MASTER EN SCIENCES INFORMATIQUES

Un réseau local intelligent

Dumont, Roland

Award date:
1979

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES NOTRE DAME DE LA PAIX - NAMUR

INSTITUT D'INFORMATIQUE

ANNEE ACADEMIQUE 1978 - 1979

UN RESEAU LOCAL INTELLIGENT

PROMOTEURS: Mr. Ph. van BASTELAER
Mr. J-D. NICOUD

Mémoire présenté par

ROLAND DUMONT

en vue de l'obtention du grade de Maitre
et Licencié en Sciences Informatiques

FACULTES UNIVERSITAIRES NOTRE DAME DE LA PAIX - NAMUR

INSTITUT D'INFORMATIQUE

ANNEE ACADEMIQUE 1978 - 1979

UN RESEAU LOCAL INTELLIGENT

PROMOTEURS: Mr. Ph. van BASTELAER
Mr. J-D. NICOUD

Mémoire présenté par

ROLAND DUMONT

en vue de l'obtention du grade de Maitre
et Licencié en Sciences Informatiques

TABLE DES MATIERES.

INTRODUCTION.

Jusqu'aux années 60, l'informatique a fait essentiellement usage d'ordinateurs isolés. Le seul accès à l'ordinateur se trouvait sur le site même de la machine.

Cette limitation d'accès fut levée par l'introduction de terminaux permettant la communication à distance avec l'ordinateur central. Un ordinateur entouré de plusieurs terminaux constituait le premier réseau informatique.

L'apparition des miniordinateurs a permis de traiter certaines informations "à la source" en évitant ainsi le passage obligatoire par l'ordinateur central. Plusieurs miniordinateurs interconnectés peuvent fournir les mêmes services qu'un gros ordinateur central.

Finalement, la venue des microprocesseurs a considérablement élargi le rôle des terminaux en les rendant capables d'exécuter des processus.

L'évolution actuelle tend à décentraliser l'intelligence en la distribuant entre les sites des utilisateurs.

Cette idée de décentralisation trouve aussi sa répercussion sur le fonctionnement interne des ordinateurs. En effet, le processeur central unique des premiers ordinateurs a tendance à être remplacé par une association de processeurs spécialisés coopérant à l'exécution des processus.

Dans ce rapport seront traités seulement les réseaux d'ordinateurs. Se trouvent par conséquent exclu de la discussion, les configurations avec un ordinateur central entouré de terminaux "stupides".

Le but de ce travail est de donner une introduction aux réseaux locaux qui rendent possible la transmission d'informations à grande vitesse sur des petites distances. Un exemple illustrera une réalisation d'un réseau local avec des solutions intelligentes.

La structure hiérarchique de ce rapport comporte: parties, sections, chapitres, divisions. Il est divisé en 3 grandes parties:

I. Introduction aux réseaux locaux en partant des réseaux d'ordinateurs traditionnels.

II. Présentation d'un réseau local avec des solutions intelligentes.

III. Apport personnel à un projet concernant le réseau local introduit dans la seconde partie.

REMERCIEMENTS.

Ces remerciements sont particulièrement adressés aux personnes suivantes:

- Prof. J.-D. NICOUD pour avoir dirigé le projet qui se trouve à la base de ce rapport
- Prof. Ph. van BASTELAER pour m'avoir donné des conseils quant au contenu et à la rédaction de ce rapport
- Mr. R. SOMMER pour m'avoir aidé dans le travail de mon projet

D'autre part, je remercie vivement tous les collaborateurs du Laboratoire de Calculatrices Digitales à l'Ecole Polytechnique Fédérale de Lausanne pour m'avoir accueilli dans leur laboratoire.

La figure 12 est publiée dans ce rapport avec l'autorisation de Mr. René SOMMER.

I° PARTIE

RESEAUX ET RESEAUX LOCAUX

I.1. INTRODUCTION AUX RESEAUX D'ORDINATEURS.

I.1.1. DEFINITION GENERALE D'UN RESEAU D'ORDINATEURS.

Le terme de réseau d'ordinateurs (computer network) est souvent utilisé dans un sens très large. Ainsi il désigne aussi bien deux processeurs qui se partagent une mémoire commune, que plusieurs ordinateurs interconnectés sur de longues distances. La définition d'un réseau d'ordinateurs ou système distribué (distributed system, distributed processors) peut s'exprimer de la façon suivante:

> plusieurs ordinateurs reliés physiquement qui doivent se distribuer une charge d'opérations en se communiquant des informations.

Les différentes parties d'un réseau d'ordinateurs (FIG.1) sont:

- les stations (host computers, computing stations), qui sont constituées d'unités capables d'exécuter des processus et de moyens de transmission d'informations
- le réseau de communication (communication subnetwork), qui forme les liaisons pour permettre aux stations de communiquer.

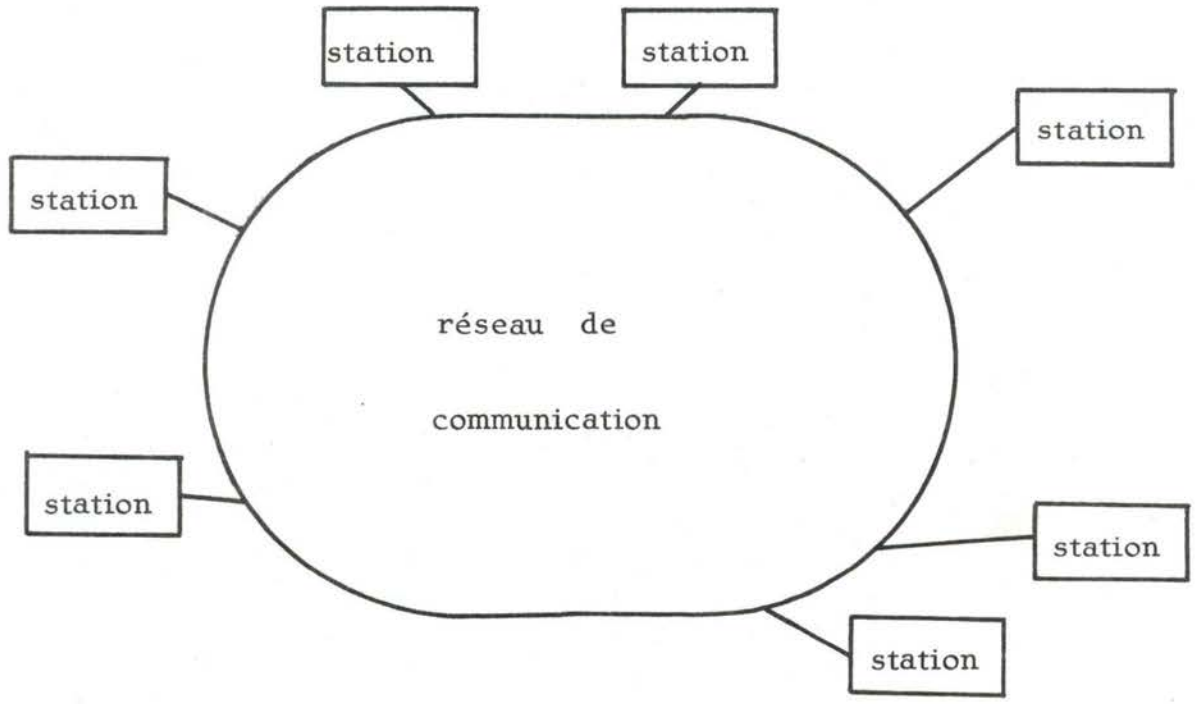


FIG.1

PARTIES D'UN RESEAU D'ORDINATEURS

I.1.2. HIERARCHISATION DES NIVEAUX DANS UN DIALOGUE ENTRE STATIONS.

La structure hardware d'une station (FIG.2) est composée de deux éléments:

- l'élément processeur (processing element), qui assure l'exécution des processus du système d'exploitation et de l'utilisateur
- l'interface (interface), une unité plus ou moins intelligente qui fait la liaison entre l'élément processeur et le réseau de communication.

FIG.2
STRUCTURE HARDWARE D'UNE STATION

La structure software d'une station est formée par le protocole (protocol), défini comme étant un ensemble de règles pour contrôler les dialogues entre les stations à travers le réseau de communication. Un protocole fonctionne à plusieurs niveaux comme on va le voir dans la suite.

Le modèle suivant a été proposé par Thurber (THURBER78). Dans un système distribué il se déroule beaucoup de processus en même temps dans les différentes stations. Donc un tel système peut être considéré comme un ensemble de processus coopérants qui communiquent à travers le réseau de communication. Ces dialogues se font par l'intermédiaire d'une hiérarchie de protocoles. La FIG.3 présente une vue simplifiée d'un système distribué formé de deux stations qui sont reliées par un réseau de communication. L'hiérarchie comporte 4 niveaux où chacun est complètement transparent à tous les niveaux supérieurs. Cela veut dire que chaque protocole d'un certain niveau s'adresse directement à son homologue dans une autre station comme si les niveaux inférieurs n'existaient pas.

Niveau 0: contrôle de ligne (data link control).

Il s'agit des règles qui définissent les envois de bits sur une ligne de transmission. A ce niveau sont implémentées, la synchronisation entre les stations pour la transmission de bits et la détection des erreurs au niveau du bit. Ce niveau est souvent implémenté en hardware et sous

le contrôle du software.

Niveau 1: contrôle de transmission (transmission protocol).

Ce sont les règles pour établir un dialogue logique entre les stations. Selon des conventions, les informations sont groupées et entourées de caractères de contrôle avant d'être envoyées sur les lignes. La détection et la correction des erreurs de transmission au niveau des caractères permettent de savoir si les informations ont bien été reçues à la destination.

Niveau 2: communication entre processus (interprocess communication protocol).

Ce protocole respecte un ensemble de conventions par lesquelles les systèmes d'exploitations des différentes stations établissent, maintiennent et terminent une liaison entre deux processus utilisateurs. C'est l'ensemble des routines et appels système directement accessibles aux utilisateurs.

Niveau 3: protocole de niveau utilisateur (user level protocol).

Ce sont les règles que doivent respecter tous les programmes utilisateurs pour pouvoir faire appel aux routines du niveau 2. D'autres programmes transforment les commandes plus évoluées en une suite d'appels qui sont implémentés au niveau précédent. Ainsi il y a des commandes pour demander le transfert d'un fichier, démarrer un processus dans une autre station, demander un service particulier à travers le réseau.

Souvent les protocoles des niveaux 0 et 1 se déroulent dans l'interface de la station, et les protocoles des niveaux 2 et 3 dans l'élément processeur de la station. Les deux derniers niveaux sont indépendants de l'implémentation particulière d'un réseau, mais dépendent de la structure du système d'exploitation dans les stations.

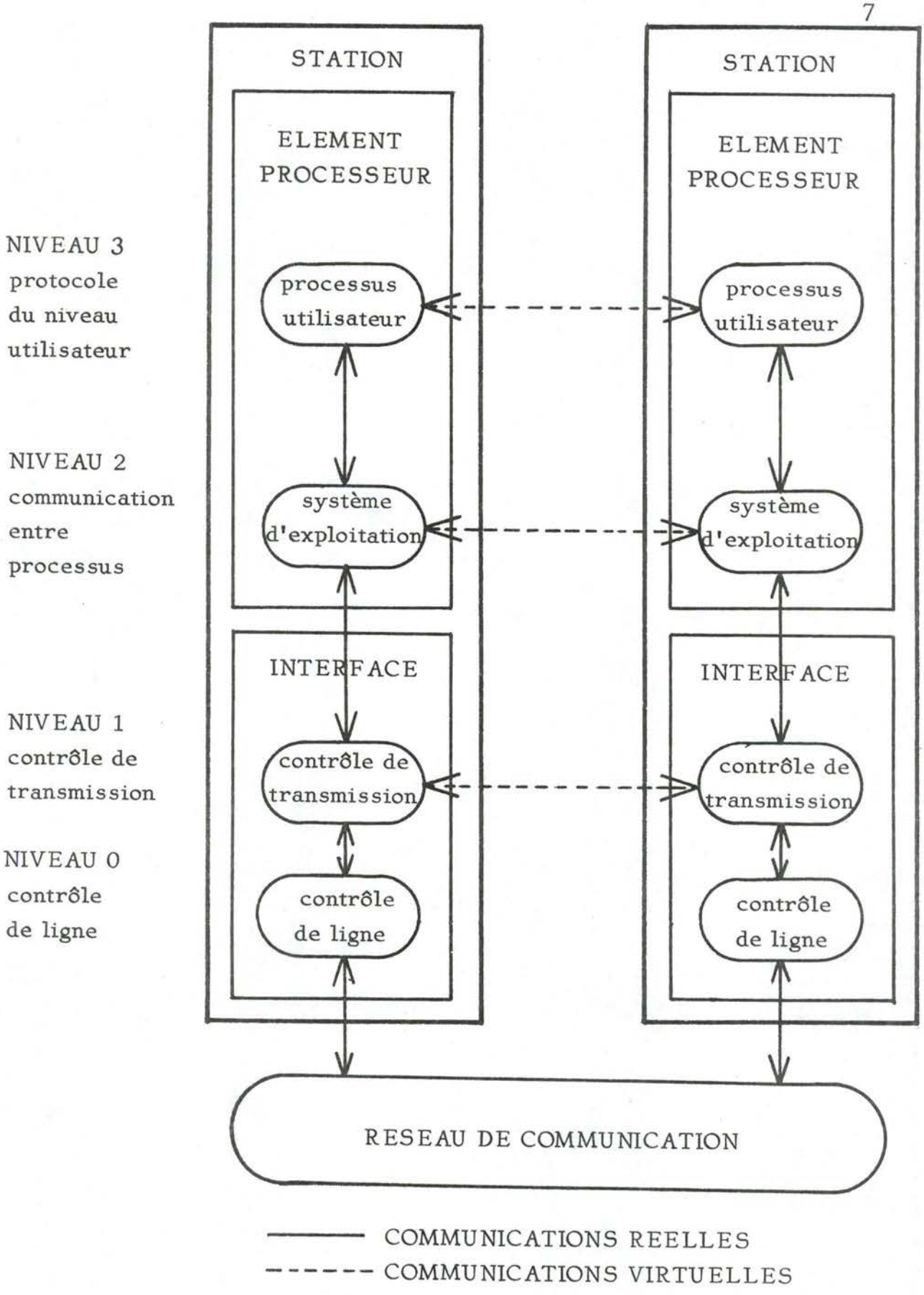


FIG.3

HIERARCHIE DES PROTOCOLES DANS UN DIALOGUE

I.1.3. STRUCTURE D'UN RESEAU DE COMMUNICATION.

Le réseau de communication (FIG.4) peut être sousdivisé en:

- lignes de transmission (transmission lines), qui sont des moyens pour véhiculer des informations (p.ex. lignes téléphoniques, cables, etc.)
- noeuds (nodes), formés d'unités plus ou moins intelligentes qui dirigent les informations à travers le réseau de communication (p.ex. concentrateur, multiplexeur, commutateur).

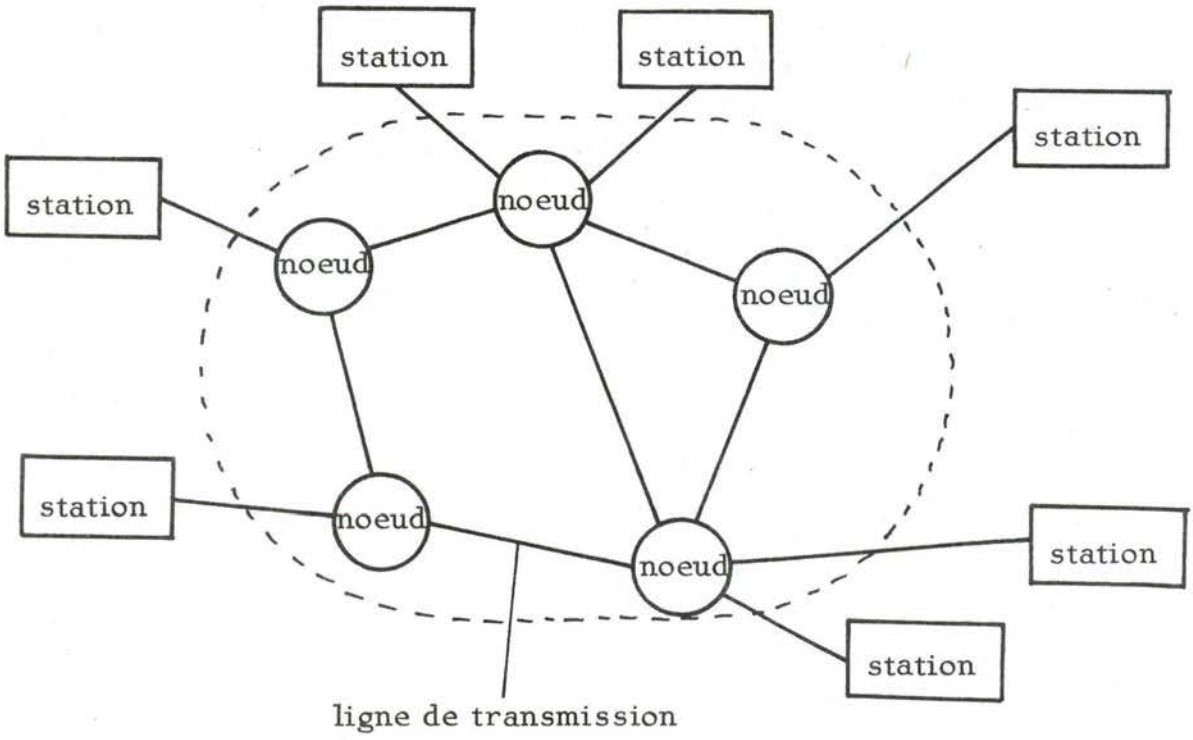


FIG.4

STRUCTURE D'UN RESEAU DE COMMUNICATION

I.1.4. TYPOLOGIE DES RESEAUX DE COMMUNICATION.

La structure des réseaux de communication permet de grouper les réseaux d'ordinateurs par types. Le modèle de Anderson et Jensen (ANDERSON75) est un essai de classification des systèmes distribués. C'est la seule typologie proposée qui englobe la plus grande partie des réseaux d'ordinateurs réalisés. Le modèle est essentiellement développé autour des éléments qui interviennent dans le transfert d'informations. Ces transferts entre les stations se font par des messages (messages), qui sont des unités d'informations transférées entre les processus qui s'exécutent dans des éléments processeurs distincts.

La classification du modèle se base sur 3 éléments hardware:

- stations, qui sont les sources et les destinations des messages
- chemins (paths), qui sont des moyens à travers lesquels les messages sont transférés entre les stations (ce qu'on a appelé ligne de transmission dans I.1.3.)
- commutateurs (switches), qui sont des machines intelligentes pour diriger les messages sur les différents chemins (le commutateur fait partie des noeuds définis sous I.1.3.).

Comme la classification se base sur 3 éléments seulement, des détails qui entrent dans l'implémentation d'une architecture de réseau d'ordinateurs sont perdus. Ainsi il n'y a pas de distinction entre l'élément processeur et l'interface vers le réseau de communication, mais les deux constituent une station. De même les réseaux de communication sont considérés à un niveau tel que les différentes techniques de commutation n'entrent pas en jeu. D'autre part, l'omission des méthodes d'adressage et des interbloquages (deadlocks) entre processus aident à simplifier le modèle.

Avec les 3 éléments hardware définis plus haut, on peut distinguer 4 niveaux de choix avant d'arriver à une architecture particulière d'un système distribué. Les deux premiers niveaux sont des choix de stratégies, et les deux derniers sont des choix d'implémentation. Les 4 choix concernent:

- la stratégie de transfert des messages:
  directe / indirecte
- la méthode de contrôle des transferts:
  centralisée / distribuée
- le type de chemin de transfert:
  dédicacé / partagé
- l'interconnexion des stations:
  anneau / bus / étoile / mémoire / structure régulière / structure irrégulière / connexion complète.

Le premier choix concerne la stratégie de transfert des messages. Le transfert entre deux stations peut se faire de manière directe (direct) sur un chemin, ou indirecte (indirect) en passant par des commutateurs. Dans ce dernier cas , il existe deux méthodes de contrôle des transferts. Le contrôle des transferts peut être centralisé (centralized routing) dans une seule machine, qui dirige tous les messages échangés entre les stations. La seconde méthode appelée distribuée (decentralized routing) fait intervenir plusieures machines pour diriger les messages. Ensuite vient le choix du

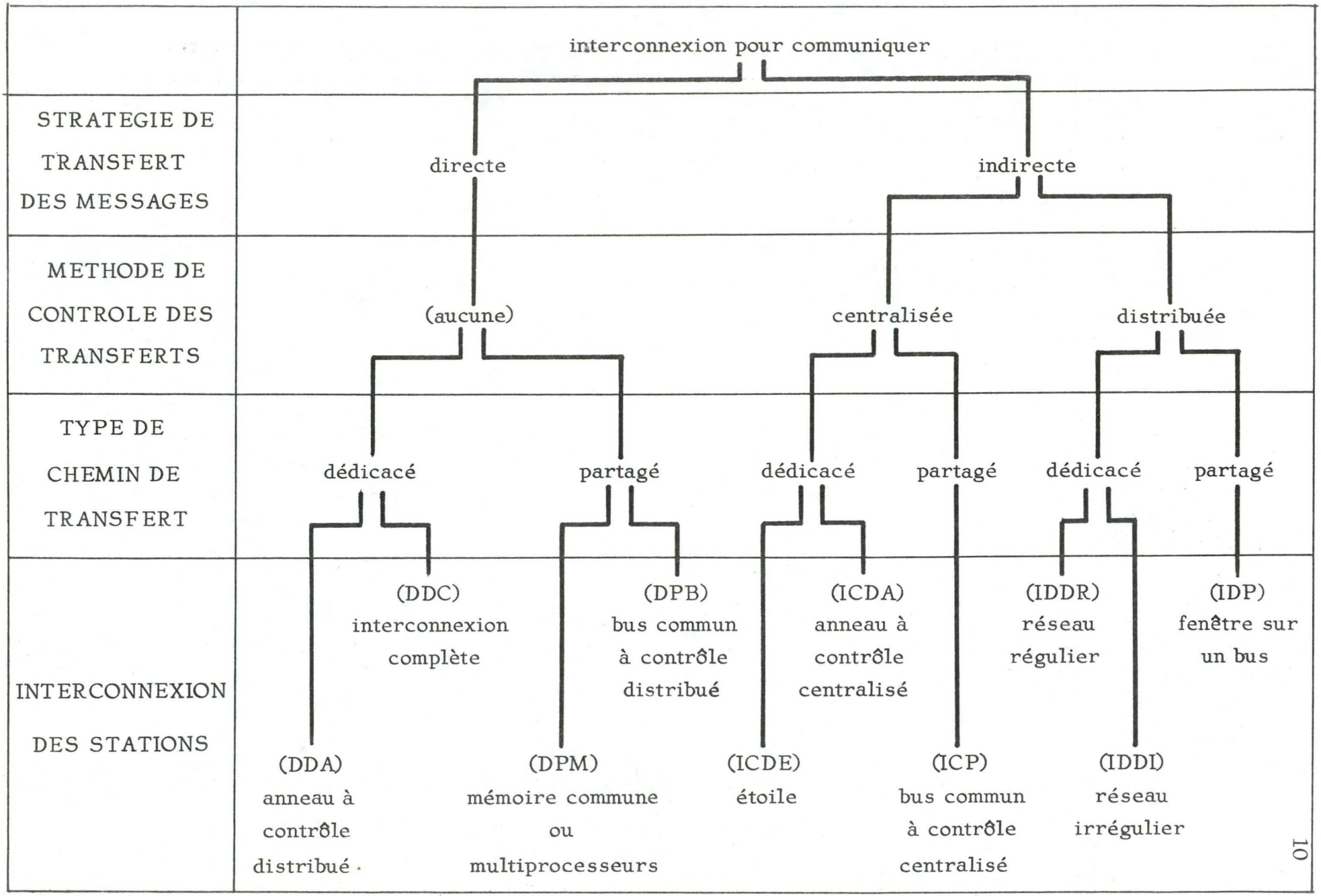
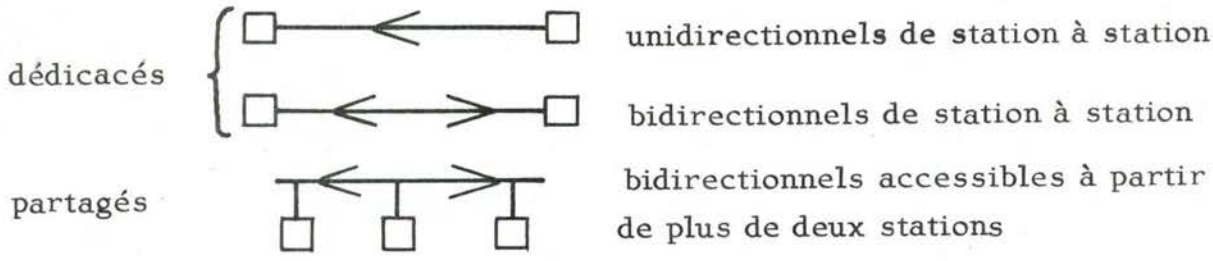


FIG.5 TYPOLOGIE DE ANDERSON ET JENSEN

type de chemin de transfert. En réalité, il existe 3 alternatives:



Les 2 premiers chemins sont regroupés en un type appelé dédicacé (dedicated path), et le troisième est dit de type partagé (shared path). Mais la notion de chemin n'implique pas encore une implémentation fixée. Plusieures possibilités pour l'interconnexion des stations restent ouvertes. Ainsi les interconnexions entre les stations peuvent prendre la forme d'un anneau, d'un bus ou d'une étoile. Elles peuvent être régulières ou irrégulières. Les stations peuvent communiquer par une mémoire commune ou être complétement interconnectées. Tous ces types seront décrits dans le chapitre I.1.6. La FIG.5 prend donc l'allure d'un graphe sous forme d'arbre. Chaque architecture finale est désignée plus simplement par les initiales des termes qui se trouvent aux noeuds de l'arbre parcouru de la racine aux feuilles (p.ex. ICDA désigne une stratégie de transfert Indirecte des messages, une méthode de contrôle Centralisée des transferts, un type de chemin de transfert Dédicacé et une interconnexion en Anneau des stations).

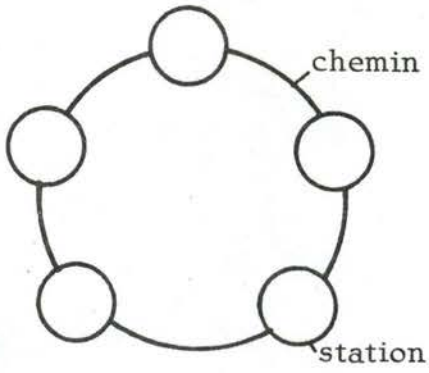
Donc ce modèle souligne des architectures types de réseaux d'ordinateurs. Il permet de clarifier les idées sur les architectures de systèmes distribués, et de se fixer sur des définitions de base nécessaires à une discussion.

I.1.5. PRINCIPALES CARACTERISTIQUES D'UNE ARCHITECTURE D'UN SYSTEME DISTRIBUE.

La modularité est la possibilité de faire des changements incrementals de la capacité du système distribué en rajoutant des éléments au fur et à mesure de leur nécessité. Une telle augmentation de la capacité a des conséquences sur les frais de reconfiguration de l'architecture. Ces conséquences sont conditionnées par la flexibilité des connexions entre les éléments du système distribué.

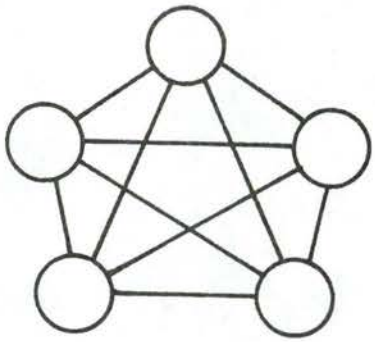
Les effets des pannes d'un système distribué forment une autre caractéristique. Dans les architectures où certains éléments sont partagés entre les stations, une panne de l'un de ces éléments provoque un arrêt complet du réseau. Pour les autres architectures, les conséquences des pannes sont moins catastrophiques.

(a)



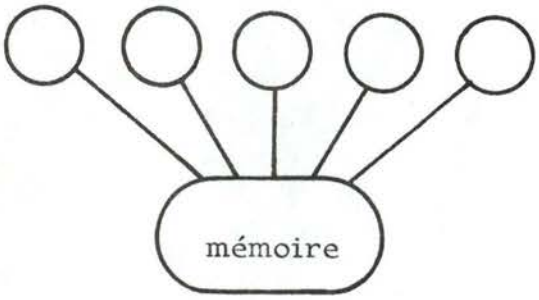
DDA anneau à contrôle distribué

(b)



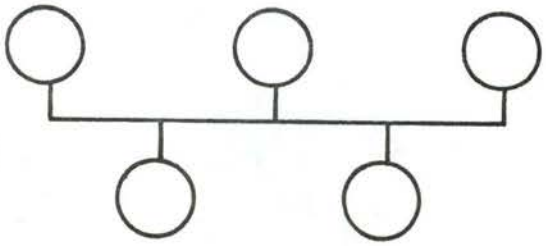
DDC interconnexion complète

(c)



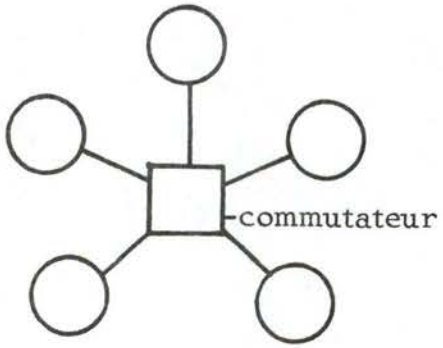
DPM mémoire commune ou multiprocesseurs

(d)



DPB bus commun à contrôle distribué

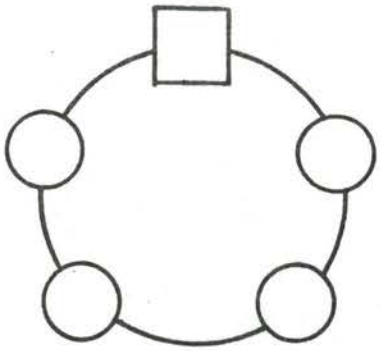
(e)



ICDE étoile

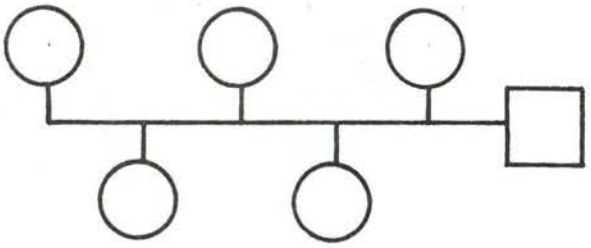
FIG.6 ARCHITECTURES TYPES DE SYSTEMES DISTRIBUES

(f)



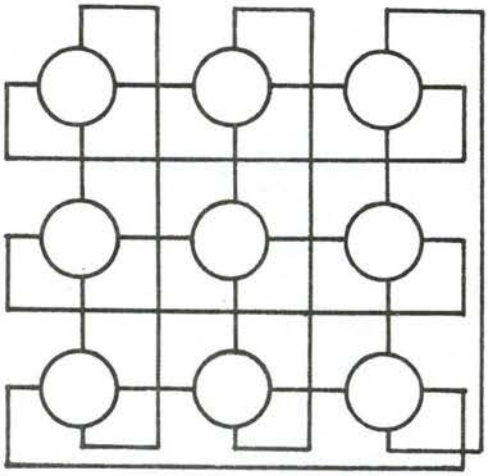
ICDA anneau à contrôle centralisé

(g)



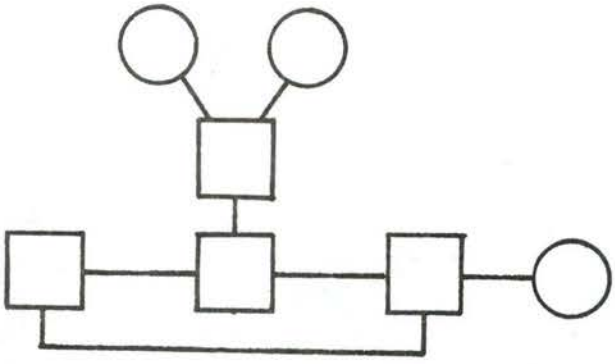
ICP bus commun à contrôle centralisé

(h)



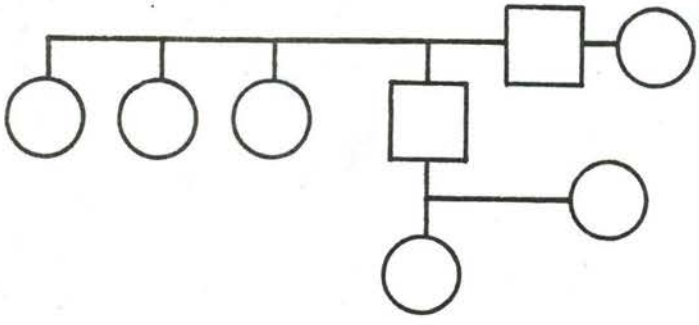
IDDR réseau régulier

(i)



IDDI réseau irrégulier

(j)



IDP fenêtre sur un bus

FIG.6 ARCHITECTURES TYPES DE SYSTEMES DISTRIBUES

La nature et le nombre de décisions à prendre pour effectuer des communications dans un système doivent aussi être considérées. Ils forment la complexité logique du système distribué.

Certains éléments du réseau de communication peuvent se saturer plus rapidement que d'autres. Alors ils n'arrivent plus à transmettre tous les messages à la vitesse maximale nécessaire et constituent alors des goulots d'étranglement pour le flux des messages.

Pour les différentes architectures types, l'analyse sera basée sur les caractéristiques suivantes:

- modularité
- effets des pannes
- complexité logique
- goulots d'étranglement.

I.1.6. ARCHITECTURES TYPES DE SYSTEMES DISTRIBUES.

Les types d'architectures qu'on va décrire dans la suite, sont ceux mis en évidence par le modèle de Anderson et Jensen, et qui se trouvent sur la FIG.5. Leurs structures sont détaillés sur la FIG.6.

I.1.6.1. ANNEAU A CONTROLE DISTRIBUE - DDA - FIG.6a .
(distributed control loop)

Un anneau consiste en un nombre de stations dont chacune est reliée à deux stations voisines. En théorie, les messages pourraient se propager dans les deux directions autour de l'anneau, mais en pratique cela s'avère trop complexe. Tous les systèmes distribués en anneau connus sont donc à sens unique. Les messages circulent dans l'anneau de la station émettrice vers la station réceptrice, en traversant des stations intermédiaires non concernées, et qui agissent comme des répétiteurs (c'est-à-dire, lisent le message sur la ligne incidente et le répètent sur la ligne partante).

Dans un anneau, les coûts sont très modulaires. La rajoute d'une station peut se faire à l'aide d'un seul chemin de communication. Par contre, les effets d'une panne sont assez désastreux. Un seul élément fautif dans l'anneau empèche toute communication. La logique des communications dans un anneau n'est pas très complexe. Une station doit:

- recevoir des messages
- répéter des messages

- créer des messages
- reconnaitre les messages qui lui sont destinés.

Les chemins entre les stations sont des goulots d'étranglement possibles lorsque le nombre de communications augmente.

I.1.6.2. INTERCONNEXION COMPLETE - DDC - FIG.6b .
(complete interconnection)

Chaque station est reliée par une ligne de transmission distincte à chaque autre station. Les messages sont échangés entre deux stations uniquement par la ligne qui les relie. Toutes les stations doivent être capables d'accepter simultanément plusieurs messages venant de plusieures autres stations.

La grande caractéristique de ce type d'architecture est la pauvre modularité des coûts. La rajoute d'une $N^{ième}$ station est assez coûteuse, car il ne faut pas seulement ajouter N-1 chemins, mais il faut aussi que le software de toutes les stations reconnaisse cette nouvelle station. L'effet d'une panne n'est pas grave, car les éléments fautifs peuvent simplement être déconnectés du système. La structure des interconnexions diminue la complexité logique des transmissions. Cette architecture ne comporte pas de goulots d'étranglement.

I.1.6.3. MEMOIRE COMMUNE OU MULTIPROCESSEURS - DPM - FIG.6c .
(common memory)

Plusieures stations sont reliées par des chemins distincts à une mémoire commune. Deux ou plusieures stations communiquent en laissant des messages dans la mémoire commune. Celle-ci est considérée comme un chemin.

Le coût d'une expansion du système est modulaire. La nouvelle station (ou processeur dans ce cas ci) doit se connecter seulement à la mémoire commune. Une panne d'une station n'est pas grave, par contre une panne de la mémoire commune entraîne un arrêt du système entier. La complexité de la logique des transmissions est assez faible dans un tel système, car chaque station a un accès directe à la mémoire commune. Une mémoire sousdimensionnée peut constituer un goulot d'étranglement pour la transmission des messages.

I.1.6.4. BUS COMMUN A CONTROLE DISTRIBUE - DPB - FIG.6d .
(common distributed control bus)

Les stations sont reliées par un seul chemin appelé bus . L'accès à ce bus est contrôlé par un schéma d'allocation. La station émettrice met alors

son message sur le bus, et toutes les stations peuvent le recevoir. La ou les stations concernées doivent reconnaitre que le message leur est destiné.

La modularité du coût est bonne pour les stations, car elles peuvent se connecter n'importe où sur le bus. Les caractéristiques de l'effet d'une panne sont très bonnes ou très mauvaises, d'après la cause de la panne. Une station en panne peut se déconnecter plus ou moins automatiquement du bus sans en déranger le fonctionnement. Par contre, une panne du chemin de transmission entraîne inévitablement l'arrêt de tout le système distribué, et peut endommager toutes les stations. La complexité logique des transmissions est moyenne, et se réduit à l'acquisition du bus et à la reconnaissance des destinations des messages. Le bus lui-même est le seul goulot d'étranglement possible.

I.1.6.5. ETOILE - ICDE - FIG.6e .
(star)

Toutes les stations sont reliées physiquement à un commutateur central. Les stations échangent des messages en passant par le commutateur central. Donc pour chaque station, celui-ci est la source et la destination apparente des messages. La fonction du commutateur est de dispenser un processus de l'obligation de connaitre la structure physique du système distribué. Comme dans le cas des multiprocesseurs et du bus, l'étoile considère le commutateur central comme un moyen de transport partagé entre les stations.

La modularité du coût est bonne pour les stations, et mauvaise pour les chemins de transmission. Chaque nouvelle station se connecte directement au commutateur central. Les conséquences d'une panne ne sont pas graves pour les stations, car elles n'éliminent que la station fautive. Mais une panne du commutateur central bloque tout le système. La complexité logique des communications est liée à ce qui se passe dans le commutateur. Celui-ci doit disposer des informations nécessaires pour diriger les messages. Le commutateur central est un goulot d'étranglement.

I.1.6.6. ANNEAU A CONTROLE CENTRALISE - ICDA - FIG.6f .
(centrally controlled loop)

D'abord, par un message spécial, la station source demande au commutateur d'établir une liaison logique avec une station destination. Puis la station source met son message sur l'anneau. Le commutateur retire ce message, remplace l'adresse source par l'adresse destination correspondante et remet le message sur l'anneau. Finalement la station destination doit reconnaitre les messages qui lui sont destinés.

Cette structure est un mélange d'anneau et d'étoile et a donc leurs caractéristiques. Le coût de la rajoute d'une station est modulaire. Cette nouvelle station doit se connecter seulement à ses deux stations les plus voisines. Comme dans toutes les structures en anneau, la panne d'un élément est assez grave, car elle bloque tout le système. Si une station fautive peut encore se déconnecter, le commutateur par contre est essentiel, et en cas de panne, le fonctionnement du réseau sera bloqué. Pour ce qui est de la complexité logique, on retrouve un mélange des caractéristiques des anneaux et des étoiles. Comme pour une étoile, le commutateur doit faire des transformations d'adresse dans les messages, et en plus, les stations doivent jouer le rôle de répétiteurs comme dans un anneau. Le goulot d'étranglement possible du système est le commutateur central.

I.1.6.7. BUS COMMUN A CONTROLE CENTRALISE - ICP - FIG.6g .
(common centrally controlled bus)

Comme pour les deux architectures précédentes, il y a les mêmes composants, mais arrangés autrement. Si une station veut envoyer un message vers une autre, elle doit d'abord acquérir le bus, et puis transmettre son message au commutateur central. Celui-ci va retransmettre ce message vers la station réceptrice, après avoir fait une transformation d'adresse.

Les caractéristiques de cette architecture sont très proches de celles du réseau en étoile. La modularité du coût est bonne, car une nouvelle station doit se connecter seulement sur le bus. En cas de panne d'une station, celle-ci peut se déconnecter, mais cela n'est pas possible pour le commutateur central. La complexité logique résulte du fait que le commutateur doit faire des transformations dans les messages, et que les stations doivent reconnaitre les messages qui leurs sont adressés. Le bus partagé n'introduit pas un goulot d'étranglement, pour autant qu'il ne se sature pas avant le commutateur.

I.1.6.8. RESEAU REGULIER - IDDR - FIG.6h .
(regular network)

Les stations sont reliées par des chemins dédicacés de façon à ce que chacune ait les mêmes relations de voisinage. Ainsi sur la FIG.6h , chaque station a des stations voisines à gauche, à droite, en dessus et en dessous. D'autres géométries d'interconnexion sont possibles, comme des structures

d'arbre par exemple. Les messages se propagent dans le réseau de station en station. Chacune décide vers quelle station suivante diriger le message qui vient d'arriver.

Les caractéristiques de modularité du coût d'une nouvelle station sont très mauvaises à cause des relations de voisinage. De même, celles de l'effet d'une panne sont très négatives dans un tel réseau. Cela est dû à la régularité des interconnexions. En cas de panne, la structure des interconnexions se dégrade, et le réseau n'est plus régulier. La complexité logique des communications peut varier selon l'algorithme intervenant dans la décision d'une route. Un réseau régulier n'a pas de goulot d'étranglement .

I.1.6.9. RESEAU IRREGULIER - IDDI - FIG.6i .
(irregular network)

La différence entre un réseau régulier et un réseau irrégulier est le fait que dans ce dernier, il n'est pas nécessaire de respecter une relation de voisinage. Donc une station peut être reliée directement à une ou plusieures autres.

Les caractéristiques d'un tel réseau varient beaucoup avec le degré de régularité des interconnexions. Ces réseaux irréguliers présentent une bonne modularité du coût parce qu'il n'y a pas besoin de respecter des relations de voisinage. Plus le schéma des interconnexions est irrégulier, plus les caractéristiques de l'effet d'une panne sont améliorées. Par contre, la logique des communications est assez complexe. Cette complexité est due au nombre variable de chemins qui mènent de la source vers la destination. Donc chaque station doit décider de la route à suivre en fonction du schéma complet des interconnexions dans tout le réseau. Pour un réseau irrégulier, il n'y a pas de problème de goulot d'étranglement.

I.1.6.10. FENETRE SUR UN BUS - IDP - FIG.6j .
(bus window)

Dans un tel système, l'accès aux commutateurs se fait à travers un bus partagé entre plusieures stations. Les messages passent seulement par un commutateur lorsqu'ils sont adressés à une station qui se trouve sur une autre branche que la source. Dans d'autres implémentations, les messages passent toujours par un commutateur, et ils peuvent être retransmis sur le même chemin, ou sur une autre branche. La FIG.6j montre une telle architecture.

Ces systèmes sont très modulaires. Une nouvelle station peut facilement se connecter sur n'importe quelle branche du bus. Une panne de la ligne de transmission peut affecter au maximum une branche entière avec ses stations. Si une seule station tombe en panne, elle peut se déconnecter de son bus. La complexité logique des communications est celle des bus communs avec un contrôle distribué sur une branche, mais augmente avec le nombre de commutateurs qu'un message doit parcourir entre la source et la destination. Les commutateurs peuvent représenter des goulots d'étranglement.

I.1.7. CRITIQUES DU MODELE DE TYPOLOGIE.

Dans son rapport (ANDERSON75), Anderson indique lui-même que sa classification n'est pas parfaite, et qu'elle est susceptible d'être améliorée. Des critiques et des essais d'améliorations par d'autres chercheurs ont été formulés dans une publication (COMPUTER77).

Voici quelques-unes des critiques qui ont été le plus souvent adressées au modèle de Anderson et Jensen:

- il est seulement orienté vers la communication par messages
- il ne fait pas de distinction entre les différentes techniques de commutation
- il décrit les niveaux rudimentaires de communication et leurs implémentations pratiques, mais en dessous de chaque niveau, beaucoup de techniques sont possibles pour réaliser un système distribué
- il ne traite pas les problèmes de communication entre les processus, c'est-à-dire de l'adressage des messages
- il ne tient pas compte des systèmes hybrides formés de diverses architectures types de système distribué.

Dans son dernier rapport (THURBER78), Thurber a essayé de faire une analyse plus profonde des architectures de réseaux d'ordinateurs. Cette analyse se base sur un modèle hiérarchique (voir chapitre I.1.2.) du software des communications et sur les différentes techniques de commutation.

I.1.8. CLASSIFICATION DE RESEAUX CONNUS.

Des réalisations de réseaux d'ordinateurs existent pour chaque architecture type du modèle de Anderson et Jensen.

Voici quelques réseaux connus qui rentrent dans cette classification.

System Network Architecture (SNA) est une étoile (ICDE).

ARPANET de Advanced Research Projects Agency est un réseau irrégulier (à commutation par paquets distribuée) (IDDI).

ALOHANET de l'université de Hawaii est un bus commun à contrôle centralisé (ICP).

ETHERNET de Xerox, et COBUS du Laboratoire de Calculatrices Digitales à l'Ecole Polytechnique Fédérale de Lausanne sont des bus communs à contrôle distribué (DPB).

I.1.9. TECHNIQUES DE COMMUTATION.

Dans le chapitre I.1.6. , on a beaucoup parlé de commutateurs, mais sans jamais s'occuper des techniques mises en oeuvre pour le faire.
Il existe deux méthodes de commutation:

- déterministe: entre 2 stations, on prend toujours la même route
- adaptative: entre 2 stations, la route future à suivre est décidée dans chaque station intermédiaire, selon le besoin ou la disponibilité.

La méthode déterministe convient plutôt à un contrôle commutatif centralisé en une seule station, tandis que pour une méthode adaptative, il faut un contrôle de la commutation du chemin suivant dans chaque station.

Un circuit ou route (circuit) est l'ensemble des chemins physiques sur lesquels on transmet l'information entre les stations émettrices et réceptrices.
Un canal (channel) est le chemin logique sur lequel se font les connexions entre les processus utilisateurs. Un circuit peut être multiplexé pour supporter plusieurs canaux.

Il y a 3 techniques de commutation principales, qui sont utilisées dans les réseaux d'ordinateurs.

La commutation par circuit (circuit switching) sert à établir un circuit unique entre deux stations. Si une station A veut entrer en liaison avec une station B, alors les commutateurs relient physiquement les chemins entre les deux stations. Une telle liaison restera pendant tout le temps de la transmission du message. Une fois le circuit établi, il existe une liaison directe entre les deux stations A et B, et il n'y a donc pas de stockage intermédiaire (store and forward) dans les commutateurs. Plusieures transmissions peuvent se faire en même temps entre des stations différentes.

La commutation par message (message switching) s'applique lorsqu'il y a des liaisons physiques permanentes entre les stations. Dans cette méthode les commutateurs établissent un canal logique entre les deux stations qui veulent communiquer. Le choix de ce canal parmi tous les circuits disponibles peut se faire de manière centralisée ou distribuée. Cette technique permet d'améliorer l'utilisation des circuits disponibles. Les stations de commutation font un stockage intermédiaire des messages avant de les transmettre vers le commutateur suivant.

La commutation par paquets (packet switching) est presque la même technique que la commutation par messages, sauf que le message est divisé en paquets . Un paquet est un morceau du message, auquel est ajouté l'information nécessaire pour reconstituer le message initial (p. ex. numéro d'ordre du paquet dans le message). La différence avec la commutation par messages est que les différents paquets d'un seul message peuvent passer par des circuits différents. D'où la nécessité de pouvoir reconstituer le message initial à la destination, et cela même si les paquets sont arrivés dans un ordre différent. Cette méthode adaptative est utilisée dans le réseau canadien DATAPAC. La commutation par paquets peut se faire aussi selon la méthode déterministe. Dans la méthode faiblement déterministe, utilisée dans le réseau TRANSPAC, le premier paquet d'un message cherche un circuit à travers le réseau, et les paquets suivants du message passent par le même circuit. Dans la méthode fortement déterministe, tous les paquets échangés entre deux stations passent toujours par le même circuit.

La commutation par paquets optimise le taux d'utilisation des circuits.

I.2. INTRODUCTION AUX RESEAUX LOCAUX.

I.2.1. DEFINITION D'UN RESEAU LOCAL.

Les réseaux locaux ou systèmes distribués locaux (local computer networks) forment un sousensemble des réseaux d'ordinateurs. En première approche, ils peuvent être définis comme suit:

> le réseau local est un réseau qui permet des transmissions de données à grande vitesse entre des utilisateurs qui se trouvent à des distances de l'ordre du kilomètre.

Cette définition sera raffinée plus tard par l'explication des caractéristiques spécifiques des réseaux locaux.

I.2.2. DISTANCES DES RESEAUX D'ORDINATEURS.

Selon la distance qui sépare les stations les plus éloignées connectées sur le réseau d'ordinateurs, celui-ci se divise en 3 sousensembles:

- les réseaux d'ordinateurs longue distance (long-haul networks), qui relient des stations sur des distances variant de quelques centaines de mètres à des milliers de kilomètres
- les réseaux locaux (local area networks) couvrent des distances de quelques mètres à plusieurs kilomètres
- les bus des architectures internes des ordinateurs (computer system busses) vont de quelques centimètres pour les systèmes à microprocesseur, jusqu'à une centaine de mètres pour les systèmes multiprocesseurs.

La FIG.7 indique les distances englobées par les 3 sortes de systèmes distribués. Les réseaux locaux se superposent au bas de gamme des réseaux longue distance, et au haut de gamme des bus d'ordinateur.

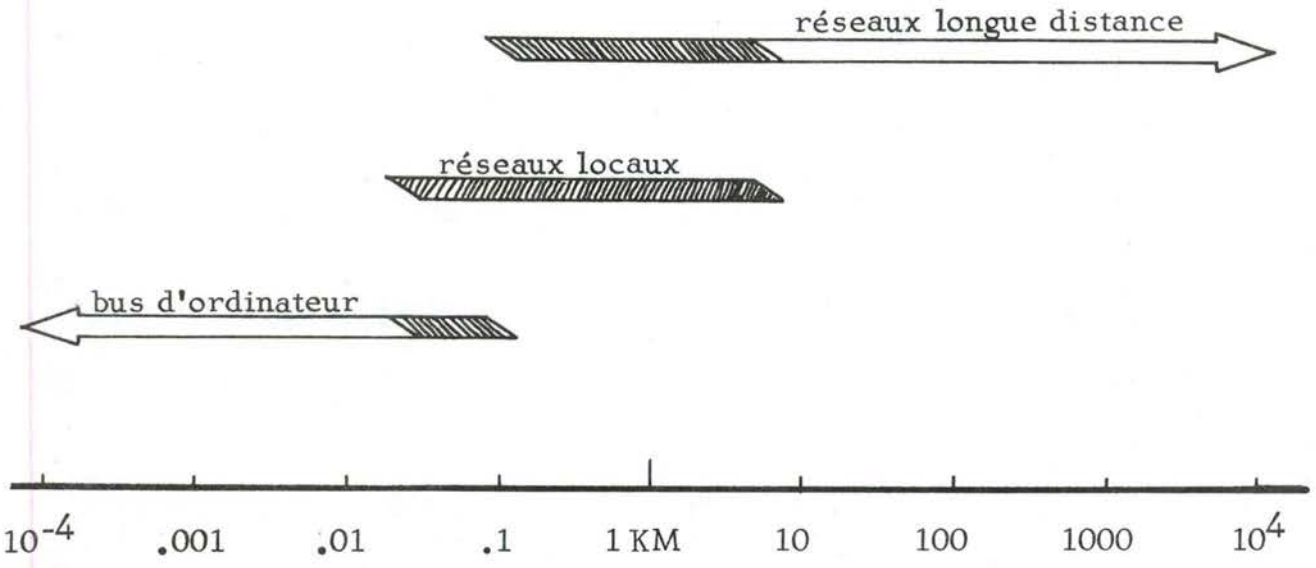


FIG.7

DISTANCES DES RESEAUX D'ORDINATEURS

I.2.3. EVOLUTION DES RESEAUX LOCAUX.

Les réseaux longue distance ont été les premiers à être commercialisés. Leur technique de transmission des données se base sur celle des réseaux téléphoniques adaptés à l'aide de modems. Comme c'était la seule technique disponible à ce moment là, elle était utilisée pour relier des ordinateurs sur n'importe quelle distance. Les vitesses de transmission de ces réseaux longue distance vont de quelques centaines à quelques milliers de bits par seconde (FIG.8a).

Une fois les réseaux longue distance installés, ils ont contribué à l'évolution des méthodes de communication. Le réseau ARPANET a permis la mise au point de la communication par paquets qui optimise l'utilisation des lignes de transmission. L'évolution technologique permet maintenant de mettre des signaux digitaux directement sur les lignes de transmission. Ces nouvelles techniques permettent des transmissions à grande vitesse (1 Mégabit / seconde) sur de petites distances (100 mètres) (FIG.8b).

Peu après leur commercialisation, les techniques des réseaux locaux ont seulement été utilisées sur de petites distances. L'évolution de la technologie et la diminution du coût des matériaux ont provoquées l'implémentation des réseaux locaux sur des distances de plus en plus grandes (FIG.8c). Il y a 2 raisons pour le déplacement de la frontière entre les

réseaux locaux et les réseaux longue distance:

- les réseaux locaux sont moins chers que les réseaux longue distance jusqu'à une distance donnée (qui est la distance maximale permise par les techniques des réseaux locaux)
- les réseaux locaux permettent des vitesses de transmission beaucoup plus élevées que les réseaux longue distance.

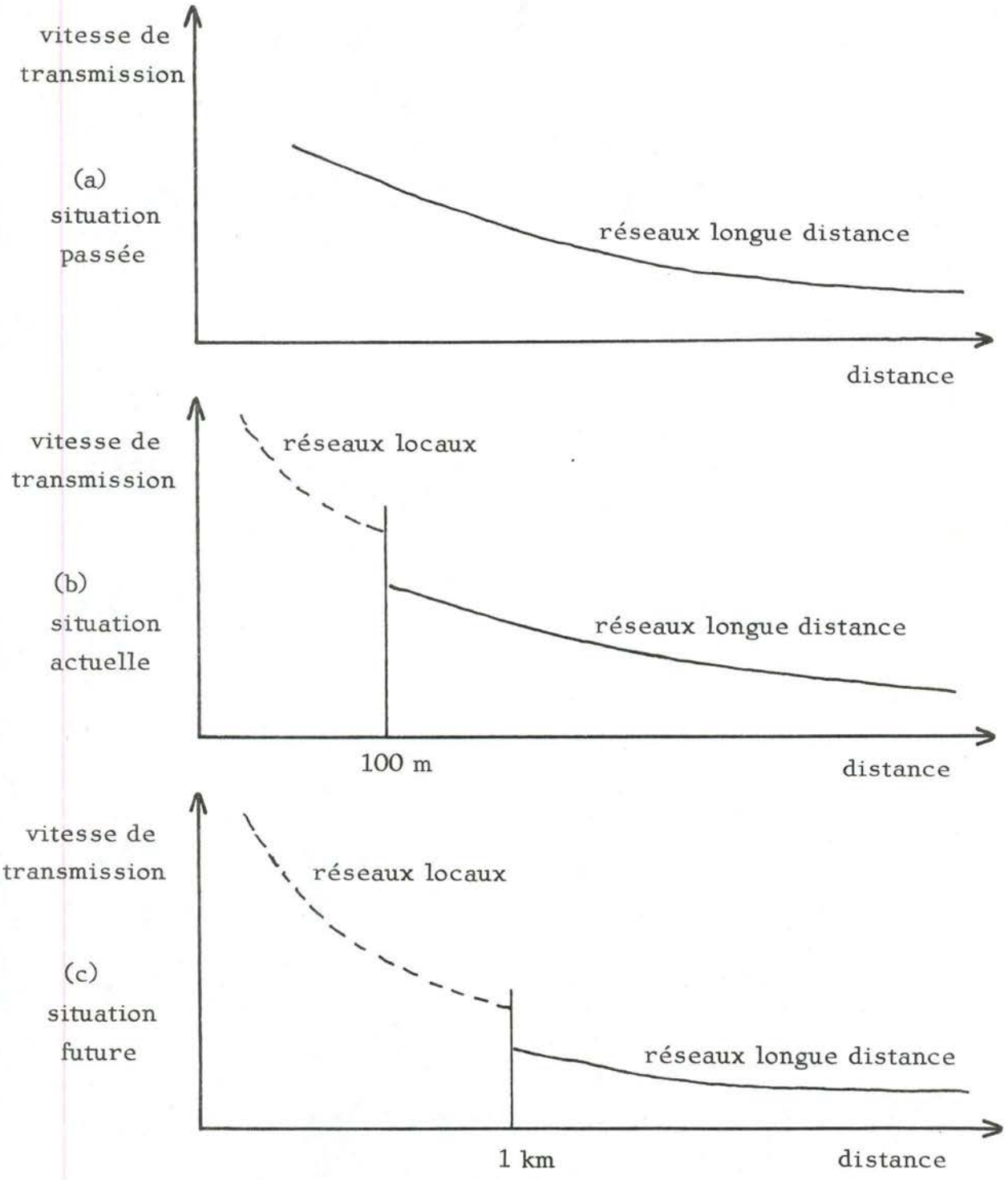


FIG.8

EVOLUTION DES RESEAUX LOCAUX

I.2.4. CARACTERISTIQUES DES RESEAUX LOCAUX.

Les réseaux locaux ont été commercialisés relativement tard par rapport aux réseaux longue distance.

Les techniques des réseaux locaux utilisent des circuits logiques hautement intégrés (Large Scale Integration, LSI) comme des microprocesseurs. Les cables coaxiaux qui sont utilisés pour l'interconnexion des stations permettent des vitesses de transmission très élevées. Comme les techniques des réseaux locaux mettent les signaux digitaux venant des circuits logiques directement sur les cables coaxiaux, sans passer par des dispositifs spéciaux de modulation, les signaux ne peuvent se propager que sur des distances relativement faibles. Donc les réseaux locaux peuvent seulement couvrir des distances allant jusqu'à 1 kilomètre.

Le coût du matériel électronique est faible dans une implémentation d'un réseau local. Le cable coaxial coûte plus cher que les cables téléphoniques des réseaux longue distance. Mais à cause des petites distances, le coût des cables n'intervient pratiquement pas dans le coût global de l'implémentation d'un réseau local.

Il ne faut pourtant pas considérer les réseaux locaux comme des solutions universelles pour tous les problèmes de communication locale entre ordinateurs. Ils peuvent aider à diminuer le coût des communications qui nécessitent une grande largeur de bande (bandwidth). Par contre, le coût du software est très élevé vis-à-vis du hardware. La nouvelle technique nécessite une nouvelle programmation qui tient compte des nouvelles possibilités offertes. C'est donc le prix du développement software qui prime dans une mise en oeuvre d'un réseau qui utilise la technologie des réseaux locaux.

I.2.5. DIFFERENCES ENTRE RESEAUX LOCAUX ET RESEAUX LONGUE DISTANCE.

Voici un résumé des différences entre les réseaux locaux et les réseaux longue distance déjà mentionnées jusqu'ici dans la section I.2 .

Les réseaux longue distance ont été commercialisés longtemps avant les réseaux locaux, qui sont assez récents. Comme leurs noms l'indiquent, les réseaux longue distance peuvent couvrir des distances beaucoup plus grandes que les réseaux locaux. Dans les systèmes distribués locaux, on retrouve les mêmes composants que dans les réseaux longue distance. Les techniques utilisées dans les réseaux locaux permettent de construire des réseaux capables de véhiculer des informations à grande vitesse, pendant que le

coût de la transmission reste assez bas. Tandis que les réseaux longue distance traditionnels transmettent les informations plus lentement, et cela pour un coût nettement plus élevé.

I.2.6. DIFFERENCES ENTRE RESEAUX LOCAUX ET BUS D'ORDINATEUR.

Les bus d'ordinateur ont fait leur apparition avant les réseaux locaux, et après les réseaux longue distance.

Les distances sur lesquelles on utilise les bus d'ordinateur sont très petites. Les techniques des réseaux locaux sont très proches de celles des bus d'ordinateur. Les vitesses de transmission sur les bus d'ordinateur sont très élevées, car elles sont du même ordre de grandeur que les vitesses de propagation des signaux électroniques entre les circuits logiques. Le coût de réalisation est minime à coté de celui de la conception d'un bus d'ordinateur.

Mais le bus d'ordinateur ne peut quand même pas être considéré comme un réseau . Il y a d'autres différences qui sont plus marquantes.

Un bus d'ordinateur est conçu pour relier les différents composants d'un ordinateur, comme processeur, mémoire et canaux d'entrée/sortie. Aucun de ces composants ne pourrait effectuer une action utile sans passer par le bus. Donc ces composants ne peuvent pas fonctionner de manière autonome. Tandis que le réseau local a été construit en vue d'interconnecter des stations qui au départ travaillent indépendamment les unes des autres. Donc chaque station peut effectuer des opérations même en l'absence de réseau local. Généralement un bus d'ordinateur est constitué de lignes (fils) parallèles, et un réseau local transmet les informations en série sur une seule ligne (2fils forment un cable). Les bus d'ordinateur restent à l'intérieur d'un ordinateur, tandis que le réseau relie les différents ordinateurs extérieurement.

I.3. ANALYSE DETAILLEE DES RESEAUX LOCAUX.

I.3.1. MOYENS DE TRANSMISSION.

La plus grande partie des réseaux locaux actuellement réalisés ont choisi une transmission en série. Cela se fait alors sur une paire de cables torsadés ou un cable coaxial. Technologiquement, c'est le cable coaxial qui se prête le mieux pour des transmissions à grande vitesse, et sur des distances assez longues. Dans un cable coaxial, un fil entoure l'autre, ce qui donne une bonne protection contre les bruits parasites de l'environnement. Son désavantage mineur réside dans le fait qu'il faut prévoir des connecteurs spéciaux pour relier les stations sur le cable. Ceci est plus facile pour une paire de fils torsadés, où les deux sont directement accessibles. La technologie des transmissions de télévision par cable (CAble TeleVision, CATV) peut s'adapter aux transmissions entre ordinateurs. Cette technique met à disposition une très grande largeur de bande, et un réseau local pourrait utiliser un ou plusieurs des canaux disponibles (bandes de fréquences). Comme la télévision par cable s'installe de plus en plus fréquemment, on trouve un réseau de communication déjà installé sur place.

Les fibres optiques semblent aussi convenir pour la transmission de données. Celles-ci permettent des vitesses de transmission de 1 à 20 mégabits par seconde sur des distances de plusieurs kilomètres. Un autre avantage de cette technologie est son insensibilité aux bruits électromagnétiques. Jusqu'ici, la technologie a seulement réussi à fabriquer des fibres optiques capables de transmettre dans une seule direction. Cette particularité élimine certaines architectures possibles pour les réseaux. Aujourd'hui, les fibres optiques sont encore très chères.

En général, les caractéristiques d'un moyen de transmission pour un réseau local devraient être:

- sécurité
- simplicité
- coût faible
- grande largeur de bande
- insensibilité au bruit électromagnétique

I.3.2. TOPOLOGIES DES RESEAUX LOCAUX.

A partir des architectures mises en évidence par le modèle de Anderson et Jensen (chapitre I.1.4.), il faut choisir celles qui s'adaptent le mieux aux exigences des réseaux locaux. D'après les caractéristiques des réseaux locaux (chapitre I.2.4.), plusieures architectures peuvent être éliminées.

Les multiprocesseurs n'arrivent pas à couvrir des distances assez grandes pour être considérés comme des réseaux locaux à part entière. Les vitesses de transmission sont réduites par les décisions de routage dans les stations et commutateurs des architectures suivantes: interconnexion complète, réseau régulier et réseau irrégulier. La structure des fenêtres sur un bus est inutilement complexe et coûteuse à implémenter. Après élimination de ces topologies, il reste alors les architectures suivantes, valables pour les réseaux locaux:

- étoile
- anneau à contrôle centralisé
- anneau à contrôle distribué
- bus commun à contrôle centralisé
- bus commun à contrôle distribué.

La topologie d'un réseau est la structure géométrique utilisée pour interconnecter toutes les stations et les commutateurs du réseau. D'après cette définition, les topologies sous-jacentes aux 5 architectures pour les réseaux locaux sont:

- étoile
- anneau
- bus.

Une topologie en étoile est acceptable pour un réseau local, si la nature des communications s'y adapte bien, c'est-à-dire, si de petites stations secondaires communiquent seulement (ou principalement) avec une importante station centrale. La vitesse de transmission dans un anneau est un peu plus lente parce que les informations doivent traverser plusieures stations. Sur un bus, les transmissions peuvent se faire à vitesse maximale car il y a toujours une liaison directe entre les stations source et destination.

Les 4 réseaux locaux les plus connus ont des topologies d'anneau ou de bus, avec des contrôles centralisés ou distribués.

Le réseau MITRIX (WILLARD73) de la firme américaine MITRE est un bus à contrôle centralisé.

Le réseau ETHERNET (METCALFE75) de la firme américaine XEROX est un bus à contrôle distribué.

Le réseau SPIDER (FRASER75) de la firme américaine BELL TELEPHONE est un anneau à contrôle centralisé.

Le réseau Distributed Computing System (DCS) (FARBER75) de l'UNIVERSITY CALIFORNIA IRVINE est un anneau à contrôle distribué.

Le choix d'une topologie pour un système distribué local dépend de sa simplicité, de sa sécurité et de la stratégie de contrôle qui lui sera associée. Mais il y a aussi un autre critère tout aussi important, c'est-à-dire la méthode d'adressage qui sera utilisée dans son système distribué. Comme méthodes possibles il y a:

- point-à-point (point-to-point)
  une station ne s'adresse toujours qu'à une seule autre station en même temps
- diffusion (broadcasting)
  une station peut communiquer avec un ensemble d'autres stations en même temps.

La topologie d'anneau est capable de supporter facilement et indifféremment les 2 méthodes d'adressage. Pour le point-à-point, le message passe de la source à la destination, qui enlève le message de l'anneau. Dans le cas de la diffusion, le message fait un tour complet autour de l'anneau, et est enlevé par sa station source. Un bus est particulièrement adapté à la diffusion de messages, mais les transmissions point-à-point sont aussi possibles. La structure d'étoile est plutôt destinée au point-à-point.

I.3.3. MECANISMES DE CONTROLE DU RESEAU DE COMMUNICATION.

Dans tout réseau viendra sûrement le moment où deux ou plus de stations veulent émettre au même instant. Alors toutes ces stations demandent accès au réseau de communication en même temps. Les mécanismes de contrôle vont alors déterminer quelle station pourra accèder au réseau de communication à quel moment.

Voici quelques mécanismes de contrôle qui peuvent être groupés dans 2 grandes catégories, qui se retrouvent dans le modèle de Anderson et Jensen: contrôle centralisé et contrôle distribué.

I.3.3.1. CONTROLE CENTRALISE.

Pour l'implémentation d'un contrôle des transmissions, divers degrés de centralisation sont possibles. Ainsi le rôle de la station qui tient le contrôle peut se limiter au démarrage du mécanisme, ou bien elle garde un contrôle continu sur toutes les transmissions. Les mécanismes décrits dans cette division sont souvent utilisés de manière centralisée dans les réseaux locaux. Cette centralisation simplifie la complexité du contrôle.

Toutes les solutions présentées ici se basent sur l'idée suivante: une permission pour émettre circule de station en station sous le contrôle plus ou moins continu d'une station particulière.

I.3.3.1.1. Sondage (polling).

A tour de rôle chaque station reçoit la permission d'émettre sur le réseau de communication. Cette permission est accordée par une station particulière, qui joue le rôle d'un commutateur. Chaque fois qu'une station a reçue la permission, elle doit répondre à la station particulière, si oui ou non elle a un message à émettre.

I.3.3.1.2. Chaînage d'un signal de contrôle (daisy-chain).

Sur un fil dédicacé à cette fonction, un signal digital passe de station en station. Si la station désire émettre un message sur le réseau de communication, elle bloque ce signal, c'est-à-dire qu'elle ne le retransmet pas vers la station suivante. Au lieu de cela, elle émet son message sur le réseau. C'est seulement lorsque le message sera arrivé à la station réceptrice, que la station émettrice libère le signal de permission, et le transmet vers la station suivante. Ce signal de permission d'émettre continue indéfiniment à circuler, et peut très bien être contrôlé par un dispositif hardware. Dans ces conditions, une panne de ce dispositif est très grave, car elle bloque tout le réseau.

I.3.3.1.3. Roulement d'enveloppes (message slots).

Dans cette méthode, ce n'est plus seulement un signal qui est envoyé à travers le réseau de communication mais des enveloppes de messages entières avec les bits de contrôle nécessaires. Ces enveloppes doivent être assez grandes pour contenir un message entier. Si une telle enveloppe arrive à une station, celle-ci doit contrôler si l'enveloppe est libre ou non. Pour une enveloppe occupée, la station regarde si le message lui est adressé, alors elle l'enlève de l'enveloppe et libère celle-ci, et puis la transmet à la station suivante. Si le message n'est pas pour la station,

elle laisse passer l'enveloppe directement à la station suivante. Au cas où une station veut émettre un message, elle attend une enveloppe libre, y met son message avec la destination, marque que cette enveloppe est occupée, et la transmet à la station suivante. Cette méthode ne peut jamais être totalement décentralisée, car il faut qu'une station génère les enveloppes libres.

I.3.3.2. CONTROLE DISTRIBUE.

Pour les mécanismes de contrôle distribué, il n'y a pas besoin de station particulière qui fasse démarrer la circulation d'un signal. Toutes les stations se disputent l'accès au réseau de communication. Le mécanisme présenté ici est de plus en plus utilisé. Il trouve beaucoup d'applications avec la topologie de bus, à laquelle il s'adapte le mieux.

Le contrôle d'un bus par contention (contention control) est très simple. Chaque station qui veut émettre un message, le met directement sur le bus. Si plusieures stations émettent alors en même temps, il y a collision sur le bus, et les messages sont détruits et perdus. Le contrôle par contention repose sur le fait que chaque station est capable de détecter une collision, et les destructions des messages. Si une station détecte une collision, et qu'elle est en train d'émettre, elle arrête son émission, attend pendant un temps aléatoirement long et puis retransmet son message en entier. L'occupation du bus est généralement telle que les collisions soient assez rares. Cela provient du fait que dans un réseau local, les lignes de transmission ont une grande largeur de bande. L'implémentation dans chaque station d'un mécanisme de reconnaissance des collisions et de génération d'intervalles de temps aléatoirement long n'est pas très complexe.

Les premiers réseaux avec un contrôle par contention utilisaient une technique simple pour détecter les collisions. A l'instant même du commencement d'une transmission, un compteur démarrait à une certaine valeur. Si au moment où ce compteur revenait à zéro la station source n'avait pas encore reçu une quittance de réception de la station destination, le message était retransmis. Cette technique conduit à une sousutilisation des canaux du réseau, ce qui provoque très rapidement une surcharge du réseau.

Des améliorations techniques peuvent augmenter la capacité maximale effective de transmission. Une première amélioration peut se réaliser en écoutant d'abord la ligne de transmission avant d'émettre. Si la station entend un signal quelconque sur la ligne, elle n'émet pas tout de suite, mais attend que la station actuellement sur la ligne ait fini d'émettre. Le nombre de collisions sur la ligne sera ainsi diminué.

Une meilleure solution consiste à écouter la ligne aussi pendant l'émission. Ceci permet de détecter les collisions effectives beaucoup plus rapidement qu'en constatant simplement l'absence d'une quittance de réception après un certain moment. Cette technique donne de bons résultats pour les cables comme moyens de transmission.

Sur un bus, toute erreur qui intervient dans une transmission sera considérée comme collision. Donc elles sont traitées automatiquement par le mécanisme de détection des collisions. Si cela se produit, le message entier est retransmis.

I.3.4. COMBINAISONS EXISTANTES DE TOPOLOGIES ET DE STRUCTURES DE CONTROLE.

Dans les chapitres précédents, on a présenté 3 topologies pour les réseaux de communication: étoile, anneau, bus; et 2 structures de contrôle: contrôle centralisé, contrôle distribué. Chaque topologie peut être utilisée avec chaque structure de contrôle.

Une étoile n'a pas besoin d'un contrôle des chemins de transmission, car ceux-ci sont dédicacés entre chaque station périphérique et la station centrale. Dans une telle topologie, la station centrale peut contrôler les transmissions de manière centralisée en faisant du sondage. Un contrôle distribué pourrait fonctionner de la manière suivante: chaque station périphérique émet quand elle veut et la station centrale doit être capable d'accepter simultanément tous les messages.

Le fonctionnement d'un anneau à contrôle centralisé a été décrit comme architecture type de Anderson et Jensen (I.1.6.6.). Les mécanismes de chaînage d'un signal de contrôle et de roulement d'enveloppes conviennent bien à une telle topologie. Mais un anneau peut aussi être contrôlé de manière distribuée par contention. Une station détecte une collision lorsque le message émis ne lui revient pas intact.

Un bus à contrôle centralisé (décrit dans la division I.1.6.7.) peut être implémenté à l'aide de n'importe lequel des mécanismes de contrôle centralisé (I.3.3.1.). Mais le mécanisme le plus adapté à la topologie de bus est la contention. Cette dernière combinaison est très prometteuse pour les réseaux locaux. Avec les cables coaxiaux, cette technique permet des transmissions à grande vitesse pour un faible coût. Très peu d'erreurs de transmission ont été constatées avec cette technique. Comme Anderson l'a déjà signalé, le fonctionnement d'une telle architecture est très fiable.

I.3.5. INTERFACES ENTRE LES STATIONS ET LE RESEAU DE COMMUNICATION.

Un interface pour une station sur un réseau local est constitué de deux parties:

- une partie orientée réseau, qui doit effectuer toutes les fonctions de contrôle d'une transmission sur le réseau
- une partie orientée station, qui doit tenir compte de la structure spécifique des entrées/sorties de la station; elle doit aussi s'occuper des échanges de données entre la station et la partie orientée réseau de l'interface.

L'élément processeur d'une station peut être constitué d'un système à microprocesseur, d'un miniordinateur ou même d'un gros ordinateur. Les parties orientées station des interfaces pour les systèmes à microprocesseur sont souvent les plus faciles à réaliser, car ces systèmes sont construits autour d'un bus simple. Les interfaces pour les miniordinateurs sont souvent complexes car ils doivent être assez performants pour utiliser toute la capacité d'un réseau local. Les interfaces pour les gros ordinateurs sont les plus complexes. Cela est dû au fait que ceux-ci ont parfois des structures d'entrées/sorties aussi complexes qu'un ordinateur complet.

En général, la topologie et la structure de contrôle d'un réseau local sont assez simples, ce qui réduit la complexité de la partie orientée réseau d'un interface. Celle-ci ne change pratiquement pas, que ce soit pour un système à microprocesseur, un miniordinateur ou un gros ordinateur.

Au départ, les interfaces pour les réseaux de communication étaient traités comme des interfaces d'entrées/sorties. Maintenant que le coût du hardware a diminué, un microprocesseur peut être incorporé dans la partie orientée station d'un interface. Celui-ci devient intelligent et n'a plus besoin d'être activé par le processeur central. Les parties orientées station des interfaces forment alors des processeurs frontaux (front-end processors) comme ils sont utilisés dans les réseaux longue distance. Par contre les interfaces orientés réseau deviennent plus simples et peuvent être standardisés.

Comme les interfaces du côté réseau sont beaucoup moins complexes pour les réseaux locaux que pour les réseaux longue distance, le coût de l'interfacage d'un ordinateur à un réseau est moins lié au réseau qu'à la station. Ce déplacement du coût a deux conséquences:

- il devient économiquement justifiable de relier des systèmes à microprocesseur et des miniordinateurs à un réseau local
- il faut réfléchir à deux fois avant de connecter un gros ordinateur directement sur un réseau local.

I.3.6. PROTOCOLES DES RESEAUX LOCAUX.

Les protocoles des réseaux locaux sont conformes au modèle de Thurber (chapitre I.1.2.).

Les techniques des réseaux locaux ont des implications sur les protocoles de bas niveau. La grande performance du hardware permet de simplifier les protocoles de contrôle.

La grande vitesse de transmission permet d'expliciter toutes les informations de contrôle et d'uniformiser le format des messages. Des méthodes pour simplifier le contrôle des communications sont:

- élaborer un format standard pour transmettre n'importe quel message, c'est-à-dire, on transmet toujours la même suite de caractères de contrôle, même s'ils ne sont parfois pas nécessaires
- travailler avec des adresses les plus explicites possibles pour économiser les transformations à l'aide de tables.

Comme les temps de transmission sont très courts dans les réseaux locaux, il y a une tendance à adopter le schéma de question-réponse pour les conversations. Ainsi il n'y a plus de canal logique qui sera établi entre les stations pour échanger une longue suite de messages, mais chaque message établit une nouvelle liaison. Ces méthodes sont favorisées par des architectures de réseaux locaux qui autorisent toutes les stations à écouter ce qui se passe sur le réseau de communication. Pour la même raison, les échanges d'informations entre stations se font souvent par diffusion de messages.

Dans les réseaux locaux, les protocoles de haut niveau s'intègrent de plus en plus dans les systèmes d'exploitation des stations. Comme les réseaux locaux relient généralement des stations très diverses, les liaisons software entre les protocoles du réseau et les systèmes d'exploitation doivent être adaptées séparément pour les différentes stations. Pour simplifier ces adaptations, certains chercheurs ont créé une nouvelle génération de systèmes d'exploitation [p.ex. MININET (MANNING74), Distributed Loop Operating System for Distributed Loop Computer Network (LIU77)] qui tiennent compte des capacités des réseaux locaux.

Voici quelques applications qui ont été implémentées sur des réseaux locaux à l'aide des protocoles de niveau utilisateur.

I.3.6.1. Accès à des ressources communes.

Lorsqu'une seule machine ne suffit plus à exécuter l'ensemble des processus assez rapidement, une seconde machine lui est associée et les processus sont répartis de manière dynamique sur les deux ordinateurs. Cela forme

alors un système distribué qui doit résoudre le problème des communications entre les processus qui sont parfois critiques du point de vue du temps. La technologie des systèmes distribués locaux permet de répondre à ce problème. Les deux ordinateurs peuvent aussi se partager des périphériques très coûteux.

I.3.6.2. Transfert de fichiers.

Dans un système distribué local, un fichier contenant des données ou un programme peut être transféré très rapidement d'un ordinateur dans un autre. Ceci nécessite des transformations dans les systèmes d'exploitation des stations pour pouvoir demander facilement le transfert de tout un fichier. En sophistiquant un peu, l'utilisateur pourra simplement demander un fichier sans savoir où il se trouve dans le système distribué. Le système d'exploitation devra se débrouiller pour retrouver le fichier demandé.

II° PARTIE

COBUS, UN EXEMPLE DE RESEAU LOCAL

II.1. PRESENTATION DU RESEAU COBUS.

Le réseau local COBUS (COaxial BUS) a été développé au Laboratoire de Calculatrices Digitales (LCD) à l'Ecole Polytechnique Fédérale de Lausanne (EPFL) en Suisse. Il s'agit d'un système pour échanger des informations entre stations intelligentes. Les transmissions sont entièrement controlées par firmware. Ce terme désigne du software qui a été gelé en ROM (Read Only Memory). Ce software utilise les instructions standards et remplace une logique cablée. Ainsi le hardware est réduit au stricte minimum.

Comme son nom l'indique, le réseau local COBUS a une topologie de bus, qui est réalisé par un cable coaxial. Les transmissions se font en série et sont contrôlées de manière distribuée par le mécanisme de contention (voir I.3.3.2.). Le bus est totalement passif, et chaque station possède un dispositif hardware pour arbitrer l'accès au bus. La vitesse de transmission sur le réseau COBUS est de 20 kilobytes par seconde. Le bus coaxial peut relier jusqu'à 64 stations intelligentes et peut atteindre une longueur de 200 mètres.

Les transmissions entre les stations émettrices et réceptrices se font de manière intelligente, c'est-à-dire, les deux stations établissent un dialogue. Ainsi un message est seulement envoyé sur le bus s'il a de grandes chances d'être accepté par la station réceptrice. Dès que l'une des stations soupçonne une erreur dans la transmission, elle arrête l'émission ou la réception, et le message est entièrement retransmis. Une quittance de réception est seulement accordée lorsque le message est arrivé intact en mémoire de la station réceptrice.

L'application première du réseau local COBUS est la mise en commun des ressources d'un miniordinateur pour des terminaux intelligents. Dans la suite, seulement les dispositifs qui comportent un processeur pour exécuter des programmes seront qualifiés d' "intelligents".

Actuellement 2 configurations types pour le réseau local COBUS peuvent être réalisées. L'une, représentée à la FIG.9, est constituée par un miniordinateur comme station de ressources et par des terminaux intelligents. L'autre comporte un microordinateur à la place du miniordinateur. Ces 2 réseaux sont indépendants et chacun ne peut supporter qu'une seule station de ressources.

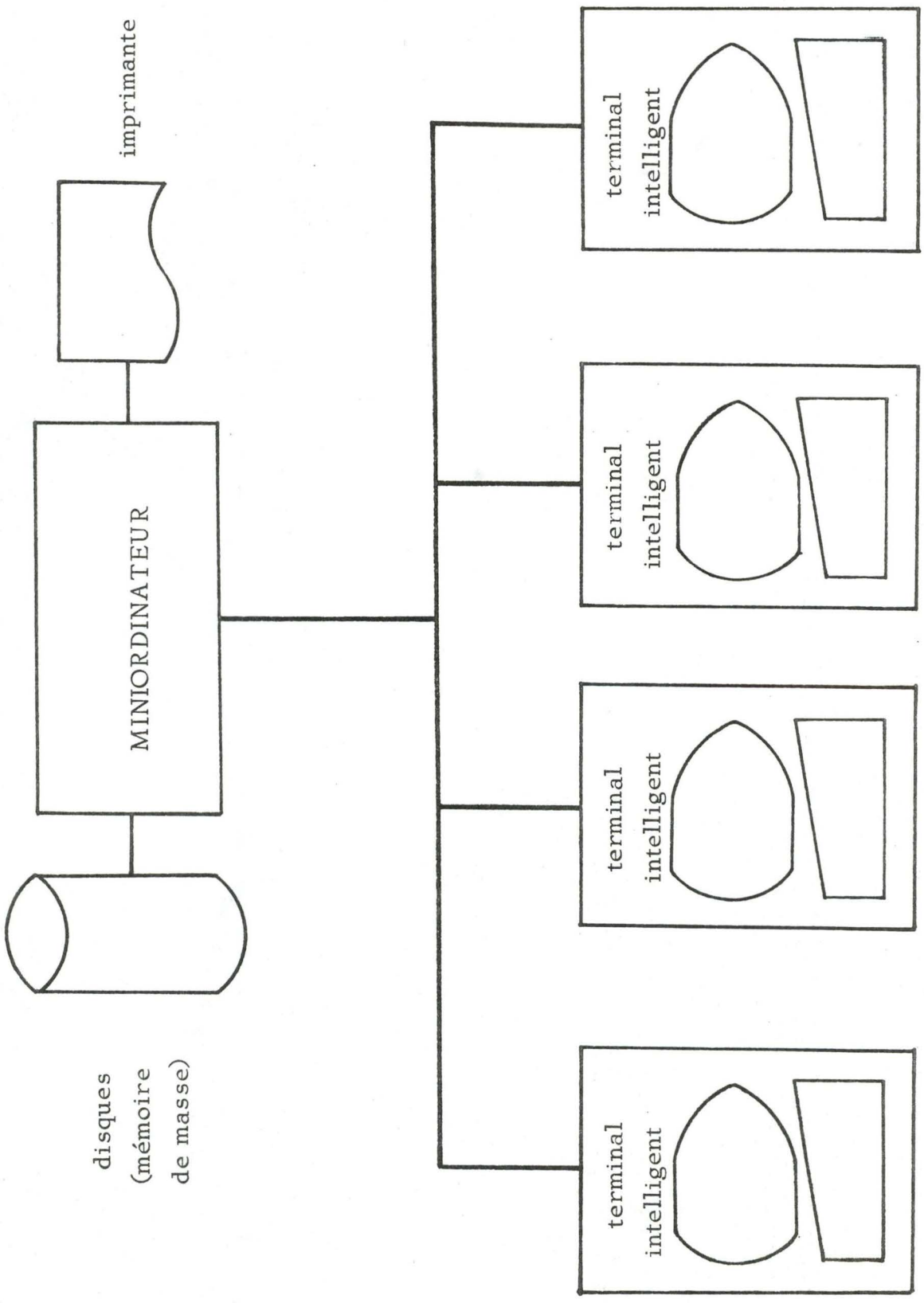


FIG.9 RESEAU LOCAL COBUS

II.2 ARCHITECTURE DU RESEAU COBUS.

II.2.1. TOPOLOGIE.

A côté des principales caractéristiques d'une architecture d'un système distribué (voir I.1.5.), d'autres critères spécifiques à la réalisation du réseau local COBUS sont intervenus dans le choix d'une topologie (NICOUD78). Le choix va s'effectuer entre les 3 topologies (étoile, anneau, bus) particulièrement adaptées aux réseaux locaux.

Pour réduire le nombre de pannes, il faut garder le nombre d'éléments actifs le plus bas possible. Comme l'emploi de ce réseau local prévoit beaucoup de transferts de fichiers, le temps de transmission d'un message doit être très court. Cela peut se réaliser de la meilleure façon lorsqu'on dispose d'une liaison directe entre les stations source et destination.

Le hardware peut être gardé minimal en choisissant la topologie qui nécessite le moins d'interfaces et de lignes de transmission.

Les 2 topologies en étoile et en anneau sont beaucoup plus vulnérables qu'un bus à cause du nombre élevé d'éléments actifs. En plus, elles n'ont pas de lignes de transmission directes entre toutes les stations.

Un bus par contre peut être implémenté de manière passive sans éléments actifs, comme dans ETHERNET (METCALFE75). Le bus commun établit des liaisons directes entre toutes les stations. L'implémentation hardware d'un bus se contente d'un seul interface par station et d'une seule ligne de transmission.

Un dernier critère très important dans le choix de la topologie de bus a été le fait que d'autres projets à l'EPFL et dans d'autres centres de recherche (XEROX à PALO ALTO) ont déjà été réalisés autour des bus.

II.2.2. STRUCTURE DE CONTROLE.

Le choix de la méthode de contrôle du bus se fait d'abord entre 2 grandes catégories mises en évidence dans le chapitre I.3.3. .

Le contrôle centralisé est éliminé à cause de l'effet désastreux d'une panne du dispositif de contrôle. D'ailleurs, les mécanismes pour le contrôle centralisé (décrits dans I.3.3.1.) ne s'adaptent pas très bien à une implémentation d'un réseau local avec une topologie de bus.

Par contre, le mécanisme de contrôle par contention (décrit dans I.3.3.2.) est facile à mettre en oeuvre sur un bus et requiert un minimum de moyens. Une autre raison importante est que certains projets (DUTOIT75) ont déjà

été faits à l'EPFL sur un mécanisme d'arbitration d'un bus.

Les mécanismes hardware sousjacents à la réalisation du contrôle de contention utilisé dans le réseau COBUS sont détaillés dans la division II.3.2.1. .

Chaque station connectée sur le réseau écoute continuellement le bus. Ainsi toutes les stations peuvent détecter le passage d'un message sur la ligne. Par analogie avec les émissions radio, on dit que les stations détectent une "porteuse" lorsqu'il y a une transmission en cours sur le bus.

Les collisions peuvent se passer seulement si deux stations commencent à émettre pratiquement au même instant. Cela veut dire que les stations ont commencé à émettre dans un intervalle temporel déterminé par le temps de propagation d'un signal d'un bout à l'autre du bus (DUTOIT75).

Lors d'une collision (FIG.10), le résultat de la superposition sur le bus de tous les bits émis en même temps par les différentes stations A, B et C correspond à une fonction logique OU (A+B+C). Les stations A,B,C qui sont en train d'émettre font un échantillonnage de l'état du bus et le comparent aux bits qu'elles viennent d'émettre. Dès que l'une de ces stations A,B,C constate une différence entre ces deux signaux, elle conclut à une collision et arrête son émission. Le protocole (voir II.4.1.2.) assure qu'il y aura toujours une seule station (A) qui ne détectera pas de collision. Cette station là sera considérée comme prioritaire et effectuera son émission sans perdre de temps. Mais il n'y a pas de station favorisée pour l'obtention du bus, car chaque interface dispose d'un réglage pour que la station découvre plus ou moins rapidement que le bus est devenu libre après une transmission.

Tous ces contrôles se font au niveau du bit et sont effectués par hardware.

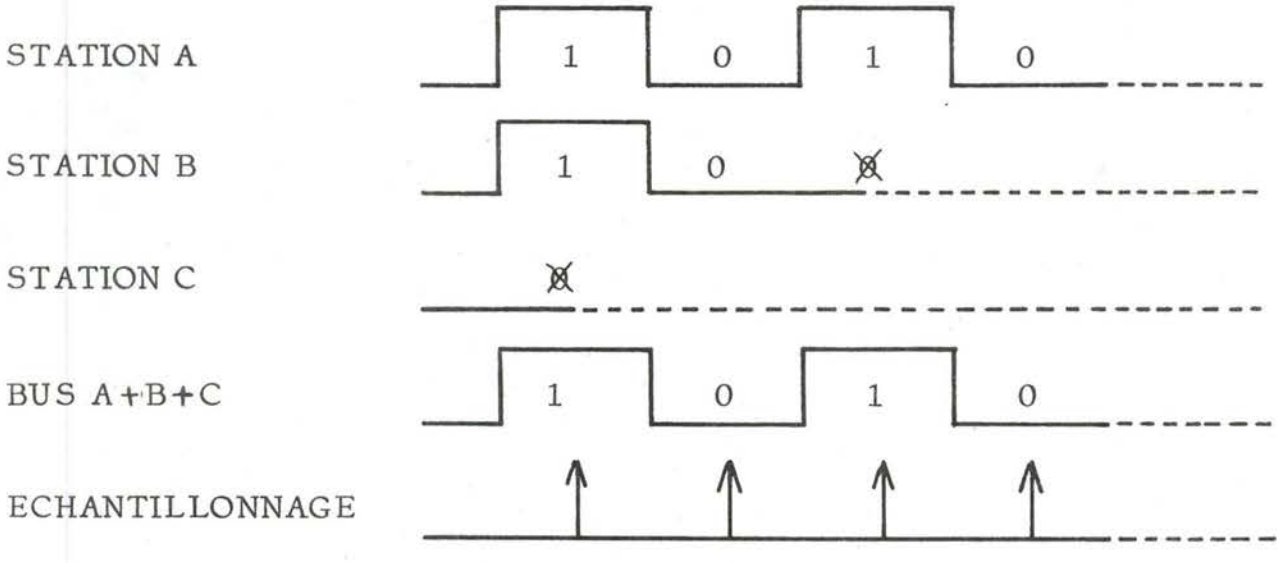


FIG.10
ARBITRAGE DU BUS

II.3. HARDWARE DU RESEAU COBUS.

II.3.1. MOYEN DE TRANSMISSION.

La ligne de transmission doit pouvoir supporter plusieures émissions simultanées et doit être capable de faire une superposition de plusieurs signaux en effectuant une fonction logique OU. De plus, chaque station doit pouvoir se connecter facilement sur le bus sans déranger des transmissions éventuelles en cours. Comme le bus doit être bidirectionnel, le signal doit se propager à partir de la station source vers les deux extrémités. Pour limiter les erreurs de transmission, il faut que la ligne de transmission soit bien isolée contre les bruits électromagnétiques de l'environnement.

Le choix final s'est fixé sur le cable coaxial, qui est bien protégé contre les bruits parasites et qui suffit à toutes les exigences énoncées ci-dessus. Ce moyen de transmission a déjà été utilisé pour d'autres réseaux locaux analogues (ETHERNET). Un inconvénient mineur est qu'il faut intercaler des connecteurs spéciaux pour relier les stations au bus.

II.3.2. INTERFACES.

Comme le réseau COBUS peut fonctionner soit avec un miniordinateur et des terminaux intelligents, soit avec un microordinateur et des terminaux intelligents, 3 types de stations doivent pouvoir se connecter sur le bus (voir FIG.11). Ces 3 types de station différents sont:

- terminaux intelligents formés par des systèmes à microprocesseur SMAKY6 (description voir III.2.1.1.) capables d'effectuer des processus
- microordinateur LSI-11/2 de Digital Equipment Corporation (description voir III.2.1.2.) avec 2 disquettes comme mémoire de masse
- miniordinateur ECLIPSE de Data General avec une grande mémoire de masse (2 gros disques) et d'autres périphériques comme imprimante et perforateur de bandes papier.

Puisque chaque type de station a une architecture interne différente, il faut créer un interface différent pour chaque type (voir FIG.11). Les 3 interfaces différents qui existent actuellement sont:

- un interface non intelligent orienté réseau encore appelé interface COBUS dans la suite et utilisé avec le terminal intelligent

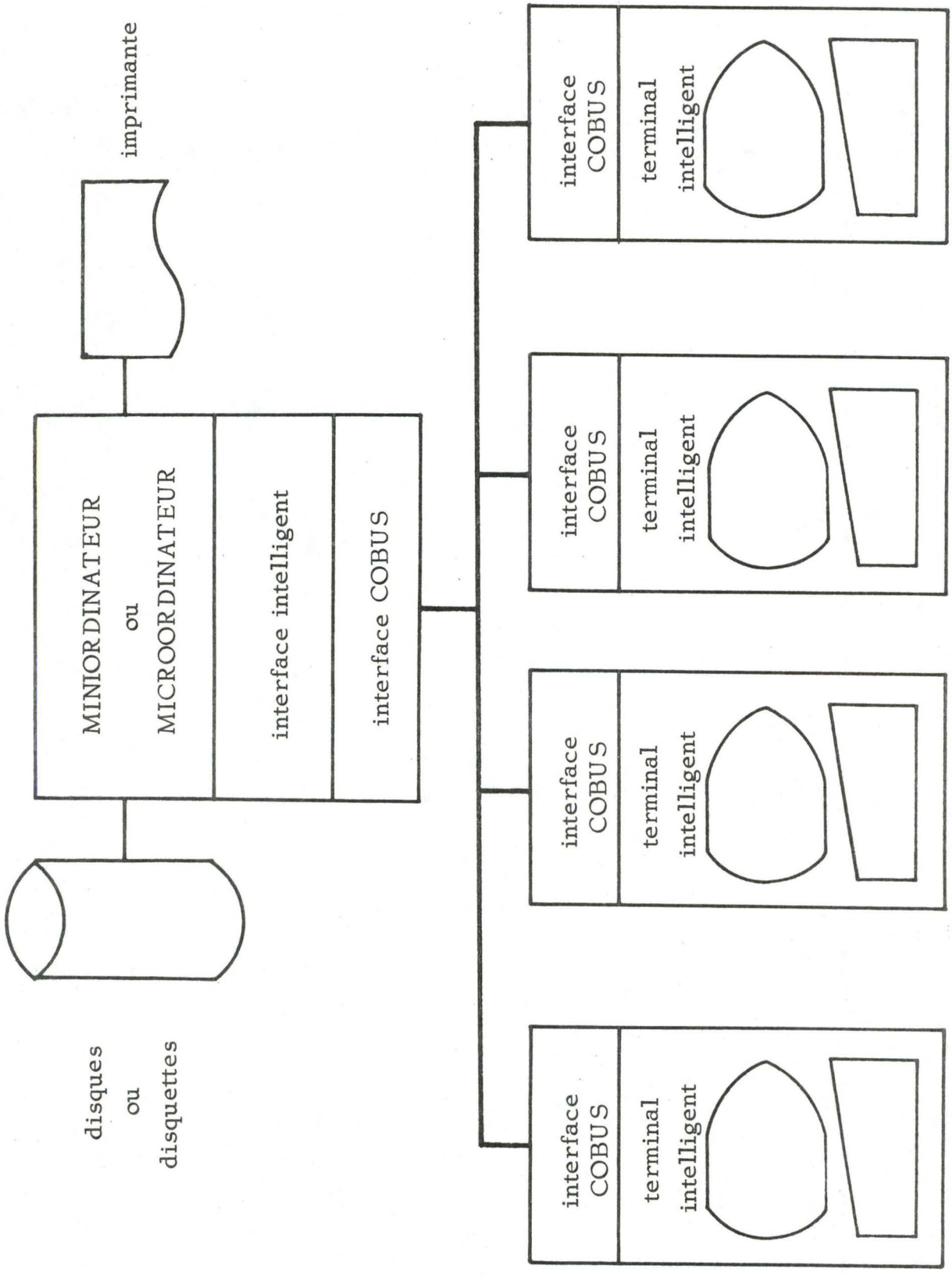


FIG.11 ORGANISATION DU RESEAU LOCAL COBUS

- un interface intelligent (avec 1 microprocesseur) orienté station pour le microordinateur, et qui utilise l'interface non intelligent vers le réseau
- un interface intelligent (avec 2 microprocesseurs) orienté station pour le miniordinateur, et qui utilise l'interface non intelligent vers le réseau.

Les places qu'occupent ces interfaces dans le réseau local COBUS sont indiquées sur la FIG.11 .

Dans les stations terminales, c'est l'unique microprocesseur de la station qui active l'interface non intelligent. Pendant une transmission, le terminal ne peut pas effectuer d'autres processus. Cela est acceptable dans les stations terminales, car elles travaillent en monoprogrammation.

Dans les stations de ressources, ce sont les microprocesseurs supplémentaires des interfaces intelligents qui dirigent les transmissions à travers l'interface côté réseau, tandis que le processeur principal de la station peut continuer à effectuer d'autres processus en parallèle.

II.3.2.1. INTERFACE COBUS.

L'interface COBUS orienté vers le réseau de communication sert à envoyer les bons signaux au bon moment sur la ligne. Le microprocesseur qui s'occupe des transmissions utilise cet interface pour faire des émissions/réceptions sur le bus. Il dirige cet interface en lui communiquant des ordres et des données à transmettre sur la ligne. Bien que l'interface COBUS orienté réseau ne soit pas intelligent, il est quand même capable de surveiller la ligne et d'indiquer l'état de celle-ci.

La FIG.12 montre l'agencement des différentes unités fonctionnelles qui se trouvent sur cet interface COBUS.

1) Le composant principal de l'interface COBUS est un ACIA MC6850 de Motorola (Asynchronous Communication Interface Adapter). Il s'agit d'un circuit intégré très complexe qui sert à adapter un interface pour des communications asynchrones. L'ACIA possède 2 registres en cascade à l'entrée (input registers) et 2 autres à la sortie (output registers). Un registre de contrôle (control register) reçoit les ordres, et un registre d'état (status register) indique l'état de la transmission. Cette unité a plusieures fonctions:
   - sérialisation/désérialisation des bytes qui passent dans les registres d'entrée et de sortie

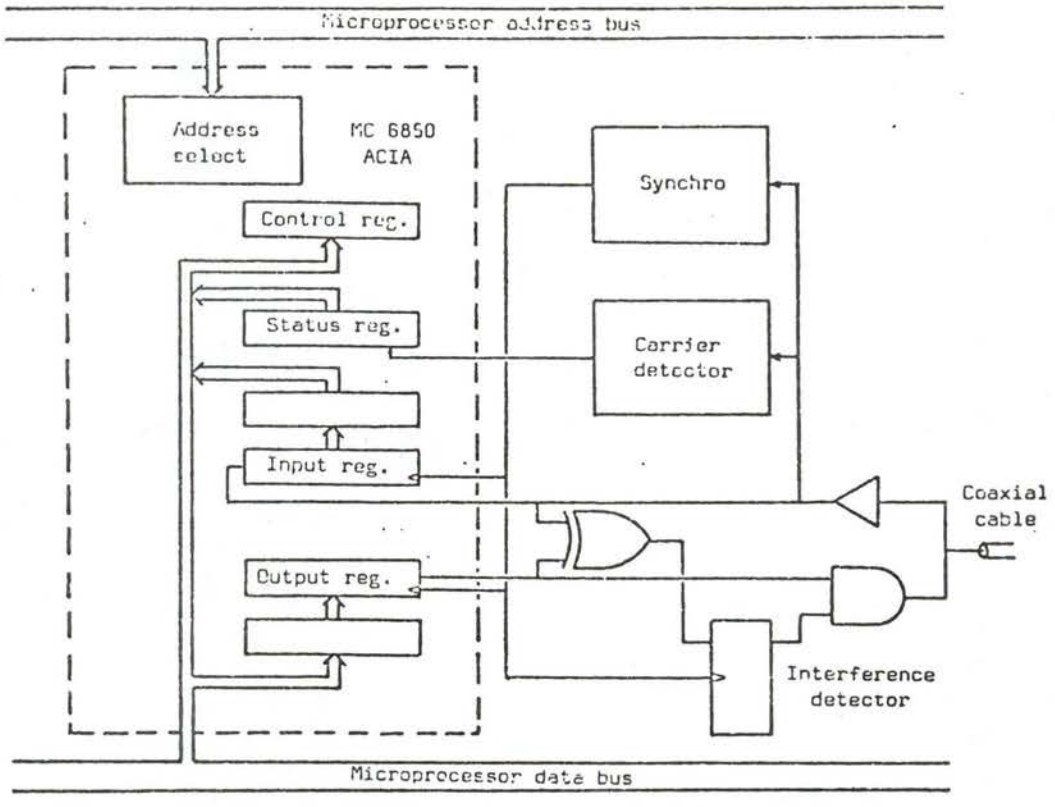


FIG.12
INTERFACE COBUS

- formattage des bytes à l'émission et à la réception, c'est-à-dire: rajoute d'un bit de parité sur chaque byte en émission, contrôle du bit de parité en réception, élimination des bits de stop ou de start
- indication de l'état de la transmission, c'est-à-dire: byte disponible en entrée, byte reçu avec mauvaise parité, bytes écrasés en entrée, manque de bit de stop ou de start, bus libre.

2) Une logique de synchronisation (synchronization) des bits, qui est rendue nécessaire par le fait que pour de grandes vitesses de transmission, il faut une synchronisation externe au ACIA. Trois vitesses de transmission sont disponibles sous contrôle du firmware. Comme les transmissions se font de manière asynchrone, les interfaces des stations source et destination doivent être resynchronisés à chaque début de message. Pour faire cela, c'est le premier bit de chaque message qui démarre la synchronisation.

3) Une logique de détection de porteuse (carrier detection) qui indique une activité sur le bus. Chaque bit à 1 qui passe sur le bus fait passer un monostable à retard (MEINADIER75) dans son état instable. Le monostable retombe dans son état stable après un temps qui correspond au temps de transmission de 2 mots (4 bytes) sur le bus. Ce délai permet aux 2 stations qui dialoguent de faire des décisions par firmware sans perdre le bus.

4) Une logique de détection d'interférences (interference detector), qui au milieu de chaque bit envoyé sur le bus, compare l'état de celui-ci avec ce que la station vient d'émettre (voir FIG.10). Dès qu'il y a une différence, la station se déconnecte du bus en bloquant l'émission. Le détecteur d'interférences signale la présence d'une station plus prioritaire lors d'une collision. Ce dispositif détecte aussi des bruits parasites sur le bus et arrête immédiatement l'émission libérant ainsi le bus.

Pour qu'un tel interface devienne utilisable, il faut pouvoir l'adresser par firmware et lui communiquer des ordres et des données. Il faut aussi que l'interface puisse fournir au firmware les données reçues, indiquer des erreurs ainsi que l'état de l'interface et celui du bus. Ces communications se font à l'aide de "registres" implémentés sur l'interface. Par "registres" on entend ici des zones de 8 bits qui peuvent être adressées en lecture ou en écriture comme des zones mémoire.

II.3.2.2. INTERFACE POUR MICROORDINATEUR.

Dans le réseau local COBUS, la station microordinateur regroupe des ressources que tous les autres terminaux veulent utiliser. Le processeur du microordinateur ne peut pas faire lui-même les transmissions à travers l'interface non intelligent, car il ne lui resterait pas assez de temps pour assurer les services que les autres stations lui demandent.

Une solution analogue à celle des processeurs frontaux simplifie la tâche du microordinateur. Un processeur de protocole est implémenté sur un interface qui se connecte directement sur le bus interne du microordinateur. D'un côté, ce microprocesseur de l'interface fait des transferts en mode DMA (Direct Memory Access) entre la mémoire centrale et l'ACIA (décrit en II.3.2.1.) et inversément. De l'autre côté, il surveille la ligne de transmission à travers l'interface non intelligent orienté réseau. Les transferts en DMA entre l'ACIA et la mémoire centrale s'avèrent nécessaires à cause de la vitesse de transmission des messages sur la ligne. Cet interface côté station est contrôlé par firmware. Le microordinateur peut lui demander d'émettre ou de recevoir un message à travers le réseau local.

L'interface intelligent pour microordinateur est constitué par:

- 1 microprocesseur Z80 de ZILOG
- 1 mémoire morte (ROM) qui contient le programme que le microprocesseur doit exécuter
- 1 mémoire vive (RAM) pour stocker les variables utilisées par le programme

- 1 interface COBUS non intelligent orienté réseau
- 1 interface DMA pour se connecter sur le bus interne du microordinateur et pour faire du DMA en mémoire centrale de celui-ci.

L'interface DMA assure l'arbitrage du bus du microordinateur LSI-11/2 et fait des transferts directs dans la mémoire centrale sans passer par le processeur. La FIG.13 montre l'organisation et l'interconnexion de ces éléments.

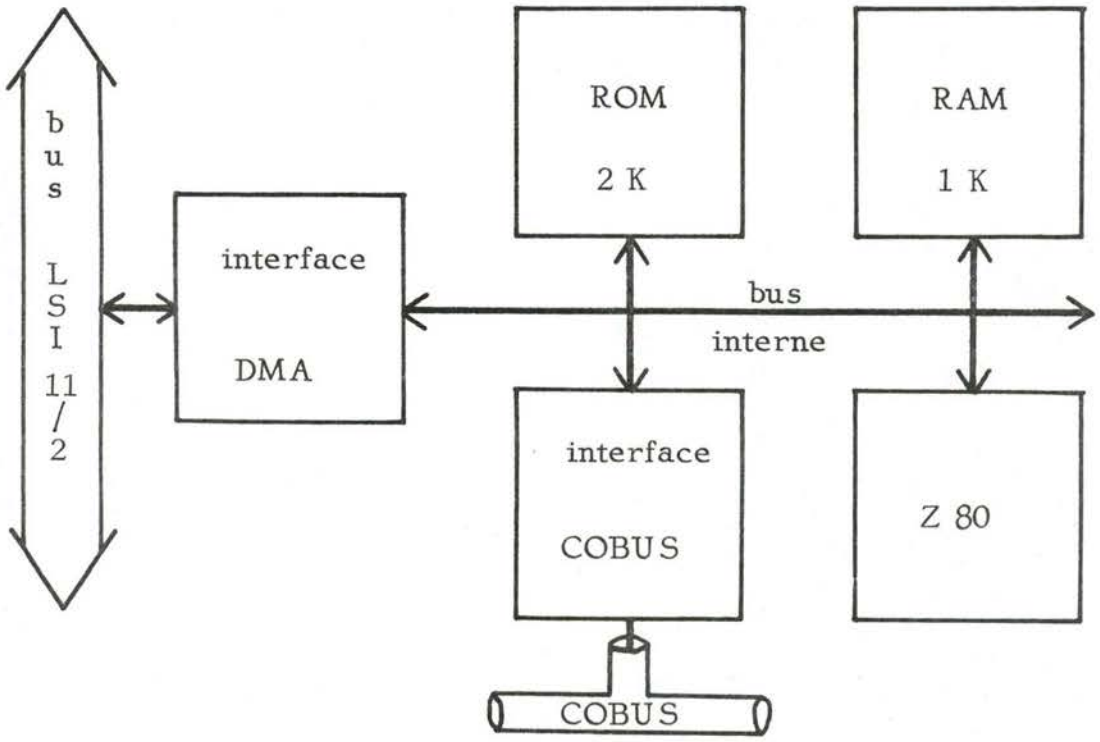


FIG.13

INTERFACE POUR MICROORDINATEUR LSI-11/2

Le microprocesseur Z 80 et le microordinateur LSI-11/2 ont les mêmes espaces d'adressage. Ces espaces sont divisés en 2 zones. Pour le microprocesseur, les adresses de 0 à 56K accèdent la mémoire centrale du microordinateur en DMA. Les 8K supérieurs recouvrent les mémoires ROM et RAM de l'interface intelligent. Pour le microordinateur, les 56K inférieurs peuvent contenir des programmes ou des données, et les 8K supérieurs correspondent aux interfaces d'entrée/sortie et à leurs registres (voir DEC77). Il y a donc une superposition des adresses de 0 à 56K pour les 2 processeurs. La FIG.14 indique les espaces d'adressage des 2 processeurs avec 2 rectangles distincts, mais les adresses de 0 à 56K n'existent physiquement qu'une seule fois en mémoire du microordinateur.

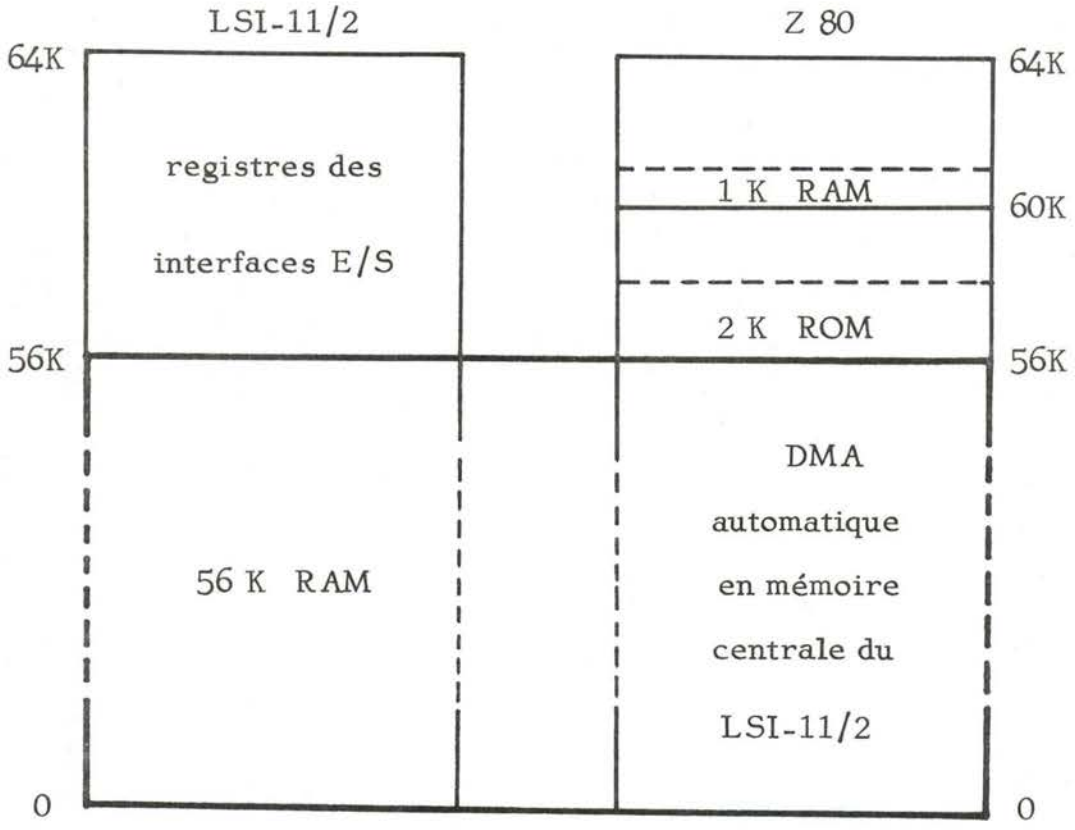


FIG.14

ESPACES D'ADRESSAGE DES 2 PROCESSEURS

Sur l'interface intelligent se trouve aussi un registre de 16 bits qui est organisé de manière conforme aux normes pour tous les interfaces qui veulent se connecter sur le bus interne du microordinateur (voir DEC77), et qui correspond à une adresse cablée en hardware sur l'interface. Ce registre est adressable par le microordinateur comme une position mémoire.

Le processeur du microordinateur travaille avec 16 bits tandis que le microprocesseur de l'interface intelligent travaille avec 8 bits. Le travail du hardware est facilité par le fait que la mémoire centrale du microordinateur peut aussi être adressé par des zones de 8 bits.

II.3.2.3. INTERFACE POUR MINIORDINATEUR.

Le miniordinateur peut aussi jouer le rôle d'une station de ressources dans le réseau COBUS. Comme pour le microordinateur, le miniordinateur se connecte sur le bus du réseau à l'aide d'un interface intelligent.

Dans le miniordinateur, qui travaille avec 16 bits, une position de 8 bits ne peut être adressée sans altérer les 8 autres bits qui y sont associés pour former un mot. Voilà pourquoi un second microprocesseur a été ajouté pour s'occuper des transferts en DMA entre la mémoire centrale du mini-

ordinateur et une mémoire locale implémentée sur l'interface. Le processeur de protocole ne s'occupe alors que des transferts de données entre la mémoire locale et l'interface non intelligent orienté vers le réseau.

Les différents composants de cet interface sont:

- 2 microprocesseurs Z 80 de ZILOG
- 2 mémoires mortes (ROM) qui sont associées aux 2 microprocesseurs respectivement, et qui contiennent les programmes que ceux-ci doivent exécuter
- 1 mémoire vive (RAM) locale qui se partage logiquement en 3 zones: une pour chaque microprocesseur pour stocker ses variables et une zone accessible aux 2 microprocesseurs pour le stockage intermédiaire des messages arrivants ou partants sur la ligne de transmission
- 1 interface COBUS non intelligent orienté réseau qui est dirigé par le processeur de protocole
- 1 interface DMA entre la mémoire RAMlocale et la mémoire centrale du miniordinateur, et qui est sous le contrôle du second microprocesseur.

Ces éléments sont reliés comme indiqué à la FIG.15 .

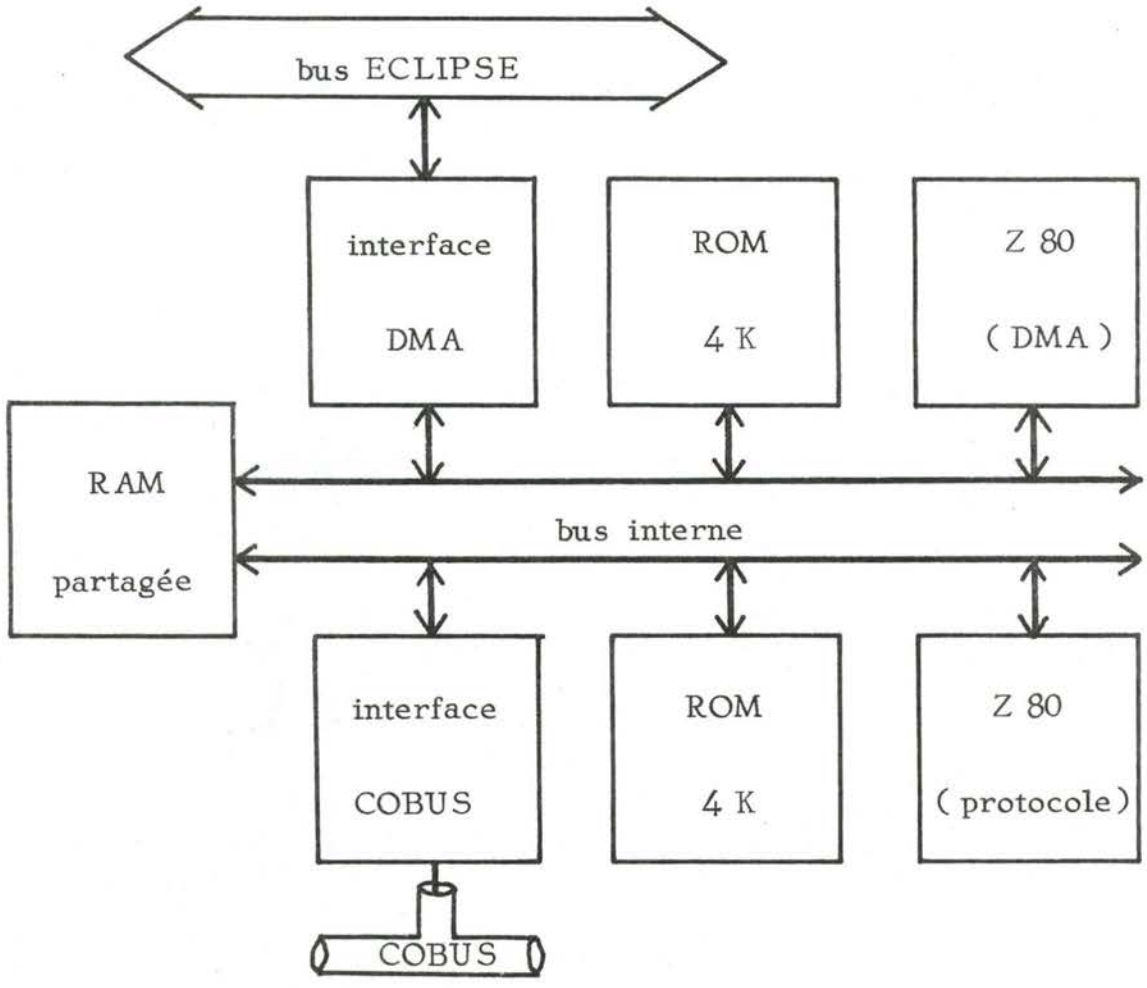


FIG.15

INTERFACE POUR MINIORDINATEUR ECLIPSE

L'interface DMA est vu par le miniordinateur comme un interface entrée/sortie en mode DMA. Mais du côté de l'interface intelligent, il est contrôlé par firmware. La partie principale de l'interface DMA est un circuit intégré pour faire des entrées/sorties parallèles, et qui peut être contrôlé par programmation. Ce circuit permet de lire un mot (2 bytes) dans la mémoire du miniordinateur, changer le byte voulu et puis réécrire le mot en entier. Toutes les communications entre le miniordinateur et l'interface intelligent passent par l'interface DMA.

II.3.3. SECURITE FOURNIE PAR HARDWARE.

Pendant la transmission d'un message sur la ligne, celui-ci peut être partiellement ou entièrement détruit. La destruction peut être due à 2 facteurs:

- impulsions parasites sur la ligne de transmission
- interférences avec d'autres messages (collisions).

Un autre phénomène fâcheux est la perte d'une partie ou d'un message entier. La perte désigne ici la rupture pure et simple d'un dialogue entre 2 stations. Cela peut se passer lorsque:

- la station destination n'est pas connectée sur le réseau ou est inactive
- une panne hardware d'un interface des 2 stations en liaison déconnecte celle-ci du réseau.

Le hardware, bien que minimal, apporte quand même sa contribution à la sécurité des transmissions.

Les impulsions parasites détruisent rarement plus que quelques bits. Bien que le hardware n'intervient qu'au niveau du bit, il ne peut détecter un bit erronné qu'au niveau du byte en entier. L'ACIA (décrit en II.3.2.1.) fait un contrôle de parité sur chaque byte reçu.

Lors des collisions, il y a bien destruction de bits à cause de la fonction logique OU réalisée sur le cable coaxial. Le détecteur d'interférences découvre aussi la destruction de bits pendant toute l'émission car il écoute toujours le bus.

En cas de rupture d'un dialogue, il n'y a plus rien qui se passe sur le bus. Le détecteur de porteuse retombe dans son état stable, ce qui provoque une interruption.

II.4. SOFTWARE DU RESEAU COBUS.

II.4.1. PROTOCOLE DE TRANSMISSION.

II.4.1.1. CONTENU DU PROTOCOLE.

Le protocole de transmission fixe la signification de la suite de bytes que 2 stations émettent ou lisent sur la ligne lors d'une transmission d'un message.

Une station qui veut envoyer des données vers une autre établit un dialogue. Cela veut dire que les 2 stations échangent des informations dans les 2 sens pour se mettre d'accord avant d'envoyer les données proprement dites. Le protocole actuellement utilisé à l'EPFL exige 4 échanges sans que les 2 stations ne perdent le contrôle du bus. Les 4 échanges représentés par des grandes flèches dans la FIG.16 sont:

- entête du message de la source vers la destination
- quittance d'entête de la destination vers la source
- données proprement dites de la source vers la destination
- quittance du message de la destination vers la source.

Voyons maintenant le contenu détaillé d'un message:

L'entête du message contient:

1) 1 byte CBDEST qui est l'adresse de la station destination (de 1 à 377 octal)
2) 1 byte CBSRCE qui est l'adresse de la station source (de 1 à 377 octal)
3) 1 byte CBPTYP qui indique la vitesse de l'émetteur, le type du message et la présence d'un duplicata
4) 1 byte CBACSM de somme de contrôle longitudinal pour protéger les trois premiers bytes 1),2) et 3).

La quittance de l'entête du message contient:

5) 1 byte CBAACK qui contient les informations suivantes:
   - vitesse du récepteur
   - disponibilité du récepteur
   - utilité du duplicata
6) 1 byte CBACSM de somme de contrôle longitudinal pour protéger le byte 5) précédent.

Les données proprement dites contiennent:

7) 1 byte CBPLEN qui indique le nombre de bytes de données (jusqu'à 400 bytes octals)
8) de 1 à 400 bytes octals de données CBPKT

9) 1 byte CBPCSM de somme de contrôle longitudinal pour protéger les bytes de 7) et 8).

La quittance du message contient:

10) 1 byte CBPACK qui indique la bonne réception de tout le message et termine le dialogue (code fixe).

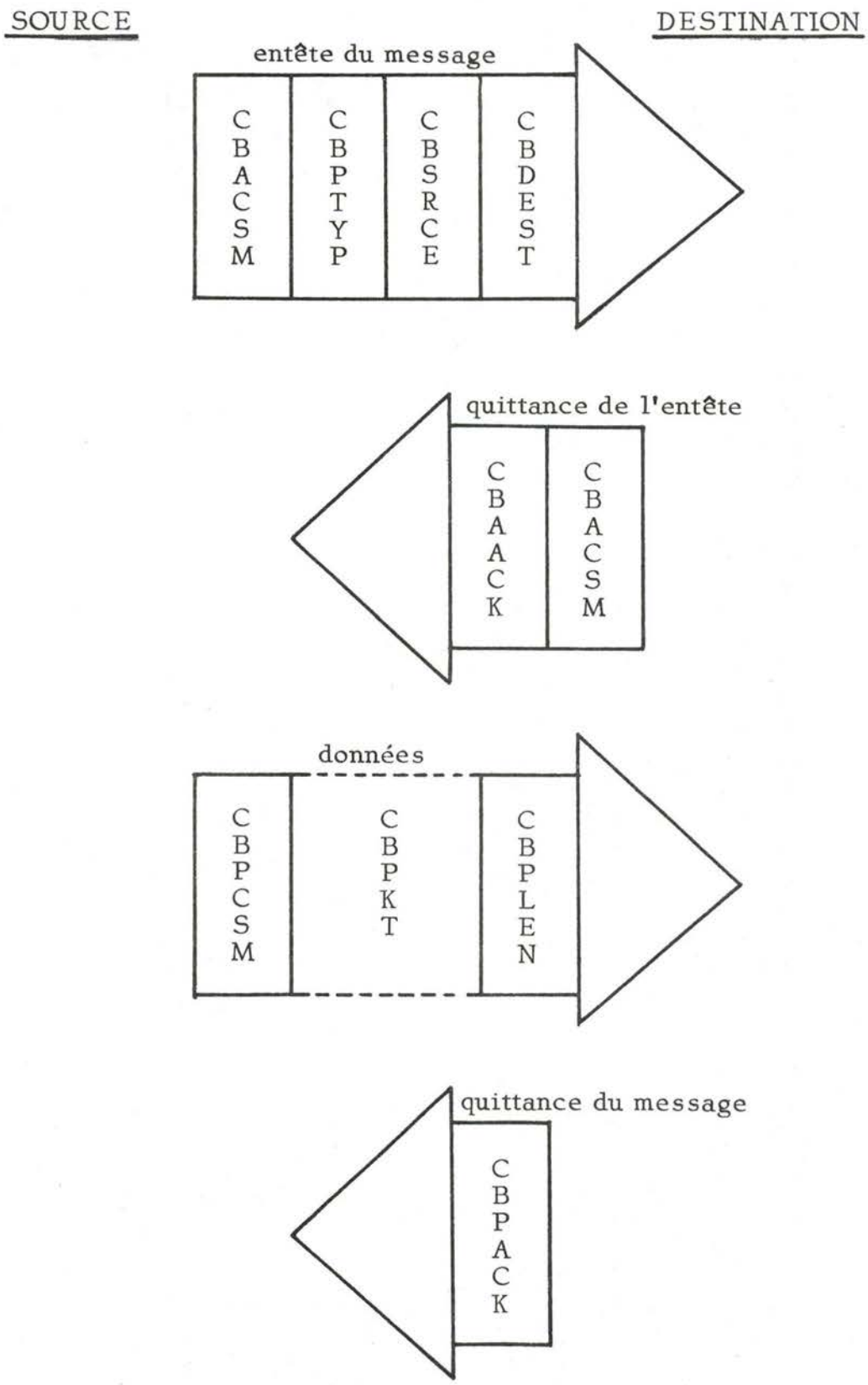


FIG.16

ECHANGES DU PROTOCOLE

II.4.1.2. SIGNIFICATION DU PROTOCOLE.

L'envoi des adresses de destination et de source sur la ligne permet une arbitration sûre du bus lors des collisions. Même si deux ou plusieures stations qui entrent en collision s'adressent à la même destination, au moins leurs propres adresses seront différentes. Donc après ces deux bytes, le bus du réseau local COBUS appartient à une seule et unique paire de stations qui peuvent alors communiquer sans être dérangées.

La transmission des bytes de 1) à 6) se fait à vitesse minimale pour permettre la reconnaissance d'adresse et d'autres contrôles effectués par firmware. La suite du dialogue se fait à une vitesse supérieure. Comme 3 vitesses sont possibles, celle-ci est contenue dans le byte CBPTYP pour que la station réceptrice puisse s'accorder à la même vitesse pour la transmission des données.

Le byte CBPTYP contient aussi un indicateur de duplicata nécessaire à cause de l'absence d'histoire (numérotation ou autre) entre les messages. L'indicateur de duplicata marque les retransmissions des messages. Le dialogue garantit la bonne réception d'un message à la fois, et le mécanisme de duplicata suffit donc pour assurer la transmission d'une suite de messages.

A la fin de l'entête il y a la somme de contrôle qui est calculée comme suit: les bytes précédents sont additionnés dans un registre de 8 bits sans s'occuper du débordement, et le complément à 2 de la somme constitue la somme de contrôle.

La quittance de l'entête du message sert à indiquer que la station destination est connectée sur le réseau. Le byte CBAACK contient aussi la vitesse de transmission qu'adoptera la station réceptrice pour recevoir les données. Cela permet à la source de s'accorder au besoin à cette vitesse. Il y a encore des informations sur la disponibilité de la station destination dans ce byte. En effet, celle-ci n'est peut-être pas encore prête à recevoir un message. Finalement ce byte contient une réponse sur la nécessité du duplicata. Lorsque tout s'est bien passé pendant un dialogue et que seulement la quittance du message a été perdue sur la ligne, alors ce message est retransmis. Mais en fait, cette retransmission est inutile et la station destination la refuse. Le byte 5) qui contient de l'information est de nouveau protégé par une somme de contrôle.

La station source envoie maintenant un byte contenant le nombre de bytes de données qui vont suivre. Ceci se fait à la vitesse convenue entre les 2 stations. Comme les données sont toujours constituées d'au moins 1 byte, le code 0 est utilisé pour indiquer le nombre maximal de bytes de données qui est de 256 (400 octal). Puis arrivent à la vitesse de transmission convenue

les bytes de données. Les bytes peuvent représenter n'importe quel code de 0 à 255 (377 octal). Le byte 7) et les bytes de données sont protégés par une somme de contrôle calculée comme tantôt.

La destination termine le dialogue par une quittance qui valide la réception de tout le message. Ce byte a un code fixe convenu une fois pour toutes et n'a pas besoin d'être protégé par une somme de contrôle.

II.4.2. ADRESSAGE.

Sur un bus, chaque station voit tous les messages qui passent. Chaque station doit donc analyser la destination de chaque message pour voir si celui-ci lui est adressé. Une telle comparaison d'adresses peut se faire par hardware ou firmware.

La solution hardware semble avantageuse car très rapide, mais elle ne correspond pas à l'objectif qui vise à implémenter les interfaces avec un minimum de hardware.

Le choix de la solution firmware a plusieures conséquences. Elle est très économique car ne nécessite pas de hardware supplémentaire. Par contre, elle est plus lente et nécessite le ralentissement des transmissions lors des émissions des adresses. Mais comme le rayon d'action du réseau COBUS dépend de la vitesse lors de l'arbitrage d'une collision, ce ralentissement permet d'augmenter la longueur du bus.

Le firmware étant très flexible, une station peut correspondre en même temps à plusieures adresses. Comme le firmware est encore raisonnablement rapide, il peut effectuer plusieures comparaisons dans le temps accordé pour prendre la décision. Ce détail trouve une application dans l'emploi particulier qui est fait de ce réseau.

A cause de la flexibilité de la reconnaissance d'adresse par firmware, il semble que la diffusion de messages (voir I.3.2.) soit facile à implémenter. Or il n'en est rien. Lorsqu'une station émet comme destination une adresse reconnue par plusieures autres stations, alors, selon le protocole décrit dans le chapitre précédent, toutes les stations réceptrices vont envoyer des quittances sur le bus. Cela provoque inévitablement des collisions irrécupérables. Il faudrait donc un protocole très différent sans quittance intégrée, ce qui ferait perdre tous les avantages de COBUS.

II.4.3. SECURITE FOURNIE PAR FIRMWARE.

Pendant la transmission, un message peut subir les dommages décrits dans le chapitre II.3.3. .

Le firmware fait un certain nombre de contrôles au niveau du byte pour assurer la sécurité de l'information. Pour détecter la destruction de un ou de plusieurs bits, le protocole prévoit des caractères de contrôle (somme de contrôle longitudinale) derrière toutes les informations qui peuvent varier d'une transmission à l'autre. C'est le cas pour les 3 premiers échanges d'un dialogue. Le 4° échange a une valeur fixe connue de toutes les stations et n'a donc plus besoin de protection. Le caractère de contrôle agit sur plusieurs bytes. Dans le cas du protocole COBUS, il représente simplement la valeur négative de la somme modulo 256 de tous les bytes à protéger. La station source calcule le caractère de contrôle avec les bytes qu'elle envoie. La station destination fait le même calcul avec les bytes reçus. Lorsque la station destination constate une différence avec le caractère de contrôle reçu de la station source, elle conclut à une erreur de transmission. Celle-ci peut avoir 2 causes, soit que l'un des caractères de l'information a été erroné, soit que le caractère de contrôle lui-même a été mal reçu. Dans les 2 cas, la transmission est arrêtée et le firmware prépare une retransmission.

Dans le cas de la rupture d'un dialogue, le firmware fait aussi une retransmission.

Après une détection d'une erreur de transmission, la station source essaie un certain nombre de fois de retransmettre le même message. L'intervalle de temps entre les essais augmente. Lorsque la station n'a pas réussi à transmettre le message sans erreur, elle conclut que la station destination n'est pas connectée ou pas disponible.

II.5 IMPLEMENTATION DU PROTOCOLE.

Cette section décrit le déroulement des programmes de firmware pour faire des transmissions selon le protocole. Le hardware des interfaces correspond au niveau de contrôle de ligne (niveau 0) et le firmware au niveau de contrôle de transmission (niveau 1) du modèle de Thurber (voir I.1.2.).

II.5.1. EMISSION.

Pour émettre un message, la station fait appel à la routine CBSEND (voir annexe B). Les paramètres en entrée sont:

- destination du message
- source du message
- nombre de bytes de données
- adresse de la zone mémoire qui contient les données.

Cette routine garnit une table de sortie avec les informations nécessaires à la transmission de ce message. Cette table contient aussi une demande d'émission qui sera utile pour les retransmissions. Puis CBSEND appelle la routine CBRTC qui initialise l'interface côté réseau et contrôle l'état du bus. Si le bus est libre, CBRTC branche à la routine CBSND qui s'occupe de l'émission selon les règles du protocole. Cette dernière routine se déroule selon l'organigramme de la FIG.17 et travaille avec les informations de la table de sortie.

En cas de retransmission, CBSND attend pendant un intervalle de temps qui augmente avec le nombre de retransmissions. En cas d'échec, elle initialise une nouvelle retransmission ou indique une erreur "pas de correspondant" lorsque le nombre maximal de retransmissions est atteint. Après avoir réussi, cette routine indique que l'émission est terminée, puis elle retourne à la routine CBSEND. Cette dernière contrôle l'indicateur d'erreur et fournit comme paramètres de sortie:

- l'indicateur d'erreur
- le code de l'erreur.

La retombée du détecteur de porteuse dans son état stable provoque une interruption qui déclenche la routine CBINT (voir chapitre suivant). Celle-ci contrôle s'il reste encore une demande d'émission dans la table de sortie, maintenant que le bus est libre.

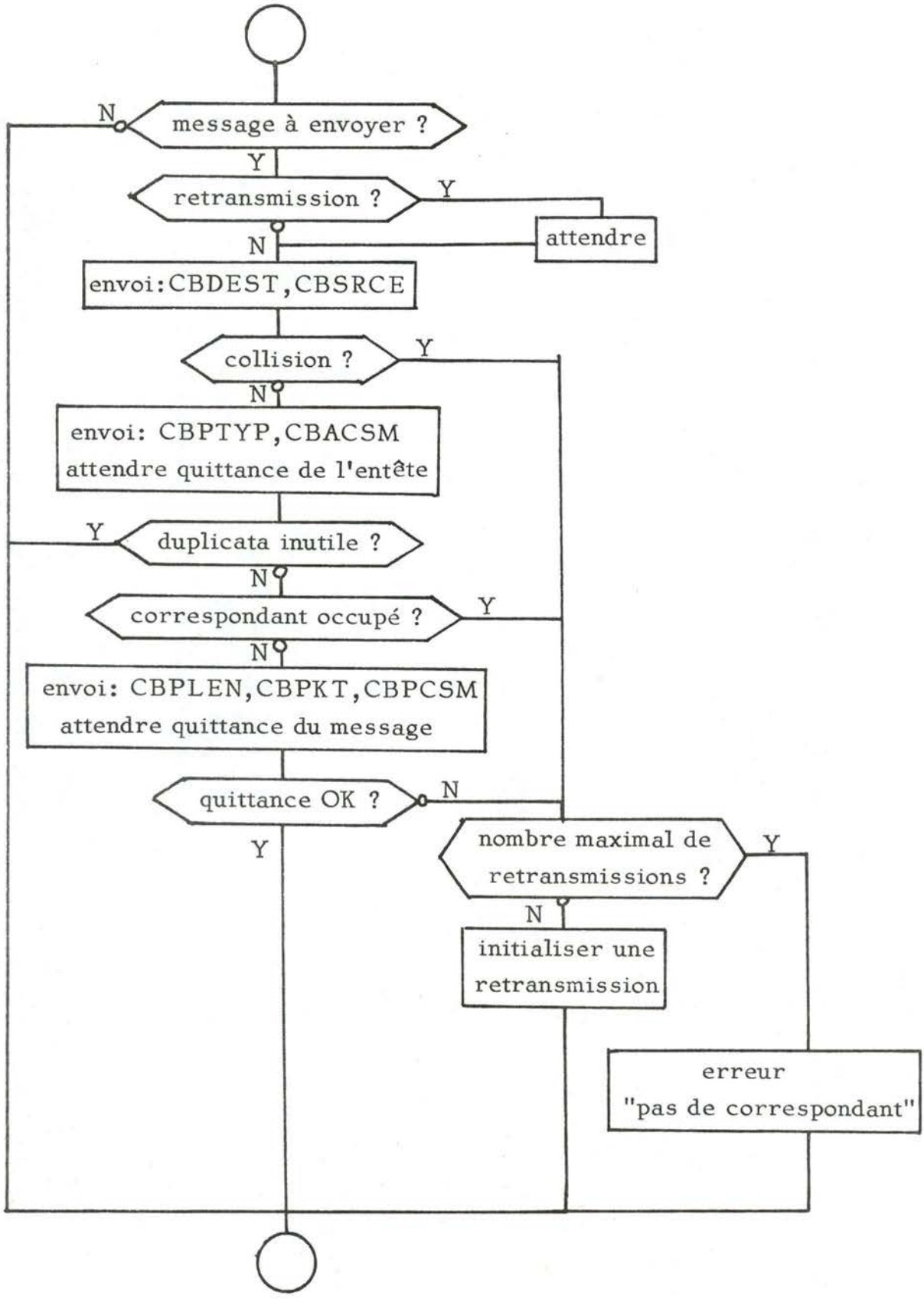


FIG.17

ROUTINE CBSND

II.5.2. RECEPTION.

Lorsqu'une station veut recevoir un message par le réseau, elle appelle la routine CBRECV. Les paramètres à fournir sont:

- source attendue (0 si on veut accepter n'importe quelle source)
- destination (adresse locale)
- adresse d'une zone mémoire pour mettre les données.

Cette routine garnit une table d'entrée avec ces informations. Cette table contient aussi une demande de réception et un indicateur de fin de réception. En attendant, cette routine ne teste plus que cet indicateur. Le passage d'un message sur le bus provoque, par l'intermédiaire du détecteur de porteuse, une interruption qui va démarrer l'exécution de la routine CBINT. Celle-ci regarde si le bus est libre ou actif. Dans le premier cas, CBINT branche à la routine CBSND, et dans le second cas à CBRCVE. Cette routine décrite par l'organigramme de la FIG. 18 s'occupe de recevoir le message selon le protocole.

Normalement, dès que le détecteur de porteuse trouve le bus occupé, il bloque le dispositif d'émission de la station. Mais le firmware peut débloquer ce dispositif. Plus généralement, l'interface côté réseau peut être mis en mode réception ou émission par programmation.

La routine CBRCVE garnit la table d'entrée avec les informations effectivement reçues par le message. Lorsque le message a bien été reçu, cette routine se termine en indiquant la fin de la réception du message. Comme il s'agit d'une routine d'interruption, le retour se fait à la routine CBRECV qui teste toujours cet indicateur de fin de réception. S'il y a eu une erreur de transmission quelconque, cet indicateur n'est pas mis et CBRECV attend une retransmission. Si tout s'est bien passé, la routine CBRECV fournit les paramètres de sortie suivants:

- source effective du message
- destination
- nombre de bytes de données
- adresse de la zone mémoire qui contient les données.

Comme dans toutes les stations, la retombée du détecteur de porteuse provoque une interruption qui déclenche la routine CBINT.

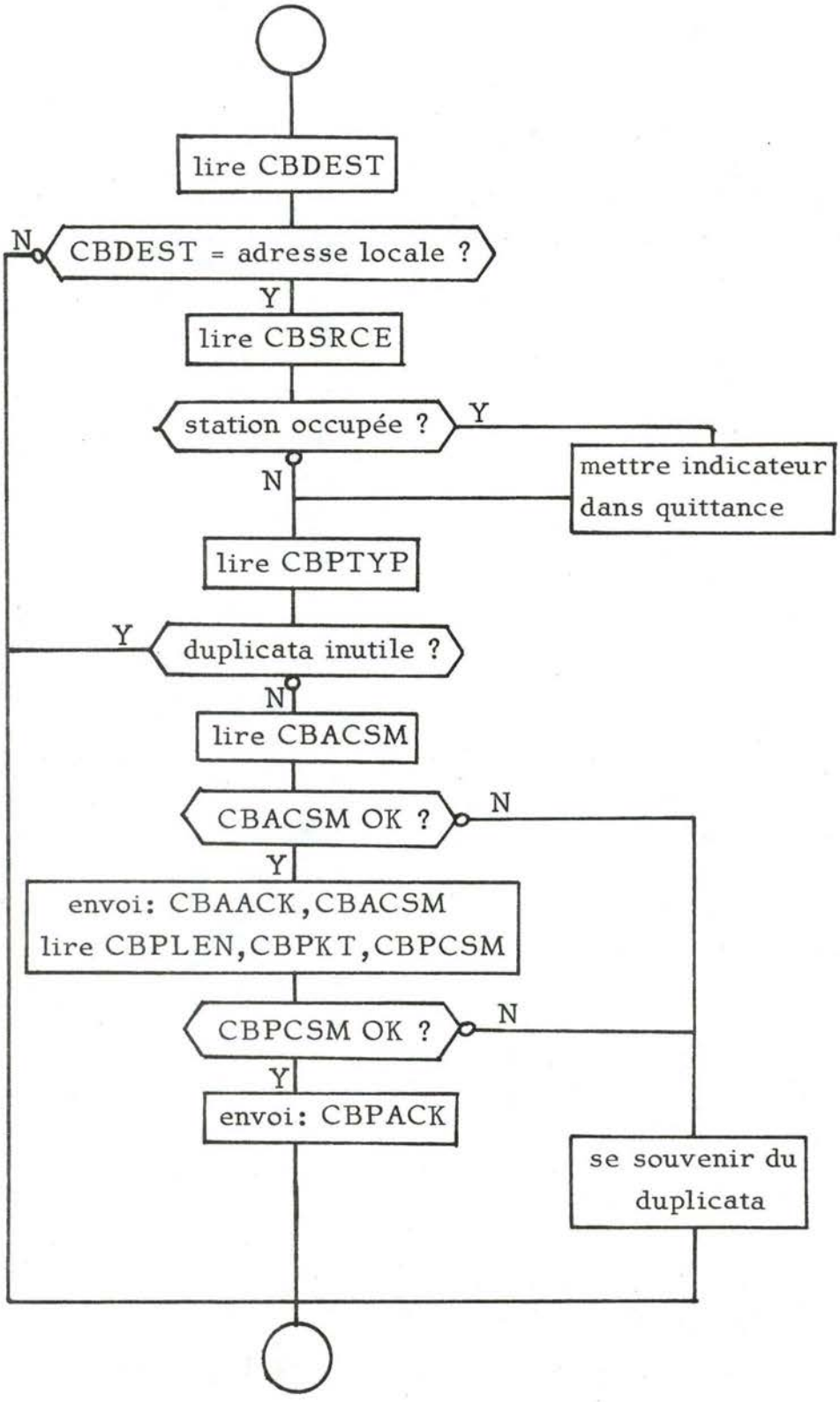


FIG.18

ROUTINE CBRCVE

II.6. LIMITES DU RESEAU LOCAL COBUS.

Le réseau local COBUS se trouve encore au stade expérimental. Des efforts continus tendent à améliorer et à faciliter l'utilisation du réseau.

Un inconvénient majeur est le fait que l'utilisateur doit connaître l'adresse physique de l'élément (processus, fichier, ressource) qu'il veut accéder. Dans un système distribué, tous les objets doivent être accessibles par leur nom symbolique sans que l'on doive s'occuper de l'endroit où ils se trouvent. Ainsi chaque station devrait tenir une liste de tous les éléments dont elle dispose. Cette méthode d'adressage symbolique comporte un risque qui provient du fait que plusieurs utilisateurs peuvent créer des objets qui ont le même nom symbolique. Alors plusieures stations vont répondre à un adressage avec ce nom symbolique et provoquer des collisions sur le bus.

Le fait que le réseau actuel ne supporte pas la diffusion de messages représente un autre inconvénient. La diffusion faciliterait largement un adressage symbolique. Mais avec le protocole actuel, cela pose des problèmes de quittance déjà décrits dans le chapitre II.4.2. . Par contre, un protocole sans quittance immédiate ne correspond plus aux objectifs de départ. Un autre problème est que la station source de la diffusion n'est jamais absolument sûre que toutes les stations vont entendre son message.

La longueur du bus du réseau COBUS est limitée à 200 mètres. D'autres réseaux locaux ont la possibilité de connecter plusieurs de ces réseaux l'un derrière l'autre pour augmenter les distances entre les stations. Ce serait aussi possible avec ce réseau-ci, mais introduirait inévitablement un stockage intermédiaire du message dans la station de raccordement entre les 2 réseaux. Ceci est contraire à la philosophie du réseau COBUS qui n'autorise aucun stockage intermédiaire pour acquitter immédiatement les messages dès qu'ils sont en mémoire de la station destination.

II.7. FONCTIONNEMENT DU RESEAU LOCAL COBUS.

Pour revenir au problème initial qui a été la mise en commun de ressources coûteuses et encombrantes, il faut rajouter des systèmes d'exploitation et des programmes utilisateurs dans les stations. Ces programmes correspondent aux niveaux 2 et 3 du modèle de Thurber.

Le réseau local COBUS actuellement exploité au LCD à l'EPFL a la configuration suivante:

- 1 station de ressources
- plusieurs terminaux intelligents.

Actuellement le réseau COBUS ne supporte pas encore plusieures stations de ressources en même temps.

II.7.1. STATION TERMINALE.

II.7.1.1. SYSTEME D'EXPLOITATION DU RESEAU LOCAL.

Chaque station terminale (ou terminal intelligent) a un système d'exploitation car elle peut être utilisée comme un petit ordinateur autonome sans être reliée au réseau local. Le système d'exploitation assure la gestion des périphériques locaux de la station terminale.

Une extension de ce système d'exploitation spécifique à la gestion du réseau comprend les routines firmware pour le contrôle du protocole. Pour permettre la manipulation de fichiers dans la station de ressources, il existe un certain nombre d'ordres primitifs. A partir de ses programmes, l'utilisateur peut demander l'exécution de ces ordres dans la station de ressources en faisant des appels standardisés au système d'exploitation. Les routines qui correspondent à ces appels envoient l'ordre à la station de ressources et demandent une réponse.

Les ordres primitifs principaux auxquels l'utilisateur a accès concernent la manipulation de fichiers comme ouverture, lecture, écriture, fermeture, changement de nom, effacement et autres. Le détail de ces ordres sera donné dans la division III.2.2.1. .

L'ensemble du système d'exploitation et son extension réseau forment le niveau 2 de communication entre processus du modèle de Thurber.

II.7.1.2. PROGRAMMES UTILISATEURS.

Un utilisateur a 2 méthodes pour utiliser les moyens de la station de ressources. Il peut introduire des commandes au clavier et il peut faire des appels au système d'exploitation à partir de ses programmes.

Dans le premier cas, un programme d'interprétation de commandes est transféré à partir du miniordinateur dans le terminal. Cet interprèteur encore appelé CLI (Command Line Interpreter) possède une liste de commandes qu'il peut traiter lui-même. Cela veut dire qu'il peut décomposer une telle commande en une séquence d'ordres primitifs (voir II.7.1.1.). Lorsque le CLI local découvre qu'il ne peut pas décomposer une commande, alors il l'envoie textuellement au miniordinateur en lui demandant d'exécuter cette commande et de renvoyer la réponse au terminal intelligent. S'il y a eu une erreur quelconque, le CLI affiche le code de l'erreur sur l'écran et attend la commande suivante. Le fonctionnement détaillé du CLI sera décrit dans la division III.2.2.2. .

L'utilisateur peut aussi écrire des programmes qui vont s'exécuter dans le terminal intelligent. Il se peut que ces programmes veulent traiter des fichiers qui se trouvent dans le miniordinateur. Alors le programmeur fait directement appel aux ordres primitifs (voir II.7.1.1.).

Le CLI et les programmes utilisateurs forment le niveau 3 de l'utilisateur dans le modèle de Thurber.

II.7.2. STATION DE RESSOURCES.

La structure de la station de ressources est hiérarchisée. Un processeur de protocole sur l'interface intelligent effectue les demandes d'émission et de réception venant du processeur central de la station. Pour simplifier l'explication générale du fonctionnement d'une station de ressources, nous faisons abstraction du processeur de DMA sur l'interface intelligent pour miniordinateur (voir II.3.2.3.).

La séparation entre les niveaux 2 et 3 de Thurber peut se faire selon les fonctions des programmes qui s'exécutent dans les 2 processeurs. Les programmes dans l'interface intelligent correspondent au niveau de communication entre processus, et ceux dans le miniordinateur forment le niveau utilisateur.

II.7.2.1. SYSTEME D'EXPLOITATION DU RESEAU LOCAL.

Outre les routines de contrôle des transmissions selon le protocole (voir II.5.), le microprocesseur de l'interface intelligent effectue la gestion des transmissions. Sur une interruption venant du processeur central, le microprocesseur va voir en mémoire centrale s'il y a des émissions ou des réceptions à faire. A la fin de l'action, le processeur de protocole le signale au processeur central à l'aide d'une interruption.

Il ne s'agit pas d'un système d'exploitation dans le sens usuel du terme, mais d'un programme qui assure la synchronisation avec le processeur central, et qui gère les transmissions demandées par le processeur central.

Ce programme peut être assimilé au niveau du système d'exploitation pour la communication entre processus du modèle de Thurber.

II.7.2.2. PROGRAMMES UTILISATEURS.

Dans le miniordinateur de la station de ressources, peuvent se dérouler 2 programmes simultanément (voir III.2.2.3.). Ces 2 programmes traitent les 2 sortes de commandes envoyées par les stations terminales (voir II.7.1.2). Le programme le plus prioritaire fait la gestion des messages reçus du réseau. Il assure aussi la synchronisation avec le programme de l'interface intelligent. C'est aussi le programme prioritaire qui effectue les ordres primitifs (voir III.2.2.1.) de manipulation de fichiers. L'autre sorte de commandes est envoyée au programme moins prioritaire qui les effectue en mode batch. Les réponses à ces commandes passent de nouveau au programme prioritaire qui demande à l'interface intelligent de les renvoyer aux terminaux respectifs.

Ces 2 programmes forment le niveau utilisateur du modèle de Thurber.

II.8. APPLICATIONS DU RESEAU LOCAL COBUS.

Outre les manipulations de fichiers, le réseau local COBUS actuellement exploité au LCD sert à la mise au point de programmes utilisateurs.

D'abord l'utilisateur doit créer un fichier et y mettre les instructions en langage d'assemblage qui constituent son programme. Cela se fait par l'intermédiaire d'un programme d'édition. Dans le réseau COBUS, l'utilisateur appelle cet éditeur à son terminal. L'éditeur est alors transféré du miniordinateur dans le terminal intelligent et commence son exécution. L'édition du programme source de l'utilisateur se fait donc localement sans encombrer le miniordinateur. Lorsque l'utilisateur a fini d'introduire son programme, celui-ci est transféré dans la mémoire de masse du miniordinateur. L'éditeur devient inutile, et le CLI local est rechargé dans le terminal.

Puis l'utilisateur demande à travers le CLI l'assemblage de son programme dans le miniordinateur. Il s'agit ici d'un cross-assemblage, car le programme objet qui en résultera va s'exécuter dans le terminal et non dans le miniordinateur. Lorsque le programme de cross-assemblage a constaté des erreurs dans le programme de l'utilisateur, celui-ci peut de nouveau éditer son programme pour corriger les erreurs. Après un assemblage correct, l'utilisateur demande l'exécution de son programme dans le terminal pour en tester le déroulement.

Une méthode de dépannage consiste à charger un programme appelé cross-moniteur dans un second terminal du réseau et de contrôler complètement le déroulement du programme à tester dans l'autre terminal à travers le réseau. Le cross-moniteur peut faire exécuter le programme utilisateur instruction par instruction ou l'arrêter en des points d'arrêt (breakpoints). L'utilisateur peut ainsi inspecter des zones mémoire et en changer le contenu, puis faire continuer l'exécution de son programme.

Finalement l'utilisateur peut demander l'impression de n'importe quel fichier sur l'imprimante du miniordinateur.

III° PARTIE

PROJET PERSONNEL

III.1. PROJET DE STAGE.

III.1.1. OBJECTIFS DU PROJET INITIAL.

Initialement le projet de stage a été défini par le texte qui se trouve dans l'annexe A de ce rapport.

L'objectif a été de connecter un microordinateur LSI-11/2 de DEC sur le réseau local COBUS déjà existant. Ce microordinateur doit gérer les ressources coûteuses et encombrantes pour les rendre accessibles aux terminaux intelligents du réseau. Ma collaboration à ce projet s'est limitée à la partie software. Mon travail a donc consisté à concevoir et écrire les programmes pour la mise au point et l'exploitation de la connexion du microordinateur sur le réseau local.

Le but final de ce projet sera atteint lorsqu'une station terminale pourra commander des transferts de fichiers entre les mémoires de masse des 2 stations de ressources (miniordinateur et microordinateur).

III.1.2. EVOLUTION DU PROJET.

Dans de longues discussions entre les collaborateurs du projet, nous avons analysé les implications des objectifs du projet sur le travail à effectuer. Jusqu'ici il y avait une seule station de ressources sur le réseau local, et tous les terminaux intelligents s'adressaient à elle. Au moment où une seconde station de ressources est connectée sur le réseau, il faut décider à laquelle des 2 station l'utilisateur veut s'adresser. De même lors d'un transfert d'un fichier entre le miniordinateur et le microordinateur, il faut pouvoir indiquer quel fichier se trouve dans quelle station. Comme le réseau local COBUS ne dispose pas encore de l'adressage symbolique (voir II.6.), cela pose des problèmes pour lesquels nous n'avons pas encore trouvé de solution satisfaisante.

Voilà les raisons pour lesquelles le projet initial a été modifié. Le second projet simplifié prévoit la construction d'un second réseau local où le microordinateur prend la place du miniordinateur. Diverses objections ont été formulées quant à l'utilité ultérieure d'un tel projet. Un usage intensif de ce réseau où la station de ressources dispose seulement d'une disquette de 240 kilobytes de mémoire de masse à la disposition de plusieures stations terminales peut justifier cette réflexion. Mais il ne faut pas oublier qu'il s'agit

d'un projet expérimental qui permettra de mettre au point un interface intelligent qui est conçu pour une large gamme de microordinateurs et miniordinateurs (le LSI-11/2 est le bas de gamme de la série des PDP-11 de DEC). D'autre part, après cette première étape, le retour au projet initial, c'est-à-dire de mettre 2 ordinateurs sur le même réseau local, sera toujours possible.

III.2. ETUDE DE L'ENVIRONNEMENT EXISTANT.

Avant de travailler sur un projet, il faut étudier les éléments auxquels on sera probablement confronté lors de la réalisation. En informatique, chaque élément a 2 côtés: un côté hardware et un côté software. Pour comprendre le fonctionnement de ces éléments en détail, il faut étudier les 2 côtés. Ceci est inévitable car souvent le hardware, le firmware et le software sont intimement liés pour obtenir le résultat voulu.

L'étude de l'environnement déjà existant est souvent longue et laborieuse, mais elle fait quand même partie du projet. Dans le cas présent, ce travail préliminaire fût assez pénible parce qu'il n'y avait pas de documentation précise et détaillée sur le fonctionnement du réseau local COBUS. Pour produire les descriptions comme elles apparaissent dans la II° partie et la suite de la III° partie de ce rapport, il fallait rassembler beaucoup de petits détails et bien comprendre le déroulement des transmissions.

Lorsqu'il s'agit maintenant de connecter un nouveau microordinateur sur le réseau, une bonne connaissance de cette machine sera utile. Pour rendre ce microordinateur compatible avec le software déjà existant, il faut connaître parfois plus de détails sur les systèmes d'exploitation qu'on n'en trouve dans les documentations.

III.2.1. HARDWARE EXISTANT.

Les éléments principaux qui entrent dans la composition du réseau local autour du microordinateur sont:

- les interfaces non intelligents côté réseau (II.3.2.1.)
- l'interface intelligent côté microordinateur (II.3.2.2.)
- les terminaux intelligents (III.2.1.1.)
- le microordinateur (III.2.1.2.).

L'étude du hardware qui existait déjà pour le réseau COBUS n'était pas explicitement requise pour mon projet, mais s'avérait quand même utile pour une bonne compréhension du fonctionnement des transmissions.

III.2.1.1. TERMINAL INTELLIGENT.

Le terminal intelligent est constitué par un petit ordinateur de table (système à microprocesseur) SMAKY6 développé au LCD à l'EPFL et construit autour d'un microprocesseur Z 80 de ZILOG. Celui-ci travaille sur des bytes de 8 bits. Il possède 2x8 registres de 8 bits dont 1 remplit la fonction d'accumulateur. 4 registres de 16 bits sont répartis comme suit: pointeur de programme, pointeur de pile et 2 autres pour le programmeur.

La structure interne du terminal intelligent se base sur un bus qui véhicule les signaux de données et de contrôle de fonctions. Un tel terminal peut fonctionner de manière autonome, et il peut se connecter sur le bus du réseau local. La configuration nécessaire pour se connecter sur le réseau COBUS est représentée à la FIG.19 .

Une telle station dispose de 4 Kbytes de ROM qui contient le système d'exploitation et l'extension réseau pour la gestion du protocole du réseau. La taille de la mémoire RAM est généralement de 32 Kbytes. Des périphériques locaux peuvent se connecter à travers les interfaces série et parallèle. La différence avec le microordinateur est que tous les composants sont implémentés sur une seule plaque à l'exception de l'interface COBUS, tandis que le microordinateur a une structure modulaire autour du bus.

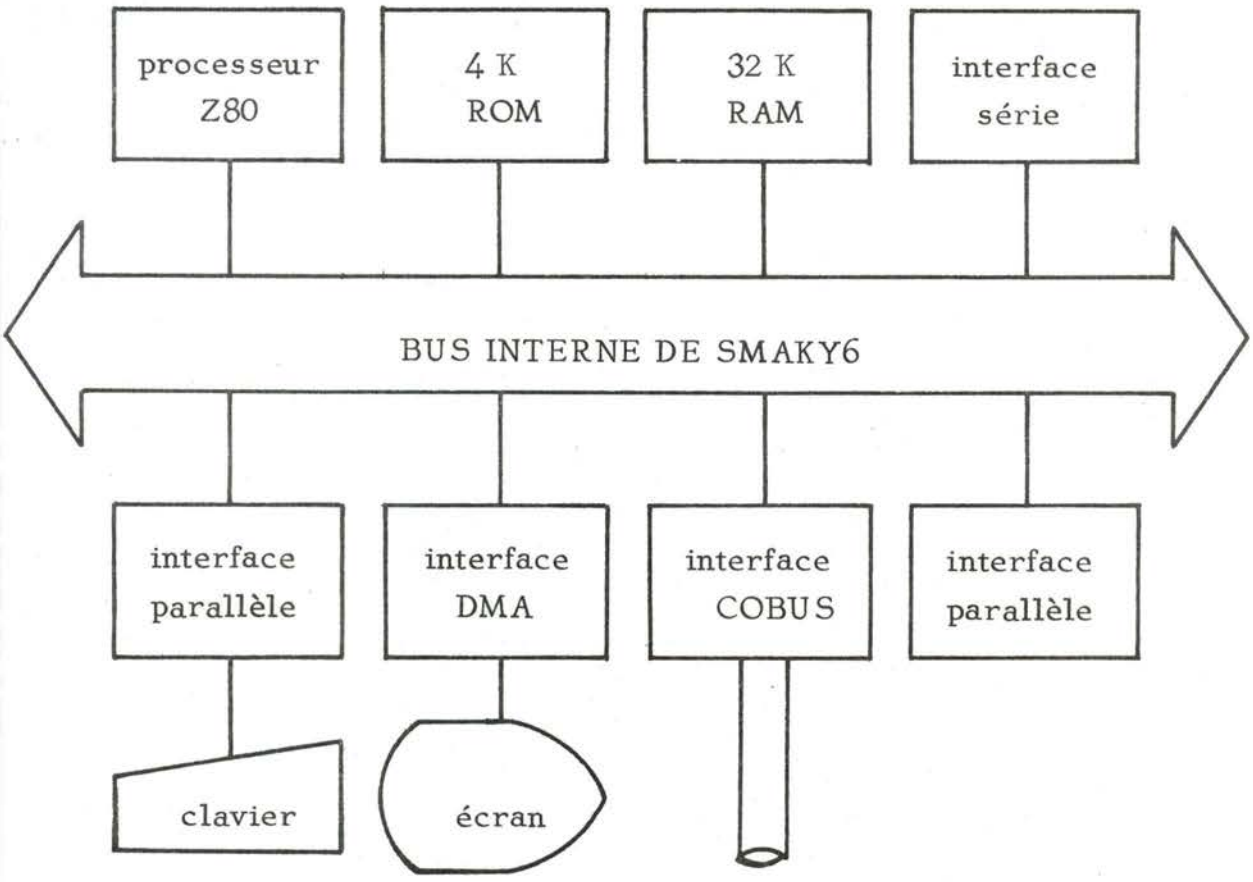


FIG.19

ORGANISATION INTERNE DU TERMINAL INTELLIGENT

III.2.1.2. MICROORDINATEUR.

Le microordinateur a une structure très analogue à celle du terminal intelligent. Les plaques de processeur, mémoire et interfaces d'entrée/sortie se connectent sur un bus interne. Une configuration typique pour le réseau COBUS est représentée à la FIG.20 .

Le processeur qui travaille sur 16 bits possède 8 registres R0 à R7. Les registres de R0 à R5 sont banalisés et à la disposition du programmeur, R6 sert de pointeur de pile et R7 de pointeur de programme.

La mémoire RAM a une taille de 28 Kmots de 16 bits qui sont aussi adressables par bytes de 8 bits. L'un des interfaces série d'entrée/sortie sert à raccorder un terminal "stupide" qui joue le rôle de console opérateur pour commander le fonctionnement du microordinateur. L'autre interface série sert à commander une imprimante. Un interface spécialisé relie le contrôleur de disquettes sur le bus. Une dernière plaque avec de la mémoire ROM contient un moniteur minimal. Sur cette plaque se trouve aussi un programme de démarrage (bootstrap) pour charger un système d'exploitation à partir d'une disquette.

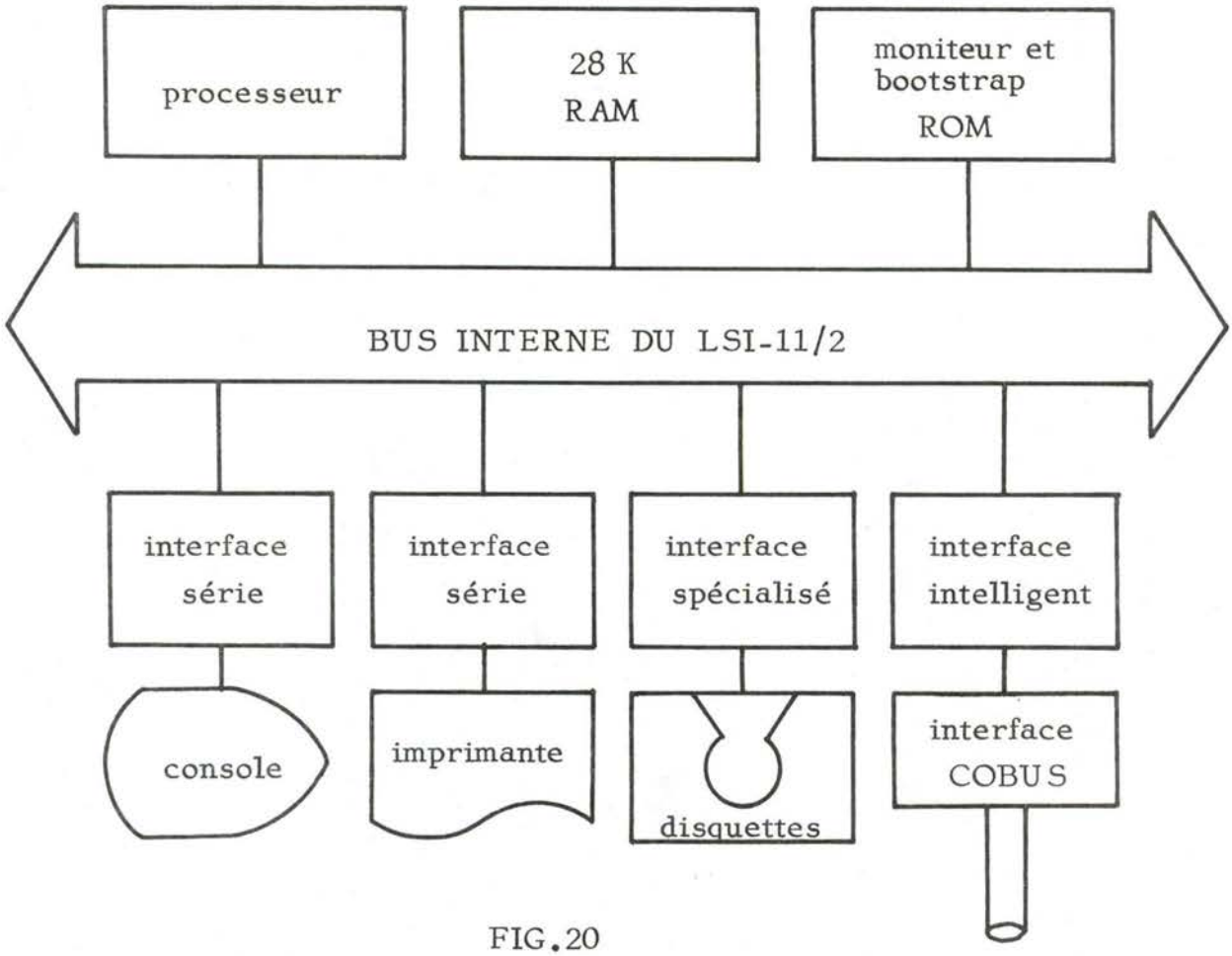


FIG.20

ORGANISATION INTERNE DU MICROORDINATEUR

L'organisation de la mémoire du microordinateur est la suivante (voir FIG.22) :

- mémoire RAM de 0 à 28K
- mots mémoire implémentés sur les interfaces d'entrée/sortie de 28K à 32K .

Les entrées/sorties se font donc avec les mêmes instructions que les lectures/écritures dans les positions mémoires.

Les interruptions sont vectorisées. Cela veut dire que chaque interface d'entrée/sortie correspond à un vecteur. Après une demande d'interruption de la part de l'interface, le processeur envoie un signal qui se propage à travers tous les interfaces (daisy-chain). L'interface concerné met alors son "vecteur" sur le bus et il est lu par le processeur. Ce vecteur pointe vers une adresse en mémoire contenant les informations pour faire exécuter la routine d'interruption associée à cet interface.

III.2.2. SOFTWARE EXISTANT.

Le réseau local COBUS qui existait déjà avant ce projet fonctionnait selon les principes expliqués dans la section II.7. . Ce réseau est construit autour d'un miniordinateur ECLIPSE de Data General comme station de ressources. Une analyse détaillée des programmes de ce réseau a permis de mieux comprendre le déroulement des transferts entre stations terminales et stations de ressources.

La décision de ne pas changer le CLI nous obligeait à adapter le système d'exploitation du microordinateur LSI-11/2 pour le réseau COBUS. Il fallait donc étudier en détail les programmes qui prennent part à une transmission pour rendre les réponses compatibles avec celles du miniordinateur ECLIPSE.

Voici les programmes qui s'exécutent dans les différents composants du réseau COBUS:

- dans la station terminale:
  - CLI ou programme utilisateur
  - routines du protocole
- dans l'interface intelligent:
  - programmes de gestion des transmissions et de synchronisation entre la station de ressources et les routines du protocole
  - routines du protocole

- dans la station de ressources:
  - programme batch qui effectue les commandes non décomposables par le CLI
  - programme de gestion et de traitement des messages qui effectue les ordres primitifs.

Les routines du protocole ont déjà été décrites dans la section II.5. . L'utilisateur a 2 méthodes pour accéder aux ressources du miniordinateur. Il peut introduire des commandes au clavier de son terminal intelligent. Le CLI local partage les commandes en commandes décomposables et commandes non décomposables. L'utilisateur peut aussi écrire des programmes qui font appel aux ordres primitifs de l'extension réseau du système d'exploitation.

III.2.2.1. ORDRES PRIMITIFS.

L'utilisateur peut faire des manipulations programmées de fichiers en faisant appel à des ordres primitifs. Voici la liste des ordres principaux auxquels le programmeur a accès:

CREATE	pour créer un nouveau fichier
DELETE	pour effacer un fichier
RENAME	pour changer le nom d'un fichier
OPEN	pour ouvrir un fichier
CLOSE	pour fermer un fichier
RESET	pour fermer tous les fichiers que cette station a ouvert
RDBYTE	pour lire un certain nombre de bytes dans un fichier
RDL	pour lire une ligne dans un fichier
WRBYTE	pour écrire un certain nombre de bytes dans un fichier
WRL	pour écrire une ligne dans un fichier
GTOD	pour obtenir l'heure
GDAY	pour obtenir la date.

La FIG.21 indique les paramètres que le programmeur doit fournir en entrée et ceux qu'il recevra en sortie. L'ordre est codé dans une routine de l'extension réseau du système d'exploitation. Le code suivi des paramètres est envoyé au programme de gestion et traitement des messages. Celui-ci correspond à une certaine adresse, et c'est le miniordinateur qui répond à cette adresse sur le réseau. Le programme de gestion et traitement des messages communique avec le programme de gestion des transmissions sur l'interface intelligent à l'aide d'une liste de blocks d'action. Ces blocks contiennent les messages et l'action à faire (voir détails dans III.4.1.1.).

Le programme de gestion et traitement des messages détecte que ce sont des ordres primitifs et les effectue. Les réponses à ces ordres sont de la forme suivante:

- OK suivi ou non des paramètres d'information
- ERREUR suivi du code d'erreur suivi ou non des informations.

Elles sont retournées au programme de gestion des transmissions à l'aide d'un block d'action. Ce programme renvoie la réponse à la station terminale. La routine de l'extension réseau correspondant à l'ordre primitif attend la réponse et fournit au programme utilisateur les paramètres de sortie si tout s'est bien passé. Dans le cas contraire, elle indique une erreur et donne le code correspondant.

APPEL	PARAMETRES D'ENTREE	PARAMETRES DE SORTIE
CREATE	nom de fichier	
DELETE	nom de fichier	
RENAME	ancien nom, nouveau nom	
OPEN	nom de fichier	canal
CLOSE	canal	
RESET		
RDBYTE	canal, nombre, zone mém.	nombre
RDL	canal, zone mémoire	nombre
WRBYTE	canal, nombre, zone mém.	nombre
WRL	canal, zone mémoire	nombre
GTOD		heures, minutes, secondes
GDAY		jour, mois, année

FIG.21

ORDRES PRIMITIFS

III.2.2.2. COMMANDES AU CLAVIER.

Le CLI local recoit les commandes de l'utilisateur à travers le clavier. Le CLI recherche alors s'il s'agit:

- d'une commande qui peut être décomposée en ordres primitifs (p. ex. transfert de fichier, effacement de fichier, concaténation de fichiers, etc.)

- d'une commande qu'il ne peut pas décomposer (p. ex. assemblage d'un fichier, impression d'un fichier, etc.)
- d'un nom de programme exécutable dans le terminal.

Lorsque la commande est décomposable en ordres primitifs, le CLI fait une suite d'appels aux routines correspondantes dans l'extension réseau du système d'exploitation. Pour les différents ordres primitifs, tout se déroule selon la procédure décrite dans la division précédente. Si une erreur non récupérable se produit, alors le CLI l'affiche sur l'écran.

Lorsque le CLI découvre que la commande n'est pas décomposable, il l'envoie textuellement vers le programme batch. Celui-ci a une adresse particulière à laquelle correspond aussi la station de ressouces (une station peut avoir plusieurs noms sur le réseau). Une fois dans la station de ressource, le programme de gestion et traitement des messages détecte qu'il s'agit d'une commande pour le programme batch. Le système d'exploitation du miniordinateur permet l'exécution simultanée de 2 programmes. Dans ce cas-ci ce sont le programme de gestion et traitement des messages et le programme batch de traitement des commandes pour le système d'exploitation (encore appelées commandes batch). Le programme de gestion et traitement des messages transmet la commande au programme batch. Celui-ci lui attribue un numéro, puis elle est interprétée et exécutée. La réponse à cette commande sera mise dans un fichier sur disque sous le nom de MESS.numéro. Dans le message qui est retourné au programme de gestion et traitement des messages se trouve un indicateur d'erreur et le numéro attribué à la commande. Ceci sera retourné au CLI du terminal. Ce dernier va alors demander le transfert du fichier MESS.numéro à l'aide des ordres primitifs. La réponse sera affichée sur l'écran de la station terminale,et le fichier sera effacé sur le disque du miniordinateur.

Lorsque le CLI reconnait un nom de programme, il exécute la suite d'ordres primitifs nécessaires pour transférer le programme objet dans la station terminale. Après le transfert, le CLI démarre l'exécution de ce programme.

III.2.2.3. SYSTEME D'EXPLOITATION DU MICROORDINATEUR.

A part le déroulement des programmes dans le réseau COBUS déjà existant, il faut aussi analyser le système d'exploitation RT-11 du microordinateur. Dans le cadre de mon projet, il s'agit de voir comment RT-11 peut être adapté pour traiter les messages venant du réseau COBUS. Il doit aussi reconnaître le nouvel interface intelligent en mode DMA.

La partie principale du système d'exploitation, qui est appelée moniteur par DEC est implémenté de façon modulaire. Les différents modules sont:

RMON moniteur résident en mémoire centrale et qui contient les tables du système et tous les appels système passent par lui

USR ensemble de routines de services à l'utilisateur qui s'occupent des manipulations de fichiers

KMON moniteur de console opérateur qui assure la communication entre l'utilisateur et le système d'exploitation.

La division de la mémoire centrale de 28 Kmots est indiquée à la FIG.22.

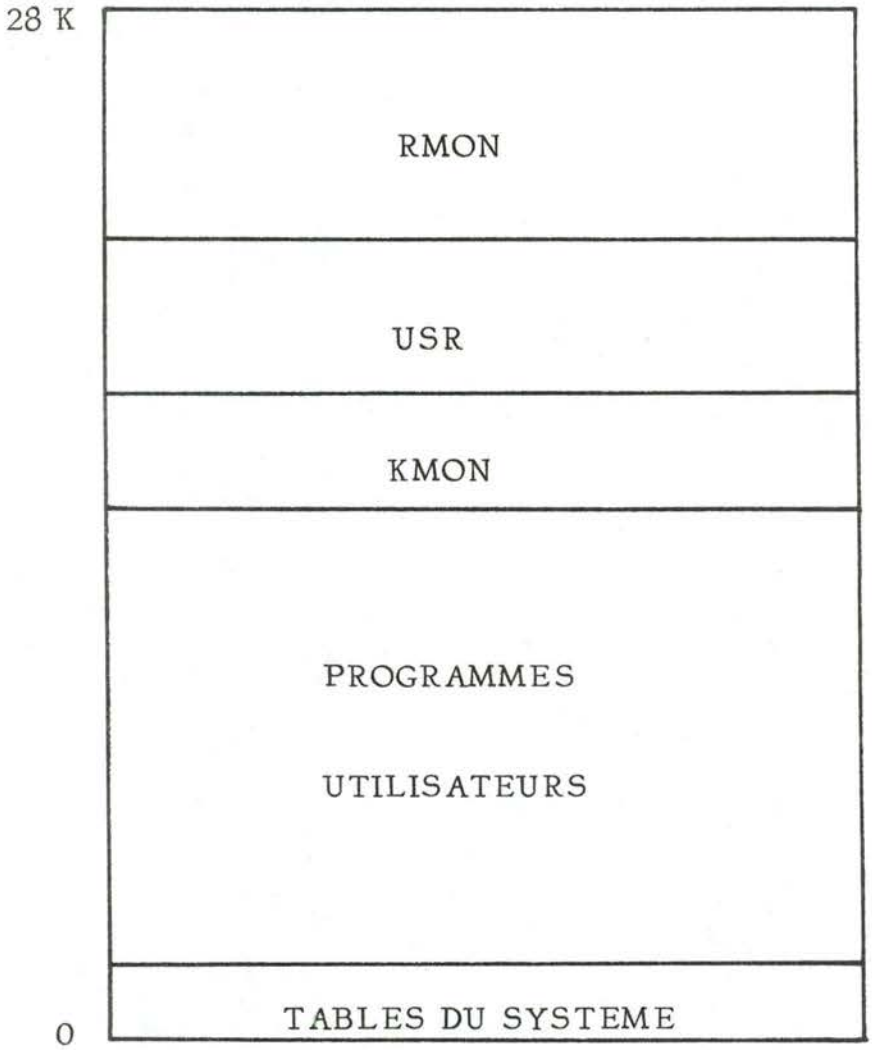


FIG. 22

ORGANISATION DE LA MEMOIRE DU MICROORDINATEUR

Les périphériques correspondent à des mots mémoire adressables par toutes les instructions du langage d'assemblage. Chaque périphérique est contrôlé par une routine d'interruption appelée "handler". Ceux-ci prennent en charge les demandes d'entrées/sorties. Le programmeur peut faire des demandes de service par des appels système (program requests) qui sont reconnus par RMON. Ainsi il existe des appels système pour les communications entre les programmes utilisateurs, pour les manipulations de fichiers et pour accéder aux variables du système.

Deux programmes utilisateurs peuvent s'exécuter simultanément dans 2 environnements distincts. Ces environnements s'appellent Foreground (FG) et Background (BG). Le programme qui se déroule dans le FG est prioritaire par rapport à celui du BG.

Le système d'exploitation possède un programme qui permet de faire exécuter une suite de commandes en BG en mode batch. Les commandes sont prises dans un fichier sur disquette et les réponses sont mises dans un autre fichier.

Une description plus détaillée du système d'exploitation RT-11 se trouve dans DEC77.

III.3. DEROULEMENT DU PROJET.

III.3.1. PLANIFICATION.

La planification des différentes étapes établie au début du projet était le suivant:

1. Etude du réseau local existant
2. Définition de l'implémentation du nouveau microordinateur sur le réseau
3. Implémentation (programmation et correction).

D'abord il fallait rassembler des documents sur le réseau local COBUS pour comprendre son fonctionnement. Ce travail formait la 1° étape de la planification établie.

Entre la 1° et la 2° étape s'est intercalée la conception et l'écriture de programmes de test pour différentes fonctions de l'interface intelligent conçu et réalisé par Mr. René SOMMER.

La 2° étape nous obligeait à rechercher dans les listings des détails sur le contenu des différents messages pour pouvoir les traiter de manière adéquate. L'étude du système d'exploitation RT-11 du LSI-11/2 était difficile à cause du manque de renseignements détaillés pour notre application particulière.

Finalement la 3° étape consistait à écrire et à tester les programmes pour le microordinateur et pour l'interface intelligent. Après les tests séparés de ces programmes, nous avons fait les premiers essais de transmission entre un terminal et le microordinateur. Le dépannage ultérieur se faisait à l'aide de moyens hardware et software.

Au fur et à mesure de l'avancement du projet, la confrontation de mes idées personnelles avec la réalité a conduit à plusieures itérations des étapes 2 et 3 de la planification initiale.

III.3.2. PROBLEMES ET SOLUTIONS.

Le système d'exploitation RT-11 est conçu pour 1 seul utilisateur. Le premier problème était de gardes actif un programme batch pour traiter toutes les commandes du RT-11 et le réveiller lorsqu'une telle commande arrivait d'un terminal. L'idée principale de la solution est de faire la même chose que le programme batch (III.2.2.3.) fourni par le constructeur.

Celui-ci possède des routines d'entrée/sortie spéciales pour communiquer avec KMON. La partie résidente du système d'exploitation contient une table avec les adresses des routines qui font communiquer KMON normalement avec la console de l'opérateur. En remplaçant ces adresses par celles de mes propres routines de communication avec KMON, nous pouvons lui communiquer les commandes venues des terminaux. Le programme de BG qui contient ces routines doit faire le remplacement de ces adresses dans la table du système et puis il doit se terminer pour passer la main au KMON. Mais alors le programme BG n'est plus actif et il en va de même des routines de communication avec KMON. Donc à la première occasion où un autre programme de traitement (p.ex. assembleur, etc.) a besoin de place mémoire, les routines de communications sont détruites. Il faut donc trouver un endroit en mémoire qui n'est pas effacé par superposition d'un autre programme. Comme les programmes FG sont résidents en mémoire centrale tant qu'ils sont actifs, ils ne sont pas effacés. L'échange des adresses dans la table du système et les routines de communication avec KMON se trouvent donc ensemble avec le programme FG.

Mes idées personnelles sur le traitement des messages et sur le déroulement d'un dialogue se sont heurtées à la décision de ne pas changer le CLI local. Ainsi par exemple, la réponse à une commande batch est mise dans un fichier sur disque par le miniordinateur et le CLI demande le transfert de ce fichier dans le terminal. Mon idée personnelle était de ne pas stocker la réponse sur les disquettes, mais de l'envoyer directement vers la station terminale. Le problème résulte du fait que le CLI local ne tient alors pas compte de cette réponse et demande quand même le transfert du fichier réponse, mais qui n'existe pas sur les disquettes du microordinateur.

Un autre problème est que le système d'exploitation du miniordinateur permet de lire un nombre quelconque de bytes sur le disque, tandis que celui du microordinateur ne peut lire que des blocks de longueur fixe. Il fallait donc des zones tampon pour faire un stockage intermédiaire des bytes lus en trop. Une telle zone était nécessaire pour chaque fichier ouvert. L'organisation du système d'exploitation d'une part et du CLI d'autre part nous obligeait à garder encore d'autres informations pour lier un terminal à un fichier. La solution consiste à créer une zone tampon par canal logique d'entrée/sortie pour la disquette. L'association de la zone tampon avec une zone pour les informations nécessaires forme un block de fichier (file block). Une description détaillée de ces blocks de fichier se trouve dans III.4.1.1. .

A cause des particularités internes du système d'exploitation RT-11 de longues routines de conversion sont parfois nécessaires pour rendre les appels système des 2 ordinateurs compatibles.

Un dernier problème survenu lors du dépannage tient au fait que le contrôle (monitoring) n'est pas possible pour les programmes qui se trouvent dans une mémoire ROM. Pour pallier cet inconvénient, une partie des programmes firmware sont chargés en mémoire RAM, et un programme cross-moniteur les contrôle à partir d'un terminal intelligent autonome (FIG.23). Grâce à ce contrôle intensif et continu des programmes, nous avons pu découvrir encore des défauts dans la méthode de synchronisation et de communication entre les 2 programmes. Au fur et à mesure des tests, ces défauts ont été corrigés et maintenant les programmes fonctionnent de manière satisfaisante.

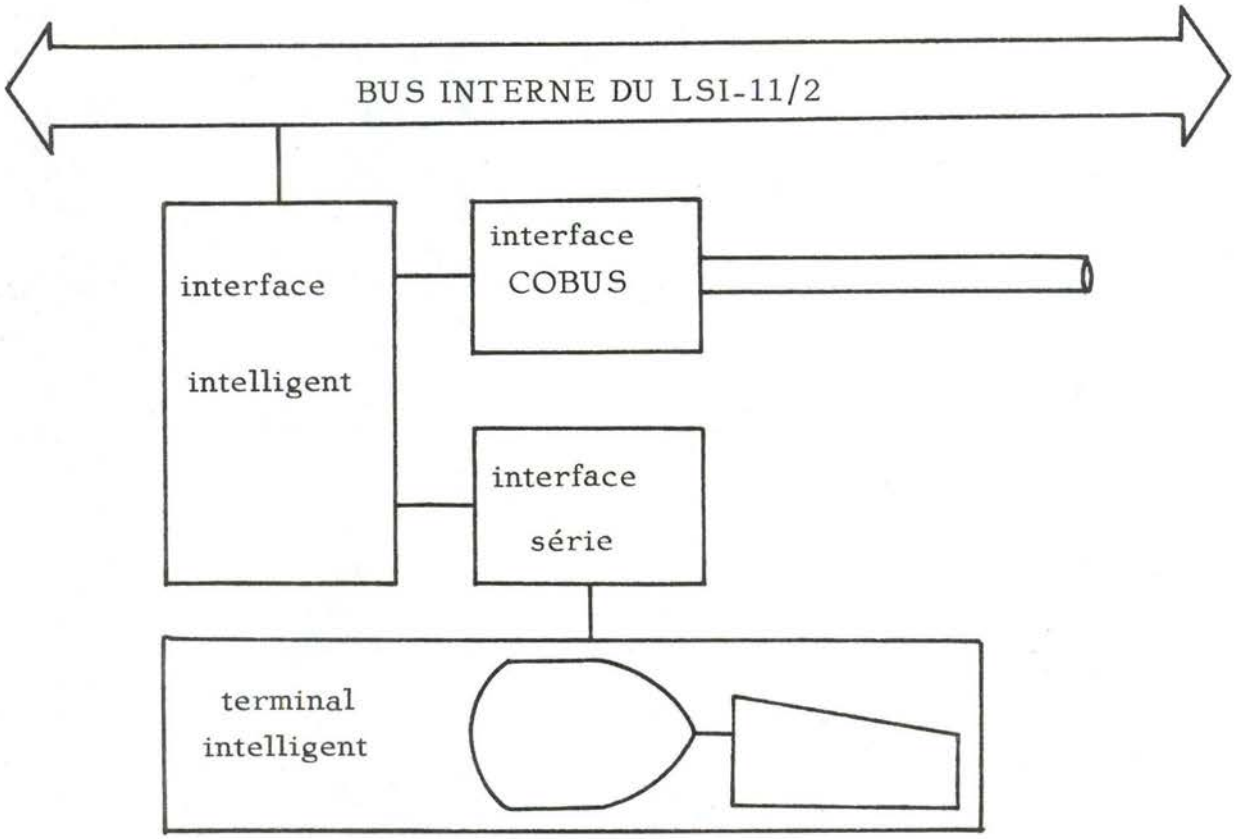


FIG.23

DEPANNAGE DES PROGRAMMES

III.4. FONCTIONNEMENT DU RESEAU LOCAL COBUS AVEC UN MICROORDINATEUR.

Mon apport personnel à ce projet est constitué par les programmes de gestion et traitement des messages pour le microordinateur et le programme de gestion des transmissions pour l'interface intelligent.

Le fonctionnement reste le même que dans le réseau local COBUS avec un miniordinateur décrit dans le chapitre III.2.2. . Les messages venant du réseau sont reçus par le programme de gestion des transmissions qui les met en mémoire du microordinateur. C'est le programme de gestion et traitement des messages qui dirige les transmissions en demandant à l'interface intelligent de faire des émissions ou des réceptions. Les relations entre les 2 programmes sont dirigées par le même mécanisme que dans la station miniordinateur. Ce sont des blocks d'action qui passent les informations nécessaires entre les 2 programmes. Une description détaillée du contenu de ces blocks d'action se trouve dans III.4.1.1..

III.4.1. PROGRAMME DE GESTION ET TRAITEMENT DES MESSAGES.

Ce programme est exécuté par le microordinateur LSI-11/2 et il est donc écrit en langage d'assemblage MACRO-11. Il s'agit ici d'un sousensemble du langage d'assemblage utilisé pour la série des PDP-11 de DEC. Le langage d'assemblage MACRO-11 est décrit dans DEC77.

Il faut dire qu'il n'y a qu'un seul programme dont une partie s'exécute en FG pour la gestion et traitement des messages et une autre partie exécutée en BG pour traiter les commandes batch. Pour des raisons techniques expliquées avant (III.3.2.), les 2 programmes n'ont pas pu être séparés.

Le chemin suivi par une commande batch est le suivant: elle est reçue par le programme FG, puis communiquée au programme BG qui doit l'analyser, l'exécuter et récupérer la réponse. La partie KMON du système d'exploitation analyse et exécute les commandes batch.

Dans le programme de gestion et traitement des messages, dont le listing se trouve en annexe C, il y a les sections suivantes:

Définition des constantes
Routines d'interruption
Boucle principale
Routines d'exécution des ordres primitifs
Routines de communication avec KMON
Réservation de zones mémoire pour les variables.

III.4.1.1. PROGRAMME FG DE GESTION DES MESSAGES ET DE TRAITEMENT DES ORDRES PRIMITIFS.

Voici une description du fonctionnement du programme FG pour mettre en évidence l'enchaînement des différentes actions. Le lecteur trouve une description détaillée des différentes routines dans le listing annexé à ce rapport. Les commentaires au début de chaque routine expliquent les fonctions de celle-ci. Les paramètres d'entrée et de sortie sont aussi mentionnés au début des routines. L'organigramme de la FIG.24 indique le déroulement de la boucle principale et infinie du programme.

Le traitement des messages venants des stations terminales est axé sur deux listes. La liste des blocks d'action (action blocks) sert à la communication entre le microordinateur et l'interface intelligent qui exécute ces actions. Les blocks d'action (voir III.2.2.1.) contiennent aussi les informations nécessaires pour ne pas avoir des interactions fâcheuses entre les 2 programmes.

La structure des blocks d'action est la suivante:

- pointeur vers une zone tampon pour les données (pas utilisé ici car le tampon se trouve avec le block d'action)
- longueur du message indique le nombre de bytes de données
- code d'action représente l'action (émission ou réception) demandée à l'interface intelligent
- adresse locale du programme FG ou BG qui demande l'action
- adresse du correspondant qui envoie ou qui doit recevoir le message
- état du block peut être libre, sous le contrôle du LSI, sous le contrôle de l'interface intelligent
- indicateur pour dire que l'interface a pris cette action en charge
- zone tampon qui contient les données.

La liste des blocks de fichier (file blocks) sert au stockage intermédiaire des données lors d'une lecture ou écriture sur disquette. Les blocks de fichier contiennent tous les renseignements pour garder une relation entre le canal, le fichier et la station terminale.

La structure des blocks de fichier est la suivante:

- numéro de canal auquel ce block est associé
- état de ce canal (libre, occupé)
- nom du fichier lié à ce canal
- adresse du terminal qui travaille avec ce canal sur ce fichier
- indicateur de fichier nouvellement créé (voir commentaire de la routine CREATE dans le listing en annexe C)

- mode d'ouverture du fichier (lecture, écriture)
- pointeur vers le dernier caractère traité (envoyé, reçu) dans la zone tampon
- zone tampon de 512 bytes.

L'enchaînement des traitements des messages est assuré par une boucle infinie. Avant d'entrer dans cette boucle, il y a une initialisation à faire. Celle-ci consiste essentiellement à remplacer les adresses des routines de communication avec le KMON dans la table du système, à garnir le vecteur d'interruption de l'interface intelligent avec l'adresse de la routine d'interruption correspondante et à interrompre l'interface intelligent pour faire transférer des informations sur la liste des blocks d'action.

La boucle principale est décrite par l'organigramme de la FIG.24. La partie de la recherche des blocks d'action à émettre à la fin de la boucle n'est peut-être pas indispensable, mais elle représente une sécurité au cas où l'interface intelligent n'aurait pas réagi à une interruption de la part du microordinateur. Pour permettre au KMON de se dérouler dans le BG, la boucle infinie du FG se met de temps en temps en suspension. Le réveil se fait alors par les 2 routines d'interruption et de complétion. Chaque fois que l'interface intelligent interrompt le LSI-11/2, cela provoque le réveil de la boucle infinie, si elle était endormie. La routine de complétion se déroule chaque fois quand le programme BG envoie une réponse au FG. Cette routine transfère la réponse dans le block d'action qui contenait la commande, demande l'émission de ce message et réveille la boucle principale.

Les routines de traitement des ordres primitifs effectuent ceux-ci à l'aide des appels système du système d'exploitation RT-11, mais en les rendant compatibles à ceux du miniordinateur. Cela est rendu nécessaire par le fait que le CLI local s'attend à recevoir les mêmes résultats des 2 ordinateurs. Certains ordres primitifs présents pour le miniordinateur (p.ex. changements d'attributs comme modes d'ouverture possibles d'un fichier, etc.) n'ont pas de signification pour le système d'exploitation du microordinateur. Ces ordres ne provoquent pas d'action, mais la réponse attendue est envoyée au CLI.

III.4.1.2. PROGRAMME BG DE TRAITEMENT DES COMMANDES BATCH.

Dans le BG, c'est le programme KMON qui dirige les opérations. En fonctionnement normal, il fait ses communications avec la console de l'opérateur par 4 routines d'entrée/sortie. Celles-ci ont maintenant été remplacées par des routines de communication avec le programme BG.

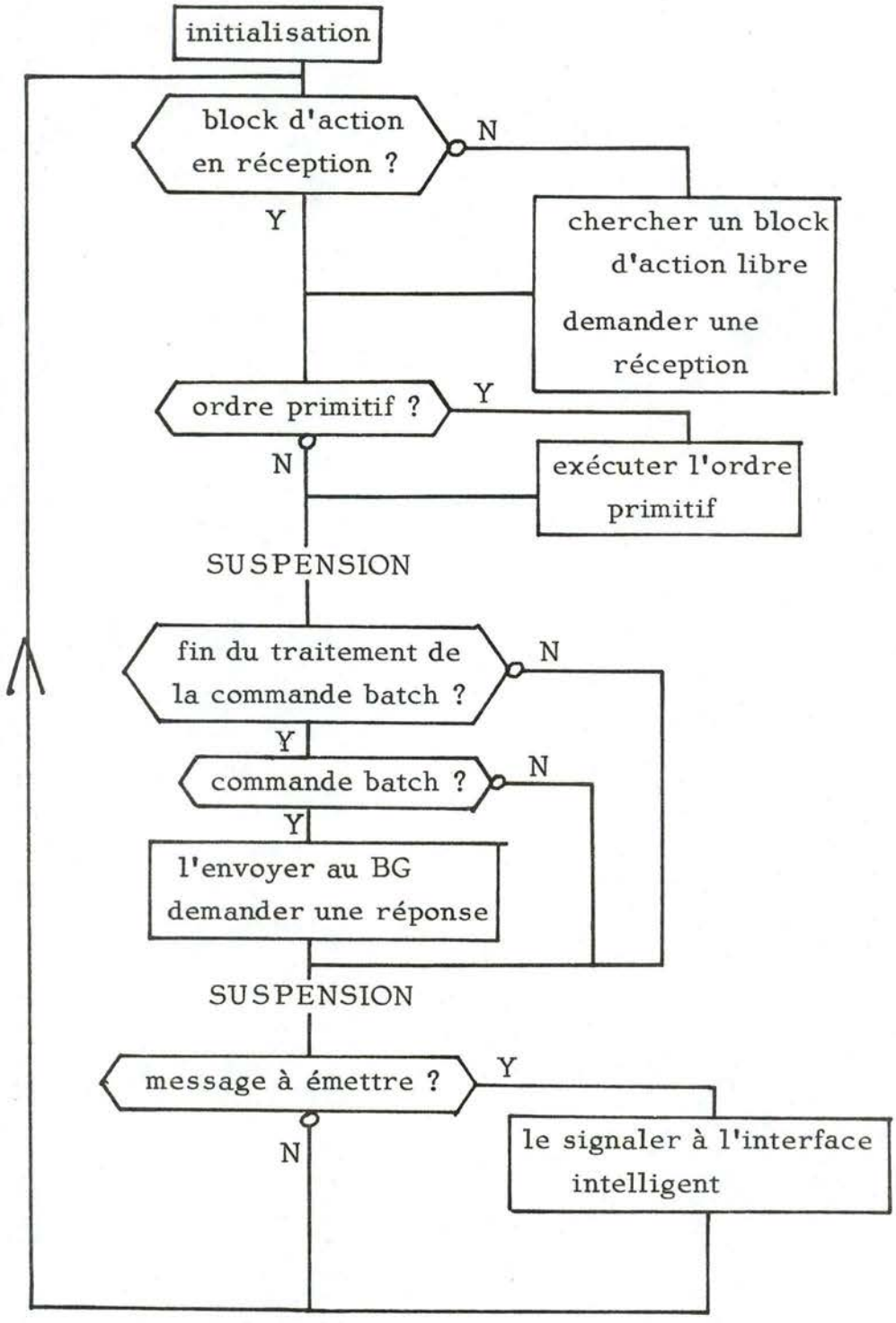


FIG.24

BOUCLE PRINCIPALE DU PROGRAMME FG

Les commentaires qui se trouvent au début des différentes routines dans l'annexe C indiquent les fonctions de celles-ci. Le KMON demande d'abord une commande à la routine TTYIN. Cette commande lui est transmise caractère par caractère jusqu'au premier retour de chariot (carriage return) qui indique la fin de la ligne. La réponse est rendue par la routine PRINT et est terminée par un point qui est rendu à travers la routine TTYOUT. A la fin d'une réponse, celle-ci est écrite dans un fichier-réponse sur la disquette, et le programme FG est averti de la fin du traitement. Puis la commande suivante est transmise au programme BG.

La première fois que KMON demande une commande, la routine TTYIN doit d'abord en demander une au FG. La routine GETCOM du BG reçoit les commandes venant du programme FG, leur attribue un numéro de 0 à 7 de manière circulaire. Le CLI de la station terminale doit donc lire le fichier-réponse avant que celui-ci ne soit écrasé par la suite des commandes. Cela empêche que ces fichiers-réponse encombrent la disquette trop longtemps.

III.4.2. PROGRAMME DE GESTION DES TRANSMISSIONS.

Ce programme se trouve en mémoire ROM de l'interface intelligent. Il est exécuté par un microprocesseur Z 80 de ZILOG et est écrit en langage CALM (Common Assembly Language for Microprocessors) qui a été élaboré au LCD à l'EPFL. Ce langage est décrit dans le rapport NICOUD76.

La structure de ce programme, dont le listing se trouve en annexe B, est décrite par l'organigramme de la FIG.25. L'initialisation n'est exécutée qu'une seule fois au démarrage du programme. Les premières instructions servent à l'adressage spécial de la mémoire locale de l'interface intelligent. L'adresse de la routine d'interruption est donnée par la valeur se trouvant à l'adresse calculée avec le contenu du registre d'interruption I augmenté par le vecteur d'interruption donné par l'interface. Tous ces calculs sont faits par le microprocesseur. Une interruption venant du LSI-11/2 fait démarrer l'initialisation des registres de l'interface COBUS, la mise en place des adresses des programmes FG et BG auxquelles la station doit répondre sur le réseau et finalement le transfert des informations sur la liste des blocks d'action dans la mémoire locale.

Le programme de gestion des transmissions attend soit une interruption du LSI-11/2, soit une arrivée d'un message. Dans le deuxième cas, les informations reçues comme paramètres de la routine CBRECV sont mises dans le block d'action libre indiqué par le microordinateur. Une interruption signale au programme de gestion et traitement des messages du FG

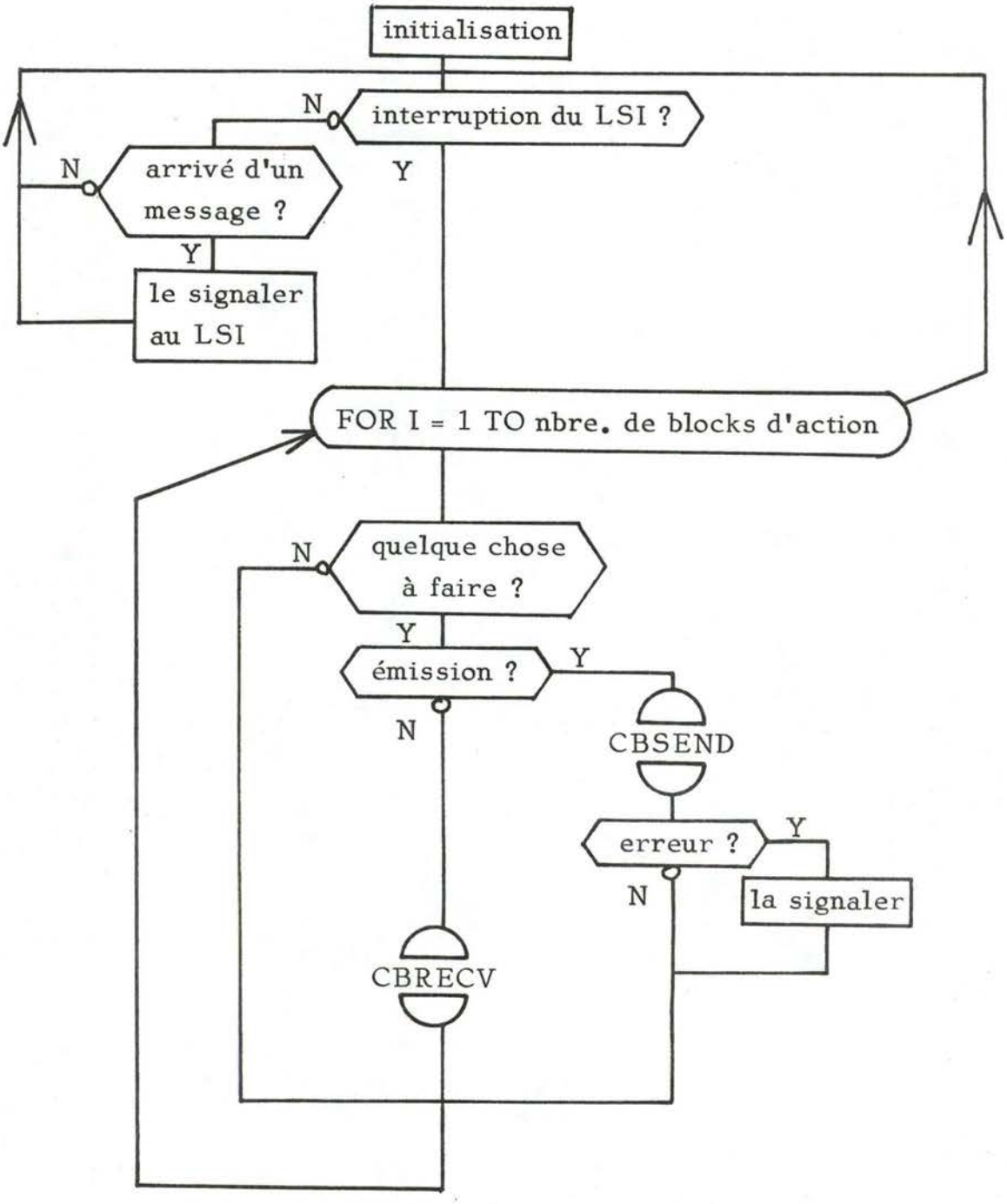


FIG. 25

PROGRAMME DE GESTION DES TRANSMISSIONS

qu'une action a été effectuée. Dans le cas d'une interruption venant du LSI-11/2, la liste des blocks d'action est parcourue une fois pour trouver les actions demandées par le microordinateur. Pour chaque demande d'action, le programme branche à la routine correspondante. Les 2 routines garnissent d'abord les paramètres d'entrée nécessaires aux routines CBSEND ou CBRECV respectivement. Ces routines de protocole ont été écrites par Mr. René SOMMER, et elles ont été détaillées dans la section II.5.. Au cas où il s'agit d'une réception d'un message, la recherche de travail dans les blocks d'action continue pendant l'attente de l'arrivée d'un message. A la fin d'une émission d'un message demandée par le LSI-11/2, un indicateur d'erreur est mis dans le block d'action. Une interruption de l'interface intelligent vers le LSI-11/2 signale la fin de l'émission. Le reste de la liste des blocks d'action est parcouru à la recherche d'autres demandes d'action avant de revenir pour attendre de nouveau une interruption du LSI-11/2 ou l'arrivée d'un message.

La routine qui traite les interruptions venant du LSI-11/2 met seulement un indicateur qui est testé par la boucle principale après avoir parcouru une fois la liste des blocks d'action.

III.5. SITUATION ACTUELLE DU PROJET.

Les deux programmes décrits précédemment représentent la dernière version testée des programmes d'exploitation pour le réseau local.

Ces tests ont été effectués avec 3 stations terminales qui utilisaient les ressources gérées par le microordinateur. Les 3 terminaux intelligents peuvent demander toutes les opérations qu'ils désirent et en même temps. Presque tous les ordres primitifs de manipulation de fichiers peuvent être exécutés sans difficulté. En effet, au dernier moment, nous avons découvert une situation qui pose des problèmes. Comme le système d'exploitation RT-11 du microordinateur ne présente aucun moyen simple de détecter si un fichier de même nom est déjà créé, mais pas encore fermé, il se peut que deux stations créent des fichiers de même nom. Alors dès qu'une des stations ferme son fichier, il est fermé pour les autres aussi. Ceci résulte en une situation inacceptable qui peut être évitée par une routine de recherche de fichiers créés et non encore fermés. D'autres ordres primitifs sont seulement implémentés de manière provisoire, comme la date et l'heure, mais ils ne représentent pas des fonctions essentielles.

Les commandes batch qui sont toutes celles reconnues par le KMON du système d'exploitation du microordinateur peuvent être exécutées. Comme au départ le système d'exploitation est fait pour 1 seul utilisateur, il faut interdire l'utilisation de certaines commandes lorsqu'on travaille avec plusieures stations terminales sur le réseau local. Les demandes d'assemblage d'un programme posent encore des problèmes qui viennent probablement du manque de place sur les disquettes. Le programme d'assemblage crée beaucoup de fichiers intermédiaires sur la disquette, et chacun de ces fichiers prend la moitié de la place restante. Ainsi ces fichiers n'ont souvent pas assez de place, et l'assemblage ne peut pas être effectué.

Voilà les problèmes que nous avons découverts en dernière minute, et que nous n'avions plus le temps de rectifier. Il faut dire que dans un système complètement dynamique comme un réseau, la détection des causes des erreurs n'est pas chose facile.

III.6. AMELIORATIONS ET DEVELOPPEMENTS FUTURS.

Une première amélioration consistera à éliminer les petits problèmes qui ont été décrits dans la division précédente.

Une autre amélioration consiste à introduire une meilleure gestion de l' l'enchaînement du traitement des messages. En modifiant légèrement la boucle principale et en introduisant une file d'attente pour les commandes batch, nous espérons augmenter la vitesse de traitement de tous les messages, et spécialement pour les ordres primitifs qui sont les plus fréquents. Pour obtenir ce résultat, la boucle principale devra prendre une structure analogue à celle du programme de gestion des transmissions décrit dans le chapitre III.4.2.. Le programme ainsi modifié existe déjà, mais les tests nécessaires n'ont pas encore été effectués.

Une amélioration qui ne concerne pas seulement mon projet est la rajoute d'autres ordres primitifs. Ces ordres vont permettre de positionner et de relire un pointeur n'importe où dans un fichier.

L'étape suivante du développement du réseau local sera de revenir au projet initial, c'est-à-dire, de mettre plusieurs ordinateurs comme stations de ressources en même temps sur un seul réseau local. La réalisation de cette étape est beaucoup plus délicate que les améliorations précédentes, car cela implique un grand remaniement dans la philosophie des programmes de gestion des transmissions, et cela dans toutes les stations.

Pour mettre plusieures stations de ressources sur un seul réseau, la possibilité de diffusion de messages serait la bienvenue. Elle permettrait de faciliter considérablement la recherche d'une ressource (fichier, processus, etc.) dans toutes les stations du réseau local COBUS.

BIBLIOGRAPHIE.

Voici la liste des articles et livres qui ont constitué la base de ce rapport. Tous n'ont pas pu être référés dans le texte.

ANDERSON75 G.A.Anderson, E.D.Jensen: Computer Interconnection Structures: Taxonomy, Characteristics, and Examples; Computing Surveys, december 1975.

ASHENHURST75 R.L.Ashenhurst, R.H.Vanderohe: A Hierarchical Network; Datamation, february 1975.

BINDER75 R.Binder, N.Abramson, F.Kuo, A.Okinaka, D.Wax: ALOHA packet broadcasting - A retrospect; National Computer Conference, 1975.

CARPENTER78 R.J.Carpenter, J.Sokol, R.Rosenthal: A microprocessor-based local Network Node; COMPCON 78 FALL, september 1978.

CASERO78 E.Casero, D.Paleari, R.Polillo, G.P.Rossi: Interfacing a Honeywell Level 62 host computer to an E.I.N. communications subnetwork with a microcomputer system; Euromicro Journal 4, 1978.

CHEVAUX78 P.Chevaux: DECNET Architecture; Conference on Interconnected Small Processors, Berne, november 1978.

CHOU75 W.Chou: Computer Communication Networks - The Parts make up the Whole; National Computer Conference, 1975.

CLARK78 D.A.Clark, K.T.Pogran, D.P.Reed: An Introduction to Local Area Networks; Proceedings of the IEEE Vol. 66, No. 11 november 1978.

COMPUTER77 Computer, Special Issue on Computer Networks; november 1977.

CONWAY78 J.Conway: Serial I/O thrusts Indecomp into asynchronous communications; EDN, august 1978.

DAVIES73 D.W.Davies, D.L.A.Barber: Communication Networks for Computers; John Wiley & Sons, 1973.

DEC77 Digital Equipment Corporation: Microcomputer Handbook; Digital 1977.

DEMARINES76 V.A.DeMarines, L.W.Hill: The Cable Bus in Data Communications; Datamation, august 1976.

DUMONT78 R.Dumont: Cobus; rapport interne au LCD, EPFL, septembre 1978.

DURNIAK78 A.Durniak: New Networks tie down distributed processing concepts; Electronics, december 1978.

DUTOIT75 D.Dutoit: Bus de transfert universel entre périphériques et ordinateurs; Systèmes Logiques n°6, mars 1975.

EMMONS78 W.F.Emmons: Network Control Protocols; COMPCON 78 FALL, september 1978.

FARBER75 D.J.Farber: A Ring Network; Datamation, february 1975.

FLETCHER78 J.G.Fletcher: Serial Communication Protocol simplifies data transmission and verification; Computer Design, july 1978.

FRASER75 A.G.Fraser: A Virtual Channel Network; Datamation, february 1975.

HEALEY78 M.Healey: Distributed Computing; Conference on Interconnected Small Processors, Berne, november 1978.

HERRMANN76 J.Herrmann: Flow Control in the ARPA Network; Computer Networks n°1, 1976.

KARP76 H.R.Karp: Basics of Data Communications; McGraw-Hill, 1976.

KEMEN78 H.Kemen, H.-H.Nagel: Experiences with a virtual network machine concept for an inhomogeneous local computer network; COMPCON 78 FALL, september 1978.

KLINGMAN77 E.E.Klingman: Microprocessor Systems Design; Prentice-Hall, 1977.

LAWS78 D.A.Laws, R.J.Levy: Data goes faster, farther with chips for drivers, receivers; Electronics, september 1978.

LEDERREY72 J.Léderrey: Modes d'interconnexion entre unités digitales ou périphériques; Systèmes Logiques n°4, juillet 1972.

LESEA78 A.Lesea, R.Zaks: Microprocessor Interfacing Techniques; SYBEX, 1978.

LIN78 K.Lin: Design of a Packet-Switched Micro-Subnetwork; COMPCON 78 FALL, september 1978

LIU77 M.T.Liu, C.C.Reames: Message communication protocol and operating system design for the distributed loop computer network; Proceedings of the 4th Annual Symposium on Computer Architecture, march 1977.

MANNING74 E.G.Manning, R.W.Peebles: A homogeneous network for data sharing communications; Computer Communications Network Group, University of Waterloo, march 1974.

MARTIN72 J.Martin: System Analysis for Data Transmission; Prentice-Hall, 1972.

MCKENDRY78 M.S.McKendry: The use of Monitors in Microprocessor Software Development; Euromicro Journal 4, 1978.

MEINADIER75 J.P.Meinadier: Structure et fonctionnement des ordinateurs; Larousse, 1975.

METCALFE75 R.M.Metcalfe, D.R.Boggs: Ethernet: Distributed packet Switching for Local Computer Networks; Palo Alto Research Center, november 1975.

NICOUD76 J.D.Nicoud: A Common Assembly Language; Proceedings of the 2nd EUROMICRO Conference, Venice, october 1976.

NICOUD77 J.D.Nicoud: Microprocessor ZILOG Z 80: Hardware and Software; Microscope n°9, august 1977.

NICOUD78 J.D.Nicoud, R.Sommer: Réseaux de terminaux pour traitement de texte; rapport interne au LCD, EPFL, mai 1978.

OSBORNE76 A.Osborne: An Introduction to Microcomputers,
Vol. I: Basic Concepts,
Vol. II: Some Real Products, June 1977 Revision;
SYBEX, 1976.

RALLAPALLI78 K.Rallapalli: CRC error-detection schemes ensure data accuracy; EDN, september 1978.

ROSEN73 S.Rosen, J.M.Steele: A Local Computer Network; COMPCON 73 Digest, february 1973.

ROSNER76 R.D.Rosner, B.Springer: Circuit and Packet Switching; Computer Networks n°1, 1976.

SOMMER76 R.Sommer: COBUS, A Firmware Controlled Data Transmission System; Second Symposium on Micro-Architecture, EUROMICRO, 1976.

SOMMER78 R.Sommer, J.D.Nicoud: Alternatives for Intelligent Interfaces; Conference on Interconnected Small Processors, Berne, november 1978.

SZURKOWSKI78 E.Szurkowski: A High Bandwidth Local Computer Network; COMPCON 78 FALL, september 1978.

THURBER78 K.J.Thurber: Computer Communication Techniques; 1978.

THURBER79 K.J.Thurber, H.A.Freeman: Local Computer Network Architectures; COMPCON 79 SPRING, february 1979.

WEST78 A.West: CNET - A Cheap Network for Distributed Computing; Conference on Interconnected Small Processors, Berne, november 1978.

WILLARD73 D.G.Willard: MITRIX: A Sophisticated Digital Cable Communications System; Proceedings of the National Telecommunications Conference, november 1973.

WULF75 W.Wulf, R.Levin: A Local Network; Datamation, february 1975.

ANNEXE A

PROJET DE STAGE DE M. DUMONT

Le LCD a développé un système d'interconnexion de terminaux par câble coaxial appelé COBUS, utilisant un protocole de gestion de bus permettant une liaison extrêmement fiable. Ce réseau est actuellement lié à un interface pour processeur NOVA. M. Dumont réalisera un interface entre un LSI-11 et COBUS. Ce travail impliquera les tâches suivantes :

- Etude des caractéristiques de COBUS et du protocole; écriture d'un programme simple effectuant une mesure de la charge du bus (assembleur Z80).
- Définition des fonctions de l'interface LSI-11 et établissement du schéma en collaboration avec un spécialiste hardware.
- Ecriture de programmes de test de l'interface (assembleur PDP-11).
- Modification du système d'exploitation RSX-11 pour supporter COBUS.
- Ecriture d'un système d'exploitation adapté à l'objectif visé et gérant en première priorité le disque, l'imprimante, des cross-assembleurs et des compilateurs (PASCAL, FORTRAN).

Chaque étape sera soigneusement documentée.

Référence : R. Sommer. Conférence EUROMICRO 76 (annexée).

Le 20.2.78
JDN/af

Prof. J.D. Nicoud

ANNEXE B

```
        .TITLE  LINDA

;RS 790214
;RS 790301
;LINDA FIRMWARE WITH
;COBUS PROTOCOL (R. SOMMER)
;LSI 11 SOFTWARE INTERFACE (ROLAND DUMONT)

        .Z80

;DEFINITIONS

        PCSYROM = 0
        PCSYRAM = 1

        ROMLINDA        = 160000
        RAMLINDA        = 170000

        .PC     PCSYROM
        .LOC    ROMLINDA
        .PC     PCSYRAM
        .LOC    RAMLINDA

        FSET    =       1
        FRESET  =       0


;COBUS ADDRESSES

        BATCHADDRESS    =       377
        SYSTADDRESS     =       376
        ANY     =       0

;COBUS DEFINITIONS

        DORMANT =       0
        REQ     =       1
        DONE    =       2

        AI      =       1
        AO      =       0

        HERROR  =       100

        SEEN    =       1
        NOSEEN  =       0

;LSI ACTION BLOCK ORGANIZATION

        ABLKPTR =       0
        DMESLEN =       ABLKPTR+2
        DAC     =       DMESLEN+2
        DADR    =       DAC+1
        DCOR    =       DADR+1
        DAS     =       DCOR+1
        DSEEN   =       DAS+1

        ABLEN   =       12
        ABUFLEN =       412
        NBRBLK  =       20


;LSI MEMORY LOCATIONS

        COMLEN  =       4
        COMTAB  =       174


;LINDA INTERFACE DEFINITIONS

        LINT    =       376
        ZINT    =       377

;CLOCK DEFINITIONS
```

```
CTRLR   =       374             ;CONTROL REG
DMAEN   =       1               ;DMA ENABLE
CLKEN   =       2               ;CLOCK ENABLE
CLRCLK  =       275             ;CLEAR CLOCK

        .PC     PCSYROM

        JUMP    HIGH            ;SET PC TO 160000
                                ;THIS INSTRUCTION IS IN FACT EXECUTED AT 0
HIGH:
        LOAD    A,#DMAEN
        OUT     CTRLR,A         ;ENABLE THE AUTOMATIC DMA INTO THE LSI 11
                                ;DESELECTS THE ROM FROM LOCATION 0
START:
        LOAD    SP,#STACK
        IM2
        LOAD    A,#PAG
        LOAD    I,A
        ION
WLOOP:
        LOAD    A,ST1
        COMP    A,#FRESET
        JUMP,NE WLOOP
        IOF
        CALL    CBINIT                  ;INIT COBUS INTERFACE
        LOAD    A,#CLKEN+DMAEN
        OUT     CTRLR,A
        LOAD    A,#BATCHADDRESS         ;ENTER COBUS ADDRESSES
        CALL    CBENTER                 ;IN RESOURCE TABLE
        LOAD    A,#SYSTADDRESS
        CALL    CBENTER
        LOAD    A,#FSET
        LOAD    ST1,A                   ;SET ST1 FOR FURTHER INTERRUPTS
        LOAD    BC,#COMLEN
        LOAD    DE,#COMLINDA
        LOAD    HL,COMTAB
        LDIR
        ION
        OUT     ZINT

START0:
        LOAD    A,ST1                   ;WAIT FOR INT FROM LSI
        COMP    A,#FRESET               ;BEFORE SCANNING
        JUMP,EQ SCAN                    ;YES, BEGIN SCANNING
        LOAD    IX,#CBITABLE
        TEST    (IX)+CBRSTATUS:CBDONB   ;A MESSAGE ARRIVED ?
        JUMP,EQ START0                  ;NO, GO ON WAITING
        LOAD    L,(IX)+CBRBUF           ;YES, RECONSTRUCT AB POINTER
        LOAD    H,(IX)+CBRBUF+1
        LOAD    BC,#-ABLEN
        ADD     HL,BC
        PUSH    HL
        POP     IY
        LOAD    A,(IX)+CBRDST           ;BEGIN TRANSFERRING
        LOAD    (IY)+DADR,A             ;INTO AB OF LSI
        LOAD    A,(IX)+CBRSRC
        LOAD    (IY)+DCOR,A
        LOAD    A,(IX)+CBRLEN
        LOAD    (IY)+DMESLEN,A
        LOAD    (IY)+DAS,#DONE          ;SET AB TO DONE
        LOAD    (IX)+CBRSTATUS,#DORMANT
        OUT     ZINT                    ;INTERRUPT LSI
        JUMP    START0                  ;GO ON WAITING

        .LOC    (./100+1)*100

.ASCII  /  LINDA REV 1.0 FIRMWARE      COBUS PROTOCOL      RS 790301       /

        PAG     = ./400         ;PAGE OF THE VECTORED INTERRUPTS

        .LOC    PAG*400+317

        .W      CBINT   ;SIMULTANEOUS INTERRUPT OF COBUS AND LSI
                        ;COBUS IS HIGHER PRIORITY

        .LOC    PAG*400+337
```

```
        .W      CBINT           ;COBUS INTERRUPT

        .LOC    PAG*400+357

        .W      INTLSI          ;LSI 11 INTERRUPT

        .LOC    PAG*400+377

        .W      INTRTC          ;LINE TIME CLOCK INTERRUPT

SCAN:
        LOAD    A,#FSET                 ;SET ST1 FOR FURTHER INT
        LOAD    ST1,A
        LOAD    IX,POOL
        LOAD    A,NBRBLOK
        LOAD    B,A
SCAN0:
        PUSH    BC
        LOAD    A,(IX)+DAS
        COMP    A,#REQ
        JUMP,NE NEXTAB
        LOAD    A,(IX)+DSEEN
        COMP    A,#SEEN
        JUMP,EQ NEXTAB
        LOAD    A,#SEEN
        LOAD    (IX)+DSEEN,A
        LOAD    A,(IX)+DAC
        COMP    A,#AO
        JUMP,EQ LSIOUT
LSIIN:
        LOAD    D,#ANY
        LOAD    E,#ANY
        PUSH    IX
        POP     HL
        LOAD    BC,#ABLEN
        ADD     HL,BC
        CALL    CBRECV
        JUMP    NEXTAB


LSIOUT:
        PUSH    IX
        POP     HL
        LOAD    BC,#ABLEN
        ADD     HL,BC
        LOAD    C,(IX)+DMESLEN
        LOAD    E,(IX)+DADR
        LOAD    D,(IX)+DCOR

        CALL    CBSEND

        JUMP,CS ERROR
RETOUR: LOAD    (IX)+DAS,#DONE
        LOAD    (IX)+DSEEN,#NOSEEN
        OUT     ZINT
        JUMP    NEXTAB

ERROR:  LOAD    (IX)+DAC,#HERROR
        JUMP    RETOUR


NEXTAB:
        LOAD    BC,#ABUFLEN
        ADD     IX,BC                   ;POINT NEXT AB
        POP     BC
        DECJ,NE B,SCAN0
        JUMP    START0

INTRTC:
        OUT     CLRCLK          ;CLEAR INTERRUPT FLIP-FLOP
        CALL    CBRTC           ;WORK
        ION
        RET

INTLSI:
```

```
        PUSH    AF
        LOAD    A,#FRESET
        LOAD    ST1,A           ;CLEAR ST1
        POP     AF
        OUT     LINT
        ION
        RET

        .PC     PCSYRAM

COMLINDA:
POOL:   .W      0
NBRBLOK:        .W      0


ST1:    .BYTE   FSET                    ;FLAG FOR INT WAIT
        .BLKB   100
STACK:

        DUAL    = 0
        LINDA   = 1

        .Z80

;GENERAL COBUS DRIVERS

;RS     26/10/78        (TESTED VERSION)
;RS     790212          LOCK ON TRANSMISSION CORRECTED

;TO BE USED WITH SINGLE PROCESSOR SYSTEMS AND DUAL PROCESSOR SYSTEMS

;FOLLOWING VARIABLES MUST BE SET

;       DUAL    = 0     IF USED ON A SINGLE PROCESSOR
;               = 1     IF USED ON A DUAL PROCESSOR SYSTEM (NOVCO)

;       IF DUAL = 1 THEN VARIABLE PROCESSOR MUST SPECIFY THE PROCESSOR

;       PROCESSOR = 1           PROCESSOR ONE (DMA CONTROLLER)
;       PROCESSOR = 2           PROCESSOR TWO (COBUS CONTROLLER)

        .IF     DUAL
;       SINGLE  = 0
;       .IF     PROCESSOR-1             ;PROCESSOR 2
;       P1      = 0
;       P2      = 1
;       FELLOW  = 1                     ;FELLOW PROCESSOR IS PROCESSOR 1
;       .ELSE   PROCESSOR-1             ;PROCESSOR 1
;       P1      = 1
;       P2      = 0
;       FELLOW  = 1
;       .ENDIF  PROCESSOR-1
        .ELSE   DUAL            ;SINGLE PROCESSOR SYSTEMS
        P1      = 1
        P2      = 1
        SINGLE  = 1
        .ENDIF  DUAL

;COBUS DEFINITIONS

;THE MAX NUMBER OF RETRIES DETERMINES ALSO THE MIN TIME BEFORE
;THE COBUS DRIVERS SIGNAL A NOCOR (NO CORRESPONDANT) ERROR

;THE WAITING TIME IN SECONDS IS APPROXIMATELY

;       WAITTIME = (CBMXTRY/10.)**2)

;FOR A 50.HZ RTC

;THUS CBMXTRY MAY BE COMPUTED AS

;       CBMXTRY = 10.*SQRT(WAITTIME)

        CBMXTRY = 30.           ;MAX NUMBER OF RETRIES

;GENERAL STATUS
```

```
        CBDORMANT       = 0

        CBREQUEST       = 1
        CBREQB  = 0

        CBDONE  = 2
        CBDONB  = 1

        CBEROR  = 200           ;COBUS ERROR
        CBERRB  = 7             ;ERROR BIT IN STATUS

        CBNOCOR = 1             ;NO CORRESPONDANT ERROR

;DATA REGISTERS

        CBDATA  = 14            ;COBUS DATA REGISTER (I & O)

;STATUS REGISTERS

        CBSTAT  = 15            ;STATUS REGISTER OF ACIA (INPUT)

        ;BITS IN STATUS REGISTER

        CBIDRF  = 1             ;INPUT DATA REGISTER FULL
        CBODRE  = 2             ;OUTPUT DATA REGISTER EMPTY

        ;ERROR FLAGS

        CBFREE  = 10            ;BUS FREE
        CBFE    = 20            ;FRAMING ERROR
        CBOVRN  = 40            ;RECEIVER OVERRUN ERROR
        CBPE    = 100           ;PARITY ERROR

        CBERR   = CBFE+CBOVRN+CBPE      ;ALL ERRORS

        CBINTS  = 16            ;COBUS INTERRUPT STATUS REGISTER

;BITS

        CBBFIB  = 1             ;BUFFER FREE INTERRUPT BIT NUMBER
        CBBAIB  = 0             ;BUFFER ACTIVE INTERRUPT BIT NUMBER

        CBCKSM  = 17            ;CHECKSUM REGISTER (INPUT)

;CONTROL REGISTERS

        CBCNTL  = 15            ;COBUS CONTROL REGISTER (OUTPUT)

        ;CONTROL BITS

        CBRST   = 3             ;RESET ACIA
        CBNORM  = 30            ;NORMAL PROGRAMMING MODE OF ACIA
        CBSEO   = 100           ;SET ENABLE OUT FLIP-FLOP


        CBSPID  = 16            ;SPEED AND INTERRUPT ENABLE CONTROL
                                ;REGISTER (OUTPUT)

        ;CONTROL BITS

        CBCRST  = 0             ;CHECKSUM AND INTERRUPT RESET
                                ;(SETS COBUS TO LOW SPEED)
        CBLOS   = 1             ;LOW SPEED
        CBMES   = 2             ;MEDIUM SPEED
        CBHIS   = 3             ;HIGH SPEED

        ;LOCAL SPEED

        CBLSPD  = CBMES

        CBSPMSK = 3             ;SPEED MASK BITS

        CBERCI  = 4             ;ENABLE RECEIVER INTERRUPTS
        CBEBFI  = 10            ;ENABLE BUS FREE INTERRUPTS
```

```
CBSYNC  = 17            ;SYNCHRONISATION CONTROL REGISTER
                        ;CONTROLS THE CLOCK DISABLE FLIP-
                        ;FLOP (OUTPUT)

CBECLK  = 0             ;ENABLE CLOCK
CBDCLK  = 1             ;DISABLE CLOCK
CBSYN   = CBDCLK        ;DISABLE CLOCK (SYNCHRONISE ON
                        ;FIRST BYTE)

;COBUS VARIABLES

CBUDUP  = 200           ;USELESS DUPLICATA
CBUDPB  = 7             ;USELESS DUPLICATA BIT

CBDUPB  = 7             ;DUPLICATA BIT NUMBER

CBROCC  = 10            ;RECEIVER OCCUPIED
CBROCB  = 3             ;RECEIVER OCCUPIED BIT

CBTTRB  = 0             ;TRIG RETRY BIT

CBBLOK  = 125

;DISPLACEMENTS IN COMMUNICATION TABLES

        ;TRANSMITTER TABLE

CBTSTATUS= 0            ;TRANSMITTER STATUS
CBTDST  = CBTSTATUS+1   ;DESTINATION ADDRESS
CBTSRC  = CBTDST+1      ;SOURCE ADDRESS
CBTBTYP = CBTSRC+1      ;BLOCK TYPE
CBTLEN  = CBTBTYPE+1    ;BYTE COUNT
CBTBUF  = CBTLEN+1      ;BUFFER POINTER
CBTDUP  = CBTBUF+2
CBTSPD  = CBTDUP+1
CBTERR  = CBTSPD+1
CBTWAIT = CBTERR+1
CBTRTRY = CBTWAIT+1
CBTFLG  = CBTRTRY+1     ;GENERAL PURPOSE FLAGS

;RECEIVER TABLE

CBRSTATUS= 0            ;RECEIVER STATUS
CBRDST  = CBRSTATUS+1   ;DESTINATION ADDRESS
CBRSRC  = CBRDST+1      ;SOURCE ADDRESS
CBRBTYP = CBRSRC+1      ;BLOCK TYPE
CBRLEN  = CBRBTYPE+1    ;BYTE COUNT
CBRBUF  = CBRLEN+1      ;BUFFER POINTER
CBRDUP  = CBRBUF+2
CBRSPD  = CBRDUP+1

        ;WATCHDOG

        CBDOG   = 50.           ;RTC INCREMENTS FOR WATCHDOG

        ;DEFINE STACK LENGTH FOR INTERRUPT ROUTINE

        CBISLEN = 40

        .PC     PCSYROM

        .IF     P1
;SUBROUTINE CBSEND

;SENDS A BLOCK WITH COBUS

;INPUT PARAMETERS:
;       D       DESTINATION ADDRESS
;       E       LOCAL   ADDRESS
;       BC      BLOCK LENGTH
;       HL      BLOCK ADDRESS

;OUTPUT PARAMETERS:
;       CARRY   SET FOR AN ERROR RETURN
;       A       ERROR CODE (ONLY IF ERROR RETURN)
```

```
CBSEND:
        PUSH    IX
        PUSH    AF
        LOAD    IX,#CBOTABLE

;PROTECT ANY TRANSMISSION

CBSND0:
        TEST    (IX)+CBTSTATUS:CBREQB   ;REQUEST PENDING ?
        JUMP,NE CBSND0
        .IF     SINGLE
        IOF
        .ENDIF
        LOAD    (IX)+CBTDST,D
        LOAD    (IX)+CBTSRC,E
        LOAD    (IX)+CBTLEN,C
        LOAD    (IX)+CBTBUF,L
        LOAD    (IX)+CBTBUF+1,H
        LOAD    (IX)+CBTBTYP,#0
        CLR     (IX)+CBTDUP:CBDUPB
        LOAD    (IX)+CBTRTRY,#0
        LOAD    (IX)+CBTWAIT,#0
        LOAD    A,#1
        LOAD    CBWDOG,A                ;KICK WATCHDOG
        LOAD    (IX)+CBTSTATUS,#CBREQUEST
        .IF     SINGLE
        CALL    CBRTC           ;SIMULATE RTC INTERRUPT
        ION
CBSND1:
        TEST    (IX)+CBTSTATUS:CBDONB   ;WAIT FOR DONE
        JUMP,EQ CBSND1
        TEST    (IX)+CBTSTATUS:CBERRB   ;ERROR ?
        LOAD    (IX)+CBTSTATUS,#CBDORMANT
        JUMP,NE CBSND2          ;YES
        POP     AF
        AND     A,A             ;CLEAR CARRY
CBSNEX:
        POP     IX
        RET

CBSND2:
        POP     AF
        LOAD    A,(IX)+CBTERR
        SETC
        JUMP    CBSNEX
        .ENDIF  SINGLE

        .IF     DUAL
        POP     AF
CBSNEX:
        POP     IX
        RET
        .ENDIF  DUAL


;SUBROUTINE CBRECV

;AWAITS A BLOCK FROM COBUS

;INPUT PARAMETERS:
;       D       AWAITED TRANSMITTER (0 ACCEPT ANY TRANSMITTER)
;       E       LOCAL ADDRESS (0 DO NOT INSERT IN TABLE)
;       HL      BUFFER ADDRESS (MIN 400 LONG)

;OUTPUT PARAMETERS:
;       D       SOURCE (EFFECTIVE TRANSMITTER)
;       E       DESTINATION (EFFECTIVE RECEIVER ADDRESS)
;       BC      BUFFER LENGTH
;       HL      BUFFER ADDRESS

CBRECV:
        PUSH    IX
        PUSH    AF
        LOAD    IX,#CBITABLE
        .IF     SINGLE
```

```
        IOF
        .ENDIF  SINGLE
        LOAD    (IX)+CBRBUF,L
        LOAD    (IX)+CBRBUF+1,H
        LOAD    (IX)+CBRSRC,D
        LOAD    (IX)+CBRDST,E
        LOAD    A,E
        OR      A,A
        CALL,NE CBENTER
        CLR     (IX)+CBRDUP:CBDUPB
        LOAD    (IX)+CBRSTATUS,#CBREQUEST
        .IF     SINGLE
        ION
        .IF     LINDA
        .ELSE
CBREC0:
        TEST    (IX)+CBRSTATUS:CBDONB   ;WAIT FOR BLOCK
        JUMP,EQ CBREC0
        LOAD    (IX)+CBRSTATUS,#CBDORMANT
        LOAD    D,(IX)+CBRSRC
        LOAD    E,(IX)+CBRDST
        LOAD    C,(IX)+CBRLEN
        LOAD    B,#0
        LOAD    A,C
        OR      A,A
        JUMP,NE CBREC1
        INC     B
CBREC1:
        .ENDIF  LINDA
        .ENDIF  SINGLE
        POP     AF
        JUMP    CBSNEX
        .ENDIF  P1

;SUBROUTINE CBINIT
;INITIALIZES THE COBUS SYSTEM
;IS CALLED ONCE AT INITIALIZATION

CBINIT:
        .IF     P1
        CALL    CBCLRT          ;CLEAR THE RESOURCE TABLE
        LOAD    IX,#CBOTABLE
        LOAD    (IX)+CBTSTATUS,#CBDORMANT
        LOAD    (IX)+CBTWAIT,#0
        LOAD    IX,#CBITABLE
        LOAD    (IX)+CBRSTATUS,#CBDORMANT
        .ENDIF  P1

        .IF     P2
        LOAD    A,#CBLOS+CBEBFI+CBERCI  ;LOW SPEED
                                        ;ENABLE ALL INTERRUPTS
        CALL    CBRSTR
        .ENDIF  P2
        RET

        .IF     P2
;REAL TIME CLOCK INTERRUPT ENTRY

CBRTC:
        LOAD    CBPSTK,SP       ;SAVE THE USER STACK
        LOAD    SP,#CBISTK      ;USE OUR OWN STACK
        EX      AF              ;SAVE WHAT YOU CAN
        SETC                    ;DO NOT ENABLE INTERRUPTS UPON EXIT
        PUSH    AF
        EX      BL
        PUSH    IX
        PUSH    IY
        LOAD    HL,#CBWDOG      ;LOOK OUT FOR WATCH DOG
        DEC     (HL)
        JUMP,NE CBCLK0          ;DON'T BARK
        LOAD    (HL),#CBDOG     ;WIND UP
CBCLK1:
        INP     A,CBSTAT        ;REINITIALIZE COBUS INTERFACE CORRECTLY
        AND     A,#CBFREE
        JUMP,NE CBSND           ;IF FREE CHECK IF ANYTHING IS PENDING
        JUMP    CBEXBO          ;ELSE WAIT FOR FREE BUS
```

```
CBCLK0:
        LOAD    IX,#CBOTABLE
        LOAD    A,(IX)+CBTWAIT  ;SOMEBODY WAITING ?
        OR      A,A
        JUMP,EQ CBEXIT          ;NO
        DEC     (IX)+CBTWAIT    ;WAITED ENOUGH ?
        JUMP,EQ CBCLK1          ;YES, CHECK BUS STATE
        JUMP    CBEXIT

;COBUS INTERRUPT ENTRY (INTERRUPT 30)

CBINT:
        LOAD    CBPSTK,SP       ;SAVE THE USER STACK
        LOAD    SP,#CBISTK      ;USE OUR OWN STACK
        EX      AF              ;SAVE WHAT YOU CAN
        AND     A,A
        PUSH    AF
        EX      BL
        PUSH    IX
        PUSH    IY
        INP     A,CBINTS        ;WHICH INTERRUPT ?
        TEST    A:CBBAIB        ;BUS ACTIVE ?
        JUMP,NE CBRCVE
        TEST    A:CBBFIB        ;BUS FREE ?
        JUMP,EQ CBEXIT          ;SPURIOUS INTERRUPT

CBSND:

;IS THERE A REQUEST TO SEND ?

        LOAD    IX,#CBOTABLE    ;POINT TO COMMUNICATION TABLE

        TEST    (IX)+CBTSTATUS:CBREQB   ;SOMETHING PENDING ?
        JUMP,EQ CBEXBF          ;READY RECEIVER SECTION

;ARE WE WAITING ?

        LOAD    A,(IX)+CBTWAIT
        OR      A,A
        JUMP,NE CBEXBF          ;YES, BE PATIENT

;THERE IS A REQUEST. TRY TO SNATCH THE BUS

        LOAD    A,#CBLOS
        CALL    CBRSTT

        SET     (IX)+CBTFLG:CBTTRB      ;TRIG FOR RETRIES

;SEND DESTINATION

        LOAD    A,(IX)+CBTDST
        OUT     CBDATA,A
        LOAD    B,A             ;CHECKSUM

;SOURCE ADDRESS (MY NAME)

        LOAD    A,(IX)+CBTSRC
        CALL    CBOUTPUT

;COLLISION DETECTION IS DONE BY READING BACK THE BYTES SENT ON
;THE BUS

        CALL    CBINPUT
        COMP    A,(IX)+CBTDST
        JUMP,NE CBXRBO          ;COLLISION

        CALL    CBINPUT
        COMP    A,(IX)+CBTSRC
        JUMP,NE CBXRBO          ;ALSO COLLISION

;SEND BLOCK TYPE

        LOAD    A,#CBLSPD       ;LOCAL SPEED
        OR      A,(IX)+CBTDUP   ;ADD IN DUPLICATA BIT
        OR      A,(IX)+CBTBTYP  ;AND BLOCK TYPE
```

```
        CALL    CBOUTPUT

;THE RECEIVER IS EMPTIED TO KEEP UP WITH THE SYNCHRONISATION
;THIS IS THE EASIEST WAY TO MONITOR THE END OF THE HEADER

        CALL    CBINPUT

;THE BUS NOW BELONGS TO THE TRANSMITTER/RECEIVER PAIR. THERE
;IS NO COLLISION POSSIBLE, THEREFORE NO CHECKING IS DONE
;THE HEADER CHECKSUM IS NOW OUTPUT

        LOAD    A,B             ;GET CHECKSUM

;MAKE TOTAL CHECKSUM NULL

        NEG     A
        CALL    CBOUTPUT

;WAIT FOR END OF TRANSMISSION

        CALL    CBINPUT

;AND READY FOR HEADER ACK
;BUS ARBITRATION HAVING BEEN RESOLVED SUCCESSFULLY, SPEED UP
;A BIT

        LOAD    A,#CBCRST       ;RESET CHECKSUM
        OUT     CBSPID,A
        LOAD    A,#CBMES        ;MEDUIM SPEED
                                ;EVERYBODY SHOULD BE ABLE TO RUN
                                ;AT THIS SPEED
        OUT     CBSPID,A
        LOAD    A,#CBSYN        ;DISABLE CLOCK TO SYNCHRONIZE
        OUT     CBSYNC,A        ;ON FIRST BYTE

;THE DOUBLE BUFFERING OF THE RECEIVER GIVES US SOME TIME
;FOR OTHER TASKS

;INITIATE ALL THE REGISTERS

        LOAD    L,(IX)+CBTBUF   ;DATA POINTER IN BUFFER
        LOAD    H,(IX)+CBTBUF+1
        LOAD    C,#CBDATA       ;DATA PORT

;DE IS LOADED WITH IMMEDIATE VALUES TO SPEED UP THE DATA
;TRANSFER LOOP AND TO PRESERVE A SMALL MARGIN

        LOAD    DE,#(CBODRE+CBFREE)*400+CBFREE

;AND WAIT

        CALL    CBINPUT
        LOAD    B,A

;GOT SOMETHING, WAIT FOR A VALIDATION

;TRY TO AGREE FOR A COMMON SPEED

        LOAD    A,#CBSPMSK      ;EXTRACT SPEED BITS
        AND     A,B
        COMP    A,#CBLSPD       ;WHO'S SLOWER ?
        JUMP,LO CB000
        LOAD    A,#CBLSPD       ;I AM, PLEASE DON'T HURRY ME
CB000:
        LOAD    (IX)+CBTSPD,A   ;REMEMBER FOR ACKNOWLEDGE

        CALL    CBINPUT

        INP     A,CBCKSM
        OR      A,A
        JUMP,NE CBXRBO

;FIRST CHECK IF WE WORK TOO MUCH (BY TRYING TO SEND A USELESS
;DUPLICATA)

        TEST    B:CBUDPB                ;BLOCK TYPE IS IN B
```

```
        JUMP,NE CBTUSDP                 ;SIGNAL A SUCCESSFULL TRANSMISSION

        TEST    B:CBROCB        ;IS CORRESPONDANT LAZY ?
        JUMP,NE CBXRBO          ;YES, WAIT FOR HIM TO WAKE UP

;FROM NOW ON ALL FOLLOWING BLOCKS WILL BE SIGNALED AS DUPLICATES

        SET     (IX)+CBTDUP:CBDUPB

        LOAD    A,(IX)+CBTSPD
        OUT     CBSPID,A
        LOAD    A,#CBRST+CBSEO  ;RESET ACIA AND SET ENABLE
        OUT     CBCNTL,A        ;OUT FLIP-FLOP
        LOAD    A,#CBNORM       ;PROGRAM ACIA
        OUT     CBCNTL,A

;DATA SECTION OF BLOCK
;FIRST THE BYTE COUNT

        LOAD    A,(IX)+CBTLEN
        OUT     CBDATA,A
        LOAD    B,A             ;BYTE COUNT IN B

;AND HERE COMES THE DATA

CB001:  INP     A,CBSTAT
        AND     A,D             ;AND    A,#CBODRE+CBFREE
        JUMP.,EQ CB001
        AND     A,E             ;AND    A,#CBFREE       ERROR ?
        JUMP,NE CBEXBF  ;YES

        OUTI

        JUMP",NE CB001          ;ZERO FLAG IS SET UPON LAST BYTE
                                ;FORCE AN ABSOLUTE JUMP (IS FASTER
                                ;THAN A RELATIVE JUMP WHEN SUCCESS-
                                ;FULL)

;NOW THE CHECKSUM

        INP     A,CBCKSM
        NEG     A
        CALL    CBOUTPUT

;SINCE WE DIT NOT HAVE TIME TO CARE FOR IT, THE RECEIVER SECTION
;IS DESYNCHRONIZED AND A BIT UPSET. WE MUST CONSIDER ANOTHER METHOD
;TO DETECT THE END OF THE TRANSMISSION.
;FILLING UP THE DOUBLE BUFFER WITH A DUMMY WORD WILL DO. THE SETTING
;OF ODRE BIT INDICATES THE END OF THE CHECKSUM BYTE.
;THE SPURIOUS START BIT (OF THE DUMMY BYTE WHICH IS FORCED
;ON THE BUS IS UNIMPORTANT SINCE BOTH TRANSMITTER AND
;RECEIVER WILL ANYHOW INITIALIZE.

        CALL    CBOUTPUT        ;DUMMY
CB002:
        INP     A,CBSTAT        ;WAIT UNTIL DUMMY BYTE FELL THROUGH
        AND     A,#CBODRE
        JUMP,EQ CB002


;AN ACKNOWLEDGE IS STILL DUE. BE READY FOR IT

        LOAD    A,(IX)+CBTSPD
        OUT     CBSPID,A
        LOAD    A,#CBRST        ;RESET ACIA
        OUT     CBCNTL,A
        LOAD    A,#CBNORM       ;SET UP ACIA
        OUT     CBCNTL,A
        LOAD    A,#CBSYN        ;DISABLE CLOCK TO SYNCHRONIZE
        OUT     CBSYNC,A        ;ON FIRST BYTE
        CALL    CBINPUT
        COMP    A,#CBBLOK       ;GOOD ACKNOWLEDGE ?
        JUMP,NE CBEXBO          ;NO ??!!  START ALL OVER AGAIN

;ALL DONE
```

```
CBTSUCCS:
        LOAD    (IX)+CBTSTATUS,#CBDONE
        JUMP    CBEXBO

CBTUSDP:
;       TRAP
        JUMP    CBTSUCCS

CBRCVE:

;GET THE DESTINATION

        LOAD    IX,#CBITABLE

        CLR     (IX)+CBTFLG:CBTTRB      ;NO WAITING ON INPUT

        CALL    CBINPUT

;TO REDUCE PROCESSOR OVERHEAD ALL THE DECISIONS CONCERNING
;THE HEADER ACKNOWLEDGE ARE MADE WITH AN UNVALIDATED DATA.
;HOWEVER THE DEFINITIV DECISION WILL BE MADE ONLY UPON THE
;VALIDATION GIVEN BY THE CHECKSUM.

;REGISTERS USED
;       B       DESTINATION ADDRESS
;       C       SOURCE ADDRESS
;       D       BLOCK TYPE

;       E       HEADER ACKNOWLEDGE (CONSTRUCTED)

;CHECK IF THIS ADDRESS IS IN OUR RESOURCE TABLE

        LOAD    HL,#CBRSCT      ;RESOURCE TABLE
        CALL    CBMKBA          ;MAKE A BIT ADDRESS

;WARNING!! WE USE THE FACT THAT CBMKBA COPIES REGISTER
;A (IN THIS CASE THE DESTINATION ADDRESS) INTO REGISTER B

        AND     A,(HL)
        JUMP,EQ CBEXBO          ;WE DO NOT FEEL CONCERNED

;READ SOURCE ADDRESS

        CALL    CBINPUT
        LOAD    C,A             ;KEEP UNTIL VALIDATED BY CHECKSUM

;CHECK IF THE RECEIVER IS LINKED TO SOMEBODY ELSE

        LOAD    E,#CBLSPD               ;INITIALIZE HEADER ACK
        LOAD    A,(IX)+CBRDST
        OR      A,A
        JUMP,EQ CBI1.1          ;NO LINKING YET, ACCEPT ANYBODY
        COMP    A,B
        JUMP,NE CBLNKED
CBI1.1:
        LOAD    A,(IX)+CBRSRC
        OR      A,A
        JUMP,EQ CBI1.2
        COMP    A,C
        JUMP,EQ CBI1.2          ;OK, FOR THE NAMES
CBLNKED:
        SET     E:CBROCB        ;SORRY, WE HAVE NOT YET BEEN INTRODUCED

;BLOCK TYPE

CBI1.2:
        CALL    CBINPUT
        LOAD    D,A             ;KEEP ALSO

        TEST    (IX)+CBRSTATUS:CBREQB ;ANSWER OCCUPIED IF NO REQUEST
        JUMP,NE CBRREQ
        SET     E:CBROCB
CBRREQ:

;DO THE DUPLICATA STUFF
```

```
        TEST    E:CBROCB
        JUMP,NE CBRDP0          ;DO NOT CHECK OWN VARIABLE IF ANSWER
                                ;IS RECEIVER OCCUPIED
        TEST    (IX)+CBRDUP:CBDUPB
        JUMP,NE CBRDP1          ;DUPLI IS EFFECTIVELY AWAITED DO
                                ;NOT BOTHER ANY MORE
CBRDP0:
        TEST    D:CBDUPB
        JUMP,EQ CBRDP1          ;ORIGINAL BLOCK
        SET     E:CBUDPB        ;USELESS DUPLICATA
CBRDP1:

;CHECKSUM

        CALL    CBINPUT

;VERIFY CHECKSUM

        INP     A,CBCKSM
        OR      A,A
        JUMP,NE CBXRBO          ;CHECKSUM ERROR

;ARE WE REFUSING THE BLOCK

        TEST    E:CBROCB
        JUMP,NE CBASWOC         ;DO NOT LOCK ON TRANSMISSION
                                ;BUT BE NICE AND ANSWER
;EVERYTHING IS OK LOCK ON TRANSMISSION

        LOAD    (IX)+CBRDST,B
        LOAD    (IX)+CBRSRC,C
        SET     (IX)+CBRDUP:CBDUPB      ;LOCK

;INITIALIZE THE TRANSMITTER SECTION

CBASWOC:
        LOAD    A,#CBMES
        OUT     CBSPID,A
        LOAD    A,#CBRST+CBSEO  ;RESET ACIA AND SET ENABLE
        OUT     CBCNTL,A        ;OUT FLIP-FLOP
        LOAD    A,#CBNORM       ;PROGRAM ACIA
        OUT     CBCNTL,A

;GET THE COMPUTED HEADER ACKNOWLEDGE

        LOAD    A,E
        OUT     CBDATA,A
        LOAD    B,A

;JUST A BIT OF TIME TO COMPUTE STUFF FOR LATER

;COMPUTE COMMON SPEED

        LOAD    A,#CBSPMSK
        AND     A,D             ;SPEED BITS
        COMP    A,#CBLSPD       ;WHO IS SLOWER
        JUMP,LO CBI00
        LOAD    A,#CBLSPD       ;ME AGAIN
CBI00:
        LOAD    (IX)+CBRSPD,A   ;REMEMBER

;SINCE WE MAY BE IN A HURRY LATER, LET US SET UP AS MANY
;REGISTERS AS POSSIBLE NOW

        LOAD    L,(IX)+CBRBUF
        LOAD    H,(IX)+CBRBUF+1
        LOAD    C,#CBDATA
        LOAD    DE,#(CBIDRF+CBERR+CBFREE)*400+(CBERR+CBFREE)

;ACKNOWLEDGE CHECKSUM

        CALL    CBINPUT
        LOAD    A,B             ;GET COMPUTED CHECKSUM
        NEG     A
        CALL    CBOUTPUT
```

```
        TEST    B:CBROCB        ;EXIT IF WE ANSWERED OCCUPIED
        JUMP,NE CBXRBO

;MONITOR END OF TRANSMISSION

        CALL    CBINPUT

        LOAD    A,#CBCRST       ;RESET CHECKSUM
        OUT     CBSPID,A
        LOAD    A,(IX)+CBRSPD
        OUT     CBSPID,A
;       LOAD    A,#CBRST        ;RESET ACIA
;       OUT     CBCNTL,A
;       LOAD    A,#CBNORM       ;SET UP ACIA
;       OUT     CBCNTL,A
        LOAD    A,#CBSYN        ;DISABLE CLOCK TO SYNCHRONIZE
        OUT     CBSYNC,A        ;ON FIRST BYTE

;BYTE COUNT

        CALL    CBINPUT
        LOAD    (IX)+CBRLEN,A
        LOAD    B,A

;READ THE DATA

CBI05:
        INP     A,CBSTAT
        AND     A,D             ;D CONTAINS CBIDRF+CBERR+CBFREE
        JUMP.,EQ CBI05

        AND     A,E             ;E: CBERR+CBFREE
        JUMP,NE CBRERR

        INI                     ;FAST INPUT AND TRANSFER AT (HL)

        JUMP",NE CBI05          ;BYTE COUNT IS UPDATED IN B

;GET THE CHECKSUM

        CALL    CBINPUT

        INP     A,CBCKSM
        OR      A,A
        JUMP,NE CBXRBO


;MESSAGE IS ACCEPTED SEND BACK AN ACKNOWLEDGE

        LOAD    A,(IX)+CBRSPD   ;USED SPEED
        CALL    CBRSTT

        LOAD    A,#CBBLOK
        CALL    CBOUTPUT

;FILL THE DOUBLE BUFFER TO DETECT THE END OF THE
;TRANSMISSION OF BLOCK ACKNOWLEDGE

        CALL    CBOUTPUT
        CALL    CBOUTPUT        ;FOR THE TEST INTRUCTIONS

        LOAD    A,#CBRST        ;SHUT DOWN THE DUMMY TRANSMISSIONS
        OUT     CBCNTL,A
        LOAD    A,#CBNORM
        OUT     CBCNTL,A

        LOAD    (IX)+CBTSTATUS,#CBDONE  ;BLOCK IS HERE

        JUMP    CBEXBO

;EXITS WHICH TEST THE RETRY FLAG

CBXRBO:
        TEST    (IX)+CBTFLG:CBTTRB
        CALL,NE CBTRFNC         ;HAVE A BETTER LOOK
```

```
        JUMP    CBEXBO

CBXRBF:
        TEST    (IX)+CBTFLG:CBTTRB
        CALL,NE CBTRFNC
        JUMP    CBEXBF

CBTRFNC:
        LOAD    A,(IX)+CBTRTRY
        COMP    A,#CBMXTRY
        JUMP,HS CBRTF0          ;VERY BAD, RECEIVER IS NOT VERY
                                ;COOPERATIVE
        INC     A
        LOAD    (IX)+CBTRTRY,A
        LOAD    (IX)+CBTWAIT,A  ;WAIT PROPORTIONALLY TO YOUR PATIENCE
        RET

CBRTF0:
        LOAD    (IX)+CBTERR,#CBNOCOR    ;NO CORRESPONDANT ERROR
        LOAD    (IX)+CBTSTATUS,#CBDONE+CBEROR
        JUMP    CBTSEX

;TEST EXIT
;IS USED IF STATE OF BUS IS NOT KNOWN
;USES APPROPRIATE EXIT

CBTSEX:
        INP     A,CBSTAT
        AND     A,#CBFREE
        JUMP,NE CBEXBF
        JUMP    CBEXBO

CBEXBF:
        LOAD    A,#CBERCI+CBLOS
        CALL    CBRSTR
        JUMP    CBEXIT

CBEXBO:
        LOAD    B,#CBEBFI+CBLOS
CBCMEX:
        LOAD    A,#CBCRST       ;RESET INTERRUPT FLAGS
        OUT     CBSPID,A
        LOAD    A,B
        OUT     CBSPID,A
CBEXIT:
        LOAD    SP,#CBISTK-6    ;CLEAN UP THE STACK
        POP     IY
        POP     IX
        EX      BL
        POP     AF              ;WAS IT AN INTERRUPT ROUTINE
        JUMP,CS CBNIRT
        EX      AF
        LOAD    SP,CBPSTK       ;RESTORE USER STACK
        ION
        RET

CBNIRT:
        EX      AF
        LOAD    SP,CBPSTK       ;RESTORE USER STACK
        RET


;OUTPUT A BYTE ROUTINE

;THE BYTE IS INPUT IN REGISTER A
;A COPY IS MADE IN REGISTER B BECAUSE A IS DESTROYED
;A CHECKSUM IS COMPUTED IN REGISTER C BECAUSE THE OUTPUT
;ROUTINE IS GENERALLY FOLLOWED BY A CALL TO THE INPUT
;ROUTINE AND THUS THE HARDWARE CHECKSUM GENERATOR
;CANNOT BE USED.

CBOUTPUT:
        PUSH    AF              ;REMEMBER DATA

;COMPUTE CHECKSUM
```

```
        ADD     A,B
        LOAD    B,A

CBOUT0:
        INP     A,CBSTAT        ;GET STATUS (IN ACIA)

;WAIT UNTIL DATA BUFFER IS FREE
;EXIT ALSO FROM THE LOOP IF CBFREE IS SET
;(THIS MAY HAPPEN IF THE ENABLE OUT FLIP-FLOP
;HAS BEEN RESET DUE TO NOISE ON THE BUS).

        AND     A,#CBODRE+CBFREE
        JUMP,EQ CBOUT0

;DO NOT CONTINUE IF THE EXIT CONDITION WAS
;A FREE BUS.

        AND     A,#CBFREE
        JUMP,NE CBOUT1          ;FREE BUS

        POP     AF
        OUT     CBDATA,A

        RET

CBOUT1: POP     DE              ;EMPTY STACK
        JUMP    CBXRBF

;INPUT A BYTE ROUTINE

;THE READ BYTE IS RETURNED IN REGISTER A

CBINPUT:
        INP     A,CBSTAT

;WAIT UNTIL DATA IS HERE
;EXIT ALSO FROM THE LOOP IF AN ERROR
;(PARITY, FRAMING, RECEIVER OVERRUN) OR
;A FREE BUS IS DETECTED

        AND     A,#CBIDRF+CBERR+CBFREE
        JUMP,EQ CBINPUT

;DO NOT CONTINUE IF AN ERROR HAS BEEN DETECTED

        AND     A,#CBERR+CBFREE
        JUMP,NE CBRERR          ;ERROR WHILE READING A BYTE

        INP     A,CBDATA        ;GET THE DATA

        RET

CBRERR:
        AND     A,#CBERR
        JUMP,NE CBXRBO
        JUMP    CBXRBF

;RESET COBUS TO TRANSMITTER MODE

;INPUT IS SPEED IN REGISTER A

CBRSTT:
        LOAD    B,A
        LOAD    A,#CBECLK       ;ENABLE CLOCK
        OUT     CBSYNC,A
        LOAD    A,#CBCRST       ;RESET CHECKSUM
        OUT     CBSPID,A
        LOAD    A,B
        OUT     CBSPID,A
        LOAD    A,#CBRST+CBSEO  ;RESET ACIA AND SET ENABLE
        OUT     CBCNTL,A        ;OUT FLIP-FLOP
        LOAD    A,#CBNORM       ;PROGRAM ACIA
        OUT     CBCNTL,A
        LOAD    B,#0            ;RESET CHECKSUM
        RET
```

```
;RESET COBUS TO THE RECEIVER MODE

;INPUT IS SPEED IN REGISTER A

CBRSTR:
        LOAD    B,A             ;REMEMBER SPEED
        LOAD    A,#CBCRST       ;RESET CHECKSUM
        OUT     CBSPID,A
        LOAD    A,B             ;SET UP SPEED
        OUT     CBSPID,A
        LOAD    A,#CBRST        ;RESET ACIA
        OUT     CBCNTL,A
        LOAD    A,#CBNORM       ;SET UP ACIA
        OUT     CBCNTL,A
        LOAD    A,#CBSYN        ;DISABLE CLOCK TO SYNCHRONIZE
        OUT     CBSYNC,A        ;ON FIRST BYTE
        RET
        .ENDIF  P2

CBENTER:
        PUSH    AF
        PUSH    BC
        PUSH    DE
        PUSH    HL
        LOAD    HL,#CBRSCT
        CALL    CBMKBA
        OR      A,(HL)
CBENEX:
        LOAD    (HL),A
        POP     HL
        POP     DE
        POP     BC
        POP     AF
        RET

CBEXTR:
        PUSH    AF
        PUSH    BC
        PUSH    DE
        PUSH    HL
        LOAD    HL,#CBRSCT
        CALL    CBMKBA
        CPL     A
        AND     A,(HL)
        JUMP    CBENEX

CBCLRT:
        LOAD    HL,#CBRSCT
        LOAD    B,#40           ;DIMENSION OF RESOURCE TABLE
                                ;(IN BYTES)
CBCLRL:
        LOAD    (HL),#0
        INC     HL
        DECJ,NE B,CBCLRL
        RET

;ROUTINE TO COMPUTE A BIT ADDRESS
;INPUT IN HL POINTER TO TABLE
;REGISTER A IS THE BIT NUMBER
;OUTPUT IS THE MASK IN A AND THE BYTE POINTER IN HL

;DE IS DESTROYED
;REGISTER A IS COPIED TO B

CBMKBA:
        PUSH    HL
        LOAD    B,A
        AND     A,#7
        LOAD    E,A
        LOAD    D,#0
        LOAD    HL,#CBBMSK
        ADD     HL,DE
        LOAD    A,(HL)
        LOAD    E,B
        SRC     E
```

```
        SRC     E
        SRC     E
        POP     HL
        ADD     HL,DE
        RET

CBBMSK: .B      1 2 4 10 20 40 100 200

        .PC     PCSYRAM

        .IF     DUAL
        .PC     COMMON
        .ENDIF  DUAL
CBOTABLE:       .BLKB   20
CBITABLE:       .BLKB   20
CBWDOG: .B      0
CBRSCT:         .BLKB   40

        .BLKB   CBISLEN
CBISTK:
CBPSTK: .W      0

        .END    START
```

ANNEXE C

```
.TITLE  COBUSF.MAC
.SBTTL  COBUS FOREGROUND O.S.

;PROGRAM BY ROLAND DUMONT        EPFL-LCD 1-FEV-79

        .MCALL  .GTJB,.SYNCH,.SPND,.RSUM
        .MCALL  .INTEN,.RCVDC,.PROTECT,.TWAIT,.SDATW,.CSISPC
        .MCALL  .ENTER,.DELETE,.RENAME,.LOOKUP,.WRITW,.CLOSE
        .MCALL  .READW
        .MCALL  .EXIT,.RCVDW,.PRINT,.SDATC

        .LIST   MEB     ;LIST MACRO EXTENSIONS


; DEFINITIONS.
;*****************

SYSPTR  =       54              ;RMON PTR
EMT16   =       316             ;RMON OFFSET TO EMT LIST
EMTRTN  =       400             ;RMON OFFSET TO EMT RETURN
CTRLC   =       3
STAR    =       52
POINT   =       56
PRO     =       1
BATCHAN =       0
LEN     =       2
SIX     =       6
FIVE    =       5
THREE   =       3
TWO     =       2
ZERO    =       0
RD50ASC =       22      ;CODES IN RAD50 MODE
RD50ZERO=       36
RD50EIGTH=      46
ZER0ASC =       60
FIRST   =       1

;VARIABLES.
;***********

WORD    =       2       ;WORD = 2 BYTES
BYTE    =       1

;COLSI DEFINITIONS.

COBCSR  =       164170  ;LINDA STATUS REG.
COBVEC  =       170     ;LINDA INTERRUPT VECTOR
PRIOR   =       7       ;INTERRUPT ROUTINE PRIORITY

;LINDA STATUS REGISTER DEFINITION.

SETIZ80 =       1       ;INTERRUPT LINDA
INTENB  =       100     ;INTERRUPT ENABLE LINDA
IDONE   =       200     ;INTERRUPT LSI

;ACTION BLOCK DEFINITIONS.

ABLKPTR =       0               ;BUFFER PTR
DMESLEN =       ABLKPTR+WORD    ;MESSAGE LENGTH
DAC     =       DMESLEN+WORD    ;ACTION CODE
DADR    =       DAC+BYTE        ;LOCAL ADDR
DCOR    =       DADR+BYTE       ;CORRESPONDANT ADDR
DAS     =       DCOR+BYTE       ;ACTION STATUS
DSEEN   =       DAS+BYTE        ;SEEN FLAG
NBRBLK  =       20              ;NBR OF ACTION BLOCKS
ABLEN   =       5*WORD          ;AB HEADER LENGTH
ABUFLEN =       BUFLEN+ABLEN    ;AB LENGTH
BUFLEN  =       400             ;BUFFER LENGTH
ANSLEN  =       2               ;ANSWER LENGTH
OPLEN   =       3               ;OPEN OK LENGTH

;ACTION STATUS CODES.

DORMANT =       0               ;AB FREE
```

```
REQ     =       1               ;UNDER LINDA CONTROL
DONE    =       2               ;UNDER LSI CONTROL

;ACTION CODE CODES.

AI      =       1               ;INPUT FROM COBUS
AO      =       0               ;OUTPUT ON COBUS

;LSI ADDRESSES ON COBUS.

FLOCALADR       =       376     ;FOREGROUND ADDR
BLOCALADR       =       377     ;BACKGROUND ADDR

;SWITCHES.

SET     =       1
RESET   =       0
FREE    =       1
FULL    =       0
OCCUPIED=       0
SEEN    =       1
NOSEEN  =       0

;SEPARATORS.

ZERO    =       0
TAB     =       11
LF      =       12
FF      =       14
CR      =       15
SPACE   =       40
COMMA   =       54

;FILE BLOCK DEFINITIONS.

CHPTR   =       0               ;CHANNEL PTR
DCHSTAT =       CHPTR+BYTE      ;CHANNEL STATUS
DFILNAM =       DCHSTAT+BYTE    ;ASSOCIATED FILE NAME
DSTATION=       DFILNAM+<RD50LEN*WORD>  ;ASS. COBUS STAT.
DNEWFIL =       DSTATION+WORD   ;CREATE SWITCH
DOPMOD  =       DNEWFIL+WORD    ;OPEN MODE
DBLKNBR =       DOPMOD+WORD     ;DISK BLOCK NBR
DFILPTR =       DBLKNBR+WORD    ;CHAR. PTR
DFILBUF=        DFILPTR+WORD    ;FILE BUFFER

FBHEADLEN=      <6+RD50LEN>*WORD        ;FB HEADER LEN.
FBUFLEN =       BUFLEN*WORD             ;FB BUFFER LENGTH
FBLKLEN =       FBHEADLEN+FBUFLEN       ;FB LENGTH

BLKEMPTY=       ZERO            ;BUFFER EMPTY
RD50LEN =       4               ;RAD50 FILE NAME LENGTH
NF      =       1               ;NEW FILE INDICATOR

OPREAD  =       0               ;OPEN FOR READING CODE
OPWRIT  =       1               ;OPEN FOR WRITING CODE
RDOPCOD =       5               ;READ CODE (COBUS)
WROPCOD =       6               ;WRITE CODE (COBUS)

;CHANNELS.

FSTCHAN =       0               ;FIRST CHANNEL
LSTCHAN =       17              ;LAST CHANNEL
NBRCHAN =       NBRBLK          ;NBR OF CHANNELS

;RT-11 LOCATIONS.

ERBYT   =       52              ;LOC. FOR ERROR CODES
HARDERR =       1               ;HARDWARE ERROR CODE (LSI)

;ANSWER CODES.

OK      =       0
ERROR   =       1

;ERROR CODES.
```

```
ERFNO   =       0       ;ILLEGAL CHANNEL NUMBER.
ERFNM   =       1       ;ILLEGAL FILE NAME.
ERICM   =       2       ;ILLEGAL SYSTEM COMMAND.
EREOF   =       6       ;END OF FILE.
ERCRE   =       11      ;FILE ALREADY EXISTS.
ERDLE   =       12      ;FILE NOT FOUND.
ERUFT   =       21      ;NO CHANNEL.
ERLLI   =       22      ;LINE LIMIT EXCEEDED
ERSPC   =       27      ;DISK FULL.
ERRFIL  =       30      ;HARD ERROR.
NOFILBLK=       106     ;NO FREE FILE BLOCK.
FATERR  =       105     ;FATAL ERROR.

.MACRO  ADDR    ADDRESS,REG
        MOV     PC,REG
        ADD     #ADDRESS-.,REG
        .ENDM
.MACRO  LINK1   N,OLD
        MOV     N*2(R1),R2
        ADD     R1,R2
        ADDR    OLD,R3
        SUB     R3,R2
        MOV     R2,OLD
        .ENDM
.MACRO  LINK2   N,LABLE
        ADDR    LABLE,R3
        SUB     R1,R3
        MOV     R3,N*2(R1)
        .ENDM

        .PAGE
        .SBTTL  INTERRUPT ROUTINE.

; INTERRUPT ROUTINE.
;*******************
;
; ON ENTRE DANS CETTE ROUTINE SUR UNE INTERRUPTION VENANT
; DE LINDA. ET ON PROVOQUE LA CONTINUATION DE LA BOUCLE
; PRINCIPALE DU PROGRAMME PAR UN .RESUME .

INT:
        BIC     #IDONE,COBCSR   ;CLEAR LINDA INTERRUPT.
        BIC     #INTENB,COBCSR  ;SET LSI IOF.
        .INTEN  PRIOR           ;SWITCH TO SYSTEM STATE
        .SYNCH  #SYNBLK         ;ASK FOR DOING PROG. REQ.
        BR      SYNFAIL
        .RSUM
        BIS     #INTENB,COBCSR  ;SET LSI ION.
        RTS     PC

SYNFAIL:
        HALT

        .PAGE
        .SBTTL  COMPLETION ROUTINE.


; COMPLETION ROUTINE.
;********************
;
; ON ENTRE DANS CETTE ROUTINE SUR RECEPTION D'UNE
; REPONSE DU BACKGROUND.
; ON LA MET DANS LE ACTION BLOCK COURANT DU B-G CURB.
; ET ON SIGNALE QU'IL FAUT ENVOYER CE BLOCK PAR COBUS.
; PUIS ON FAIT UN .RESUME POUR LA BOUCLE PRINCIPALE
; DU PROG.

COMPL:
        MOV     R0,-(SP)        ;SAVE REGS ON STACK
        MOV     R1,-(SP)
        MOV     R2,-(SP)
        MOV     #RESCOD,R0      ;ANALYSE ANSWER
        CMPB    #ERROR,(R0)
        BEQ     COMPL1
        MOV     #3,R1           ;AND INIT TRANSFER
        BR      COMPL2
```

```
COMPL1: MOV     #2,R1
COMPL2: MOV     CURB,R2          ;TRANSFER INTO AB
        MOVB    R1,DMESLEN(R2)
        ADD     #ABLEN,R2
1$:     MOVB    (R0)+,(R2)+
        SOB     R1,1$
        MOV     CURB,R2          ;SET AB FOR OUTPUT
        MOVB    #AO,DAC(R2)
        MOVB    #REQ,DAS(R2)
        BIS     #SETIZ80,COBCSR
        .RSUM                    ;RESUME PROG EXEC
        MOV     (SP)+,R2         ;AND RESTORE REGS
        MOV     (SP)+,R1
        MOV     (SP)+,R0
        RETURN

        .PAGE
        .SBTTL  MAIN PROGRAM.

; MAIN PROGRAM.
;**************
;
; CECI EST LA BOUCLE PRINCIPALE DU SYSTEME COBUS.
;
; INITIAL:
;       ON REMPLACE LES ROUTINES POUR LES E/S SUR
;       TERMINAL PAR CELLES DU BATCH COBUS, ET ON
;       GARNIT L'ADRESSE DE LA ROUTINE D'INTERRUPTION.
; WLOOP:
;       ON SYNCHRONISE AVEC LINDA ET ON LUI PASSE DES
;       INFORMATIONS SUR DES LOCATIONS DANS LE LSI.
; START:
;       ON LAISSE TOUJOURS UN ACTION BLOCK EN RECEPTION
;       COBUS.
; M3:
;       ON RECHERCHE SI IL Y A QCH. A FAIRE POUR LE FOREGROUND
;       (MANIPULATIONS DE FICHIER), ET ON BRANCHE A LA ROUTINE
;       CORRESPONDANTE AU CODE DE L'APPEL SYSTEME.
;       APRES ON SIGNALE UNE REPONSE A LINDA.
; SUSPENSION.
; M77:
;       SI LE BACKGROUND A FINI SON TRAVAIL, ON CHERCHE SI IL
;       Y A ENCORE QCH. A FAIRE POUR B-G ET ON LUI ENVOIE
;       LA COMMANDE.
; SUSPENSION.
; M4:
;       ON RECHERCHE LES ACTION BLOCKS DEJA ENVOYES PAR COBUS
;       ET ON LES LIBERE POUR LA SUITE.
; M7:
;       ON BOUCLE A START.
;
;
; SOUSROUTINES UTILISEES:
;
;       - NEXTFREEAB
;       - NEXTFG
;       - NEXTBG
;       - NEXTO
;       - CREATE
;       - DELETE
;       - RENAME
;       - NOIMOK
;       - OPEN
;       - APPEND
;       - CLOSE
;       - CRESET
;       - RDBYTE
;       - RDL
;       - WRBYTE
;       - WRL
;       - NOIMIC
;

PROTERR:
        HALT
```

```
INITIAL:
        MOV     @#SYSPTR,R0     ;POINT TO RMON
        MOV     EMT16(R0),R1    ;GET PTR TO EMT LIST
        ADD     R0,R1           ;POINT TO EMT LIST
        LINK1   0,O$TTIN        ;SAVE DISTANCE
        LINK1   1,O$TTOUT       ;FROM ORIGINAL
        LINK1   10,O$EXIT       ;ROUTINE TO USER
        LINK1   11,O$PRINT      ;ROUTINES HERE

        LINK2   0,FB$TTIN       ;STORE IN RMON
        LINK2   1,FB$TTOUT      ;EMT LIST, THE
        LINK2   10,FB$EXIT      ;DISTANCE TO
        LINK2   11,FB$PRINT     ;USER ROUTINES.

        MOVB    #PRO,SWPRINT    ;HANDLING, BECAUSE SYSTEM
        MOVB    #PRO,SWEXIT     ;STARTS BY PROMPTING
                                ;I.E. MAKES TTOUT.

        MOV     #BUFFER,R0      ;INIT ANSWER BUFFER
        MOV     #5*256.,R1
1$:     MOV     #ZERO,(R0)+
        SOB     R1,1$
        MOV     #RD50ZERO,NUMCOM        ;INIT BATCH COMMAND NUMBER

        MOV     #COBVEC,R5              ;GET VECTOR ADDR
        .PROTECT#IOB,R5         ;ASK FOR VECTOR PROTECTION
        BCS     PROTERR         ;VECTOR ALREADY IN USE
        MOV     #INT,(R5)+      ;SET ADDRESS OF ROUTINE
        MOV     #PRIOR,(R5)+    ;SET PRIORITY
        MOV     #COMTAB,(R5)    ;SET ADDR OF COMMUNICATION TAB
        MOV     #JOBARG,R5
        .GTJB   #IOB,R5         ;NEEDED IN .SYNC IN INT ROUTINE
        MOV     (R5),SYNBLK+WORD
        BIS     #SETIZ80,COBCSR ;AND INTERRUPT LINDA
        ADD     #POOL,LASTAB    ;INIT PTR TO END OF POOL


WLOOP:
        BIT     #IDONE,COBCSR   ;TRANSFER OF COMTAB FINISHED
        BEQ     WLOOP           ;NO, WAIT
        BIC     #IDONE,COBCSR   ;YES, CLEAR INT FROM LINDA
        BIS     #INTENB,COBCSR  ;FROM NOW ENABLE INTERRUPTS


START:
        MOV     #NBRBLK,R1      ;INIT NBRE OF BLOCKS
        MOV     #0,DISPONIBLES  ;INIT CTR OF INPUT BLOCKS
        MOV     #POOL,R2        ;POINT BLOCK POOL
M2:     CMPB    #AI,DAC(R2)     ;INPUT BLOCK ?
        BNE     M1              ;NO
        CMPB    #REQ,DAS(R2)    ;BLOCK UNDER Z80 CONTROL ?
        BNE     M1              ;NO
        ADD     #1,DISPONIBLES  ;YES, INCREMENT CTR
M1:     ADD     #ABUFLEN,R2     ;POINT NEXT BLOCK
        SOB     R1,M2           ;AT END OF POOL.
        CMP     #0,DISPONIBLES  ;NBR OF INPUT BLOCKS = 0 ?
        BNE     M3              ;NO
        JSR     PC,NEXTFREEAB   ;SEARCH FOR NEXT FREE AB
        CMP     #SET,LSIFREE    ;IS LSI FREE ?
        BNE     M4              ;NO
        MOV     CURFREEAB,R0    ;YES, GET CURRENT FREE BLOCK
        MOVB    #REQ,DAS(R0)    ;SET UNDER Z80 CONTROL
        MOVB    #AI,DAC(R0)             ;SET FOR INPUT ACTION
        BIS     #SETIZ80,COBCSR         ;INTERRUPT Z80
M3:     JSR     PC,NEXTFG       ;YES, SEARCH FG BLOCK
        CMP     #RESET,NOFG     ;SOMETHING TO DO ?
        BNE     M5              ;NO
        CLR     R1              ;YES,CLEAR R1
        MOV     CURF,R0         ;GET FG-BLOCK
        MOVB    #NOSEEN,DSEEN(R0)
        MOVB    ABLEN(R0),R1            ;GET SYSTEM CODE
        ASL     R1              ;MAKE WORD ADDRESS
        ADD     #TABSYS,R1      ;POINT CORRESP. ADDR
        JSR     PC,@0(R1)       ;GOTO CORRESP. ROUTINE
```

```
        BIS     #SETIZ80,COBCSR
M5:     .SPND
M77:    MOV     CURB,R0             ;GET CURRENT BG-BLOCK
        CMPB    #BLOCALADR,DADR(R0)     ;TEST THREE SWITCHES
        BNE     M10                 ; COBUS DRIVER, DONE, AI
        CMPB    #DONE,DAS(R0)       ;AND SO THE BG HAS NOT YET
        BNE     M10                 ;FINISHED, AND THEN WE SKIP
        CMPB    #AI,DAC(R0)         ;FURTHER SCANNING,
        BNE     M10                 ;ELSE WE SEARCH NEW WORK.
        BR      M6                  ;REASON: INIT AND CURB MAY
                                    ;POINT IMPOSSIBLE BLOCK.
M10:    JSR     PC,NEXTBG           ;YES, SEARCH NEXT BG-BLOCK
        CMP     #RESET,NOBG         ;SOMETHING TO DO FOR BG ?
        BNE     M4                  ;NO
                                    ;AFTER NEXTBG R0 POINTS
                                    ;NEW BG BLOCK.
        MOV     CURB,R1             ;BEGIN TO SET UP
        MOVB    #NOSEEN,DSEEN(R1)
        MOV     DMESLEN(R1),R2
        ADD     #ABLEN,R1           ;ARGUMENTS
        .SDATW  #IOB,R1,R2
        BCS     STOP
        .RCVDC  #IOB,#ANSWER,#BUFLEN+WORD,#COMPL
M6:     .SPND

M4:     JSR     PC,NEXTO            ;SEARCH FOR OUTPUT
        CMP     #RESET,NOO          ;SOMETHING TO OUTPUT ?
        BNE     M7                  ;NO
        MOV     CURO,R0
        MOVB    #AI,DAC(R0)
        MOVB    #DORMANT,DAS(R0)
        MOVB    #NOSEEN,DSEEN(R0)
M7:     JMP     START

STOP:
        HALT

        .PAGE
        .SBTTL  GET NEXT FREE ACTION BLOCK.


; NEXT FREE ACTION BLOCK
;************************
;
; A PARTIR DU ACTION BLOCK PONTE PAR CURFREEAB,
; ON RECHERCHE UN ACTION LIBRE (DORMANT + FREE).
; IL SERA POINTE PAR CURFREEAB.
; ON FAIT UNE FOIS LE TOUR DU POOL DES ACTION BLOCKS,
; ET SI ON NE TROUVE RIEN ON MET LE FLAG LSIFREE FULL.
;
;       PARAMS D'ENTREE: CURFREEAB
;       PARAMS DE SORTIE: CURFREEAB = AB  LIBRE TROUVE
;                         LSIFREE = FULL SI PAS DE AB LIBRE
; DETRUIT R0,R1.


NEXTFREEAB:
        MOV     #NBRBLK,R1              ;INIT NBR OF BLOCKS
N0:     MOV     CURFREEAB,R0
        CMPB    #DORMANT,DAS(R0)        ;DORMANT ?
        BNE     N1                      ;NO
        MOV     #FREE,LSIFREE           ;YES BLOCK FOUND RETURN
        BR      NRET
N1:     CMP     LASTAB,CURFREEAB        ;END OF POOL ?
        BNE     N2
        MOV     #POOL,CURFREEAB         ;RESET TO BEGIN OF POOL
        BR      N3
N2:     ADD     #ABUFLEN,CURFREEAB      ;OR GET NEXT BLOCK
N3:     SOB     R1,N0                   ;WE TURNED AROUND
        MOV     #FULL,LSIFREE           ;AND NO FREE BLOCK FOUND.
NRET:   RTS     PC

        .PAGE
        .SBTTL  GET NEXT FOREGROUND ACTION BLOCK.
```

```
; GET NEXT FOREGROUND ORDER
;****************************
;
; A PARTIR DU ACTION BLOCK POINTE PAR CURF,
; ON CHERCHE L'ACTION BLOCK SUIVANT QUI CONTIENT
; UN APPEL SYSTEME A EFFECTUER PAR F-G.
; ON FAIT UNE FOIS LE TOUR DU POOL DES ACTION BLOCKS,
; ET SI ON NE TROUVE RIEN ON MET LE FLAG NOFG SET.
;
; PARAMS D'ENTREE:      CURF = AB ACTUEL
; PARAMS DE SORTIE:     CURF = AB A TRAITER PAR F-G
;                       NOFG = SET SI RIEN TROUVE
;                              RESET SINON
; DETRUIT R0,R1.

NEXTFG:
        MOV     #NBRBLK,R1
F0:     MOV     CURF,R0
        CMPB    #FLOCALADR,DADR(R0)
        BNE     F1
        CMPB    #DONE,DAS(R0)
        BNE     F1
        CMPB    #AI,DAC(R0)
        BNE     F1
        MOV     #RESET,NOFG
        BR      FRET
F1:     CMP     LASTAB,CURF
        BNE     F2
        MOV     #POOL,CURF
        BR      F3
F2:     ADD     #ABUFLEN,CURF
F3:     SOB     R1,F0
        MOV     #SET,NOFG
FRET:   RTS     PC

        .PAGE
        .SBTTL  GET NEXT BACKGROUND ACTION BLOCK.

; GET NEXT BACKGROUND ORDER
;****************************
;
; A PARTIR DU ACTION BLOCK POINTE PAR CURB,
; ON CHERCHE L'ACTION BLOCK SUIVANT QUI CONTIENT UNE COMMANDE
; A EFFECTUER PAR LE B-G.
; ON FAIT UNE FOIS LE TOUR DU POOL DES ACTION BLOCKS,
; ET SI ON NE TROUVE RIEN ON MET LE FLAG NOBG SET.
;
; PARAMS D'ENTREE:      CURB = AB ACTUEL
; PARAMS DE SORTIE:     CURB = AB A TRAITER PAR B-G
;                       NOBG = SET SI RIEN TROUVE
;                              RESET SINON
;
; DETRUIT R0,R1

NEXTBG:
        MOV     #NBRBLK,R1
B0:     MOV     CURB,R0
        CMPB    #BLOCALADR,DADR(R0)
        BNE     B1
        CMPB    #DONE,DAS(R0)
        BNE     B1
        CMPB    #AI,DAC(R0)
        BNE     B1
        MOV     #RESET,NOBG
        BR      BRET
B1:     CMP     LASTAB,CURB
        BNE     B2
        MOV     #POOL,CURB
        BR      B3
B2:     ADD     #ABUFLEN,CURB
B3:     SOB     R1,B0
        MOV     #SET,NOBG
BRET:   RTS     PC
```

```
        .PAGE
        .SBTTL  GET NEXT OUTPUT ACTION BLOCK.

; GET NEXT OUTPUT MESSAGE
;*************************
;
; A PARTIR DU ACTION BLOCK POINTE PAR CURO,
; ON CHERCHE L'ACTION BLOCK SUIVANT QUI A DEJA ETE ENVOYE
; PAR LINDA SUR COBUS (A0+DONE).
; ON FAIT UNE FOIS LE TOUR DU POOL DES ACTION BLOCKS,
; ET SI ON NE TROUVE RIEN ON MET LE FLAG NOO SET.
;
; PARAMS D'ENTREE:      CURO = AB ACTUEL
; PARAMS DE SORTIE:     CURO = AB A LIBERER
;                       NOO = SET SI RIEN TROUVE
;                             RESET SINON
;
; DETRUIT R0,R1.

NEXTO:
        MOV     #NBRBLK,R1
00:     MOV     CURO,R0
        CMPB    #AO,DAC(R0)
        BNE     01
        CMPB    #DONE,DAS(R0)
        BNE     01
        MOV     #RESET,NOO
        BR      ORET
01:     CMP     LASTAB,CURO
        BNE     02
        MOV     #POOL,CURO
        BR      03
02:     ADD     #ABUFLEN,CURO
03:     SOB     R1,00
        MOV     #SET,NOO
ORET:   RTS     PC

        .PAGE
        .SBTTL MACROS UTILISEES PAR APPELS SYSTEMES.


.MACRO  SYSCALL X,Y

        MOV     CURF,R0
        MOV     R0,R1
        ADD     #ABLEN+BYTE,R0
        JSR     PC,ASCRD50
        CMP     #SET,ILLFIL
        BNE     X
        MOVB    #ERROR,ABLEN(R1)
        MOVB    #ERFNM,ABLEN+BYTE(R1)
        BR      Y
.ENDM
.MACRO  PTCHAN  X

        MOV     CURF,R0
        MOV     R0,R1
        ADD     #ABLEN+BYTE,R0
        MOVB    (R0),R2
        MOV     #FILPOOL,R3
        CMP     #FSTCHAN,R2
        BEQ     X
1$:     ADD     #FBLKLEN,R3
        SOB     R2,1$
.ENDM

        .PAGE
        .SBTTL  CONVERTION OF ASCII TO RAD50 CODE.

; CONVERT ASCII TO RAD50 CODE.
;*****************************
```

```
;
; CETTE ROUTINE CONVERTIT UN NOM DE FICHIER DU CODE ASCII
; EN CODE RAD50 INTERNE A RT11.
;
; PARAMS D'ENTREE:      R0 POINTE LE DEBUT DU TEXTE ASCII
; PARAMS DE SORTIE:     ILLFIL = SET SI NOM DE FICHIER ILLEGAL
;                               RESET SINON
;                       OUTFIL = DEBUT DU NOM EN RAD50
;                       R5 POINTE LE BYTE APRES LE NOM ASCII
;
; DETRUIT R0,R1,R5.

ASCRD50:
        MOV     #FILASN,R2      ;GET ASCII TEXT PTR
CONV2:  CMPB    #SPACE,(R0)     ;IS IT A SPACE
        BEQ     CONV1           ;YES
        CMPB    #COMMA,(R0)
        BEQ     CONV1
        CMPB    #ZERO,(R0)
        BEQ     CONV1
        CMPB    #TAB,(R0)
        BEQ     CONV1
        CMPB    #CR,(R0)
        BEQ     CONV1
        CMPB    #LF,(R0)
        BEQ     CONV1
        MOVB    (R0)+,(R2)+     ;NO, FILL FOR CONVERTION
        BR      CONV2           ;AND GO ON.
CONV1:  MOV     R0,R5           ;BECAUSE PR DESTROYS R0
        MOVB    #ZERO,(R2)      ;SET ZERO BYTE AT END
        .CSISPC #OUTFIL,#DEFEXT,#FILASC ;MAKES CONVERTION.
        BCC     CONV3           ;SUCCES ?
        MOV     #SET,ILLFIL
        BR      CONV4
CONV3:  MOV     #RESET,ILLFIL
        ADD     #2,SP
CONV4:  RTS     PC


        .PAGE
        .SBTTL  CREATE.


; CREATE A NEW FILE.
;*******************
;
; CREATE:
;       D'ABORD ON CONVERTIT LE NOM DU FICHIER EN CODE
;       RAD50.
; CREA1:
;       PUIS ON CHERCHE UN CANAL LIBRE.
; CREA3:
;       ON REGARDE SI LE FICHIER EXISTE DEJA.
; CREA6:
;       ON CREE UNE ENTREE POUR LE NOUVEAU FICHIER.
; CREA4:
;       ON GARNIT L'ENTETE DU FILE BLOCK ASSOCIE AU CANAL.
; CREA2:
;       ON GARNIT LA REPONSE DANS CURF.
;
; PARAMS D'ENTREE:      CURF = AB A TRAITER
; PARAMS DE SORTIE:     CURF CONTIENT LA REPONSE (AO+REQ)
; SOUSROUTINES UTILISEES:
;       - ASCRD50
;       - FREECHAN
; DETRUIT R0,R1,R2,R3,R4,R5.
;
; ATTENTION!!!!
; CONTRAIREMENT AU NOVA SI ON CREE UN NOUVEAU FICHIER,
; CELUI-CI RESTE OUVERT. POUR GARDER LA COMPATIBILITE
; AVEC LE SYSTEME COBUS EXISTANT ON SIGNALE DANS LE FILE
; BLOCK ASSOCIE A CE FICHIER QU'IL S'AGIT D'UNE CREATION
; NOUVELLE. ET LORS DU .LOOKUP QUI SUIVRA POUR CE FICHIER
; ON DOIT CONTROLER S'IL NE S'AGIT PAS D'UN NOUVEAU
; FICHIER LORSQU'ON AURA UNE ERREUR.
;
```

```
; DETRUIT R0,R1,R2,R3.

CREATE:
        SYSCALL CREA1,CREA2
CREA1:  JSR     PC,FREECHAN
        CMP     #SET,NOCHAN
        BNE     CREA3
        MOVB    #ERROR,ABLEN(R5)
        MOVB    #ERUFT,ABLEN+BYTE(R5)
        BR      CREA2
CREA3:  MOVB    (R5),R3
        .LOOKUP #IOB,R3,#FILRD50
        BCS     CREA5
        MOVB    #ERROR,ABLEN(R1)
        MOVB    #ERCRE,ABLEN+BYTE(R1)
        .CLOSE  R3
        BR      CREA2
CREA5:  TSTB    ERBYT
        BNE     CREA6
        HALT
CREA6:  .ENTER  #IOB,R3,#FILRD50,#0
        BCC     CREA4
        MOVB    ERBYT,R0
        ADD     #ERCREA,R0
        MOVB    #ERROR,ABLEN(R1)
        MOVB    (R0),ABLEN+BYTE(R1)
        BR      CREA2
CREA4:  MOVB    #OCCUPIED,DCHSTAT(R5)
        MOV     #FILRD50,R2
        MOV     R5,R3
        ADD     #DFILNAM,R3
        MOV     #RD50LEN,R4
1$:     MOV     (R2)+,(R3)+
        SOB     R4,1$
        MOVB    DCOR(R1),DSTATION(R5)
        MOV     #NF,DNEWFIL(R5)
        MOV     #OPWRIT,DOPMOD(R5)
        MOV     #OK,ABLEN(R1)
CREA2:  MOV     #ANSLEN,DMESLEN(R1)
        MOVB    #AO,DAC(R1)
        MOVB    #REQ,DAS(R1)
        RTS     PC

        .PAGE
        .SBTTL  DELETE.


; DELETE A FILE.
;***************
;
; DELETE:
;       ON TRADUIT LE NOM DU FICHIER EN CODE RAD50.
; DEL1:
;       ON CHERCHE UN CANAL LIBRE.
; DEL3:
;       ON DELETE LE FICHIER.
; DEL4:
;       ON MET LA REPONSE DANS CURF.
;
; PARAMS D'ENTREE:      CURF = AB A TRAITER
; PARAMS DE SORTIE:     CURF CONTIENT LA REPONSE
; SOUSROUTINES UTILISEES:
;       - ASCRD50
;       - FREECHAN
; DETRUIT R0,R1,R3,R5.

DELETE:
        SYSCALL DEL1,DEL2
DEL1:   JSR     PC,FREECHAN
        CMP     #SET,NOCHAN
        BNE     DEL3
        MOVB    #ERROR,ABLEN(R1)
        MOVB    #ERUFT,ABLEN+BYTE(R1)
        BR      DEL2
DEL3:   MOVB    (R5),R3
        .DELETE #IOB,R3,#FILRD50
```

```
        BCC     DEL4
        MOVB    ERBYT,R0
        ADD     #ERDEL,R0
        MOVB    #ERROR,ABLEN(R1)
        MOVB    (R0),ABLEN+BYTE(R1)
        BR      DEL2
DEL4:   MOV     #OK,ABLEN(R1)
DEL2:   MOV     #ANSLEN,DMESLEN(R1)
        MOVB    #AO,DAC(R1)
        MOVB    #REQ,DAS(R1)
        RTS     PC


        .PAGE
        .SBTTL RENAME

; RENAME.
;
;***************
;
; RENAME:
;       ON TRADUIT L'ANCIEN NOM DU FICHIER EN CODE RAD50
; REN1:
;       ON TRADUIT LE NOUVEAU NOM DE FICHIER EN CODE RAD50
; REN3:
;       RECHERCHE D'UN CANAL LIBRE
; REN4:
;       RENAME DU FICHIER
; REN5:
;       METTRE LA REPONSE DANS CURF
;
; PARAMS D'ENTREE:      CURF = AB A TRAITER
; PARAMS DE SORTIE:     CURF CONTIENT LA REPONSE
; SOUSROUTINES UTILISEES:
;       - ASCRD50
;       - FREECHAN
; DETRUIT R0,R1,R2,R3,R4,R5.

RENAME:
        SYSCALL REN1,REN2
REN1:   MOV     #FILRD50,R2
        MOV     #OLDFILRD50,R3
        MOV     #RD50LEN,R4
1$:     MOV     (R2)+,(R3)+
        SOB     R4,1$
        MOV     R5,R0
        INC     R0
        JSR     PC,ASCRD50
        CMP     #SET,ILLFIL
        BNE     REN3
        MOVB    #ERROR,ABLEN(R1)
        MOVB    #ERFNM,ABLEN+BYTE(R1)
        BR      REN2
REN3:   MOV     #FILRD50,R2
        MOV     #RD50LEN,R4
1$:     MOV     (R2)+,(R3)+
        SOB     R4,1$
        JSR     PC,FREECHAN
        CMP     #SET,NOCHAN
        BNE     REN4
        MOVB    #ERROR,ABLEN(R1)
        MOVB    #ERFNM,ABLEN+BYTE(R1)
        BR      REN2
```

```
REN4:   MOVB    @CURFB,R2
        .RENAME #IOB,R2,#OLDFILRD50
        BCC     REN5
        MOVB    ERBYT,R0
        ADD     #ERREN,R0
        MOVB    #ERROR,ABLEN(R1)
        MOVB    (R0),ABLEN+BYTE(R1)
        BR      REN2
REN5:   MOV     #OK,ABLEN(R1)
REN2:   MOV     #ANSLEN,DMESLEN(R1)
        MOVB    #AO,DAC(R1)
        MOVB    #REQ,DAS(R1)
        RTS     PC

        .PAGE
        .SBTTL  OPEN.


; OPEN A FILE FOR IN- OR OUT- PUT.
;*********************************
;
; OPEN:
; APPEND:
;       ON TRADUIT LE NOM DU FICHIER EN CODE RAD50
; OP1:
;       EST-CE-QUE C'EST UN FICHIER NOUVELLEMENT CREE?
; OP6:
;       OUVERTURE DU FICHIER.
; OP7:
;       GARNITURE DU FILE BLOCK ASSOCIE AU CANAL.
; OP9:
;       METTRE LA REPONSE DANS CURF.
;
; PARAMS D'ENTREE:      CURF = AB A TRAITER
; PARAMS DE SORTIE:     CURF CONTIENT LA REPONSE
; SOUSROUTINES UTILISEES:
;       - ASCRD50
;       - FREECHAN
; DETRUIT R0,R1,R2,R3,R4,R5.

OPEN:
APPEND:
        SYSCALL OP1,OP2
OP1:    MOV     #NBRCHAN,R0
        MOV     #FILPOOL,R2
OP4:    CMP     #NF,DNEWFIL(R2)
        BEQ     OP3
OP5:    ADD     #FBLKLEN,R2
        SOB     R0,OP4
        JSR     PC,FREECHAN
        CMP     #SET,NOCHAN
        BNE     OP6
        MOVB    #ERROR,ABLEN(R1)
        MOVB    #ERUFT,ABLEN+BYTE(R1)
        BR      OP2
OP6:    MOVB    (R5),R3
        .LOOKUP #IOB,R3,#FILRD50
        BCC     OP7
        MOVB    ERBYT,R0
        ADD     #EROP,R0
        MOVB    #ERROR,ABLEN(R1)
        MOVB    (R0),ABLEN+BYTE(R1)
        BR      OP2
OP7:    MOVB    #OCCUPIED,DCHSTAT(R5)
        MOVB    DCOR(R1),DSTATION(R5)
        CMPB    #RDOPCOD,ABLEN(R1)
        BNE     OP8
        MOV     #OPREAD,DOPMOD(R5)
        BR      OP10
OP8:    MOV     #OPWRIT,DOPMOD(R5)
OP10:   MOV     #FILRD50,R2
        MOV     R5,R3
        ADD     #DFILNAM,R3
        MOV     #RD50LEN,R4
1$:     MOV     (R2)+,(R3)+
        SOB     R4,1$
```

```
OP9:    MOV     #OK,ABLEN(R1)
        MOVB    (R5),ABLEN+WORD(R1)
        MOV     #OPLEN,DMESLEN(R1)
        BR      OP11
OP3:    MOV     #FILRD50,R3
        MOV     R2,R4
        ADD     #DFILNAM,R4
        MOV     #RD50LEN,R5
1$:     CMP     (R3)+,(R4)+
        BNE     OP5
        SOB     R5,1$
        MOV     #RESET,DNEWFIL(R2)
        MOV     R2,R5
        BR      OP9
OP2:    MOV     #ANSLEN,DMESLEN(R1)
OP11:   MOVB    #AO,DAC(R1)
        MOVB    #REQ,DAS(R1)
        RTS     PC

        .PAGE
        .SBTTL  CLOSE.

; CLOSE A FILE ON A CHANNEL.
;***************************
;
; CLOSE:
;       PRENDRE LE CANAL ET LE FILE BLOCK ASSOCIE
; CL01:
;       ENCORE QCH. A ECRIRE?
; CL03:
;       SI OUI, L'ECRIRE.
; CL04:
;       FERMER LE CANAL
;       REINITIALISER LE FILE BLOCK ASSOCIE
; CL02:
;       GARNIR LA REPONSE.
;
; PARAMS D'ENTREE:      CURF = AB A TRAITER
; PARAMS DE SORTIE:     CURF CONTIENT LA REPONSE
; DETRUIT R0,R1,R2,R3,R4,R5.

CLOSE:
        PTCHAN  CL01
CL01:   CMP     #OPWRIT,DOPMOD(R3)
        BNE     CL04
        MOV     R3,R2
        ADD     #DFILBUF,R2
        CMP     DFILPTR(R3),R2
        BEQ     CL04
        MOV     R3,R4
        ADD     #FBLKLEN,R4
        MOV     DFILPTR(R3),R2
        SUB     R2,R4
        CMP     #ZERO,R4
        BEQ     CL03
1$:     CLRB    (R2)+
        SOB     R4,1$
CL03:   ADD     #1,DBLKNBR(R3)
        MOVB    (R3),R2
        MOV     R3,R4
        ADD     #DFILBUF,R4
        MOV     DBLKNBR(R3),R5
        .WRITW  #IOB,R2,R4,#256.,R5
        BCC     CL05
        MOVB    ERBYT,R0
        ADD     #ERWRIT,R0
        MOVB    #ERROR,ABLEN(R1)
        MOVB    (R0),ABLEN+BYTE(R1)
        CMPB    #ERSPC,ABLEN+BYTE(R1)
        BEQ     CL05
        BR      CL02
CL04:   MOVB    (R3),R2
CL05:   .CLOSE  R2
        MOVB    #FREE,DCHSTAT(R3)
        MOV     #RESET,DNEWFIL(R3)
```

```
        MOV     #-1,DBLKNBR(R3)
        MOV     R3,R4
        ADD     #DFILBUF,R4
        MOV     R4,DFILPTR(R3)
        MOV     #OK,ABLEN(R1)
CLO2:   MOV     #ANSLEN,DMESLEN(R1)
        MOVB    #AO,DAC(R1)
        MOVB    #REQ,DAS(R1)
        RTS     PC

        .PAGE
        .SBTTL  READ.

; CRESET.
;*******
;
; CRESET:
;       CHERCHE LES CANAUX OUVERTS POUR CETTE STATION
; RES3:
;       FERME CES CANAUX (PAR ROUTINE CLOSE)
; RES2:
;       GARNIT LA REPONSE DANS CURF.
;
; PARAMS D'ENTREE:      CURF = AB A TRAITER
; PARAMS DE SORTIE:     CURF CONTIENT LA REPONSE
; SOUSROUTINES UTILISEES:
;       - CLOSE
; DETRUIT R0,R1,R2,R3,R4,R5.

CRESET:
        MOV     CURF,R0
        MOV     R0,R1
        MOV     #NBRCHAN,R2
        MOV     #FILPOOL,R3
RES3:   CMPB    DCOR(R0),DSTATION(R2)
        BNE     RES1
        JSR     PC,CLO1
        CMPB    #ERROR,ABLEN(R1)
        BEQ     RES2
RES1:   ADD     #FBLKLEN,R3
        SOB     R2,RES3
        MOV     #OK,ABLEN(R1)
        MOV     #ANSLEN,DMESLEN(R1)
RES2:   MOVB    #AO,DAC(R1)
        MOVB    #REQ,DAS(R1)
        RTS     PC
        .PAGE
        .SBTTL RDBYTE.


; READ A NUMBER OF BYTES IN A FILE.
;**********************************
;
; RDBYTE:
;       CHERCHER LE CANAL ET LE FILE BLOCK ASSOCIE.
; READ1:
;       INITIALISER LE TRANSFERT.
; READ4:
;       SI PLUS RIEN DANS LE FILE BUFFER,
;       ALORS LIRE LE BLOCK SUIVANT DU DISQUE.
; READ3:
;       TRANSFERT FINI?
; READ5:
;       GARNIR LA REPONSE DANS CURF.
;
; PARAMS D'ENTREE:      CURF = AB A TRAITER
; PARAMS DE SORTIE:     CURF CONTIENT LA REPONSE
; DETRUIT R0,R1,R2,R3,R4,R5.

RDBYTE:
        PTCHAN  READ1
READ1:  MOV     BYTE(R0),BYTWANT
        SWAB    BYTWANT
        CLR     BYTTRAN
```

```
        MOV     R1,DATABUF
        ADD     #ABLEN+<2*WORD>,DATABUF
READ4:  MOV     R3,R2
        ADD     #DFILBUF,R2
        CMP     R2,DFILPTR(R3)
        BNE     READ3
        ADD     #1,DBLKNBR(R3)
        MOVB    (R3),R2
        MOV     R3,R4
        ADD     #DFILBUF,R4
        MOV     DBLKNBR(R3),R5
        .READW  #IOB,R2,R4,#256.,R5
        BCC     READ3
        MOVB    ERBYT,R0
        ADD     #ERREAD,R0
        MOVB    #ERROR,ABLEN(R1)
        MOVB    (R0),ABLEN+BYTE(R1)
        MOV     #ANSLEN,DMESLEN(R1)
        CMPB    #EREOF,ABLEN+BYTE(R1)
        BNE     READ2
        MOV     BYTTRAN,ABLEN+WORD(R1)
        SWAB    ABLEN+WORD(R1)
        ADD     #2*WORD,BYTTRAN
        MOV     BYTTRAN,DMESLEN(R1)
        BR      READ2
READ3:  CMP     #ZERO,BYTWANT
        BEQ     READ5
        MOV     R3,R2
        ADD     #FBLKLEN,R2
        CMP     DFILPTR(R3),R2
        BEQ     READ6
        MOVB    @DFILPTR(R3),@DATABUF
        INC     DFILPTR(R3)
        INC     DATABUF
        INC     BYTTRAN
        DEC     BYTWANT
        BR      READ4
READ6:  MOV     R3,DFILPTR(R3)
        ADD     #DFILBUF,DFILPTR(R3)
        BR      READ4
READ5:  MOV     #OK,ABLEN(R1)
        MOV     BYTTRAN,ABLEN+WORD(R1)
        SWAB    ABLEN+WORD(R1)
        ADD     #2*WORD,BYTTRAN
        MOV     BYTTRAN,DMESLEN(R1)
READ2:  MOVB    #AO,DAC(R1)
        MOVB    #REQ,DAS(R1)
        RTS     PC

        .PAGE
        .SBTTL  READ LINE.

; RDL.
;*****
;
; RDL:
;       CHERCHER LE CANAL ET LE FILE BLOCK ASSOCIE.
; RDL1:
;       GARNIR LE MAX DE CHAR PAR LIGNE.
; RDL4:
;       SI PLUS RIEN DANS LE FILE BUFFER,
;       ALORS LIRE LE BLOCK SUIVANT DU DISQUE.
; RDL3:
;       LIGNE TROP LONGUE?
;       CHERCHER LA FIN DE LA LIGNE.
; RDL7:
;       TRANSFERT FINI.
; RDL2:
;       GARNIR LA REPONSE.
;
; PARAMS D'ENTREE:      CURF = AB A TRAITER
; PARAML DE SORTIE:     CURF CONTIENT LA REPONSE
; DETRUIT R0,R1,R2,R3,R4,R5.

RDL:
        PTCHAN  RDL1
```

```
RDL1:   MOV     #200,BYTWANT
        CLR     BYTTRAN
        MOV     R1,DATABUF
        ADD     #ABLEN+<2*WORD>,DATABUF
RDL4:   MOV     R3,R2
        ADD     #DFILBUF,R2
        CMP     R2,DFILPTR(R3)
        BNE     RDL3
        ADD     #1,DBLKNBR(R3)
        MOVB    (R3),R2
        MOV     R3,R4
        ADD     #DFILBUF,R4
        MOV     DBLKNBR(R3),R5
        .READW  #IOB,R2,R4,#256.,R5
        BCC     RDL3
        MOVB    ERBYT,R0
        ADD     #ERREAD,R0
        MOVB    #ERROR,ABLEN(R1)
        MOVB    (R0),ABLEN+BYTE(R1)
        MOV     #ANSLEN,DMESLEN(R1)
        CMPB    #EREOF,ABLEN+BYTE(R1)
        BNE     RDL2
        MOV     BYTTRAN,ABLEN+WORD(R1)
        SWAB    ABLEN+WORD(R1)
        ADD     #2*WORD,BYTTRAN
        MOV     BYTTRAN,DMESLEN(R1)
        BR      RDL2
RDL3:   CMP     #ZERO,BYTWANT
        BEQ     RDL5
        MOV     R3,R2
        ADD     #FBLKLEN,R2
        CMP     DFILPTR(R3),R2
        BEQ     RDL6
        MOVB    @DFILPTR(R3),@DATABUF
        CMPB    #CR,@DFILPTR(R3)
        BEQ     RDL7
        CMPB    #LF,@DFILPTR(R3)
        BEQ     RDL7
        CMPB    #FF,@DFILPTR(R3)
        BEQ     RDL7
        CMPB    #ZERO,@DFILPTR(R3)
        BEQ     RDL10
        INC     DFILPTR(R3)
        INC     BYTTRAN
        INC     DATABUF
        DEC     BYTWANT
        BR      RDL4
RDL7:   INC     DFILPTR(R3)
        INC     BYTTRAN
        MOV     #OK,ABLEN(R1)
        MOV     BYTTRAN,ABLEN+WORD(R1)
        SWAB    ABLEN+WORD(R1)
        ADD     #2*WORD,BYTTRAN
        MOV     BYTTRAN,DMESLEN(R1)
        BR      RDL2
RDL6:   MOV     R3,DFILPTR(R3)
        ADD     #DFILBUF,DFILPTR(R3)
        BR      RDL4
RDL10:  INC     DFILPTR(R3)
        CMP     #ZERO,BYTTRAN
        BEQ     RDL4
        BR      RDL7
RDL5:   MOVB    #ERROR,ABLEN(R1)
        MOVB    #ERLLI,ABLEN+BYTE(R1)
        MOV     #ANSLEN,DMESLEN(R1)
RDL2:   MOVB    #AO,DAC(R1)
        MOVB    #REQ,DAS(R1)
        RTS     PC

        .PAGE
        .SBTTL  WRITE.

; WRITE A CERTAIN NUMBER OF BYTES IN A FILE.
;*******************************************
```

```
;
; WRBYTE:
;       CHERCHER CANAL ET FILE BLOCK ASSOCIE.
; WRITE1:
;       INITIALISER LE TRANSFERT.
; WRITE5:
;       SI FILE BUFFER PLEIN,
;       ALORS ECRIRE BLOCK SUIVANT SUR DISQUE.
; WRITE3:
;       FIN TRANSFERT?
; WRITE6:
;       GARNIR LA REPONSE DANS CURF.
;
; PARAMS D'ENTREE:      CURF = AB A TRAITER
; PARAMS DE SORTIE:     CURF CONTIENT LA REPONSE
; DETRUIT R0,R1,R2,R3,R4,R5.

WRBYTE:
        PTCHAN  WRITE1
WRITE1: MOV     BYTE(R0),BYTWANT
        SWAB    BYTWANT
        CLR     BYTTRAN
        MOV     R1,DATABUF
        ADD     #ABLEN+<2*WORD>,DATABUF
WRITE5: MOV     R3,R2
        ADD     #FBLKLEN,R2
        CMP     DFILPTR(R3),R2
        BNE     WRITE3
        ADD     #1,DBLKNBR(R3)
        MOVB    (R3),R2
        MOV     R3,R4
        ADD     #DFILBUF,R4
        MOV     DBLKNBR(R3),R5
        .WRITW  #IOB,R2,R4,#256.,R5
        BCC     WRITE4
        MOVB    ERBYT,R0
        ADD     #ERWRIT,R0
        MOVB    #ERROR,ABLEN(R1)
        MOVB    (R0),ABLEN+BYTE(R1)
        MOV     #ANSLEN,DMESLEN(R1)
        CMPB    #ERSPC,ABLEN+BYTE(R1)
        BNE     WRITE2
        MOV     #ZERO,ABLEN+WORD(R1)
        MOV     #2*WORD,DMESLEN(R1)
        BR      WRITE2
WRITE4: MOV     R3,DFILPTR(R3)
        ADD     #DFILBUF,DFILPTR(R3)
WRITE3: CMP     #ZERO,BYTWANT
        BEQ     WRITE6
        MOVB    @DATABUF,@DFILPTR(R3)
        INC     DFILPTR(R3)
        INC     DATABUF
        INC     BYTTRAN
        DEC     BYTWANT
        BNE     WRITE5
WRITE6: MOV     #OK,ABLEN(R1)
        MOV     BYTTRAN,ABLEN+WORD(R1)
        SWAB    ABLEN+WORD(R1)
        MOV     #2*WORD,DMESLEN(R1)
WRITE2: MOVB    #AO,DAC(R1)
        MOVB    #REQ,DAS(R1)
        RTS     PC


        .PAGE
        .SBTTL  WRITE LINE.

; WRL.
;*****
;
; WRL:
;       CHERCHER LE CANAL ET LE FILE BLOCK ASSOCIE.
; 1$:
;       DETERMINER LE NOMBRE DE CHAR. DANS LA LIGNE A ECRIRE.
; WRL1:
;       INITIALISER LE TRANSFERT.
```

```
;       ALLER A LA ROUTINE WRBYTE.
;
; PARAMS D'ENTREE:      CURF = AB A TRAITER
; PARAMS DE SORTIE:     CURF CONTIENT LA REPONSE
; SOUSROUTINES UTILISEES:
;       - WRBYTE
; DETRUIT R0,R1,R2,R3,R4,R5.

WRL:
        PTCHAN  WRL1
WRL1:   CLR     BYTWANT
        MOVB    DMESLEN(R1),BYTWANT
        SUB     #WORD,BYTWANT
        MOV     R1,DATABUF
        ADD     #ABLEN+WORD,DATABUF
        JMP     WRITE5

        .PAGE
        .SBTTL DATE AND TIME.

; DATE AND TIME.
;*******************
;
; SI ON DEMANDE LA DATE OU L'HEURE, ON REND LA
; VALEUR ZERO.

DATE:
TIME:
        MOV     CURF,R1
        MOV     #OK,ABLEN(R1)
        MOVB    #ZERO,ABLEN+WORD(R1)
        MOVB    #ZERO,ABLEN+WORD+BYTE(R1)
        MOVB    #ZERO,ABLEN+WORD+WORD(R1)
        MOV     #5,DMESLEN(R1)
        MOVB    #AO,DAC(R1)
        MOVB    #REQ,DAS(R1)
        RTS     PC

; NOT IMPLEMENTED SYSTEM CALLS.
;******************************
;
; TOUS LES APPELS SYSTEMES REGROUPES ICI NE
; NECESSITENT QU'UNE REPONSE OK.

NOIMOK:
        MOV     CURF,R0
        MOV     #OK,ABLEN(R0)
        MOV     #ANSLEN,DMESLEN(R0)
        MOVB    #AO,DAC(R0)
        MOVB    #REQ,DAS(R0)
        RTS     PC

; TOUS LES APPELS SYSTEMES REGROUPES ICI NE SONT PAS
; IMPLEMENTES ET PROVOQUENT UNE ERREUR "APPEL SYSTEME
; ILLEGAL".

NOIMIC:
        MOV     CURF,R0
        MOVB    #ERROR,ABLEN(R0)
        MOVB    #ERICM,ABLEN+BYTE(R0)
        MOV     #ANSLEN,DMESLEN(R0)
        MOVB    #AO,DAC(R0)
        MOVB    #REQ,DAS(R0)
        RTS     PC


; GET NEXT FREE FILE BLOCK.
;**************************
;
; ON RECHERCHE LE PREMIER CANAL LIBRE DANS LA LISTE
; FILPOOL.
;
; PARAMS D'ENTREE:      NONE
```

```
; PARAMS DE SORTIE:     R5 POINTE LE CANAL LIBRE ET FILE BLOCK
; DETRUIT R4,R5.


FREECHAN:
        MOV     #RESET,NOCHAN
        MOV     #NBRCHAN,R4
        MOV     #FILPOOL,R5
CH0:    CMPB    #OCCUPIED,DCHSTAT(R5)
        BNE     CH1
        ADD     #FBLKLEN,R5
        SOB     R4,CH0
        MOV     #SET,NOCHAN
CH1:    RTS     PC


        .PAGE
        .SBTTL  ROUTINES DU BATCH COBUS.

; ROUTINES BATCH POUR COBUS.
;*****************************
;
; A PARTIR DE CET ENDROIT ON TROUVE LES ROUTINES UTILISEES
; PAR LE KEYBOARD MONITOR DU RT-11 POUR TRAITER LES
; DEMANDES DE BATCH PAR COBUS.

; SWITCH.
;********
;
; ICI SE FAIT UN AIGUILLAGE ENTRE LES ROUTINES
; DU SYSTEME OU CELLES DU BATCH COBUS.


FB$TTIN:TSTB    SWTTIN          ;ORIG OR USER ROUTINE?
        BEQ     CO$TTIN         ;ON 0 GOTO USER ROUTINE
J$TTIN: ADD     (PC),PC         ;ON 1 GOTO SYSTEM ROUTINE
O$TTIN: .-.
FB$TTOUT:TSTB   SWTTOUT
        BEQ     CO$TTOUT
J$TTOUT:ADD     (PC),PC
O$TTOUT:.-.
FB$EXIT:TSTB    SWEXIT
        BNE     J$EXIT
        JMP     CO$EXIT
J$EXIT: ADD     (PC),PC
O$EXIT: .-.
FB$PRINT:TSTB   SWPRINT
        BNE     J$PRINT
        JMP     CO$PRINT
J$PRINT:ADD     (PC),PC
O$PRINT:.-.

        .PAGE
        .SBTTL  TTYIN.


; TTYIN ROUTINE.
;*****************
;
; CETTE ROUTINE TRANSMET LES CARACTERES DU MESSAGE
; VERS LE KEYBOARD MONITOR CARACTERE PAR CARACTERE.
; LA PREMIERE FOIS ON DOIT ALLER LIRE UN MESSAGE
; DANS LE FOREGROUND. ON DETECTE LES CR ENVOYES
; PAR COBUS ET ON REMPLACE PAR LF POUR TERMINER
; UN MESSAGE.

CO$TTIN:CLRB    SWTTOUT         ;SET SWITCHES FOR
        CLRB    SWPRINT         ;USER HANDLING
        CMP     #FIRST,FLAG
        BNE     TTI10
        JSR     PC,GETCOM
        CLR     FLAG
TTI10:
```

```
        CMPB    #CR,@ADR
        BEQ     TTI1
        MOVB    @ADR,R0
TTI2:   INC     ADR
        JMP     COBRT0
TTI1:   MOV     #LF,R0
        BR      TTI2


        .PAGE
        .SBTTL  TTYOUT.

; TTYOUT ROUTINE.
;******************
;
; CETTE ROUTINE EST OFFECTUEE PAR LE SYSTEME A LA FIN
; D'UNE COMMANDE. DONC AUSSI AVANT QU'ON COMMENCE A
; TRAITER LES COMMANDES DE COBUS. LA PREMIERE FOIS
; ON  VA CHERCHER UN MESSAGE.
; LES AUTRES FOIS ON DETECTE LES POINTS, ET ON
; ECRIT LES REPONSES SUR DISQUE PUIS ON LES
; ENVOIE AU FOREGROUND. APRES ON DEMANDE
; UN NOUVEAU MESSAGE AU FOREGROUND

CO$TTOUT:

        CMP     #FIRST,FLAG
        BEQ     TTO1
        CMPB    #STAR,R0
        BNE     TTO10
        JMP     COBRTI
TTO10:
        CMPB    #POINT,R0
        BEQ     TTO11
        MOVB    R0,@PTR
        INC     PTR
        JMP     COBRTI
TTO11:  JSR     PC,WRMESS
        JSR     PC,RESP
TTO1:   CLR     FLAG
        JSR     PC,GETCOM
        JMP     COBRTI


        .PAGE
        .SBTTL  PRINT.

; PRINT ROUTINE.
;******************
;
; ON RECUPERE LES LONGUES REPONSES DU SYSTEME, ET
; ON LES METS DANS LE BUFFER DE REPONSE. ON RAJOUTE
; UN CR POUR LA FIN DU MESSAGE.


CO$PRINT:
        CMPB    #200,(R0)
        BEQ     PR1
        CMPB    #ZERO,(R0)
        BEQ     PR2
        MOVB    (R0)+,@PTR
        INC PTR
        BR      CO$PRINT
PR2:    MOVB    #CR,@PTR
        INC     PTR
PR1:    JMP     COBRTI


        .PAGE
        .SBTTL  EXIT.


; EXIT ROUTINE.
```

```
;****************
;
; CETTE ROUTINE N'EST PAS UTILISEE.
; ELLE POURRA SERVIR POUR RECUPERER LES ABORTS
; DES COMMANDES DE COBUS.

CO$EXIT:
        JMP     J$EXIT          ;FINISH EXIT

        .PAGE
        .SBTTL  GET NEXT MESSAGE FROM FG.


; SOUSROUTINE GETCOM.
;
; ON ATTEND UN MESSAGE DU FOREGROUND, ET ON LUI
; ATTRIBUE UN NUMERO. ON CONVERTIT CE NUMERO EN
; CODE RAD50. PUIS ON FAIT AUSSI LES INITIALISATIONS
; NECESSAIRES POUR TRAITER UNE COMMANDE.

GETCOM: .RCVDW  #IOB,#COM-WORD,#BUFLEN/2        ;WAIT FOR NEXT MESS,
                                                ;TAKE MAXLEN AND PLACE
                                                ;IT INTO COM
        BCS     COMERR
        MOV     #COM,ADR                        ;INIT COMMAND POINTER
        INC     NUMCOM
        CMP     #RD50EIGTH,NUMCOM
        BEQ     GET1
        BR      GET2
GET1:   MOV     #RD50ZER0,NUMCOM
GET2:   MOV     NUMCOM,R0                       ;RD50*50*50
        MOV     #THREE,R1
1$:     ASL     R0
        SOB     R1,1$
        MOV     R0,R2
        MOV     #TWO,R1
2$:     ASL     R0
        SOB     R1,2$
        ADD     R0,R2
        ADD     #135600,R2
        MOV     R2,MESSEXT
        MOV     NUMCOM,R0
        SUB     #RD50ZERO,R0
        MOV     R0,ASCINUM
        MOV     #ZERO,BLKNBR
        RTS     PC
COMERR: HALT                                    ;ERROR

                                                ;ON FG -> BG TRANSMISSION

        .PAGE
        .SBTTL  WRITE ANSWER ON DISK

; SOUSROUTINE WRMESS.
;
; ECRIT LA REPONSE A UNE COMMANDE SUR LE DISQUE.
; CETTE REPONSE EST MISE DANS UN FICHIER SOUS LE
; NOM DE MESS.NUMERO DE COMMANDE.


WRMESS:
        .ENTER  #IOB,#BATCHAN,#MESSFILE,#5
        BCC     WR1
        MOVB    ERBYT,R0
        ADD     #ERCREA,R0
        MOV     #BANSWER,R1
        MOVB    #ERROR,(R1)
        MOVB    (R0),BYTE(R1)
        BR      WRRET
WR1:    MOV     #ZERO,R5
        MOV     #BUFFER,R4
WR3:    .WRITW  #IOB,#BATCHAN,R4,#256.,R5
```

```
        BCC     WR2
        MOVB    ERBYT,R0
        ADD     #ERWRIT,R0
        MOV     #BANSWER,R1
        MOVB    #ERROR,(R1)
        MOVB    (R0),BYTE(R1)
        BR      WRRET
WR2:    INC     R5
        ADD     #256.*WORD,R4
        CMP     #SIX,R5
        BNE     WR3
WRRET:  .CLOSE  #BATCHAN
        MOV     #OK,BANSWER
        MOV     ASCINUM,ANSNUM
        RTS     PC

SENDER: HALT

; SOUSROUTINE RESP.
;
; RENVOIE LA REPONSE A UN MESSAGE AU FOREGROUND.
; CETTE REPONSE CONTIENT LE NUMERO ATTRIBUE
; A CETTE COMMANDE. PUIS ON VIDE LE BUFFER DE
; REPONSE.


RESP:
        .SDATC  #IOB,#BANSWER,#LEN,#FGGOT
        BCS     SENDER
        MOV     #BUFFER,R0
        MOV     #5*256.,R1
1$:     MOV     #ZERO,(R0)+
        SOB     R1,1$
        MOV     #BUFFER,PTR
        RTS     PC

FGGOT:
        RETURN
        .PAGE
        .SBTTL  COMMON RETURN TO SYSTEM.

; ROUTINES COMMUNES POUR RETOURNER AU KEYBOARD
; MONITOR APRES LES QUATRE ROUTINES DE E/S.


COBRT0: MOV     R0,(SP)         ;PUSH R0 ON STACK
COBRTI: CLR     R2              ;CLEAR REG 2
        MOV     @#SYSPTR,R0     ;GET PTR TO RMON
        ADD     EMTRTN(R0),R0   ;POINT TO EMT RETURN ROUTINE
        JMP     (R0)            ;GOTO SYSTEM EMT RETURN

        .PAGE
        .SBTTL POINTEURS ET BUFFERS.


; POINTERS
;**********

CURF:   .WORD   POOL    ;CURRENT FOREGROUND AB POINTER
CURB:   .WORD   POOL    ;CURRENT BACKGROUND AB POINTER
CURO:   .WORD   POOL    ;CURRENT OUTPUT AB POINTER
CURFREEAB:
        .WORD   POOL    ;CURRENT FREE AB
NOFG:   .WORD   RESET   ;NO MORE FOREGROUND ORDERS
NOBG:   .WORD   RESET   ;NO MORE BACKGRPUND ORDERS
NOO:    .WORD   RESET   ;NO MORE ANSWERS TO OUTPUT
ILLFIL: .WORD   RESET   ;ILL FILE NAME SWITCH
LSIFREE:.WORD   FREE    ;SATURATION INDICATOR (ALLOWS ONLY OUTPUT)
DISPONIBLES:
        .WORD   0       ;NUMBER OF INPUT BUFFERS
CRFLAG: .WORD   0
NOCHAN: .WORD   0
IOB:    .BLKW   10      ;EMT ARGUMENT LIST
LASTAB: .WORD   NBRBLK-1*ABUFLEN        ;LAST AB POINTER
SYNBLK: .BLKW   5
```

```
        .WORD   -1,0
JCBARG: .BLKW   8.


; POINTERS FOR ORDER TREATMENT.
;******************************

CURFB:   .WORD   FILPOOL ;CURRENT FILE BLOCK
NCFB:    .WORD   RESET   ;NO FILE BLOCK SWITCH
DEFEXT:  .BLKW   3       ;MAY CONTAIN DEFAULT EXTENSIONS
OLDFILRD50:      .BLKW   2*RD50LEN       ;PLACE FOR RENAMING
OUTFIL:  .BLKW   17      ;PLACE TO RECEIVE RAD50 CONVERTED NAMES
FILRD50: .BLKW   30      ;PLACE TO GET CONVERTED NAME
FILASC:  .ASCII  /DX1:/  ;ONLY AUTHORIZED DEVICE FOR FILES
FILASN:  .BLKB   11      ;PLACE TO SET ASCII FILE NAME
         .EVEN

BYTWANT:         .WORD   0
BYTTRAN:         .WORD   0
DATABUF:         .WORD   0


         .PAGE
         .SBTTL  POOLS.

; BLOCKS.
;*********


ERCREA:  .BYTE   ERFNO
         .BYTE   ERSPC
ERDEL:   .BYTE   ERFNO
         .BYTE   ERDLE
EROP:    .BYTE   ERFNO
         .BYTE   ERDLE
ERREN:   .BYTE   ERFNO
         .BYTE   ERDLE
ERWRIT:  .BYTE   ERSPC
         .BYTE   ERRFIL
         .BYTE   ERFNO
ERREAD:  .BYTE   EREOF
         .BYTE   ERRFIL
         .BYTE   ERFNO

COMTAB:                  ;COMMUNICATION TABLE
CTPOOL:          .WORD   POOL
CTNBRBLK:        .WORD   NBRBLK

FILPOOL:
         .IRP    I,<0,1,2,3,4,5,6,7,10,11,12,13,14,15,16>
         .BYTE   I
         .BYTE   FREE
         .BLKW   RD50LEN
         .WORD   0       ;COBUS STATION
         .WORD   0
         .WORD   0
         .WORD   -1
         .WORD   .+WORD
         .BLKW   BUFLEN
         .ENDM


ANSWER:
BCLEN:   .WORD   0               ;LENGTH OF BG ANSWER
RESCOD:  .WORD   0               ;CODE OF ANSWER
ANSBUF:  .BLKW   BUFLEN          ;TEXT OF ANSWER


TABSYS:  .WORD   CREATE          ;TABLE OF ADDRESSES
         .WORD   DELETE
         .WORD   RENAME
         .WORD   NOIMOK
         .WORD   NOIMOK
         .WORD   OPEN
```

```
        .WORD   APPEND
        .WORD   CLOSE
        .WORD   CRESET
        .WORD   RDBYTE
        .WORD   RDL
        .WORD   WRBYTE
        .WORD   WRL
        .WORD   NOIMIC
        .WORD   NOIMIC
        .WORD   NOIMIC
        .WORD   TIME
        .WORD   DATE
        .WORD   NOIMIC
        .WORD   NOIMIC
        .WORD   NOIMIC
        .WORD   NOIMIC
        .WORD   NOIMIC
        .WORD   NOIMIC
        .WORD   NOIMIC

POOL:   .REPT   NBRBLK          ;POOL OF 20 BLOCKS
        .WORD   0               ;BUFFER PTR
        .WORD   0               ;MESSAGE LENGTH
        .BYTE   AI              ;ACTION CODE
        .BYTE   0               ;COBUS DRIVER
        .BYTE   0               ;ADDR OF CORRESPONDANT
        .BYTE   DORMANT         ;ACTION STATUS
        .WORD   0               ;SEEN BY LINDA FLAG
        .BLKB   BUFLEN          ;BUFFER
        .ENDR


        .NLIST  MEB     ;STOP MACRO EXTENSION LISTING



        .EVEN


ADR:    .WORD   COM     ;COMMAND POINTER
EFFLEN: .WORD   0       ;LENGTH OF TRANSMITTED MESSAGE
COM:    .BLKB   BUFLEN  ;COMMAND OR MESSAGE


BANSWER:.WORD   0
ANSNUM: .WORD   0
        .BLKW   BUFLEN/2
FLAG:   .WORD   FIRST
        .EVEN
ASCINUM:        .WORD   0
NUMCOM: .WORD   0
BLKNBR: .WORD   0
MESSFILE:
        .RAD50  /DX1/
        .RAD50  /MES/
        .RAD50  /S/
MESSEXT:.WORD   0

        .PAGE
        .SBTTL  VARIABLES.
SWTTIN: .BYTE   0               ;TTIN SWITCH
SWTTOUT:.BYTE   0               ;TTOU SWITCH
SWPRINT:.BYTE   0               ;PRINT SWITCH
SWEXIT: .BYTE   0               ;EXIT SWITCH

; PLACE TO PUT RESPONSE FOR FG
PTR:    .WORD   BUFFER
BUFFER: .BLKW   5*256.


        .NLIST  MEB

        .END    INITIAL
```