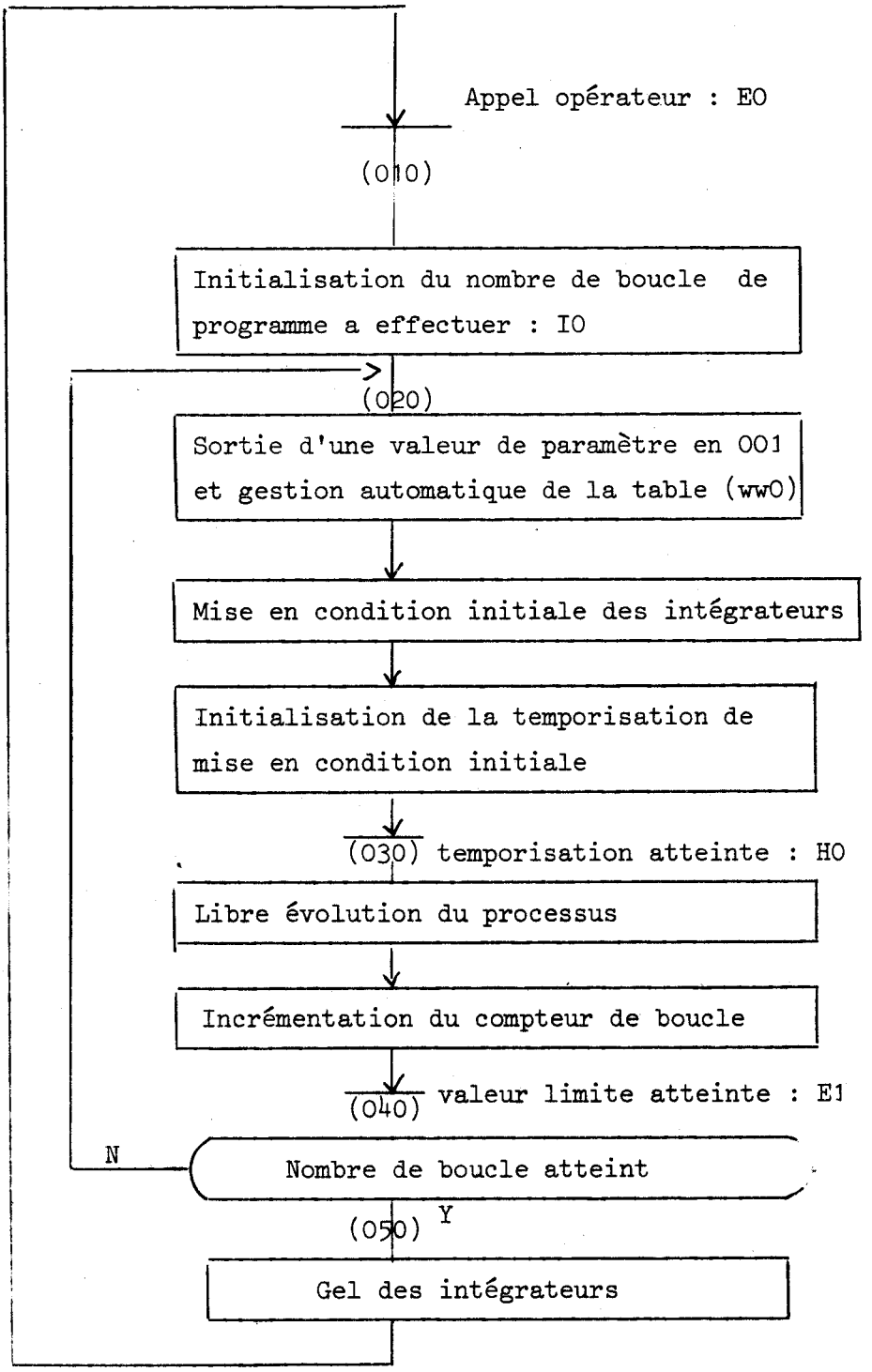


Figure II.2 a

Organigramme de principe

```
* COM  :  TRACE  D'UN  RESEAU  DE  COURBES

* DEB      DEBUT  DU  PROGRAMME

(010)E0   :  I0 = - 10          <initialisation du nombre de boucles
                                <de programme
(020)     :  SOA, 0D1, (WW0)    <sortie d'une valeur de paramètre
             MIC, 2A1           <initialisation du processus
             PH0 = - 10         <temporisation
(030)H0   :  MOP, 2A1           <lancement de la résolution
             INC, I0            <décompte du nombre de boucles
(040)E1   :  RPC, I0 ,(010),(050) <bouclage
(050)     :  GEL, 2A1           <gel des intégrateurs
             RPI,(010)          <retour au début

* FIN  <  FIN  DU  PROGRAMME
```

Figure II.2 b

Programme hybride

PHO associé à un des compteur de l'horloge temps réel. La temporisation est suivie de la résolution des équations différentielles. Le compteur de boucle de programme est alors incrémenté.

Lorsque la variable de sortie atteint une valeur de référence l'événement E1 est positionné. Celui-ci déclenche soit une nouvelle boucle, si le nombre de boucles maximum n'est pas atteint, soit dans le cas contraire, l'arrêt du processus par mise en gel des intégrateurs.

Au cours de l'évolution du processus la variable de sortie, ainsi que d'autres informations présentes, sur la partie cablée, peuvent être visualisées ou enregistrées sur oscilloscope ou table traçante. Dans le deuxième cas, il convient de prévoir l'ordre de relévement de la plume pour éviter le tracé du retour de plume vers l'origine.

Ce programme peut être décomposé en plusieurs macro-instructions. La première est lancée sur ordre (étiquettes (010) et (020)).
Le déclenchement de la deuxième correspond à la fin de la temporisation. Elle lance la résolution et gère le nombre de boucles à éffectuer.

La troisème macro-instruction est double. Déclenchée par l'événement E1, elle débute par une rupture conditionnelle. Suivant la valeur de la condition c'est-à-dire du nombre de boucles, elle se termine soit par l'initialisation d'une nouvelle boucle en (020) soit par la fin du programme en (050).

Cet exemple fait apparaitre clairement le parallélisme entre l'organigramme de principe et le programme hybride. Le fait qu'il n'y ait pas d'expansion lors du codage autorise une écriture du symbolisme trés aisée. Pratiquement les seules erreurs possibles sont les erreurs de syntaxe. Nous mettrons en évidence la façon dont ces erreurs sont systématiquement détectées ainsi que les possibilités de les corriger dans le paragraphe suivant.

III - FONCTIONNEMENT DU COMPILATEUR

Nous avons décrit antérieurement les fonctions essentielles du compilateur. Nous proposons maintenant d'étudier le principe de génération du code binaire translatable et de détection des erreurs de syntaxe et de programmation.

Rappelons qu'une phrase est définie par un code symbolique que nous subdiviserons en plusieurs champs.

- Etiquette
- Variable de déclenchement
- Code opération avec ses paramètres

⌈ (ETI) ⌈DECL :⌉ OPE ⌈PARAM1⌉ , PARAM⌉ RC

L'inscription de commentaire se fait en complétant la phrase par le symbole < suivi d'un texte . Les symboles compris < et le retour chariot sont alors ignorés du compilateur.

Afin de simplifier le repérage des macro-instructions, nous proposons d'imposer leur étiquettage systématique. Notons toutefois qu'une instruction contenue dans une macro-instruction ne doit pas être, à contrario, automatiquement étiquettée.

A l'exception des pseudo-instructions (§ III.2) nous distinguerons pour une phrase trois types d'écriture, ce qu' illustre l'exemple suivant :

```
(010) EO : RPI, (040)
(020)    : MOP , 2A1
           SOA , 0D1 , + 4000
```

De manière à préciser le fonctionnement du compilateur, nous proposons de présenter tout d'abord le principe de reconnaissance des champs puis les traitements qui en découlent.

III.1 - Reconnaissance de la nature et des champs d'une phrase.

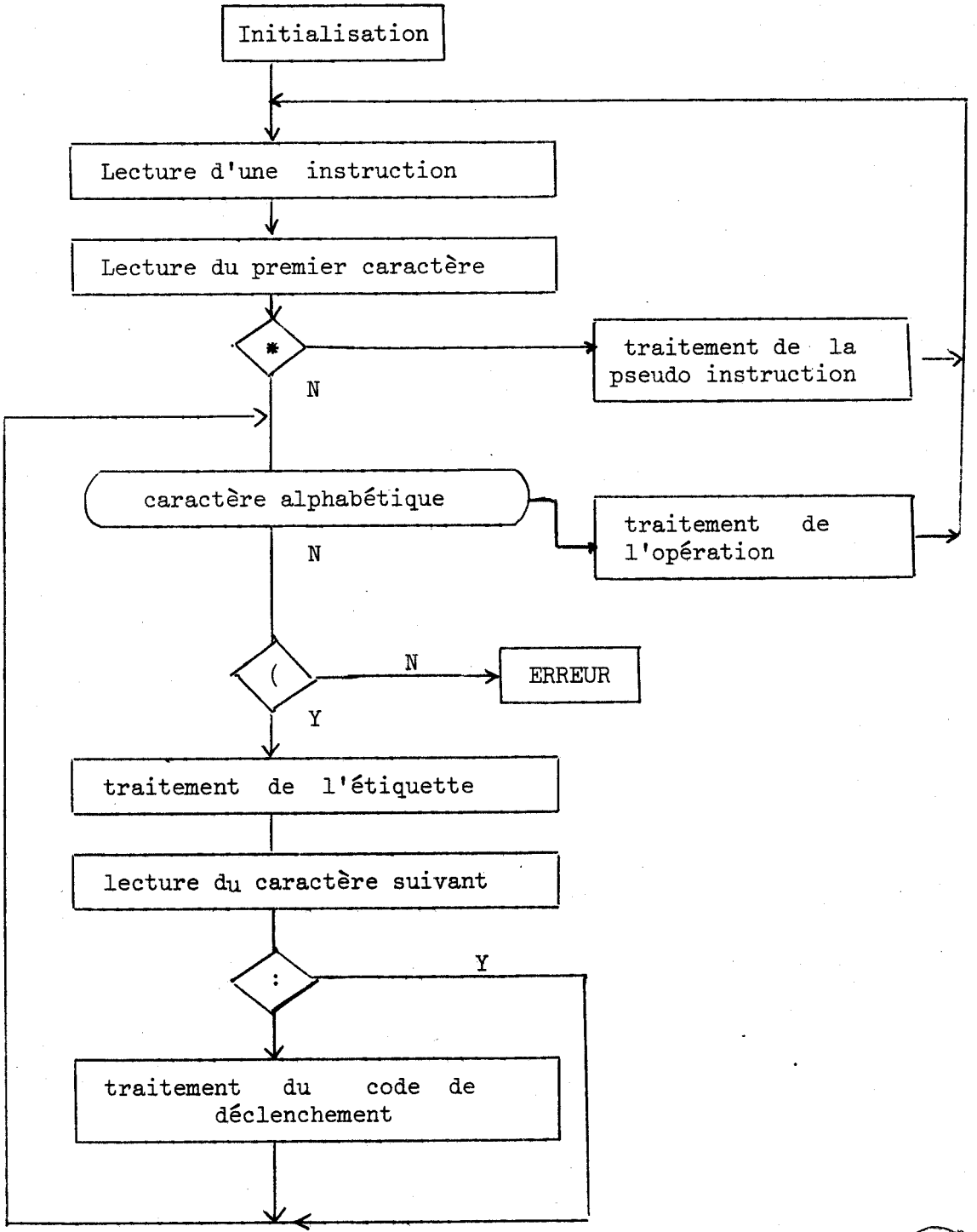
Le compilateur traite les programmes hybrides phrase par phrase. Plutôt que d'entreprendre une analyse séquentielle, il commence par enregistrer une phrase complète. Cet enregistrement se fait par lecture sur un fichier d'entrée précisé par l'utilisateur et qui peut être soit le télétype soit le lecteur rapide.

Le traitement d'une phrase est décrit sur la figure III.1. La première opération effectuée concerne la reconnaissance du premier caractère celui-ci permet de définir d'abord la nature de la phrase. En effet une étoile indique qu'il s'agit d'une pseudo-instruction ; Celle-ci sera traitée immédiatement. Les autres symboles définissent par défaut les instructions.

Dans le cas d'une instruction, la nature du premier caractère permet de préciser le type du premier champ d'une instruction.

- Un caractère alphabétique indique un code opération et implique un traitement immédiat.
- Une parenthèse indique une en-tête et plus particulièrement une étiquette.
- Les autres symboles éventuels sont considérés comme étant des erreurs de syntaxe. Dans ce cas le compilateur efface l'enregistrement ,sort un message d'erreur sur le télétype et attend la frappe d'une nouvelle phrase corrigée.

S'il s'agit d'une étiquette le traitement de la phrase se poursuit par le calcul de l'adresse absolue associée et sa mémorisation dans la table des étiquettes puis par la reconnaissance du champ suivant.



Organigramme de reconnaissance de la nature et des différents champs d'une instruction

Figure III.1

Celui-ci correspond alors au champ instruction si le caractère à prendre alors en considération est délimiteur ":". Dans le cas contraire il correspond à celui des variables de déclenchement.

La fin d'une phrase coincide nécessairement avec la fin du traitement d'une instruction ou d'une pseudo-instruction, le délimiteur de fin de phrase correspond au code retour chariot. L'enchainement automatique en séquence conduit à analyser la phrase suivante. Le compilateur arrète son analyse lorsqu'il rencontre la pseudo-instruction * FIN.

III.2 - Traitement des pseudo-instructions

Les pseudo-instructions ne sont utilisées que par le compilateur et n'entrainent en conséquence la génération d'aucun code binaire translatable.

La figure III.2 décrit le schéma d'analyse d'une pseudo-instruction. A la suite de la reconnaissance du code le compilateur fait appel aux sous programmes relatifs au traitement de la pseudo-instruction.

A la rencontre de * DEB le compteur de mots binaires générés est additionné à l'adresse de chargement du programme précédent. Le résultat obtenu est l'adresse de début du programme introduit par le * DEB. Le compteur est ensuite remis à zéro et les divers paramètres de travail sont réinitialisés.

La pseudo-instruction * EXT charge dans la table des étiquettes l'adresse absolue associée à un code. Ces éléments sont donnés syntaxiquement de la façon suivante :

* EXT (ETI) ADRE, où ETI est le nom symbolique de l'étiquette et ADRE l'adresse que l'utilisateur désire lui associer.

Les programmes se terminent par * FIN.
Dans ce cas la phase compilation est terminée. La phase d'assemblage-chargement lui succède.

Les commentaires peuvent suivre les codes * DEB et * FIN ou encore être introduits en tout point d'un programme par * COM.
Dans tous les cas ils sont ignorés du compilateur.

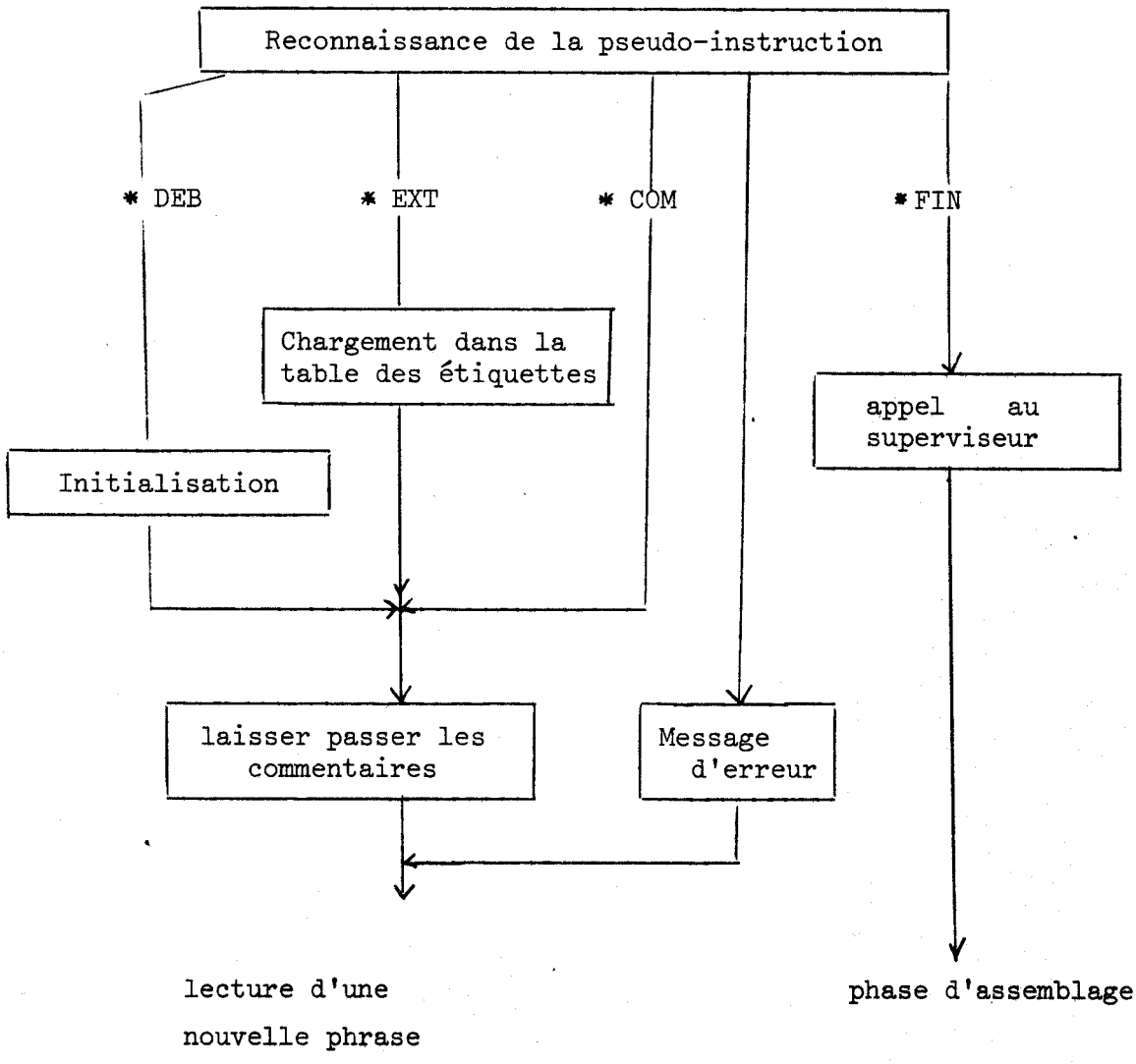


Figure III.2

Traitement des pseudo-instructions

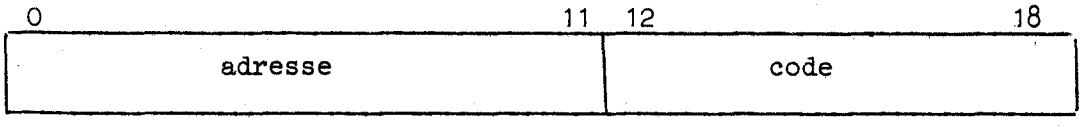
III.3 - Traitement des étiquettes

Les étiquettes ont été introduites pour faciliter l'écriture des programmes hybrides. Elles permettent d'une part de repérer les instructions, d'autre part d'associer un code symbolique à certains paramètres.

Dans un but de synthèse nous proposons d'étudier également, dans ce paragraphe les différents traitements associés aux étiquettes. Afin de pouvoir distinguer une étiquette d'un paramètre nous avons choisi de répérer le code étiquettes en le délimitant par deux parenthèses. Les symboles compris entre les parenthèses sont alphanumériques et strictement limités à trois caractères.

Pour des raisons d'encombrement mémoire le volume maximum d'étiquettes a été limité à 128. Le code et l'adresse associés ont été regroupé sur le même mot.

Une seule table de 128 mots suffit donc à définir la correspondance entre ces étiquettes symboliques et les adresses associées. L'utilisation d'un bloc de 4 K mots implique un adressage immédiat sur 12 bits. Il en reste donc 7 pour le code étiquette. Le format utilisé pour stocker les informations relatives aux étiquettes est donc :



Le nombre limité de bits réservés au code rend nécessaire un précodage des symboles. Celui-ci est réalisé par troncature de leur code ASCII. Pour le premier caractère n'est conservé que le bit de poids faible et pour les deux suivant ne sont gardés que les 3 derniers bits.

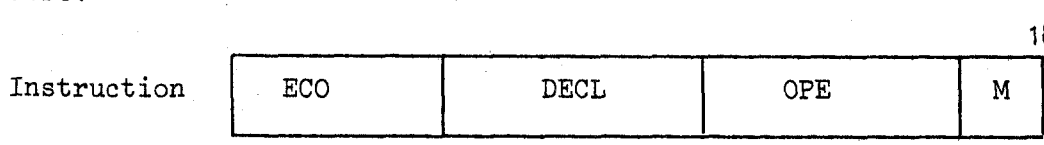
Deux étiquettes peuvent donc aboutir au même codage. Dans ce cas une erreur est détectée et la frappe d'une nouvelle étiquette est demandée.

L'utilisation des codes (001) à (177) en octal ou des caractères alphabétiques A,B, C, D, E, F, G, H seuls permet d'éviter cet inconvénient. L'adresse associée à un code peut être déclarée par la pseudo-instruction * EXT soit calculée. Le calcul a lieu à la compilation lors de la rencontre d'une étiquette utilisée comme repère d'instruction. Il consiste à additionner l'adresse d'implantation du programme à la valeur du compteur de mots binaires générés et à ranger le résultat dans la table des étiquettes.

Les étiquettes utilisées comme paramètres d'opération n'entrainent aucune modification dans la table. En effet, elles ne peuvent en aucun cas être considérées comme déclaration d'étiquette. Nous verrons dans le paragraphe relatif au champ des opérations comment elles sont traitées.

III.4 - Traitement du code de déclenchement.

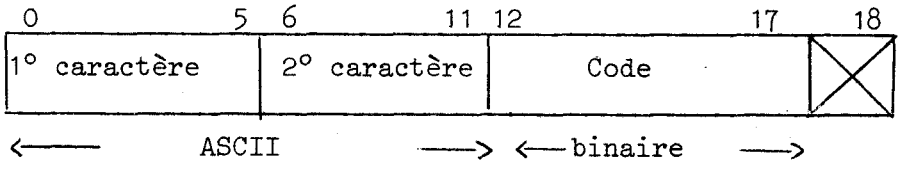
Ce code désigne l'événement qui doit déclencher l'instruction. Sa présence indique que le mode est du type déclenché et le bit 18 du code instruction est alors positionné à 1. A contrario son absence signifie par défaut que le mode est du type séquentiel et le bit 18 est remis à zéro.



M : Mode : M = 1 mode déclenché, M = 0 mode séquentiel
DECL : code de déclenchement.

Dans le cas du mode déclenché il convient de vérifier la validité du code symbolique déclaré et de lui associer un code binaire. Le symbolisme adopté est constitué de deux caractères alphanumériques. En général le premier indique le type de l'événement (E pour appel externe - V pour voie de comptage). Le deuxième caractère précise pour un type donné le numéro d'ordre

de l'événement.
Une table décrit l'ensemble des codes symboliques autorisés et leur associe un code binaire. L'utilisation de deux caractères alphanumériques a permis de n'utiliser qu'un mot pour le codage d'un événement. Le format d'un mot de la table des codes d'événement est le suivant :



Le compilateur recherche par balayage de cette table le code symbolique précisé dans l'instruction courante analysée puis range le code objet associé dans le mot binaire correspondant à l'instruction codée. En cas d'erreur de syntaxe, la demande de correction est précédée d'un message sur télétype.

III.5 - Traitement des opérations.

L'organigramme de la figure III.5 a illustre le traitement des opérations. L'analyse d'une opération s'éffectue en plusieurs phases. Il faut en effet coder à la fois le type de l'opération et les différents paramètres qui lui sont associés. Le code opération est généralement constitué de trois caractères alphanumériques. Deux exceptions doivent cependant être signalées. En effet, pour avoir une écriture plus explicite l'initialisation des paramètres utilise le signe égal et dans le même esprit les opérations originales d'un problème peuvent être écrites directement en octal.

Le traitement des opérations débute par une phase de reconnaissance du type de l'opération. Lorsque l'analyse d'une phrase conduit à reconnaître

comme premier caractère la lettre P, le compilateur est directement orienté vers le traitement de l'opération d'initialisation de paramètre. Tous les autres caractères correspondent au codage des opérations et le symbole ∅ caractérise les instructions écrites en octale.

Le schéma de la figure III.5 b résume cette première phase du traitement.

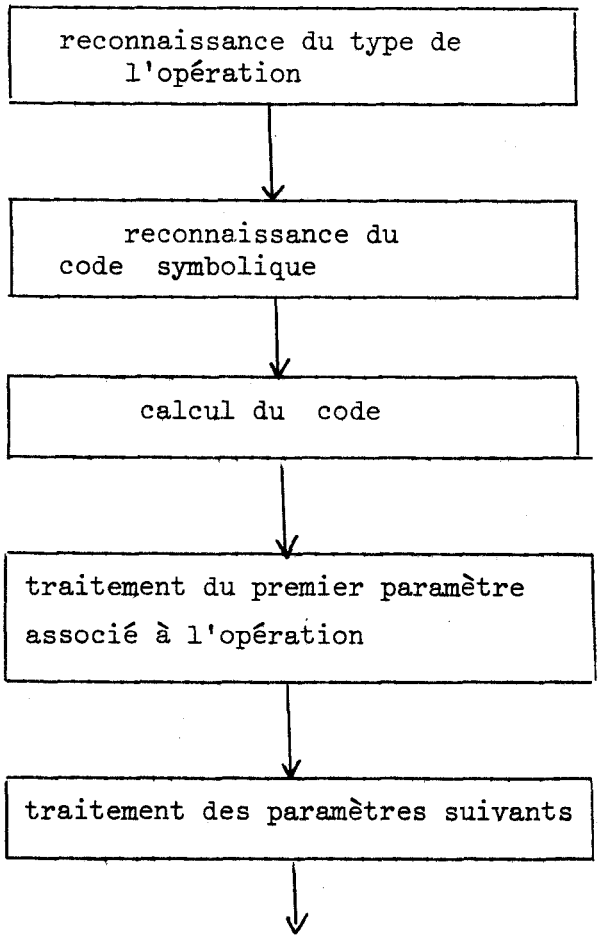


Figure III.5 a

Principe de traitement d'une opération

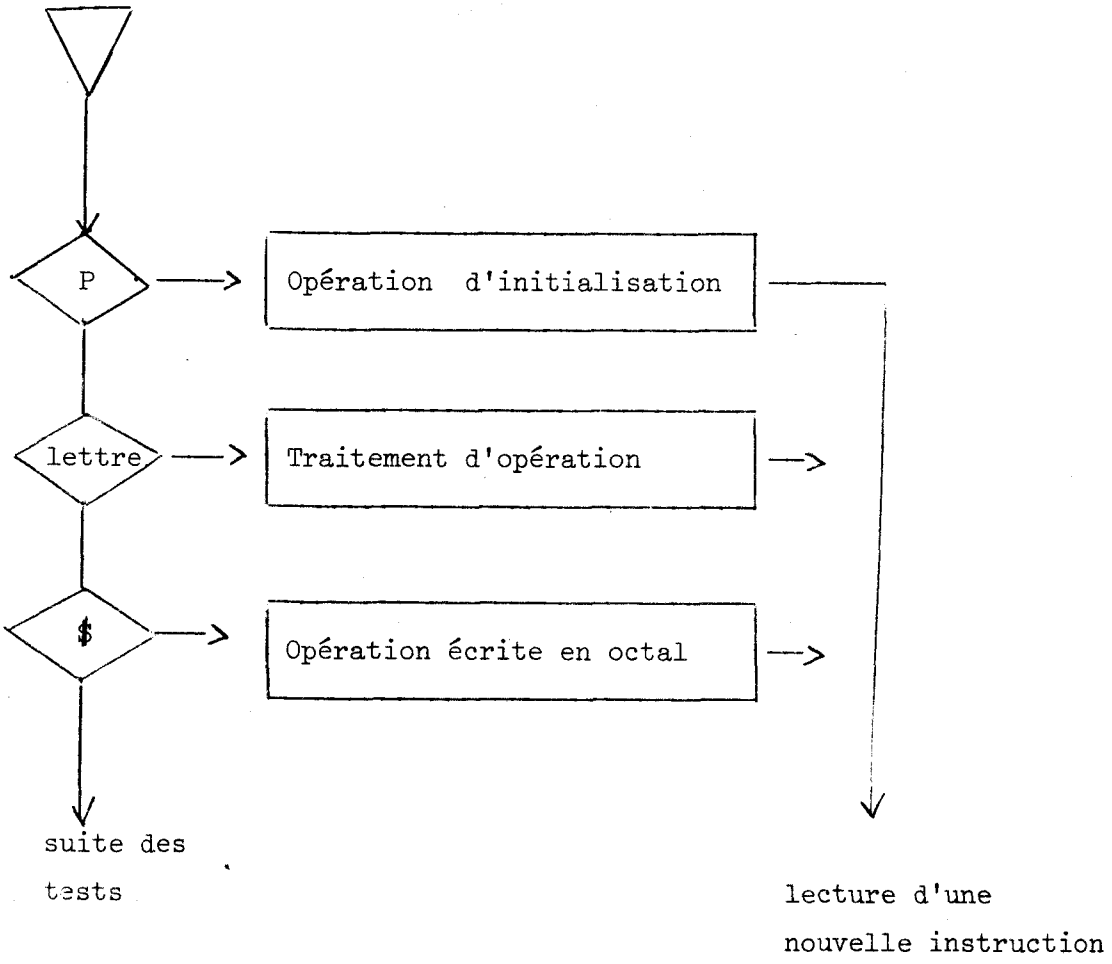


Figure III.5 b

Reconnaissance du 1° caractère
et traitements associés

III.5.1 - Paramètres hybrides et opération d'initialisation

Le moniteur hybride utilise deux types de paramètres :

- Tout d'abord les paramètres de comptage associés aux voies de comptage où à l'horloge temps réel. Leur but essentiel est de définir simplement des temporisations,

- En second lieu les paramètres incrémentables par instruction. Ces derniers sont généralement affectés à la gestion des boucles de programme.

La condition relative au passage par zéro d'un paramètre du premier type génère un événement. Pour ceux du deuxième type, elle est utilisée systématiquement dans l'instruction de rupture conditionnelle.

Trois caractères sont nécessaires au codage symbolique d'un paramètre. Le premier caractère "P" indique qu'il s'agit d'un opérande du type paramètre. Le second définit sa nature.

V pour les voies de comptage

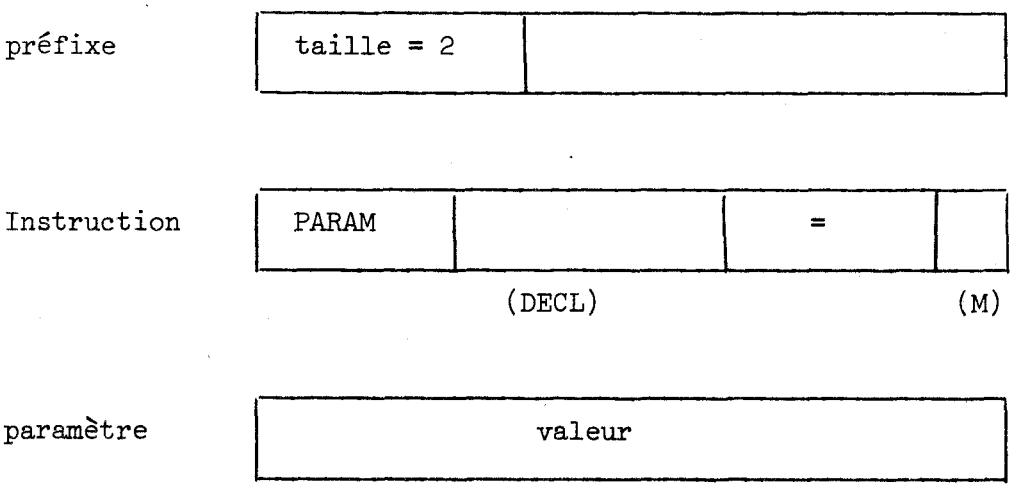
H pour les compteurs associés à l'horloge temps réel

I et J pour les paramètres du second type.

Le dernier caractère donne un numéro d'ordre.
Par exemple : PV0, PH2, PI1.....

Le branchement au sous programme qui traite l'initialisation du paramètre se fait aprés reconnaissance de la lettre P. Il suffit donc de vérifier la validité des deux derniers caractères. Une table de référence permet d'effectuer ce test. Elle donne d'une part les codes symboliques autorisés et d'autre part le code binaire associé, cadré sur six bits.

Lorsque le paramètre a été reconnu le compilateur charge successivement la taille de l'opération dans le préfixe, le code binaire de l'opération et celui du paramètre dans le mot réservé à l'instruction. Puis aprés avoir vérifié que la valeur d'initialisation était écrite avec uniquement des caractères numériques il les range dans un mot supplémentaire. La configuration obtenue est alors la suivante :



Nous verrons plus loin le traitement relatif aux paramètres utilisés non pas par le moniteur hybride mais par les opérations.

III.5.2 - Calcul du code des opérations.

Les opérations peuvent en général utiliser plusieurs paramètres et donc aboutir à des traitements complexes. La première phase du traitement amène à la génération du code binaire de l'opération.

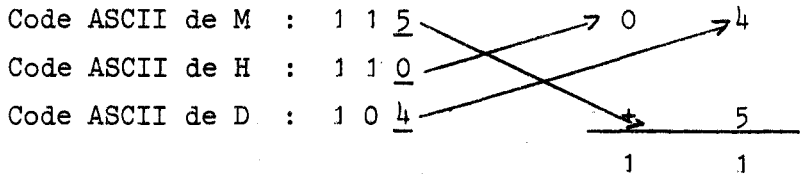
De façon à disposer d'une écriture des instructions à la fois

explicite et dense le nombre de caractères alphanumériques servant à décrire le code symbolique d'une opération a été fixé à trois. Pour en vérifier la validité, ce code est comparé au contenu d'une table de référence.

Cette dernière est composée de n couple de mots pour un nombre d'opérations maximum égal à n. Le premier mot de chaque couple contient les 18 bits nécessaires au codage des 3 caractères ASCII.
Le second contient la taille de l'opération, son type et d'autres renseignements qui seront utilisés dans la suite du traitement.

Afin d'économiser le volume mémoire utilisé, le code binaire n'est pas rangé dans cette table mais calculé. L'écriture et la simplicité du traitement informatique a présidé au choix de la règle de calcul. Cette dernière consiste aprés avoir gardé le dernier chiffre octal de chaque caractère à concaténer les deux derniers et à leur additionner le premier.

L'exemple suivant illustre le calcul du code de l'opération de mise en hold des intégrations. Cette dernière s'écrit MHD.



Le code ainsi obtenu est rangé dans la partie code opération du mot instruction et la taille prise dans la table des codes opérations est mémorisée dans le préfixe.

La phase qui suit le calcul du code opération consiste à reconnaître le type du paramètre immédiat utilisé, et à le ranger dans ECO.

III.5.3 - Traitement des paramètres immédiats.

Les paramètres immédiats sont des paramètres dont le code peut être rangé sur les six bits de la partie extérieure du code opération d'un mot instruction. Il n'en existe donc qu'un seul par opération

ECO extension du code opération ou paramètre immédiat.

L'écriture et le codage de ces paramètres dépendent de leur nature et par conséquent plusieurs programmes distincts sont nécessaires pour les traiter. Dans ce sens les opérations sont regroupées en classes ce qu'illustre la figure III.5.3.
Selons l'utilisation du paramètre ces classes peuvent correspondre à des adresses sur la calculatrice analogique, à des paramètres utilisés par le moniteur hybride, à des étiquettes , ou à des couples de caractères permettant de générer les paramètres d'opérations particulières.
La nature de la classe est donnée par le deuxième mot des couples de la table des codes opération.

5.3.1 - Adresse de la calculatrice analogique

Les adresses de la calculatrice analogique sont par définition appelées opérandes. Ils reprennent le symbolisme utilisé pour le matériel à savoir trois caractères alphanumériques.

Pour en vérifier la validité, les codes sont comparés au contenu d'une table de référence. Cette table, en outre, donne le code numérique associé.

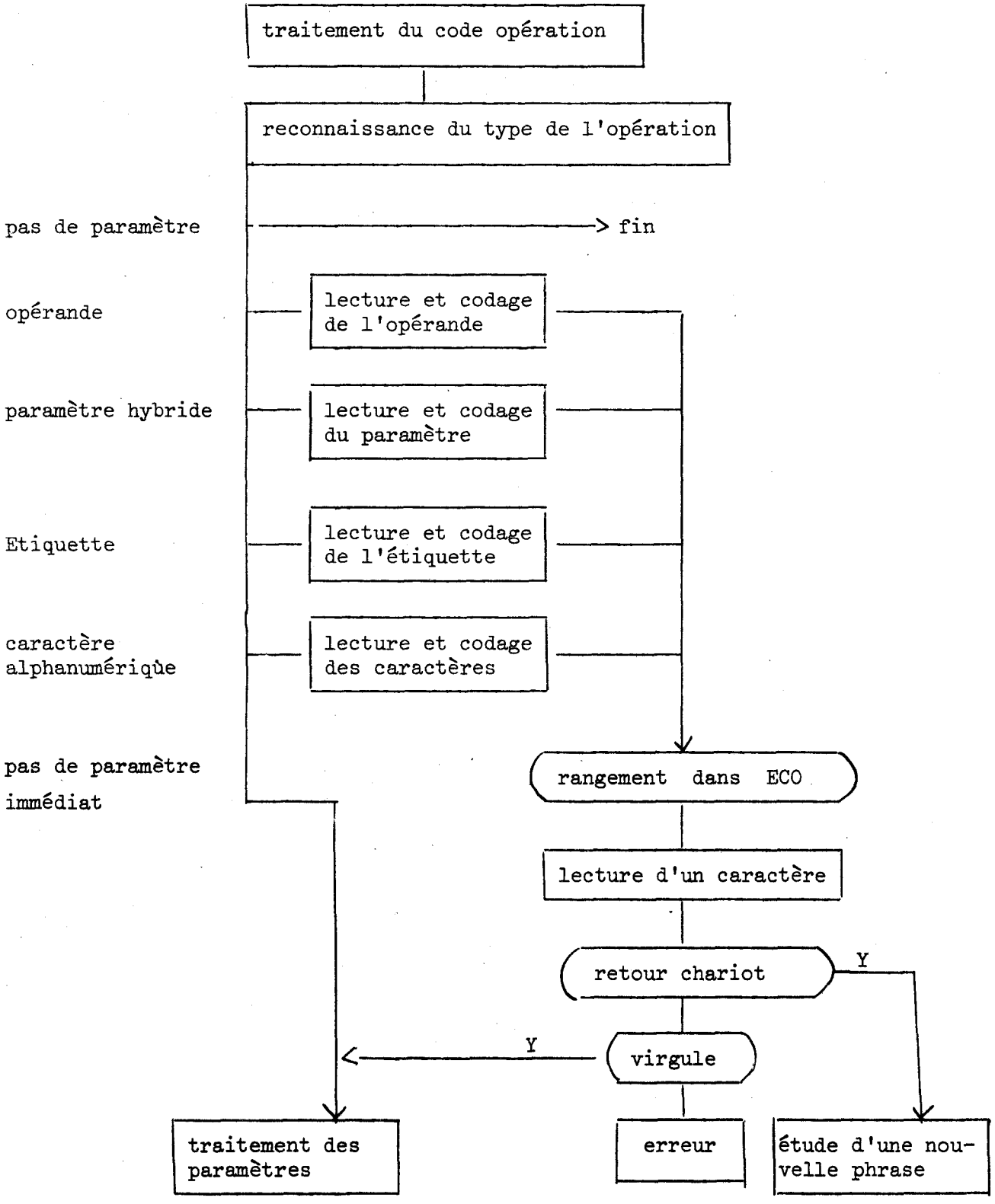


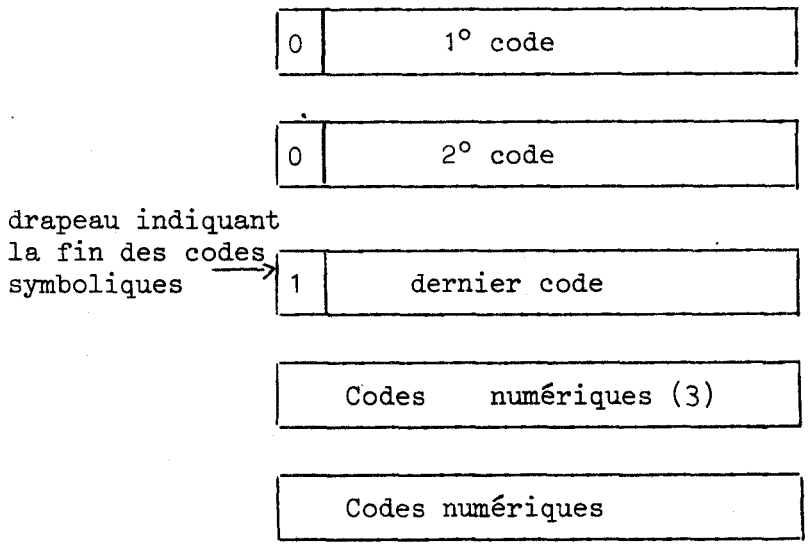
Figure III.5.3

Organigramme de traitement des opérations

Les codes symboliques utilisant trois caractères occupent dans la table des mots complets. Le fait d'associer à chacun un deuxième mot donnant le code numérique aurait impliqué une perte de place importante. En effet le taux de remplissage de ce deuxième mot aurait été inférieur à un tiers.

Nous avons donc été amenés à choisir une organisation mieux adaptée à la contrainte d'encombrement de la mémoire vive.

La table des opérandes est divisée en plusieurs "sous tables". Chaque sous table correspond aux opérandes possibles d'une opération. Elle est divisée en deux parties, la première donne la liste des codes permis et la deuxième les codes numériques associés. Ces codes sont regroupés à raison de 3 par mot.
Le schéma suivant précise la structure d'une telle sous table.



L'accés à la deuxième partie de cette table est controlée par des pointeurs "d'abcisses" et d'"ordonnées" incrémentés au cours de balayage de la première partie.

5.3.2 - Paramètres utilisés par le moniteur hybride

Les paramètres utilisés par le moniteur hybride ont été décrits au paragraphe III.5.1. Ils sont utilisés par les opérations d'initialisation de paramètre, d'incrémentation et de rupture conditionnelle. Les codes symboliques sont constitués de trois symboles alphanumériques. Le premier est obligatoirement la lettre P. Les deux derniers symboles sont comparés au contenu d'une table de référence. Cette table donne en outre le code binaire associé.

5.3.3. - Etiquettes et symboles alphanumériques.

Les étiquettes et les couples de caractères alphanumériques ont été introduits pour permettre une écriture en symbolique ou mnemotechnique de certains paramètres immédiats. Les pointeurs de numéro de tables sont de ce type. Le code numérique associé est obtenu par troncature des codes ASCII. Les derniers chiffres octaux des caractères ou des deux derniers symboles autre que la parenthèse pour les étiquettes sont gardés et rangés dans ECO. Aprés avoir chargé l'extension du code opération (ECO) avec la valeur du paramètre immédiat le compilateur étudie les autres paramètres. Par opposition aux paramètres immédiats qui n'utilisent que six bits, ces paramètres sont dits de "type long".

III.5.4 - Traitement des paramètres de type long.

Ces paramètres sont rangés dans les mots qui suivent le mot instruction proprement dit. Ils peuvent avoir deux significations essentielles :

i) En tant qu'adresse ils sont utilisés par les opérations de rupture ou de branchement. Dans ce cas ils sont souvent repérés par des étiquettes.

ii) En tant que valeur ils représentent des données. Ils peuvent être écrits en décimal, ou en octal. Dans le premier cas ils sont cadrés en poids faible et peuvent être positifs ou négatifs. Dans le deuxième cas, ils sont cadrés en poids fort et sont alors principalement utilisés comme variables logiques. Ils peuvent être repérés dans les deux cas par des étiquettes.

Le traitement d'un paramètre de type long commence donc nécessairement par une phase de reconnaissance. La figure III.5.4 en donne le principe. Suivant la nature du premier caractère il y a branchement à un sous programme de vérification spécifique puis codage.

Une parenthèse correspond à une étiquette. Le code est calculé par une troncature, ce qui permettra au compilateur de retrouver le code homologue par balayage de la table des étiquettes lors de la phase d'assemblage-chargement. De plus la présence de cette étiquette est signalée au niveau du préfixe par un bit drapeau. Le code obtenu est alors rangé dans le mot réservé au paramètre et aucune modification n'est apportée au niveau de la table des étiquettes.

Les valeurs décimales sont repérées par la présence des signes + ou -. Elles n'utilisent que des caractères numériques. Dans le cas contraire une erreur est détectée et la correction de phrase est demandée sur le télétype. La valeur décimale est codée en binaire, puis rangée dans le mot réservé au paramètre.

L'absence des caractères parenthèse d'une part, plus ou moins d'autre part implique alors la définition d'une valeur octale. La frappe d'un premier caractère erroné pour un paramètre est donc détectée à ce niveau. Les valeurs octales sont composées de quatre caractères numériques compris entre 0 et 7.

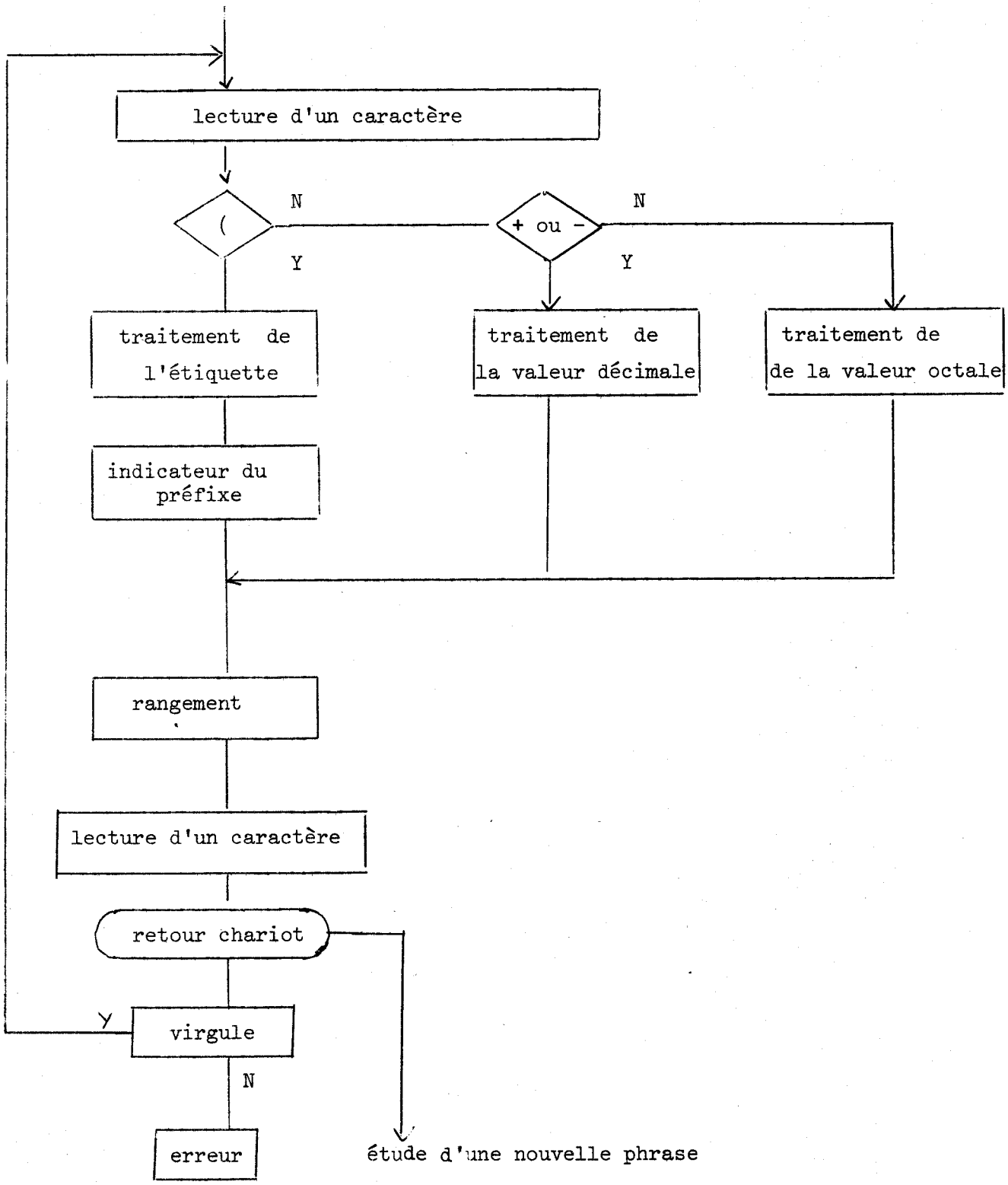


Figure III.5.4

Traitement des paramètres de type long

En fin de traitement un nouveau caractère est analysé aprés lecture. S'il s'agit d'une virgule et que la taille de l'opération c'est-à-dire le nombre maximum de paramètres n'est pas atteint le compilateur boucle sur le traitement des paramètres. S'il s'agit d'un retour chariot les traitements relatifs à l'instruction en cours sont terminés. Le compilateur passe alors à l'étude de la phrase.

III.5.5 - Opérations codées en octal.

Lorsqu'une nouvelle opération est introduite il faut d'une part compléter, la bibliothèque du moniteur et son sommaire, d'autre part modifier plusieurs tables de l'interpréteur. S'il s'agit d'opérations spécifiques relatives à un problème particulier il peut être intéressant de simplifier la procédure.
Dans ce cas, les opérations peuvent être directement décrites en octal ce qui évite toute modification au niveau du compilateur.

Le symbole dollar $ caractérise les phrases décrivant de telles opérations. La taille de l'opération est écrite immédiatement aprés ce symbole. Le type de l'opération n'est pas explicité. En cas d'opérande immédiat celui-ci devra nécessairement être écrit en octal.

De façon à faciliter l'emploi d'une telle procédure le mode et le code de déclenchement peuvent être indifféremment écrits en octal ou en symbolique. Le compilateur effectue l'union des codes binaires obtenus pour les champs variables de déclenchement et mode de déclenchement.

Les paramètres de type long subissent le même traitement que pour une instruction entièrement écrite en symbolique.

L'exemple suivant illustre les différentes écritures possibles pour une même phrase. L'instruction de rupture inconditionnelle RPI et la variable de déclenchement V3 ont respectivement pour code 03 et 53. Le codage de l'instruction utilise deux mots.
Le symbole ∅ sera donc suivi de l'indicateur de taille 2.

L'écriture symbolique est la suivante

(002) V3 : RPI, (004)

en octal les écritures suivantes sont équivalentes :

(002) : ∅2 0053 034 , (004)

ou

(002) V3 : ∅2 0000 030 , (004)

Dans le deuxième cas le mode déclenché est défini par la présence de la variable de déclenchement. Il est donc inutile de le préciser dans l'instruction.

III.6 - Message d'erreur

Le compilateur effectue l'analyse et le codage d'une phrase de façon séquentielle. S'il découvre une erreur d'une part il la signale, d'autre part il se met en attente de la réécriture sur le télétype de la partie érronée.

Le message d'erreur se compose de trois parties.

- En premier lieu la nature de l'erreur est codée suivant la syntaxe

"ERR ij" , ij ∈ 0,19

La signification de ces erreurs est donnée en annexe 2.

- En second lieu pour permettre une localisation aisée de la phrase erronée lorsque le programme source est sur bande perforée le compilateur indique la dernière étiquette correctement compilée suivie du nombre d'instructions analysées depuis cette étiquette. La syntaxe de cette information est

  ETI : (x,x,x) + y

  (xxx) est le code octal de l'étiquette.

- En dernier lieu le compilateur réécrit complétement la phrase erronée.

L'exemple suivant illustre la façon dont une erreur dans la frappe du code de l'opération MIC est signalée

```
ERR 09
ETI (020) + 00
(020) V2 : M1C , 2A1
```

L'étiquette (020) et la variable de déclenchement V2 ont déjà été compilées, l'opérateur n'aura donc à refrapper que le code opération et les informations qui le suivent, c'est-à-dire :

```
MIC , 2A1.
```

IV - ASSEMBLEUR - CHARGEUR

L'assembleur-chargeur répond à deux objectifs essentiels.

Tout d'abord, il est chargé de traduire le code binaire translatable en binaire absolu. Pour cela il doit supprimer les préfixes et remplacer les symboles non résolus par leur valeur absolue prise dans la table des étiquettes.

En second lieu au fur et à mesure de la traduction, il est chargé d'implanter le code binaire en mémoire vive évitant ainsi une phase supplémentaire de chargement.

L'organigramme de traitement réalisé est donné sur la figure IV. L'assembleur-chargeur traite le programme source mot par mot. Une instruction est délimitée par l'indicateur de taille contenue dans le préfixe. Le chargement commence par celui du premier mot qui contient essentiellement le code opération et la variable de déclenchement. Ce mot ne subit aucune modification. Le chargement se poursuit par celui des paramètres. Lorsque le préfixe indique que le paramètre est du type non résolu, ce dernier avant d'être chargé est remplacé par l'adresse absolue associée.

A ce niveau les programmes hybrides sont traités en un seul bloc. La fin du chargement est atteinte lorsque la taille du programme codé en binaire est égale à celle calculée au cours de la phase compilation par sommation des tailles de chaque instruction.

La distinction entre les différents programmes sera faite par le moniteur hybride par l'intermédiaire des adresses de début de programmes calculées au cours de la première phase.

V - SUPERVISEUR D'ENCHAINEMENT

Avant d'être exécuté sous le contrôle du moniteur, un programme hybride écrit sous forme symbolique doit être compilé, puis assemblé et chargé en mémoire vive. De manière à simplifier les manipulations nous avons écrit un superviseur essentiellement chargé de gérer l'enchainement de ces trois phases. Trois autres fonctions ont été par ailleurs assignées au superviseur de manière à accroître la souplesse d'utilisation.

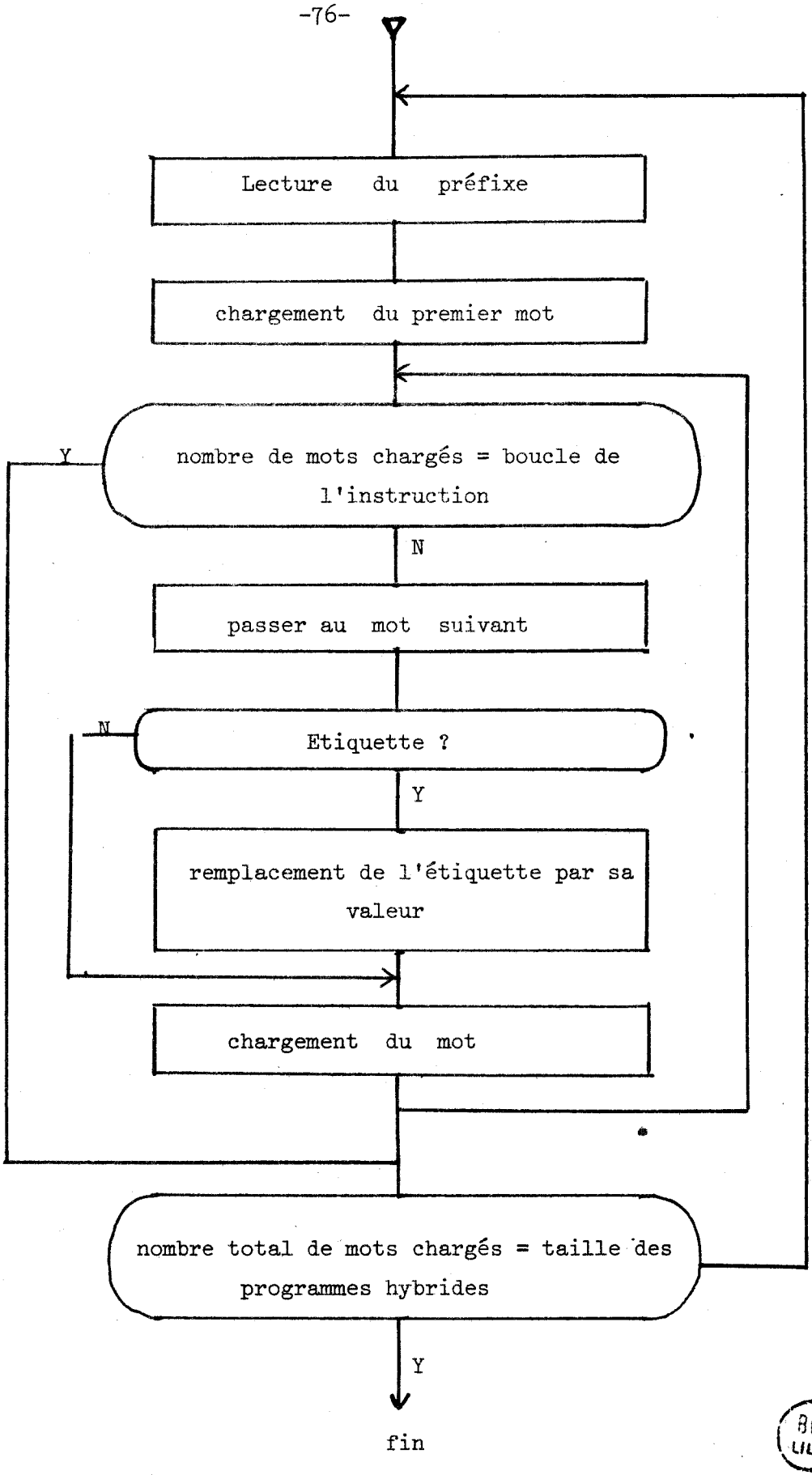


Figure IV - Principe de fonctionnement de l'assembleur chargeur.

Tout d'abord, il prévient l'utilisateur d'un fonctionnement erroné dans l'une des trois phases précédentes.

En second lieu, il permet en mode conversationnel d'une part l'assignation des fichiers source, d'autre part l'inhibition sélective par masquage des interruptions qui ne doivent pas être activées.

En dernier lieu il convient de signaler que certains modules de programmes concernent les trois phases, et doivent également résider en mémoire lors de la phase d'exécution du superviseur. C'est le cas des programmes de gestion de table et des drivers de périphérique.

A titre d'illustration nous donnons ci-dessous l'exemple d'un dialogue relatif à l'utilisation de ce superviseur. Le programme source est introduit sur le lecteur rapide sous forme d'un ruban perforé.

Dialogue	Chargement en mémoire et exécution des programmes
SUPERVISEUR HYBRIDE 1976	compilateur
LR OU TC ? : LR	compilation
DAC SH	assembleur-chargeur
GO	assemblage-chargement
ETAT ?	moniteur
01 77	phase opérationnelle

CONCLUSION

Le langage symbolique présenté dans ce chapitre a été défini de manière à simplifier la description des fonctions hybrides. La façon dont il a été structuré répond à trois objectifs.

Tout d'abord les pseudo-instructions permettent de délimiter les programme réalisant la fonction de multiprocesseur et d'effectuer des transmissions de paramètres de l'un vers l'autre.

En second lieu l'emploi systématique d'étiquettes simplifient le calcul des points de branchement relatifs aux instructions de rupture de séquence conditionnelle et inconditionnelle et de plus facilitent le repérage des paramètres.

En dernier lieu les opérations hybrides proprement dites ainsi que les opérandes et paramètres associés ont été codés selon un symbolisme qui tient compte des implantations effectives des opérateurs sur les consoles logiques et analogiques.

Le compilateur a donc pour objet d'une part la vérification de la syntaxe des instructions et éventuellement la gestion des corrections, d'autre part le codage binaire des instructions écrites en langage symbolique.

Le langage hybride présenté sous cette forme est d'une assimilation relativement aisée , en particulier il ne nécessite pas de la part de l'utilisateur de connaissance informatique préalable.

CHAPITRE III

EXEMPLES PRATIQUES D'UTILISATION DU LOGICIEL HYBRIDE

INTRODUCTION

Nous proposons dans ce chapitre d'illustrer sur trois exemples particuliers, les techniques spécifiques du calcul hybride découlant de l'utilisation du logiciel présenté dans ce mémoire.

Lorsqu'il s'agit d'analyser un problème de simulation nécessitant l'utilisation d'un couplage numérique/analogique, il convient d'étudier soigneusement le choix éventuel des opérateurs utilisables. En effet il est généralement possible en décomposant le problème de mettre en évidence plusieurs niveaux de programmation. Dans ce sens, il faut considérer qu'un "programme hybride" résulte de l'association globale et synchronisée d'un cablage analogique, d'un cablage logique, et d'un programme réalisé sur la base du logiciel étudié. Dans ce contexte la synchronisation par événement assure l'enchainement des opérations réparties sur les différents sous ensembles du calculateur.

Un premier exemple illustre globalement ce point de vue en précisant l'intérêt du symbolisme utilisé.

Afin de mettre en évidence les avantages que peuvent apporter les techniques de calcul hybride de type II par rapport d'une part aux techniques numériques et d'autre part aux techniques analogiques, nous proposons sur un exemple simple, de traiter un problème d'optimation statique.

En dernier lieu un problème relatif à la résolution d'équations aux dérivées partielles permet d'évaluer globalement l'intérêt pratique d'un tel système de simulation. Dans ce cas, en effet, le logiciel permet de résoudre simplement un problème de simulation relativement complexe.

III.1 - Résolution d'un problème de cinétique chimique.

Le premier paragraphe reprend l'exemple de recherche paramétrique d'extréma exposé au chapitre II et qui concernait l'étude du régime transitoire de la cinétique d'une série de réactions chimiques.

III.1.1 - Position du problème

Un problème relativement usuel en cinétique chimique /24/ est relatif à la recherche des concentrations maximales d'un ou plusieurs produits intermédiaires intervenant dans une série de réactions consécutives, irréversibles.

Envisageons le cas des 3 réactions décrites en (1).
Les produits A,B,C ont respectivement pour concentration $[A]$, $[B]$ et $[C]$, les constantes de vitesse sont notées k_1, k_2, k_3.

$$\begin{aligned} A &\xrightarrow{k_1} B \\ B &\xrightarrow{k_2} C \\ C &\xrightarrow{k_3} D \end{aligned} \qquad (1)$$

On cherche à maximiser la concentration du produit C.
Nous supposerons pour abréger les calculs, que la constante de temps de la réaction de dégradation de C est fixé. Le problème se réduit alors à chercher le maximum de $[C]$ par rapport aux paramètres k_1 et k_2.

La programmation hybride peut être envisagée sur plusieurs niveaux et permet de décomposer l'analyse de ce problème en plusieurs phases.

Dans un premier temps nous étudierons la simulation analogique des équations différentielles régissant le modèle.

Dans un deuxième temps nous définirons les opérateurs logiques et les commandes qu'il est possible d'utiliser sur le panneau logique.

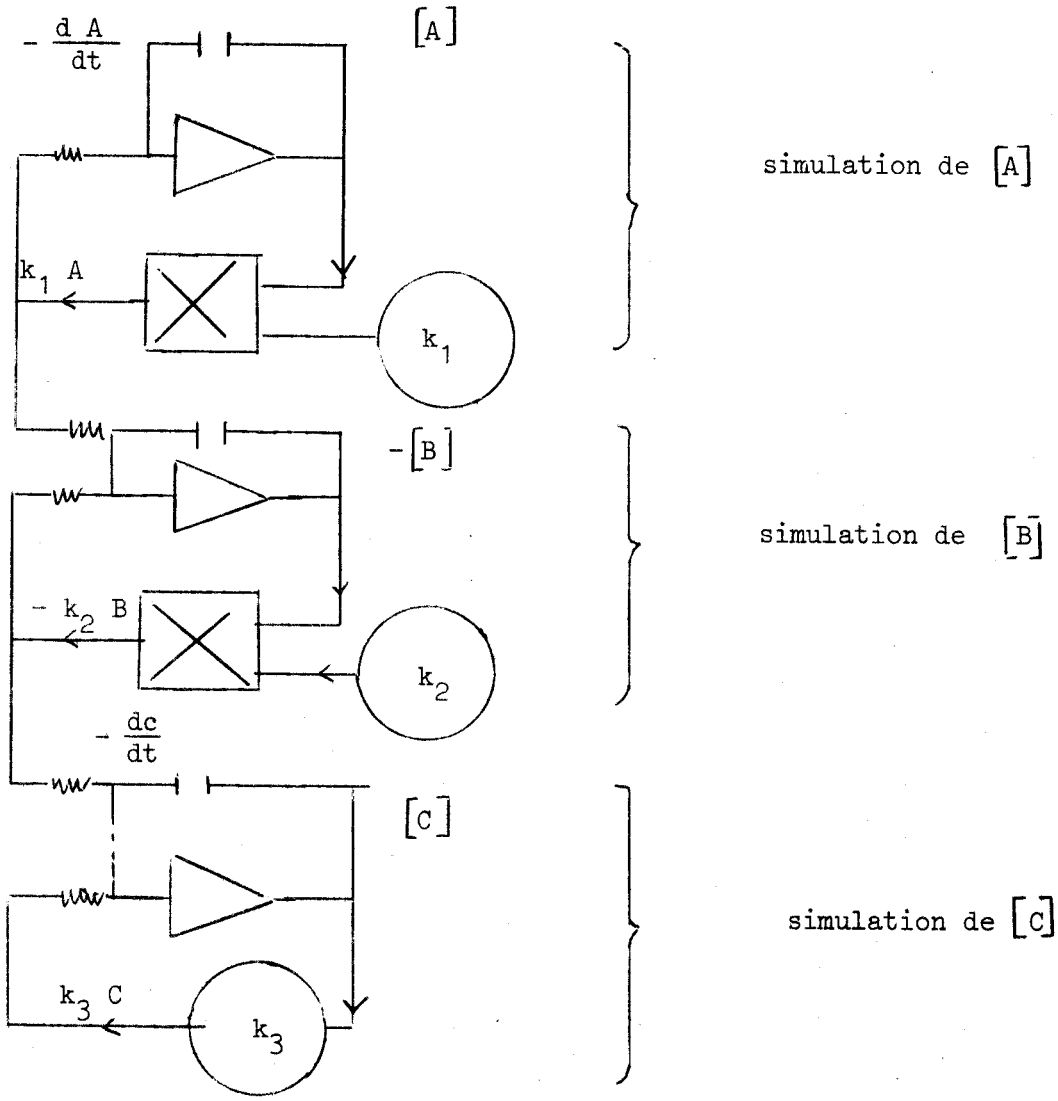
Enfin nous définirons les programmes hybrides nécessaires à la gestion de la simulation de l'ensemble.

III.1.2 - Simulation de l'évolution temporelle des concentrations.

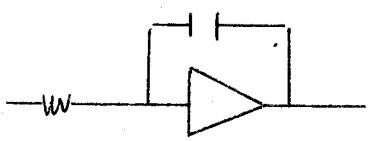
L'étude mathématique des équations qui régissent l'évolution temporelle des concentrations de chaque produit conduit à écrire le système différentiel (2)

$$\begin{aligned} \frac{d[A]}{dt} &= - k_1[A] \\ \frac{d[B]}{dt} &= - k_2[B] + k_1[A] \\ \frac{d[C]}{dt} &= - k_3[C] + k_2[B] \\ \frac{d[D]}{dt} &= k_3[C] \end{aligned} \qquad (2)$$

La simulation de ces équations est donnée par le schéma de cablage suivant :

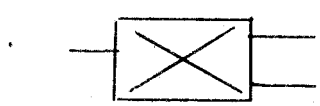


Symboles utilisés :



représente un intégrateur

et



un multiplieur

Figure III.1 - Schéma de cablage analogique

III.1.3 - Logique de commande de la recherche d'extremum

La concentration $[C]$ est extremum lorsque sa dérivée s'annule ; le passage par zéro de $\frac{d[C]}{dt}$ déclenche donc la prise de mesure de $[C]$.

Une étude théorique ou une simulation préalable (figure III.1) permet de démontrer que la courbe $[C] = f(t)$ ne possède qu'un seul maximum. Il est possible dans ces conditions, pour accélérer les calculs, d'arréter la résolution lorsque l'extremum est atteint.

Cet extremum dépendant de la valeur de couple (k_1, k_2), il est donc possible d'enregistrer une famille de courbes des maxima $[C] = f(k_2)$ à k_1 donné ou encore des maxima de $[C] = f(k_1)$ k_2 donné.

La commande du panneau analogique comprend quatre éléments :

1) La détection du passage par zéro de $\frac{d[C]}{dt}$

2) l'ordre de prise de mesures et d'enregistrement de $[C]$

3) la sortie des valeurs de k_1 et k_2

4) la sortie des ordres concernant les intégrateurs c'est-à-dire les mises en condition initiale, en résolution ou en mémoire.

La détection du passage par zéro de $\frac{d[C]}{dt}$ est réalisée par un comparateur analogique connecté à cette variable. La sortie du comparateur est branchée sur un appel externe qui engendre l'événement "$[C]$ maximum".

concentrations

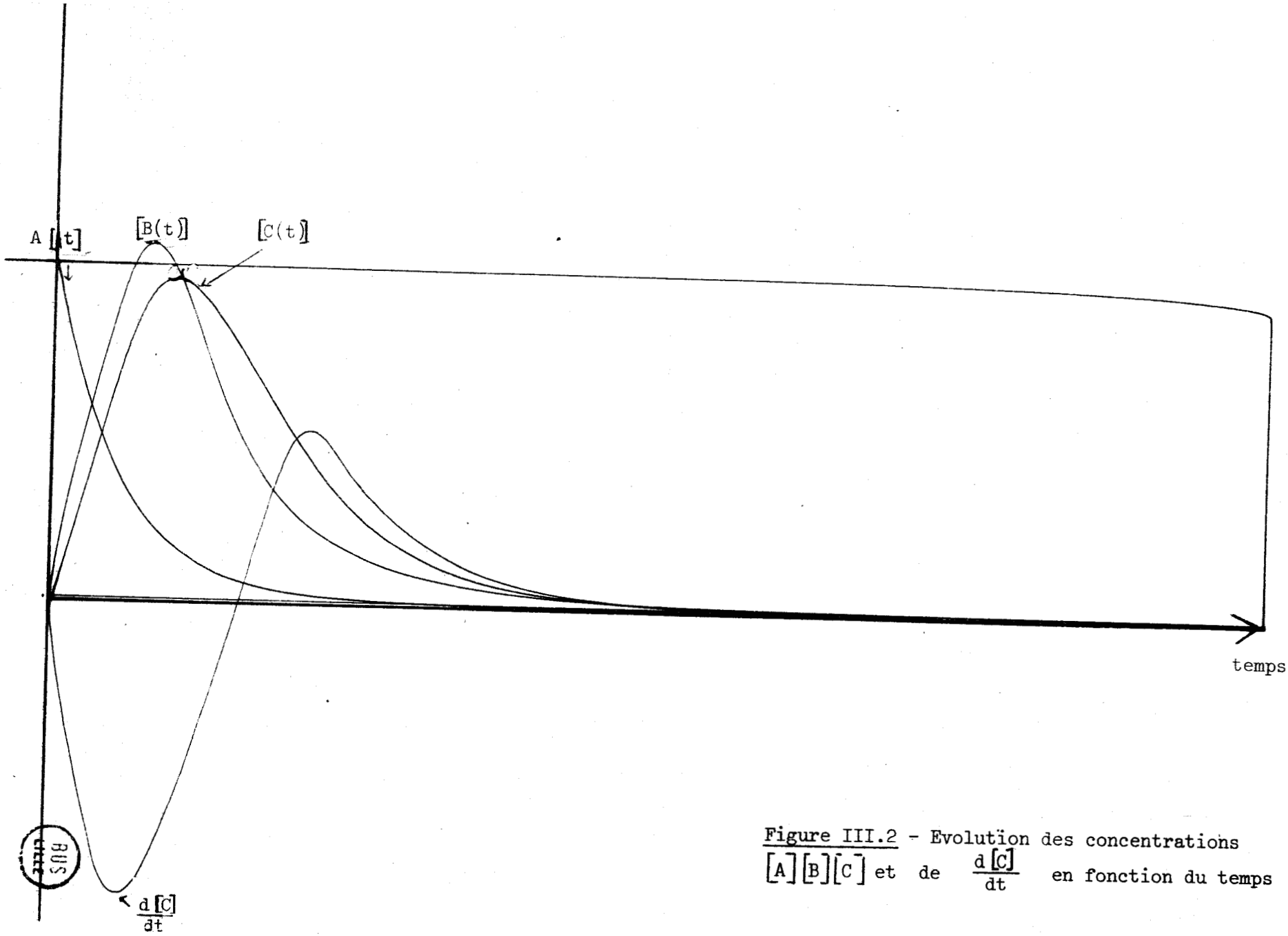


Figure III.2 - Evolution des concentrations [A] [B] [C] et de $\frac{d[C]}{dt}$ en fonction du temps

L'utilisation d'un oscilloscope à mémoire comme périphérique d'enregistrement nécessite l'échantillonage et la mémorisation de [C]max. Les ordres de prise de mesure s'appliquent dans ces conditions à un track and store qui délivre un signal sur une voie Y de l'oscilloscope.

La figure III.3 donne le schéma de cablage complet. Les différents éléments y ont été regroupé en champs correspondant chacun à un des trois niveaux de programmation.

III.1.4 - Programme hybride

Suivant leur nature les programmes peuvent être regroupés en deux ensembles.

- Premièrement les sous ensembles de calcul de k_1 et k_2 qui sont écrits en langage machine. Ils ont pour fonction d'initialiser k_1 et k_2 puis de les incrémenter à chaque appel.

- Deuxièmement le programme hybride proprement dit. Les figures III.4 et III.5 en donnent l'organigramme et le listage. Il se décompose en quatre phases principales.

En premier lieu, il débute par l'initialisation des nombres de boucles correspondant au nombre de valeurs choisies pour k_1 et k_2.

Ensuite les valeurs de k_1 et k_2 sont calculées puis sorties et la résolution est lancée.

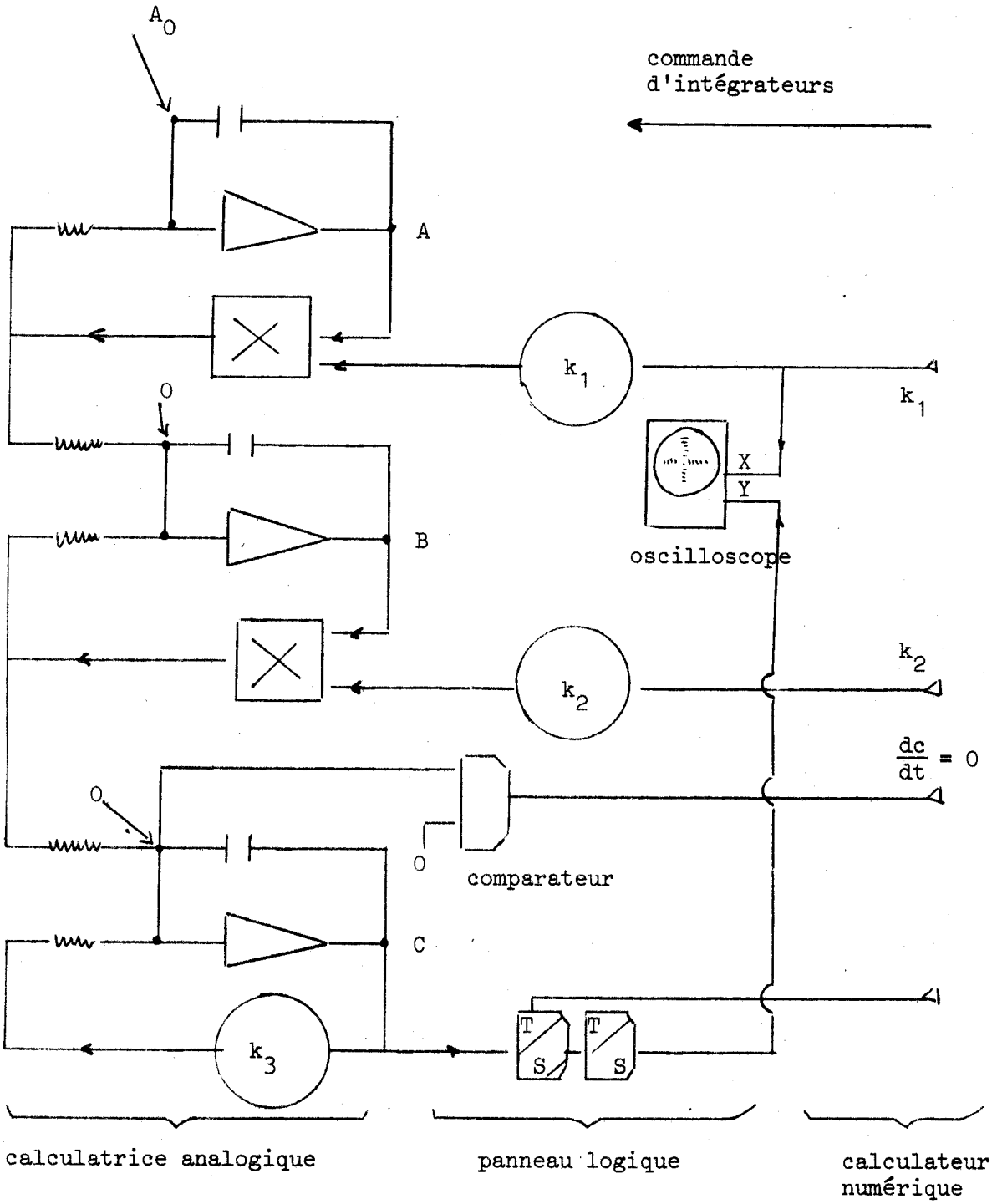


Figure III.3 - Schéma de cablage

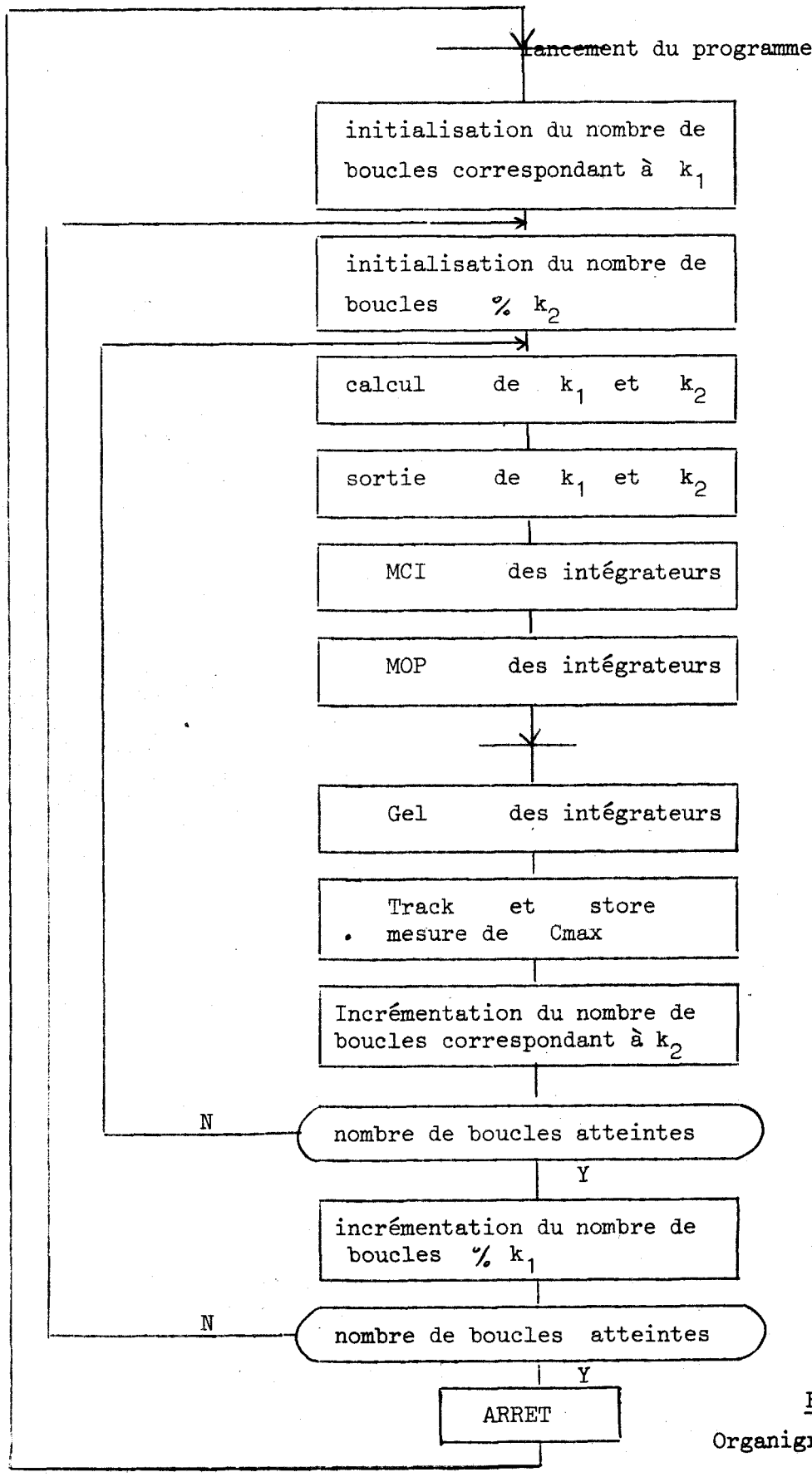


Figure III.4

Organigramme de principe

```
    * COM       RECHERCHE  D'EXTREMUM
    * COM       A —> B (K1)    B →C (K2)     C →D (K3)
    * COM       MAXIMUM DE C (K1, K2)
    * EXT(KO1)  6000  (KO2) 6001
    * EXT(INI)  6002  (CAL) 6007
    * DEB
(010)EO : PI1 = -5                    <NB BOUCLES SUR K1
          BRI,(INI)                   <init   k1 k2
(021)   : PI2 = -32                   <NB BOUCLES SUR K2
(022)   : BRI,(CAL)                   <CALCUL DE K1,K2
          SOA,(KO1),OD1               <SORTIE DE K1
          SOA,(KO2),OD2               <SORTIE DE K2
          PHO = -10                   <TEMPORISATION POUR
          MCI,2A1                     <MCI
(030)HO : MOP,2A1                     <RESOLUTION
(040)E1 : GEL                         <ARRET RESOLUTION
          TRK,2B9                     <prise de mesure
          INC,PI2                     <INCREMENT NB BOUCLE
          PHO = -10                   <TEMPORISATION POUR
(050)HO : STO,2B9                     <TRK
          RPC,PI2,(022),(060)
(060)     INC,PI1                     <INCREMENT NB BOUCLE
          RPC,PI1,(021),(070)
(070)     ARR
          RPI,(010)
    * FIN
```

Le sous programme (INI) remet à zéro k_1 et k_2 pointés par (KO1) et (KO2). Le sous programme (CAL) incrémente ces valeurs figure III.5 programme hybride.

La troisième phase commence lorsque l'événement $\frac{d(C)}{dt} = 0$ est effectif. Le calcul est arrêté et les ordres sont envoyés au track store pour qu'il effectue l'enregistrement de $[C]$ maximum.

La dernière phase permet de relancer la séquence tant que le nombre de boucle n'est pas atteint c'est-à-dire tant que toutes les valeurs de k_1 et k_2 n'ont pas été balayées.

Les figures III.6 et III.7 montrent respectivement l'évolution de k_1 et de k_2, et la famille de courbe $[C]max = \wp_{k_1}(k_2)$.

Cet exemple trés simple a mis en évidence l'utilisation des techniques du calcul hybride et les méthodes d'analyse qui lui sont associées. La technique de simulation utilisée consiste à décomposer le problème en deux parties relatives à la simulation du processus et à l'élaboration de sa commande.

Le choix des opérateurs apparamment implicite est cependant fixé par la recherche d'une meilleure adaptation du modèle simulé au processus concret dont il est l'image.

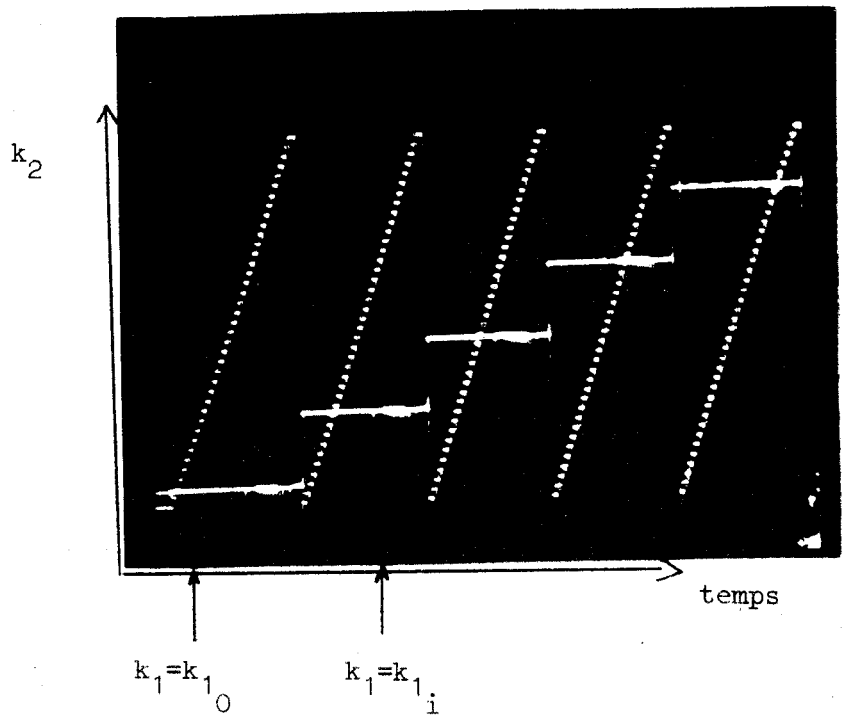


Figure III.6 - Evolution de k_2

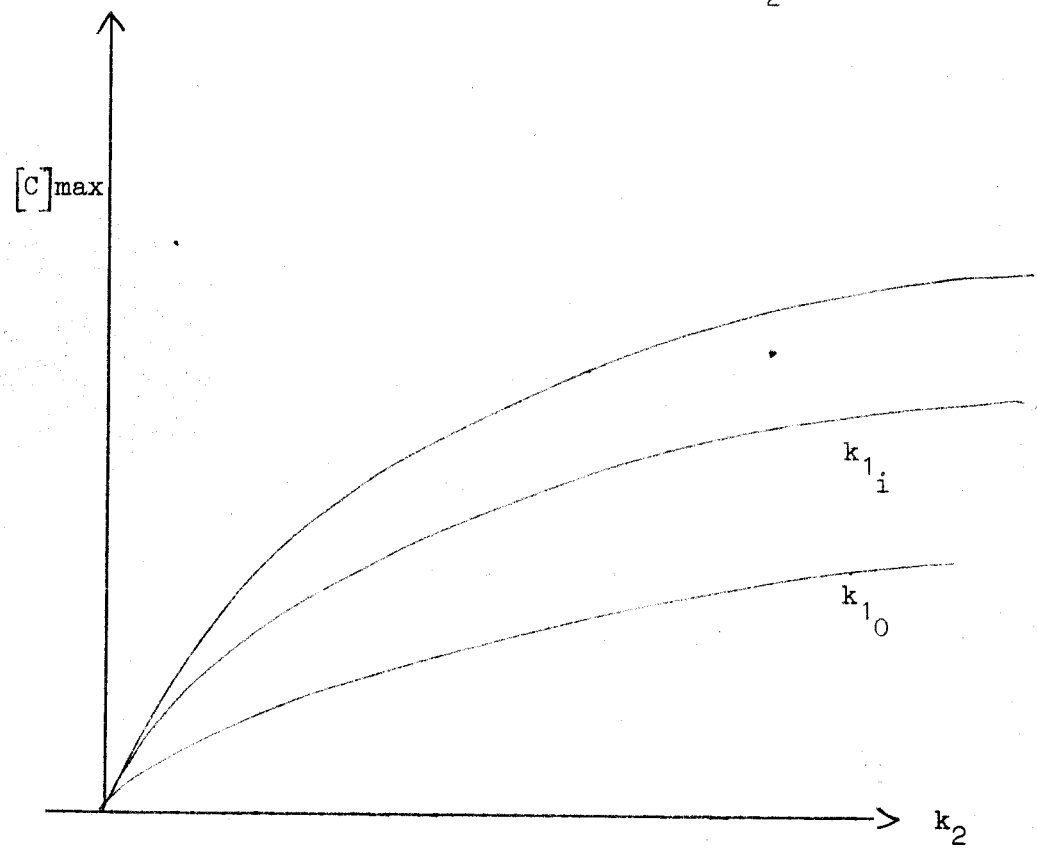


Figure III.7 - Courbes $[C]max = f_{k_1}(k_2)$

III.2 - Stabilité d'un algorithme de calcul

L'exemple que nous proposons de traiter maintenant permet d'illustrer une application des simulateurs hybrides en analyse numérique. Il s'agit de mettre en évidence les caractéristiques de stabilité d'un algorithme en fonction des conditions initiales et de certains paramètres. Un algorithme hybride basé sur la méthode du minimax permettra de visualiser en continu l'évolution de la solution en fonction de ces paramètres.

III.2.1 - Présentation du problème

Le problème que nous proposons de résoudre concerne la recherche du maximum de la fonction f { $f: x \in R^q \rightarrow f(x) \in R$} en présence des contraintes égalité et inégalité définies par l'équation (3) /25/, /26/, /27/.

$$\begin{aligned} g_i(x) = 0 \quad & \{g_i : x \in R^q \rightarrow g_i(x) \in R, \\ & \forall i = 1, 2, \ldots m\} \\ h_i(x) < 0 \quad & \{h_i : x \in R^q \rightarrow h_i(x) \in R, \\ & \forall i = 1, 2, \ldots p\} \end{aligned} \qquad (3)$$

De manière à rappeler le théorème du minimax /26/, notons $\phi(x, \lambda, \nu)$ la fonction suivante :

$$(x, \lambda, \nu) \quad \{\phi: x \in R^q, \lambda \in R^m, \nu \in R^p_+ \rightarrow \phi(x, \lambda, \nu) \in R\}$$

où λ sont les paramètres de Lagrange /16/ et ν ceux de Kuhn et Tücker /27/ ; et tels que :

$$\begin{aligned} \phi(x, \lambda, \nu) &= -f(x) + \lambda^T g(x) + \nu^T h(x) \\ g(x) &= \{g_i(x)\} \qquad i = 1, 2, \ldots m \\ h(x) &= \{h_i(x)\} \qquad i = 1, 2, \ldots p \end{aligned} \tag{4}$$

Le théorème du minimax, appliqué au système discret, permet de définir l'algorithme d'optimisation suivant :

$$\begin{aligned} \Delta x &= x_{n+1} - x_n = -k_1 \left(\frac{\partial \phi}{\partial x}\right), \qquad x = x_n \\ \Delta \lambda &= \lambda_{n+1} - \lambda_n = +k_2 \left(\frac{\partial \phi}{\partial x}\right), \qquad \lambda = \lambda_n \end{aligned} \tag{5}$$

$$\begin{cases} \Delta \nu_j = k_3 \, h_j(x) \quad , x = x_m \quad \text{si} \quad h_j(x) > 0, \; \nu(0) = 0 \\ \nu_j = 0 \quad \text{si} \quad h_j(x) < 0 \qquad \forall j = 1, 2, \ldots p \end{cases}$$

Si cet algorithme est stable, les points d'équilibre sont les maxima locaux de la fonction f en tenant compte des contraintes décrites en (3).
Pour différentes valeurs de conditions initiales ou de paramètres au voisinage d'un équilibre il est possible de mettre en évidence pour la solution des évolutions qui peuvent être stables ou instables..

III.2.2 - Simulation de la partie continue de l'algorithme

La simulation des contraintes inégalités est difficilement réalisable selon une technique purement analogique. En effet, elle fait apparaitre une non linéarité particulière, et nous proposons de la traiter

par programmation hybride.

Dans ces conditions l'algorithme de la partie continue se réduit au système d'équations (6)

$$\left.\begin{aligned} \overset{\circ}{x} &= -k_1 \frac{\partial \phi}{\partial x} \\ \overset{\circ}{\lambda} &= k_2 \frac{\partial \phi}{\partial x} \end{aligned}\right\} \qquad (6)$$

Pour illustrer cet exemple nous prendrons une fonction f de deux variables α et β avec une contrainte égalité et une contrainte inégalité.

$$\left.\begin{aligned} &f(\alpha,\beta) = 93{,}5+2{,}8\alpha+3{,}6\beta-7\alpha^2-3\beta^2+2{,}7\ \alpha\beta \\ &g(\alpha,\beta) = 85{,}3-11{,}7\alpha-18{,}3\beta-5{,}2\alpha^2-9{,}2\beta^2-6{,}3\alpha\beta = 0 \\ &\alpha < 0 \end{aligned}\right\} \qquad (7)$$

La fonction ϕ devient dans ces conditions

$$\phi = f(\alpha,\beta) + \lambda g(\alpha,\beta) + \nu\alpha$$

Et les équations continues correspondant au modèle continu sont les suivantes :

$$\begin{aligned} \overset{\circ}{\beta} &= -\frac{K_1}{100}\frac{\partial \phi}{\partial \beta} = \frac{K_1}{100}\left|-6\beta+2{,}7\alpha+18{,}4\alpha\beta+6{,}3\lambda\alpha+18{,}3\lambda+3{,}6\right| \\ \overset{\circ}{\alpha} &= \frac{K_1}{100}\frac{\partial \phi}{\partial \alpha} = \frac{K_1}{100}\left|2{,}7\beta-14\alpha+6{,}3\lambda\beta+10{,}4\lambda\alpha+11{,}7\lambda+2{,}8\ -\nu\right| \\ \overset{\circ}{\lambda} &= \frac{K_2}{100}\frac{\partial \phi}{\partial \lambda} = \left|\frac{K_2}{100}\ -18{,}3\rho-11{,}7\alpha-5{,}2\alpha^2-0{,}2\beta^2-6{,}3\alpha\beta+85{,}3\right| \end{aligned} \qquad (8)$$

La figure III.8 précise le cablage analogique correspondant à ce système (8)

III.2.3 - Programmes hybrides

Dans cet exemple trois fonctions sont à réaliser par programmation.

Tout d'abord il convient de simuler la contrainte inégalité sur la composante α. Dans ce but le paramètre de Kuhn et Tücker ν est périodiquement calculé à partir des valeurs mesurées de α , puis sorti au niveau du panneau analogique. Le calcul est effectué par un sous programme écrit en assembleur.

En deuxième lieu les programmes hybrides assurent d'une part le lancement de la résolution de l'algorithme et d'autre part l'arrêt de celle-ci au gré de l'utilisateur.

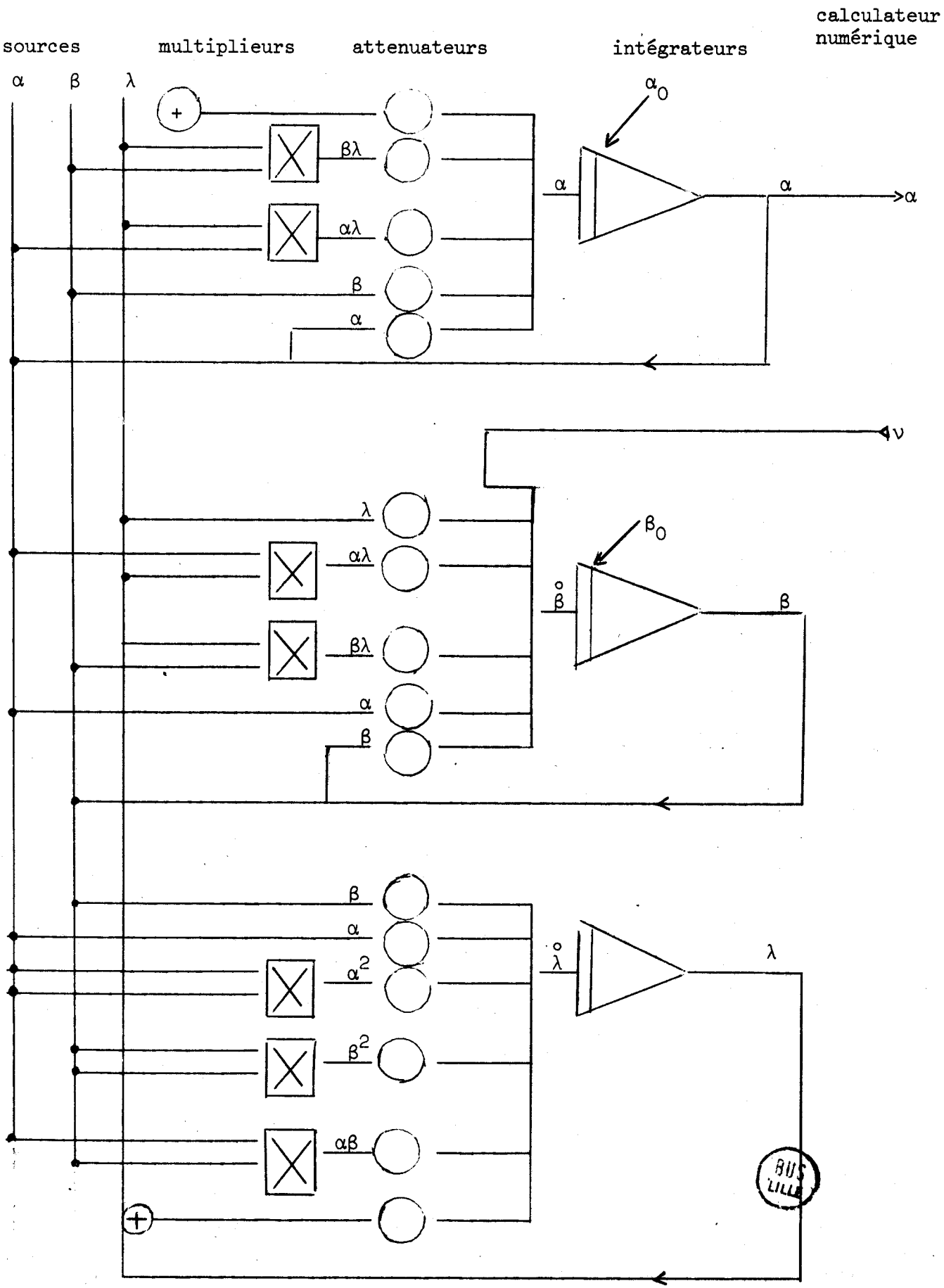


Figure III.8 - Cablage analogique

Le lancement de la résolution et la gestion de la contrainte sont réalisées par un premier programme de pointeur d'instruction HP1.

L'arrêt de la résolution est obtenu par un deuxième programme se déroulant en parallèle avec le premier. Il repositionne le pointeur HP1 sur la macro-instruction dont le code de déclenchement correspond à l'ordre de lancement. Le calculateur se met alors en attente d'un nouvel ordre de lancement.

Les organigrammes et les listages de ces programmes sont donnés par les figures (III.9) et (III.10).

III.2.4 - Résultats

La figure (III.11) représente les évolutions de α et β pour différentes valeurs des conditions intiales et de gains. Elle met en évidence que pour un même optimum l'algorithme peut se dérouler de façon stable (a et b) ou instable (c).

Dans le deuxième cas il est impossible d'obtenir la valeur recherchée.

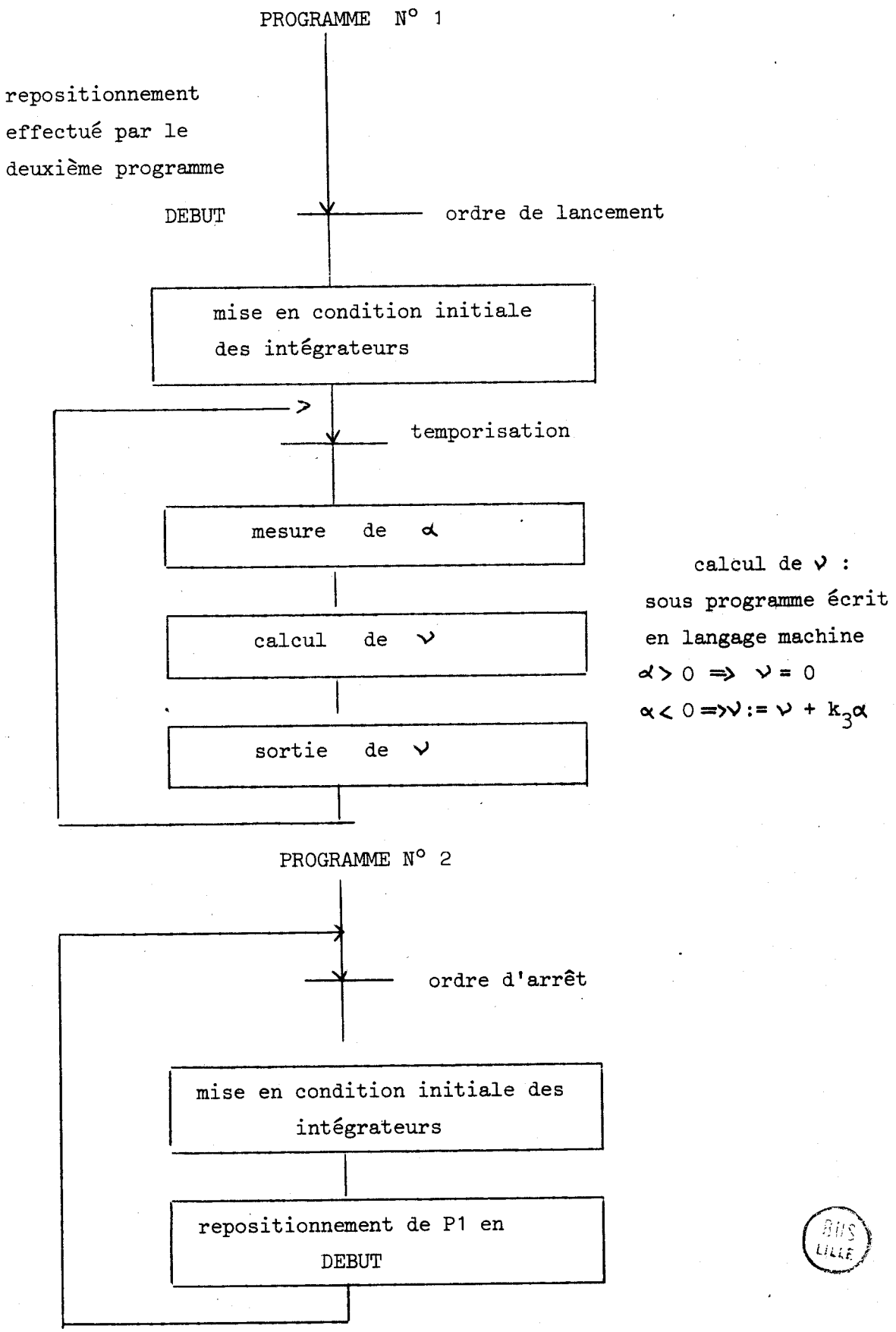
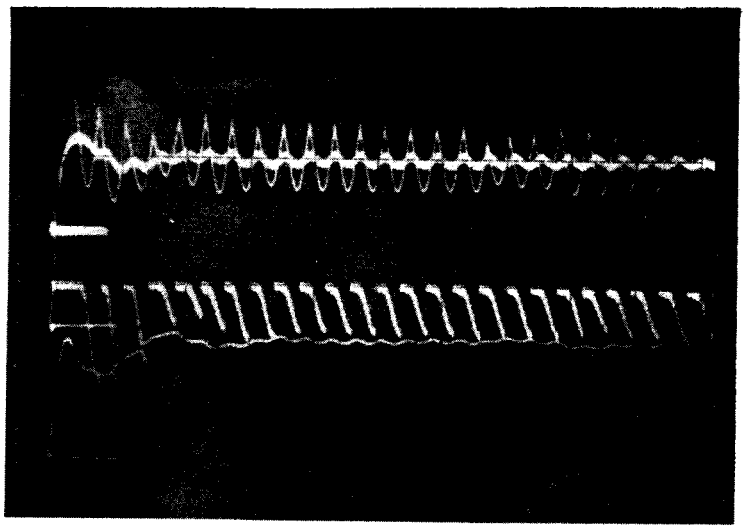
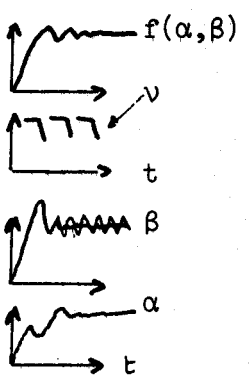


Figure III.9 - Organigrammes de principe

```
*DEB  Programme N°1
(001)C1:MCI,2A1 <initialisation des intégrateurs        } 1ère
        PHO=-1000 <initialisation de la temporisation   } macro
(002)HO:EAS,0A1,(ALP) <mesure de α
        BCI,(CAL) <Branchement au SP de calcul de ν     } 2ème
        SOA,0D1(NU) <sortie de ν                        } macro
        PHO=-5        <temporisation                    }
        RPI,(002)     <saut inconditionnel en (002)     }

*DEB PROGRAMME N°2
*COM REINITIALISATION DE P1
(003)E0:MCI,2A1 <initialisation des intégrateurs
        SYP,P1,(001) <repositionnement de HP1 en (001)
        RPI,(003)    <saut inconditionnel en (003)

*COM P1 EST LANCEE SUR APPEL C1
*COM C2 EST LANCEE SUR APPEL E0
*EXT(CAL)2002        <pour la transmission des paramètres
*EXT(ALP)2000 (NU)2001
*FIN                 <FIN des programmes hybrides

ALP : ;              <Déclarations
NU  : ;
CAL : CA ALP         <
      NRV RETURN     <TEST de α
      MP  K3         < α > 0
      AD  NU         < ν:= K3 α+ν
      RA  NU
      RTN HYB        <RETOUR au nombre hybride
 RETURN : RZ NU      <  α  <  0   ν  = 0
      RTN HYB        <RETOUR au moniteur hybride
```

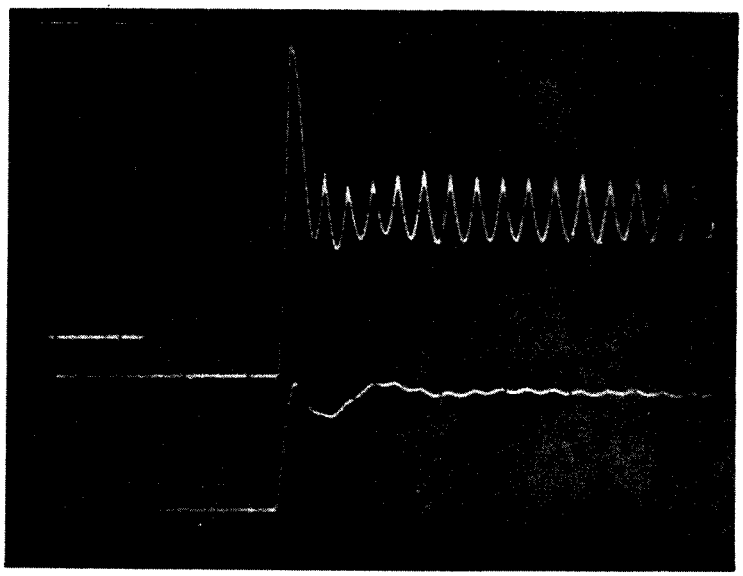

Figure III.10 - Listage des programmes

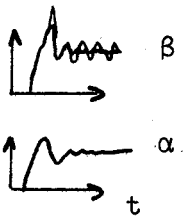




$\alpha_0 = \beta_0 = -0,5$

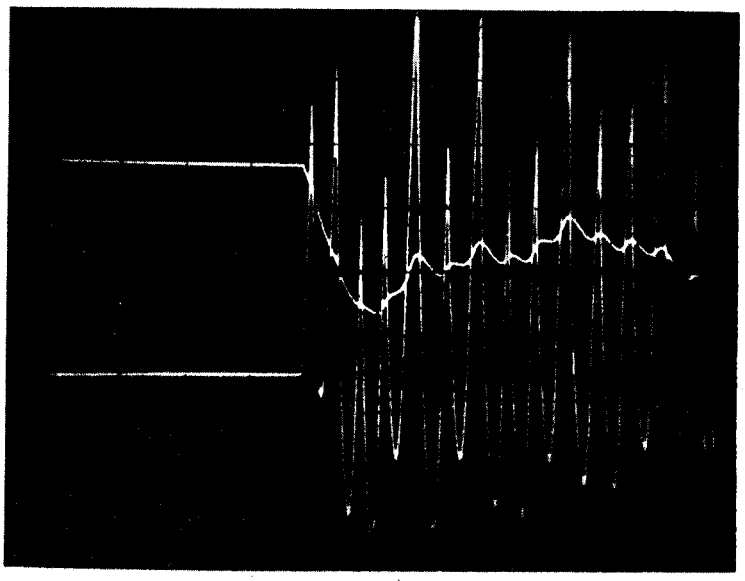
a) algorithme stable

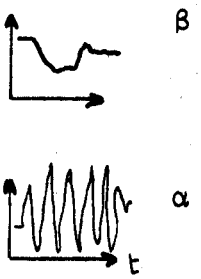




$\alpha_0 = \beta_0 = 0,5$

b) algorithme stables





$\alpha_0 = 0,5 \qquad \beta_0 = 0,5$

c) algorithme instable

Displays of α, β (20^V/cm), ν (2^V/cm), f (α, β) (5^V/cm).

Figure III.11

Cet exemple illustre trois caractéristiques d'utilisation du moniteur hybride et du langage associé.

Premièrement les équations décrivant l'algorithme sont relativement compliquées. Le cablage du panneau analogique permet de les "programmer" simplement et d'une façon systématique. Cette technique autorise en particulier, par bouclage, la description des intéractions entre variables. Le calcul portant ainsi simultanément sur toutes les variables est trés rapide et il n'apparait pas de difficultés dues au cumul des erreurs de discrétisation.

Deuxièmement toutes les variables étudiées sont directement accessibles et leur visualisation ne demande aucune procédure particulière Cette caractéristique associée à la possibilité d'itéra tion des calcul facilite l'appréhension des phénomènes étudiés.

En dernier lieu la notion de multiprocesseur permet au niveau de l'analyse, de décomposer le problème en plusieurs unités logiques indépendantes. Ici l'une réalise la simulation d'ensemble, l'autre la réinitialisation générale.

III.3 - Résolution hybride d'un problème régi par un modèle du type "à constantes réparties"

Nous étudierons dans cette partie un problème trés simple régi par une équation aux dérivées partielles parabolique de la forme suivante :

$$\frac{\partial^2 \phi}{\partial x^2} + \alpha \frac{\partial \phi}{\partial x} = \beta \frac{\partial \phi}{\partial t} \qquad (9)$$

$\alpha, \beta, x, \phi \in \mathbb{R}$, $t \in \mathcal{T}$, $\mathcal{T} = [0, +\infty[$

cette équation comporte 2 variables indépendantes, une variable d'espace x, et le temps t. La résolution d'un système de cette nature peut être envisagée selon 4 méthodes .

i) Tout d'abord, dans des cas trés particuliers on peut envisager une résolution formelle en continue en utilisant la transformée de Laplace à 2 variables /28/. Il peut être également possible de construire un modèle électrique à 2 dimensions constitué de condensateurs et de résistances. Ceci implique en fait le choix de capacités trés faibles associées à une échelle de temps trop rapide. Cette technique ne peut donc être généralement utilisable.

ii) On peut envisager de discrétiser la variable d'espace.
Dans ce cas, on obtient un système d'équations différentielles linéaires ou non linéaires selon la nature des coefficients α et β. Un tel modèle ne peut en fait être valable qu'au sens d'une approximation. Le schéma de cablage analogique est alors obtenu par la répétition du cablage d'une cellule de base connectée généralement en amont et en aval. Cette technique ne permet donc pas d'isoler une cellule analogique et d'envisager une résolution itérative hybride. Dans ce cas, il apparait de toute évidence qu'une fonction du nombre restreint d'opérateurs simultanément disponibles sur un calculateur analogique, il ne peut être possible de résoudre un grand nombre de ces cellules./29

iii) En discrétisant à la fois x et t il est possible de définir un modèle récurrent et d'envisager une résolution purement numérique.

iv) En dernier lieu si l'on cherche à discrétiser le temps t, on peut obtenir un modèle décrit par un système différentiel par rapport à la variable x.
Dans ce cas, il est possible dans un premier temps d'envisager une solution analogique également par cablage répétitif d'une cellule de base. Mais il apparait ici que chaque cellule de base de rang i peut être isolée au sens ou son évolution dynamique en dépend que des informations captées sur la cellule de rang $i-1$.

Sur ce dernier principe nous proposons donc de résoudre le système (9) à partir de la gestion numérique d'un sous programme analogique affecté à la résolution d'une cellule selon une technique itérative. Cette technique constitue une extension des travaux de Jury /30/ concernant l'utilisation des "mémoires analogiques continues".

III.3.1 - Description du problème étudié.

Nous proposons à titre d'illustration de résoudre un problème de conduction thermique usuel mieux connu sous le nom de "choc thermique". Il s'agit essentiellement d'étudier par simulation , la dynamique de la température θ d'un mur d'épaisseur constante soumis à un échelon de température à partir d'un état initial supposé en équilibre.

La loi de Fourier exprimant le principe de base de la conductibilité thermique conduit à exprimer l'évolution dynamique de la température d'une masse sous la forme du modèle suivant :

$$\text{Div}\left[\lambda \cdot \text{Grad}\theta\right] = \rho c \frac{\partial \theta}{\partial t} \qquad (10)$$

ρ et c sont des constantes désignant respectivement la masse volumique du mur, et sa chaleur massique.
Le coefficient de conductibilité λ est en général fonction de la température θ .
Dans un premier temps et pour simplifier l'exposé λ est supposé constant
Nous supposerons ensuite λ fonction non constante de la température.
On suppose le mur de dimension infini, ce qui permet d'étudier le problème dans la seule dimension relative à l'épaisseur du mur, en négligeant les effets limites.

Dans ces conditions l'équation (10) peut être écrite sous la forme simplifiée suivante :

$$\begin{cases} \lambda = a\theta + b \\ b \dfrac{\partial^2\theta}{\partial x^2} + a \left[\dfrac{\partial\theta}{\partial x}\right]^2 = \rho c \dfrac{\partial\theta}{\partial t} \end{cases} \tag{11}$$

ou si λ est constant :

$$\begin{cases} \lambda = b = \text{constante} \\ b \dfrac{\partial^2\theta}{\theta x^2} = \rho c \dfrac{\partial\theta}{\partial t} \end{cases}$$

$$\Longleftrightarrow \begin{cases} \lambda = b \\ \dfrac{\lambda}{\rho c} \dfrac{\partial^2\theta}{\partial x^2} = \dfrac{\partial\theta}{\partial t} \end{cases} \tag{12}$$

Selon que l'on envisage la résolution dans le sens rétrograde ou dans le sens normal d'évolution du temps la discrétisation de la variable t dans l'équation (12) conduit aux deux modèles (13) et (14)

$$\frac{\lambda}{\rho c} \frac{d^2\theta_i}{dx^2} = \theta_i - \theta_{i-1} \qquad \text{t croissant} \tag{13}$$

$$\frac{\lambda}{\rho c} \frac{d^2\theta_{i+1}}{d\,x^2} = \theta_{i+1} - \theta_i \qquad \text{t rétrograde} \tag{14}$$

Nous supposerons les changements de variables effectués.
En conséquence θ représente la température en variables réduites évoluant dans l'échelle $[0,1]$: $\theta \in [0,1]$, $x \in [0,1]$

Les conditions limites variables à un instant t_i donné sont donc sur la surface du mur $\theta(1,t_i) = 1$ et à l'intérieur du mur $\theta(x,t_i) = 0$ $\forall x \in [0,1]$.
Les équations différentielles (13) ou (14) étant du second ordre il est nécessaire de se donner aussi une condition initiale sur la dérivée $\frac{d\theta}{dx}$. Celle-ci n'est pas donnée à priori par les conditions limites, il est donc nécessaire de prévoir un algorithme de recherche paramétrique de cette variable.

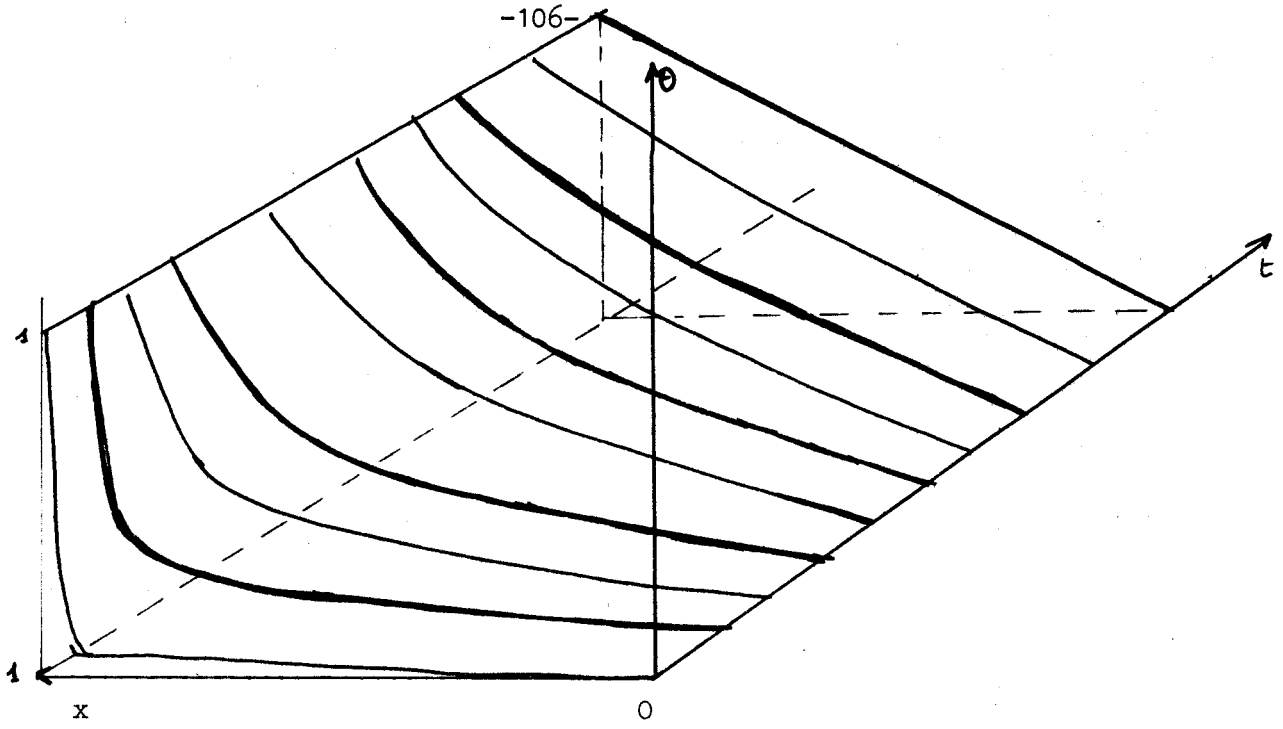
III.3.2 - Choix d'un mode de résolution

Le choix du modèle de simulation de la cellule de base dans le sens des équations (13) ou (14), peut être déduit d'une étude de stabilité du système décrit par l'équation aux dérivées partielles (9).

En effet, l'étude du régime limite permet de conclure d'une part à la stabilité de ce régime , d'autre part d'en définir dans le cas simple qui nous intéresse une expression évidente :

$$\lim_{t\to\infty} \theta(x,t) = x \quad \begin{cases} \theta(0) = 0 \\ \\ \theta(1) = 1 \end{cases}$$

L'allure des solutions peut être représentée à titre indicatif dans l'espace (θ,x,t) sous la forme suivante :



Allure des solutions $\theta(x,t)$

Si l'on s'intéresse tout d'abord à une résolution à partir de l'équation de base (14). Dans ce cas la résolution itérative s'éffectue dans le sens rétrograde ; ici à partir du régime limit $t\to\infty$ vers le régime initial $t=0$. Une telle méthode présente alors deux inconvénients majeurs. Tout d'abord la résolution correspond en fait à celle du système rétrograde instable :

$$\frac{\partial^2\theta}{\partial x^2} = -\frac{\lambda}{\rho c}\frac{\partial\theta}{\partial t}$$

ceci implique donc une amplification inévitable des erreurs cumulées du fait de la discrétisation. De plus, l'instabilité de ce système ne permet pas de fixer de manière précise la condition limite $\theta(x,0)$.

Une résolution basée sur l'utilisation du modèle (13) permet au contraire de tenir compte des conditions limites. Dans ce cas le comportement dynamique du modèle tend nécéssairement, du fait de la stabilité, vers le régime limite de l'équation aux dérivées partielles initiales. Dans ce cas, la stabilité du système étudié implique nécessairement la décroissance de l'erreur de discrétisation avec le temps.

III.3.3 - Méthode de résolution

Nous proposons donc pour résoudre une cellule de base de simuler l'équation différentielle (13) en recherchant préalablement la condition initiale portant sur la dérivée $(\overset{o}{\theta}(0,t_i))$.

La solution obtenue $\theta(x,t_i)$ est enregistrée pour servir de condition initiale au pas suivant de résolution à l'instant t_{i+1}.

La solution de l'équation différentielle (13) est définie de manière unique à partir des conditions aux limites $\theta(0,t_i)$, $\theta(1,t_i)$. Une résolution itérative conduit à effectuer une recherche paramétrique de la condition initiale $\overset{o}{\theta}(0,t_i)$. La contrainte à respecter sur la condition finale $\theta(1,t_i)$ permet alors de valider ou de rejeter la solution obtenue.

L'organigramme de la figure (III.12) illustre cette méthode.

La recherche de la valeur initiale de la dérivée sur le pas d'itération courant est suivie d'une phase d'enregistrement. Le pas suivant est ensuite initialisé et lancé.

III.3.4 - Simulation analogique

La cellule de base est décrite par une équation différentielle du deuxième ordre. Sa simulation est donc obtenue par cablage des deux intégrateurs (figure III.13) dont le mode de calcul est géré par le processeur digital.

La durée de résolution est fixe et est commandée à partir de l'horloge temps réel du calculateur numérique.

Le test de la valeur finale : $1 - \varepsilon < \theta < 1 + \varepsilon$ est effectué en fin de résolution par la lecture de l'état de comparateurs analogiques. Ces derniers reçoivent en permanence d'une part la contrainte $\theta(1,t) = 1 + \varepsilon$ $\forall t$, d'autre part la valeur au pas de rang i de $\theta\ (1,t_1)$. La valeur de ε trés petite devant l'unité est d'autant plus faible que l'incrément portant sur $\overset{\circ}{\theta}\ (0,t_i)$ lors de sa recherche est faible.

III.3.5 - Programmation hybride

Le programme hybride décrit par les figures (III.12) et III.14) assure deux fonctions essentielles.

Premièrement pour chaque pas de résolution il gère d'une part la recherche de la condition initiale sur $\overset{\circ}{\theta}$ et d'autre part l'enregistrement des valeurs de la température θ.

Deuxièmement il assure le passage d'un pas à l'autre.
Pour un pas donné de rang i deux tables sont nécessaires. La première contient les valeurs de la température $\theta(x,i-1), x \in [0,1]$ enregistrées au pas précédent . Ces valeurs sont restituées au niveau du panneau analogique ou elles sont nécessaires au calcul de $\theta(x,i)$. Ces valeurs correspondent

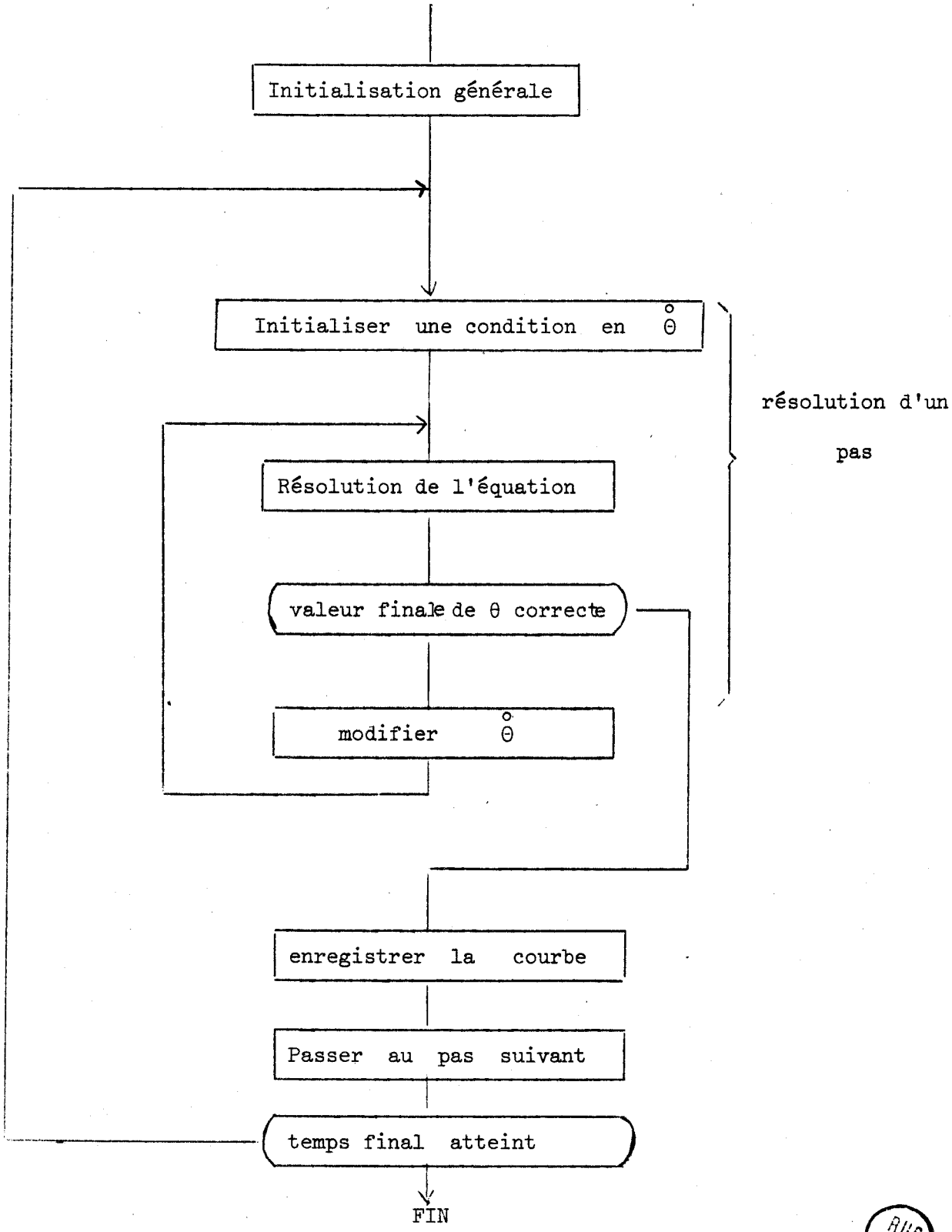
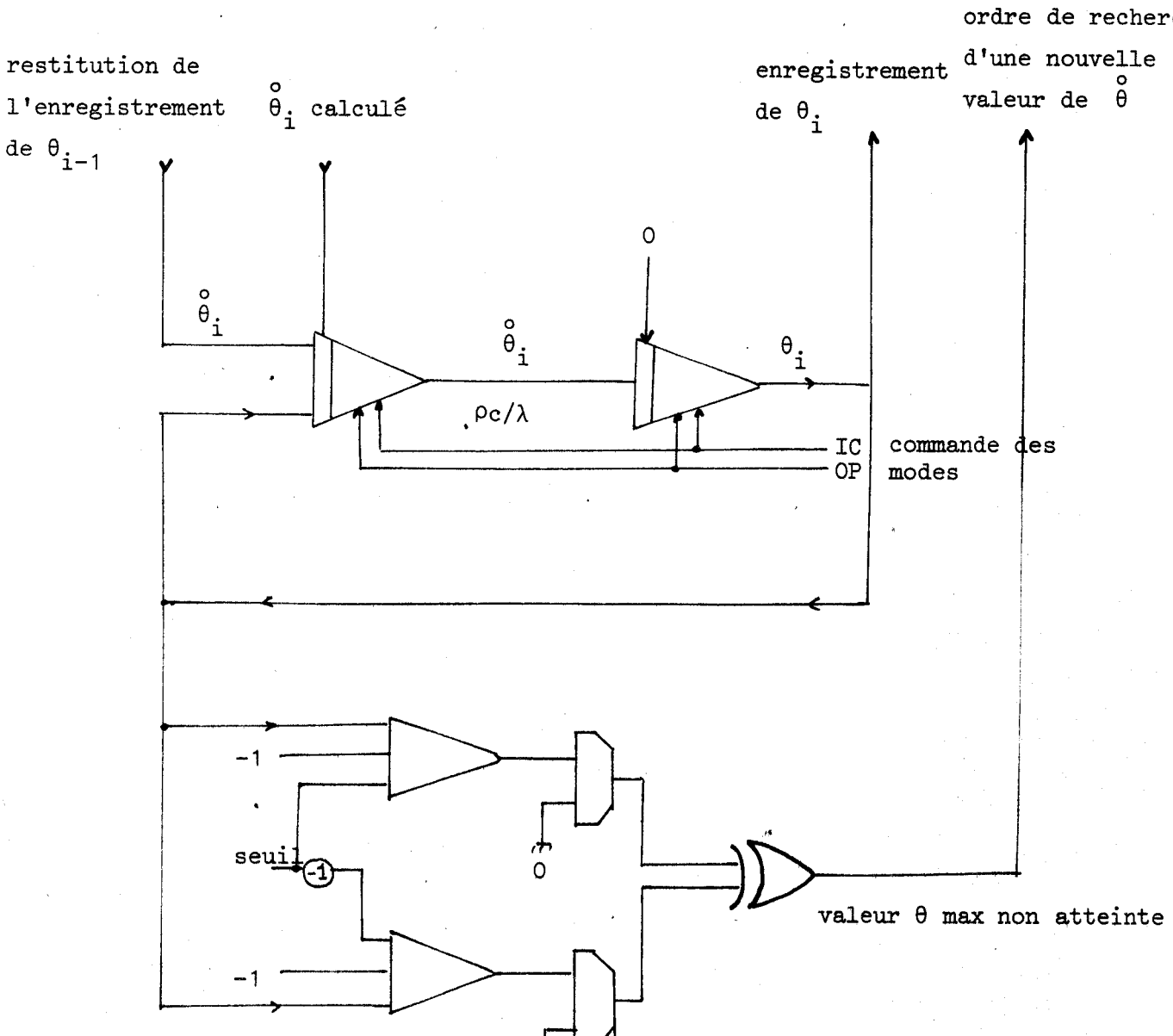


Figure III.12 - Organigramme de principe



élaboration de $1 - \varepsilon < \theta_i < 1 + \varepsilon$

Figure III.13 - Schéma de cablage

aux informations transmises d'un pas à l'autre. La deuxième table reçoit les enregistrements des valeurs de $\theta(x,t_i)$, valeurs qui seront transmises à leur tour au pas suivant i+1. Il est donc possible de n'utiliser que deux tables travaillant en bascule ou flip-flop.

Le passage d'un pas à l'autre consiste donc d'une part à effectuer l'enregistrement et la visualisation de la températeure lorsque les conditions initiales sont correctes,et d'autre part à gérer le basculement des tables.

III.3.6 - Résultats

L'enregistrement III.1.5 illustre les résultats obtenus par cette méthode. La température en ordonnée évolue en fonction de l'abcisse dans le mur, le choc thermique venant de la droite. La famille de courbe a pour paramètre le temps.

La solution a été obtenue en vingt secondes. Il est possible d'accélérer la résolution en maximisant, en particulier, la fréquence d'échantillonnage.

```
* DEB
* COM INITIALISATION GENERALE
  (010) EO : PIO = 31              <RAZ DE LA TABLE (WWO)
  (011)    : EAS,0A8,(WWO)         <PAR ENREGISTREMENT DE LA
             INC,PIO               <VALEUR
             RPC,PIO, (011),(044)
  (044)    : ETY,(YPP)             <1° INITIALISATION DE Ŷ°
             ETY,(EPS)             <INCREMENT SUR Ŷ°
  (046)    : PJ1 = +1              <PARAMETRE DE FLIP-FLOP DES TABLES
* EXT(CAL) 2003                    <POUR LA TRANSMISSION DE PARAMETRES
* EXT(EPS) 2001 (YPP) 2002         <VERS LE SP DE CALCUL
* COM REALISATION D'UN PAS DE RESOLUTION
  (047)    : BCI,(CAL)             <CALCUL DE Ŷ° : Ŷ° = Ŷ° + EPS
             FET,(WWO)
             FET,(WW1)
             SOA,0D2,(YPP)         <SORTIE DE Ŷ°
             SAV,0D1,+0            <SORTIE DE Y
             PHO = -10
             MCI,2A1               <MISE EN CONDITION INITIALE
             PIO = -20             <NOMBRE D ECHANTILLONS SUR Y
  (050) HO : MOP,2A1               <LANCEMENT DES CALCULS
  (051)    : PHO = -10             <DUREE ENTRE 2 ECHANTILLONNAGES
  (052) HO : RPC,PJ1,(054),(056)   <SUIVANT FLIP-FLOP (054) OU (056)
  (054)    : EAS,0A1,(WW1)         <ENREGISTREMENT
             SOA,0D1,(WWO)         <RESTITUTION
             RPI,(060)
  (056)    : EAS,0A1,(WWO)         <IDEM AVEC SI AUTRE TABLE
             SOA,0D1,(WWO)
  (060)    : INC,PIO
             RPC,PIO,(051),(062)   <BOUCLAGE SUR N ECHANTILLONNAGES
  (062)    : PHO = -10
  (064) HO : MHD,2A1               <X = 1
             LCO,2DN               <LECTURE DU COMPARATEUR
             RPC,PJ7,(047),(065)   <PJ7 SI θ =  θMAX
```

```
* COM          TRACE ET ENREGISTREMENT DE LA TEMPERATURE
  (065) : FET,(WWO)                    <REINITIALISATION DE LA DERNIERE RESOLUTION
          FET,(WW1)
          SOA,OD2,(YPP)
          SAV,OD1,+000
          PHO = -50
          MCI,2A1
          PIO = -20
  (066)HO: MOP,2A1                     <RELANCER LA DERNIERE RESOLUTION
          INF,2D9                      <ORDRE D'ENREGISTREMENT
  (067) : PHO = -10
  (070)HO: RPC,PJ1,(072),(074)         <FLIP-FLOP TABLE
  (072) : EAS,OA1,2000                 <ENREGISTREMENT
          SOA,OD1,(WWO)
          RPI,(076)
  (074) : EAS,OA1,2000
          SOA,OD1,(WW1)
  (076) : INC,PIO
          RPC,PIO,(067),(101)
  (101) : PHO = -10
  (102)HO: IND,2D9                     <FIN ORDRE D'ENREGISTREMENT
* COM     PASSAGE AU PAS SUIVANT
          MCI,2A1
          RPC,PJ1,(104),(106)          <REALISATION DU FLIP-FLOP DES TABLES
  (104) : PJ1 = +0
          RPI,(047)
  (106) : PJ1 = +1
          RPI,(047)
* FIN
```

Figure III.14 - Programme hybride

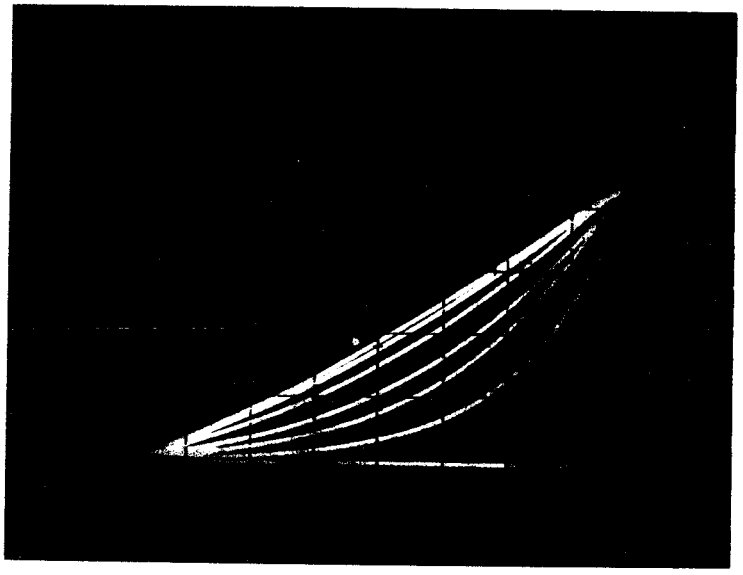


Figure III-15

Evolution de la température en l'intérieur du mur.

III.3.7 - Extension au cas λ non constant

Nous envisagerons deux cas, le cas où λ est une fonction affine de la température et celui où λ prend plusieurs valeurs distinctes.

Le premier cas correspond à un mur non homogène dont la conductibilité thermique est de la forme :

$$\lambda = a\theta + b$$

Dans ces conditions la loi de Fourier s'exprime par l'équation (11)

$$b \frac{\partial^2 \theta}{\partial x^2} + a \left[\frac{\partial \theta}{\partial x}\right]^2 = \rho c \frac{\partial \theta}{\partial x} \qquad (11)$$

qui aprés discrétisation sur le temps devient

$$\frac{b}{\rho c} \frac{d^2 \theta_i}{d x^2} + \frac{a}{\rho c} \left[\frac{d\theta}{dx}\right]^2 = \theta_i - \theta_{i-1} \qquad (15)$$

Par rapport au cas ou λ est constant , cette équation contient un terme supplémentaire en $\frac{d\theta}{dt}$ mais n'utilise pas d'autre valeur de θ que celles aux instants t_i et t_{i-1}. La simulation analogique est donc modifiée ainsi que l'indique la figure III-15. Par contre la partie hybride de la simulation est identique.

Le fait que cette modification ne porte que sur la simulation analogique se généralise aux cas où λ est une fonction non linéaire de la température.

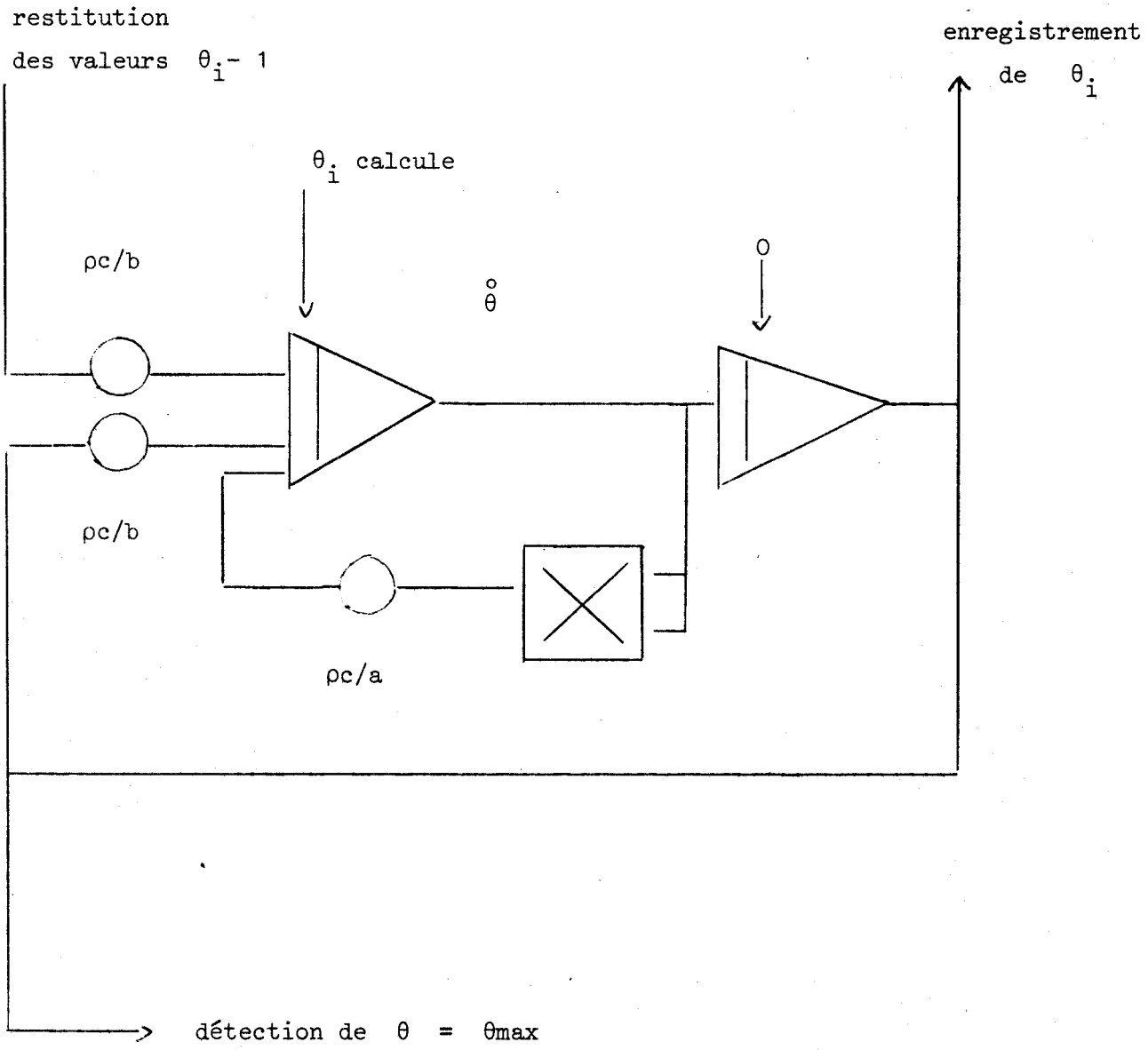


Figure III-15 - Schéma de cablage :: $\lambda = a\theta + b$

Le cas où le mur est composé de plusieurs couches de conductibilité thermique différente peut se résoudre au niveau des commandes. En effet, l'évolution de la variable X est assimilée à l'évolution du temps.
Cette évolution étant mesurée par l'horloge temps réel du calculateur numérique, il est possible de commander aux instants proprices les changements de valeur de la constante λ en commutant les atténuateurs. De plus cette commande peut être rendue indépendante du programme hybride général en utilisant la possibilité de programmation parallèle offerte par le multiprocesseur.

En conclusion la discrétisation sur le temps seul, permet par le calcul hybride de simuler un choc thermique quelque soit la nature, linéaire ou non, du coefficient de conductibilité thermique.

CONCLUSION

Ces trois exemples ont permis d'illustrer les possibilités offertes par le moniteur hybride et la facilité de mise en oeuvre due à l'utilisation d'un langage de simulation spécialisé.

L'intérêt principal de ces techniques réside en la possibilité de décomposer le problème étudié en plusieurs unités logiques grâce au multiprocesseur.
Chacune de ces parties peut elle-même être scindée en plusieurs unités de programmation qui chacune utilise l'outil le mieux adapté à sa description ou sa simulation : la programmation hybride ou le cablage analogique.

Le deuxième intérêt de cette méthode est sa facilité et sa rapidité de mise en oeuvre. L'implémentation qu'elle se fasse par cablage ou par programmation n'est que le codage immédiat des résultats fournis par l'analyse formelle du problème à traiter.

Notons enfin que la facilité d'exploitation des techniques hybrides autòrise l'abord de problème délicat, ce qu'illustre le troisième exemple relatif au traitement des équations aux dérivées partielles.

CONCLUSION GENERALE

Le système de simulation présenté constitue sur le plan matériel et logiciel un outil remarquablement adapté à l'étude des processus. Il permet une mise en oeuvre directe des méthodes d'analyse et de synthèse des systèmes sans les restrictions généralement dues aux langages utilisés ou à la spécificité des matériels utilisés.
L'utilisation de deux calculateurs de natures différentes et d'un multiprocesseur permet de répartir les taches tout en adaptant au mieux les opérateurs disponibles qu'il s'agisse des éléments logiques, analogiques ou des fonctions digitales.

La console analogique donne au système la possibilité d'une simulation en temps réel, accéléré ou ralenti des processus d'évolution continue. Les opérateurs hybrides associés aux possibilités de mémorisation , de calcul arithmétique et de décision du mini calculateur permettent d'agir à tout moment sur l'évolution de ce processus. Enfin la conception modulaire des programmes permet d'étendre et de modifier au gré des utilisateurs la bibliothèque des opérateurs.

Les domaines d'applications d'un tel système sont multiples.
En effet la possibilité offerte par le calculateur analogique d'un traitement parallèle de l'information peut dans un premier temps permettre de décharger le calculateur digital des algorithmes d'intégration et de résolution des systèmes différentiels. Le calculateur analogique est tout à fait adapté aux traitements de cette nature. Un autre domaine d'utilisation, extrêmement important, concerne les problèmes relatifs à la simulation et au contrôle des processus. Le logiciel hybride a permis d'étendre trés sensiblement le champs potentiel des utilisateurs possibles. En effet l'appréhension du langage hybride est relativement rapide, et conduit à mettre rapidement ce matériel à la portée d'utilisateurs non spécialisés. La recherche y trouve un outil d'investigation souple et puissant, l'enseignement le moyen d'illustrer les

problèmes généraux relatifs à l'étude des processus et systèmes complexes et l'industrie un outil parfaitement adapté au contrôle temps réel.

Il convient cependant de noter les limites d'utilisation matérielle d'un tel ensemble. En effet la fréquence maximale des événements traités pendant un intervalle de temps donné est de l'ordre du kHz, la précision des opérateurs analogiques est de l'ordre de 10^{-3}. Les matériels plus récents qui sont actuellement en cours de couplage devraient permettre d'améliorer trés sensiblement la précision et la fréquence maximale de traitement des déclenchements.
Le système actuel donne entière satisfaction sur le plan de la recherche mais aussi de l'enseignement.
Dans ce sens diverses séances de travaux pratiques de simulation dans des domaines trés variés tels que :

- Le génie chimique
- L'électrotechnique
- Le transport
- L'automatique
- L'informatique

sont actuellement réalisées et régulièrement exploitées.

B I B L I O G R A P H I E

[1] JACKSON, A.S. -"*Analog Computation*"- Mc Graw-Hill - 1960.

[2] FIFER, S. -"*Analogue Computation : Theory, Techniques and Applications*"- 4 Vols. - Mc Graw-Hill - New York - 1961.

[3] TOMOVIC, R. and KARPLUS, W.I. -"*High-Speed Analog Computers*"- John Wiley - New York - 1962.

[4] Electronic Associates, Inc - "*Introduction to hybrid computations*" - Electronics Associates, Inc. - Long Branch, N.J. - 1963.

[5] CONNELLY, M.E. - "*Real-Time Analog-Digital Computation*" - IRE Transactions on Electronic Computers - EC-11, 31-41 - Feb. 1962.

[6] BEKEY, G.A. et KARPLUS, W.J. - "*Hybrid Computation*" - Wiley Int. Ed. - 1968.

[7] LAURENT, F. - "*Les machines hybrides et leurs applications en Automatique*" - Seconde thèse Doctorat ès Sc. Ph. Univ. de Lille I - Juin 1968.

[8] GREEN, C. - D'HOOP, H. and DEBROUX, A. - "APACHE-A *Breakthrough in Analog Computing*" - IRE Transactions on Electronic Computers - EC-11 - 699-706 Oct. 1962.

[9] LINEBARGER, R. and BRENNAN, R.D. - "*A survey of digital simulation*" - Simulation - 3, 22-37 - December 1964.

[10] TIECHROEW, D. - LUBIN, J.F. - "*Discussion of Computer Simulation Techniques and Comparison of Languages*" - Simulation - 9 - 181-190 - Oct. 1967.

[11] McGHEE, R.B. and LEW, A.Y. - "*Software for Hybrid Computers*" - Simulation 5 - 367-374 - Dec. 1965.

[12] Electronic Associates, Inc. - "*HYTRAN Operation Interpreter*" - Pub. No. 07 800 0008-1 - 1967.

[13] Electronic Associates, Inc. - "*HYTRAN Simulation Language Programming Manuals*" - Pub. No. 07 800 0006 - 1967.

[14] KOVACS, J. and STRAUSS J. - "*An Approach to Hybrid Programming Language*" - Sci Third Annual Simulation Sofware Meeting - St Louis - Mo

[15] MARQUETTE, M. et GENTINA, J.C. - "*Exemple d'une gestion par minicalculateur d'un ensemble de simulation hybride*" - Mini and micro computers and their applications - 31 Mai , 3 Juin 1976 - Zurich Suisse.

[16] DUMAS, J.M. - "*L'organiphase : un modèle facilitant l'analyse et la synthèse des automatres logiques*". - Thèse de Docteur de Spécialité Montpellier - Avril 1974.

[17] DUMAS, J.M. - PRUNET, F. - "*Introduction à la modélisation naturelle des structures de commandes : l'organiphase*". - Revue RAIRO (J2 - juillet 1974.

[18] DUMAS, J.M. - PRUNET, F. - "*Une représentation parallèle du cahier des charges d'automates industriels*" - Revue Automatisme - Mai 1975.

[19] DUMAS, J.M. - PRUNET, F. - "*Un modèle d'analyse et de synthèse des systèmes de contrôle*" - Extrait de AFC.1.

[20] GIRARD, P. et NASLIN, P. - "*Construction des machines séquentielles industrielles*" - Dunod 1974.

[21] GIRARD, P. et NASLIN P. - "*Nouvelle méthode de synthèse des automates séquentiels*" - Extrait de AFC.1.

[22] PATIL, S.S. - DENNIS, J.B. - "*The description and realization of digital systems*" - Revue RAIRO - J1 Fev. 1973.

[23] TOURRES, L. - "*Représentation des automatismes par réseaux de Pétri*". Document E.D.F. (Direction des Etudes et Recherches - Septembre 1975.

[24] LAURENT, F. - MELIN, C. - TOULOTTE, J.M. - LUCQUIN - MONTASTIER, J. - "*Analyse sur calculatrice hybride des courbes d'évolution de populations bactériennes*" - Congrés AICA/IFIP - Munich - Aout 1970.

[25] DANTZIG - "*Constructive proof of the minimax theorem*" - Pacif. Journal of Math - 6 - 1956.

[26] BOUDAREL, J. - DELMAS, J. - GUICHET, P. - "*Commande optimale des processus*" - T2 Dunod - Paris - 1968.

[27] KUHN, H.W. - TUCKER, A.W. - *"Non linear programming"* - Proc 2nd - Berkeley symposium on Math. Stat. and Prob. - pp; 481-492 - U. Cf. Calif. Press. Berkeley - 1951.

[28]

[29] KARPLUS, W. J. - *"Analog Simulation : Solution of Field Problems"* - McGraw-Hill - New York - 1958.

[30] JURY, S.H. - *"Solving Partial Differential Equations"* - Industrial and Engineering Chem. - 53 - 177-180 - 1961.

Annexe 1

LOGICIEL DE SIMULATION T 1000 / AD 32

APPEL DU SUPERVISEUR

Le superviseur hybride se compose de 4 programmes :

- la bande d'appel implantée dans le bloc 0 de 6 000 à 6 100
- le programme de gestion
- le compilateur composé de deux parties :
  - ASH BT traduit le langage symbolique en binaire
  - ASH CB remplace les étiquettes par leur valeur
- le moniteur hybride SH

UTILISATION :

La plaque logique de l'AD 32 doit être déconnectée. ⚠

- Charger la bande d'appel sous CHARP
  - . sortie du message :
    SOB < Appel superviseur hybride 1976 > 6 000, 6 100 ;
- I N I
- MARCHE
  - . sortie du message :
    superviseur hybride 1976
    LR ou TC ?

- TC — Le programme de gestion est en mémoire si on tape le programme sur le clavier du télétype
- LR — le superviseur attend l'écriture du programme si on place une bande sur le lecteur rapide ; l'écriture de LR déclenche la lecture de la bande

la réponse LR ou TC implante ASH BT en mémoire

- . sortie du message :
  DAC ASH
- GO — la première partie de l'assemblage est terminée ASH CB est en mémoire ; la deuxième partie de l'assemblage s'effectue
- . sortie du message :
  DIM PRO HYB : xxxx
  ETAT ? :
- XXXX — on démasque les appels qu'on utilise pour déclencher les instructions
  exemple : 0337 si les seules variables de déclenchement utilisées sont E0 et E1 le superviseur est alors en mémoire
- MARCHE — exécution du programme

On peut maintenant connecter la plaque logique de l'AD 32 ⊘

.../...

. Pour réinitialiser et relancer le programme faire :

ARRET, INI, MARCHE

. Pour faire un nouvel assemblage faire :

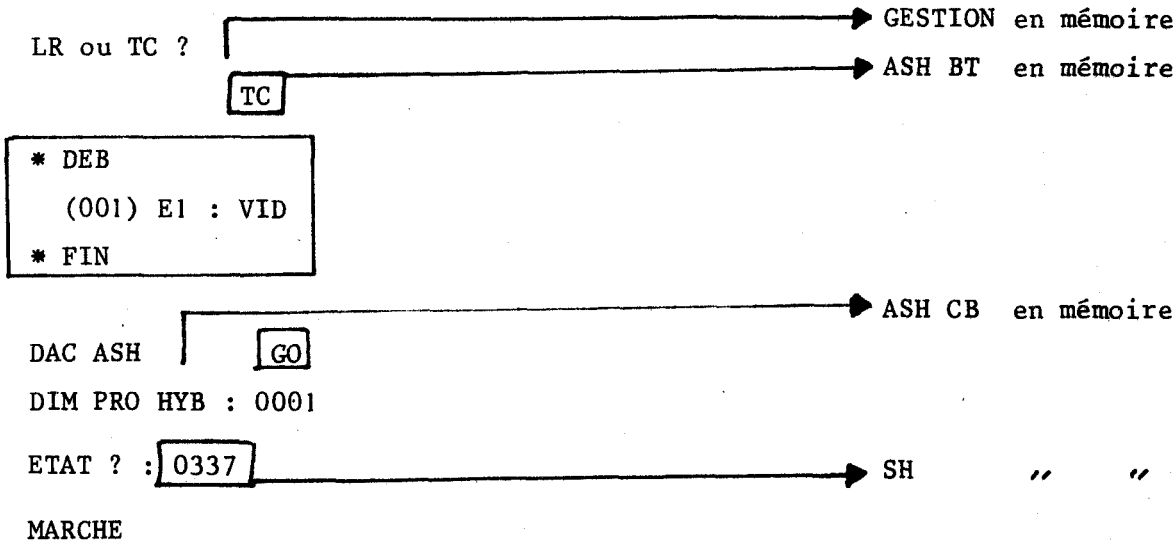
P = 6 000 000

MARCHE

SCHEMA D'UTILISATION

Sob < Appel superviseur hybride 1976 > 6 000, 6 100 ; bande d'appel sous CHARP

Superviseur hybride 1976



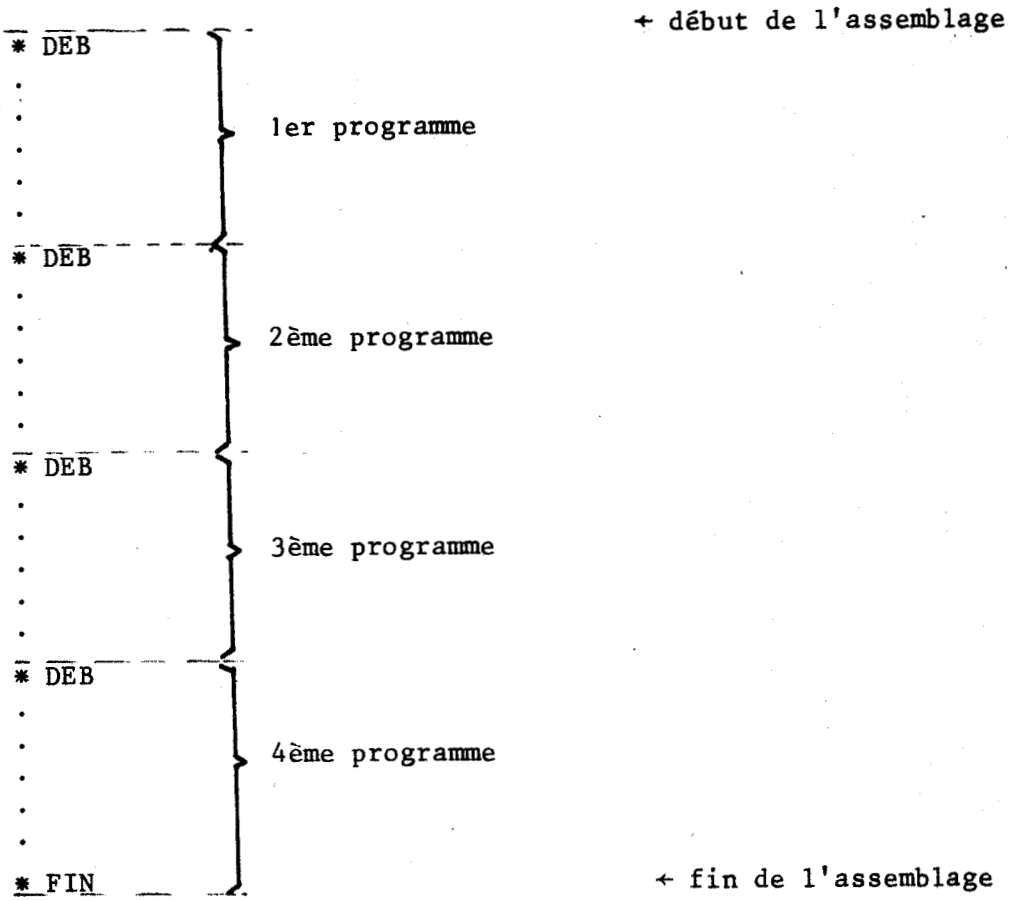
Pour relancer le programme : ARRET, INI, MARCHE

Pour refaire un assemblage : P = 6 000 000, MARCHE

ECRITURE DES PROGRAMMES

Il est possible de faire exécuter parallélement 4 programmes hybrides.
Chaque programme doit commencer obligatoirement par la pseudo-instruction * DEB.
Il se termine par la rencontre de * DEB (début d'un autre programme) ou par la pseudo-instruction * FIN pour le dernier programme.

.../...



ECRITURE DE COMMENTAIRES

Pour insérer un commentaire dans un programme on utilise la pseudo-instruction * COM. Elle indique que les caractères qui suivent jusqu'au retour chariot suivant sont des commentaires. Ils sont ignorés par le compilateur. La détection d'une pseudo-instruction * COM entraîne simplement un appel de la ligne suivante.
On ne doit toutefois pas dépasser 40 caractères par ligne.

ECRITURE DES INSTRUCTIONS

Chaque instruction est indépendante des autres et ne peut être déclenchée que par un "événement extérieur".
On peut cependant exécuter des séquences d'instructions ; la première instruction est déclenchée par une variable de déclenchement, les suivantes s'effectuent automatiquement et séquentiellement ; ces séquences sont appelées "macro-instructions".

Chaque instruction comprend :

- une variable de déclenchement précédée d'une étiquette
- le code de l'opération
- les opérandes nécessaires pour l'exécution dont le nombre dépend du type de l'opération ; il ne peut excéder quatre.

La fin de phrase est le retour chariot.

.../...

FORMAT GENERAL

(ETI) VD : OPE, opérandes RC.

ETI	étiquette (trois symboles entre parenthèses)
VD	variable de déclenchement (une lettre suivie d'un chiffre
:	séparateur
OPE	opération à effectuer (trois lettres)
,	séparateur
opérandes	sont séparés les uns des autres par une virgule
RC	retour chariot = fin d'instruction

Si l'instruction fait partie d'une macro-instruction on omet la variable de déclenchement.

ETIQUETTES

Elles sont constituées de trois symboles entre parenthèses.

Ces symboles sont quelconques à l'exception de :

retour chariot, espace, saut de ligne, <, ←, blanc.

Ils sont codés par troncature de leur code ASCII : on garde le bit 8 pour le premier et le troisième chiffre ASCII pour les suivants.

Table de correspondance

code octal :	1	2	3	4	5	6	7	0
symboles :	A	B	C	D	E	F	G	H
	I	J	K	L	M	N	Ø	P
	Q	R	S	T	U	V	W	X
	Y	Z						
	9							8

⚠ ne pas utiliser deux étiquettes qui correspondent aux mêmes valeurs ;
exemple : (ABC) et (IJK) sont la même signification
Pour éviter ce problème adopter la règle suivante :
toutes les étiquettes devront commencer par une lettre allant de A à H et ne comportant que 3 caractères pris parmi ces lettres.
exemple : (ADE) ≡ 145 (DEF) ≡ 056...

VARIABLES DE DECLENCHEMENT

Les interruptions (appels externes, appel d'une voie de comptage, de l'horloge temps réel, du bac d'entrée (sortie) aboutissent au codage d'une variable de déclenchement. C'est ce qui permettra, en cas de validation, l'exécution des instructions correspondantes.

Elles sont obligatoirement précédées d'une étiquette.

La liste des variables utilisables se trouve à la fin de la notice.

.../...

OPERATIONS

Le code symbolique se compose de 3 caractères

On peut utiliser les instructions suivantes :

- initialisation de paramètres
- incrémentation de paramètres
- ruptures
- sorties analogiques ; sorties sur télétype
- entrées analogiques ; entrées sur télétype
- fonctionnement des intégrateurs
  des track-store
  des interrupteurs et relais

La liste des opérations utilisables se trouve à la fin de la notice

OPERANDES

A chaque opération correspond certains opérandes, qu'on écrit après le code opération, en les séparant par une virgule

Ces opérandes peuvent être :

- des commandes de 1'AD 32 ou de 1'EAI
- des paramètres
- des étiquettes
- des adresses
- des valeurs décimales
- des valeurs octales

Les commandes

Elles sont relatives aux intégrateurs
track-store
interrupteurs, relais
sorties
entrées

Les paramètres

- PIO à PI7, PJO à PJ7, PH6 et PH7 sont des paramètres d'usage général ils sont initialisables et incrémentables
- PVO à PV7 sont relatifs aux voies de comptage.
  PVi est initialisable et est incrémenté automatiquement à chaque impulsion lancée sur la voie de comptage i ; l'annulation de PVi provoque un appel de la voie de comptage i et aboutit au codage de la variable de déclenchement Vi.
  Seules les voies de comptage 0, 1 et 4 sont utilisables.
- PH0 à PH5 sont relatifs à 1'horloge temps réel ; ils sont initialisables et sont incrémentés automatiquement à chaque appel lancé par 1'horloge temps réel.

.../...

L'annulation de PHi aboutit au codage de la variable de déclenchement Hi.

L'initialisation d'un paramètre se fait par une opération du type :

PIO = - 5 (valeur décimale)

PJ5 = 1716 (valeur octale)

L'incrémentation se fait par une opération du type : INC, PIO

Valeurs décimales

Les valeurs décimales sont comprises entre - 524 287 et + 524 287 ; elles sont codées sur 19 bits.

Elles s'écrivent : un signe suivi d'un groupe de un à six chiffres décimaux.

Exceptions : . pour l'instruction ETY (entrée sur TTY d'une tension), la valeur décimale doit être comprise entre - 9995 et + 9995 ; l'unité est le millivolt.

. pour l'instruction SAV (sortie d'une tension sur calculatrice analogique), la valeur décimale doit être comprise entre - 999 et + 999 ; l'unité est le centième de volt.

Valeurs octales

Les valeurs sont codées sur 12 bits.

Elles sont comprises entre 0000 et 7777.

Elles sont constituées par un groupe de 4 chiffres octaux.

TRAITEMENT DES ERREURS

A la détection d'une erreur il y a sortie du message :

Eij

ETI (---) + XY "instruction erronnée)

Eij (ij de 00 à 19) donne la signification de l'erreur

(---) est la dernière étiquette rencontrée

XY est le nombre d'instructions passées depuis la dernière étiquette.

Un coup de sonnette appelle l'écriture sur télétype de l'instruction corrigée. Au retour chariot il y a reprise de la compilation.

Le symbole ← introduit avant un retour chariot permet d'annuler la dernière instruction écrite ; la partie de phrase entre le dernier retour chariot et le symbole ← est ignorée du compilateur.

La liste des messages d'erreur et des corrections à apporter se trouve à la fin de la notice.

.../...

ECRITURE D'UNE OPERATION EN BINAIRE

On peut coder tout ou partie d'une instruction en binaire.

Une étiquette peut être d'abord déclarée en symbolique ; la variable de déclenchement peut être déclarée, soit en symbolique, soit en binaire.

La partie binaire de l'instruction doit être précédée par le symbole $ suivi du nombre de mots constituant l'opération (NB).

A l'exception du premier, l'écriture des paramètres se fait normalement

exemple :

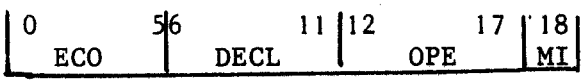
```
(002) V3 : RPC, PI1, 1234, (001)
     V3   code 63
    RPC   code 05
    PI1   code 51
```

on peut écrire :

```
(002) V3 : $3  5100 050, 1234, (001)
(002)    : $3  5163 054  1234, (001)
```

En mémoire, on trouve :

Premier mot :



ECO : code du premier paramètre
DECL : code de la variable de déclenchement
OPE : code de l'opération
MI : bit de macro-instruction

Mots suivants :

PSEUDO-INSTRUCTIONS

Elles s'écrivent * PSD

* symbole caractéristique des pseudo-instructions
PSD code de la pseudo-instruction (trois lettres)

Les pseudo-instructions ne sont utilisées que par le compilateur ; il n'y a donc pas de format binaire associé.

Les pseudo-instructions utilisables sont :

* DEB
* FIN
* COM
* EXT

.../...

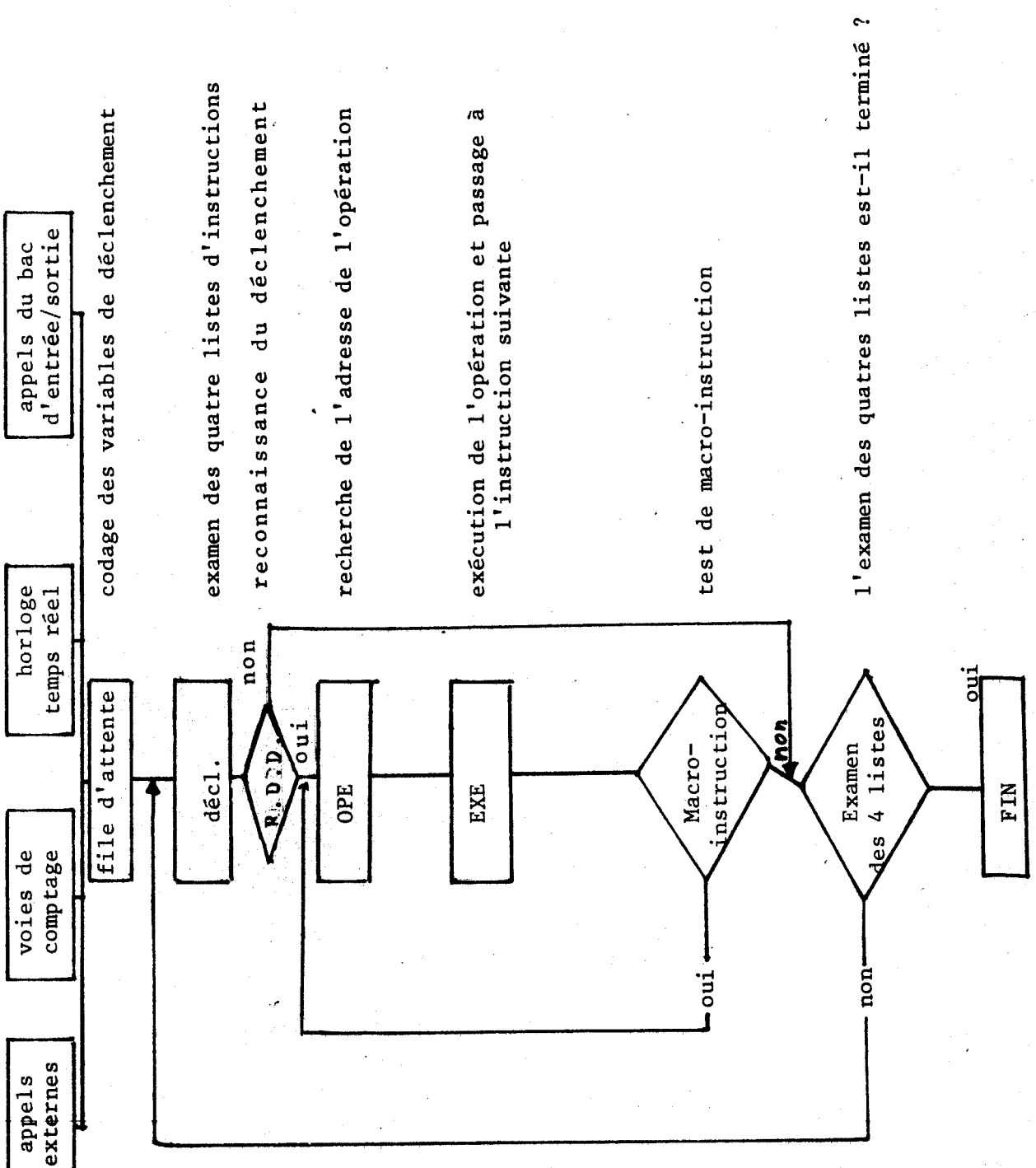
* EXT a pour but d'attribuer à des adresses des étiquettes compréhensibles par l'assembleur.

exemple :

* EXT (017) 4 122 commentaire (121) 0173 RC
* EXT (134) 6 000 (010) 6 012 (003) 6 473 RC

Une instruction se composant de 40 caractères au maximum, on ne peut déclarer que quatre étiquettes à la fois, à la condition de ne pas insérer d'espace.

EXECUTION D'UN PROGRAMME



.../...

Chaque interruption (appels externes, appel d'une voie de comptage, de l'horloge temps réel, du bac d'entrée (sortie) aboutit au codage de la variable de déclenchement correspondante. Cette variable est rangée dans une file d'attente à laquelle est associée un pointeur d'écriture et un pointeur de lecture.

A chacun des 4 programmes hybrides est associé un pointeur qui indique l'instruction à exécuter.

Lors de l'exécution le pointeur de lecture de la file d'attente indique la variable de déclenchement à considérer.

On compare cette variable aux variables correspondant aux instructions indiquées par les pointeurs des 4 programmes hybrides.

Si la comparaison est positive on exécute les instructions et les pointeurs des programmes hybrides ainsi que celui de la file d'attente sont incrémentés.

Si la comparaison est négative on incrémente seulementle pointeur de lecture de la file d'attente.

On ne peut donc pas lancer les appels pour coder les variables de déclenchement utilisées dans le programme dans un ordre quelconque ; il faut réaliser une séquence d'appels qui permet l'exécution de tout le programme.

<u>exemple</u> :

```
* DEB
   (001) E0 : ----
              :
              :          1ère macro-instruction
              :
   (002) E1 : ----
              :          2ème macro-instruction
              :
              :
* FIN
```

la séquence E1, E0 ne permet l'exécution que de la 1ère macro-instruction.

la séquence E1, E0, E1 permet l'exécution de tout le programme.

Une même variable de déclenchement peut être utilisée plusieurs fois au cours du même programme

<u>exemple</u> :

```
* DEB
   (001) E0 : ----
              :
              :
   (002) E0 : ----
              :
   (003) E1 : ----
              :
              :
   (004) E0 : ----
              :
              :
* FIN
```

pour exécuter le programme il suffit d'envoyer les appels dans l'ordre : E0, E0, E1, E0.

.../...

PARAMETRES

Code symbolique	code octal	
PH0	00	
PH1	01	
PH2	02	horloge temps réel
PH3	03	
PH4	04	
PH5	05	
PV0	60	voies de comptage AD32
PV1	61	
PV4	64	voie de comptage EAI
PI0	10	
PI1	11	
PI2	12	
PI3	13	usage général
PI4	14	
PI5	15	
PI6	16	
PI7	17	
PJ0	20	
PJ1	21	
PJ2	22	
PJ3	23	usage général
PJ4	24	
PJ5	25	
PJ6	26	
PJ7	27	
PH6	06	usage général
PH7	07	

.../...

OPERANDES

code symbolique	code octal	borne de cablage	
2A1	06	2A1	
2A2	04	2A2	
2A3	02	2A3	intégrateurs
2A4	00	2A4	
2A0	01	2A10	
2AN	02	2AN	
2A9	03	2A9	track-store
2B9	00	2B9	
2D9	00	2D9	
2CN	01	2CN	
2C9	02	2C9	interrupteurs relais
2BN	03	2BN	
1UP	04	A1UP	
1DW	05	A1DN	
2UP	06	A2UP	
OD1	03	D1	
OD2	13	D2	sorties analogiques
OD3	02	D3	AD 32
OD4	12	D4	
OD5	01	D5	
9EX	11	trunk X	sortie analogique EAI
OA1	37	A1	
OA2	36	A2	
OA3	35	A3	
OA4	34	A4	entrées analogiques
OA5	33	A5	AD 32
OA6	32	A6	
OA7	31	A7	
OA8	30	A8	
OA9	27	A9	
OA0	26	A10	

.../...

MESSAGES D'ERREUR

E00 retour chariot sauvage dans l'écriture d'une instruction

E01 instruction commencée par un caractère non réglementaire

E02 étiquette déjà définie

E03 oubli de parenthèse dans l'écriture d'une étiquette

E04 plus de quatre * DEB

E05 pas de : après une variable de déclenchement

E06 pas de * FIN et 50 caractères nuls

E07 pas de * DEB pour commencer un programme

E08 pas de retour chariot en fin d'instruction

E09 -pas d'opération ayant cette écriture
-mauvaise déclaration de la taille d'une opération

E10 -pas de variable de déclenchement ou de paramètre ayant cette écriture
-étiquette erronnée dans l'écriture d'une opération

E11 erreur dans l'écriture d'une pseudo-instruction

E12 pas de = dans l'écriture de l'opération d'initialisation de paramètre

E13

E14 instruction se composant de plus de 40 caractères

E15 manque une , dans l'écriture d'une instruction

E16 erreur dans l'écriture d'un caractère numérique

E17 -erreur dans l'écriture de TC/LR/GØ récrire la directive
-étiquette non définie

E18 pas d'opérande ayant ce code

E19

D défaut disque
pour relancer le programme faire marche.

.../...

CORRECTIONS A APPORTER SUIVANT LE TYPE D'INSTRUCTION ERRONNEE

1 - erreur dans l'écriture d'une étiquette de définition, en début d'instruction
réécrire l'instruction complète

2 - erreur dans l'écriture d'une instruction du type :

OPE

ou

(ETI) : OPE

réécrire l'opération seule (avec ses paramètres)

3 - erreur dans l'écriture d'une instruction du type :

(ETI) DECL : OPE

réécrire l'instruction avec une nouvelle étiquette ; cette étiquette aura la même définition que l'étiquette initiale ; par la suite on pourra utiliser indifféremment l'une ou l'autre.

VARIABLES DE DECLENCHEMENT

code symbolique	code octal	
H0	00	horloge temps réel
H1	01	
H2	02	
H3	03	
H4	04	
H5	05	
E0	50	appels externes
E1	51	
V0	60	voies de comptage AD 32
V1	61	
V4	64	voie de comptage EAI
C1	71	appel télétype appel disque

.../...

INSTRUCTIONS

code octal	code symbolique
01	=
03	RPI
05	RPC
11	MHD
12	CHØ
20	EAS
21	SAV
22	VID
23	ARR
25	BRC
27	TRK
30	XCP
33	BCI
35	XCM
36	MCI
37	CCL
40	TST
44	STY
46	ETY
47	STT
52	STØ
62	FET
63	GEL
64	INC
67	INF
70	INØ
74	SØA
75	MØP

INSTRUCTIONS

code symbolique	code octal	fonction	nbre de mémoires	opérandes
MCI	36	mise en IC d'un groupe d'intégrateurs	1	2A1 06 2A2 04 2A3 02 2A4 00
MOP	75	mise en OP.....	1	
MHD	11	mise en GEL.....	1	
GEL	63	gel de tous les intégrateurs	1	
CHØ	12	dégel de tous les intégrateurs	1	
TRK	27	mise en track	1	2AN 02 2B9 00 2A0 01 2A9 03
STØ	52	mise en store	1	
STT	47	mise en store 1er opérande en track (2è opérande)	1	2A0(2B9) 01 2A9(2AN) 03
TST	40	mise en track 1er opérande en store (2è opérande)	1	
INØ	70	interrupteur ouvert	1	2D9 00 2CN 01 2C9 02 2BN 03 1UP 04 1DW 05 2UP 06
INF	67	interrupteur fermé	1	
SØA	74	sortie analogique sur adresse	2	OD1 03 OD2 13 OD3 02 OD4 12 OD5 01 9EX 11
SAV	21	sortie analogique sur valeur	2	
STY	44	sortie sur TTY de tables	2	
ETY	46	entrée sur TTY de tensions	2	WW0 70 WW1 71 WW2 72 WW3 73
FET	62	fin d'échange table	1	

.../...

EAS	20	entrée analogique simple	2	OA1 37 OA2 36 OA3 35 OA4 34 OA5 33 OA6 32 OA7 31 OA8 30 OA9 27 OA0 26
INC	64	incrémentation d'un paramètre de 1 unité	1	PI0 10 . . . . . . PI7 17 PJ0 20 . . . . . . PJ7 27 PH6 06 PH7 07
VID	22	instruction vide	1	
ARR	23	instruction arrêt	1	
=	01	initialisation d'un paramètre	2	PI0...PI7 10...17 PJ0...PJ7 20...27 PH0...PH7 00...07 PV0...PV7 60...67
CCL	37	remise à zéro de la logique	1	
XCP	30	multiplie par 100 la vitesse de calcul	1	
XCM	35	divise par 100 la vitesse de calcul	1	
RPC	05	rupture conditionnelle sur progr. HYBRIDE — 1ère adresse si paramètre non nul	3	PI0...PI7 10...17 PJ0...PJ7 20...27
BRC	25	rupture conditionnelle sur progr. ASMAT — 2è adresse si paramètre nul	3	PH6,PH7 06,07
BCI	33	rupture inconditionnelle sur progr. ASMAT	2	
RPI	03	rupture inconditionnelle sur progr. HYBRIDE	2	

Annexe 2

LOGICIEL HYBRIDE T. 1000/AD 32

Liste des Instructions

M C I

XX	VD	36	M

Nom : MCI
Code : 36
Nbre de mots : 1

M O P

XX	VD	75	M

Nom : MOP
Code : 75
Nbre de mots : 1

M H D

XX	VD	11	M

Nom : MHD
Code : 11
Nbre de mots : 1

XX : code de l'opérande 2 Ai i = 1, 2, 3, 4 (06), (04), (02), (00)

VD : code de la variable de déclenchement

M : bit 18 : est égal à 1 si macro-instruction

Ecriture : MCI, 2Ai
MOP, 2Ai
MHD, 2Ai

But : Ces opérations ont pour but de mettre en conditions initiales, en opération, en hold un groupe d'intégrateurs

MCI fait sortir 1 niveau logique 1 sur la borne 2 Ai, IC et 1 niveau 0 sur 2 Ai. OP

MOP fait sortir 1 niveau logique 0 sur la borne 2 Ai, IC et 1 niveau 1 sur 2 Ai, OP

MHD fait sortir 1 niveau logique 0 sur la borne 2 Ai, IC et 1 niveau 0 sur 2 Ai, OP

G E L

	VD	63	M

Nom : GEL
Code : 63
Nbre de mots : 1

Ecriture : GEL

But : Cette opération a pour but de mettre en hold tous les intégrateurs, leur état est maintenu constant.

.../...

C H Ø

	VD	12	M

Nom : CHØ
Code : 12
Nbre de mots : 1

Ecriture : CHØ

But : Cette opération met en dégel tous les intégrateurs ; utilisée après l'instruction "gel" elle permet aux intégrateurs de retrouver le mode (IC ou OP) dans lequel ils étaient avant l'instruction GEL.

GEL fait sortir un niveau logique 0 sur toutes les bornes 2 Ai, IC et 2 Ai, OP
CHØ fait sortir les niveaux logiques qu'on avait avant l'instruction GEL sur les bornes 2 Ai, IC et 2 Ai, OP

T R K

XX	VD	27	M

Nom : TRK
Code : 27
Nbre de mots : 1

S T Ø

XX	VD	52	M

Nom : STØ
Code : 52
Nbre de mots : 1

XX : code de l'opérande 2 AN(02), 2B9 (00), 2A0 (01), 2A9 (03)
VD : code de la variable de déclenchement
M : bit de macro-instruction

Ecriture : TRK, opérande
STØ, opérande

But : Ces opérations ont pour but de mettre en track ou en store un groupe de "track and store".

TRK fait sortir un niveau logique 1 sur la borne 2 AN (ou 2B9, 2A0, 2A9)
STØ fait sortir un niveau logique 0 sur la borne 2 AN (ou 2B9, 2A0, 2A9)

S T T

XX	VD	47	M

Nom : STT
Code : 47
Nbre de mots : 1

.../...

T S T

XX	VD	40	M

Nom : TST
Code : 40
Nbre de mots : 1

XX : code de l'opérande 2A0 (01) ou 2A9 (03)

Ecriture : STT, 2A0 ou STT, 2A9
TST, 2A0 ou TST, 2A9

Buts : STT met en store l'opérande 2A0 (resp. 2A9) celui qui est associé, 2B9 (resp. 2AN) est mis en track
TST met en track l'opérande 2A0 (resp. 2A9) celui qui lui est associé, 2B9 (resp. 2AN) est mis en store

STT fait sortir un niveau logique 0 sur la borne 2A0 (resp. 2A9) et un niveau logique 1 sur 2B9 (resp. 2AN)

TST fait sortir un niveau logique 1 sur la borne 2A0 (resp. 2A9) et un niveau logique 0 sur 2B9 (resp. 2AN)

I N Ø

XX	VD	70	M

Nom : INØ
Code : 70
Nbre de mots : 1

I N F

XX	VD	67	M

Nom : INF
Code : 67
Nbre de mots : 1

XX : code de l'opérande 2D9 (00), 2CN (01), 2C9 (02), 2BN (03) 1UP (04), 1DW (05) 2UP (06)

Ecriture : INØ, opérande
INF, opérande

Buts : Ces opérations ont pour but d'ouvrir (resp. de fermer) un groupe d'interrupteurs ou de relais.

INØ, 2D9 (ou 2CN, 2C9, 2BN) fait sortir 1 niveau 0 sur la borne 2D9 (ou 2CN, 2C9, 2BN)

INF, 2D9 (ou 2CN, 2C9, 2BN) fait sortir 1 niveau 2 sur la borne 2D9 (ou 2CN, 2C9, 2BN)

Attention :

INØ, 1UP fait sortir 1 niveau 0 sur la borne 2B3/OP
INF, 1UP fait sortir 1 niveau 1 sur la borne 2B3/OP

INØ, 1DW fait sortir 1 niveau 0 sur la borne 2B3/IC
INF, 1DW fait sortir 1 niveau 1 sur la borne 2B3/IC

INØ, 2UP fait sortir 1 niveau 0 sur la borne 2B2/OP
INF, 2UP fait sortir 1 niveau 1 sur la borne 2B2/OP

.../...

SØA

XX	VD	74	M
adresse			

Nom : SØA
Code : 74
Nbre de mots : 2

XX : code de l'opérande OD1 (03), OD2 (13), OD3 (02) OD4 (12), OD5 (01), 9EX (11)

Ecriture : SØA, OD1, 2 000
SØA, OD1, (010)
SØA, OD1, (WWi)

But : Cette instruction permet la sortie sur la borne Di (i = 1, 2, 3, 4, 5) de l'AD 32 ou sur le trunk X de l'EAI (code 9EX) d'une tension contenue dans la mémoire dont on indique l'adresse.
SØA, OD1, 2 000 soit le contenu de la mémoire d'adresse octale 2 000
SØA, OD1, (010) soit le contenu de la mémoire d'étiquette (010)
SØA, OD1, (WWi) soit le contenu de la mémoire de la table (WWi) repérée par le pointeur affecté à cette table.

Rem : - pour affecter une adresse à l'étiquette (010) on utilise la pseudo-instruction
* EXT
* EXT (010) 5 000
- pour sortir le contenu de la première mémoire de la table (WWi) il faut initialiser sa lecture en utilisant l'instruction FET

SAV

XX	VD	21	M
valeur			

Nom : SAV
Code : 21
Nbre de mots : 2

XX : code de l'opérande OD1 (03), OD2 (13), OD3 (02), OD4 (12), OD5 (01), 9 EX (11)

Ecriture : SAV, ODi, 2 000 (valeur octale)
SAV, ODi, +5 00 (valeur décimale)

But : Cette instruction permet la sortie d'une tension sur la borne Di (i = 1, 2, 3, 4, 5) de l'AD 32 ou sur le trunk X de l'EAI (code 9 EX)
Les valeurs décimales sont comprises entre - 999 et + 999 ; l'unité est le centième de volt.
Les valeurs octales sont comprises entre 0000 et 7777.

.../...

S T Y

XX	VD	44	M
N (octal)			

Nom : STY
Code : 44
Nbre de mots : 2

XX : code de l'opérande WWi (7i) i = 0, 1, 2, 3

N : nombre de mots à sortir (en octal)

Ecriture : STY, (WWi), XXXX (nbre de mots à sortir (en octal)) i = 0, 1, 2, 3

But : Cette instruction effectue la sortie sur télétype du contenu de N mémoires de la table (WWi) à partir de l'adresse indiquée par le pointeur affecté à cette table ; ces valeurs sont des tensions codées en millivolts.

Attention : pour lire les N premières valeurs de la table (WWi) il faut initialiser au préalable son pointeur en utilisant l'instruction F E T

Message inscrit sur télétype

n° table - pointeur de table - valeur octale - valeur décimale en millivolts.

E T Y

	VD	46	M
adresse			

Nom : ETY
Code : 46
Nbre de mots : 2

Ecriture : ETY, adresse
ETY, (WWi) i = 0, 1, 2, 3

But : Entrer à partir de la télétype une valeur en millivolts dans la mémoire dont on définit l'adresse ou dans la table définie

ETY, adresse entrera la valeur dans la mémoire dont l'adresse est définie

ETY, (WWi) entrera la valeur dans la table (WWi) à l'adresse indiquée par le pointeur affecté à cette table.

pour écrire la valeur dans la 1ère mémoire de la table (WWi) il faut l'initialiser au préalable en utilisant l'instruction FET

Utilisation : l'appel d'une valeur se fait par l'impression de : sur télétype.

les valeurs sont comprises entre - 9995 et + 9995 ;

l'unité est le millivolt

elles doivent toujours être précédées du signe ; + ou - ;

le nombre de chiffres ne doit pas excéder 4 ;

le retour chariot entraîne la reprise du déroulement du programme.

.../...

F E T

XX	VD	62	M

Nom : FET
Code : 62
Nbre de mots : 1

XX : code de l'opérande WWi (7i) i = 0, 1, 2, 3

Ecriture : FET, (WWi) i = 0, 1, 2, 3

But : Cette instruction permet la fin d'échange table ;
elle termine l'écriture ou la lecture d'une table et l'initialise pour une nouvelle lecture ou écriture

Attention : si on veut lire plusieurs fois la même table il faut à chaque fois terminer complétement sa lecture ou faire FET

Rem : en début de SH les tables sont initialisées

Application : à utiliser avant STY et SØA sur table et éventuellement ETY.

E A S

XX	VD	20	M
adresse			

Nom : EAS
Code : 20
Nbre de mots : 2

XX : code de l'opérande OA1 (37), OA2 (36), OA3 (35), OA4 (34), OA5 (33), OA6 (32), OA7 (31), OA8 (30), OA9 (27), OA0 (26)

Ecriture : - EAS, OA1, 2 000
- EAS, OA1, (010)
  * EXT (010) 2 000
- EAS, OA1, (WWi)

But : Cette instruction permet d'entrer dans la mémoire dont on indique l'adresse la tension envoyée sur la borne Ai de l'AD 32.

Rem : EAS, OA1, (WWi) entre la tension envoyée sur la borne A1 dans la table WWi à l'adresse indiquée par le pointeur associé à cette table.

.../...

I N C

XX	VD	64	M

Nom : INC
Code : 64
Nbre de mots : 1

XX : code de l'opérande PIO..... PI7 (10 ..... 17)
PJO..... PJ7 (20 ..... 27)
PH6 (06)
PH7 (07)

Ecriture : INC, paramètre

But : Cette opération augmente de 1 unité la valeur du paramètre

V I D

	VD	22	M

Nom : VID
Code : 22
Nbre de mots : 1

Ecriture : VID

But : Cette instruction n'effectue aucune opération

A R R

	VD	23	M

Nom : ARR
Code : 23
Nbre de mots : 1

Ecriture : ARR

But : Cette instruction interrompt l'exécution du programme
pour relancer l'exécution du programme faire MARCHE

.../...

=

XX	VD	01	M
Valeur			

Nom : =
Code : 01
Nbre de mots : 2

XX : code de l'opérande PIO.....PI7 (10 ..... 17)
PJO.....PJ7 (20 ..... 27)
PHO.....PH7 (00 ..... 07)
PVO.....PV7 (60 ..... 67)

Ecriture : paramètre = - 500 (valeur décimale)
paramètre = 1 716 (valeur octale)

But : Cette opération attribue à un paramètre la valeur écrite après le signe = Cette valeur peut être décimale ou octale
Les valeurs décimales sont comprises entre - 524 287 et + 524 287 ; elles sont codées dans la 2ème mémoire sur 19 bits.
Les valeurs octales sont comprises entre 0000 et 7777 ; elles sont codées dans la 2ème mémoire sur 12 bits.

C C L

	VD	37	M

Nom : CCL
Code : 37
Nbre de mots : 1

Ecriture : CCL

But : L'instruction "common clear" remet à zéro toute la logique

X C P

	VD	30	M

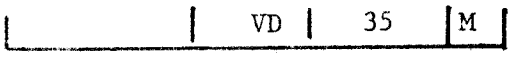
Nom : XCP
Code : 30
Nbre de mots : 1

Ecriture : XCP

But : Cette instruction multiplie par 100 la vitesse de calcul

.../...

X C M

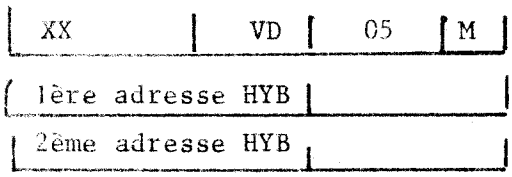


Nom : XCM
Code : 35
Nbre de mots : 1

Ecriture : XCM

But : Cette instruction divise par 100 la vitesse de calcul

R P C



Nom : RPC
Code : 05
Nbre de mots : 3

XX : code de l'opérande PIO.....PI7 (10 ..... 17)
PJ0.....PJ7 (20 ..... 27)
PH6 (06)PH7 (07)

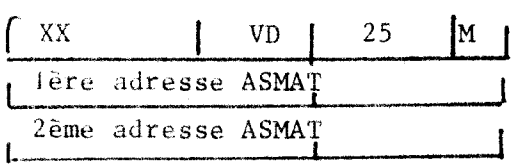
1 ère adresse : rupture à cette adresse si le paramètre est non nul

2 ème adresse : rupture à cette adresse si le paramètre est nul

Ecriture : RPC, PI5, 5 200, 5 300

But : Cette instruction permet de faire des aiguillages dans un programme hybride ; les adresses sont donc des adresses relatives au programme hybride ; l'assemblage implante les programmes hybrides (4 au maximum) les uns à la suite des autres à partir de l'adresse 5 000 ; la connaissance du nombre des mémoires utilisées par chaque instruction hybride permet de calculer les adresses.

B R C



Nom : BRC
Code : 25
Nbre de mots : 3

XX : code de l'opérande PIO.....PI7 (10 ..... 17)
PJ0.....PJ7 (20 ..... 27)
PH6 (06)PH7 (07)

1ère adresse ASMAT: rupture à cette adresse si le paramètre est non nul

2ème adresse ASMAT: rupture à cette adresse si le paramètre est nul

.../...

Ecriture : . BRC, PI5, 2 200, 2 300
. BRC, PI5, (010), (011)
* EXT (010) 2 200 (011) 2 300

But : Cette instruction permet de sortir d'un programme hybride et d'effectuer un programme ASMAT (qui sera par exemple un programme de calcul).

implantation des programmes ASMAT

on peut les implanter de 2 000 à 2 367 du bloc 0 ; on peut aussi utiliser la zone laissée libre par les programmes hybrides (de la dernière adresse du dernier programme hybride jusqu'à 1'adresse 6 000 du bloc 0)

chargement des programmes ASMAT

on ne peut utiliser le monoconsole.

- en conséquence on doit assembler le programme ASMAT avant d'appeler le superviseur et en sortir une bande binaire
- appeler ensuite le superviseur, assembler le programme hybride
- charger la bande binaire du programme ASMAT avec CHARP

écriture des programmes ASMAT

- faire les sauvegardes nécessaires
  en particulier sauvegarder l'index et l'état si on les modifie au cours du programme
- retour au programme hybride

★ si paramètre ≠ 0 il s'effectue par l'instruction IRV 3 002 ; (= RETOU -1) pour le 1er programme ASMAT

★ si paramètre = 0 il s'effectue par l'instruction IRV 3 003 ; (= RETOU) pour le 2ème programme ASMAT

B C I

VD	33	M
adresse ASMAT		

Nom : BCI
Code : 33
Nbre de mots : 2

adresse ASMAT : rupture à cette adresse

Ecriture : . BCI, 2 200
. BCI, (010)
* EXT (010) 2 200

But : Cette instruction permet de sortir d'un programme hybride et d'effectuer un programme ASMAT
pour l'implantation, le chargement et l'écriture du programme, voir l'instruction BRC

.../...

Attention :

Le retour au programme hybride se fait par l'instruction IRV 3 003 ;
3 003 = RETOU

R P I

	VD	03	M

adresse hybride

Nom : RPI
Code : 03
Nbre de mots : 2

adresse hybride : rupture à cette adresse

Ecriture : RPI, 5 000
RPI, (001)

But : Cette instruction permet de faire un aiguillage inconditionnel dans un programme hybride.

S Y P

P_i	VD	13	M

adresse hybride

Nom : SYP
Code : 13
Nbre de mots : 2

Ecriture : SYP, P1, (001)

But : Cette instruction permet de repositionner le pointeur P_i de l'un des 4 programes hybrides.

L C Ø

Param	VD	43	M

Nom : LCØ
Code : 43
Nbre de mots : 2
Opérandes : 2DN,2OO,2CO,2BO

Ecriture : LCØ , 2DN

But : Cette instruction permet de lire l'état d'un comparateur 2ON, 2DO,2CO,2BO, le paramètre PJ7 est chargé à 1 ou 0 selon l'état du comparateur.

REPONSE A LA QUESTION ETAT ?

VARIABLES DE DECLENCHEMENT UTILISEES	ETAT
Vi	0 1 7 7
Hi	0 2 7 7
Ei	0 3 3 7
C_1	0 3 6 7
Vi + Hi	0 0 7 7
Vi + Ei	0 1 3 7
Vi + C_1	0 1 6 7
Hi + Ei	0 2 3 7
Hi + C_1	0 2 6 7
Ei + C_1	0 3 2 7
Vi + Hi + Ei	0 0 3 7
Vi + Hi + C_1	0 0 6 7
Hi + Ei + C_1	0 2 2 7
Vi + Ei + C_1	0 1 2 7
Vi + Hi + Ei + C_1	0 0 2 7

-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

Annexe 3

CARACTERISTIQUES DU CALCULATEUR T 1000

Caractéristiques du calculateur T1000

- Mini calculateur industriel
- Mots de 19 bits
- Organisation par bloc de 4Kmots
- Temps moyen d'instruction 10µs à 12 µs
- 8 niveaux d'interruptions cablées

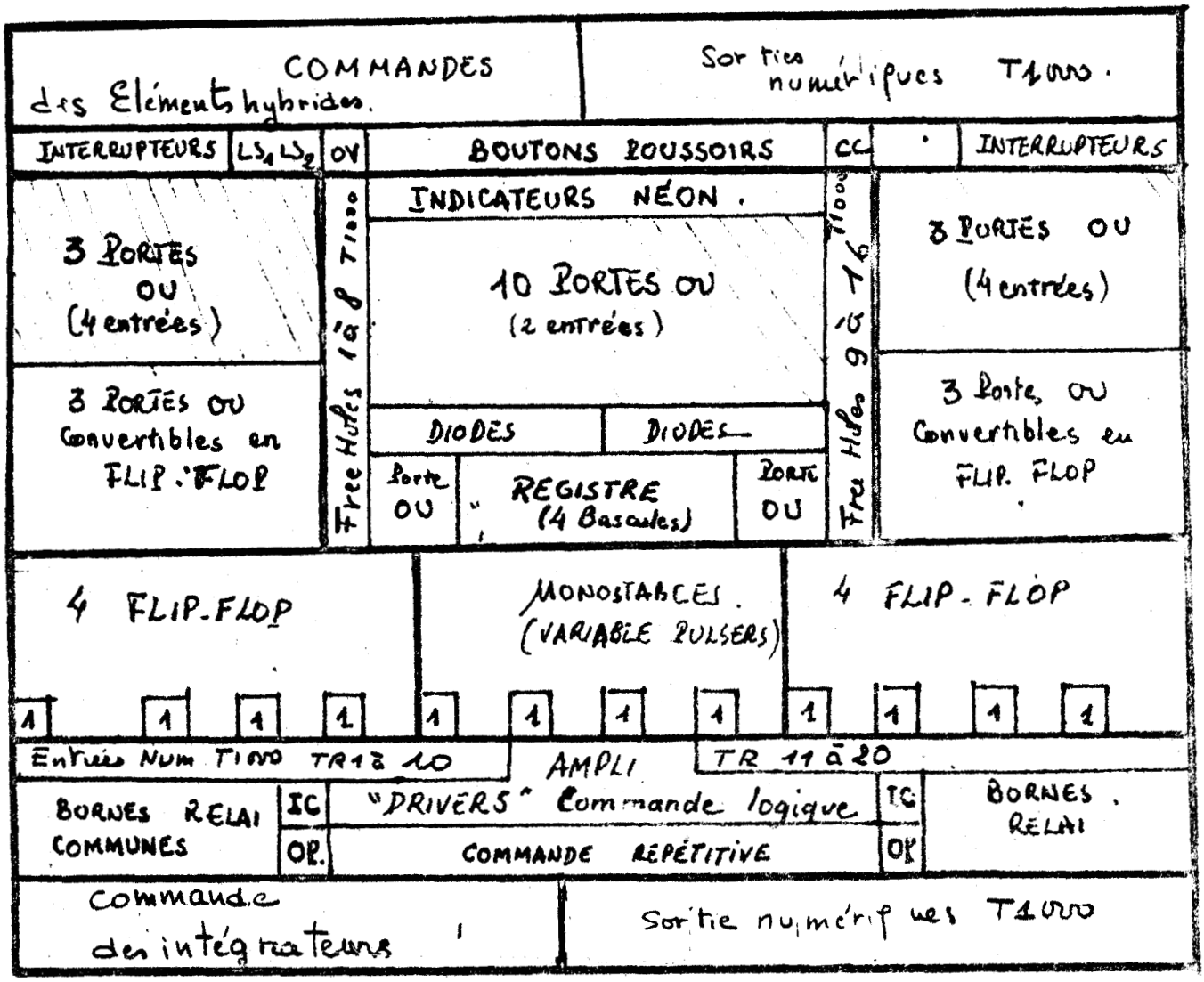
  - Voie de comptage
  - Horloge temps réel
  - appels externes
  - Bac d'entrée-sortie
  - accés direct mémoire

Configuration informatique

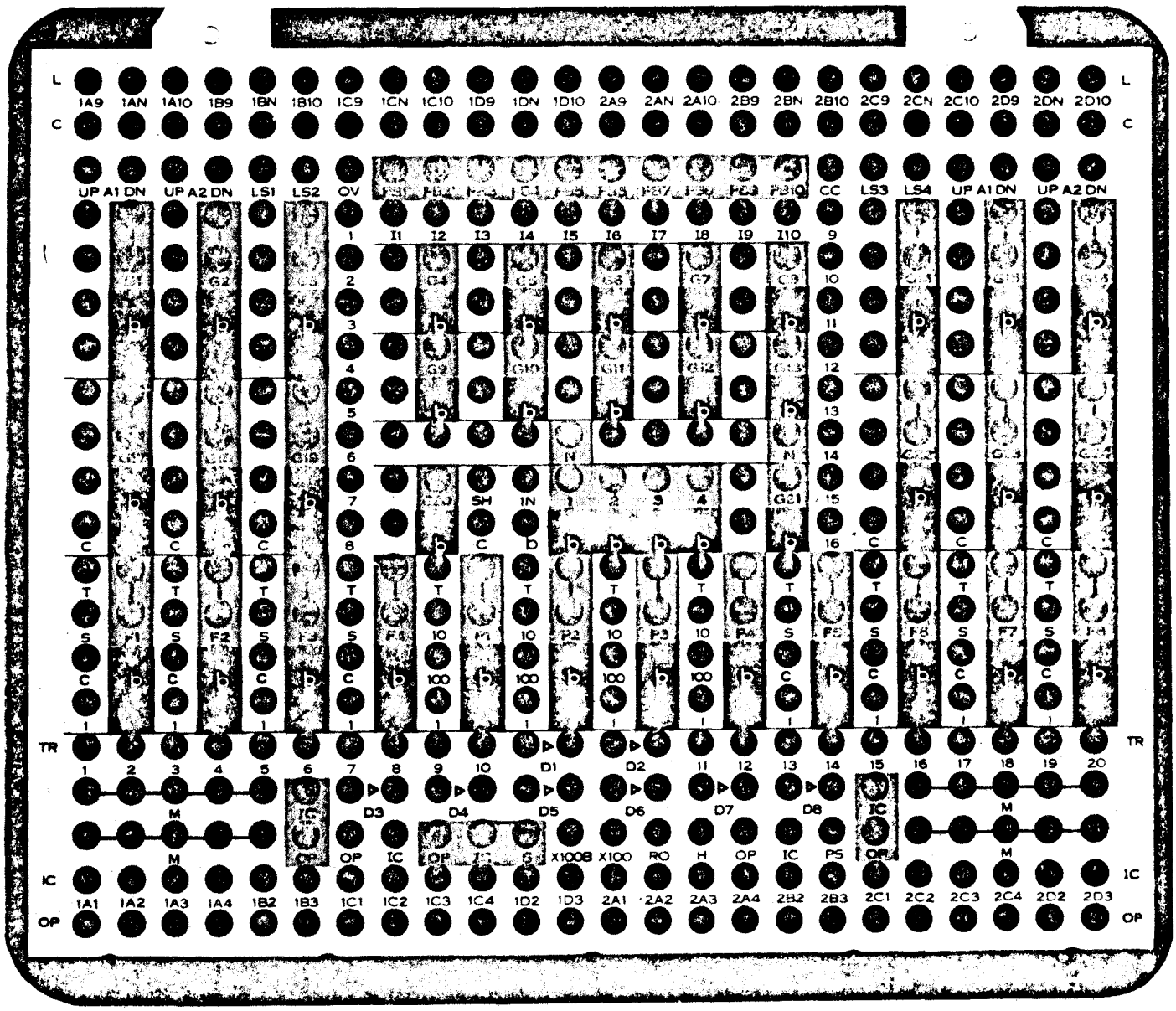
- Unité centrale 8 K mots
- Périphériques informatiques
  - télétype
  - lecteur de ruban
  - disque à tête fixe

- Périphérique de couplage
  - Chaine de mesure
  - interface numérique /numérique
    numérique /analogique
    avec une AD 32 et un EAI

Eléments du cablage AD 32



Eléments du panneau logique



Codes de cablage du panneau logique

connection logique/digitale

S O M M A I R E

page

page

S O M M A I R E