MASTER EN SCIENCES INFORMATIQUES

Ébauche d'un système élémentaire d'exploitation d'un mini-ordinateur

Henriet, A.

Award date:
1973

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX - NAMUR
INSTITUT D'INFORMATIQUE

ANNEE ACADEMIQUE 1972-1973

Ebauche d'un système élémentaire d'exploitation d'un mini-ordinateur

A. HENRIET

Mémoire présenté en vue de l'obtention
du grade de Licencié et Maître en Informatique.

C'est avec plaisir que nous remercions Monsieur BRUNIN pour son dynamisme inventif et sa direction enthousiaste.

L'équipe d'étude du projet EPRON nous a fourni les détails hardware indispensables à la bonne compréhension de la machine. Nous lui exprimons notre reconnaissance pour sa collaboration.

Nous remercions aussi tous les professeurs de l'INSTITUT D'INFORMATIQUE, et plus particulièrement ceux dont les cours nous ont fait découvrir le domaine des "Operating Systems" et des problèmes qu'ils posent.

Nous tenons à signaler le travail attentif de Madame WIBIN. Qu'elle trouve ici l'expression de notre gratitude.

NAMUR, 1er août 1973.

INTRODUCTION

Il était une fois .... un petit EPRON.
Tel est le nom provisoirement retenu pour ce mini-ordinateur, actuellement à l'étude aux facultés de Namur (EPRON = Experimental PROCESSOR Namur).
Mini-ordinateur, EPRON le sera sur base de critères de prix et d'encombrement (1).
Pour ne pas être trop coûteux, le hardware ne sera pas trop spécialisé. Les mécanismes performants de la machine lui seront fournis par son firmware à la fois riche et dynamique, ce dynamisme permettant d'adapter le firmware au type d'application à traiter, sans pour autant modifier le hardware.
Ce mini-ordinateur micro-programmé sera susceptible de travailler avec toute la gamme des périphériques classiques, cette connexion se faisant progressivement..
Une première phase ne fournira que des périphériques élémentaires (lecteur de cartes, imprimante rudimentaire).
Au stade suivant seront adjoints des périphériques plus performants tels que mini-cassette ou disque (mini).
Doté de cet environnement, EPRON pourra fonctionner comme terminal évolué d'un gros ordinateur.
Nous n'avons pas la prétention de présenter ici une description détaillée du hardware et du firmware de l'EPRON.
Le lecteur la trouvera dans le travail de J. DEMARTEAU (1).

Le software de base implémenté sur l'EPRON devra fournir à l'utilisateur un outil simple et performant, et tirer parti, au maximum, des possibilités offertes par l'infrastructure hardware-Firmware.
Dans une première phase, la décision a été prise de travailler uniquement dans un contexte de monoprogrammation.
Cette option, qui supprime les problèmes d'allocation de ressources et de conflits d'accès, facilitera la tâche des réalisateurs, tout en leur permettant de tester l'adéquation du software au firmware.

---

(1) Cf. la définition d'EPRON dans "Architecture de systèmes et micro-programmation dynamique" J. DEMARTEAU p.0.3.

Cette correspondance entre les routines software et le micro-programme, nous avons cherché à la mettre en évidence dans le traitement de deux mécanismes : la gestion des interruptions et les entrées-sorties (IØ). Ceux-ci font l'objet respectivement des chapitres 1 et 2.

Afin de situer ces modules dans l'ensemble du software de l'EPRON nous fournissons, au chapitre 3, un bref aperçu des différents constituants du software élémentaire.
Il est clair que ceux-ci ne constituent qu'une petite partie du vaste domaine du software de base, dans lequel on rassemble généralement les compilateurs, système d'exploitation, gestion de fichiers, programmes utilitaires.

Nous attirons l'attention du lecteur sur le fait que cet exposé ne constitue pas la description d'un système existant, appuyé par des mesures de performances.
Il détermine seulement une base de travail, en quelque sorte un cahier de charge que devront respecter les réalisateurs du système.

TABLE DES MATIERES

CHAPITRE I - GESTION DES INTERRUPTIONS

CHAPITRE II - ENTREES - SORTIES

CHAPITRE III - AUTRES MODULES

CONFIGURATION MEMOIRE.

- - - - - - -

CHAPITRE I - GESTION DES INTERRUPTIONS.

1. INTRODUCTION.

1.1. Définitions.

Notre machine est munie d'un système d'interruptions, cela signifie qu'elle possède des mécanismes spécialisés permettant d'interrompre une séquence à certains moments et moyennant certaines conditions.

Nous envisagerons trois façons d'interrompre une séquence : les Vols de cycle, les débranchements et les interruptions de programme.

Le vol de cycle consiste seulement en un arrêt momentané, sans répercussion sur le déroulement de la séquence.

Le débranchement et l'interruption de programme entraînent un arrêt, mais aussi un déroutement provisoire ou définitf (la décision étant prise par la séquence de déroutement).
Le débranchement a lieu à la demande de la séquence interrompue, il est donc synchronisé avec celle-ci, et peut être pris en charge dès qu'il est demandé.
L'interruption de programme est provoquée par un facteur échappant au contrôle de la séquence en cours. Sa survenance est donc aléatoire, et désynchronisée par rapport à la séquence en cours. C'est pourquoi certaines précautions doivent être prises avant d'autoriser leur prise en charge.

1.2. Classification.

Chacun de ces types possède son domaine d'application.

1.2.1. Vols de cycle.

Ils sont essentiellement destinés à la gestion de "process" indépendants du programme déroulé par le CPU, mais devant accéder à des ressources internes.
Nous y trouverons les transferts de bytes entre mémoire et canal (bytes de données ou de commande).

1.2.2. Débranchement.

Dans le cadre limité de la monoprogrammation, ce sont essentiellement des appels du programme utilisateur à destination d'un des modules du système.
Parmi ces modules, nous trouverons le superviseur d'IØ, l'Executive, mais aussi, dans une phase ultérieure, le système de gestion de fichiers.

1.2.3. Interruption de programme.

Il y aura lieu de distinguer trois classes en fonction de leur origine : Externe, interne software, interne machine.

1.2.3.1. Externe.

Ces demandes proviennent essentiellement des périphériques ou du timer (qui, fonctionnellement, sera considéré comme un périphérique particulier).
Dans cette classe, nous distinguerons trois familles.

END : signal envoyé par le canal pour annoncer la fin de transfert d'un ou plusieurs blocs (en relation directe avec le END DEVICE).

ANOMALIE : signal annonçant la détection d'une erreur en cours de bloc.
Il est envoyé dans le cas d'erreurs intéressantes à signaler ou à traiter avant la fin du bloc, ou dans le cas où les erreurs entravent le déroulement de l'opération et empêchent l'apparition du signal END.
A titre d'exemple, nous citerons la détection d'une erreur de parité dans tampon du périphérique, et pour autant que la répétition puisse se faire avant la validation de l'écriture sur le périphérique. (seulement pour les périphériques de sortie, dont le buffer a la dimension du bloc physique).
Nous pourrions aussi considérer, en télétraitement, les erreurs de transmission de bloc (par exemple, erreur de parité dans le contrôleur de transmission).
Selon le cas, cette erreur sera simplement signalée (remplacer caractère invalide par FF), ou l'opération sera arrêtée (décision prise par le hardware, conjointement à l'envoi du signal I_{ANOM})

APPEL : signal utilisé par les périphériques dont l'appel peut initialiser une opération d'IØ (alors que, en général, l'IØ démarre à l'initiative du programme par exécution de STIØ).
Nous retiendrons les appels console, le timer (lorsque l'intervalle de temps alloué est terminé), et, en télétraitement, la réception par le contrôleur d'un STX entrant.

N.B. : Nous avons introduit, à titre d'exemples, certains cas relatifs au télétraitement. Nous n'y reviendrons pas par la suite, le télétraitement fera l'objet d'un travail séparé.

1.2.3.2. Internes software.

Nous trouverons là des interruptions provoquées par des erreurs de programme, et qui consistent en :

1. CODE INVALIDE.

   Le code décrit dans l'instruction n'existe pas dans le langage machine, c'est-à-dire qu'il ne réfèrence aucune des mini-instructions situées dans la zone des mini-instructions de la mémoire et qui interprètent le code fonction.
   Il y aurait moyen de raffiner cette notion en testant la compatibilité entre le code fourni et les opérandes qui lui sont associés.
   Ce test serait facilité par la découpe standard des instructions (cf. travail parallèle traitant du langage machine).

   Le code est aussi invalide lorsque l'on veut exécuter, en mode esclave, une instruction privilégiée, c'est-à-dire valide uniquement en mode maître.

2. ADRESSE INVALIDE.

   L'adresse indiquée dans l'instruction n'est pas dans la zone adresse tolérée : soit que sa valeur dépasse la dimension existant sur la configuration, soit qu'il y ait violation de la protection de mémoire.

3. RESULTAT FAUTIF D'INSTRUCTIONS.

   Nous retiendrons :

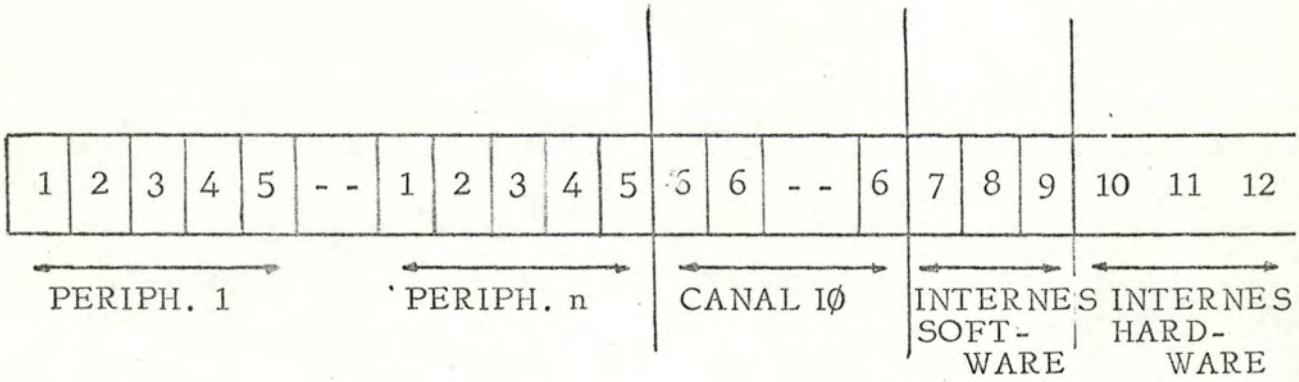
   - addition ou soustraction avec résultat incorrect.
   - tentative de division par 0 , ou par un nombre trop petit pour la précision de la machine.
   - détection d'overflow après CVB.
   - shift arithmétique et perte du signe.

1.2.3.3. Interne machine.

Ces interruptions seront déclenchées lors de la détection de malfonctionnement du hardware du CPU.
Nous trouverons notamment les erreurs de parité mémoire :

- à l'entrée en mémoire (provenant de registres ou des bus externes).
- à la sortie de la mémoire.

En groupant toutes les demandes indépendantes, du programme, (vols de cycle-interruptions), en fonction de leur origine, nous trouverons le tableau décrit à la fig. 1 .1.



1 R_i

2 R_0

3 END ~~SERVICE~~ DEVICE

4 I_{APPEL}

5 I_{ANOM}

6 END CANAL (provoqué par firmware)

7 CODE

8 ADRESSE

9 DATA (avec possibilité de plusieurs signaux :

- overflow CVB
- division par 0 , etc...

10 PARITE ENTREE MC

11 PARITE SORTIE MC

12 POWER FAILURE.

Fig. I. 1

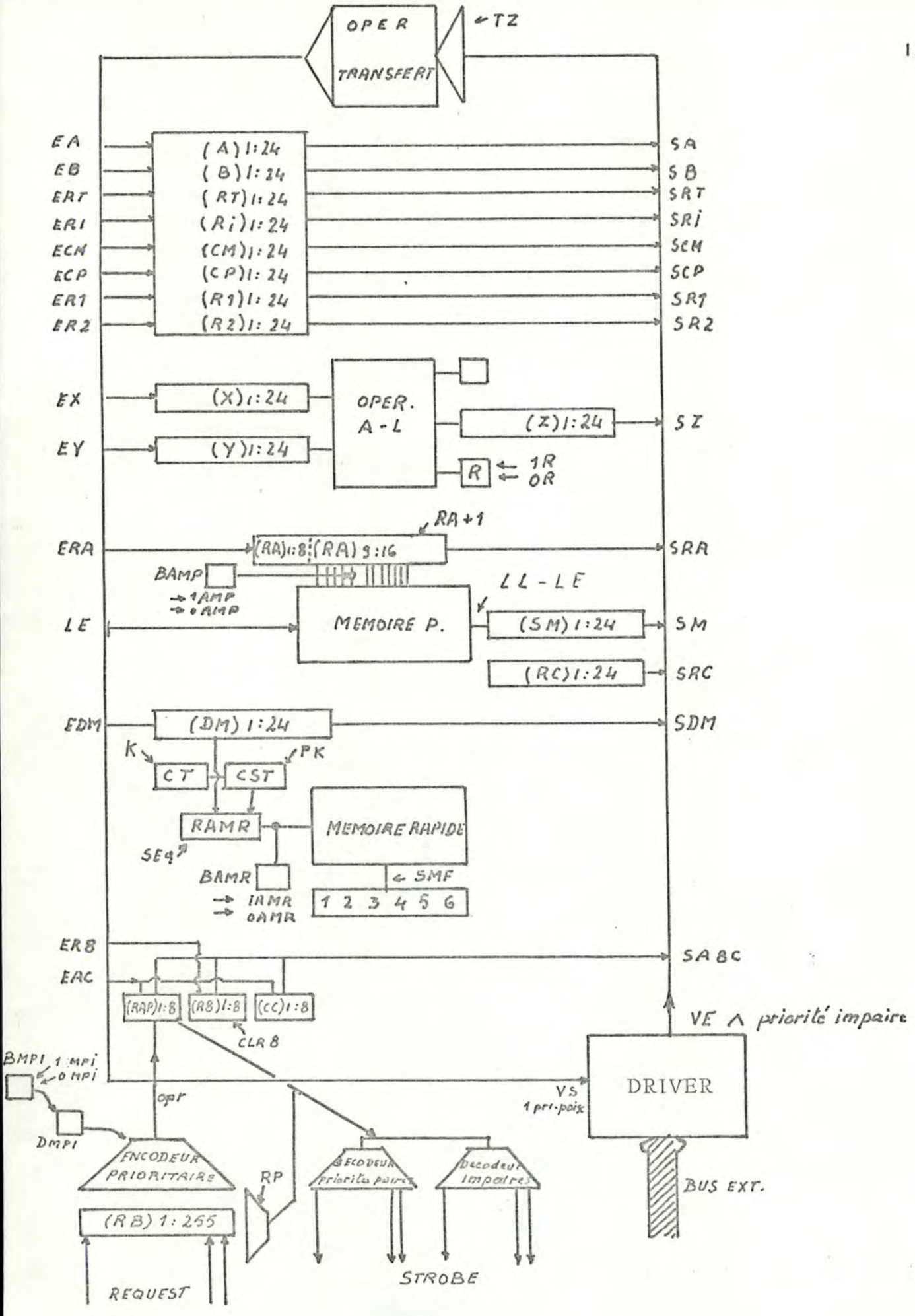

OPER
TRANSFERT
TZ
EA
EB
ERT
ERI
ECM
ECP
ER1
ER2
(A) 1:24
(B) 1:24
(RT) 1:24
(Ri) 1:24
(CM) 1:24
(CP) 1:24
(R1) 1:24
(R2) 1:24
SA
SB
SRT
SRi
SCM
SCP
SR1
SR2
EX
EY
(X) 1:24
(Y) 1:24
OPER. A-L
(Z) 1:24
SZ
R
1R
OR
RA+1
ERA
(RA) 1:8
(RA) 9:16
SRA
BAMP
1AMP
0AMP
LL - LE
LE
MEMOIRE P.
(SM) 1:24
SM
(RC) 1:24
SRC
EDM
(DM) 1:24
SDM
K
PK
CT
CST
RAMR
MEMOIRE RAPIDE
SEq
BAMR
IRMR
0AMR
SMF
1 2 3 4 5 6
ER8
EAC
SABC
(RAP) 1:8
(RB) 1:8
(CC) 1:8
CLR 8
VE ∧ priorité impaire
BMPI
1 MPI
0 MPI
DMPI
opr
VS
1 pri. pair
DRIVER
ENCODEUR PRIORITAIRE
RP
(RB) 1:255
REQUEST
Décodeur priorités paires
Décodeur impaires
STROBE
BUS EXT.

Fig. I. 2 ([D] 1.35)

2. OUTIL HARDWARE - FIRMWARE.

Pour un développement plus détaillé du mécanisme, nous renvoyons au travail de J. DEMARTEAU.
Du schéma de base de l'EPRON (cf. fig. I.2), il est possible d'extraire le schéma suivant du traitement hardware des interruptions (cf. fig. I.3).

2.1. Demandes.

De façon asynchrone par rapport au déroulement du programme, une demande d'interruption est signalée par l'apparition d'un signal Request (1) et mémorisée dans le banc des Requests (2).

Un encodeur prioritaire (3) sélectionne le request à traiter : le résultat de l'encodage donne dans APR le n° du Request à traiter (4).

L'encodeur étant câblé, les priorités accordées aux Requests sont fixes pour une configuration de machine.

2.2. Prise en charge.

Celle-ci s'effectuera au temps To.
Rappelons que To est la phase de prise en charge d'une mini-instruction (ou ligne μ).
Trois cas sont possibles :

TOB : bouclage sur la mini-instruction,

TOI : prise en charge d'un Request.

TON : poursuite normale du programme.

Le choix de la phase et l'exécution de celle-ci se font par hardware.
Néanmoins, on peut en trouver une description fonctionnelle en langage μ (cf. [D] I.49).

Lorsqu'il y a détection du fait que le contenu de APR n'est pas nul, la séquence TOI se déroule, c'est-à-dire :

- chargement dans RAP (5) de 4 * n° Request + X (X étant l'adresse de début de la zone des Mots réservés).
- chargement de cette valeur dans RA (6).
- lecture en mémoire de la mini-instruction logée à cette adresse, et stockage de celle-ci dans le registre DM (7).

Ainsi, la nouvelle mini-instruction prise en charge ne sera pas celle indiquée par CM, mais celle qui répond au Request.
Notons que, dans ce mécanisme, les registres du programme n'ont pas été perturbés, puisque nous n'avons fait appel qu'à DM, devenu inopérant (phase To)

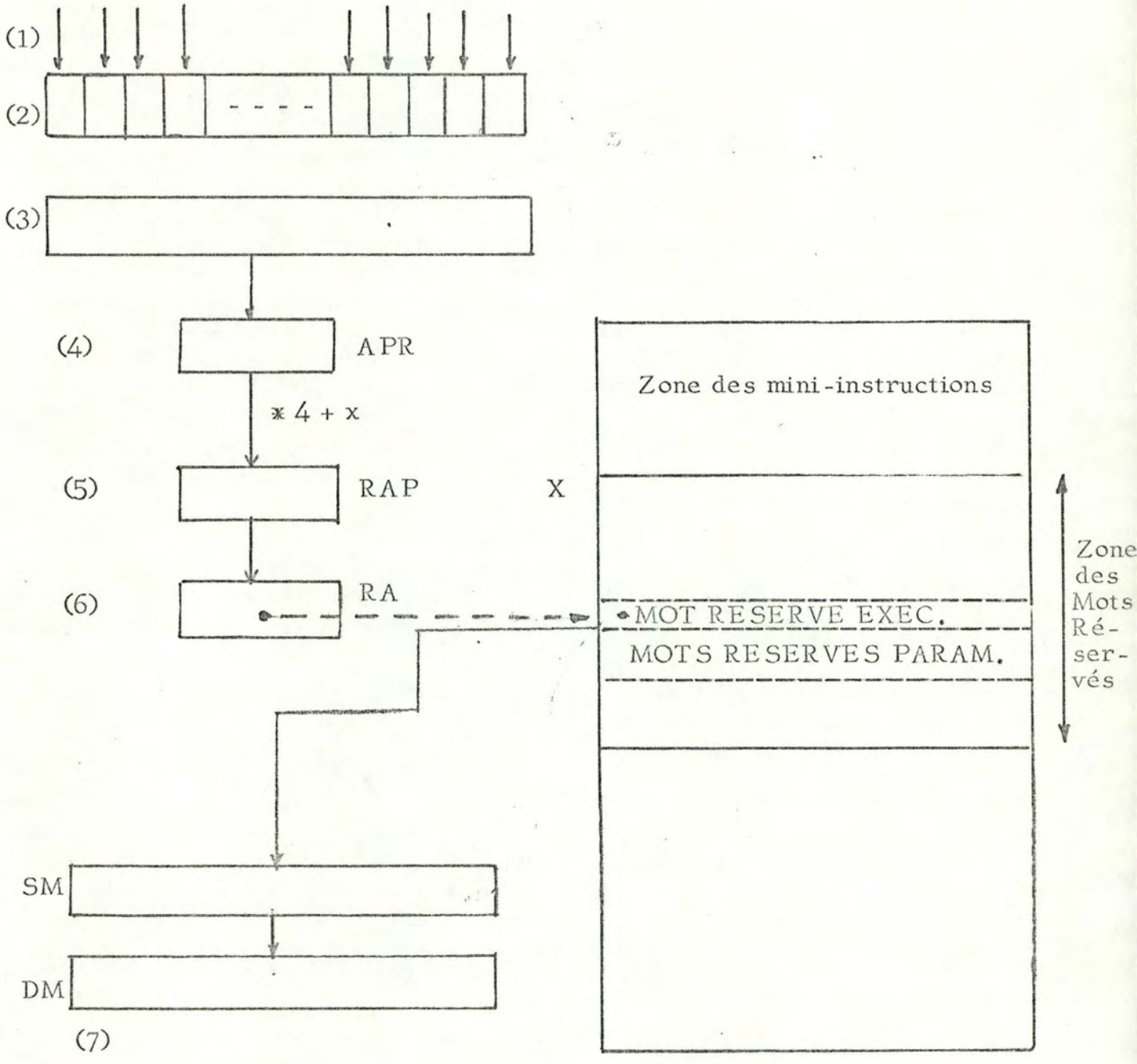


Fig. I. 3

------> Adresse

——————> Transfert

La zone des "Mots réservés" est une zone mémoire de dimension 4n, n étant le nombre de Requests implantés sur la machine.
A chaque Request correspondent ainsi 4 mots réservés :

1. 1 MOT RESERVE EXECUTIF
où sera stockée la mini-instruction de réponse au Request.

2. 3 MOTS RESERVES PARAMETRES
contenant les paramètres nécessaires à cette mini-instruction.

Il va sans dire que la zone associée à un Request aura dû être chargée avant la prise en charge de celui-ci, soit en cours de programme, soit au chargement du système.
Il convient aussi de remarquer qu'un certain nombre de requests sont susceptibles de masquage à l'entrée de l'encodeur.

Une façon de réaliser ce masquage est l'utilisation d'une bascule BMPI , positionnée et dépositionnée respectivement par les ordres MPI et DMPI, ce qui nous donnerait un masque "tout ou rien", et permettrait de considérer deux "étages" de Requests.

1. ceux qui sont testés à chaque To.

2. ceux qui ne sont testés que lorsque le système le permet, par positionnement du masque.

N.B. : Pour la clarté de l'exposé, nous avons mentionné la zone de "Mots Réservés" associée aux Requests. En fait, suite à cette partie accessible par hardware se loge une autre partie, accessible à partir du software : RAP étant chargé non plus du contenu de APR, mais d'une valeur fournie par le firmware (cf. instructions d'IØ).

3. MECANISMES.

3.1. VOLS DE CYCLE.

3.1.1. Demande.

Elle sera annoncée par un signal Request, et donc mémorisée dans le banc des Requests.

3.1.2. Prise en charge.

Celle-ci se fera en To, et suivant l'encodage prioritaire, qui, rappelons-le, est câblé.
Le temps de réponse à un request sera fonction du nombre de Requests plus prioritaires positionnés avant sa prise en compte.

3.1.3. Traitement.

Celui-ci se limitera à l'exécution pure et simple de la mini-instruction logée dans le "Mot Réservé" correspondant.
Ainsi que nous l'avons signalé, il n'aura donc aucune répercussion sur les éléments de travail du programme ni sur son fonctionnement ultérieur.

Dans le cadre actuel de l'EPRON, les seules applications du vol de cycle seront les séquences de transfert de bytes (DTS), et les fins de périphériques (END DEVICE).

N.B. : Alors que, sur des machines plus puissantes, les vols de cycle sont câblés, EPRON sera muni de vols de cycle micro-programmés.

3.2. INTERRUPTIONS.

3.2.1. Demandes.

Suivant leur appartenance à l'une ou l'autre des classes définies précédemment, les demandes d'interruption sont formulées de façon différente.

- Les demandes EXTERNES et les demandes INTERNES. MACHINE sont formulées par des signaux émis par le hardware respectivement des périphériques et du CPU, (signaux Requests), ce qui entraînera leur mémorisation dans le BANC DES REQUESTS.
- Les demandes INTERNES SOFTWARE sont provoquées soit par hardware (opérateur arithmétique), soit par firmware. Dans ce dernier cas, il est possible d'envisager deux fonctionnements différents :
  d'une part, sous-jacent au firmware de prise en charge des instructions, existe le hardware qui détectera les erreurs, ce qui nous ramène aux classes précédentes, avec utilisation du BANC DES REQUESTS.
  D'autre part, le firmware gère lui-même la détection et la mémorisation de ces demandes, dans ce cas, il serait plus aisé de mémoriser ces demandes en MEMOIRE CENTRALE, dans une zone réservée à cet effet, que nous appellerions "zone des interruptions".
  Cette option compliquerait quelque peu l'étape de prise en charge des interruptions, puisqu'elle nécessiterait la consultation de deux zones distinctes: le banc des Requests, et la "zone des interruptions".
  D'autre part, elle permettrait une plus grande souplesse dans l'établissement des priorités des interruptions software puisque celles-ci ne seraient pas câblées.

  Il appartiendra aux réalisateurs du hardware et du firmware de définir le choix final.

  Toutefois, pour donner de l'ensemble une vue synthétique, nous prendrons comme base de travail la première option proposée : toutes les demandes d'interruption sont enregistrées dans le banc des Requests.

3.2.2. Prise en charge.

Du fait que le traitement d'une interruption s'accompagne toujours d'un DEROUTEMENT, deux précisions doivent être apportées : le moment et les priorités de prise en charge.

L'interruption doit être prise en charge à un moment tel que ce mécanisme ne détruise (ou ne risque de détruire) aucun des éléments de la séquence interrompue.

C'est ainsi que les interruptions indépendantes du programme ne seront autorisées que lorsque l'instruction en cours sera terminée (en fin de phase RNI). De cette manière, les zones contenant des résultats intermédiaires peuvent être perdues, et le sauvetage s'effectuera uniquement pour les éléments utilisés d'une instruction à l'autre (Registres généraux, Registres ordinaux CP et CM).

En ce qui concerne les erreurs de programme, il n'est même pas nécessaire de garantir les valeurs intermédiaires, puisqu'elles se sont avérées inexactes. C'est pourquoi la prise en charge de l'interruption peut débuter avant la fin de l'instruction en cours.

La SELECTION de la demande à traiter s'effectuera suivant l'une des trois méthodes suivantes :
first in - first out , l'introduction de priorités, ou l'utilisation de niveaux d'interruptibilité.
Dans le premier cas, seul compte l'ordre d'arrivée. Cette méthode serait appropriée pour des interruptions présentant toutes le même degré d'urgence.

Le travail avec priorités permet de favoriser certaines demandes, considérées comme plus urgentes.
Si le mécanisme de prise en charge est uniquement hardware, la sélection de la demande à traiter est immédiate : celle-ci est fournie à la sortie d'un encodeur prioritaire câblé.

Une autre possibilité serait de stocker dans une file d'attente les demandes ainsi que certaines informations relatives à leur identification.
Le prélèvement de la demande la plus prioritaire se ferait alors sous le contrôle du software; ce qui ralentirait le traitement des interruptions, mais permettrait aussi une structure moins figée:
les priorités pourraient être modifiées selon les applications, ou même dans le temps (par ex. en cas d'engorgement).

L'utilisation de niveaux d'interruptibilité permet d'accentuer l'avantage offert aux interruptions les plus urgentes; toute demande d'un niveau donné peut interrompre toutes les séquences répondant à des demandes de niveau strictement inférieur.
A chaque niveau est associé un MASQUE d'INTERRUPTION.
Un des problèmes essentiels réside dans la définition du nombre de niveaux, et la répartition des interruptions entre ces différents niveaux.

Si nous voulions utiliser cette méthode, nous pourrions, par exemple, introduire une correspondance entre les niveaux d'interruptibilité et les différentes classes que nous avons définies au début de ce chapitre, une priorité étant accordée à chacune des demandes à l'intérieur des niveaux.

Une telle méthode nécessiterait quelques modifications du hardware actuellement envisagé, puisque, dans le schéma proposé, une seule bascule (BMPI) est prévue pour tenir compte des masquages et démasquages.
C'est pourquoi nous proposons le système suivant : il existe deux types d'interruptions, celles qui sont masquables et celles qui ne le sont pas.

1. INTERRUPTIONS MASQUABLES.

Correspondent à un niveau d'interruptibilité 1 (Alors que le programme utilisateur est affecté d'une niveau 0).

Elles peuvent donc interrompre le programme utilisateur, mais ne peuvent s'interrompre mutuellement : dès que l'une d'elles est prise en compte, le signal de masquage est émis, et la bascule (BMPI) positionnée jusqu'à la fin de la routine.

Ce domaine de niveau 1 coïncide avec celui des interruptions qui ne peuvent intervenir qu'à la fin de l'instruction en cours.

Dans cette catégorie, nous trouverons :
toutes les interruptions externes (END CANAL, I_{APPEL}, I_{ANOM}), et certaines interruptions software :
celles qui ne sont pas liées à la détection d'erreurs du programme.
(par exemple, dans un contexte de multitasking, le bloquage d'une tâche par une autre tâche).

2. INTERRUPTIONS NON MASQUABLES.

La prise en compte de l'une d'elles provoque le positionnement du masque d'interruptions, donc le masquage des demandes de niveau 1 et 0 (programme utilisateur).
Mais elles-mêmes ne sont jamais masquées.

D'autre part, ces interruptions se confondent avec celles que l'on peut traiter sans attendre la fin de l'instruction en cours.

Ce type d'interruptions regroupera les erreurs internes, parce que celles-ci doivent être signalées et traitées dès qu'elles sont détectées par le hardware ou le firmware.
Nous y trouverons donc les erreurs machine et les erreurs de software interne.

Un tel système appelle cependant certaines réserves :

1. Du fait que certaines interruptions ne sont jamais masquées, nous risquons de rencontrer des appels récursifs relatifs à ces priorités, et de boucler indéfiniment.
   Un solution serait de comptabiliser les appels, et dans le cas de répétition de la même demande, signaler l'erreur à la console.
   La seule solution serait la réinitilisation du système.

2. Certaines erreurs du software ne nécessitent peut-être pas une action aussi immédiate, dans ce cas, elles seront traitées de la même façon que celles du premier type (avec masquage).

La concordance entre ces options et le mécanisme hardware décrit plus haut se réalise donc de la façon suivante :

1. les demandes d'interruptions seront toutes signalées par le banc des Requests (sauf avis contraire des responsables du hardware).

2. en To seront susceptibles d'être prises en charge les interruptions non masquables : HARDWARE INTERNE, SOFTWARE INTERNE (du moins adresse et code invalides).

3. à la fin d'une instruction, et pour peu que l'on ne soit pas en cours d'interruption (BMPI = 0), seront prises en charge les interruptions masquables:
   EXTERNES, et certaines interruptions SOFTWARE INTERNE.

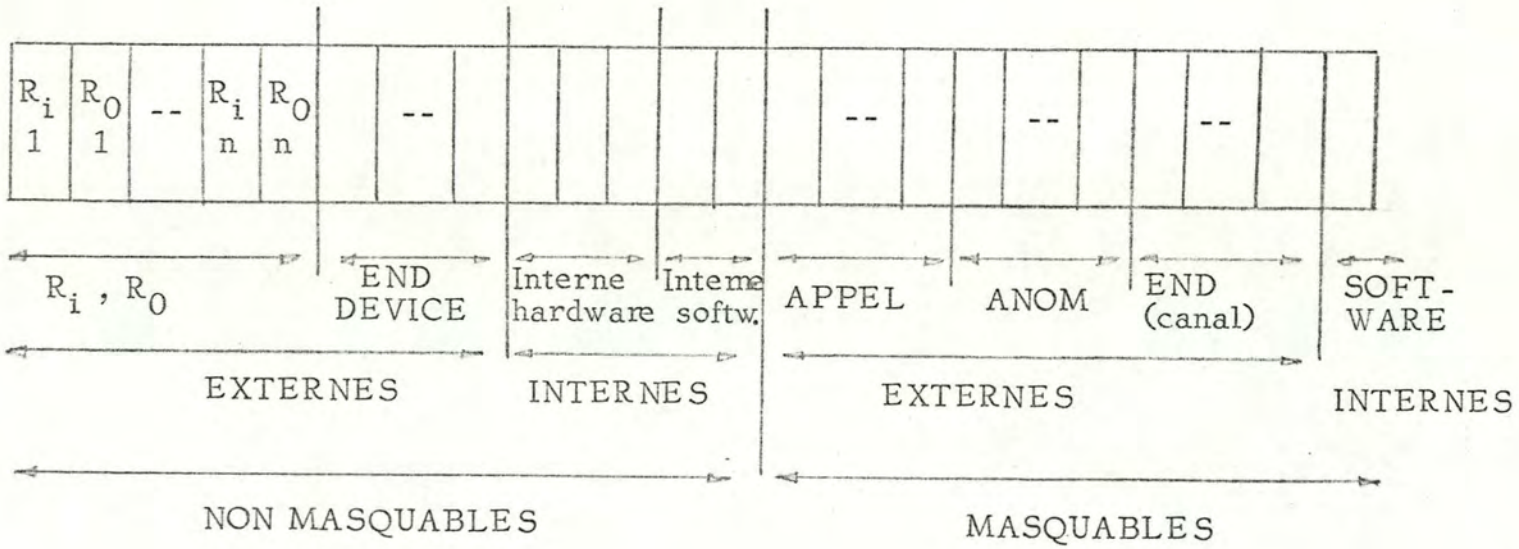


Fig. I.4

CLASSIFICATION REQUESTS PAR PRIORITES

N.B. : La numérotation des périphériques (1, n) est ici fonction des priorités de ceux-ci, c'est-à-dire leur vitesse et leur importance dans le système.

3.2.3. TRAITEMENT.

Le traitement d'une interruption comprend trois phases :

1. La préparation (Aiguillage).
2. Le traitement proprement dit.
3. Le retour.

3.2.3.1. PREPARATION - AIGUILLAGE.

C'est elle qui assure le passage de la séquence interrompue à la routine de traitement de l'interruption, tout en garantissant que ce déroutement s'opère sans dégâts pour le programme interrompu.

Selon que l'optique choisie sera "tronc commun" ou pas, cette phase sera réalisée par une routine séparée, et commune à toutes les demandes, ou sera un segment de la routine invoquée.

Trois types d'opérations se succéderont dans cette phase : la modification des paramètres de fonctionnement du CPU, le sauvetage des registres de travail, et la transmission des paramètres.

MODIFICATION DES PARAMETRES DE FONCTIONNEMENT.

Les 3 composants à traiter sont : le masque d'interruptions, le mode de travail, et la protection mémoire.

Ainsi que nous l'avons dit précédemment, la prise en charge d'une interruption entraîne le positionnement du masque d'interruption (c'est-à-dire de la bascule BMPI).

Le mode de travail MAITRE/ESCLAVE est lié à la notion d'instructions privilégiées : en mode maître, tous les codes opérations sont valides; en mode esclave, certaines instructions sont inexécutables. Parmi celles-ci, nous noterons les instructions d'IØ.

Une tentative d'exécution de ces instructions en mode esclave provoque une demande d'interruption de programme "code invalide", parce que le firmware ne détecte pas un code exécutable; ainsi que le montre le schéma suivant.

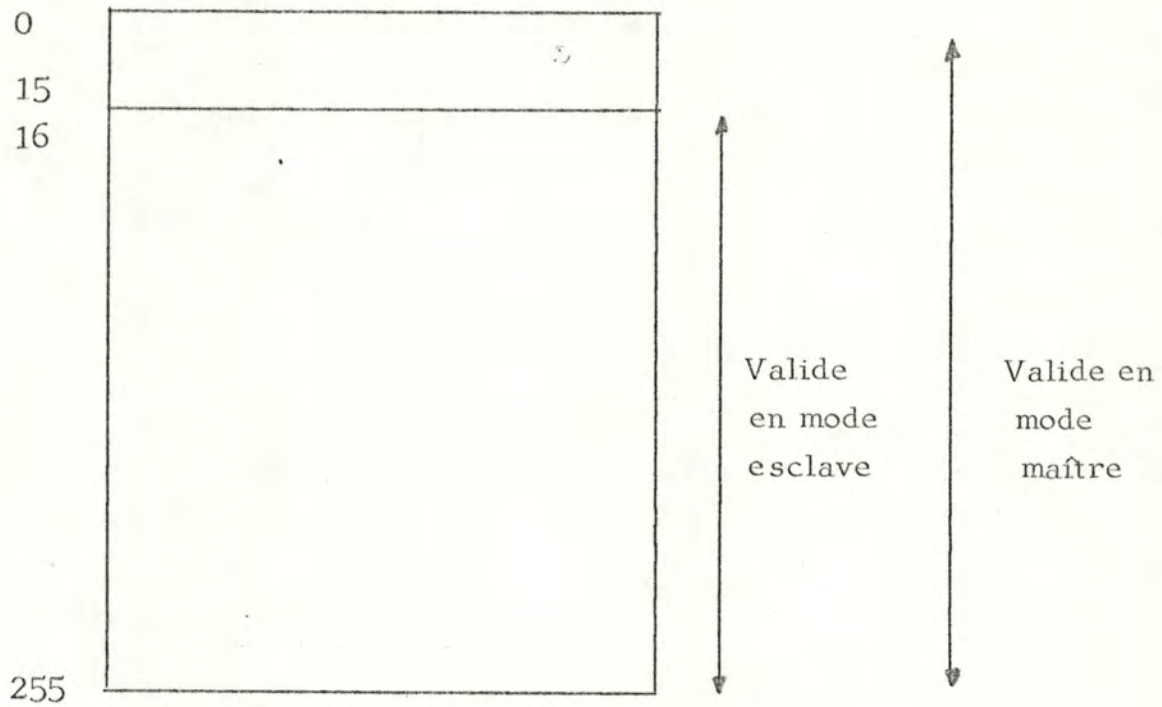


Fig. I.5

ZONE MEMOIRE des MINI-INSTRUCTIONS interprétant les codes opération.

L'indication du mode dans lequel on travaille(donc du Range de Validité du code) est fournie par la bascule BAMP.

Si BAMP = 0 toute la zone est accessible, tous les codes sont validés.

Si BAMP = 1 les cellules 0 à 15 sont interdites, une tentative d'accès à l'une d'elles génère CODE INVALIDE.

Pour plus de renseignements, nous renvoyons le lecteur au travail de J. DEMARTEAU (cf. [D] I,37).

La protection mémoire est le complément indispensable au travail en mode Maître /esclave pour protéger l'un de l'autre les programmes utilisateurs, et pour garantir la zone système. (Dans le cadre de monoprogrammation, il est clair que seule cette action est réalisée).

La protection d'une zone mémoire peut être totale (c'est-à-dire que tout accès en lecture ou en écriture est interdit pour tout programme ne "montrant pas patte blanche".)

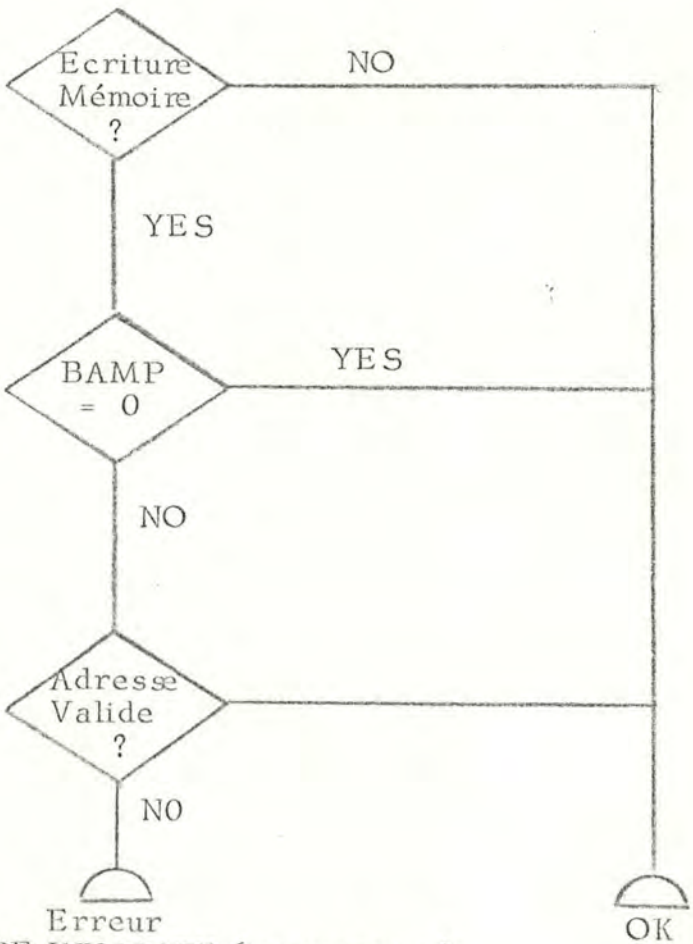
Un autre type de protection est celui qui contrôle l'accès en écriture seulement (c'est-à-dire dans le cas où un programme risque de détruire une zone qui ne lui est pas attribuée.)

Ce contrôle peut être réalisé par utilisation de clé de protection (Cf. IBM 360), ou par registres associés à chaque programme et contenant les bornes de la zone autorisée pour celui-ci.

POUR EPRON, il a été décidé de limiter la protection aux opérations d'écriture, et de le réaliser par utilisation de registres de bornes.

N.B. : puisque nous sommes en monoprogrammation, la protection est inefficienteen mode MAITRE.

La protection mémoire se traduira donc par l'organigramme ci-dessous :



ADRESSE INVALIDE (interruption)

N.B. : Le test "Adresse Valide" consiste en la comparaison de l'adresse de l'opérande (situé dans RI) avec les bornes situées dans les registres.

REMARQUE IMPORTANTE.

Même si, apparamment, elles jouent des rôles complémentaires, il faut éviter de confondre les notions de niveaux d'interruptibilité, protection mémoire, et mode Maître/Esclave.

Dans un système qui travaille en monoprogrammation, ces trois mécanismes sont quelque peu redondants, mais dans un système plus complexe, leurs rôles respectifs apparaissent plus clairement.

La protection Mémoire garantit chacune des applications de destruction possible par d'autres.

Le mode Maître/Esclave assure la ségrégation programme/système.

Les niveaux d'interruptibilité sont liés au degré d'urgence des applications.

SAUVETAGE DES REGISTRES DE TRAVAIL.

Il est à noter que cette fonction, de même que la transmission des paramètres, n'est pas une caractéristique des problèmes d'interruptions, mais se retrouve dans tous les appels à une sous-routine. Nous distinguerons trois aspects à la question : que sauver, qui les sauve, où les sauver ?

Les quantités sauvées sont les différents registres contenant des informations utilisées d'une instruction à l'autre, c'est-à-dire les registres généraux, les accumulateurs, les compteurs ordinaux (CP, CM) et le code condition.

Le sauvetage sera effectué soit par la routine commune de sauvetage et d'aiguillage, soit par chacune des routines d'interruption.

Une zone de sauvetage doit être prévue pour chaque niveau d'interruptibilité, afin d'éviter tout écrasement.
Nous aurons donc : une zone associée à toutes les demandes de niveau 1 (Masquable) et une zone associée à chacune des demandes non masquables (puisque, en fait, chacune d'elles constitue un niveau).

TRANSMISSION DES PARAMETRES.

Pour atteindre la routine de "traitement proprement dit", il sera nécessaire d'en connaître soit l'adresse, soit le type, auquel cas le branchement se fera via la table des SVC (cf. aussi le chapitre suivant). (1) .
Ce paramètre sera constant pour une configuration du système.

D'autre part, il est nécessaire de fournir à la routine les paramètres qu'elle devra utiliser : par exemple, les routines traitant d'entrée/sortie devront accéder à l'identification du périphérique, au mot d'état de celui-ci, ainsi que l'autres valeurs contenues dans le CCB (cf. chapitre suivant). Ce sera donc l'adresse du CCB qui sera fournie comme paramètre.

Ces paramètres seront transmis par l'intermédiaire des "mots Réservés Paramètres" associés par le hardware au Request pris en charge.

N.B. : Pour que ces paramètres restent accessibles pendant l'exécution de la routine, il faudra assurer le sauvetage de l'adresse de ces mots réservés. En effet, au moment de l'appel, l'adresse est contenue dans RA, mais sera perdue dès la prise en charge de la première instruction de la routine.

Ce sauvetage pourrait se faire, par exemple, dans la table des routines du système, ce qui exclurait la possibilité d'appels récursifs à ces routines.
(cf. cas particulier du superviseur 1Ø).

(1) : Cette dénomination désigne la table que nous appellerons plus loin "Table des routines du système".

3.2.3.2. TRAITEMENT PROPREMENT DIT.

Ainsi que nous l'avons signalé précédemment, la routine de traitement peut inclure la phase de préparation, ou être indépendante de celle-ci.
Sa structure peut donc être schématiquement représentée par l'une des figures suivantes :

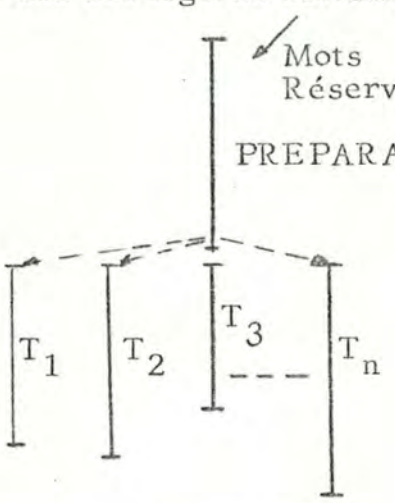


Fig. I.6a

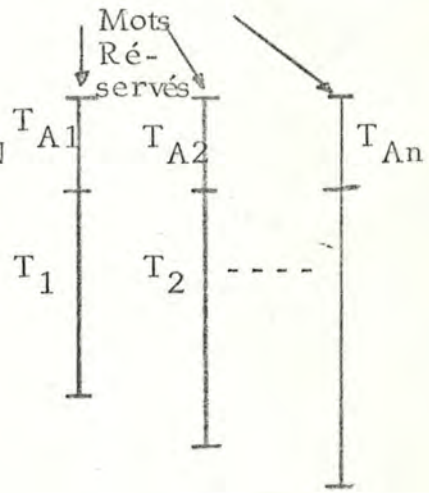


Fig. I.6b

Dans le premier cas (fig. 6a), le firmware de réponse au Request oriente toujours vers la routine d'aiguillage qui effectuera les trois fonctions décrites à la phase précédente, puis branchera à la routine adéquate.

Dans le second cas (fig. 6b), le firmware de réponse au Request oriente directement vers "sa" routine, auquel cas toutes les routines débuteront par une séquence quasi-identique, celle qui effectue les trois fonctions de préparation. Elle occasionne donc une perte de place mémoire, mais assure un gain de temps, puisque le branchement à la routine est automatique.

En tirant parti au maximum des possibilités du firmware, il serait peut être possible d'adopter la solution suivante, compactage des deux méthodes;
la mini-instruction de réponse au Request assure elle-même l'exécution de la phase 1 (Prise en compte et phase 1 sont donc confondues), puis oriente vers la routine de traitement proprement dit. Cette solution présenterait l'avantage de traiter toute la phase 1 comme une action indécomposable (puisque réalisée en une seule mini-instruction), ce qui la protégerait des interventions des interruptions non masquées.

La routine de traitement proprement dit sera soit commune à une famille d'interruptions (Ex. : END CANAL), soit propre à une interruption (Ex. : RECOVERY PERIPHERIQUE sera fonction du périphérique).

1. Les interruptions externes,

   donneront lieu à l'exécution d'une séquence du superviseur 1Ø. Cette partie est traitée en détail au chapitre suivant.

2. Erreurs de software,

   se solderont par un message console et l'abandon du programme, sauf avis contraire de l'utilisateur (cf.chapitre 3).

3. Erreurs machine,

   après tentative infructueuse de répétition correcte, on fera appel à une routine pas à pas destinée à localiser le sens de l'erreur (sortie ou entrée mémoire).
   Cette routine serait à envisager dans un ensemble de programmes de maintenance.

3.2.3.3. RETOUR-RESTAURATION.

La décision de retour ou non à la séquence interrompue dépendra de la routine qui traite l'interruption.
(par ex., celle-ci peut provoquer un branchement au module d'éjection du programme).
Dans le cas de retour, nous suivrons une démarche symétrique à celle de l'aiguillage:

1. Restauration des quantités sauvées,
2. Modification des paramètres de fonctionnement du CPU,
3. Branchement à l'adresse de retour.

La structure dépendra de celle adoptée pour l'aiguillage : englober cette routine dans la routine de traitement, ou utiliser une routine commune.
(la solution optimale étant la réalisation de trois opérations dans une même mini-instruction).

Le mécanisme global de traitement d'interruption est résumé à la fig. I.8.

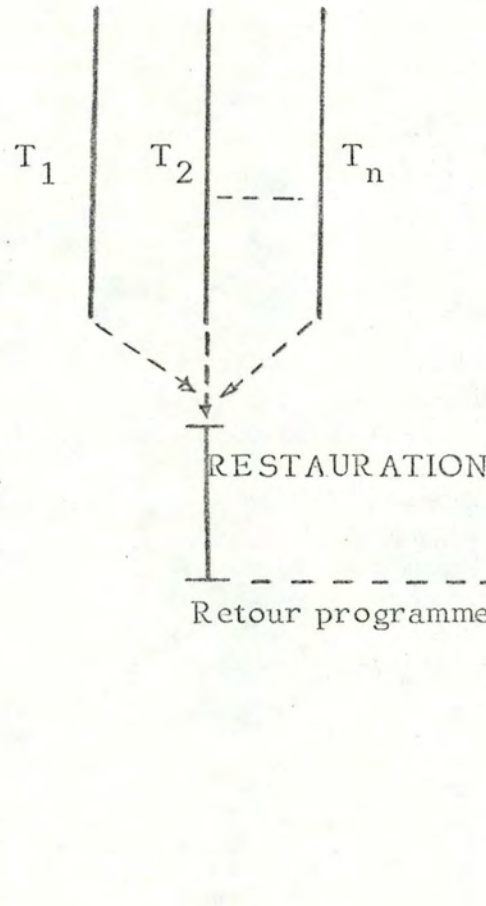


Fig. I.7a

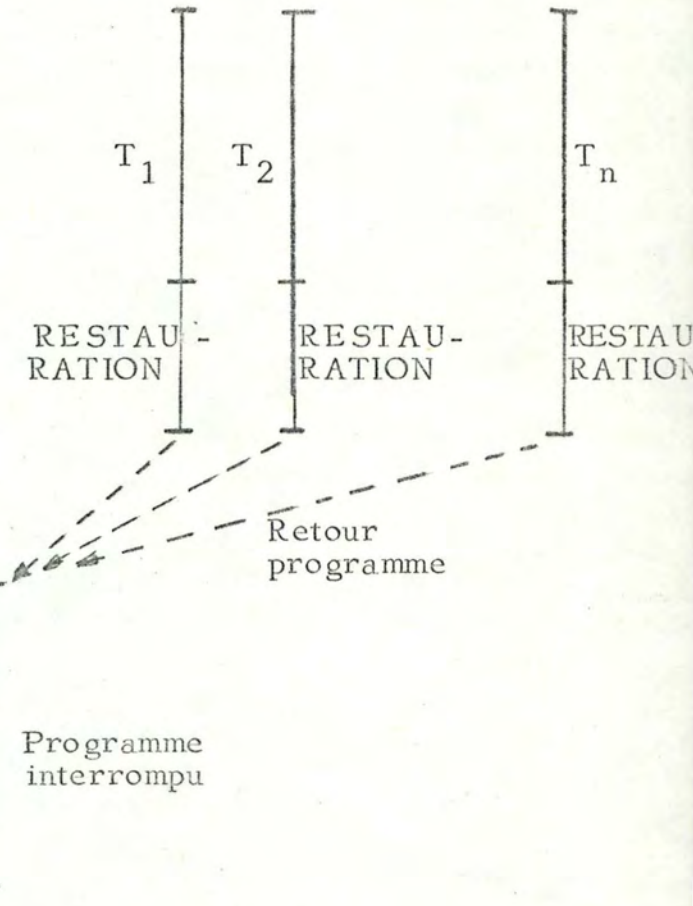


Fig. I.7b

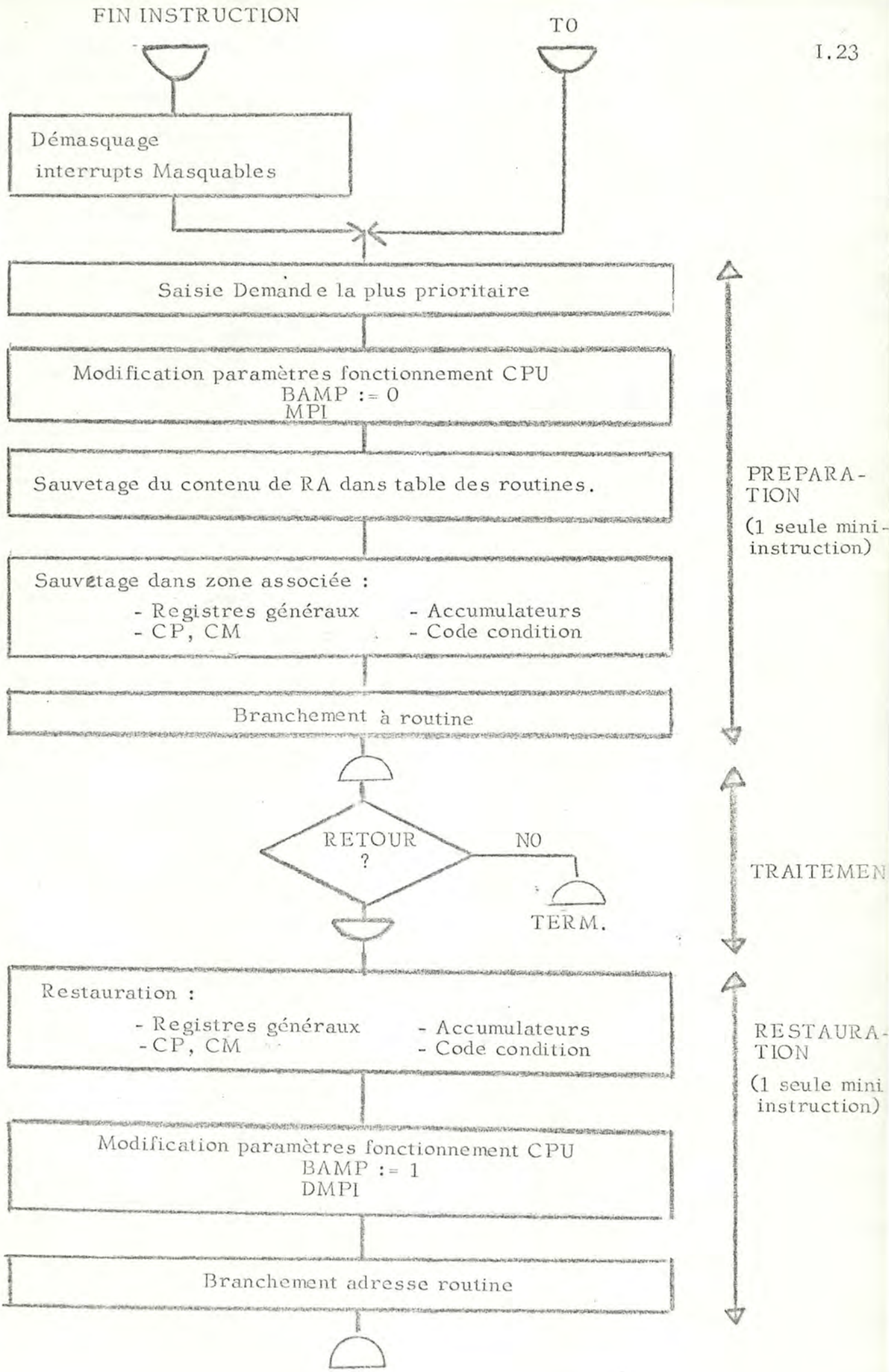

FIN INSTRUCTION
TO
Démasquage
interrupts Masquables
Saisie Demand e la plus prioritaire
Modification paramètres fonctionnement CPU
BAMP := 0
MPI
Sauvetage du contenu de RA dans table des routines.
Sauvetage dans zone associée :
- Registres généraux
- Accumulateurs
- CP, CM
- Code condition
Branchement à routine
RETOUR ?
NO
TERM.
Restauration :
- Registres généraux
- Accumulateurs
- CP, CM
- Code condition
Modification paramètres fonctionnement CPU
BAMP := 1
DMPI
Branchement adresse routine
PREPARA-
TION
(1 seule mini-
instruction)
TRAITEMEN
RESTAURA-
TION
(1 seule mini
instruction)

Fig. 1.8

3.3. DEBRANCHEMENTS.

3.3.1. Demandes.

Contrairement aux deux mécanismes décrits précédemment, le débranchement n'est pas sollicité par un Request, mais par le programme à interrompre.

Ce genre de demande n'entraîne pas de mémorisation, puisque l'utilisateur connaît et veut l'interruption.
La demande sera formulée de la façon suivant :

SVC	TYPE
A (PARAMETRES)	

3.3.2. Prise en charge.

Le code SVC de l'instruction est interpété par la mini-instruction correspondante (de même que toutes les instructions).

Cette interprétation consiste en un test de validité du type : si celui-ci n'appartient pas au "Range" autorisé, il y a demande d'interruption "code invalide".
Lorsque le type est reconnu, le passage est autorisé à la phase de traitement.

N.B. : Ce contrôle du type permet une protection du système à l'égard des accès fautifs de la part de l'utilisateur.

3.3.3. Traitement.

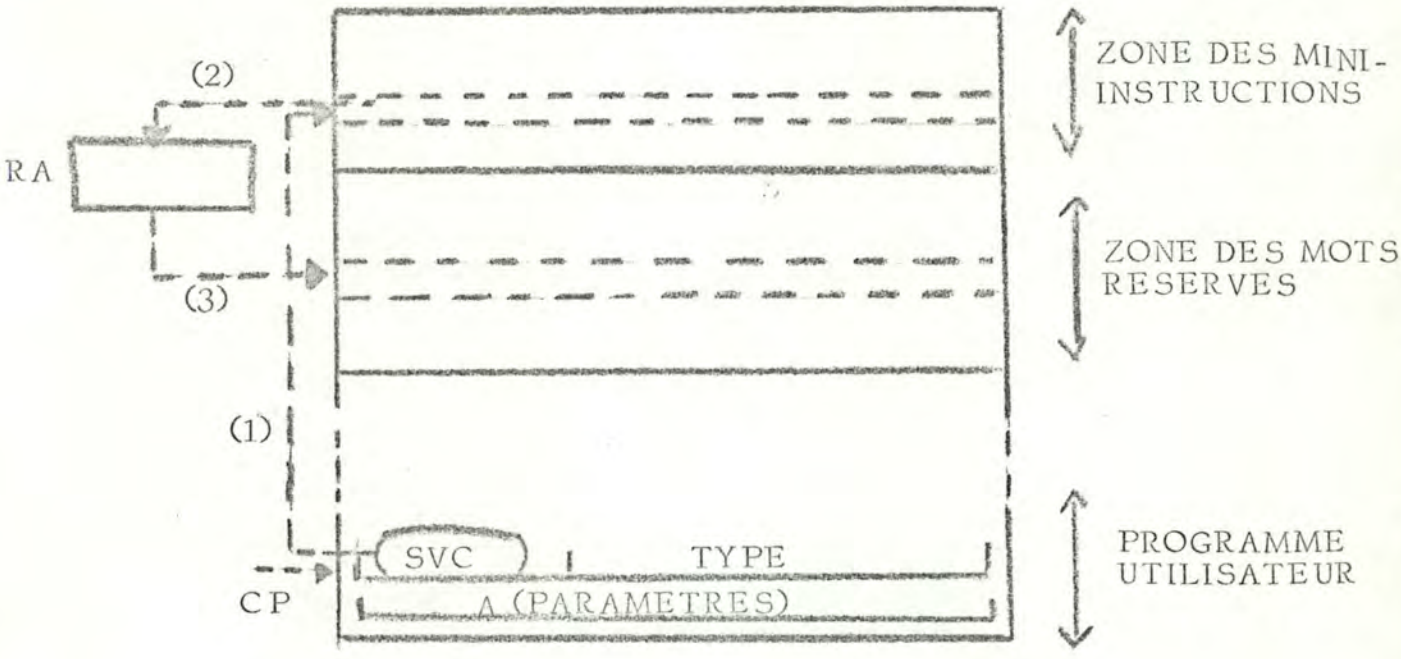
Il est analogue à celui des interruptions, cela signifie qu'il regroupe les mêmes phases de préparation, traitement proprement dit, et Retour.

Il est à noter toutefois que la transmission des paramètres se fera non plus à partir des mots réservés, mais à partir de CP puisqu'ils figurent comme 2e opérande de l'instruction SVC.

De même que pour les interruptions, cette adresse doit rester accessible en cours d'exécution de la routine, c'est pourquoi elle sera sauvée dans la table des routines du système.

D'autre part, le branchement à la routine se fera obligatoirement via la table des routines du système, le type du SVC fournissant l'index.

Ce mécanisme est décrite à la fig. I.9.


(2)
RA
(3)
(1)
CP
SVC
TYPE
A (PARAMETRES)
ZONE DES MINI-INSTRUCTIONS
ZONE DES MOTS RESERVES
PROGRAMME UTILISATEUR

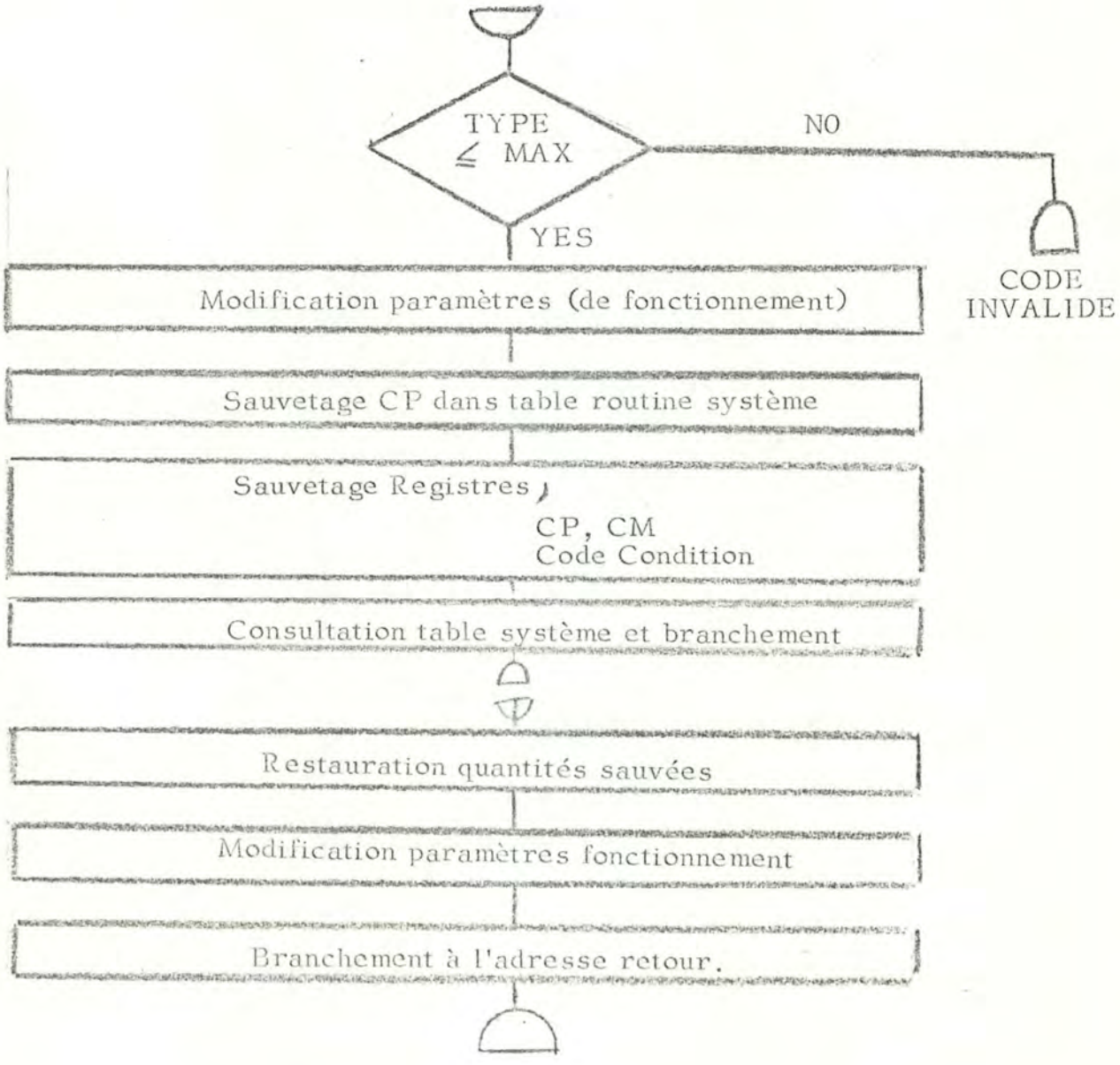

TYPE ≤ MAX
NO
YES
CODE INVALIDE
Modification paramètres (de fonctionnement)
Sauvetage CP dans table routine système
Sauvetage Registres ,
CP, CM
Code Condition
Consultation table système et branchement
Restauration quantités sauvées
Modification paramètres fonctionnement
Branchement à l'adresse retour.

Fig. I.9

CHAPITRE II - ENTREES/SORTIES

1. INTRODUCTION : CONCEPTS GENERAUX.

Avant de décrire le mécanisme de gestion des entrées-sorties adopté pour l'EPRON, il nous a semblé intéressant de rappeler ou de mettre en évidence certaines notions et distinctions assez fondamentales exploitées par EPRON.

1.1. IØ : Définitions - Niveaux.

Initialement, le transfert des données entre mémoire et périphériques (IN ou ØUT) s'exécutait directement sous le contrôle du programme de l'utilisateur : c'est lui qui envoyait les ordres au périphérique, et qui gérait le transfert des données. Deux conclusions s'imposaient :

Pendant toute l'exécution de l'IØ, le programme utilisateur était mobilisé pour la gestion de cette IØ. En raison des différences de base de temps (CPU et périphérique), cela occasionnait une PERTE de TEMPS CPU de plus en plus considérable au fur et à mesure que la technologie s'améliorait.

Le fait que la gestion des IØ soit directement sous le contrôle de l'utilisateur nécessitait de celui-ci une parfaite connaissance des caractéristiques et du fonctionnement des périphériques utilisés.
Dans ce cas, les risques d'erreurs étaient considérables, et parfois celles-ci se révélaient irrécupérables.

Pour tenter de remédier à ces inconvénients, deux interfaces ont été introduits : les canaux d'entrée-sortie (ou canaux IØ), et le superviseur des entrées-sorties.

Les CANAUX IØ ont pour but de réduire la perte de temps programme occasionnée par la gestion des IØ.
Suivant les machines, ces canaux possèdent une logique propre plus ou moins développée leur permettant d'assurer la gestion des IØ de façon plus ou moins autonome.
(A la limite, ils deviennent de véritables ordinateurs satellites).
Pour peu que la logique du programme le permette, nous pourrons avoir le déroulement simultané de plusieurs "processus" parallèles:

- le programme utilisateur géré par le CPU,
- les "programmes canaux" gérant les IØ en cours.

La communication se fera, dans le sens CPU-canal, au moyen des instructions IØ, et dans le sens canal-CPU, par les interruptions de programme réservées aux canaux.

N.B. Pour obtenir une réelle efficience de cette intervention des canaux, il est nécessaire que le programme soit pensé en fonction de ce parallélisme.
Si, une fois la demande d'IØ formulée, le programme se bloque dans l'attente de fin de cette IØ, les performances sont sérieusement réduites.
Plusieurs solutions peuvent être introduites :
le double buffering, le multitasking, ou la multiprogrammation.

En double buffering, l'utilisateur travaillera en bascule sur l'un des tampons (ou buffers), tandis que le canal remplira ou videra l'autre.

En multitasking, l'utilisateur partage son job en "tâches" indépendantes. Lorsque l'une d'elles se bloque sur l'attente d'une IØ, une autre démarre. Il est à noter que tous les programmes ne se prêtent pas à ce genre de "dissection".

La multiprogrammation permet de récupérer ce temps d'attente non plus au niveau du programme, mais au niveau du CPU : lorsqu'un programme est mis en attente d'IØ, le contrôle passe à un autre programme. Alors se pose le problème d'une judicieuse répartition de programmes : programme demandant beaucoup d'IØ, programme au contraire orienté CPU.

Le SUPERVISEUR des entrées/sorties constitue essentiellement un interface entre programme utilisateur et canaux IØ. En fonction des demandes du programme, c'est lui qui initialise les "programmes canaux", et qui assure la validité des data transférées (en entamant, quand nécessaire, les procédures de recouvrement d'erreurs).

Nous considérerons donc une opération IØ à 4 niveaux :

- le programme utilisateur,
- le superviseur IØ,
- les canaux IØ,
- les périphériques (eux-mêmes composés du périphérique proprement dit, et d'une unité de contrôle, ou coupleur).

A chacun de ces niveaux correspond un type d'action que nous allons brièvement évoquer. Après quoi, nous expliciterons les réalisations sur l'EPRON de chacun de ces niveaux et de leurs actions.

1.2. ROLES DES DIFFERENTS NIVEAUX.

1.2.1. Périphériques.

Le périphérique recevra, reconnaîtra et exécutera les ordres que lui envoie le canal IØ auquel il est connecté.
Ces ordres seront envoyés, soit sur des fils propres à chaque périphérique, soit sur des bus communs, mais accompagnés d'un signal d'identification du périphérique.
Ils auront pour but l'exécution soit d'une fonction de service (par exemple : rebobinage pour un dérouleur, ou saut de ligne pour imprimante), soit d'un transfert de données (c'est-à-dire envoi de ces données du support vers le tampon, ou du tampon vers le support.
Le transfert entre ce tampon et la mémoire centrale sera du ressort du canal.

1.2.2. Canal IØ.

Le canal IØ assure la gestion de l'opération IØ à partir de la demande du superviseur.
(Instruction d'entrée/sortie).
A partir de là, il devra exécuter :

1. la sélection et le test du périphérique (sélection nécessaire si le canal est multiplexeur);
2. l'envoi de l'ordre à exécuter;
3. le contrôle de l'exécution de celui-ci;
   Dans le cas de transfert de données, il régira le transfert entre la mémoire et le tampon du périphérique, et la mise à jour des paramètres nécessaires à ce transfert, et ce, jusqu'à réception du signal END émis par le périphérique.
   Dans le cas d'une fonction de service, il assurera l'exécution de la fonction jusqu'à détection d'une condition de fin (par exemple, un ordre de REWIND sera effectif jusqu'à détection de début de bande);
4. générer une demande d'interruption de programme à partir du signal de fin reçu du périphérique.

Cette séquence constitue le minimum attendu d'un canal.
Pour permettre une optimisation du fonctionnement du canal IØ, certains systèmes ont introduit des fonctions supplémentaires permettant soit un gain de place mémoire périphérique, soit un gain de temps dans la gestion d'entrée/sortie. (cf. IBM360, SIEMENS 4004). Parmi celles-ci, nous retiendrons le chaînage de données, le chaînage de commandes, la fonction SKIP, et la suppression du contrôle de longueur.

1.2.2.1. Chaînage de données.

Permet de stocker sur un même bloc physique des données provenant de zones mémoires différentes (ØUT), ou de transférer le contenu d'un même bloc physique vers des zones mémoires différentes (IN).
L'avantage de cette fonction consiste en un gain de place sur périphérique.
(sur une bande magnétique, il nous permet de récupérer l'espace destiné au GAP).
Il est à noter que le périphérique n'est pas conscient du changement de buffer mémoire : il ne voit que le transfert du bloc physique.
Cette fonction est illustrée par la fig. II.1.

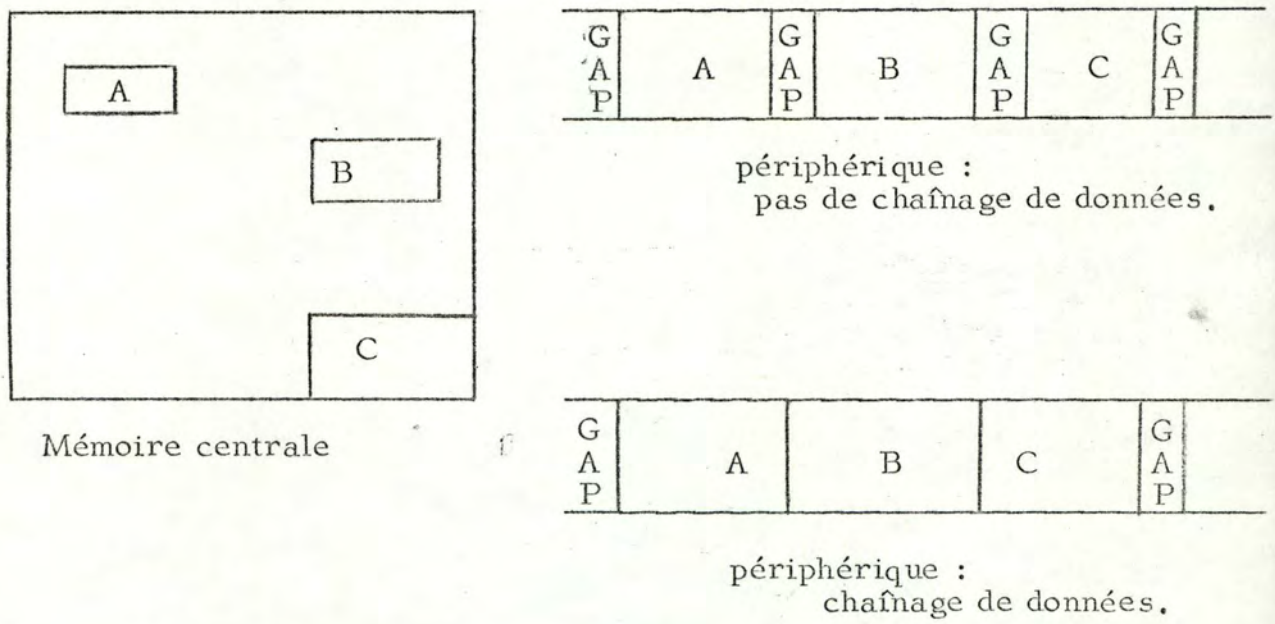


Fig. II. 1.

1.2.2.2. Chaînage de commandes.

Cette option a pour but de faire correspondre, à une instruction IØ, la succession de plusieurs ordres à destination du périphérique. C'est le canal qui assure le passage de l'un à l'autre.
Cette séquence d'ordres peut être, soit la répétition d'un même ordre, portant sur plusieurs blocs physiques, (par exemple, lecture de plusieurs blocs sur bande), soit l'exécution d'ordres différents (par exemple, pour l'imprimante, impression d'une ligne, suivie d'un saut à la page suivante).

L'intérêt de cette fonction est double.
Du point de vue CPU, il permet au programme de se dérouleur plus longtemps sans interruption (puisque l'interruption de programme n'interviendra que lorsque toute la chaîne de commandes aura été exécutée).

Mais l'avantage le plus important est sans doute d'optimiser le temps d'exécution d'IØ, sur périphériques rapides.

Par exemple, dans le cas d'un dérouleur de bandes, le chaînage de commande permet de franchir le GAP sans un arrêt total et redémarrage.

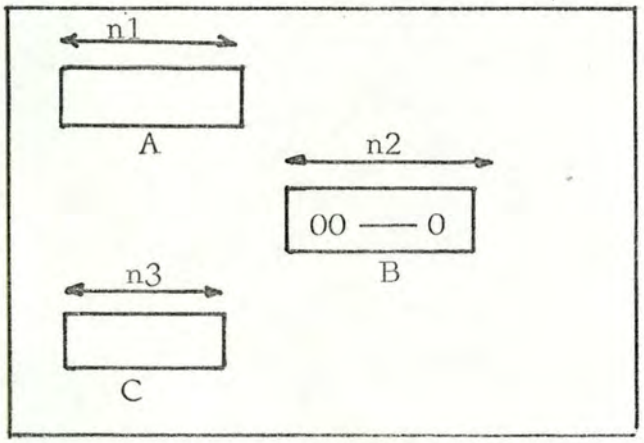
1.2.2.3. Fonction de SKIP.

Cette fonction permet d'inhiber l'enregistrement (en mémoire ou sur le périphérique) et ce sur une longueur spécifiée par ailleurs.

De même que le chaînage de données, cet ordre ne concerne en rien le périphérique, mais seulement l'échange canal - mémoire.

L'avantage, minime semble-t-il, est de regagner les cycles lecture ou écriture en mémoire.

La réalisation de cette fonction est illustrée par les figures II.2a et II.2b.



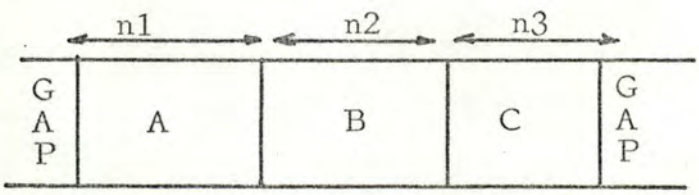
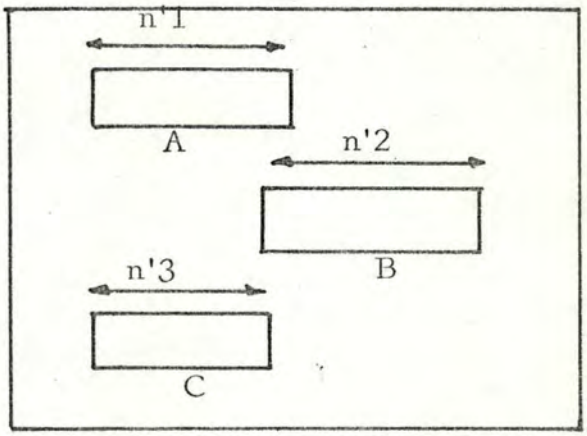


Fig. II.2a
SKIP en entrée de la partie B



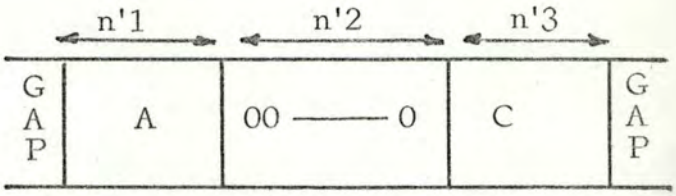


Fig. II.2b
SKIP en sortie de la partie B

1.2.2.4. Suppression contrôle de longueur (SLI).

Normalement la détection de "compteur de bytes ≠ 0" lors de la réception du signal END signifie qu'il y a erreur dans le transfert, puisque le bloc physique réel ne correspond pas au bloc physique attendu.
Pour les périphériques dont les blocs sont de longueur fixe, une telle erreur va provoquer une interruption d'erreur.
Par contre, pour les périphériques dont les blocs sont de longueur variable, il fallait contourner la difficulté.

En écriture, l'utilisateur peut prévoir avec exactitude la longueur du bloc écrit. Donc BC ≠ 0 est une cause d'erreur.

Mais en lecture, l'utilisateur prévoiera une longueur maximale. Si le bloc réel est plus court, l'interruption d'erreur pourra être masquée grâce à cette action SLI du canal.

1.2.3. Superviseur IØ.

C'est sur lui que l'utilisateur se décharge de la gestion des IØ, c'est-à-dire les communications avec les canaux appropriés, pour l'initialisation des opérations, les contrôles de bon fonctionnement et les éventuelles tentatives de correction en cas d'erreurs.

Par définition, il doit pouvoir accéder aux canaux et agir sur leur fonctionnement : c'est pourquoi il travaillera essentiellement en mode maître, ce qui lui permettra l'usage de la panoplie des instructions privilégiées.

1.2.4. Utilisateur.

Il fournit au superviseur des demandes d'exécution d'une fonction d'IØ ou ayant trait à une IØ (par ex. WAIT), et décrit les paramètres de cette fonction. En réponse à ces inputs fournis à une "Black Box", il reçoit le résultat de l'IØ.

2. GESTION DES ENTREES-SORTIES DE L'EPRON.

Ainsi que nous l'avons annoncé, nous envisagerons les entrées-sorties à quatre niveaux.

Leurs rôles respectifs et les intercommunications sont brièvement résumés à la fig. II.3.

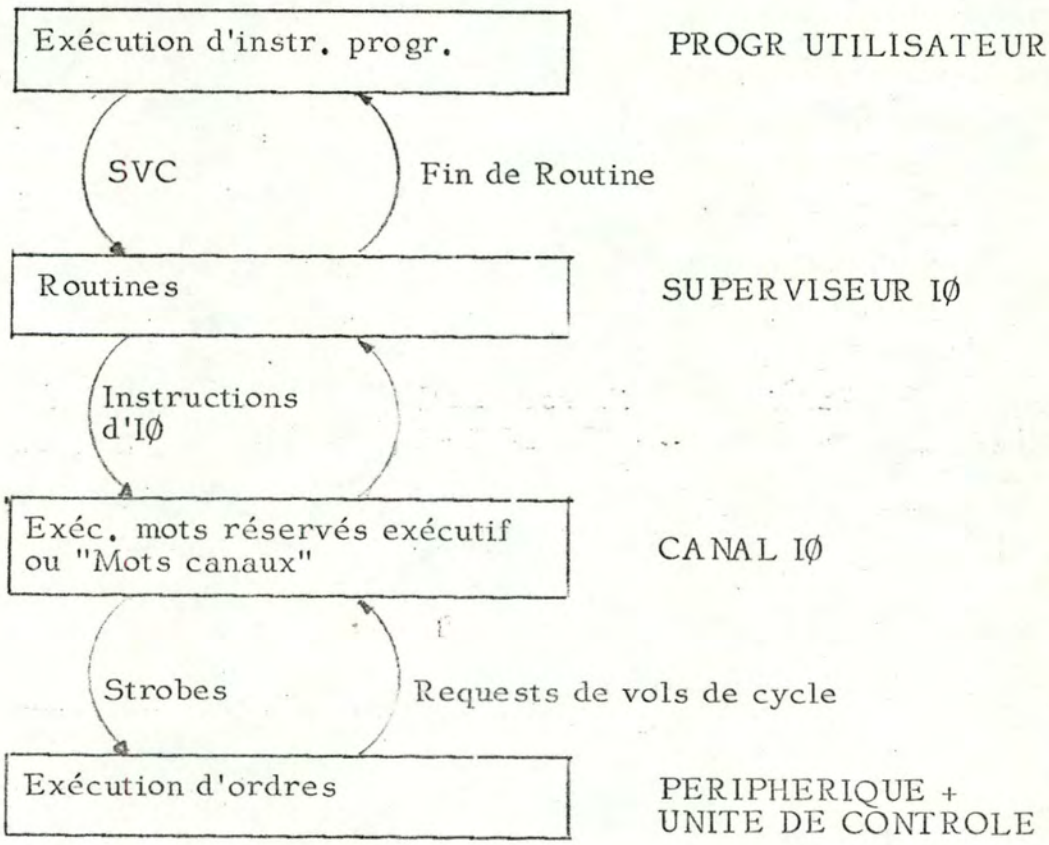


Fig. II.3.

2.1. PERIPHERIQUES.

Sous le nom global de périphérique, nous regroupons en fait deux entités distinctes : le périphérique proprement dit, et l'unité de contrôle associée ou coupleur.

La partie "périphérique proprement dit" est variable d'un périphérique à l'autre et comporte le support, l'électronique et la mécanique qui le gèrent et l'interface qui le relie à l'unité de contrôle. Nous laisserons aux spécialistes du hardware le soin de définir avec précision sa description et son fonctionnement.

Quant à nous, nous étudierons plutôt l'unité de contrôle, c'est-à-dire la partie que nous retrouverons sur tous les périphériques et qui mettra en branle le périphérique, à partir des signaux et ordres reçus du canal.

Dans cette unité de contrôle, nous trouverons :

2.1.1. REGISTRE DE DATA (ou tampon du périphérique).

Suivant le périphérique, sa dimension sera un byte, un mot ou un bloc (pour les périphériques dits "bufferisés").

Les transferts de données entre ce registre et la mémoire centrale s'effectue par les bus externes (ou Data Bus).

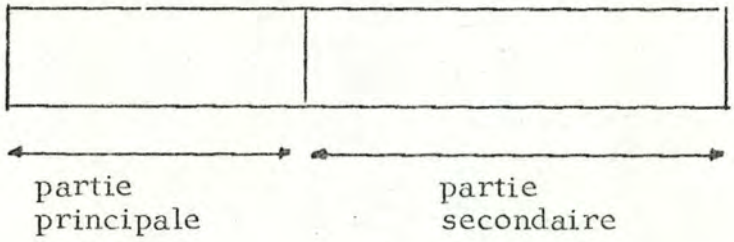
2.1.2. REGISTRE DE FONCTION.

Il est destiné au stockage de l'ordre envoyé par le canal. Le décodage de cet ordre par l'interface unité de contrôle - périphérique permettra la génération des impulsions activant le périphérique.

Le fait de rencontrer dans ce registre une combinaison invalide pour le périphérique entrainera le positionnement du bit correspondant du sensebyte (nous en reparlerons plus loin).

Les différents codes envisagés par périphériques sont résumés à la fig. II.4.

La configuration exacte de ces ordres est à définir de concert avec les responsables du hardware. Il serait intéressant de l'envisager sous la forme :



- La partie principale indiquant le type de fonction à réaliser, c'est-à-dire essentiellement :

  - READ
  - WRITE
  - SERVICE.

- La partie secondaire fournissant des précisions sur la façon dont cette fonction doit être réalisée.
  Par ex. : pour READ : Read Forward, Backward, Binary, EBCDIC, Read Tape Mark

Il est à noter que certaines fonctions peuvent être différenciées au niveau unité de contrôle, ou au niveau canal.

Par exemple, la reconnaissance d'un TM peut être seulement du ressort du canal, si l'on veut que l'unité de contrôle reste simple et standardisée.

ORDRES	Lect. cartes	Printer	Bande dér.	Disque
READ FORWARD			x	
BACKWARD			x	
BINARY	x			
EBCDIC	x			x
TAPE MARK			x	
IDENTIFIER				x
KEY				x
WRITE FORWARD			x	
BACKWARD			x	
BINARY				
EBCDIC		x		
TAPE MARK			x	
IDENTIFIER				x
KEY				x
ERASE (effact)			x	x
AVEC RELECT.			x	x
SAUT AV. 1 bloc			x	
AR. 1 bloc			x	
1 ligne		x		
A la page		x		
REWIND			x	
SEEK				x

Fig. II.4 - Ordres des périphériques

2.1.3. Registre STATUS.

Destiné à stocker le status byte du périphérique.
Celui-ci indique l'état du périphérique en rapport avec les signaux envoyés au (ou reçus du) canal IØ.

En effet, quand on considère le périphérique tel qu'il est vu par le canal, on peut le considérer comme un automate fini, dont l'état est indiqué dans le status byte, et modifié par l'envoi de signaux du périphérique au canal et vice-versa (par ce que nous appellerons plus loin les Requests et les Strobes).

2.1.3.1. Composition du status byte.

Nous nous arrêterons à cinq valeurs intéressantes du status byte :

READY, BUSY, DEVICE END, ERREUR, INOP.

- READY : signifie que le périphérique n'exécute aucune fonction, et qu'il est en état d'en prendre une en charge, sans aucune intervention de l'opérateur.

- BUSY : Positionné lors de la réception d'un ordre valide. Signifie que le périphérique est occupé à l'exécution d'un ordre, ou aux manipulations achevant cet ordre.
  Ex. : pour un lecteur de cartes, le temps d'éjection de la carte après lecture.

- DEVICE END : Positionné lorsque le périphérique a détecté la fin du bloc physique à traiter.
  Ex. : détection de fin de cartes, détection de GAP.
  En même temps, le Request END sera envoyé au canal.
  A partir de cet instant, le code stocké dans le regi stre fonction cesse d'être un ordre valable.

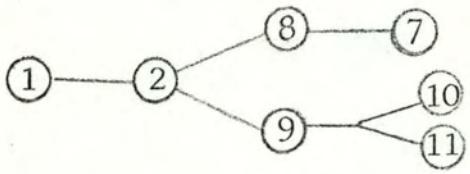
- ERREUR : positionné dès qu'un des bits du sense byte passe à 1.
  Indique que le périphérique a détecté une condition exceptionnelle ou une erreur.

- INOP : positionné lorsque le périphérique n'est pas en état de dialoguer avec le canal. Cela signifie qu'il est hors service, déconnecté, ou RESET.

N.B. : Certains introduisent aussi la notion de Mode travail MANUEL/AUTOMATIQUE, qui permet de détecter quand le périphérique travaille en local.
Il nous semble possible de recouvrir ce cas, par exemple, par la combinaison BUSY.AND.INOP.

2.1.3.2. Changement d'états.

Les changements d'états du périphérique en fonction des interventions manuelles, de la survenance d'erreurs ou de la réception de signaux du canal sont décrits à la fig. II.5.
Les changements autres que ceux retenus sont considérés comme invalides. L'apparition de l'un d'eux serait considérée comme une erreur du hardware de l'unité de contrôle.
Dans le cas d'une IØ se déroulant sans erreur, nous aurions la suite :



La séquence (a) correspond à une seule opération d'IØ , sur un périphérique tel que la manipulation mécanique après le transfert est plus longue que le dialogue avec le canal.
Ex. : cartes magnétiques.

La séquence (b) correspond au chaînage de commande : au lieu d'envoyer le strobe END, le canal réexcite le périphérique pour une nouvelle opération.

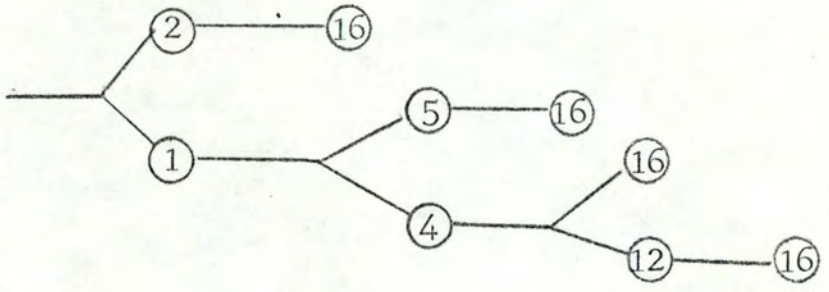
La séquence (c) correspond à une seule opération d'IØ, l'action mécanique se terminant avant l'échange avec le canal.

	Actions - Signaux	Etat présent					Etat suivant				
		READY	BUSY	END	ERR.	INOP	READY	BUSY	END	ERR.	INOP
1	Strobe FCT + ordre valide	1	0	0	0	0	0	1	0	0	0
2	Strobe FCT + odre invalide						0	0	0	1	0
3	Fin Bloc (⟹ REQUEST END)	0	1	0	0	0	0	1	1	0	0
4	Erreur en cours de bloc						0	1	0	1	0
5	Erreur en fin de bloc						0	1	1	1	0
6	Passage en Manuel (⟺ LOCAL)						0	1	0	0	1
7	Fin d'action mécanique (après END)						1	0	0	0	0
8	Strobe END avant Fin d'action mécénique	0	1	1	0	0	0	1	0	0	0
9	Fin d'action mécanique avant strobe END						0	0	1	0	0
10	Strobe Fct. (chaînage commande)	0	0	1	0	0	0	1	0	0	0
11	Strobe END (Acquittement IØ)						1	0	0	0	0
12	Fin IØ avec Erreur (⟷ $\overline{I_{AN}}$)	0	1	0	1	0	0	1	1	1	0
13	Fin action Locale	0	1	0	0	1	1	0	0	0	0
14	Déconnection, hors service	x	x	x	x	0	0	0	0	0	1
15	Action manuelle : remise en ordre de marche	0	0	0	0	1	1	0	0	0	0
16	Strobe FCT + ordre HIO.	0	x	x	1	0	1	0	0	0	0

N.B. : x = indifféremment 0 ou 1.

Fig. II.5

Par contre, pour les IØ avec erreur, nous aurions :



(a)

(b)

(c)

(d)

(a) l'IØ n'est pas lancée.

(b) l'Erreur n'est détectée qu'en fin de bloc.

(c) La détection d'erreur en cours de bloc entraîne l'arrêt de l'IØ (provoque interrupt d'anomalie).

(d) La détection d'erreur en cours de bloc ne sera prise en compte qu'à la fin du bloc.

2.1.4. Sense byte du périphérique.

Il intervient comme complément du status byte pour expliciter l'état du périphérique.

Mais tandis que le status byte reflète ce qui se passe au niveau de l'unité de contrôle (et possède une structure standardisée pour tous les périphériques), le sense byte indique les erreurs propres au périphérique proprement dit. Sa configuration varie donc suivant les périphériques.

Pour plus de détails concernant ce sense byte, nous renvoyons le lecteur à la routine du superviseur qui traite les recouvrements d'erreur.

La description de l'unité de contrôle ne serait pas complète si nous ne faisions pas mention de l'INTERFACE périphérique-canal. Signalons brièvement qu'il est constitué d'une part des Data Bus (ou bus externes), et d'autre part des signaux de commandes.

Les Data Bus permettent le transfert en parallèle de 8 bits (+ parité). Ces 8 bits peuvent avoir des significations différentes (donc provenir de ou être introduits dans des registres différents de l'unité de contrôle).
Ce sont les signaux de commande qui spécifieront la nature du contenu des data Bus . Ces signaux sont soit des strobes (envoyés au périphérique) soit des Requests (émis par le périphérique).

Le rôle exact de chacun de ces signaux sera décrit dans la partie traitant du canal.

2.2. CANAL IØ.

2.2.1. Description du canal IØ.

Au niveau hardware, l'EPRON ne possède pas de canal IØ tel que nous l'avons défini dans la première partie.
Cela signifie qu'il ne possède pas de hardware canal physiquement indépendant du CPU, et capable de fonctionner de façon totalement parallèle.

D'aucuns pourraient donc nous reprocher de parler de canal IØ. Cependant, il nous a semblé intéressant d'introduire cette terminologie pour désigner l'outillage firmware qui assure l'interface entre les périphériques et le superviseur d'IØ. En fait, le "programme canal" est greffé sur le mécanisme firmware de prise en charge des interruptions :
à chaque périphérique sont associés un certain nombre de "Vols de cycle" et de demandes d'interruptions de programme. (Nous y retrouvons les Requests Ri, RØ, END, I_{AN}, I_{APP}).
Pour le fonctionnement précis de la gestion des vols de cycle et des interruptions de programme, nous renvoyons au chapitre précédent.

Nous pourrions donner de ce mécanisme une vue plus directement orientée IØ, et dire que l'EPRON est muni d'un CANAL MULTIPLEXEUR, un sous-canal étant associé à chaque périphérique (au sens large c'est-à-dire périphérique + unité de contrôle).
Par l'utilisation des vols de cycle tels qu'ils ont été décrits, EPRON travaille essentiellement en MODE MULTIPLEX (au sens de IBM 360), c'est-à-dire que plusieurs instructions peuvent être exécutées en même temps sur des sous-canaux différents.

Dans la mesure où cela s'avérerait nécessaire, il serait possible, par une légère modification du hardware, de permettre aussi le BURST mode : il suffirait d'introduire pour les Requests de vol de cycle un système de masquage analogue à celui qu nous utilisons pour les interruptions de programme.

N.B. : Cette modification sera nécessaire lorsque nous raccorderons à l'EPRON des périphériques très rapides, par exemple, des dérouleurs de bande, et surtout des disques.
Le programme du canal IØ sera exécuté par les différentes mini-instructions logées dans les "Mots réservés" correspondant aux Requests externes.

2.2.2. Interface canal-périphérique.

Ainsi qu'annoncé, cet interface comprend, outre les Data Bus, un ensemble de signaux Requests et un ensemble de signaux Strobes.

Les REQUESTS, envoyés par le périphérique, entraînent le positionnement des bits correspondants dans le banc des Requests : ils ne seront pas nécessairement pris en charge directement puisqu'ils sont asynchrones par rapport au CPU.

Les STROBES sont envoyés par le canal aux périphériques, en général simultanément à l'envoi d'informations sur les Data Bus. Ces strobes identifient et valident ces informations.

Le jeu des Requests est composé de :

1. R_i (Data Transfer sequence INPUT)
   Le périphérique signale que son tampon est garni et prêt à envoyer son contenu en mémoire via les Data Bus.

2. R_0 (Data Transfer sequence OUTPUT)
   Le périphérique annonce que le tampon est prêt à recevoir des informations via les Data Bus.

3. END.
   Le périphérique annonce qu'il a détecté une fin de bloc physique. Il considère comme terminé l'ordre qu'il exécutait. Cela ne signifie rien quant à la validité des données transférées.

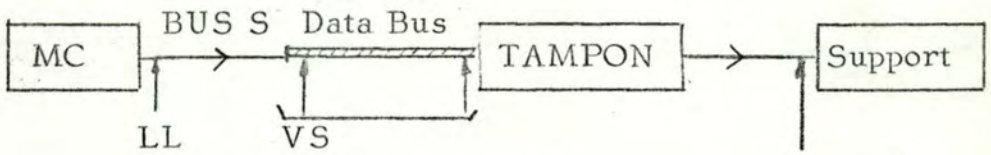
4. ANOMALIE.
   Nous renvoyons le lecteur à la définition p. I 2.

5. APPEL.
   Nous renvoyons le lecteur à la définition p. I 3.

Dans le jeu des strobes, nous trouvons :

1. DTS OUT (Réponse à R_0) ——→ Provoqué par Routine {IR, ITC, IT}
   permet l'identification des data à leur passage sur les bus externes, ainsi que l'ouverture du tampon du périphérique.

Le cheminement des data sera donc :
(en référence aussi à fig. I.1).

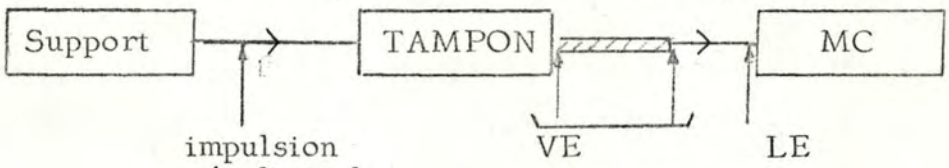


(Strobe DTS IN)

impulsion résultant de l'ordre présent dans registre FONCTION

2. DTS IN (Réponse à R_i) ⟶ provoqué par routine { OR / OTC / OT

permet l'ouverture du tampon du périphérique (donc transmission des Data sur les Data Bus), et le passage des Data Bus vers les Bus E.
Cheminement des data :



impulsion résultant de l'ordre

(Strobe DTS IN)

3. Strobe FONCTION. (ou strobe ORDRE)

Il valide l'envoi dans le registre Fonction via les Data Bus de l'ordre destiné au périphérique . Cet ordre sera l'une des valeurs envisagées à la fig. II4.

4. Strobe TERMINATE.

Le Canal envoie ce signal au périphérique lorsqu'il a détecté que le transfert devait se terminer (BYTE COUNT = 0).

Le rôle de ce signal variera suivant que l'on se trouve en face d'un périphérique d'entrée ou de sortie, à blocs de longueur fixe ou variable.

Dans le cas de périphériques d'entrée, ou de périphériques de sortie quand les blocs sont de longueur fixe, il suffira de commander l'arrêt du transfert des données : quand l'unité de contrôle reçoit ce signal, elle inhibe tout Request R_i ou R_0.

Si le périphérique est en sortie et travaille avec des blocs de longueur variable, l'unité de contrôle devra générer un signal de fin de bloc.
(par ex. : pour un dérouleur de bande magnétique, générer un GAP).

N.B. : Au cas où l'on voudrait réduire le nombre de fils entre le canal et l'unité de contrôle du périphérique il y aurait moyen de remplacer ce signal par l'envoi d'un ordre au périphérique (par la combinaison de Strobe FCT et ordre TERMINATE sur les Data Bus).

5. Strobe (Status byte
(Sense byte

Valide l'entrée en mémoire (toujours via les Data bus) du contenu du registre STATUS (respectivement SENSE) du périphérique.
Il résultera des opérations TIØ
TDV.

6. Strobe END.

Constitue la réponse au Request END du périphérique, et provoque son RESET.
Ce signal ne sera pas utilisé pour tous les périphériques.

Le lecteur nous reprochera peut-être l'abondance des signaux STROBE utilisés dans cet interface.

En fait, deux lignes de conduite étaient possibles :

- d'une part, nous multipliions les strobes, ceux-ci étant significatifs d'une fonction à exécuter par le hardware de l'unité de contrôle.

- d'autre part, nous réduisions le nombre de strobes, mais alors ils devaient être accompagnés d'information complémentaire. C'est ainsi que les strobes STATUS, SENSE, TERMINATE pourraient être supprimés et remplacés par l'envoi du strobe ORDRE accompagné de la valeur adéquate de l'ordre.
  Cette solution exigeait de l'unité de contrôle qu'elle puisse décoder et initialiser ces fonctions, cela nécessitait donc une unité de contrôle plus performante.

C'est pourquoi nous avons provisoirement retenu la première option. Mais s'il s'avérait nécessaire, le passage à la seconde optique se ferait aisément.

N.B. : Cet ensemble de signaux strobes et requests se retrouve pour chacun des périphériques.
Chacun d'eux recevra un numéro d'identification, défini, par la fonction d'une part, et le numéro de périphérique d'autre part.
Dans la suite du travail, le symbole "## " signifiera "numéro d'identification".

2.2.3. Routines du canal.

Par ce terme, nous désignerons les mini-instructions logées dans les "Mots Réservés Exécutif".

Ces mini-instructions peuvent être atteintes :

- à partir de Request du périphérique; dans ce cas, RAP contient automatiquement le n° d'identification (ou ##) de ce Request.
- à partir du software, par l'intermédiaire des instructions d'IØ. Il faut alors que le software assure le chargement de RAP : c'est ce dont se chargeront les instructions d'entrée-sortie initialisées par le superviseur.

Nous allons nous attacher dans une première partie, aux mini-instructions de réponse aux Requests, c'est-à-dire :

1. les vols de cycle DTS (IR, ØR, IT, ØT, ITC, ØTC).
2. les vols de cycle END (chaînage de commande).
3. les demandes d'interruption de programme (IP).

Nous verrons ensuite les mini-instructions initialisées par les instructions IØ. (STIØ, TIØ, TDV, ~~AIØ,~~ HIØ).

2.2.3.1. Vol de cycle DTS.

En réponse au Request R_i (ou R_Q), cette mini-instruction plus ou moins complexe exécute le transfert tampon-mémoire (ou mémoire-tampon) en un temps inversément proportionnel à sa complexité.

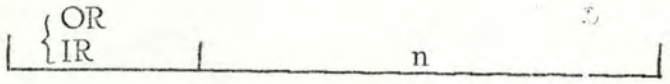
C'est ainsi que la première possibilité (IR, ØR) exécute le minimum de fonctions, transfert et progression d'adresse.

La possibilité suivante (IT, ØT), assure en plus un test sur le byte count, et le cadrage des caractères.
Elle s'adresse donc à des périphériques dont le mode de transfert est le caractère, et dont la cadence plus lente permet l'exécution d'une fonction plus performante.

Enfin, (ITC, ØTC) assure en plus le chaînage de Data : procédé plus coûteux en temps, mais qui permet un gain de place sur le périphérique. Il est à noter que l'option de pointeur (adressage indirect) réduit nettement la perte de temps lors du changement de paramètres.

Il est évident que d'autres possibilités existent soit en regroupant autrement les facteurs qui nous avons ici envisagés, soit en imaginant d'autres facteurs.

1°. { IR = INPUT } rapide
{ OR OUTPUT }



n = adresse du tampon mémoire centrale.

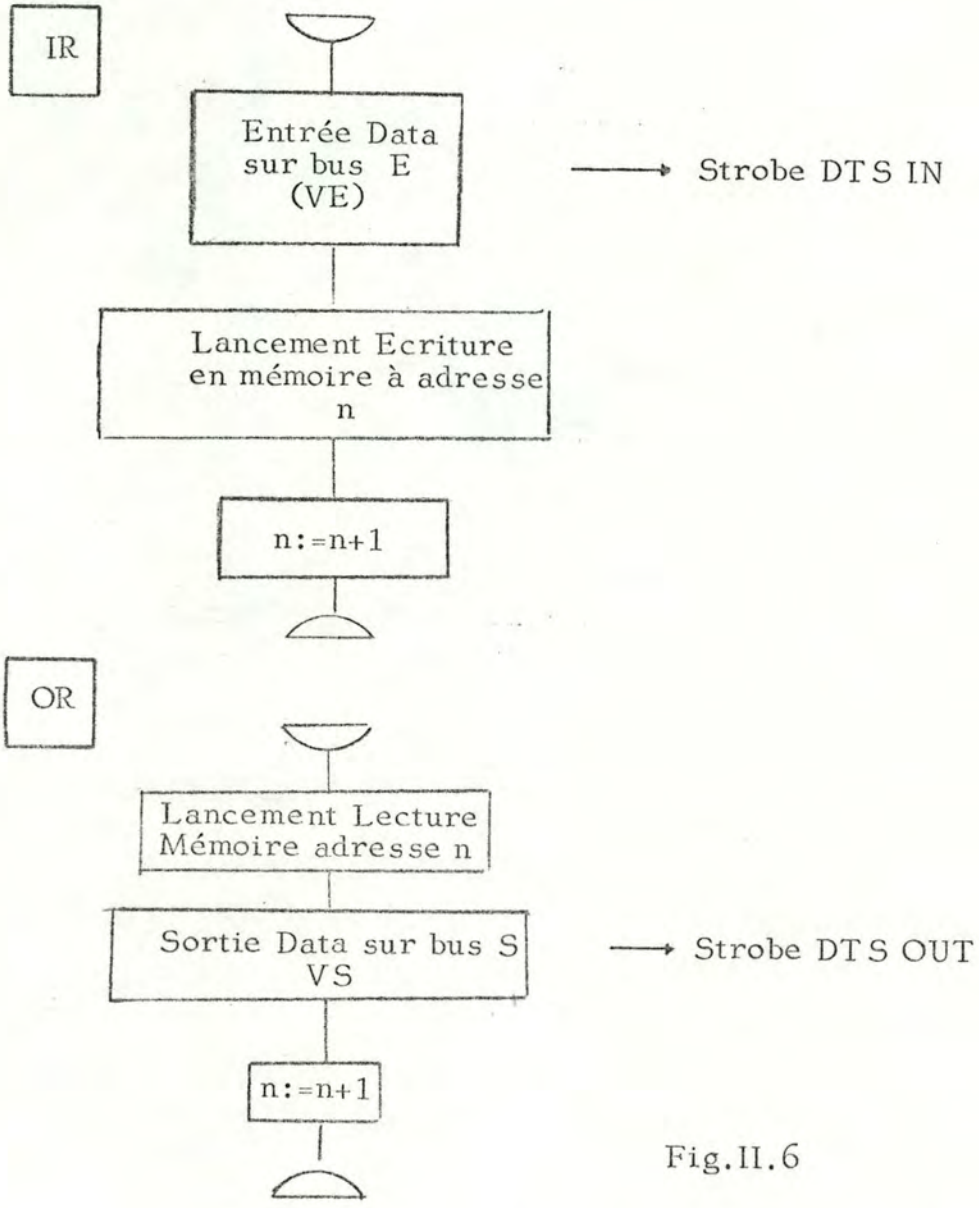


Fig.II.6

Avantage : opérations très rapides : 2 cycles mémoires (stockage Data, progression de n).

Inconvénient : - pas de test de fin
- transfert d'un mot (or la plupart des périph. : transfert par byte).

Conclusion : destiné seulement à certains périphériques rapides.

2°. IT (input avec test et cadrage).

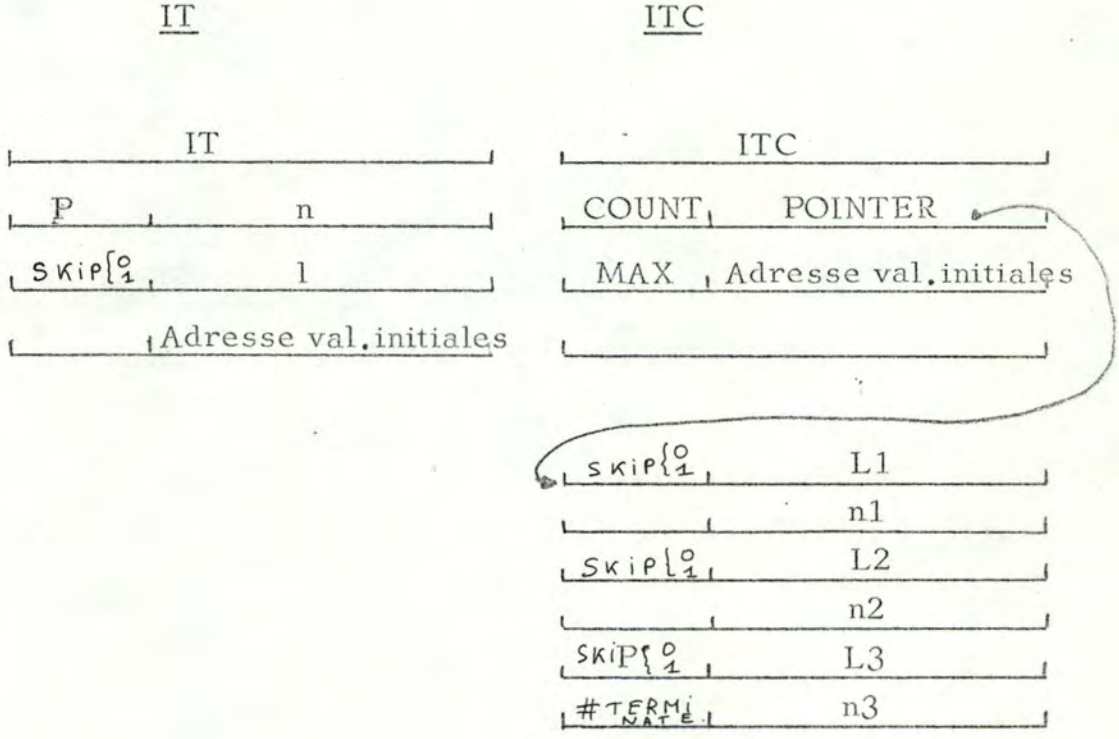
ITC (input avec test, cadrage, et possibilité de chaînage de données.

Fonctionnellement, les deux routines sont très semblables. La différence réside dans le fait que la seconde admet le chaînage de données. Cette option aura des répercussions sur la configuration des "mots réservés paramètres", et entraînera quelques modifications dans la routine firmware de prise en charge.

Ci-dessous, la description des mots réservés dans les deux cas.

La figure II.7 donne une description fonctionnelle de la routine ITC.

Dans le cas IT, nous aurions suppression du test "COUNT ? MAX", et passage par la branche EP p.



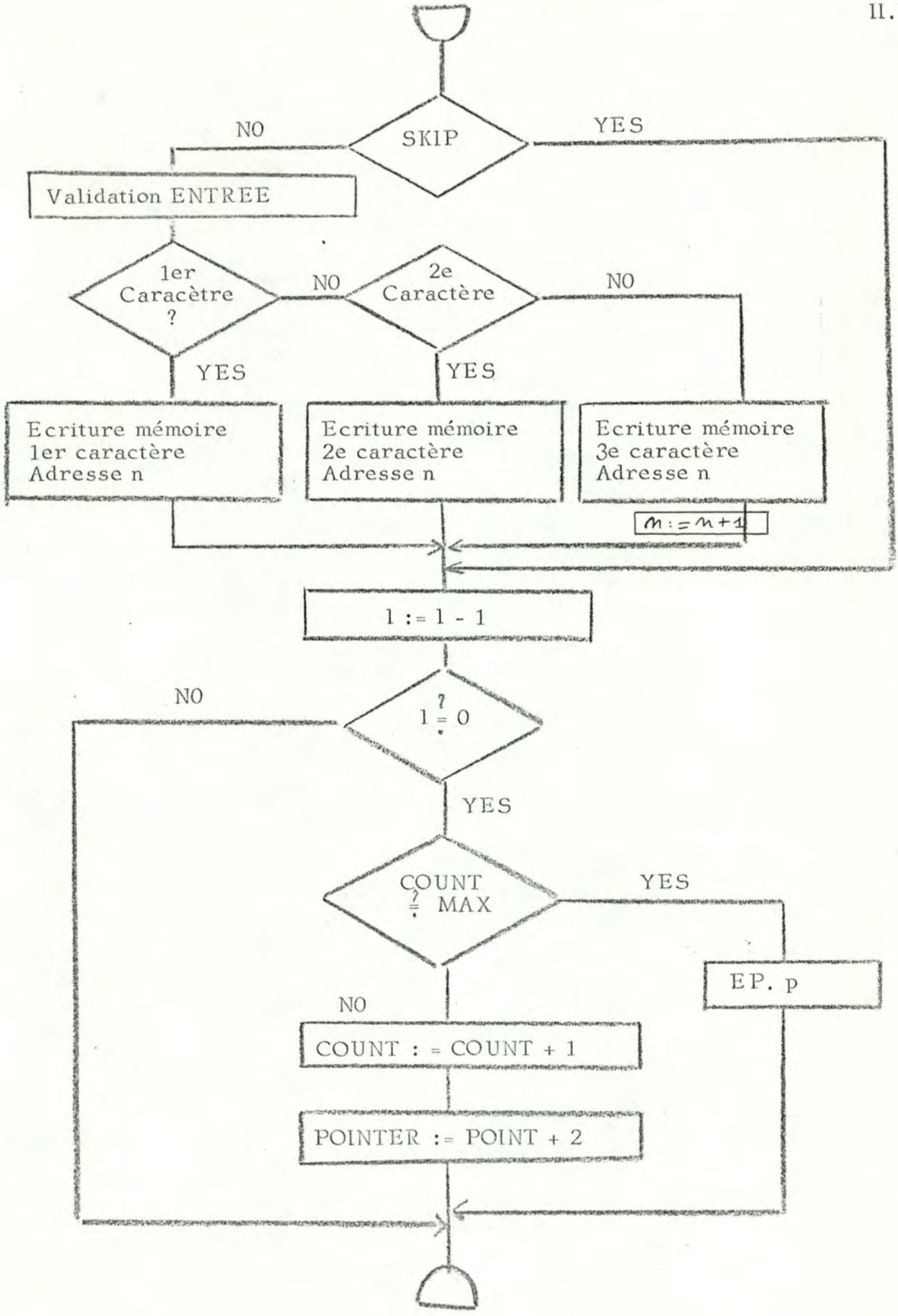

SKIP
NO
YES
Validation ENTREE
1er Caracètre ?
NO
2e Caractère
NO
YES
YES
Ecriture mémoire 1er caractère Adresse n
Ecriture mémoire 2e caractère Adresse n
Ecriture mémoire 3e caractère Adresse n
n := n+1
l := l - 1
NO
l = 0 ?
YES
COUNT = MAX ?
YES
EP. p
NO
COUNT : = COUNT + 1
POINTER := POINT + 2

Fig.II.7

l (ou l_i) = nombre de bytes à transférer.

n (ou n_i) = adresse du tampon en mémoire centrale (aussi appelé IØ AREA).

p = priorité en cas de fin (= valeur à stocker dans RAP pour envoi du TERMINATE).

Adresse valeurs initiales : adresse où l'utilisateur a stocké les valeurs initiales de ces paramètres (cf. 2.4). Cette adresse doit nous permettre de retrouver les valeurs initiales en cas de reprise de l'opération (si détection d'erreur) puisque l et n seront modifiés par les DTS.

pour ITC - pointer = adresse de la zone où sont stockés les paramètres d'une même chaîne.

- count = nombre de "maillons" dans la chaîne.

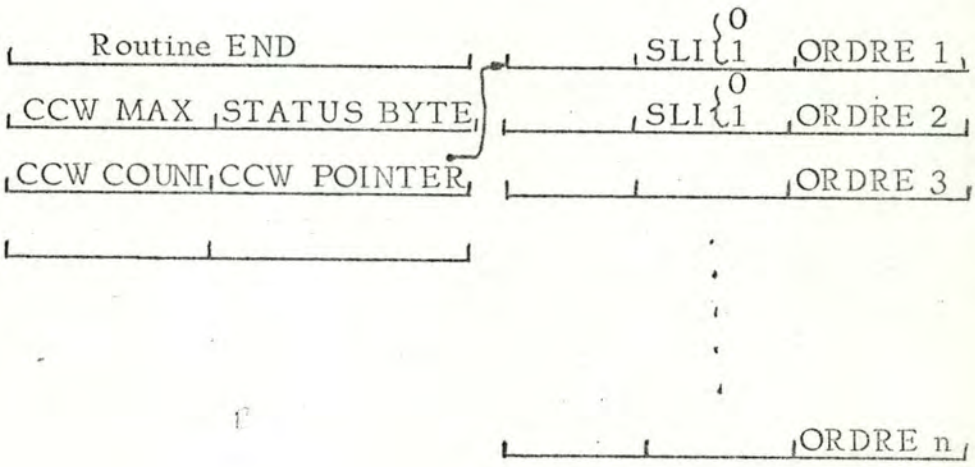
N.B. : pour une explicitation détaillée de ITC : cf. [D] p.II.11.

Les routines OT, OTC sont exactement symétriques de IT, ITC (Le strobe VS suivra l'opération de lecture mémoire).

2.2.3.2. Routine END.

Cette routine a pour but de permettre le chaînage de commandes. Dans ce cas, la prise en charge de l'ordre suivant doit être suffisamment rapide pour respecter les performances du périphérique (sinon, on perd tout avantage du chaînage de commandes).

Dans le cas de détection d'erreur en cours de chaîne, la chaîne est rompue, et la routine est abandonnée, après génération d'une demande d'interruption de fin de canal. C'est aussi ce qui se passe lorsqu'il n'y a pas de chaînage, ou que la fin de chaîne est détectée. La configuration des mots réservés sera celle décrite ci-dessous.
Quant à la routine elle-même, elle est décrite à la fig. II.8.



ROUTINE END

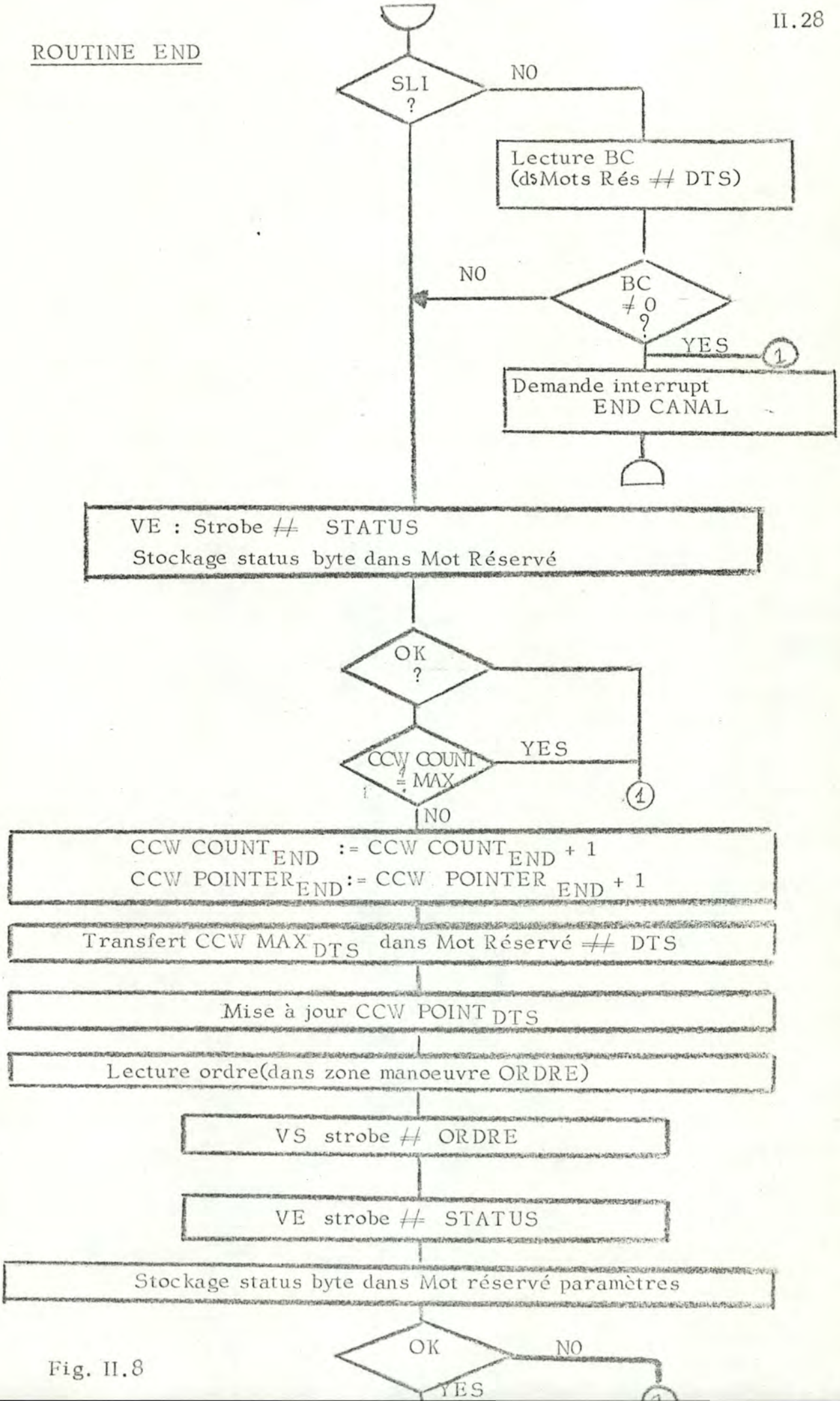
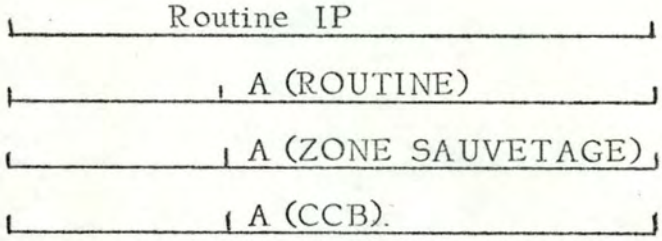


Fig. II.8

2.2.3.3. Interruptions.

Conformément à la solution suggérée au chapitre 1 (3.2.2.), les mots réservés associés aux routines de demande d'interruption auront la forme suivante :



La description de IP se trouve dans le chapitre 1
A (ROUTINE) constitue une référence au point d'entrée dans la routine du superviseur.
Elle peut être soit directement l'adresse de la routine, soit seulement le type de routine, auquel cas il y aurait consultation de la table du superviseur avant le branchement, ce qui allonge le temps de réponse, mais permet des modifications d'adresse des routines sans répercussion sur les mots réservés.

2.2.3.4. Instructions IØ.

Ces instructions sont à la paire canal-superviseur ce que les signaux Requests et Strobes sont à la paire périphérique/canal.

Exécutées dans les routines du superviseur, elles ont pour rôle de lancer l'exécution de routines du canal (c'est-à-dire de mini-instructions réservées au canal).

A la différence des DTS ou des END, ces routines sont donc lancées non plus par hardware mais par software. L'implémentation de ces instructions devra répondre à cet impératif. Une solution est proposée à la fig. II.9. Le FONCTIONNEMENT est le suivant :

1. comme pour toutes les instructions, le code de l'instruction est interprété par une mini-instruction. Rappelons que les instructions d'IØ font partie des instructions privilégiées. Le code ne sera donc validé que si BAMP = 0.

2. Pour interpréter ce code, le firmware utilisera le type (TIØ, HIØ, STØ, TDV,) et le numéro du périphérique. En fonction de ces paramètres, il provoquera l'exécution de la routine canal située dans le "mot Réservé IØ", à l'adresse :

   N + 4 * (TYPE + PERIPH)

   (N étant l'adresse du début de la zone des Mots Réservés IØ).
   Les paramètres de cette routine canal seront chargés lors du chargement du canal dans la fonction EXCP.

   La zone des "Mots réservés IØ" sera en fait le prolongement de la zone des Mots Réservés.

   L'inconvénient majeur de cette solution est d'être très prodigue en place mémoire . En effet, nous aurons 4 mots réservés par type d'IØ et par périphérique. Son avantage est de correspondre exactement à la définition.

   Pour réduire l'encombrement de la mémoire (ce qui sera sans doute souhaitable si le nombre de périphériques augmente de façon considérable), nous pourrions opter pour une solution moins élégante, mais plus économique : envisager une seule routine canal par type IØ (quel que soit le périphérique) ou une seule routine canal par périphérique (quel que soit le type).

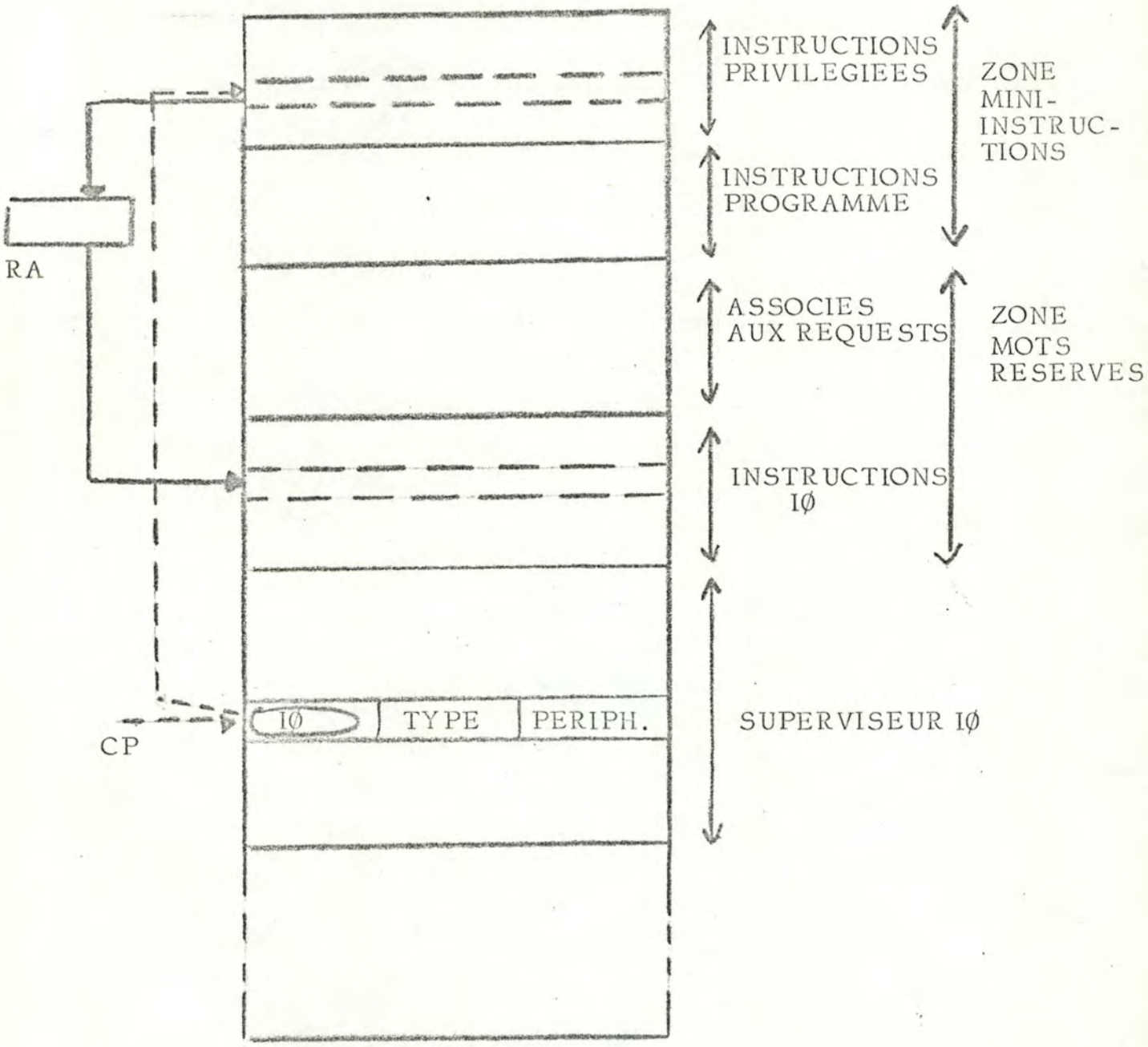

RA
CP
IØ
TYPE
PERIPH.
INSTRUCTIONS PRIVILEGIEES
INSTRUCTIONS PROGRAMME
ZONE MINI-INSTRUC-TIONS
ASSOCIES AUX REQUESTS
INSTRUCTIONS IØ
ZONE MOTS RESERVES
SUPERVISEUR IØ

Fig. II.9

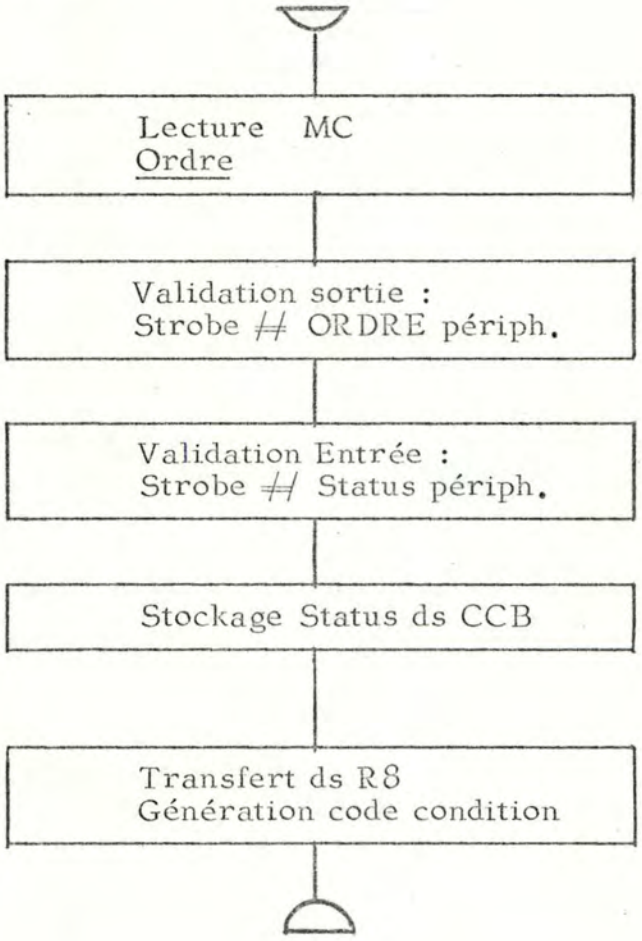
Les types d'instructions IØ seront les suivants :

STIO, TIØ, TDV, HIØ, AIØ.

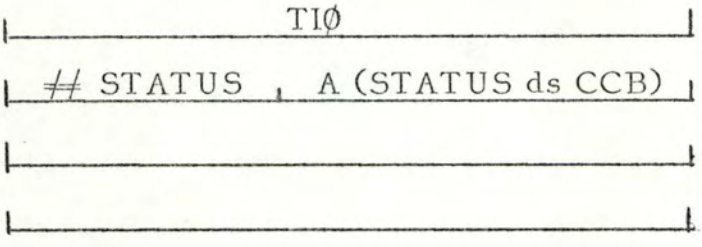
1. STIØ : provoque le lancement de l'opération par envoi de l'ordre au périphérique.

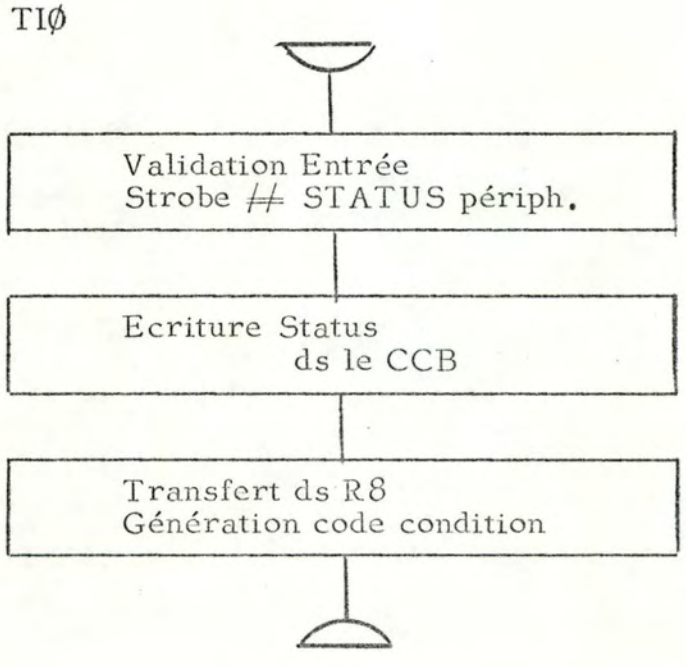
STIØ	
# ORDRE	A (ODRE)
# STATUS	A (STATUS ds CCB)

STIØ

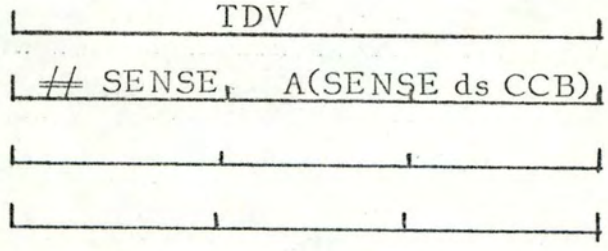


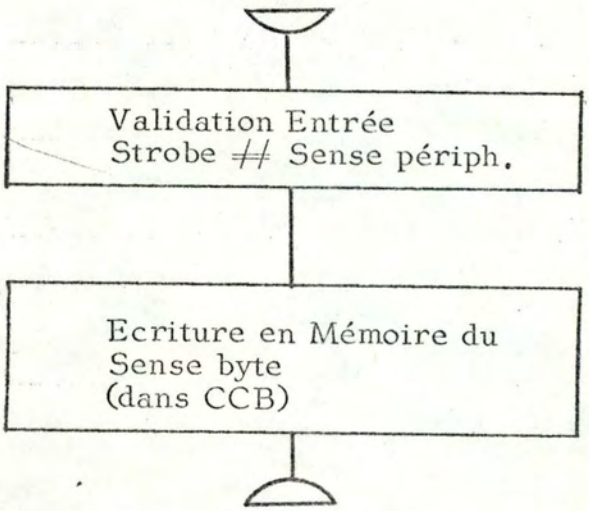
2. TIØ : Demande au périphérique son status byte et le range dans la zone prévue à cet effet dans le CCB.



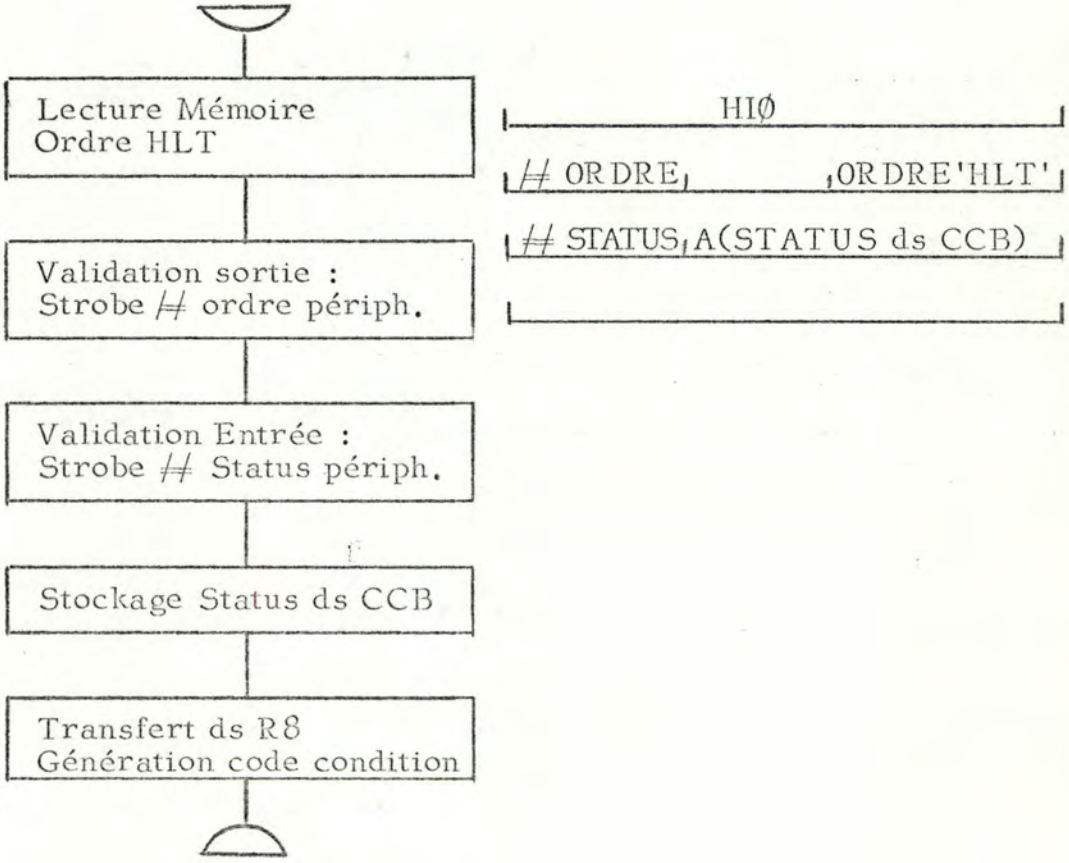


3. TDV : Demande au périphérique son (ses) SENSE BYTE(S) qui sera stocké dans le CCB.

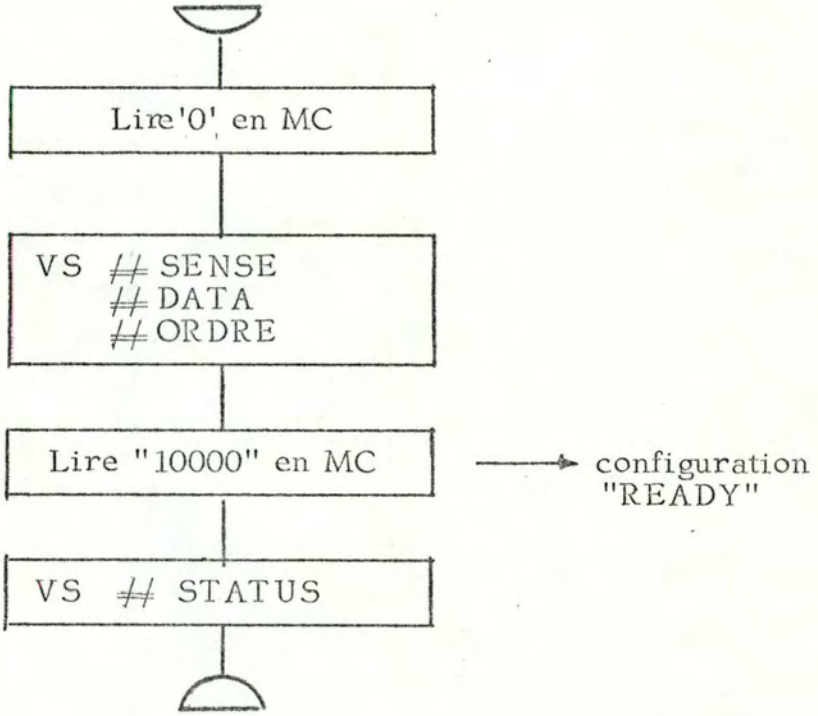


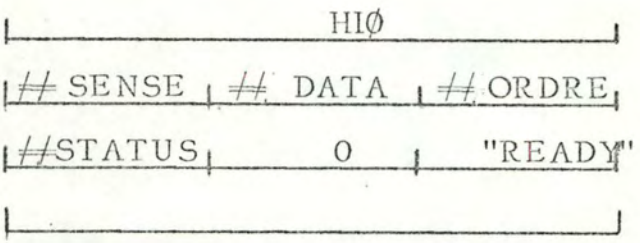


4. HIØ : a pour but de provoquer l'arrêt du périphérique, quelle que soit l'opération en cours.
Deux optiques sont possibles : d'une part, si l'unité de contrôle est suffisamment évoluée, l'instruction HIØ consiste seulement en l'envoi d'un ordre d'arrêt. Cet ordre décodé dans l'unité de contrôle, provoquera la mise à 0 des registres Data-Sense et le passage du périphérique à l'état READY.
Dans ce cas, la configuration des mots réservés et la fonction seront comme ci-dessous.



Si, d'autre part, nous voulons réduire les fonctions de l'unité de contrôle, c'est le canal qui agira sur chacun des registres. Dans ce cas, la fonction sera la suivante :





Code condition.

Il est à noter que l'exécution d'une instruction d'entrée-sortie provoque, ainsi que certaines autres, le positionnement du code condition. Ce code condition est établi à partir de la valeur du status byte, stocké dans R8.
Deux restrictions sont à formuler en ce qui concerne l'utilisation de ce code condition.

D'une part, l'utilité de ce positionnement est assez réduite, puisque les zones de travail du pseudo canal IØ (les mots réservés paramètres) sont directement accessibles au système . Il est donc possible pour celui-ci de tester la valeur du status byte à partir des mots réservés. Par contre, pour permettre à l'utilisateur d'effectuer ces tests, il faudrait assurer le stockage du status byte dans le CCB.
L'utilisation du code condition permet donc d'accélérer ces tests.

D'autre part, puisque le code condition peut être positionné par d'autres instructions, sa valeur par rapport à l'entrée/sortie est provisoire. C'est pourquoi il est souhaitable de stocker soit sa valeur soit celle du status byte dans le CCB.

Ces restrictions posées, nous avons envisagé le positionnement du code condition de la façon suivante :

Status byte	Code condition
END DEVICE	1
OCCUPE	1
PRET	0
ERREUR	2
INOPERABLE	3

Gestion des paramètres du canal IØ.

1. Origine.

Ces paramètres sont fournis au canal par le superviseur qui les prélève dans ses zones de travail.

Deux catégories doivent être distinguées :

1. les paramètres destinés à des fonctions utilitaires du canal, c'est-à-dire des fonctions qui ne traitent ni de l'envoi ni de l'exécution d'un ordre au périphérique.

   Par ex. : la demande du status byte, (ou du sense byte) provoquée par TIØ (ou TDV), n'attend comme paramètre que l'adresse où stocker le STATUS (SENSE) byte.

   Ces paramètres conserveront une valeur constante tout au cours du "programme canal" : chargés au début de l'EXCP, il resteront constants. Leur valeur sera connue à partir du CCB, dont nous reparlerons, en tant qu'interface superviseur-programme.

2. les paramètres destinés à l'exécution d'ORDRES par le périphérique. Ils seront prélevés à partir de "mots de commande" ou CCW, dont la configuration est la suivante :

| ORDRE | FLAGS | ADRESSE | LONGUEUR |

ORDRE : constitue l'ordre à envoyer au périphérique. Il prendra une des valeurs proposées à la fig.II. 4

FLAGS : destinés au canal, ils permettent à celui-ci d'exécuter ou non une séquence élaborée, avec possibilité de :

- chaînage de données,
- chaînage de commandes,
- SKIP,
- SLI

(la signification de ces flags a été définie au début de ce chapitre).

ADRESSE : Dans le cas d'un ordre de transfert, cette adresse est celle de la zone mémoire où doit être lu ou écrit le bloc. Dans le cas d'un ordre de service, elle fournit les renseignements complémentaires indispensables à l'exécution de l'ordre.

Ex. : pour un SEEK, ADRESSE fournit l'adresse de la piste à laquelle il faut accéder.

LONGUEUR : Dans le cas d'un ordre de transfert, il indique le nombre de bytes à transférer.

Dans le cas d'un ordre de service, il indique la portée de cet ordre de service.

Ex. : saut avant de n blocs sur bande magnétique.

En ce qui concerne l'origine de ces CCW, deux options sont possibles :

1. ils sont décrits directement par l'UTILISATEUR, qui signifie ainsi le détail de l'IØ qu'il veut exécuter.

2. ils sont générés par le SUPERVISEUR ou l'ASSEMBLEUR à partir de paramètres plus globaux fournis par l'utilisateur, et de squelettes de séquences fournis par le système.

   Cette deuxième optique s'insère davantage dans notre conception des IØ à 4 niveaux. Mais son inconvénient est de coûter cher en temps (génération de la chaîne) et en place (stockage des "squelettes"). C'est pourquoi, nous réaliserons sans doute la première solution.

2. Chargement (en cours de traitement).

En réponse à la demande d'entrée/sortie (par appel EXCP de l'utilisateur), le superviseur effectue ce que nous appellerons le "chargement du programme canal", c'est-à-dire le garnissage des mots réservés paramètres correspondant aux priorités susceptibles d'intervenir au cours de cette IØ.

La complexité de ce chargement variera d'une part avec la puissance des routines du canal, et d'autre part avec "aération de la mémoire".

Plus la routine du canal est évoluée et nécessite de nombreux paramètres, plus le chargement de ses paramètres se compliquera (cf. chaînage de Data, de commandes).

Nous avons vu aussi que, pour réaliser des économies de place en mémoire, nous pourrions être amenés à regrouper certaines routines avec l'obligation de manipuler certains paramètres supplémentaires. Citons le cas des instructions d'IØ, avec l'hypothèse suivante : 1 routine canal par type d'opération. Nous voyons que, dans ce cas, à chaque utilisation de ces routines les # des Strobes doivent être réajustés.

La façon de stocker les paramètres dépend de leur utilisation au cours de l'opération d'IØ. Nous distinguerons deux catégories;

- les paramètres qui ne sont pas modifiés en cours d'IØ (par ex. l'ORDRE envoyé au périph.), auquel cas il suffit de fournir l'adresse de ces paramètres dans le CCB, bien que le fait de fournir la valeur permette une accélération du traitement, puisqu'il réduit le nombre d'accès mémoire (Il n'existe pas d'adressage indirect dans le Hardware).

- les paramètres modifiés en cours d'IØ (par ex. l'ADRESSE MEMOIRE, ou le byte count). Dans ce cas, il faut les stocker dans une zone autre que le CCB, car il est indispensable de conserver leur valeur initiale en cas de mauvais fonctionnement nécessitant une reprise.

Nous allons donc examiner en quoi consiste ce chargement pour ces différentes routines du canal envisagées précédemment, c'est-à-dire :

1. Réponse aux Requests:

   DTS, END, APPEL, ANOMALIE

2. Instructions IØ :

   STIØ, TIØ, MIØ, TDV, AIØ.

DTS Ces paramètres sont modifiés en cours d'1Ø, il est donc nécessaire de stocker leur valeur dans les mots accessibles au canal.

1. dans le cas où il n'y a ni chaînage de données, ni chaînage de commandes, les valeurs du Byte COUNT, Adresse Mémoire, ainsi que l'option SKIP seront chargés dans les mots réservés, dont la configuration sera celle définie pour IR/ØR (cf. fig. II.6).

2. dans le cas de chaînage de données, le stockage des valeurs "modifiables" se fera soit globalement, soit progressivement. La première option donnera lieu à l'organisation cf. fig. II.11a, et permettra un gain de temps d'exécution de la DTS. La seconde générera l'organisation représentée à la fig. II.11b et permettra un gain de place Mémoire.

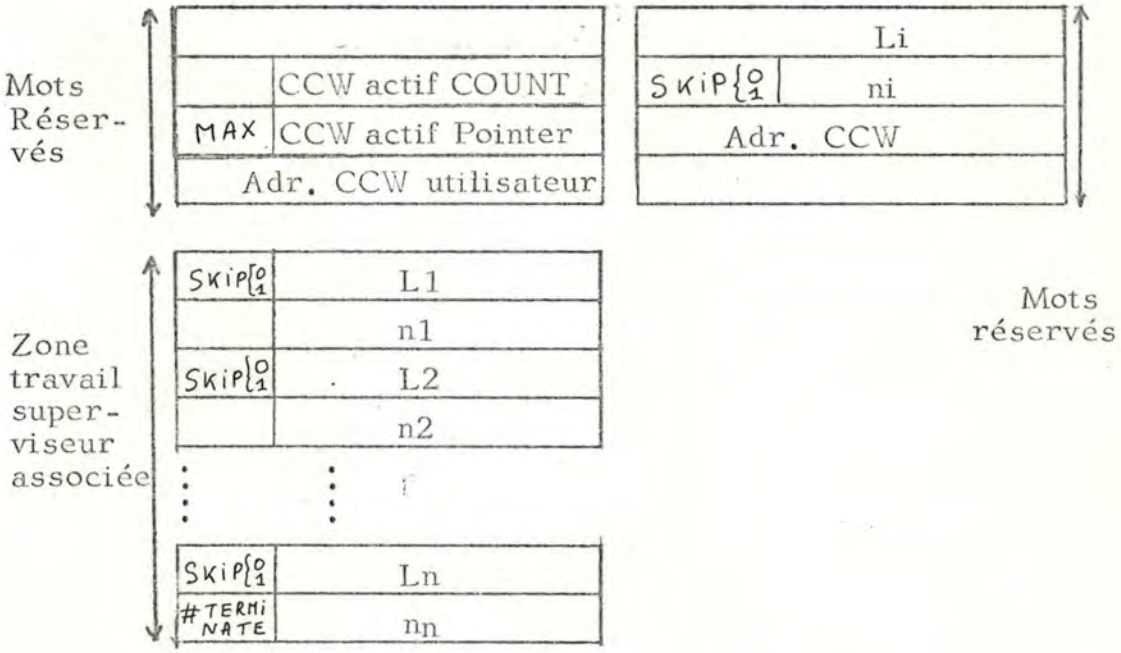
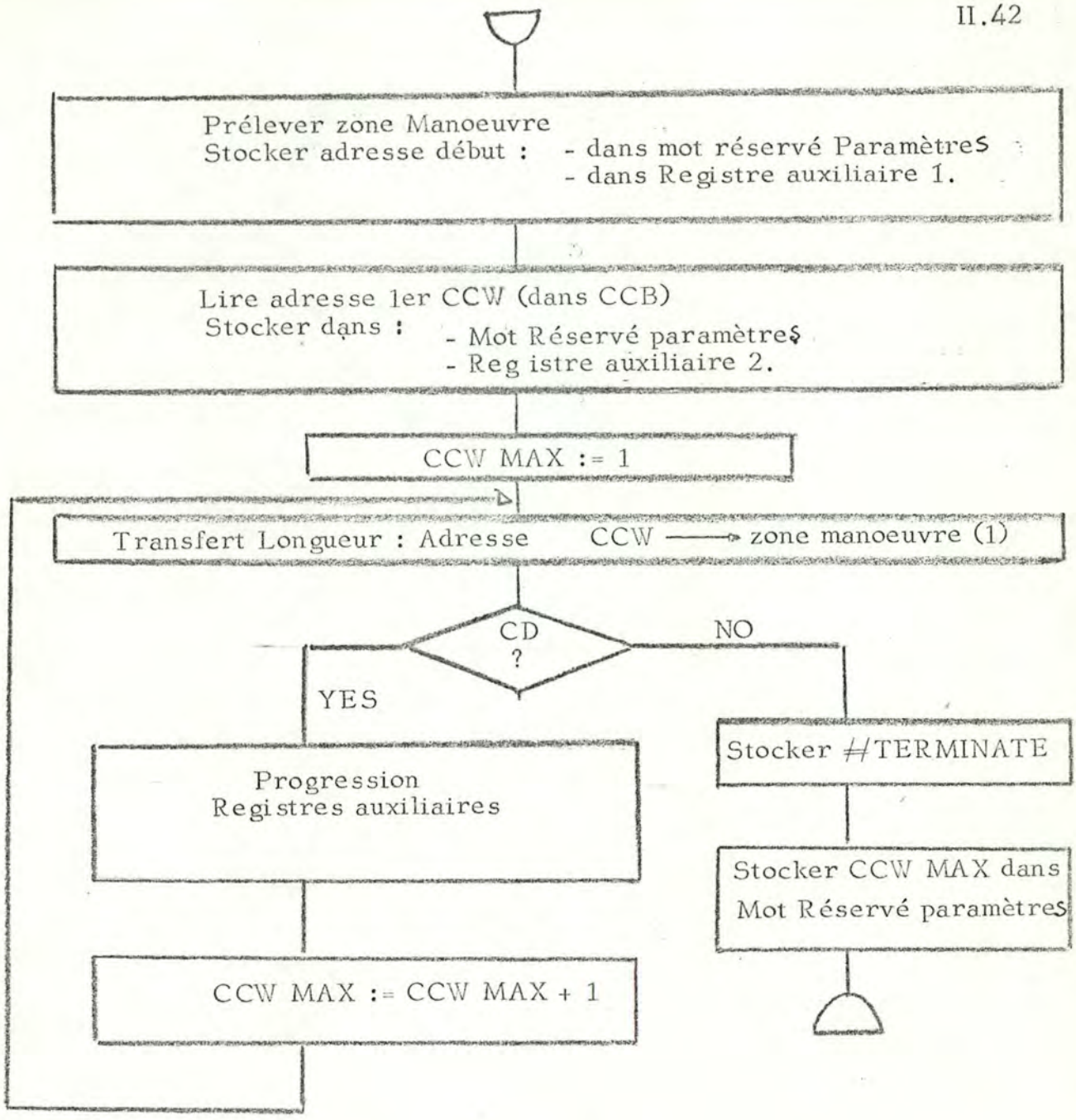


Fig. II.11a Fig. II.11b

Dans le premier cas, le chargement des mots réservés sera exécuté au début de la routine EXCP.
Il sera plus lourd que dans l'autre cas, mais permettra un passage plus rapide d'un paramètre à l'autre.
(dans certains cas, une cadence inférieure ne permettrait pas le respect de cadence d'arrivée des Requests du périphérique).
Le chargement se fera suivant l'organigramme représenté à la fig. II.12.



(1) Par l'intermédiaire des registres auxiliaires :

- Registre auxiliaire 1 : Adresse CCW courant
- Registre auxiliaire 2 : Adresse courante zone manoeuvre.

Fig. II.12

Pour la deuxième option, le chargement se fera à chaque prise en compte de nouvelles valeurs (c'est-à-dire reconnaissance de BC = 0).
Il permet une procédure plus simple (pas de nécessité d'utiliser zone de manoeuvre, pas de calcul du nombre de CCW) et occasionne un gain de place mémoire (pas de reproduction de la chaîne des CCW).
Par contre, le transfert des valeurs occasionne une perte de temps, et ce à un moment plus critique que dans le premier cas.

3. Dans le cas de chaînage de données et de commandes, les chaînes DTS sont imbriquées dans les chaînes de commandes. Nous verrons donc le détail du chargement dans le chargement des ordres chaînés.

END (avec chaînage de commandes).

Les paramètres à charger sont les ordres à envoyer au périphérique.
Dans le cas où le chaînage de commandes n'est jamais employé simultanément au chaînage de données, il n'est pas nécessaire d'opérer une copie de la chaîne des ordres (on utilisera l'option adressage indirect).
Mais cette contrainte étant rarement satisfaite, nous exploiterons plutôt le cas où les deux types de chaînage sont mêlés. Il est alors nécessaire de "décanter les deux chaînes", puisqu'elles s'adressent à des priorités différentes.
Le chargement s'effectuera comme décrit à la fig.II.13, et donnera lieu à l'organisation définie à la fig.II.14.

N.B. : la méthode exposée se base sur les restrictions suivantes

1. il n'existe pas de chaînage de commande conditionnel. Tant qu'il n'y a pas rupture de la chaîne pour cause d'erreur, la routine END accède automatiquement à l'ordre suivant.
   Pour assurer le passage à un ORDRE en fonction de la réalisation de conditions, il nous faudrait augmenter la complexité de la routine END, c'est-à-dire son extension, or, il nous faut assurer toute la fonction en une mini-instruction.

2. Dans l'accès aux CCW, nous supposons que les CCW constituants d'une même chaîne sont situés dans des zones mémoires consécutives (sinon, l'accès au CCW suivant poserait un problème supplémentaire, facilement résolu, mais occasionnant une nouvelle perte de temps.

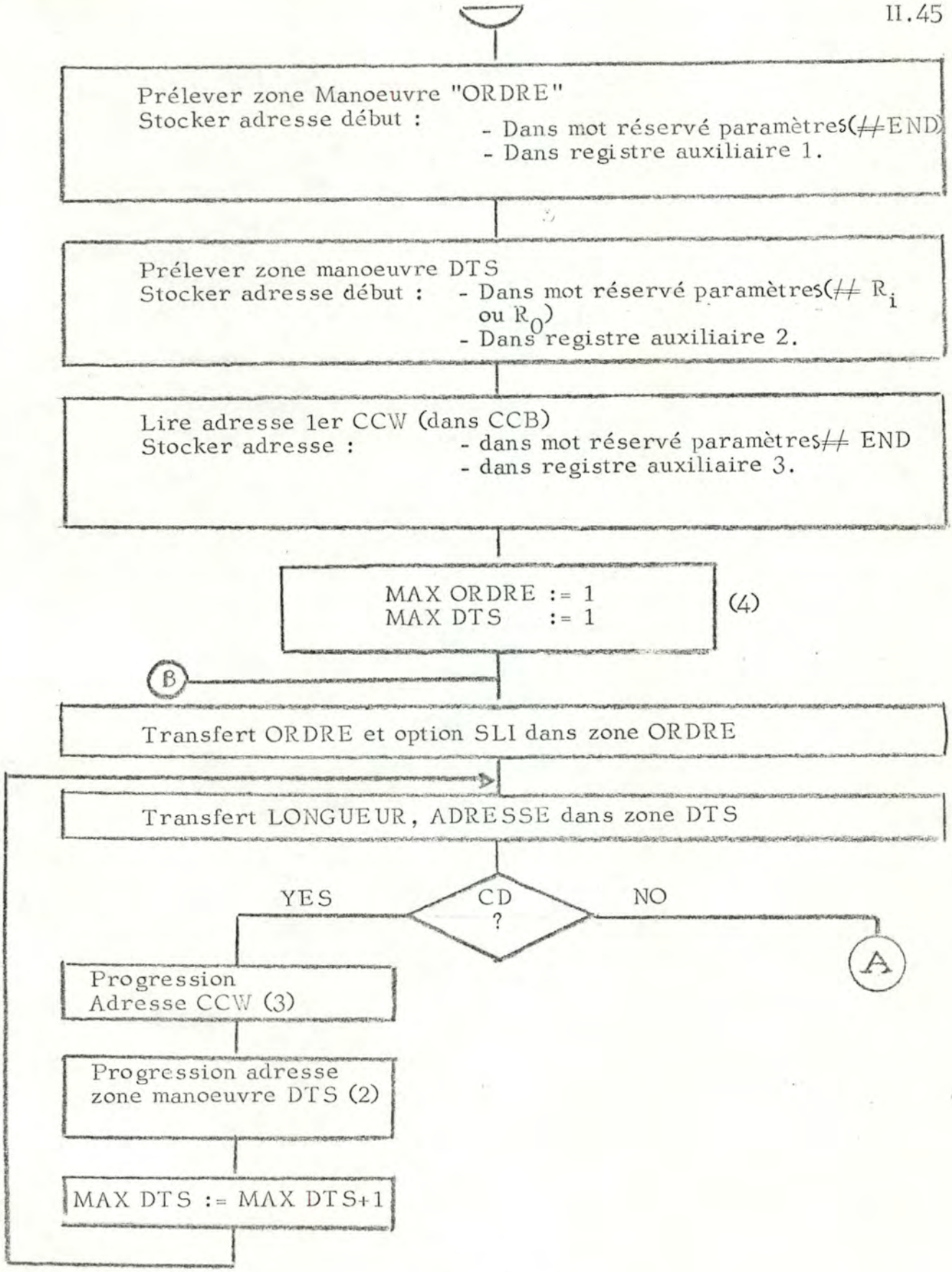
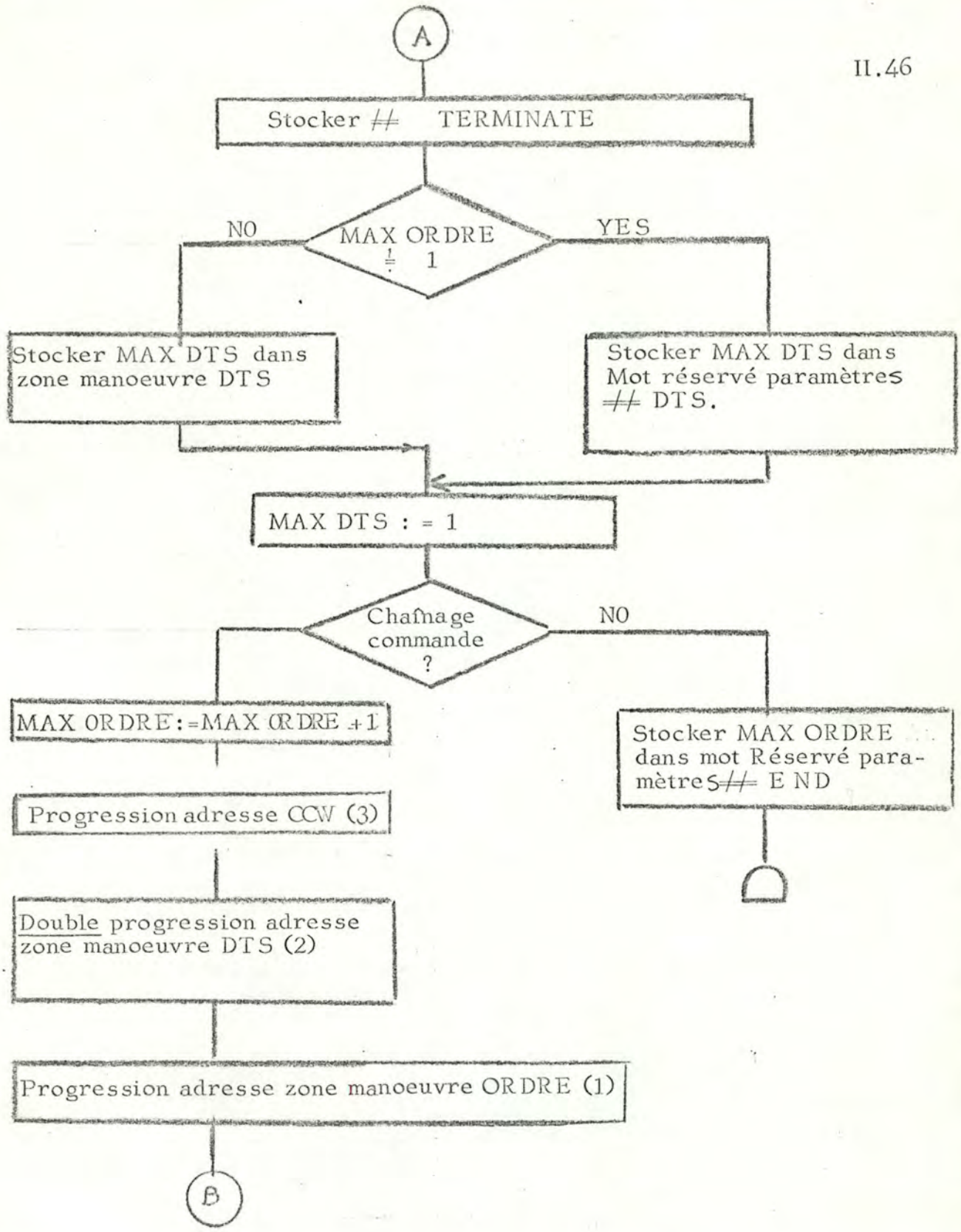
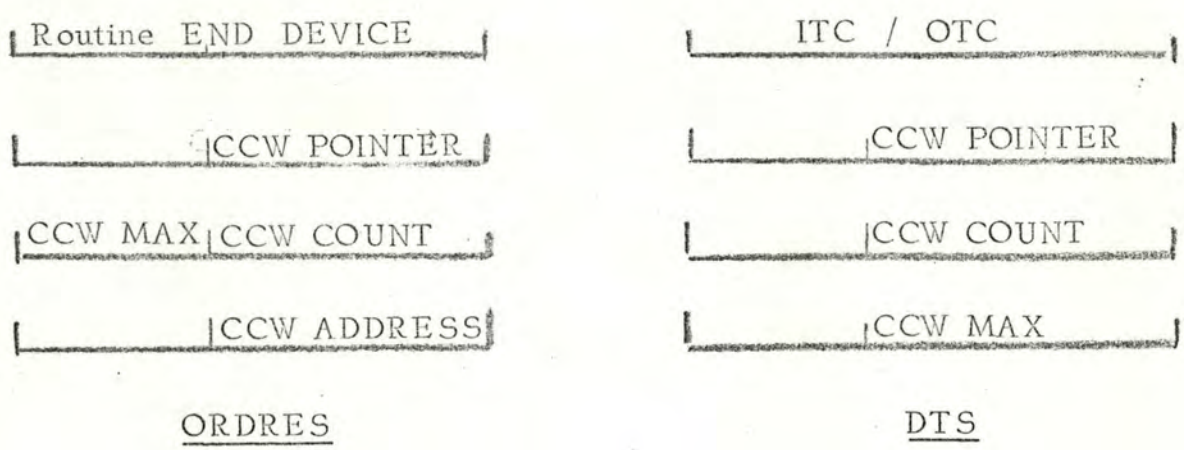

Prélever zone Manoeuvre "ORDRE"
Stocker adresse début :
- Dans mot réservé paramètres(// END)
- Dans registre auxiliaire 1.
Prélever zone manoeuvre DTS
Stocker adresse début :
- Dans mot réservé paramètres(// Ri ou R0)
- Dans registre auxiliaire 2.
Lire adresse 1er CCW (dans CCB)
Stocker adresse :
- dans mot réservé paramètres// END
- dans registre auxiliaire 3.
MAX ORDRE := 1
MAX DTS := 1
(4)
B
Transfert ORDRE et option SLI dans zone ORDRE
Transfert LONGUEUR, ADRESSE dans zone DTS
CD ?
YES
NO
A
Progression Adresse CCW (3)
Progression adresse zone manoeuvre DTS (2)
MAX DTS := MAX DTS+1

Fig. II.13

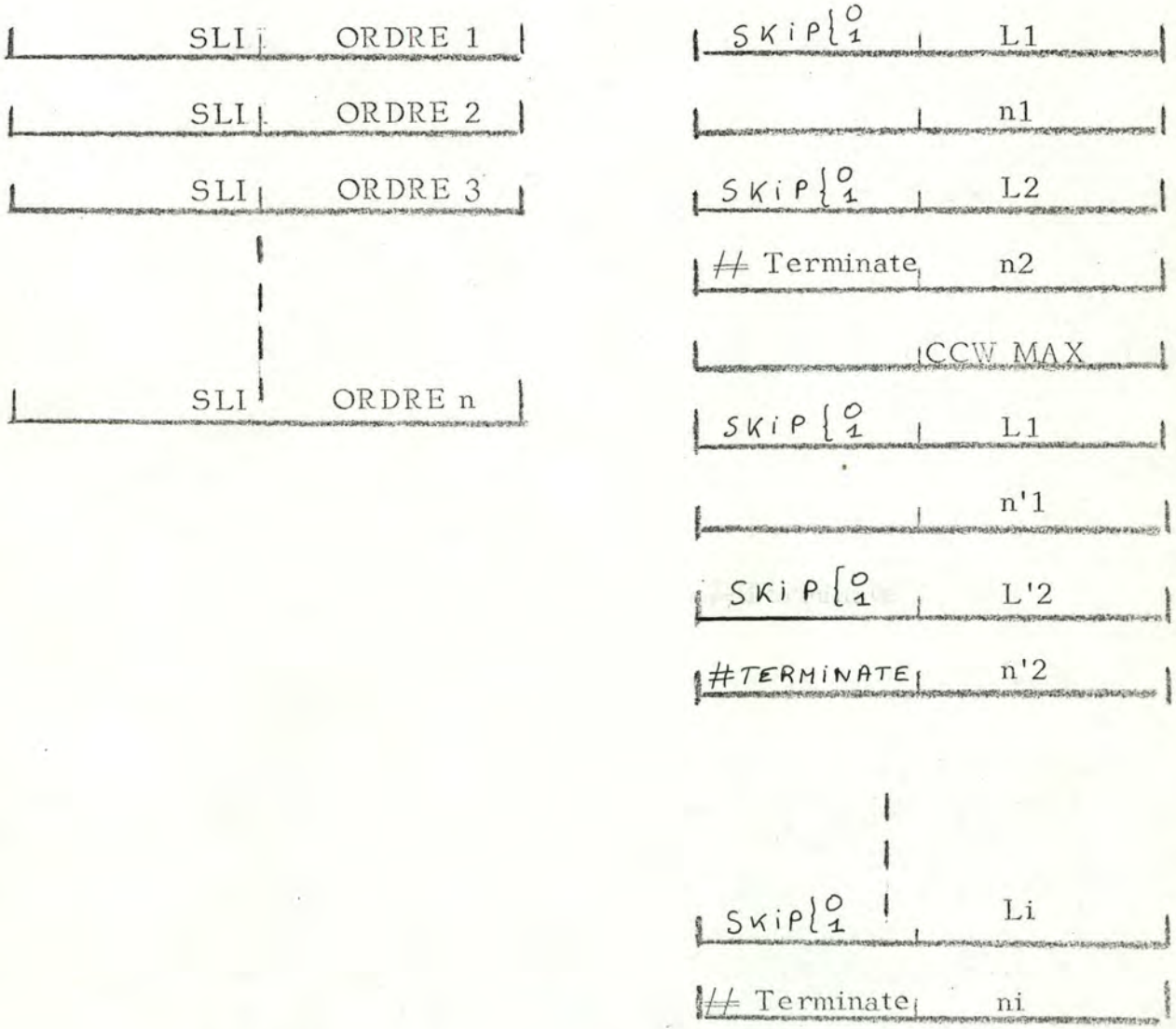


(1) par registre auxiliaire 1.
(2) par registre auxiliaire 2.
(3) par registre auxiliaire 3.
(4) compteurs de travail servant au calcul du maximum.

Fig. II. 13 (fin)



Mots Réservés



Zone Manoeuvre

Fig. II.14

APPEL - ANOMALIE - END.

Le seul paramètre à charger sera l'adresse du CCB.
Les autres paramètres, constantes du système, seront chargés à l'initialisation (Adresse Routine, priorité à exciter .....).

INSTRUCTION IØ

De même que dans le cas précédent, et dans l'optique choisie, les priorités à exciter sont des constantes du système.
Les seuls paramètres variables seront : l'adresse du CCW (pour transmission de l'ordre dans STIØ) et l'adresse du CCB pour le stockage des status et sense bytes.

Une vue récapitulative des paramètres à charger et du moment de leur chargement est fournie à la fig. II.15.

Il est à noter que la relative complexité de ces chargements résulte de la configuration "partagée" du canal IØ de l'EPRON : à un même périphérique (donc à un même sous-canal) correspondent plusieurs Requests, condition indispensable à un fonctionnement rentable de ce canal . Mais chacun de ces Requests étant associé à sa propre routine canal, il est nécessaire, au moment du chargement, de disperser les paramètres vers lesroutines adéquates.

	DTS	END DEVICE	END CANAL	I_{APPEL} I_{ANOM}	STIO	HIO	TDV	TIO
ADRESSE 1er CCW UTILISATEUR	EXCP	EXCP	-	-	EXCP	-	-	-
COMPTEUR CCW	EXCP	EXCP	-	-	-	-	-	-
POINTEUR CCW ACTUEL	EXCP	EXCP	-	-	-	-	-	-
BYTE COUNT	EXCP	-	-	-	-	-	-	-
ADRESSE IOAREA	EXCP	-	-	-	-	-	-	-
FLAGS SLI	-	EXCP	-	-	-	-	-	-
SKIP	EXCP	-	-	-	-	-	-	-
ORDRES	-	EXCP	-	-	-	INIT	-	-
ADRESSE CCB	EXCP	EXCP	EXCP	EXCP	EXCP	EXCP	EXCP	EXCP
# STATUS	-	-	-	-	INIT	INIT	-	INIT
# ORDRE	-	EXCP	-	-	INIT	INIT	-	-
# SENSE	-	-	-	-	-	-	INIT	-
# END	-	-	-	-	-	-	-	-
# TERMINATE	EXCP	-	-	-	-	-	-	-
Adresse zone sauvetage Registres	-	-	INIT	INIT	-	-	-	-
Adresse (de type) routine à déclencher	-	-	INIT	INIT	-	-	-	-

N.B. : EXCP : paramètre fourni à la routine canal lors de l'exécution de EXCP.
INIT : chargé à l'initialisation du système.

Fig. II.15

2.3. SUPERVISEUR IØ.

Cet ensemble constitue un module du système, au même titre que l'EXECUTIVE qui s'intéresse aux paramètres de fonctionnement d'un programme, ou que le MONITEUR qui assure l'enchaînement des Jobs dans une session Moniteur.

Le SUPERVISEUR IØ (ou des entrées/sorties) regroupe un ensemble de routines propres à la gestion des IØ, c'est-à-dire leur lancement, les contrôles de fin, et les éventuels traitements d'erreurs.

Interface entre l'utilisateur et le canal IØ, le superviseur peut être appelé par l'un et par l'autre. L'utilisateur formulera un "appel au superviseur", et via la table du superviseur, obtiendra le déroulement de la séquence désirée par exécution d'un DEBRANCHEMENT.

Les appels du canal se traduiront par la génération d'interruptions, interruptions dont le traitement assurera le déroulement de la routine superviseur souhaitée.

Nous ne reviendrons pas sur le détail de ces prises en charge, elles ont fait l'objet du chapitre précédent.

Nous allons maintenant examiner en quoi consistent ces "routines souhaitées" et la table du superviseur, base d'accès à ces routines.

2.3.1. TABLE DU SUPERVISEUR.

Cette table comprendra, d'une part les adresses des routines et d'autre part l'adresse des paramètres à utiliser pour l'appel à traiter.

ADRESSE DES ROUTINES : Adresses des points d'entrée aux différents segments du superviseur, en fonction du type de routines.

ADRESSE DES PARAMETRES (fournie à l'appel). Dans le cas de l'interruption de programme, celle-ci est fournie via les mots Réservés paramètres, et sauvée par RA, comme décrit à la fig.II.16.

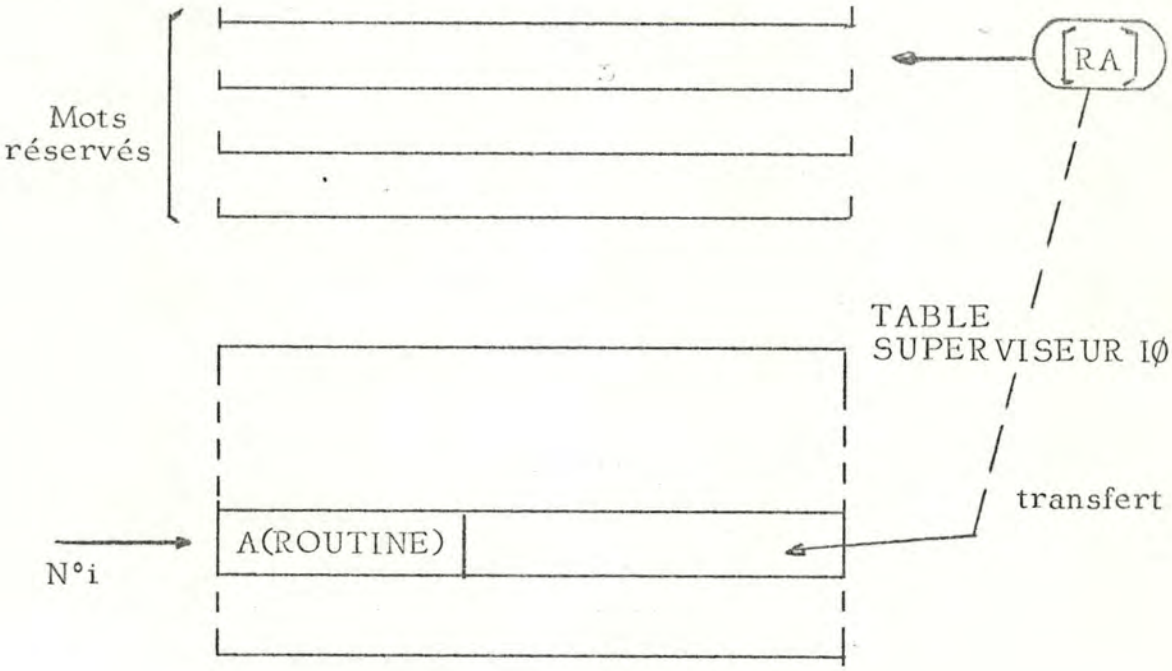


Fig. II.16

Dans le cas du débranchement, l'adresse des paramètres est fournie comme 2e adresse du SVC.
Pour que cette adresse soit à la disposition du superviseur, il faudra donc, avant toute autre manipulation, sauver le contenu de CP dans la table du superviseur.

La table du superviseur aura la forme suivante :

TYPE	A (ROUTINE)	A (PARAMETRE)
1	constantes	chargées
2	pour configuration	à chaque
3	donnée	appel
4	du système	
.		
.		
.		
n		

2.3.2. ROUTINES DU SUPERVISEUR.

Les routines retenues pour le superviseur IØ seront les suivantes :

EXCP	initialisation d'une opération IØ.
WAIT	synchronisation sur une IØ.
RECOVERY	(par périphérique) Récupération des erreurs externes.
TYPE	écriture message à la console.
TYPAN	écriture message console, avec réponse.
END	contrôle de fin d'IØ.
TIMER	déclenchement du timer.

2.3.2.1. EXCP (Execute channel program).

Cette routine assure le lancement de l'opération IØ "globale".
Nous distinguerons trois phases :

1. TEST.

   . Il est nécessaire de s'assurer que le périphérique existe dans la configuration existante, et dans ce cas, tester s'il est disponible pour l'IØ à lancer.

2. CHARGEMENT du programme canal.

   Assure le chargement des différentes routines du canal, ayant trait à ce périphérique et donc susceptibles d'intervenir au cours de cette IØ.

3. LANCEMENT du programme canal.

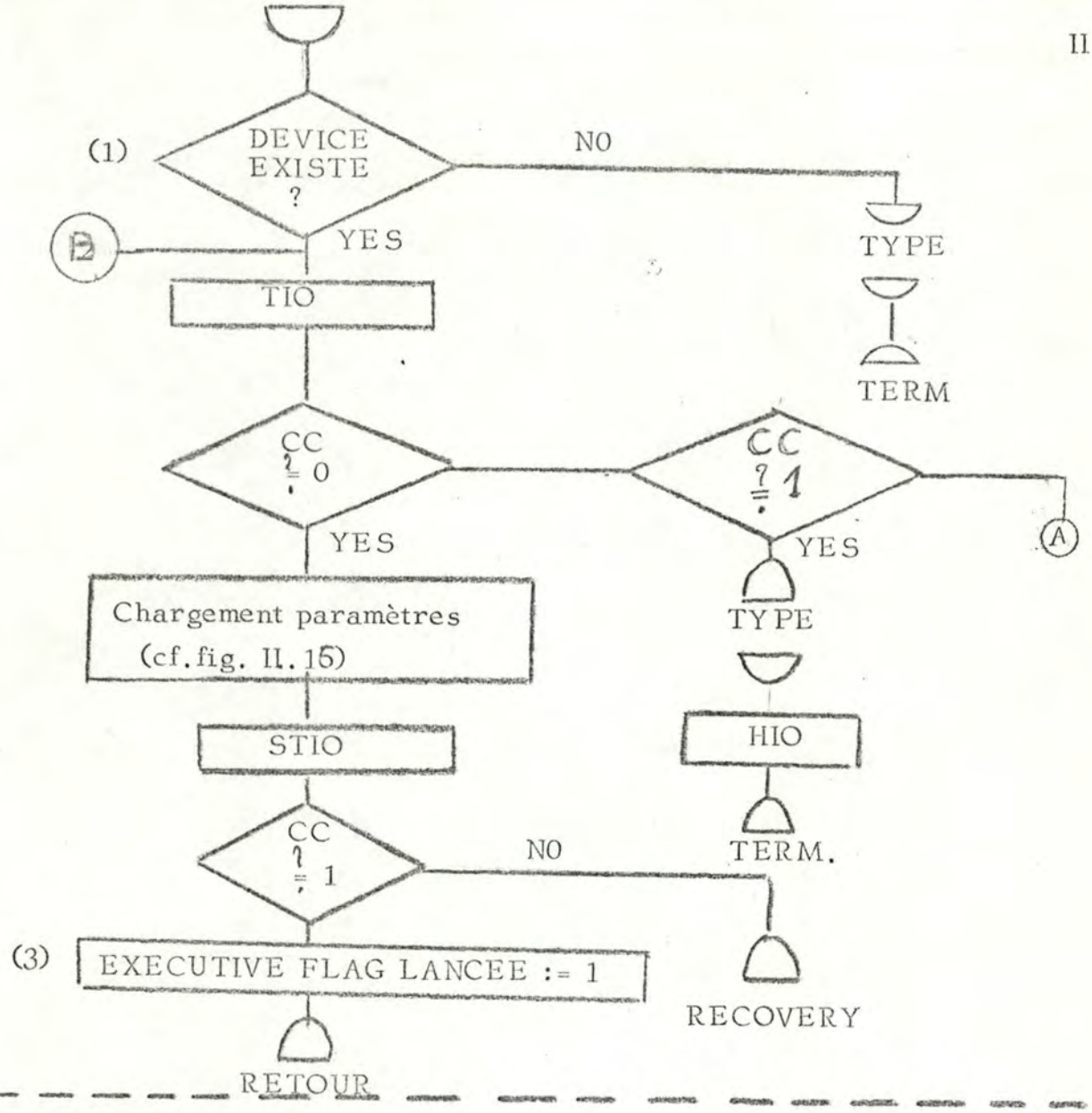
   Constitue le démarrage de l'opération d'IØ, c'est-à-dire de l'exécution d'un ordre ou d'une chaîne d'ordres.

La fonction est décrite à la fig. II.17.

(1) Par consultation de la "DEVICE LIST", il est possible de déterminer si le périphérique existe effectivement sur la configuration envisagée.
S'il n'existe pas, l'erreur est signalée à la console, et le programme abandonné.

(2) Le cas de périphérique occupé lors du lancement d'une IØ révèle une erreur soit au niveau du périphérique (s'il ne parvient pas à détecter une fin de bloc), soit au niveau du programme (si l'utilisateur n'a pas prévu de synchronisation entre le lancement de deux IØ sur le même périphérique).
Nous avons opté pour la solution la plus radicale : message d'erreur, reset du périphérique et abandon du programme. Des solutions plus élaborées auraient pu être envisagées, par ex. : la gestion d'une file d'attente (pour parer à l'absence de WAIT).
Il est à noter que, dans le cadre de multiprogrammation et de périphérique partagé, c'est vers un solution de ce genre que nous devrions nous tourner.

(3) L'exécutive Flag est traité en détails dans la partie relative à la communication superviseur-programme.


(1)
DEVICE EXISTE ?
NO
TYPE
TERM
YES
B
TIO
CC ? = 0
CC ? = 1
YES
YES
A
Chargement paramètres (cf. fig. II.15)
TYPE
HIO
STIO
TERM.
CC ? = 1
NO
(3)
EXECUTIVE FLAG LANCEE := 1
RECOVERY
RETOUR

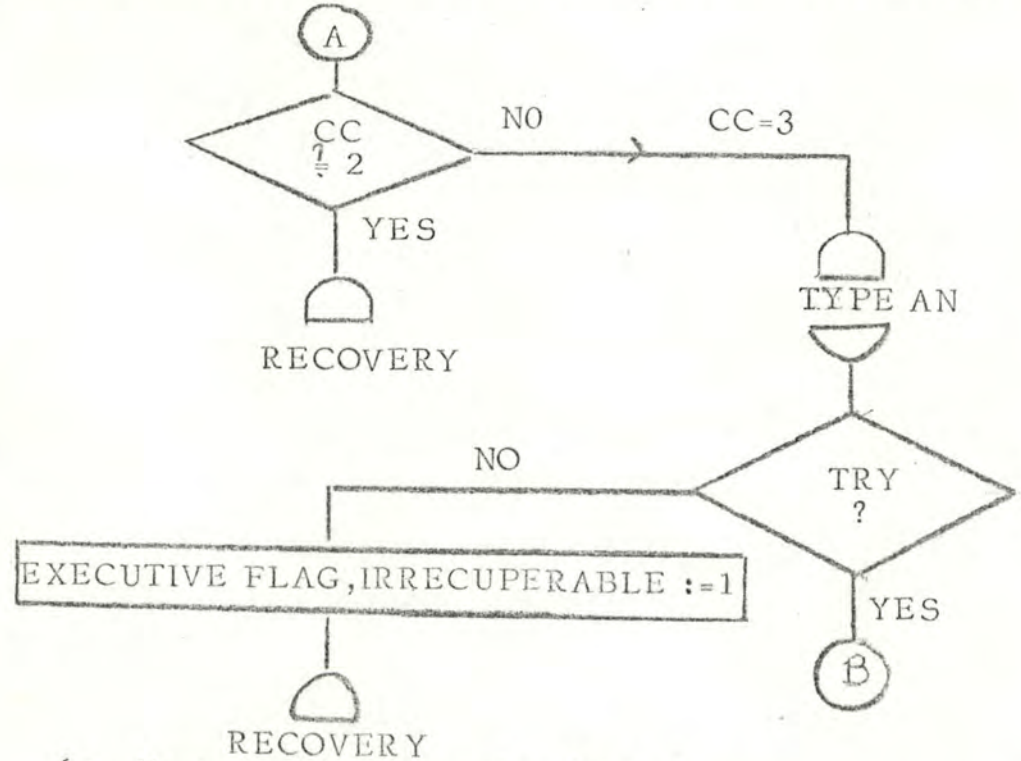
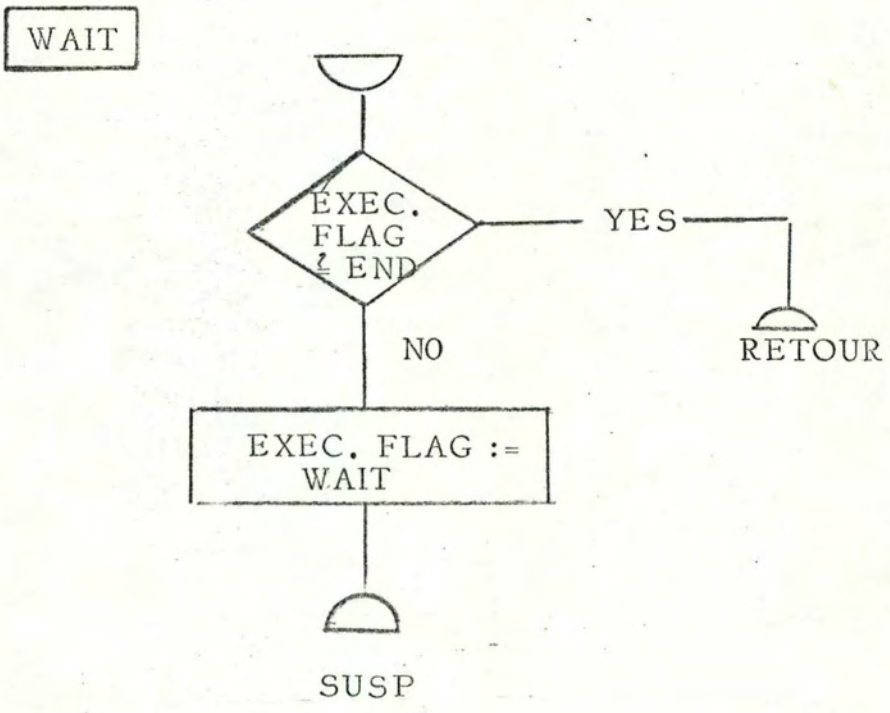

A
CC ? = 2
NO
CC=3
YES
TYPE AN
RECOVERY
NO
TRY ?
EXECUTIVE FLAG, IRRECUPERABLE := 1
YES
B
RECOVERY
(pt. d'entrée : Erreur irrécupérable)

Fig. II.17

2.3.2.2. WAIT.

L'utilisateur fait appel à cette routine lorsqu'il veut se resynchroniser par rapport à une IØ qu'il a fait lancer précédemment. Dans ce cas, il sera mis en état bloqué aussi longtemps que l'IØ désignée n'est pas terminée.



2.3.2.3. RECOVERY.

Du fait des particularités de chaque périphérique, nous sommes amenés à considérer une routine de récupération par périphérique : ce que d'aucuns appellent les "DRIVERS" des périphériques.

Il nous a cependant semblé bon de donner une vue d'ensemble des erreurs rencontrées, ainsi que de la logique de leur traitement. C'est pourquoi nous avons essayé de définir, de façon générale, des classes d'erreurs, leurs répercussions sur le fonctionnement des IØ et les conséquences quant à la fonction réalisée d'une part, et l'état du périphérique d'autre part.
Les décisions seront prises à partir de ces deux résultats, en tenant compte aussi des options de l'utilisateur.

Ces généralités peuvent alors s'appliquer pour chacun des périphériques utilisés : il suffit de spécifier dans quelles classes se retrouveront les erreurs propres au périphérique traité, et suivre la filière résumée ci-dessus.
C'est ce que nous proposons pour le lecteur de cartes et le dérouleur de bande.

1. DEMARCHE GENERALE.

A. Classes d'erreurs.

1. ERREURS DE PROGRAMME.

Provoquées par la non reconnaissance par le périphérique d'ordres ou de data.

Par ex. : ordre inconnu pour le périphérique, caractère non admis.

2. ERREURS DE MECANIQUE DU PERIPHERIQUE.

a. Statiques : par ex. : pas de cartes dans le perfo., indicateur start non positionné.

b. de fonctionnement : dues à une manoeuvre incorrecte. par ex. : bourrage de cartes, tête non positionnée sur piste.

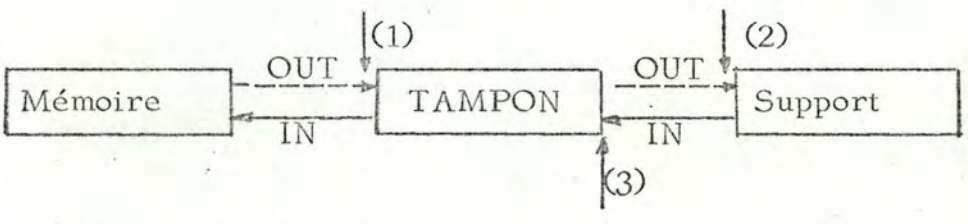
3. CONDITIONS EXCEPTIONNELLES.

Selon l'application, elles seront attendues ou fautives. Elles seront en général attendues comme résultat de fonctions de service (ex. BT en fin de Rebobinage).

4. ERREURS DE TRANSMISSION.

Nous y distinguerons trois parties : les erreurs de parité, et les erreurs d'overrun.

Les erreurs PARITE peuvent être longitudinales ou transversales (c'est-à-dire porter sur un bloc ou sur un caractère seulement). D'autre part, elles peuvent surgir soit entre le support et le tampon du périphérique, soit entre ce tampon et la mémoire.



Nous retiendrons trois cas.

TRANSMISSION : erreur de parité à l'entrée du tampon, en provenance de la Mémoire (1).

RELECTURE : erreur de parité des données stockées sur le support (2). Cette erreur est détectée par le poste de lecture de contrôle, donc ne s'applique pas à tous les périphériques (ex. : il existe pourdérouleur de bande).

LECTURE : erreur de parité lors du transfert du support vers le tampon (3).

Il est à noter que dans les cas (1) et (3), les erreurs de parité longitudinale ne seront détectées que pour les périphériques munis d'un tampon dont la longueur est celle du bloc (par. ex. : pour imprimante bufferisée).

Les erreurs d'OVERRUN sont provoquées par le non-respect des cadences propres aux périphériques:

L'unité de contrôle reçoit du support l'impulsion destinée à générer le Request, alors que le strobe de réponse au Request précédent ne lui a pas encore été envoyé : il y a donc surcharge du canal IØ.

B. Répercussion sur les IØ.

Nous procéderons en 3 étapes, : le résultat final permettant de prendre une décision en ce qui concerne l'action à entreprendre.

1. Survenance des classes d'erreurs : définition de types
   cf. fig. II.18.
2. Conséquence de ces types sur le déroulement des IØ
   cf. fig. II.19.
3. Résultat de l'IØ en ce qui concerne les données, et en ce qui concerne l'état du périphérique.
   cf. fig.II.20.

C. Action entamée.

4 critères vont définir l'action à initialiser :
les options de l'utilisateur (fournies dans le CCB), la valeur des données fournies par l'IØ entachée d'erreur, l'état du périphérique, et l'origine de l'erreur.

Origine de l'erreur.

L'erreur peut être occasionnée :

- par le PROGRAMME : dans ce cas, les tentatives de correction par répétition sont inutiles.

Survenance de classes d'erreurs.

SURV. / CLASSE	DEBUT	EN COURS	FIN
PROGR	1	2	
MEC STAT	3	4	5
MEC MALFCT		6	7
PARITE TRANSM	8	9	9'
PARITE RELECT		10	10'
PARITE READ		11	11'
OVERRUN		12	
COND EXCEPT		13	

csq sur le déroulement de IØ. Fig.II.18

	NON INIT.	ARRETEE	ACHEVEE
1 PROG DEBUT	A		
2 PROG COURS			B
3 M.STAT.DEBUT	C		
4 M.STAT.COURS		D	
5 M.STAT.FIN			E
6 M.MALFCT COURS		F	F'
7 M.MALFCT FIN			G
8 PAR.TRANSM.DEB	H		
9 PAR.TRANSM. COURS			I
10 PAR RELECT.			J
11 PAR. READ			K
12 OVERRUN		L	
13 COND.EXCEPT.		M	

Fig.II.19

ORIGINE/VALEUR des DONNEES/ETAT PERIPHERIQUE.

		ORIGINE		VALEUR DATA			PERIPH.			
		PROG.	PERIPH.	INEXIST.	INVAL.	VALIDES	OK	RESET	RESET/DEF	DEF
A	PROGR. DEBUT	X		X			X			
B	PROGR. COURS	X			X		X			
C	M.STAT DEB.		X	X				X		
D	M.STAT COURS		X	X(1)	X(1)			X		
E	M.STAT FIN		X			X		X		
F	M.MALFCT COURS		X		X				X	
G	M.MALFCT FIN		X			X			X	
H	PARITE TRANSM.DEBUT		X	X			X(3)			X(3)
I	PARITE TRANSM.COURS		X		X		X(3)			X(3)
J	PARITE RELECTURE		X		X		X(3)			X(3)
K	PARITE READ		X		X		X(3)			X(3)
L	OVERRUN		X		X		X(3)			X(3)
M	COND. EXCEPT.		X			X(2)	X			

(1) Les données peuvent être soit invalides, soit partiellement valides.
Il est plus prudent de considérer que tout le bloc est entaché d'erreur, plutôt que de tenter de récupérer cette partie.

(2) Certaines conditions exceptionnelles, incompatibles avec le code en vigueur, provoquent erreur.

(3) Ces erreurs sont initialement considérées ..récupérables.
Mais après un certain nombre de répétitions infructueuses, l'erreur est considérée comme irrécupérable, et le hardware défectueux.

Fig. II.20

On pourrait envisager de laisser à l'opérateur le soin de corriger l'ordre et de relancer l'IØ. Quant à nous, nous avons pris l'option d'abandonner l'IØ et le programme.

- par le PERIPHERIQUE et son unité de contrôle.
  Dans ce cas, il est possible d'envisager une procédure de correction par intervention manuelle et/ou répétition.

Option utilisateur.

Pour certains périphériques, il existe des options particulières, reconnues seulement par le Driver approprié, par ex. la demande de reperforer une carte en cas de détection d'erreur de parité.
Nous retiendrons ici les possibilités générales suivantes : prise en charge de toutes les erreurs, prise en charge des erreurs irrécupérables, demande d'intervention de l'opérateur en cas d'erreur irrécupérable.

1. PRISE EN CHARGE DE TOUTES LES ERREURS.

   Dans ce cas, le superviseur rend la main à la routine appropriée de l'utilisateur. Celle-ci procédera à l'analyse du sense byte, et prendra les décisions en fonction de l'application, par exemple, considérer ce genre d'erreur comme négligeable.

2. PRISE EN CHARGE DES ERREURS IRRECUPERABLES.

   Après avoir vainement tenté une correction, le superviseur rend la main à l'utilisateur, à la routine indiquée. Celle-ci peut, par exemple, demander la modification d'assignation du périphérique, ou sauter le traitement du bloc entaché d'erreur.

3. DEMANDE D'INTERVENTION DE L'OPERATEUR.

   C'est l'opérateur qui prendra la décision de modifier les assignations, ou d'abandonner le programme.

Etat du périphérique.

En fonction des états définis à la fig. II 20 , la correction comprendra une intervention sur le périphérique.

1. Hardware défectueux.

   Aucune action sur le périphérique, et l'erreur est marquée "irrécupérable" par le superviseur.

2. Hardware RESET.

   Pour entamer la correction, il est nécessaire que l'opérateur agisse sur le périphérique.
   Cette action est spécifiée par un message console.

3. Hardware'RESET ou défectueux."

   La décision est du ressort de l'opérateur.
   L'erreur est donc signalée par un message console, celui-ci attendant une réponse.
   La suite du traitement dépend de cette réponse.

4. Hardware OK.

   La correction ne nécessite aucune intervention sur le Hardware : les éventuels repositionnements se feront sous le contrôle du superviseur.

Etat des données.

1. Inexistantes.

   L'IØ n'a pas eu lieu. Il suffit de la relancer, sans modification des paramètres du canal.

2. Invalides.

   Le transfert des données s'est effectué de façon incorrecte. Avant de relancer l'IØ, il faut assurer d'une part, le repositionnement du support, et d'autre part la réinitialisation des paramètres du canal.

   Le positionnement du support se fera de façonsdifférentes, suivant les périphériques : le lecteur de carte nécessitera une intervention manuelle, le dérouleurde bande utilisera une séquence de CCW destinés au saut de blocs, le disque utilisera le SEEK, qui était prévu dans sa séquence.

3. Valides.

   La correction ne comprend pas de reprise du transfert des données. Deux cas peuvent se présenter : suivant que l'on détecte ou non une condition exceptionnelle.

   La condition exceptionnelle sera mentionnée à l'utilisateur par positionnement du bit adéquat dans l'EXECUTIVE FLAG du CCB (ex. : détection de début de bande à l'exécution d'un ordre REWIND.

Il est à noter que certaines conditions exceptionnelles seront reconnues comme fautives et occasionneront un traitement d'erreur.

Par ex. : détection de ET en cours de lecture ou d'écriture sur bande magnétique.

(1), (12), (13) Ces conditions seront testées par consultation du "user Flag" fourni par l'utilisateur dans le CCB. Dans les cas (1) et (12), l'adresse à laquelle il faut se brancher est également fournie dans le CCB.

(2), (3), (4), (5), (6) Ces valeurs seront définies en fonction de la valeur affichée dans le SENSE BYTE, et varieront en fonction du périphérique.

(7) Les erreurs de transmission, en plus du traitement général, nécessitent la mise à jour d'un compteur de répétitions. Une fois la valeur maximale atteinte, l'erreur est considérée comme irrécupérable et due au hardware.

(8), (9) Nous dissocions les interventions manuelles (8) des séquences où l'opérateur doit prendre une décision en ce qui concerne la reprise de l'opération.

(10), (11) Ces conditions sont déterminées non seulement par la valeur affichée dans le SENSE BYTE, mais aussi par le moment de survenance de l'erreur (consultation de l'Executive Flag).

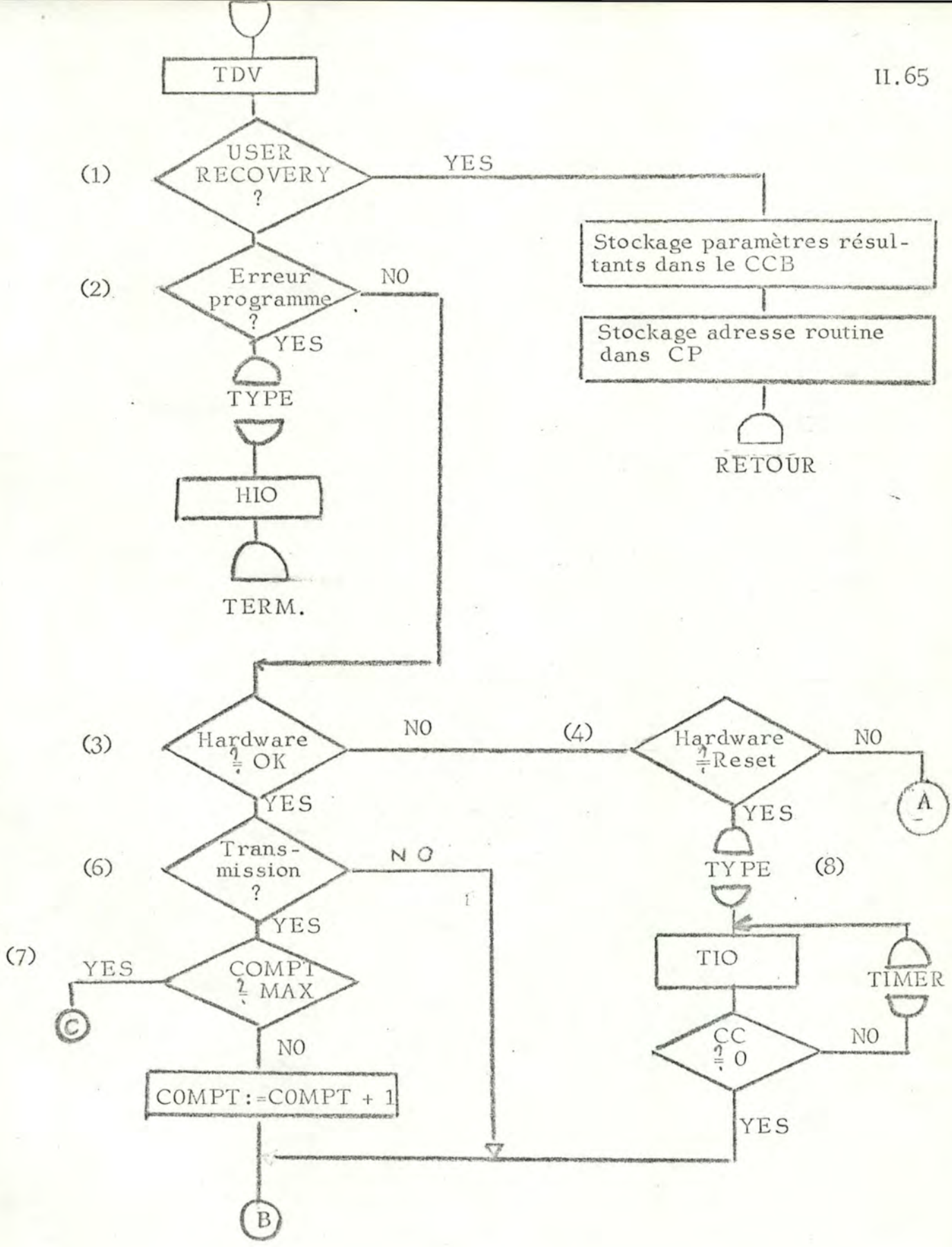

TDV
(1)
USER RECOVERY ?
YES
Stockage paramètres résultants dans le CCB
Stockage adresse routine dans CP
RETOUR
(2)
Erreur programme ?
NO
YES
TYPE
HIO
TERM.
(3)
Hardware ≠ OK
NO
(4)
Hardware ≠ Reset
NO
A
YES
TYPE
(8)
TIO
TIMER
CC ≠ 0
NO
YES
(6)
Transmission ?
NO
YES
(7)
YES
COMPT ≥ MAX
C
NO
COMPT := COMPT + 1
B

Fig. II.21 (1)

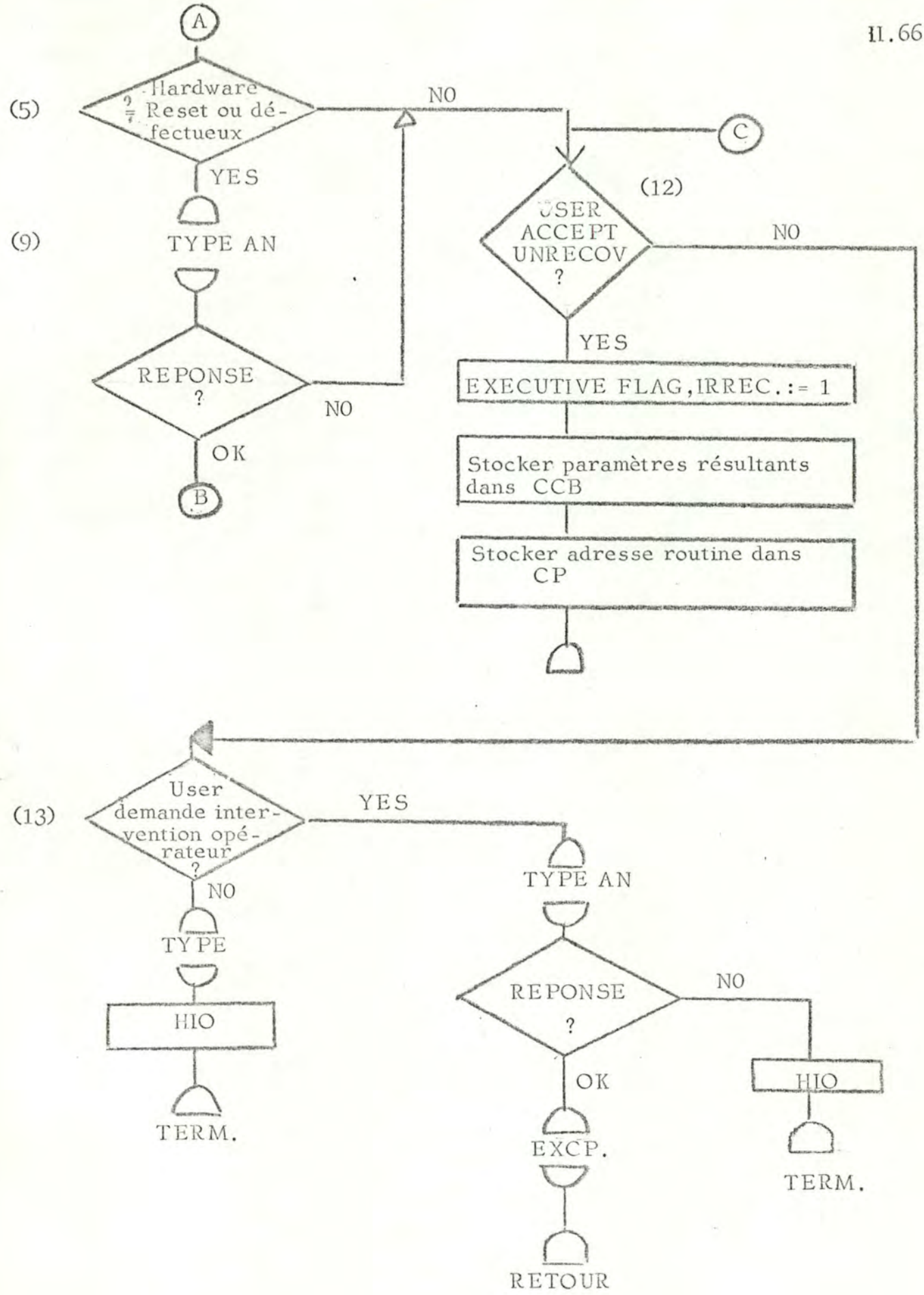

A
(5)
Hardware ? Reset ou défectueux
NO
C
YES
(12)
USER ACCEPT UNRECOV ?
NO
(9)
TYPE AN
REPONSE ?
NO
YES
EXECUTIVE FLAG,IRREC.:= 1
OK
B
Stocker paramètres résultants dans CCB
Stocker adresse routine dans CP
(13)
User demande intervention opérateur ?
YES
NO
TYPE AN
TYPE
REPONSE ?
NO
HIO
OK
HIO
TERM.
EXCP.
TERM.
RETOUR

Fig. II.21 (2)

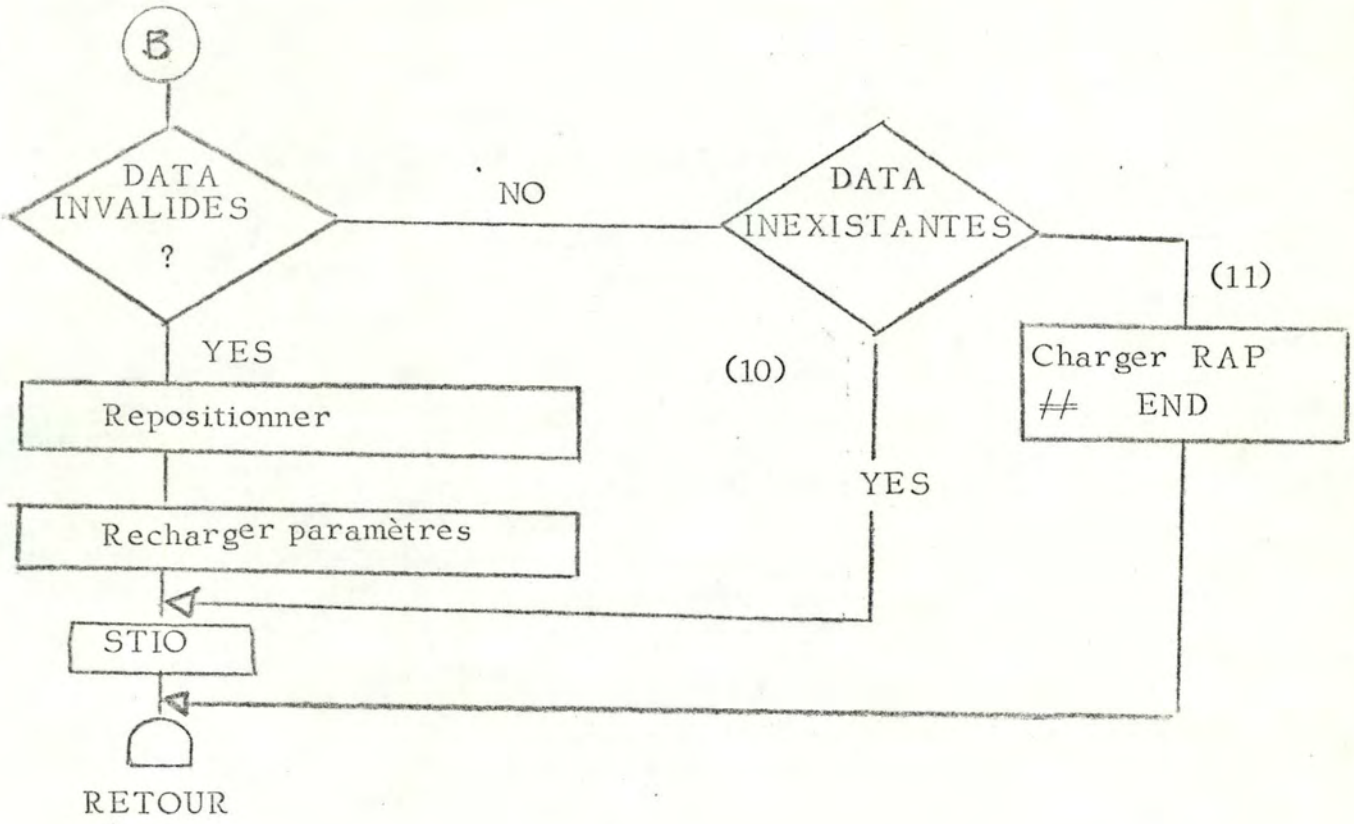

B
DATA INVALIDES ?
NO
DATA INEXISTANTES
(11)
YES
(10)
Charger RAP
END
Repositionner
YES
Recharger paramètres
STIO
RETOUR

Fig. II.21 (3)

2. APPLICATIONS

2.1. LECTEUR DE CARTES.

Considérons un lecteur de carte dont le sense byte a la configuration suivante :

- ORDRE illégal.
- HOLD (STOP),
- code perforation invalide,
- sélection trop tardive case réceptrice,
- OVERRUN,
- READ (parity),
- Bourrage / Alimentation double.

Nous pouvons, à partir de là, construire le tableau ci-dessous et l'organigramme résultant (cf. fig. II.22).

NATURE	TYPE	ORIG.	VAL. DATA	PERIPH.
ORDRE illégal	PROGR.DEBUT	PROGR.	INEXIST	OK
HOLD	M.STAT.DEBUT	PERIPH.	INEXIST	RESET
Code perfo.illégal.	PROGR.COURS	PROGR.	INVAL.	OK
Sélection trop tardive	M. STAT.FIN	PERIPH.	VALIDES	RESET/DEF
OVERRUN	OVERRUN	PERIPH.	INVAL.	OK
READ (parity)	PARITE COURS	PERIPH.	INVAL.	OK
Bourrage	M.MALFCT	PERIPH.	INVAL.	RESET/
Alimentation double	COURS		(douteux)	DEF

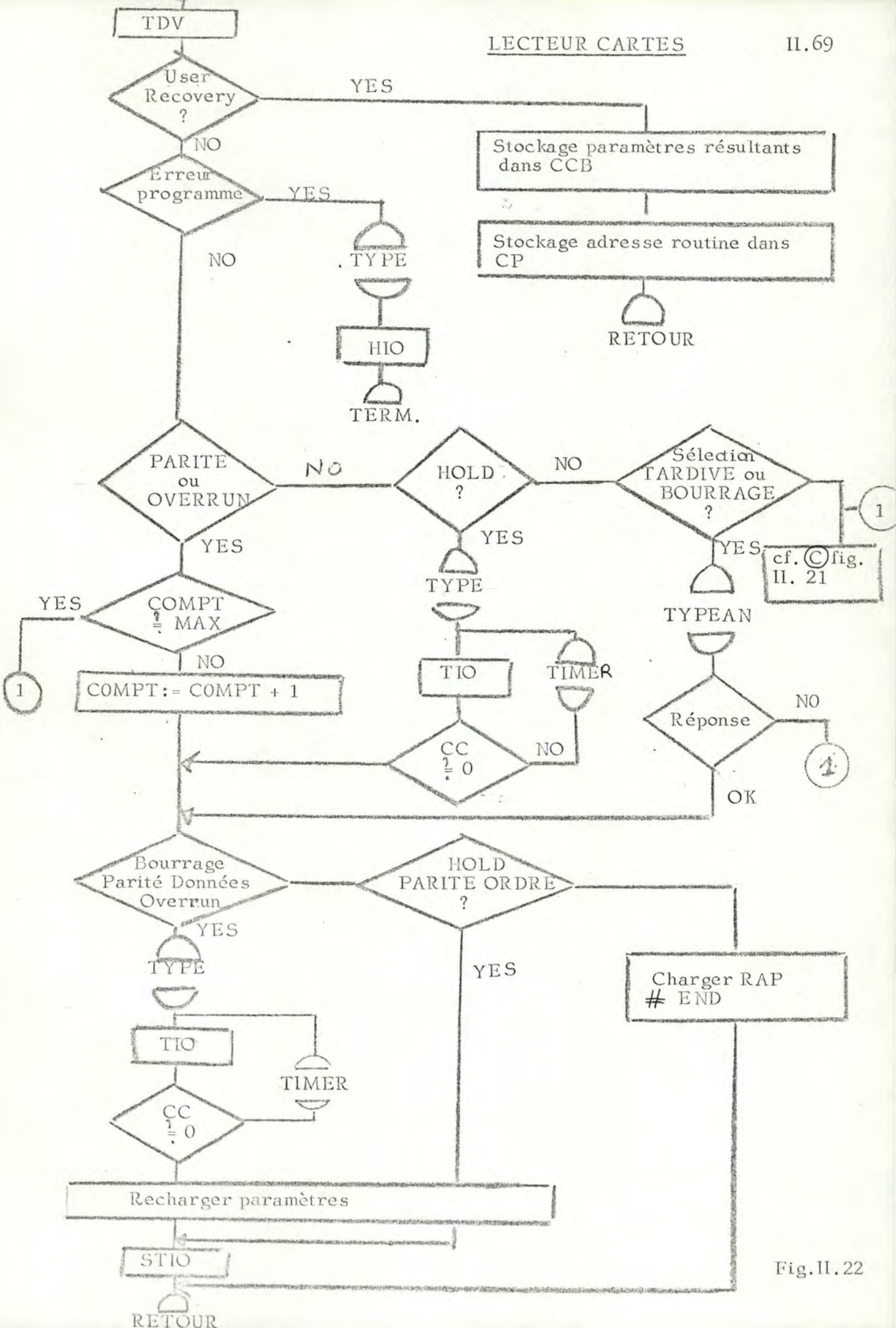

TDV
User Recovery ?
YES
NO
Stockage paramètres résultants dans CCB
Erreur programme
YES
NO
. TYPE
HIO
TERM.
Stockage adresse routine dans CP
RETOUR
PARITE ou OVERRUN
NO
HOLD ?
NO
Sélection TARDIVE ou BOURRAGE ?
1
YES
YES
YES
cf. © fig. II. 21
TYPE
TYPEAN
YES
COMPT ≠ MAX
NO
1
COMPT := COMPT + 1
TIO
TIMER
CC ≠ 0
NO
Réponse
NO
1
OK
Bourrage Parité Données Overrun
HOLD PARITE ORDRE ?
YES
TYPE
YES
Charger RAP # END
TIO
TIMER
CC ≠ 0
Recharger paramètres
STIO
RETOUR

Fig. II. 22

2.2. DEROULEUR BANDE MAGNETIQUE.

Considérons un dérouleur de bande pouvu du sense byte décrit ci-dessous.
(Il est à noter quecette configuration est citée à titre d'exemple, et à adapter suivant le modèle précis du périphérique employé).

	Lect.	Ecrit.
1. Ordre invalide	x	x
2. PARITE LECTURE	x	
3. PARITE TRANSMISSION		x
4. PARITE RELECTURE		x
5. Détection TM	x	
6. Détection BT/ET	x	x
7. OVERRUN	x	x
8. Lecture Bloc trop court	x	
9. Anneau écriture manquant		x
10 STOP ou LOCAL	x	x

N.B. : Ces combinaisons peuvent être regroupées sur un seul byte, un bit prenant une signification ≠ suivant que le dérouleur travaille en écriture ou en lecture.

Ces différentes erreurs seront regroupées en classes (cf. fig. II.23), ainsi que nous l'avons défini dans notre traitement général et traitées comme indiqué à la fig. II.24.

	CLASSE	SURVENANCE DEBUT	SURVENANCE COURS	CATEG.	VALIDITE DATA	PERIPH.	ORIGINE
ORDRE INVALIDE	PROG.	x		A	INEXIST.	OK	PROG.
PARITE LECTURE	TRANSMISSION		x	I	INVALIDES	OK	PERIPH.
PARITE TRANSMISSION	TRANSMISSION	x	x	H - I	INEXIST/INVAL	OK	PERIPH.
PARITE RELECTURE	TRANSMISSION		x	I	INVALIDES	OK	PERIPH.
DETECTION TM	COND.EXCEPT.		x	M	VALIDES	OK	PERIPH.
DETECTION ET/BT	COND.EXCEPT.		x	M	VAL./INVAL.	OK/OPER	PERIPH:
OVERRUN	TRANSMISSION		x	L	INVALIDES	OK	PERIPH.
BLOC trop court	TRANSMISSION		x		INVALIDES	OK	PERIPH.
ANNEAU MANQUANT	MEC. STAT.	x		C	INEXIST.	RESET	PERIPH.
STOP / LOCAL	MECT. STAT.	x		C	INEXIST.	RESET	PERIPH.

Fig. II.23

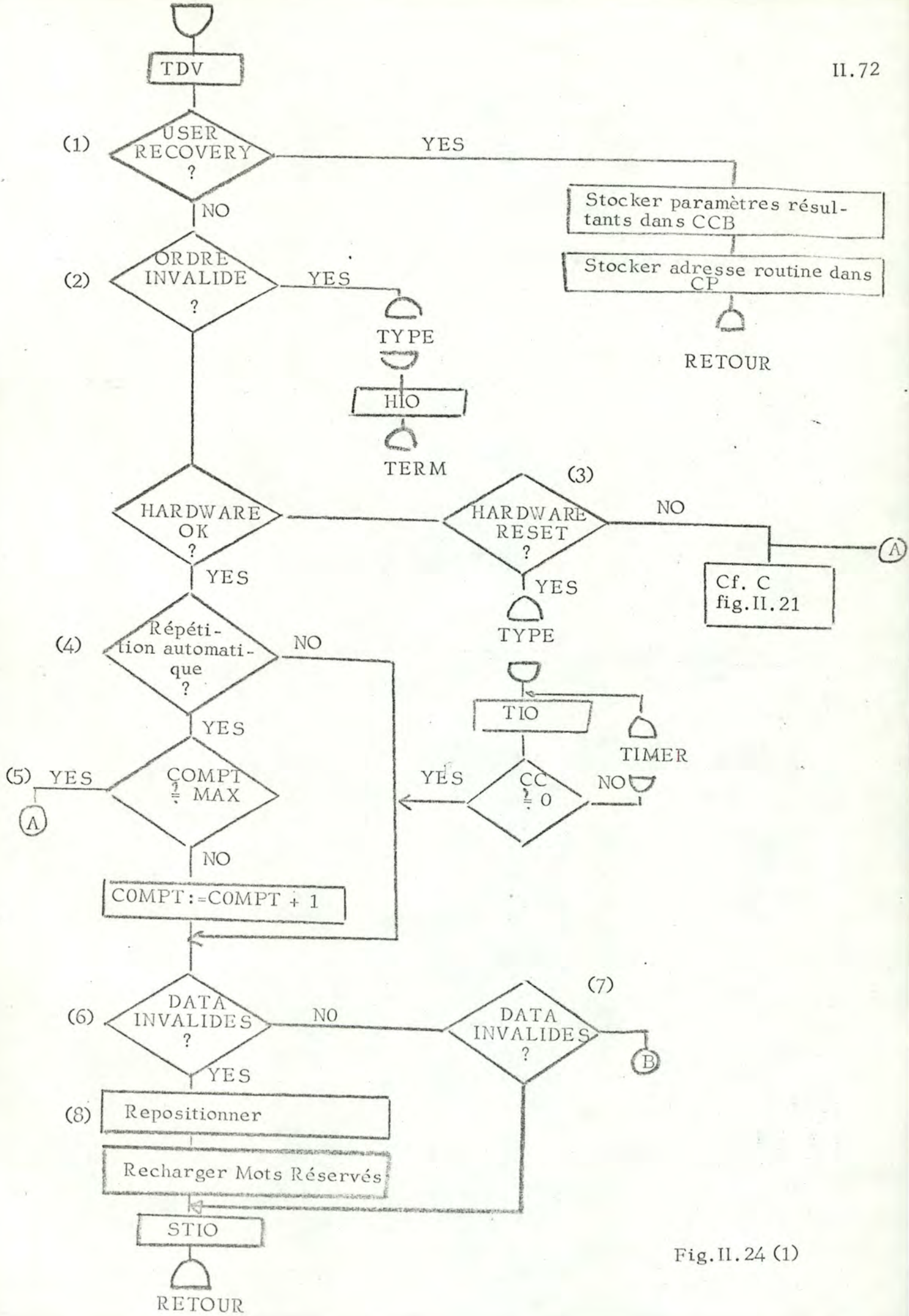

TDV
(1)
USER RECOVERY ?
YES
NO
Stocker paramètres résultants dans CCB
Stocker adresse routine dans CP
RETOUR
(2)
ORDRE INVALIDE ?
YES
TYPE
HIO
TERM
HARDWARE OK ?
(3)
HARDWARE RESET ?
NO
A
Cf. C fig.II.21
YES
TYPE
YES
(4)
Répétition automatique ?
NO
YES
TIO
TIMER
(5)
YES
COMPT ≟ MAX
YES
CC ≟ 0
NO
A
NO
COMPT:=COMPT + 1
(6)
DATA INVALIDES ?
NO
(7)
DATA INVALIDES ?
B
YES
(8)
Repositionner
Recharger Mots Réservés
STIO
RETOUR

Fig.II.24 (1)

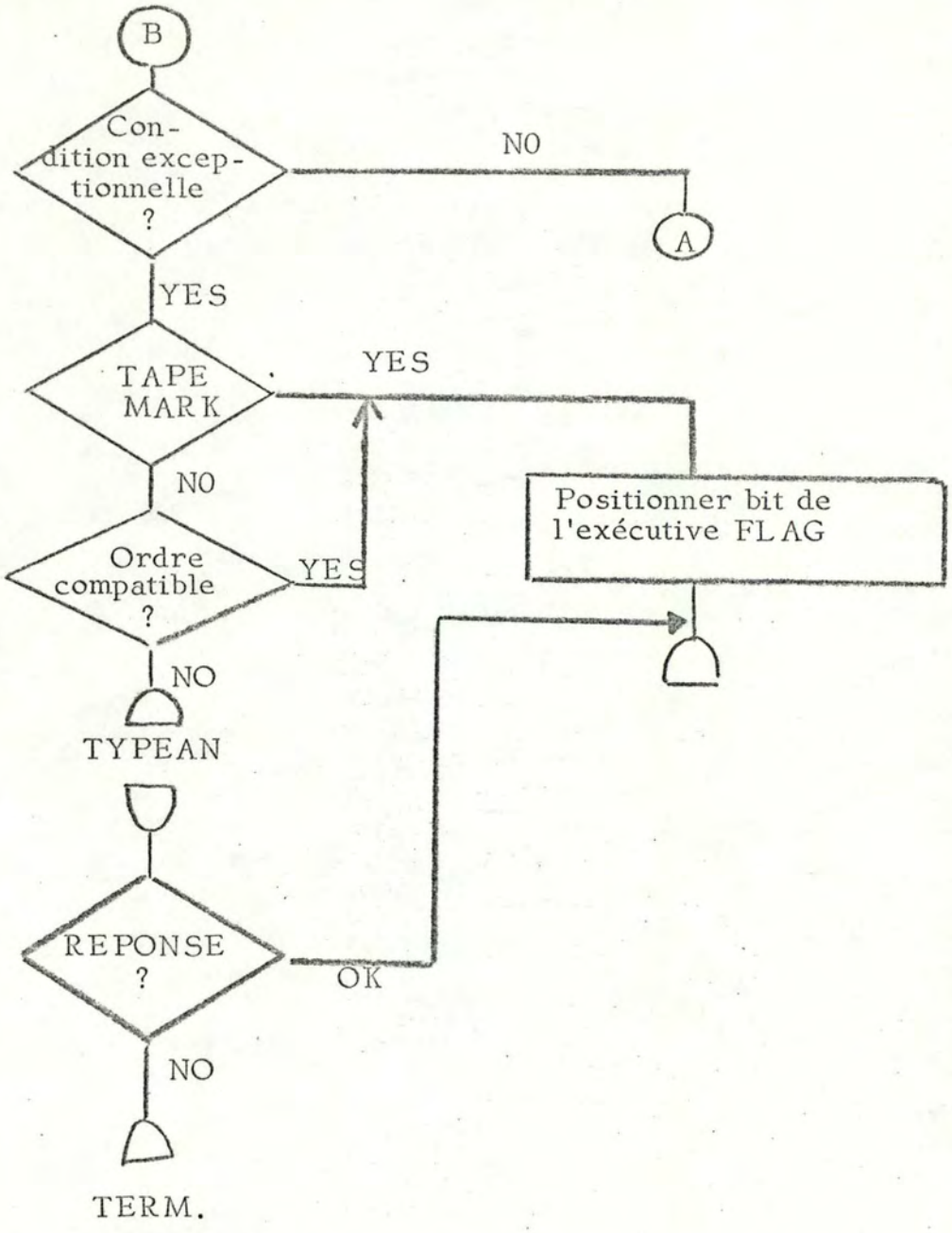

B
Con-
dition excep-
tionnelle
?
NO
A
YES
TAPE
MARK
YES
NO
Positionner bit de
l'exécutive FLAG
Ordre
compatible
?
YES
NO
TYPEAN
REPONSE
?
OK
NO
TERM.

Fig. II.24 (2)

(3) Cette condition sera réalisée pour les configurations suivantes du SENSE BYTE :

- Anneau écriture manquant.
- HOLD
- LOCAL.

(4) Cette condition sera vérifiée pour :

- PARITE (READ/RELECTURE/TRANSMISSION)
- OVERRUN
- BLOC TROP COURT.

(6) Nous le trouverons pour :

- Anneau écriture
- HOLD
- LOCAL

pour lesquels l'ordre n'a pas été transmis
- PARITE alors que l'IØ n'a pas été lancée
(donc parité lors du transfert de l'ordre).

(7) Cette erreur existera pour toutes les erreurs de transmission (cf. (4).

(8) Le repositionnement consiste en l'introduction dans la chaîne du canal de CCW de saut AV/AR.
Il faudra également opérer une remise en état des paramètres du canal.

2.3.2.4. TYPE.

Cette routine aura pour but de sortir à la console un message pour lequel on n'attend pas de réponse.
La console fonctionne donc uniquement comme imprimante.
La fonction TYPE assurera :

1. le garnissage des mots réservés associés à la console (DTS, END) pour peu que la console soit disponible.
2. le lancement de l'écriture (avec mise préalable de la console en mode écriture).
3. l'attente de fin normale de l'opération.

Ce fonctionnement est décrit à la fig. II.25.

Les risques de conflit d'accès entre le CPU (utilisateur ou système) qui veut lancer la fonction TYPE, et l'opérateur qui veut introduire un ordre à la console (cf. chap. III), sont supprimés par l'attribution de priorités différentes aux deux types d'appel, et par l'attribution du même niveau d'interruptibilité.

Lorsque les deux demandes existent, la plus prioritaire (I_{APPEL}) sera traitée.
Mais lorsque l'une des deux est en cours de traitement, la survenance de l'autre n'interrompra pas son traitement.

(1) Le CCB relatif à la gestion de la console est toujours stocké à une adresse fixe en mémoire, il en va de même pour les paramètres qui lui sont relatifs dans les mots réservés.
Donc TIO peut s'effectuer sans chargement préalable.

(2) Ce garnissage variera suivant qu'on exécute TYPE ou TYPEAN. Cette dernière fonction implique nécessairement l'utilisation de chaînage de commandes.

(3) Si le timer atteint la fin de l'intervalle alloué avant apparition du END ;

- Dans le cas de TYPE, il y aura ERREUR, donc branchement en Ⓐ
- Dans le cas de TYPEAN, différents cas sont possibles : branchement en Ⓐ, répétition du message, envoi d'un autre message, ces deux derniers cas étant accompagnés d'une réinitialisation du timer.

(4) Ce branchement s'exécutera lorsque le END DEVICE détectera l'absence de nouvel ordre à traiter : soit qu'il n'existe qu'un seul ordre (certains cas de TYPE), soit que la chaîne soit terminée.

N.B. : Pour la console, il n'y aura donc pas de passage par END CANAL.

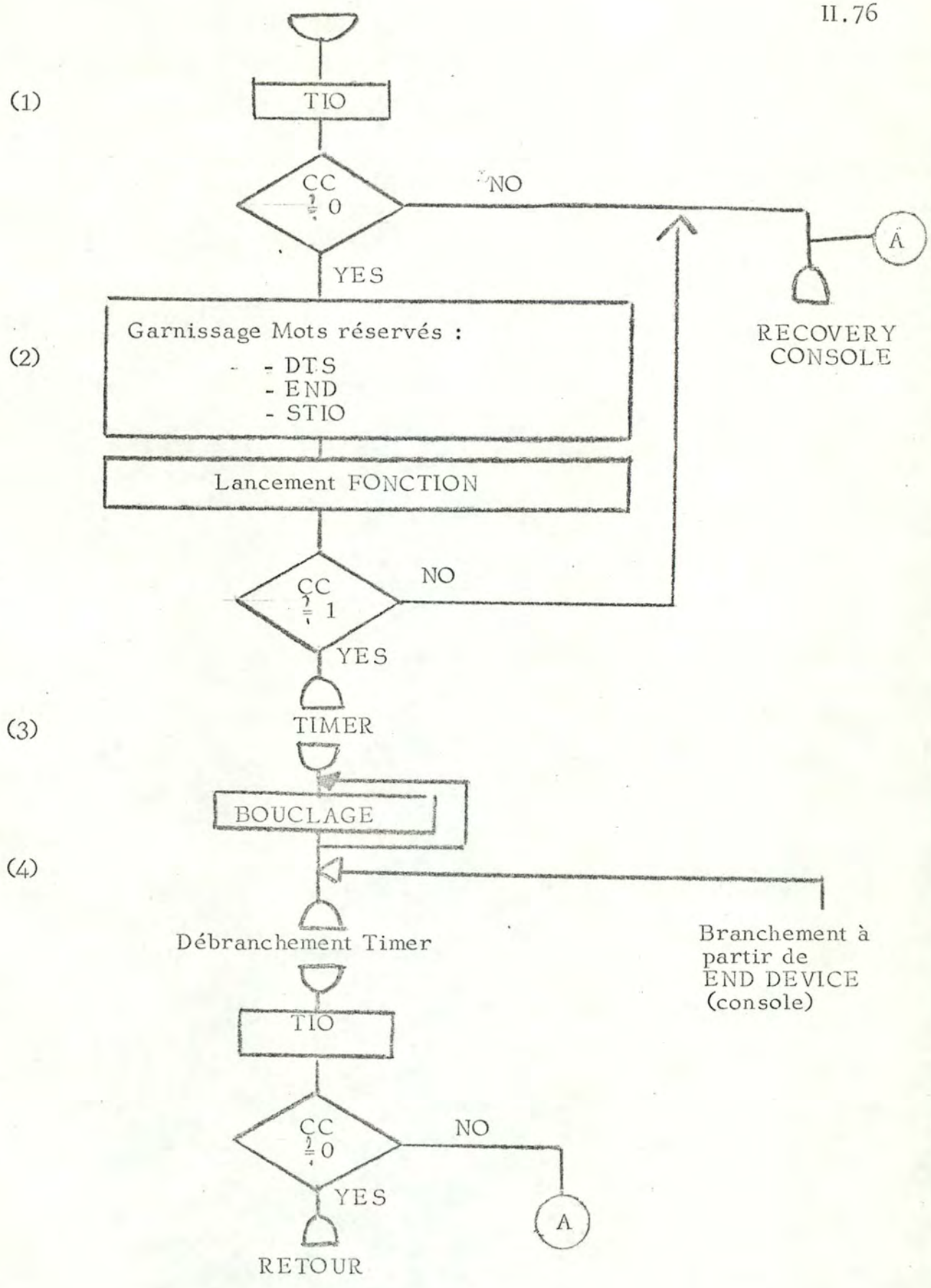

(1)
TIO
CC ≠ 0
NO
A
YES
RECOVERY CONSOLE
(2)
Garnissage Mots réservés :
- DTS
- END
- STIO
Lancement FONCTION
CC ≠ 1
NO
YES
(3)
TIMER
BOUCLAGE
(4)
Débranchement Timer
Branchement à partir de END DEVICE (console)
TIO
CC ≠ 0
NO
YES
A
RETOUR

Fig. II.25

2.3.2.5. TYPE AN.

Cette routine assure la sortie d'un message à la console, suivie de l'entrée de la réponse.
Celle-ci fonctionnera donc successivement comme imprimante (mode OUT), et comme périphérique d'entrée.
La fonction TYPE AN assure :

1. garnissage des mots réservés associés à la console (avec chaînage des fonctions d'écriture et de lecture).

2. Lancement de la première fonction (la console, mise en mode OUT par cet ordre, passera en mode IN à la réception de l'ordre de lecture).

3. Attente de fin d'échange de message.

Ce fonctionnement est décrit à la fig. II.25.

N.B. : Nous avons considéré que la réponse au message était entrée sous la responsabilité du CPU: la machine attend la réponse de l'opérateur. Pendant ce temps, toutes les interruptions de programme masquables restent masquées.

Une solution peut-être meilleure aurait été de considérer l'envoi du message et sa réponse comme deux fonctions distinctes, l'une s'effectuant sous le contrôle du CPU, (TYPE), et l'autre étant déclenchée par l'interruption I_{APPEL} : la réponse à l'opérateur serait alors considérée comme une commande de l'opérateur au système.

2.3.2.6. TIMER.

Cette routine permet à la séquence qui l'appelle de se faire interrompre après un intervalle de temps.
Pour ce faire, celle-ci fournira deux paramètres à TIMER :

- la valeur de l'intervalle de temps,
- l'adresse où brancher lorsque celui-ci sera écoulé.

Le fonctionnement de la routine TIMER est décrit ci-dessous :

1. chargement des paramètres (intervalle de temps, adresse de la routine).

2. lancement de l'ordre au timer,

3. retour à la séquence appelante, qui décidera de continuer ou de se bloquer (SUSP).

Le Timer fonctionnera de la façon suivante :

toutes les X micro-secondes,
(X étant défini par le hardware), la valeur de l'intervalle sera décrémentée.
Lorsqu'elle atteindra 0, l'APPEL du timer sera excité et positionnera le bit correspondant du banc des Requests.
La réponse à ce request provoquera le branchement à l'adresse fournie comme second paramètre à TIMER.

N.B. : il sera nécessaire d'introduire un mécanisme de protection du Timer, celui-ci ne pouvant pas fonctionner simultanément pour plusieurs demandes.

2.3.2.7. END CANAL.

Cette routine interviendra en réponse au Request END CANAL, celui-ci étant positionné par la routine de réponse au END DEVICE dans les cas suivants :

- il n'y a pas de chaînage de commande, ou la chaîne est terminée (CCW COUNT = CCW MAX).
- la chaîne doit être rompue à cause de la détection d'erreur.

La fonction est décrite à la fig. II.26.

(2) Dans le cas où le périphérique présente une erreur, la routine RECOVERY sera appelée.

(6) Ce Strobe permet au périphérique de retourner à l'état RESET, il n'est sans doute pas nécessaire pour tous les périphériques.

(7) Il faut aussi assurer la synchronisation par rapport au programme utilisateur, c'est-à-dire, soit le débloquer, soit lui signaler la fin de l'opération. Nous avons en fait un mécanisme élémentaire de sémaphores.

N.B. : L'attribution d'un Request "END CANAL" à chaque périphérique risque d'être coûteuse. Il serait peut-être bon de prévoir un seul Request pour l'ensemble des périphériques. Dans ce cas, au mécanisme élémentaire décrit ci-dessus devra s'ajouter une gestion de file d'attente des demandes provenant des différents sous-canaux.

Rappelons que ces routines ont été décrites uniquement au point de vue fonctionnel, en mettant l'accent sur l'intégration des éléments définis aux niveaux inférieurs (signaux du canal, instructions IØ) . Les rares précisions apportées ne se trouvaient là que pour éclairer la logique du déroulement.
Nous n'avons donc tenu compte d'aucune contrainte de programmation : celles-ci seraient à définir avec exactitude lors de la phase d'implémentation de ces procédures.

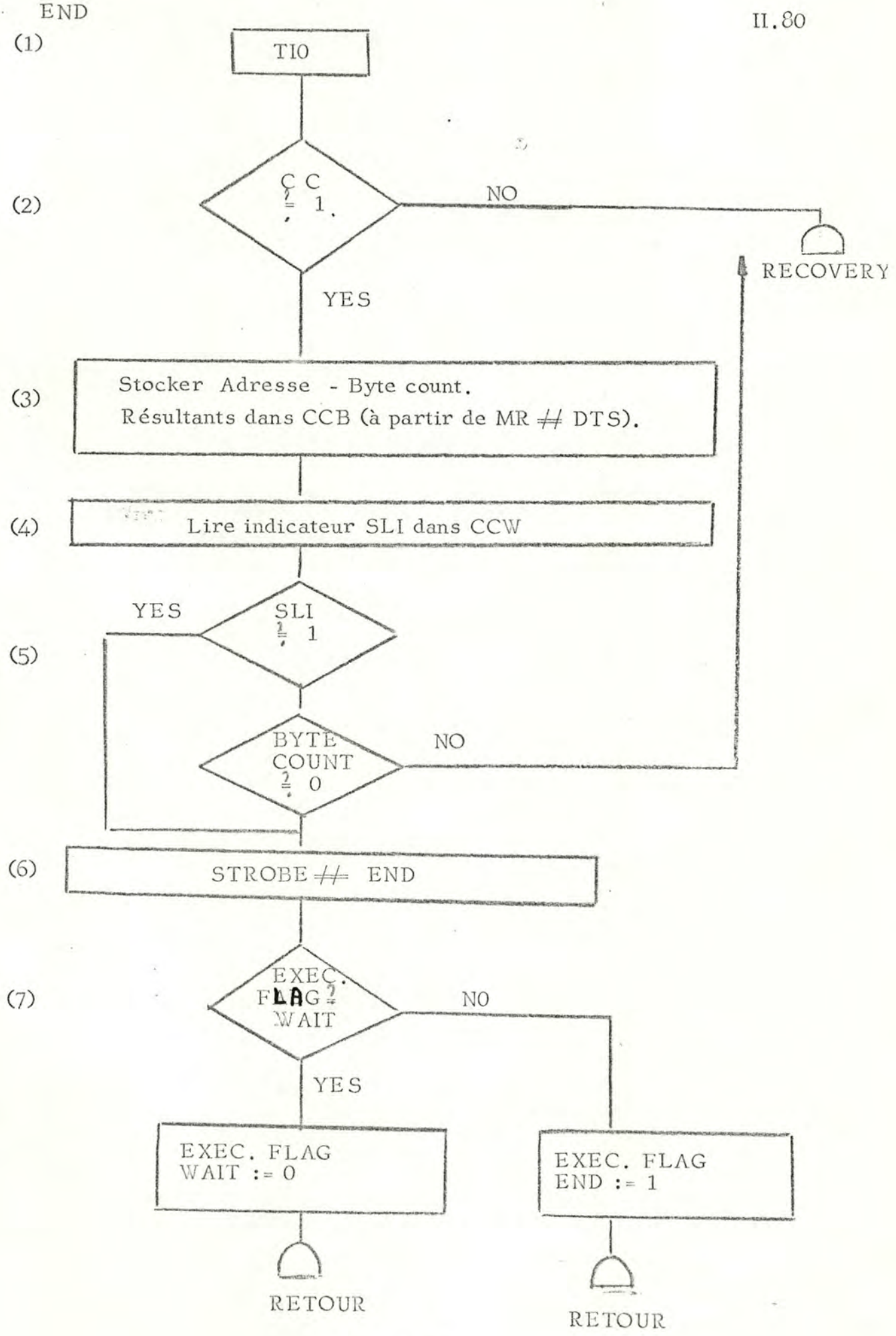

END
(1)
TIO
(2)
CC ? = 1
NO
RECOVERY
YES
(3)
Stocker Adresse - Byte count.
Résultants dans CCB (à partir de MR # DTS).
(4)
Lire indicateur SLI dans CCW
YES
SLI ? = 1
(5)
BYTE COUNT ? = 0
NO
(6)
STROBE # END
(7)
EXEC. FLAG WAIT ?
NO
YES
EXEC. FLAG WAIT := 0
EXEC. FLAG END := 1
RETOUR
RETOUR

Fig.II.26

2.4. PROGRAMME UTILISATEUR.

Le programme utilisateur exécute les IØ exclusivement par l'intermédiaire du superviseur d'IØ, en lui adressant des demandes accompagnées des paramètres nécessaires.

Deux aspects sont donc à développer : d'une part le comportement du programme en début d'IØ, pendant, et à la fin, et d'autre part la description des paramètres fournis par le programme.

2.4.1. Actions.

La demande formulée par le programme au moyen d'un SVC fera référence à une zone de description de l'IØ (CCB chez SIEMENS, DCB chez IBM).

Une fois la demande traitée, (c'est-à-dire, la plupart du temps, l'IØ lancée), le contrôle est rendu par le superviseur au programme à l'adresse sauvée dans l'ESA(1).La décision de travailler ou non parallèlement au déroulement de l'IØ est donc entièrement du ressort du programme (suivant que le traitement continue, ou se bloque sur un SVC WAIT).

La fin d'IØ, traitée par interruption, ne sera reconnue par le programme que par la fonction WAIT.

Pour faciliter le travail du programmeur lorsqu'il veut organiser son programme en tâches asynchrones, permettant le déroulement parallèle, il serait intéressant d'introduire la notion de multitasking et les modules de l'Executive nécessaires pour la gestion d'une telle organisation.

2.4.2. Descriptions.

Le second point à préciser est relatif aux communications d'informations entre le programme et le superviseur. Une zone de communication (ou CCB) sera associé à chaque opération d'IØ, une par le programme utilisateur

Cette zone comprendra deux parties : la première regroupera les paramètres fournis au superviseur par le programme utilisateur, la seconde mettra à la disposition du programme utilisateur les informations résultant du travail du superviseur.

---

(1) ESA = nom donné chez SIEMENS à la zone de sauvetage des registres.

Cette partie regroupera donc les zones de stockage des :

- STATUS BYTE du périphérique,
- SENSE BYTE du périphérique,
- paramètres résultants du programme canal. (BYTE COUNT, ADRESSE MEMOIRE).
- EXECUTIVE FLAG qui résume pour l'utilisateur le traitement effectué par le superviseur concernant cette IØ. Nous y trouverons les indicateurs suivants :
    - END traité (positionné par routine END CANAL),
    - WAIT (positionné par routine WAIT),
    - IRRECUPERABLE (positionné par routine RECOVERY quand elle détecte une erreur irrécupérable).
    - IØ LANCEE (positionné par EXCP),
    - les différentes conditions exceptionnelles.

L'autre partie, garnie par l'utilisateur, comprendra :

- identification du PERIPHERIQUE,
- référence à la chaîne de CCW (ou au CCW) (en tenant compte de la remarque formulée à leur propos cf. p. II 38).
- option de l'utilisateur pour CORRECTION .
- Adresse de sortie d'erreur dans le cas où l'utilisateur souhaite tenter la correction.

La génération du CCB se fera à l'assemblage, à partir d'une pseudo-instruction fournissant les paramètres qu'il doit transmettre.

CONFIGURATION DU CCB.

Status byte	BYTE COUNT	
EXECUTIVE FLAG	A (IOAREA)	(1)
	SENSE BYTE(S)	
IDENTIFICATION PERIPHERIQUE	A (CCW)	(2)
USER RECOVERY FLAG	A(ROUTINE DE RECOVERY)	

Fig. II.27

---

(1) partie garnie grâce aux paramètres fournis par l'utilisateur.

(2) garni en cours d'IØ par le superviseur.

CHAPITRE III - AUTRES MODULES.

CONFIGURATION MEMOIRE.

Outre les mécanismes décrits dans les deux chapitres procédents, le système, pour être utilisable, doit compter d'autres modules. Sont actuellement prévus : l'assembleur (et éditeur de liens), le module de prise en charge du programme objet, le chargeur initial du système, et un système élémentaire de gestion de fichiers. Tous ces modules sont encore soit à l'étude, soit à définir. C'est pourquoi nous en fournirons une vue très rudimentaire.

Au vu du système ainsi défini, nous proposerons une configuration mémoire qui tient compte des différents modules évoqués, ainsi que des tables propres à chacun d'eux.

1. ASSEMBLEUR.

Une étude préliminaire consistait en la définition exacte du langage machine, celle-ci a fait l'objet d'un travail parallèle. (cf. SEMINAIRE "Ordinateur Pédagogique").

L'assembleur devra traiter les instructions du programme (contrôle, en référence à la définition du langage machine, et traduction des codes et adresses mnémoniques).
Il devra aussi reconnaître un certain nombre de pseudo-instructions (disparues du programme après assemblage). Parmi celles-ci, nous retiendrons : les réservations de constantes, les réservations de place, et la génération de tables (description de fichiers, CCB).

L'éditeur de liens sera le complément indispensable de l'assembleur pour permettre le déroulement de programmes un peu complexes (c'est-à-dire regroupant plus d'une routine, ou "control section").

2. MODULE DE PRISE EN CHARGE PROGRAMME.

2.1. ROLE.

Nous distinguerons trois types de routines dans ce module : celles qui traitent le chargement du programme, celles qui interviennent en cours d'exécution, et celles qui assurent sa terminaison.

2.1.1. CHARGEMENT.

Il est nécessaire d'assurer d'une part l'assignation et la réservation des périphériques demandés pour ce programme, et d'autre part, le chargement et le lancement du programme.

L'ASSIGNATION des périphériques fournit à ceux-ci des noms symboliques (ou logiques). Le programme ne connaîtra les périphériques qu'il utilise que par ces noms symboliques, susceptibles d'être différents des noms mnémoniques (ou physiques) que le système utilise.
Cette précaution permet à l'utilisateur d'écrire un programme relativement indépendant de la configuration physique de l'installation. Lorsqu'il n'existe aucune assignation explicite, le système utilise les assignations standard.

La RESERVATION des périphériques protège les accès d'un programme aux périphériques qu'il s'est réservé, cette réservation pouvant être valable pour toute la durée du programme, ou seulement pendant l'exécution d'une IØ.
En cas de monoprogrammation, cette réservation présente une utilité très réduite.

Le CHARGEMENT d'un programme s'effectue par le rangement en mémoire centrale du programme objet, prêt à l'exécution; celui-ci provient soit d'une bibliothèque, soit du périphérique d'entrée du système (en général, lecteur de cartes).

Dans le cadre de la multiprogrammation, l'introduction en mémoire doit être accompagnée de l'affectation à ce programme d'un numéro de priorité et de l'allocation d'une zone mémoire (bornes utilisées pour protection mémoire).
Puisque nous débutons dans un contexte de monoprogrammation, le programmeur pourra disposer de toute la mémoire non utilisée par le système.

Le LANCEMENT se fera automatiquement après chargement du programme, par stockage dans le p-counter (registre CP) de l'adresse du point d'entrée.
Ici aussi, l'hypothèse de monoprogrammation simplifie la tâche en éliminant les possibilités de conflit entre plusieurs programmes chargés, prêts pour le lancement.

2.1.2. EN COURS D'EXECUTION.

Nous citerons, en premier lieu, les fonctions susceptibles de modifier l'état du programme, en particulier son passage de l'état ACTIF à l'état BLOQUE.

Cette fonction pourrait se réaliser de deux façons :

- le programme demande lui-même à être bloqué dans l'attente d'un évènement extérieur.
  Dans ce cas, le débloquage dépendra de celui-ci.
  (Ex. : TIMER, fin IØ, ordre console).

- un programme bloque une tâche ou un programme qui dépend de lui.
  Cette application n'est possible que si l'on travaille en multitasking ou en multiprogrammation.

Une autre fonction qu'il serait bon d'introduire est la demande formulée par le programme de gérer lui-même certaines interruptions.

Pour cela, les mots Réservés associés à l'interruption concernée devront contenir l'adresse de la routine utilisateur. Le chargement de ceux-ci ne peut se faire que sous le contrôle du système ; il se fera à la demande du programme, en cours d'exécution, quand l'utilisateur veut que ses routines interviennent.

N.B. : une telle gestion impliquera la nécessité de mise à jour des mots réservés lors du chargement de chaque programme.

2.1.3. TERMINAISON.

Celle-ci peut correspondre à la fin normale du programme et s'exécuter à sa demande mais elle peut être anticipée et demandée par le système, par exemple, dans le cas d'erreurs graves et irrécupérables par le système.

D'autre part, elle peut avoir pour conséquences, la mise en attente du système, ou la prise en compte automatique du programme suit.

N.B. : Les fonctions évoquées ci-dessus se rapportent directement à la gestion du programme.

Pour faciliter l'utilisation du système, il serait utile d'introduire des fonctions annexes.

Parmi celles-ci, nous retiendrons :

1. communication de l'heure (en début et fin de programme, pour comptabilité).
2. communication de tables du système.
3. vidage de zones de la mémoire.

Il est clair que cette liste n'est pas exhaustive, et sera complétée en temps utile.

Dans le suite du chapitre, nous ne tiendrons pas compte de ces différentes demandes.

2.2. COMMANDES.

Les différentes fonctions que le système exécutera pour assurer la gestion correcte du programme peuvent avoir 3 origines différentes : le système, le programme lui-même et l'opérateur à la console.
(Il faudra aussi tenir compte des interventions du MONITEUR lorsque celui-ci sera introduit).

Lorsque le programme devra faire appel à une routine de ce module (SUSP, EOJ), il utilisera la procédure des SVC, décrite précédemment.

Le passage d'un module à l'autre du système ne présente aucun problème, dans la mesure où ceux-ci font partie du système résident; sinon il faudra faire appel au module de gestion des overlays.

N.B. : une organisation par overlays n'est rentable que lorsque le système non résident est rangé sur un support suffisamment performant bande ou disque
Dans une première phase, il serait souhaitable que le système soit suffisamment réduit pour être résident, en raison de la faiblesse des périphériques.

Le troisième moyen de lancer une fonction du module qui gère le programme est fourni par la console.
Dans le cadre des IØ, nous avons défini certaines fonctions de la console (TYPE, TYPEAN), où celle-ci intervenait en tant que périphérique, qui assurait l'entrée ou la sortie de messages, et ce, sous le contrôle du CPU.
Mais la console remplit aussi un rôle privilégié de périphérique de commande, en fournissant des ordres au système. Dans ce cas, l'opérateur prend l'initiative de l'intervention (I_{APPEL}). Le traitement de l'interruption consistera en l'analyse du CONTENU du message, et le branchement à la routine adéquate, via la "table des routines du système".

Ces appels de l'opérateur peuvent intervenir à n'importe quel moment de l'exécution du programme.
Cependant, ils seront supprimés lorsque la console est déjà occupée par l'échange de message, sous le contrôle du CPU (fonction TYPE ou TYPE AN).

Dans une première définition, nous retiendrons les fonctions suivantes :

ASSGN	assignation d'un nom symbolique (logique) à un périphérique.
EOJ	Fin normale du programme utilisateur.
INTRPT	Prise en charge d'interruption par le programme utilisateur.
LOAD	Chargement et lancement du programme.
RESV	Réservation périphérique.
SUSP	Mise du programme dans l'état bloqué.
TERM	Fin anticipée du programme à la demande du système.
TERMD	idem + vidage Mémoire.
TERMS	idem + prise en charge du programme suivant.

Ces différentes fonctions pourro nt être appelées par l'un ou l'autre des moyens signalés, comme indiqué dans la fig. ci-dessous.

	Programme	Système	Opérateur
ASSGN	x	x	x
EOJ	x		
INTRPT	x		
LOAD		x	x
RESV	x		
SUSP	x	x	
TERM		x	x
TERMD		x	x
TERMS		x	x

N.B. : Cette liste ne se veut pas exhaustive : certaines fonctions supplémentaires s'avéreront sans doute nécessaires. Néanmoins, elle peut donner une vue globale des fonctions nécessaires à la gestion d'un programme.

3. INITIALISATION DU SYSTEME.

Signalons simplement que celui-ci s'effectuera sous le contrôle du firmware.
L'exécution d'un nombre limité de micro-instructions câblées dans la mémoire rapide permettra le chargement de la mémoire rapide (micro-instructions vives) et de la mémoire centrale (mini-instructions puis software).

4. SYSTEME DE GESTION DE FICHIERS (SGF).

Interface entre le superviseur IØ et le programme utilisateur, le système de gestion de fichiers permet à ce dernier de manipuler les articles logiques de son fichier, et de se décharger de la gestion des IØ physiques : en fonction des opérations sur les articles, et de la configuration du fichier, le système de gestion de fichier enverra les ordres adéquats au superviseur IØ.

L'élaboration de ce système a débuté dans le cadre du séminaire "ordinateur pédagogique".
Ce travail se limite au traitement de fichiers à organisation séquentielle. Cette restriction de taille se justifie par la nature des périphériques dont la machine disposera, du moins dans les premiers temps.
Il est d'ailleurs à noter que, dans la première phase, ce système de gestion de fichiers sera inexistant ou extrêmement réduit, puisque les seuls périphériques utilisés seront un lecteur de cartes et une imprimante élémentaire.

L'étude comporte deux parties : la description des fichiers, et les fonctions de commande.
La description des fichiers est à la communication programme-système de gestion de fichiers, ce que le CCB était à la communication programme-superviseur IØ : à partir de paramètres fournis par l'utilisateur (dans une pseudo-instruction), l'assembleur construit la table que consultera le SGF pour l'exécution des fonctions de commande.

Celles-ci regrouperont les opérations sur la structure globale du fichier (OPEN, CLOSE) et les manipulations d'articles (PUT, GET).

Les modules que nous avons évoqués ici constitueront donc, avec le système IØ et la prise en charge des interruptions, le système de base.

Il est fréquent de regrouper sous un même vocable l'ensemble des modules résidents, utilisés p our la gestion des IØ.

C'est pourquoi, pour éviter la lourdeur des notations, nous utiliserons le terme de NUCLEUS pour désigner l'ensemble suivant : superviseur IØ, interruptions, et module de gestion du programme.
(parfois appelé aussi "EXECUTIVE".)

5. TABLES DU SYSTEME.

Outre les modules que nous venons de décrire, le système de base comprendra diverses tables que consulteront ou garniront les routines. Nous retiendrons:

1. ZONE DES MINI-INSTRUCTIONS.

A chaque code instruction correspondent un ou plusieurs mots mémoires (suivant complexité de l'instruction).
Dans ces mots sont rangées les adresses en mémoire rapide des micro-instructions composantes de cette instruction. Cette zone, de longueur constante, sera chargée à l'initialisation du système.
Elle sera consultée lors de la prise en charge de chaque instruction.

2. ZONE DES MOTS RESERVES.

A chaque Request câblé, et à chaque instruction d'IØ est associé un groupe de 4 mots Réservés.
(cf. chapitre 1).

Cette zone sera garnie partiellement à l'initialisation du système (paramètres constants, mini-instructions), tandis que certains paramètres seront chargés en cours d'exécution des programmes (cf. routines du canal IØ, chapitre 2).

Sa longueur sera fixe pour une configuration donnée : elle sera définie par le nombre de Requests, et le nombre de périphériques (pour instructions IØ).
Elle sera utilisée pour la réponse aux interruptions ou pour le lancement d'instructions IØ.

3. LISTE DES PERIPHERIQUES.

A chaque périphérique existant dans la configuration correspondra une zone de cette liste, contenant les renseignements suivants :

- caractéristiques physiques du périphérique.
- numéro physique (par lequel le système peut accéder aux mots réservés qui lui sont associés).
- éventuellement, certains renseignements relatifs à l'état du périphérique (Réservé, en Recovery, occupé).
  Ceux-ci seront à définir lors de la réalisation des routines IØ proposées au chapitre 2.

Cette table aura donc une longueur fixe pour une configuration donnée, dépendant du nombre de périphériques connectés.
Elle sera chargée à l'initialisation du système, et sera consultée lors du lancement d'opérations IØ.

4. TABLE D'ASSIGNATION.

Elle établit la correspondance entre les numéros physiques des périphériques, et les noms symboliques par lesquels le programme les désigne.

Cette table est donc mise à jour au lancement de chaque programme. Sa longueur dépend, elle aussi du nombre de périphériques existant sur la configuration.

5. TABLE DES ROUTINES DU SYSTEME.

Celle-ci regroupera les références aux routines du NUCLEUS. A partir des routines définies jusqu'ici , nous pouvons décrire cette table de la façon suivante :

		Adresse des paramètres
EXCP		A (CCB)
WAIT		A (CCB)
RECOVERY (1)		A (CCB)
TYPE	ADRESSE	A (Msg)
TYPEAN	DES	A (Msg)
TIMER	ROUTINES	intervalle de temps
END CANAL		A (CCB)
ASSGN		- (3)
EOJ		-
INTRPT		A(Routine), # Interruption
LOAD		- (3)
RESV		- (3)
SUSP		- (3)
TERM		- (3)
TERMD		- (3)
TERMS		- (3)

Cette table sera chargée lors de l'initialisation du système, et consultée à chaque introduction dans le système.

N.B. : (1) Pour simplifier l'écriture, nous avons envisagé un seul point d'entrée dans RECOVERY. Il serait peut-être utile d'en prévoir un par périphérique.

(2) Ces routines peuvent être initialisées par l'opérateur, le système, ou l'utilisateur. Il serait peut-être bon de spécifier cette origine dans la table.

(3) Pour les routines utilisant des tables stockées dans des zones constantes pour le système, les adresses des paramètres ne doivent pas être spécifiées.

6. TABLE DU PROGRAMME.

Fournit les renseignements relatifs à l'identification et l'état du programme.
Nous citerons :

- nom du programme,
- adresse de début et fin,
- état (actif/bloqué),
- temps de début - fin;
  (comptabilité).

D'autres éléments seront peut être à introduire.
Il est clair que cette table, de longueur fixe, sera garnie lors de la prise en charge de chaque programme.

7. ZONE DE SAUVETAGE.

A chaque niveau d'interruptibilité correspondent les mots mémoires destinés au sauvetage des registres, p-counter et code condition.
En raison de la manière dont est organisée la prise en charge des interruptions, une zone de sauvetage est associée au niveau interrompant et non à la séquence interrompue.
Nous aurons :

1 zone pour toutes les interruptions de niveau 1.
(Externes + éventuellement certaines erreurs software).

1 zone pour chacune des interruptions non masquables.
(Code invalide, adresse invalide, erreur machine, et éventuellement les autres erreurs software).

8. ZONE OVERFLOW DES MOTS RESERVES.

Ainsi que nous l'avons vu dans le chapitre 2 (cf. II), certaines routines du canal IØ nécessitent une zone plus étendue que celle fournie par les mots réservés paramètres.
C'est pourquoi les "zones manoeuvre" seront prélevées dans cette zone, que nous pouvons subdiviser en morceaux fixes, ou gérer à la façon d'un POOL, ce qui compliquerait quelque peu la procédure, mais permettrait un gain de place mémoire.

9. TABLE DU NUCLEUS.

Parmi les tables que nous venons de citer, certaines sont de longueur fixe, mais certaines sont de longueur variable.
Nous aurons donc une récapitulation des adresses de ces différentes tables dans la table du Nucleus.
Celle-ci contiendra :

1. Données générales : JOUR/MOIS/ANNEE, heure.
   utilisées pour la comptabilité.

2. Les adresses des tables :

   A (ZONE MOTS RESERVES)
   A (TABLE ROUTINES DU SYSTEME)
   A (DEVICE LIST)
   A (TABLE D'ASSIGNATION)
   A (TABLE PROGRAMME)
   A (ZONES SAUVETAGE)

   La configuration mémoire résultante est indiquée à la fig. III.1 .

CONFIGURATION MEMOIRE.

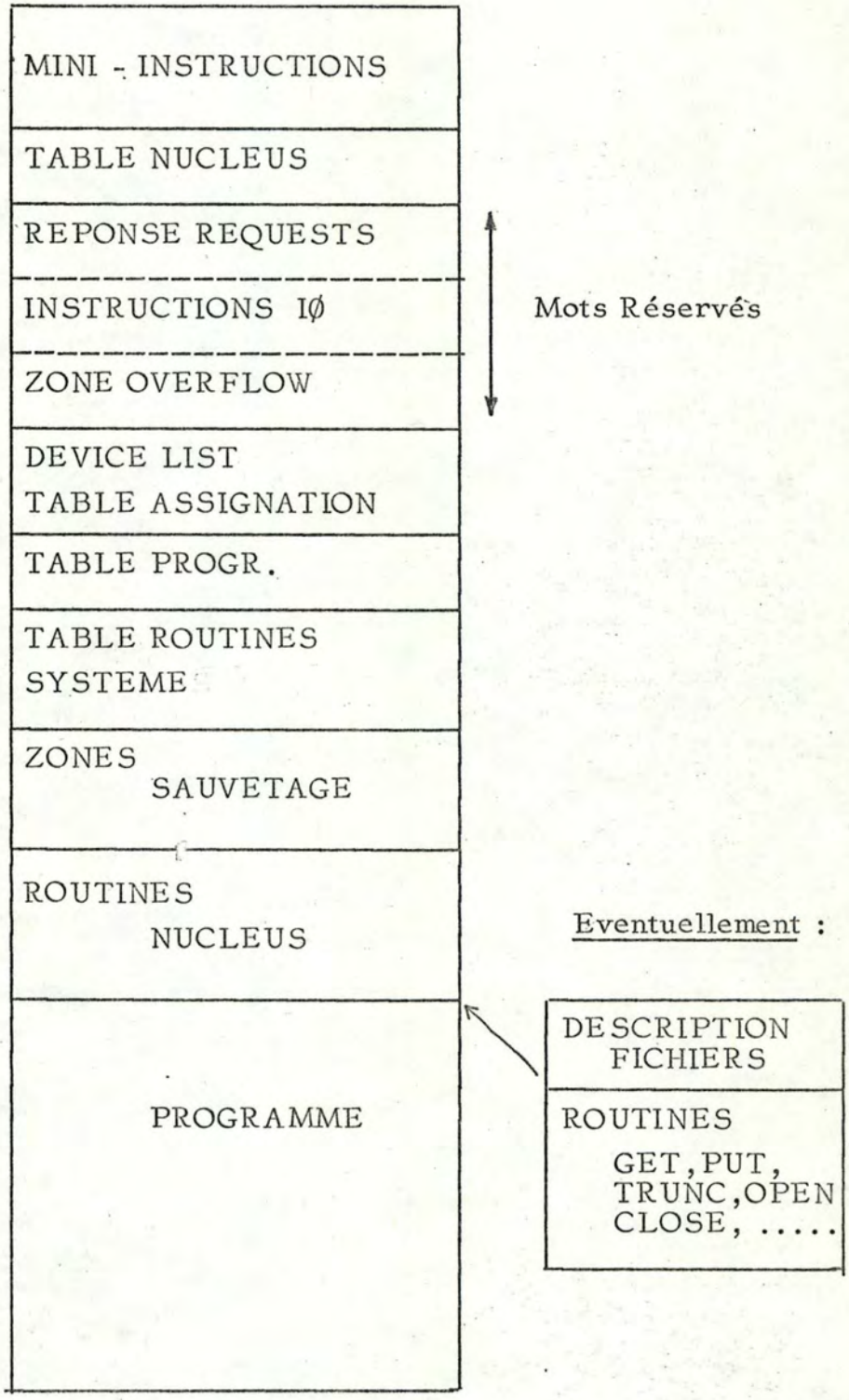


Fig.III.1

CONCLUSION

Ainsi que nous l'avons signalé au départ, cette description est très fragmentaire : même du point de vue strictement fonctionnel, de nombreux modules sont encore à préciser.

L'étape la plus décisive sera celle de la réalisation, rendue possible lorsque le langage de base sera complètement défini. Rappelons toutefois, que dans l'optique du firmware dynamique, celui-ci ne sera jamais totalement défini : il existera toujours la possibilité de créer de nouvelles instructions en fonction des besoins.
C'est ainsi que pour des procédures appelées à fonctionner fréquemment, il serait intéressant d'appliquer le mécanisme suivant : écrire les procédures en software (à partir des instructions existantes), les mettre au point, les optimiser, puis les implémenter dans le firmware, en créant de nouvelles instructions se référant à ces séquences firmware. Ceci aurait pour conséquence de faciliter la tâche du programmeur (écriture d'une seule instruction), et d'accélérer l'exécution de ces procédures.

Lorsque ce système élémentaire sera implémenté, il faudra songer à l'enrichir, en fonction du type d'applications que devra traiter l'EPRON : télétraitement, calcul scientifique, ou gestion, autant de possibilités qui nécessitent des modules différents.

Il est à noter que, parmi les apports au système élémentaire, il sera sans doute indispensable d'introduire le multitasking ou la multiprogrammation.

Il nous reste à souhaiter que l'EPRON ne reste pas à l'état de projet ou de vieux papier : qu'un jour nous puissions dire :
" il vécut de longues années, et il y eut beaucoup de petits EPRON. ".

BIBLIOGRAPHIE

"IBM SYSTEM/360 - PRINCIPLES OF OPERATION".
GA 22 - 6821 - 8.

"SUPERVISORY AND MONITOR SYSTEMS".
Robert F. ROSIN
Computing Surveys, Mars 1969.

"ARCHITECTURE OF THE IBM 360".
AMDAHL, BLAAUW, BROOKS
IBM Journal - Avril 1964.

"ARCHITECTURE DE SYSTEMES ET MULTIPROGRAMMATION DYNAMIQUE".
J. DEMARTEAU ([D])
Institut d'Informatique - 1973.

"PHYSICAL FCP PROGRAMMERS'TRAINING MANUAL".
RCA.

"SIEMENS 4004 : PBS EXECUTIVE".

TELEMECANIQUE : "Manuel de présentation".

BURROUGHS 5500 : Reference Manuel.

- - - - - -