MASTER EN SCIENCES INFORMATIQUES

Contribution à la mise au point d'un système d'acquisition de données

Debrun, Jacques

Award date:
1984

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique

Année Académique 1983-1984

CONTRIBUTION A LA MISE AU POINT D'UN SYSTEME D'ACQUISITION DE DONNEES

DEBRUN Jacques

Mémoire présenté en vue de l'obtention du grade de Licencié et Maître en Informatique.
Promoteur de mémoire: Monsieur le Professeur Van Bastelaer.

Je tiens à remercier vivement Monsieur P. Van Bastelaer, promoteur de ce mémoire, qui par ses conseils judicieux et son constant dévouement m'a aidé à réaliser ce travail, qu'il trouve ici l'expression de ma profonde gratitude.
J'associe de tout coeur à ces remerciements Monsieur André Michel et Monsieur Guy Daelen.

TABLE DES MATIERES

CHAPITRE I

CHAPITRE II

CHAPITRE I

CHAPITRE I. INTRODUCTION

I. 1. POSITION DU PROBLEME

Aujourd'hui, l'informatique est devenue un outil privilégié pour les scientifiques expérimentateurs: c'est grâce à elle qu'un nombre impressionnant d'expériences complexes ont pu être menées à bien.
Que ce soit pour effectuer des mesures, contrôler des paramètres de l'expérience, changer certaines conditions expérimentales à des instants bien précis, stocker les résultats de mesures et les traiter de manière à faciliter leur interprétation, l'ordinateur est devenu le compagnon fidèle de tout expérimentateur digne de ce nom.

Dans de nombreux laboratoires de recherche, la réalisation des expériences est contrôlée par des micro-ordinateurs, voire même des minis. Le traitement des mesures est, quant à lui, confié à des ordinateurs de taille plus importante. (cfr figure I.1.)
La gestion d'une expérience à l'aide d'un système informatique se heurte à deux difficultés fondamentales: la première est celle de l'interfacage de l'ordinateur de contrôle avec le dispositif expérimental; la deuxième difficulté est relative au traitement des mesures. Le système informatique assurant ce service étant en général différent de celui contrôlant l'acquisition des mesures, il est nécessaire d'effectuer un transfert de données entre des deux systèmes.

Le but de ce mémoire est de donner une solution logicielle aux deux problèmes que nous venons d'évoquer. Ce travail a été demandé par le Département de Chimie des Facultés N.D. de la Paix à Namur.
Nous avons développé deux produits différents: le premier consiste en un ensemble de routines utilitaires, écrites en assembleur, qui permettent à un programme utilisateur, écrit en Fortran, de réaliser des échanges de données avec des dispositifs expérimentaux.
Le deuxième programme, écrit en Pascal, permet la connexion du système informatique du laboratoire, au DEC-20 du Centre de Calcul, afin de pouvoir, entre autres, transférer des fichiers de données entre ces deux systèmes. (cfr figure I.2.)
Ces programmes sont destinés à être exécutés sur des micro-ordinateurs, sous système d'exploitation CP/M, et, moyennant des modifications mineures, sur des machines équipées d'un microprocesseur Z-80.

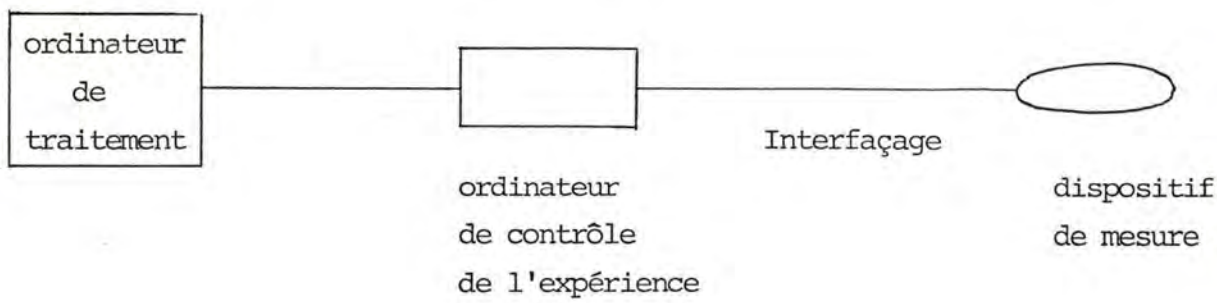


Figure I.1.: Dispositif général.

En ce qui concerne la structure de ce travail, le chapitre suivant I.2. est consacré à la description du matériel utilisé pour la réalisation de ces deux logiciels. Les logiciels de base utilisés, tels que système d'exploitation, éditeur, compilateur, sont présentés au chapitre I.3..

La deuxième partie du mémoire porte sur les spécifications des logiciels que nous avons développés. L'architecture de chacun des logiciels est le sujet de la troisième partie. La quatrième partie traite de la philosophie des tests relatifs aux logiciels développés.
Dans la cinquième partie, nous évoquons les problèmes rencontrés dans le développement du projet. Nous montrons, dans la sixième partie, l'état final du travail, les limitations et les extensions possibles de celui-ci. Nous terminerons par quelques conclusions. Le lecteur peut trouver en annexe, le mode d'emploi des logiciels, l'ensemble des modules organiques avec leurs spécifications, la liste des programmes, ainsi que les schémas électroniques des circuits d'acquisition de données.

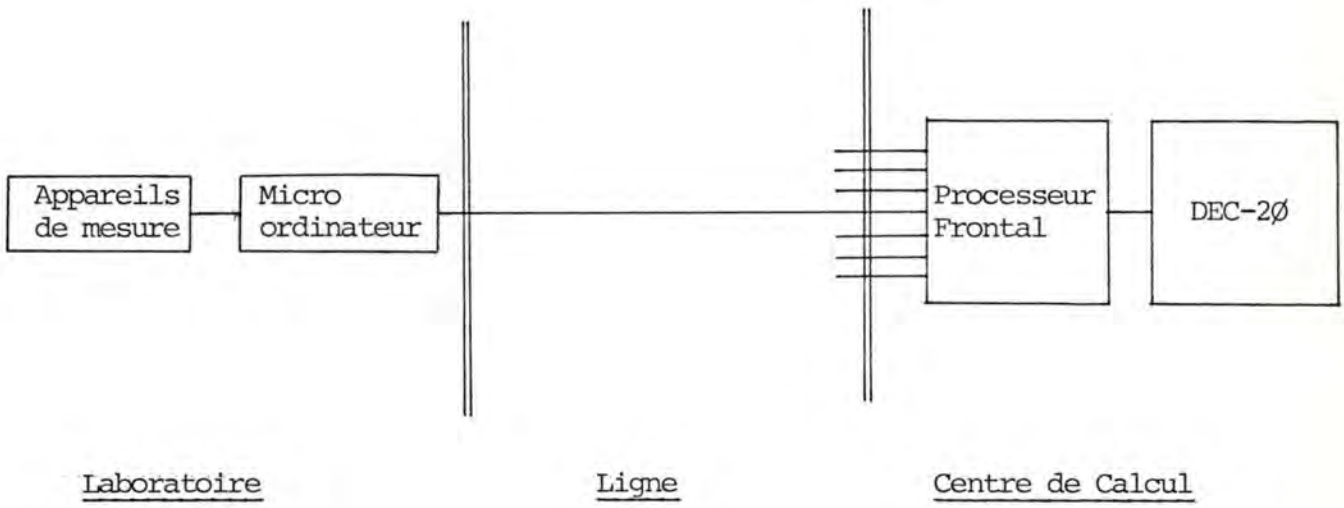


Figure I.2.: configuration générale du matériel.

I.2. DESCRIPTION DU MATERIEL

I.2.1. Ordinateur de contrôle.

I.2.1.1. Description générale.

L'ordinateur de contrôle utilisé pour la réalisation de ce mémoire, appartient à la catégorie des micro-ordinateurs. Il s'agit d'une machine de marque Bigboard dont voici les principales caractéristiques:

- unité centrale: microprocesseur Z-80, architecture 8 bits
- horloge: 2 MHz
- mémoire centrale: 64 Koctets RAM + 4 Koctets de ROM occupée par le moniteur.
- interface: - 2 circuits d'entrée-sortie parallèles, appelés PIO, de marque Zilog, équipés de deux ports de 8 bits parallèles.
  - 1 circuit d'entrée-sortie série, appelé SIO, de marque Zilog, équipés de 2 ports série.
- horloge temps réel: CTC de Zilog.
- mémoire de masse: double unité de disquettes de 8 pouces de 250 Koctets

Un des deux circuits PIO (PIO 1 dans figure I.3.) est utilisé par les disquettes; l'autre est disponible pour d'autres périphériques.

Un des ports du SIO est réservé pour le terminal TELEVIDEO 912 C.

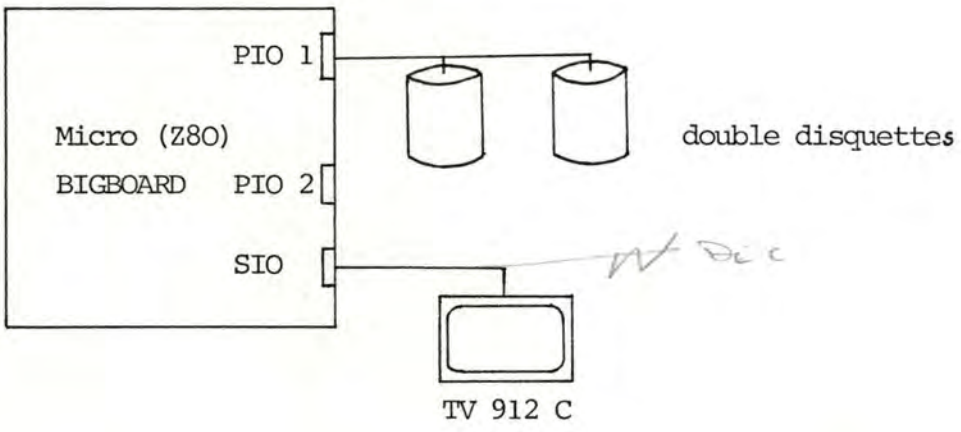


Figure I.3.: schéma du système informatique de contrôle.

I.2.1.2. Circuit d'entrée/sortie parallèle

Un circuit d'entrée/sortie parallèle PIO comporte deux ports parallèles, appelés A et B, de 8 bits chacun. Les deux ports d'un PIO peuvent être programmés en différents modes:

Port A: il peut travailler en - mode Ø correspondant à l'entrée d'une donnée de 8 bits
- mode 1 correspondant à la sortie d'une donnée de 8 bits
- mode 2 ou mode bidirectionnel
- mode 3 ou mode de contrôle, dans lequel chaque bit d'un port peut être programmé soit en entrée, soit en sortie.

Port B: il peut travailler seulement en mode Ø, 1 ou 3.
Le mode bidirectionnel n'est pas accessible.

Il existe cependant une contrainte quant à la programmation des ports: lorsque le port A est placé en mode bidirectionnel, le port B ne peut être programmé qu'en mode de contrôle.
Le tableau ci-dessous reprend les différentes possibilités de programmation d'un circuit PIO.

Port A	Port B
mode Ø	mode Ø
mode Ø	mode 1
mode Ø	mode 3
mode 1	mode Ø
mode 1	mode 1
mode 1	mode 3
mode 2	mode 3
mode 3	mode O
mode 3	mode 1
mode 3	mode 3

I.2.1.3. circuit d'entrée/sortie série.

Le circuit SIO comporte deux ports d'entrée/sortie (également désignés par A et B) ainsi qu'un ensemble de registres de contrôle, qui doivent être convenablement initialisés avant toute opération d'entrée/sortie.
A chaque port du SIO est associé un banc de ces registres. Ces registres contiennent les caractéristiques de la transmission série, à savoir:

- vitesse de la transmission
- nombre de bits de stop
- nombre de bits par caractères
- mode de transmission des caractères: synchrone ou asynchrone.

Le port B du SIO est occupé par la liaison du micro-ordinateur avec le terminal (cfr supra). Le port A est réservé pour la connexion de la ligne téléphonique du DEC-2Ø.

I.2.2. Problèmes relatifs aux systèmes d'interfaçage entre systèmes informatiques et instruments de mesure.

L'échange d'informations entre des appareils de mesure et des systèmes informatiques pose de nombreux problèmes tant sur le plan technique que sur le plan logiciel.
Devant l'importance du développement des instruments de mesure et des systèmes informatiques et vu la difficulté croissante de permettre une communication entre eux, il devint nécessaire de concevoir un système d'interfaçage universel.

Face à l'évolution de ces systèmes, une normalisation de ces derniers devint nécessaire, principalement devant la volonté des utilisateurs de pouvoir connecter des instruments issus de constructeurs différents, et ceci sans nécessiter d'adaptation particulière.

Dans ce cadre, des travaux menés parallèlement par des constructeurs européens et américains permirent de définir une norme, approuvée par le bureau de normalisation I.E.E.E. en 1974.
Cette norme, connue sous le nom de norme I.E.E.E.-488, régit les aspects électriques, mécaniques et fonctionnels d'un système d'interfaçage reposant sur une structure de bus. Ce bus par référence à la norme, porte le nom de bus I.E.E.E.-488. L'avantage du système de bus consiste dans sa souplesse lors du transfert d'informations: les différents éléments de la configuration étant interconnectés en parallèle grâce au bus, un appareil peut adresser un autre appareil, voire même plusieurs autres appareils de façon à leur envoyer des informations.
Il est également aisé d'incorporer un (ou plusieurs éventuellement) organe de contrôle pour assurer la gestion du bus.

I.2.3. Etude succinte de la norme I.E.E.E.-488

I.2.3.1. Principes généraux

Le bus IEEE-488, également appelé HP-IB ou encore IEC-625, comprend un ensemble de 16 lignes sur lequel il est possible de connecter jusqu'à 15 périphériques. Chacun de ces derniers est raccordé en parallèle sur 16 lignes du bus. Un périphérique peut échanger de l'information avec un ou plusieurs périphériques (cfr figure I.4.).

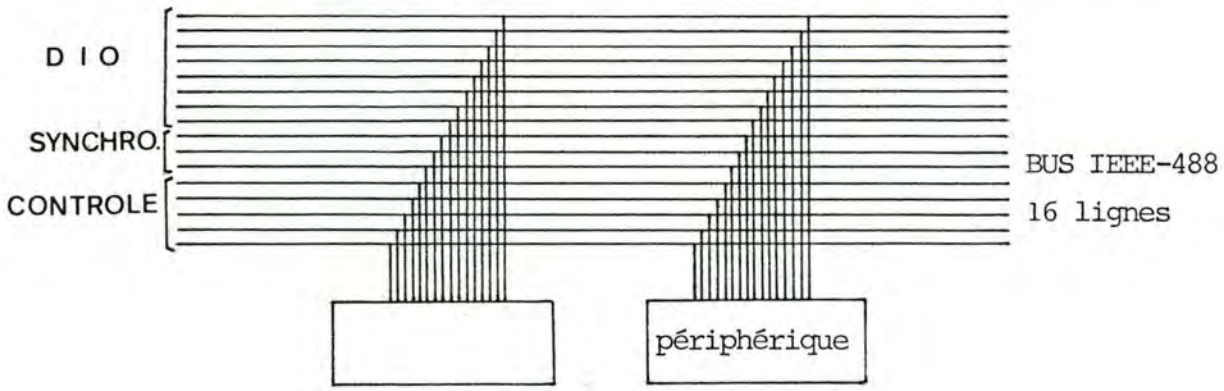


Figure I.4.: Bus IEEE-488

Les périphériques raccordés au bus peuvent avoir trois fonctions distinctes soit de manière permanente, soit de manière transitoire:

- un périphérique est capable de recevoir des données venant d'autres périphériques. Il est donc "écouteur" ou, dans la terminologie IEEE-488, "listener". Il s'agit par exemple de lecteur de cartes, multimètre, imprimante, écrans de visualisation.
- un périphérique est capable d'envoyer des données à d'autres. Il est "parleur" ou "talker": par exemple, un multimètre, une unité à disques.
- un périphérique est capable de gérer les communications se déroulant sur le bus: il s'agit d'un contrôleur.

La plupart des périphériques possèdent alternativement les deux premières fonctions; ils peuvent, suivant les circonstances, être talker ou listener: c'est le cas, par exemple, d'une imprimante qui est capable de signaler qu'elle est à court de papier, une unité de disquettes qui peut soit lire soit écrire des informations, un multimètre à qui l'on peut indiquer qu'il doit mesurer des tensions, des courants ou des résistances et qui ensuite peut donner le résultat de ses mesures.

Le rôle du contrôleur, tenu généralement par un ordinateur, consiste à gérer les échanges d'information entre les périphériques connectés au bus. Il désigne, parmi l'ensemble des périphériques celui qui doit fournir les informations (talker) et celui ou ceux qui doivent les recevoir (listener).

A un instant donné, il ne peut y avoir sur le bus qu'un seul talker, ceci afin d'éviter un mélange des messages émis, mais, plusieurs listeners peuvent écouter simultanément un même message.

I.2.3.2. Structure du bus

Du point de vue fonctionnel, le bus est divisé en trois parties (cfr figure I.4.).

1. un groupe de 8 fils (notés DIOsur la figure I.4.) est utilisé pour transmettre des données sous forme "bit parallèle, byte série". Normalement, le code utilisé pour représenter les données est le code ASCII à 7 moments: le huitième bit peut être utilisé comme bit de parité. Les données sont échangées de façon asynchrone, suivant un protocole que nous décrirons au paragraphe I.2.3.7.

2. un groupe de 3 fils dont le rôle est de permettre la synchronisation des appareils en communication.

Les 3 lignes sont appelées:
- DAV pour DAta Valid (donnée valide)
- NRFD pour Not Ready For Data (pas prêt pour recevoir la donnée)
- NDAC pour Not Data ACcepted (donnée non acceptée).

3. un groupe de 5 fils dont le rôle est d'assurer le contrôle du bus.
   Les 5 lignes sont:
   - ATN pour ATtentioN (attention)
   - REN pour Remote ENable (autorisation de télécommande)
   - IFC pour InterFace Clear (libération du bus)
   - EOI pour End Or Identify (fin ou identification)
   - SRQ pour Service ReQuest (demande de service).

I.2.3.3. Adresses physiques

Chaque périphérique connecté au bus possède une adresse, fixée par l'utilisateur à l'aide d'un ensemble d'inverseurs placés à l'arrière des périphériques. Le nombre de ces inverseurs est limité à 5, ce qui permet de constituer 32 adresses physiques différentes; parmi celles-ci, 31 seulement sont utilisables: la justification en sera donnée plus loin.
Les adresses physiques vont en binaire de la valeur 00000 à 11110, c'est-à-dire de 0 à 30 en décimal.

I.2.3.4. Déroulement d'un transfert

Un transfert de données se déroule en deux phases:
1. le contrôleur désigne le ou les périphériques entre le(s)quel(s) l'échange va se dérouler: rappelons qu'il peut y avoir au plus un talker mais qu'il peut y avoir plusieurs listeners.
   Au cours de cette phase, que nous désignerons par phase d'adressage, le contrôleur désigne non seulement les périphériques concernés par le transfert, mais également le rôle (talker ou listener) que vont tenir les périphériques désignés.

Il peut également procéder à l'envoi de commandes à tous, ou seulement à certains périphériques. Nous décrivons cette phase d'adressage au paragraphe I.2.3.5..

2. le contrôleur cède ensuite la place au talker, qu'il a sélectionné lors de la première phase. Les données sont alors échangées entre le talker et le(s) listener(s). Cette phase, que nous dénommerons phase d'échange, est décrite au paragraphe I.2.3.6.

I.2.3.5. Phase d'adressage.

Le contrôleur, lorsqu'il choisit les partenaires pour le transfert, place la ligne ATN au niveau logique vrai, puis, envoie sur le bus de données (lignes DIO) un mot de 8 bits, que l'on peut désigner par "adresse logique", dont la structure est la suivante: les 5 bits les moins significatifs de ce mot représentent l'adresse physique du périphérique.
Le bit de poids le plus fort est mis à zéro, et les deux bits suivants vont permettre de préciser le rôle du périphérique dans l'échange qui va se produire: 01 correspond au rôle de listener et 10 correspond au rôle de talker.
Illustrons ceci par un exemple: pour indiquer au périphérique 6 qu'il va être listener, le contrôleur place sur les lignes des données l'octet
0 01 00110 ce qui correspond au caractère ASCII & .

En fonction du rôle tenu par un périphérique au cours d'un échange, on peut distinguer deux groupes d'adresses logiques: le groupe d'adresses des listeners (listener address group ou LAG) et le groupe d'adresses des talkers (talker address group ou TAG).
Les adresses appartenant au LAG vont de la valeur binaire 0010000 à la valeur 00111110; l'adresse 00111111 joue un rôle particulier que nous expliquerons au paragraphe I.2.3.8. Quant aux adresses du TAG, elles vont de la valeur 01000000 à 01011110.

L'adresse 01011111 a elle aussi une fonction particulière: nous l'étudierons en I.2.3.8.
Au cours de la phase d'adressage, le contrôleur peut également procéder à l'envoi de commandes.
Nous parlerons de cette possibilité au paragraphe I.2.3.8.

I.2.3.6. Phase d'échange

Une fois les partenaires désignés, suite à la phase d'adressage, l'échange de données peut commencer. Le contrôleur signale que l'échange peut débuter en plaçant la ligne ATN au niveau logique faux.
Le talker qui a été choisi, entame alors l'échange des données avec le ou les listeners. La fin de l'échange est signalée au contrôleur par le talker, en plaçant la ligne EOI à la valeur logique vraie.

I.2.3.7. Protocole de transfert d'informations.

Nous avons vu, dans les deux paragraphes précédents, que les lignes de données DIO transportent différentes informations: lors de la phase d'adressage, les lignes DIO permettent au contrôleur d'envoyer des adresses à tous les périphériques présents sur le bus. Les lignes DIO servent également au talker pour envoyer des données aux listeners, pendant la phase d'échange. Enfin, les lignes DIO sont utilisées par le contrôleur, pendant la phase d'adressage, pour envoyer des commandes aux périphériques connectés au bus: des exemples de commandes seront donnés au paragraphe suivant.

Toutes ces informations sont transférées sur le bus via un protocole que nous allons maintenant étudier.

Il est bon de préciser, avant toute chose, la terminologie que nous allons utiliser; nous désignerons par le terme "émetteur" un dispositif dont la fonction va être d'envoyer des informations sur le bus de données:

un émetteur est donc, suivant les circonstances , soit le contrôleur en train d'envoyer des commandes ou des adresses aux périphériques, soit un talker, désigné comme tel par le contrôleur, en train d'envoyer des données à des listeners.
Nous désignerons par le terme "récepteur" un dispositif dont la fonction est de recevoir des informations sur le bus de données: il s'agit donc, soit d'un périphérique qui lit les commandes ou adresses envoyées par le contrôleur, soit d'un listener en train de recevoir des données d'un talker.

Le protocole repose sur les deux lois suivantes:

1. Un émetteur ne parle pas si un récepteur n'est pas prêt à écouter.
2. Un émetteur parle à la cadence à laquelle le plus lent des récepteurs peut l'écouter.

Le protocole est établi grâce aux trois lignes du bus de synchronisation.
La signification de chacune de ces trois lignes est:

- DAV (DAtaValid): lorsque ce signal est vrai, cela signifie que l'information présente sur le bus DIO est valide. Cette ligne est placée au niveau vrai par l'émetteur, après que celui-ci ait placé l'information sur les lignes DIO.
  Ceci permet de signaler au(x) récepteur(s) qu'il(s) peu(ven)t lire l'information.

- NRFD (Not Ready For Data): lorsque ce signal est faux, cela veut dire que le récepteur est prêt pour recevoir une nouvelle information. Lorsque ce signal est vrai, le récepteur n'est pas encore prêt pour recevoir une nouvelle information. Lorsqu'il y a plusieurs récepteurs, tant que tous les récepteurs ne sont pas prêts pour recevoir une nouvelle information, NRFD reste vrai. Lorsque tous les récepteurs sont prêts, NRFD devient faux. La ligne NRFD du bus est le résultat d'une opération logique "ou" sur chaque ligne NRFD propre à chaque périphérique. (cfr figure I.5.)

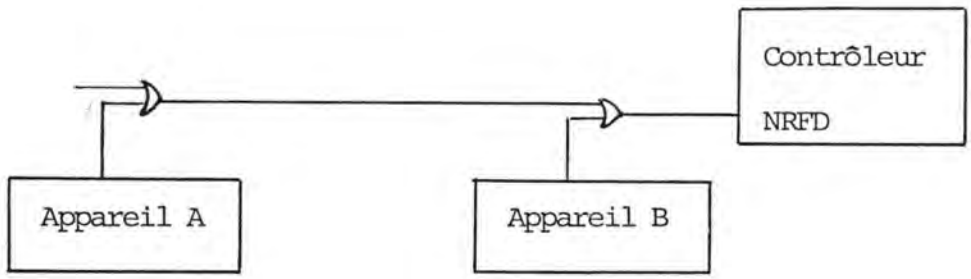


Figure I.5.: représentation symbolique de la ligne NRFD.

- NDAC (Not Data ACcepted): lorsque ce signal est faux, cela signifie que le récepteur a accepté l'information. Tant que cette dernière n'est pas lue, le signal NDAC reste vrai. Si plusieurs récepteurs doivent lire la même information, c'est le récepteur le plus lent qui conditionne le passage du signal NDAC de l'état vrai à l'état faux.
  Tant que tous les récepteurs n'ont pas accepté une information, NDAC reste vrai.
  Lorsque tous les récepteurs ont enfin lu l'information, NDAC passe à l'état faux.
  Comme pour la ligne NRFD, la ligne NDAC est cablée en "ou" logique.

Examinons à présent le déroulement d'une communication entre un émetteur et plusieurs récepteurs:

1. Au début de la communication, le signal DAV est faux: les informations ne sont pas valides.
2. Les lignes NRFD et NDAC sont mises au niveau logique vrai par les récepteurs: ceux-ci ne sont pas prêts pour recevoir une donnée, et les données ne sont pas acceptées.
3. L'émetteur teste l'état des lignes NRFD et NDAC: si NRFD et NADC sont toutes les deux fausses, c'est qu'il y a un problème, et le transfert est avorté.
4. L'émetteur place l'information sur les lignes DIO.

5. Les récepteurs signalent qu'ils sont prêts à la recevoir, en activant la ligne NRFD.
   Lorsque tous les récepteurs sont prêts, NRFD passe à la valeur fausse.
6. L'émetteur place DAV au niveau vrai, pour signaler que la donnée est disponible.
7. Les récepteurs placent NRFD vrai pour signaler à l'émetteur qu'ils ne sont pas prêts pour recevoir une nouvelle information.
8. Les récepteurs acceptent l'information présente sur les lignes DIO, grâce à la ligne NDAC.
   NDAC passe à l'état faux lorsque tous les récepteurs ont lu l'information.
9. L'émetteur met DAV au niveau logique faux pour indiquer que l'information présente sur les lignes DIO n'est plus valide.
10. Les récepteurs détectent que l'information n'est plus valide , et, par conséquent, ils placent NDAC à l'état vrai.

L'organigramme représentant un transfert d'informations entre un émetteur et un récepteur est présenté ci-après (figure I.6.).
Un chronogramme du transfert est également montré à la figure I.7.:
le bus IEEE-488 travaille en logique négative; le niveau logique vrai correspond à une tension basse (inférieur ou égal à 0.8 volts), tandis que le niveau logique faux correspond à une tension haute (supérieur ou égal à 2 Volts).
La numérotation des différentes étapes correspond à celle utilisée dans le texte explicatif.

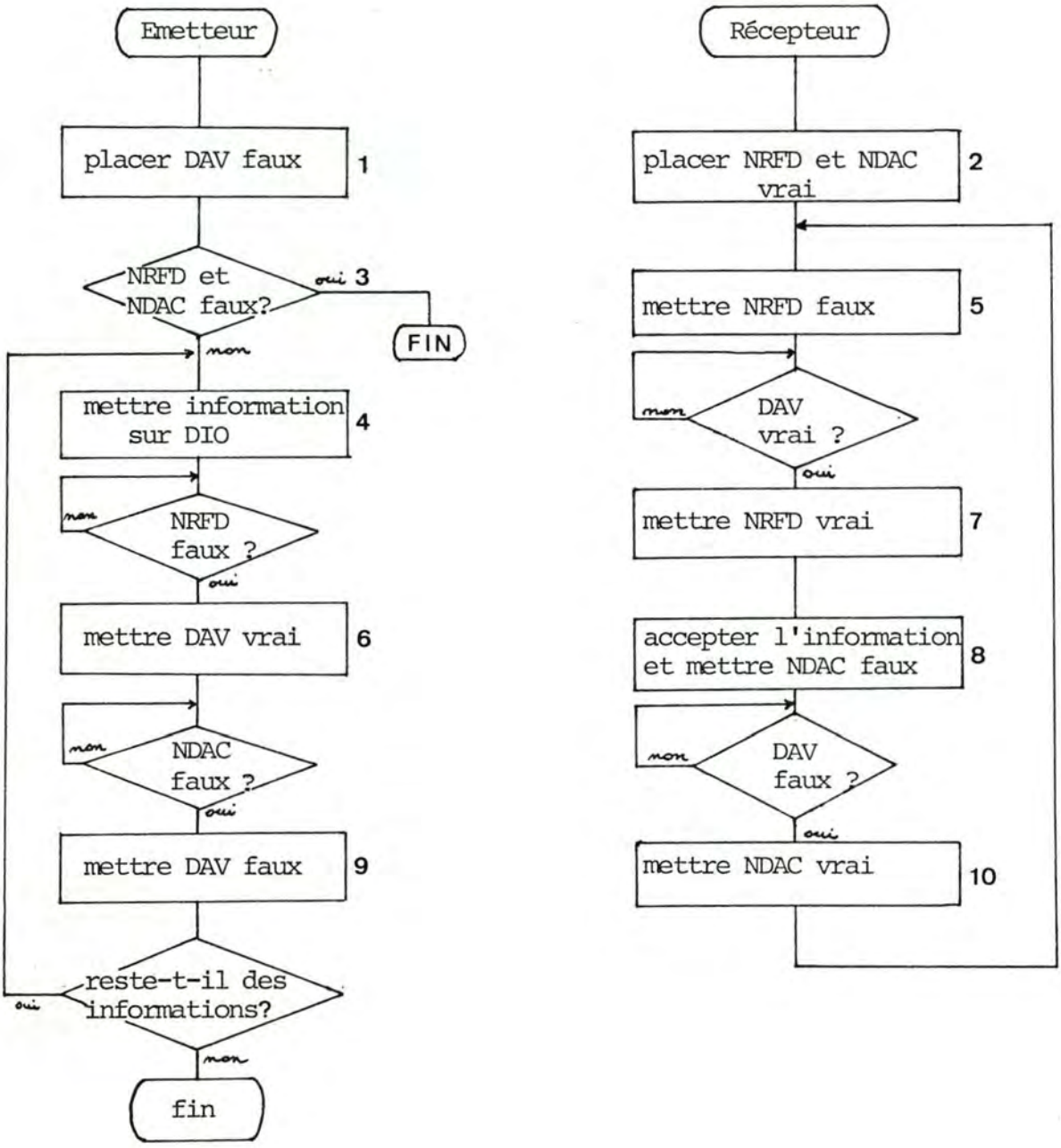


Figure I.6.: organigramme du protocole de transfert.

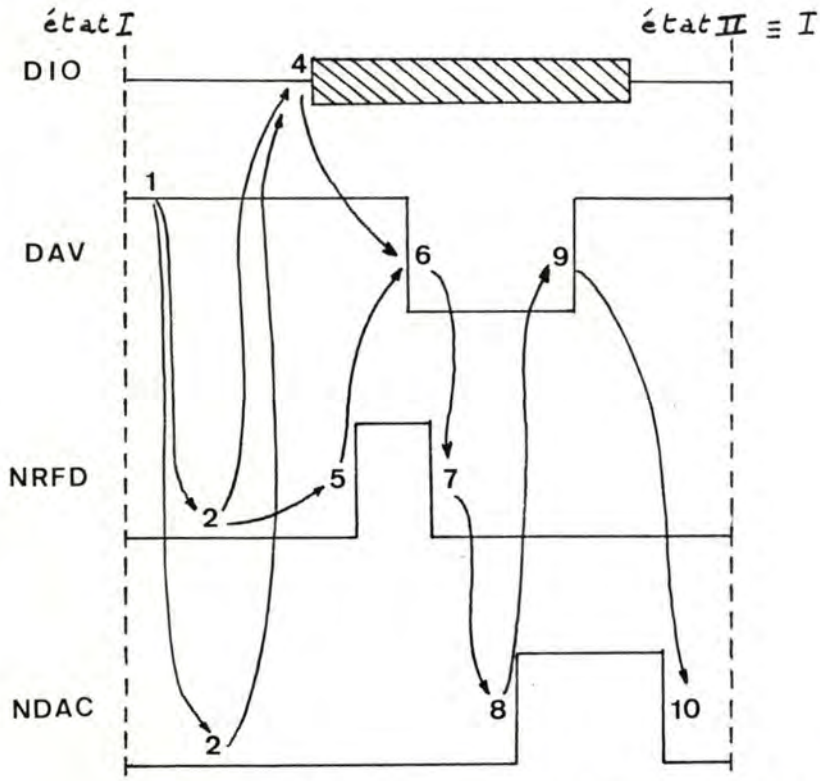


Figure I.7.: chronogramme du protocole de transfert.

I.2.3.8. Commandes particulières.

Au cours de la phase d'adressage, c'est-à-dire lorsque la ligne ATN est placée au niveau logique vrai par le contrôleur, ce dernier peut procéder à l'envoi d'adresses logiques (cfr.1.2.3.5.): il peut également envoyer des commandes. La norme IEEE-488 définit un certain nombre de commandes. Seuls deux d'entre celles-ci vont retenir notre attention.

La première commande que nous allons étudier est celle désignée par UNL ou Unlisten et représentée par la valeur binaire 00111111.
Sa fonction est de désactiver tous les listeners actifs.

A priori, la commande UNL n'est pas indispensable: on pourrait imaginer que tous les listeners potentiels écoutent tous les messages circulant sur le bus.

Cette commande a malgré cela deux raisons d'être: premièrement, un récepteur lent (imprimante par exemple) ralentirait considérablement et inutilement une communication qui ne lui serait pas explicitement destinée (contrôleur communiquant avec un disque par exemple). Deuxièmement, il pourrait interpréter de façon incorrecte un message explicitement destiné à un autre et qui pourrait modifier de façon imprévisible son comportement ultérieur.

Examinons un scénario de transfert illustrant la fonction de UNL.

EOI	ATN	DIO	Commentaires
			A l'instant initial, la configuration comporte un contrôleur et 3 périphériques d'adresses physiques 1, 2 et 3. Aucun périphérique n'est adressé.
	vrai		Début de la phase d'adressage
	vrai	00100001	Adressage du périphérique 1 comme listener
	vrai	01000011	Adressage du périphérique 3 comme talker
	faux		Phase d'échange entre 1 et 3 (3⟶1)
	faux	Donnée 1	
	faux	Donnée n	
vrai			Fin de l'échange signalée au contrôleur
	vrai		Le contrôleur procède à une nouvelle phase d'adressage
	vrai	UNL	Désactivation du seul listener actif, c'est-à-dire le périphérique 1
	vrai	00100010	Adressage du périphérique 2 comme listener
	faux		Phase d'échange de 3 vers 2
	faux	Donnée n+1	Echange de k nouvelles données de 3 vers 2
	faux	Donnée n+k	
vrai			Fin de l'échange

Dans un scénario identique au précédent, mais au cours duquel on ne procéderait pas à l'envoi de UNL, le deuxième groupe de données (n + 1 à n + k) seraient échangées entre le périphérique talker 3 et les 2 périphériques 1 et 2.

L'autre commande, appelée UNT ou Untalk, et représentée par la valeur bianire 01011111, permet de désactiver le talker actif sur le bus.
A ce moment, il n'y a plus de talker adressé sur le bus. La norme IEEE-488 impose qu'il n'y ait qu'un seul talker actif sur le bus à un moment donné.
C'est pourquoi il est prévu que lorsque l'on adresse un talker en vue de réaliser un échange de données, tout autre talker précédemment adressé est automatiquement désactivé.
Il est donc possible de désactiver un talker, en envoyant, pendant la phase d'adressage, soit la commande UNT, soit une adresse de talker inutilisée.

I.2.3.9. Signaux de contrôle

A T N :

Ainsi que nous l'avons déjà vu, la ligne ATN permet de différentier la nature des informations se trouvant sur les lignes de données DIO; lorsque ATN est vrai, le bus est en mode de commande; c'est le contrôleur qui est actif et les périphériques attendent les commandes:

Ces dernières peuvent être des
- adresses de talkers
- adresses de listeners
- commandes particulières telles que UNL ou UNT.

Lorsque ATN est faux, le bus est dans le mode de données, ce qui signifie que tous les appareils qui ont été adressés lorsque le bus était en mode de commande, peuvent accéder au bus de données soit pour recevoir, soit pour envoyer des données, selon le rôle que le contrôleur leur a assigné. Les autres appareils qui n'ont pas été adressés sont isolés du bus de données.

S R Q :

Certains périphériques ont la possibilité de demander le service du contrôleur, en plaçant la ligne SRQ au niveau logique vrai.
Le contrôleur a la liberté de choisir s'il va traiter ou non la demande.

E O I :

La ligne EOI est utilisée par un talker pour signaler au contrôleur la fin d'un transfert de données.

R E N :

La ligne REN permet au contrôleur de sélectionner le mode de travail, soit local soit à distance (ou télécommande), pour un périphérique raccordé au bus: le mode local permet à un périphérique d'être configuré ou programmé par l'intermédiaire de son panneau de commande; le mode à distance consiste à configurer le périphérique par l'intermédiaire du contrôleur.
Pour placer le périphérique en mode télécommande, le contrôleur place REN vrai et adresse le périphérique.

I F C :

Cette ligne est activée uniquement par le contrôleur.
Lorsqu'il place cette ligne au niveau vrai, toute activité sur le bus est immédiatement arrêtée et tous les périphériques présents sur le bus se désactivent d'eux-même.

I.2.4. Problèmes relatifs à la connexion d'un bus IEEE-488 sur notre système informatique

I.2.4.1. Nature des besoins

Le système informatique, que nous avons décrit en I.2.1., joue dans notre application le rôle de contrôleur: c'est donc lui qui a pour mission d'assigner des rôles aux périphériques présents dans le configuration et susceptibles de réaliser les opérations de transfert.

Dans l'application envisagée, les échanges de données se déroulent soit du contrôleur vers un ou plusieurs périphériques (listeners), soit d'un périphérique (talker) vers le contrôleur. Les échanges de données entre périphériques ne sont pas envisagées dans l'immédiat.

Il se pose néanmoins un problème délicat: le système informatique dont nous disposons est dépourvu d'interface IEEE-488. Le seul circuit d'entrée/sortie disponible est un circuit d'entrée/sortie parallèle (Pio 2 dans la figure I.3.).

Ces considérations nous ont amenés à considérer la conception d'une interface, entre le contrôleur et le bus, simplifiée de façon à n'offrir au contrôleur qu'un sous-ensemble des fonctions prévues dans la norme, mais répondant aux besoins décrits.

I.2.4.2. Spécifications relatives à l'interface.

Par conséquent, vu les restrictions que nous nous sommes imposées par rapport à l'ensemble des fonctions normalement prévues pour un contrôleur, nous n'avons pas jugé nécessaire de concevoir une interface capable de gérer toutes les lignes du bus IEEE-488.

L'interface entre le bus et le contrôleur est un élément du système complètement passif: son seul rôle est d'assurer la compatibilité électrique entre les lignes d'entrée/sortie du micro-ordinateur contrôleur et les lignes du bus.

Les seules lignes du bus qui sont assurées par l'interface sont les lignes de données DIO, les lignes de synchronisation NRFD, NDAC, DAV, ainsi que la ligne de contrôle ATN.

La figure I.8. montre le schéma de l'interface que nous avons adopté.

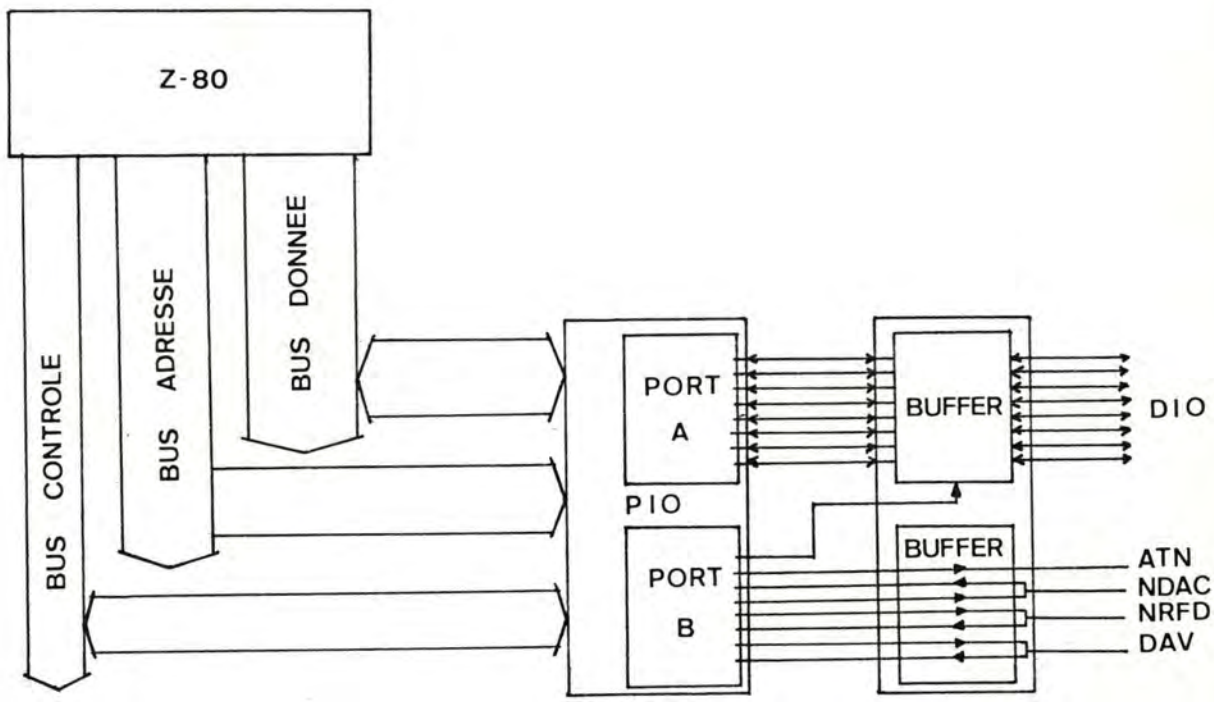


Figure I.8.: Interface micro-ordinateur/bus.

L'interface est simplement constituée de circuits transceivers (référence MC 3446) permettant d'adopter électriquement les lignes du PIO (décrit en I.2.1.2.) et les lignes du bus.

Les lignes DIO sont raccordées au port A car c'est le seul port du DIO qui est bidirectionnel.
Les circuits transceivers du port A sont commutés en mode entrée ou en mode sortie par le bit de poids fort du port B: lorsque ce bit est à 1, les transceivers du port A sont en mode d'entrée de données; lorsqu'il est nul, les transceivers sont en mode de sortie.

Le port B est programmé en mode de contrôle: chaque bit du port est donc configuré soit en entrée soit en sortie, conformément à ce qui est indiqué à la figure I.8.

I.2.4.3. Limitations dues à l'interface

Nous avons déjà fait remarquer que le contrôleur ne dispose pas de l'ensemble des fonctions prescrites par la norme.
Ces limitations viennent de la structure adoptée pour l'interface, mais elles ne sont de toute façon pas gênantes pour l'application que nous étudions.

Le fait de supprimer certaines lignes du bus de contrôle (SRQ, EOI, REN et IFC) entraîne une perte de certaines fonctions pour le contrôleur:

SRQ: le contrôleur n'a pas la possibilité de détecter une demande de service effectuée par un périphérique. La nature des périphériques que nous utilisons ne requiert aucunement ce type de possibilité.

IFC: le contrôleur perd la faculté de mettre fin à un échange se déroulant entre 2 périphériques. Comme dans notre application, le contrôleur décide du nombre de données qu'il veut recevoir ou envoyer et que les échanges de données ont lieu uniquement entre contrôleur et périphérique, cette limitation n'est pas gênante.

EOI: la même remarque que pour la ligne IFC s'applique ici: c'est le contrôleur qui décide du nombre de données qu'il veut recevoir et les échanges de données ne se déroulent qu'entre contrôleur et périphérique.

REN: en supprimant cette ligne, le contrôleur n'est pas capable de sélectionner le mode de commande local ou le mode télécommandé d'un périphérique. Or les périphériques que nous utilisons ne disposent pas de panneau de commande: ils sont exclusivement commandés par le contrôleur.

I.2.5. Problèmes liés à l'acquisition de données analogiques

I.2.5.1. Nature des besoins

Un nombre important d'appareils de mesure et de visualisation utilisés dans des applications scientifiques et techniques sont dépourvus d'entrées ou de sorties digitales.
Ceci constitue une sérieuse limitation lorsque l'on désire automatiser des expériences grâce à un système informatique de contrôle.
Ce système, utilisé pour la conduite de l'expérience, sert également à la mémorisation et éventuellement au traitement des résultats.
Dans ce but, une conversion analogique/digitale s'avère indispensable.
De même, le traitement d'une donnée, émanant d'un système informatique, destinée à un appareil de visualisation (traceur de courbes par exemple) nécessite une conversion digitale/analogique de cette donnée.

Ceci nous a amené à la conception d'un ensemble de périphériques capables entre autres d'effectuer ces opérations de conversion.
Ces périphériques sont connectés au système informatique par le bus et l'interface décrits précédemment.

I.2.5.2. Spécifications générales des périphériques

Trois périphériques ont été élaborés, chacun ayant une fonction bien précise (cfr figure I.9).

- périphérique de conversion analogique/digitale: assure la conversion d'une mesure envoyée par un appareil, en une donnée binaire que le système informatique peut alors traiter.
  Ce périphérique comporte un convertisseur A/D.

- périphérique de conversion digitale/analogique: il s'agit d'un convertisseur D/A; ce périphérique permet la transmission d'une tension (grandeur analogique) à un appareil, à partir de la conversion d'une donnée binaire, envoyée par le système informatique de contrôle.

- périphérique de multiplexage qui autorise le système informatique à choisir l'(les) appareil(s) impliqué(s) dans une acquisition ou un envoi de données.
  La sélection d'une (ou plusieurs) des bornes d'entrées/sorties du multiplexeur est assurée grâce à l'envoi d'une donnée par le contrôleur au périphérique de multiplexage.

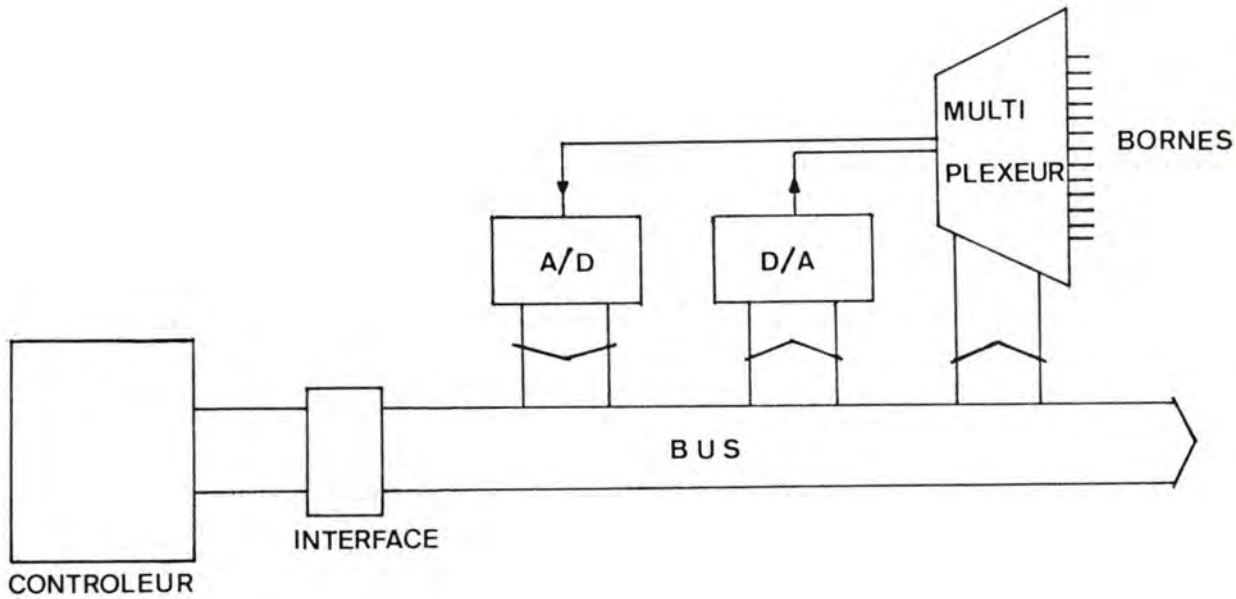


Figure I.9. dispositif d'acquisition/transmission de données.

Chacun de ces périphériques possède une adresse, conformément aux spécifications de la norme IEEE-488. Le convertisseur A/D est un talker, tandis que le convertisseur D/A et le multiplexeur sont des listeners. Tout transfert d'informations se passant entre le contrôleur et les périphériques, se déroule suivant le protocole défini par la norme IEEE-488.

I.2.5.3. Convertisseur A/D.

Ce circuit est un convertisseur analogique/digital de 12 bits (référence AD574) équipé d'un buffer permettant de le raccorder à des bus de 8, 12 ou 16 bits.

Il peut travailler en 4 échelles différentes:

- unipolaire: 0 à + 10 Volts
  0 à + 20 Volts
- bipolaire: - 5 à + 5 Volts
  - 10 à + 10 Volts

Une donnée de 12 bits à la sortie du convertisseur peut être lue soit comme un mot de 12 bits, soit comme deux mots de 8 bits.
Dans ce deuxième cas, le premier mot sortant du A/D consiste en les bits de poids fort de la donnée de 12 bits, et le deuxième mot en les 4 bits de poids faible de la donnée, cadrés à gauche et suivi de 4 bits mis à Ø.

Exemple: soit la donnée 010011010100 présente à la sortie du A/D
le convertisseur peut sortir cette donnée sous la forme

01001101	1e octet sorti
0100 | 0000	2e octet sorti

Ceci constitue une particularité très intéressante: nous pouvons, en effet, la mettre à profit pour sortir ces octets sur les lignes de données du bus décrit en I.2.4.
Dernière particularité de ce convertisseur: le codage des données.
Ce convertisseur emploie un système de codage de données appelé complément Offset Binary.
Considérons le cas où le convertisseur travaille sur l'échelle $[-10, +10]$.
Désignons par B une valeur binaire et par V une grandeur analogique présente à l'entrée du A/D, alors V et B sont liés par la fonction suivante:

$$B = \text{partie entière} \left(2047 - \frac{2048 * V}{10} \right)$$

Le tableau suivant reprend quelques valeurs binaires typiques et leurs correspondants analogiques: l'échelle choisie est celle allant de - 10 Volts à + 10 Volts.

digital	décimal	analogique
000000000000	0	+ 9.9951 Volts
011111111111	2047	0.0000 Volts
100000000000	2048	- 0.0049 Volts
111111111111	4095	-10.0000 Volts

Le convertisseur est câblé de façon à travailler sur l'échelle s'étendant de - 10 à + 10 Volts, pour toute l'application envisagée.

I.2.5.4. Convertisseur D/A

Ce circuit est également en convertisseur de 12 bits (référence DAC 80). Il peut travailler en 5 échelles différentes:

unipolaire:	0	à	+ 5 Volts
	0	à	+ 10 Volts
bipolaire:	- 2.5	à	+2.5 Volts
	- 5	à	+ 5 Volts
	- 10	à	+ 10 Volts

Le DAC 80 utilise le même codage de données que le AD 574.
Sa fonction de conversion est l'inverse du AD 574 soit dans le cas où l'échelle de travail du convertisseur est - 10, + 10

$$V = \frac{10}{2048} \quad (2047 - B)$$

Comme pour le convertisseur A/D, le D/A est câblé pour travailler entre - 10 et + 10 Volts.

Contrairement au AD574, le DAC 80 ne peut être raccordé qu'à un bus de 12 bits. Or le bus de données que nous utilisons ne comporte que 8 lignes. Pour bénéficier de la pleine précision du convertisseur, un dispositif électronique a été conçu pour que, à partir de deux mots de 8 bits lus consécutivement sur le bus, il puisse composer une donnée de 12 bits, qui peut alors être convertie par le DAC 80.
La première donnée envoyée au dispositif doit avoir le format suivant:

10 - - - - - - où les signes - représentent les 6 bits de poids faible du mot de 12 bits qui est la donnée à convertir

La deuxième donnée a la forme:

01 + + + + + + où les signes + représentent les 6 bits de poids fort du mot de 12 bits.

Exemple: soit à convertir la donnée 010011 | 010100
Le premier octet à envoyer au dispositif est 10 | 010100
et le deuxième octet 01 | 010011

I.2.5.5. Multiplexeur

Le multiplexeur a comme rôle de sélectionner, parmi un ensemble d'appareils connectés aux bornes d'entrée/sortie du multiplexeur, celui ou ceux qui vont être impliqués dans une communication avec le système de contrôle.
Le multiplexeur est programmé à partir d'un octet lu sur le bus de données. La valeur de cet octet permet de choisir une ou plusieurs des 13 bornes d'entrée/sortie (numérotées Ø à 12 dans la figure I.10).

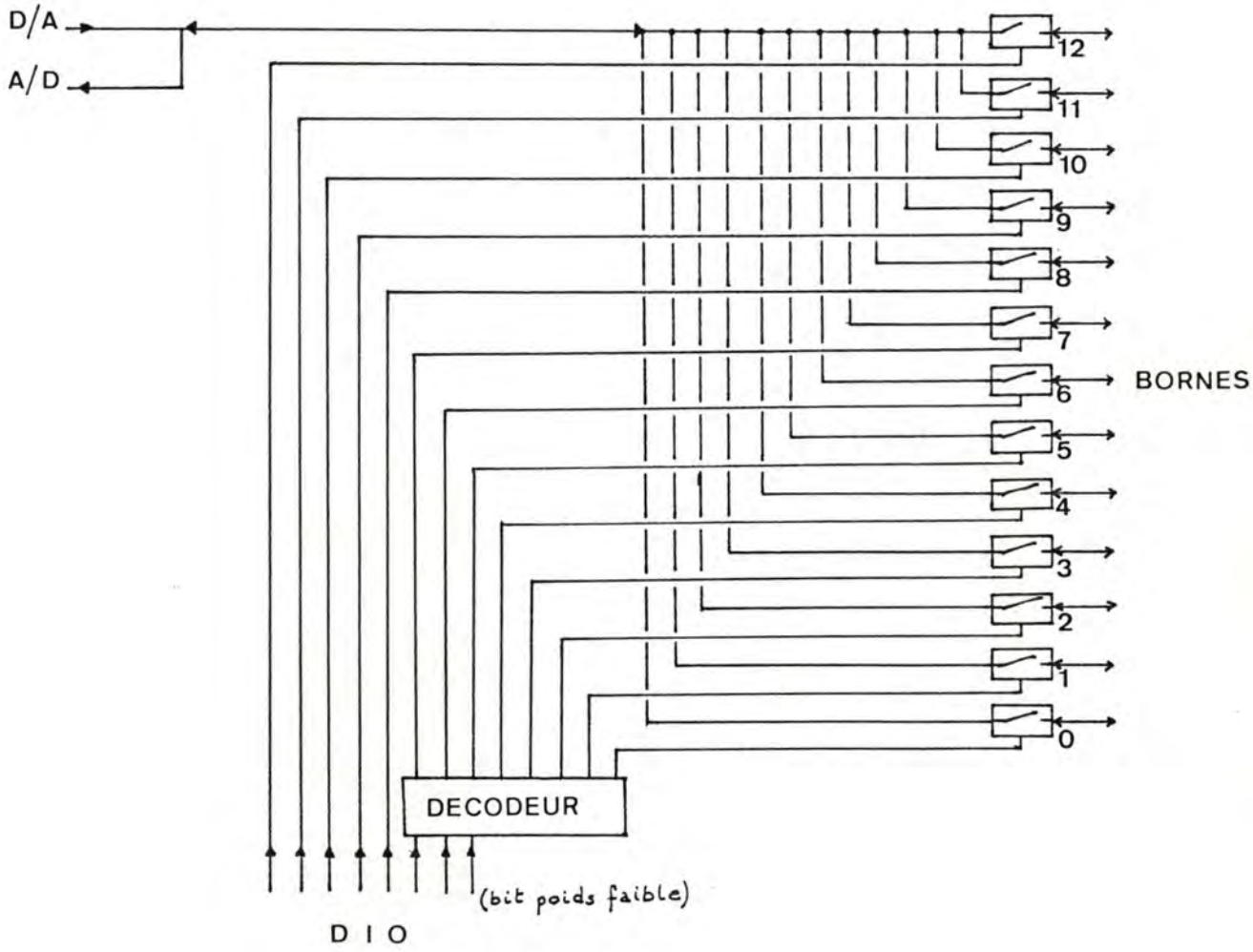


Figure I.10. Multiplexeur.

La figure I.10 montre le schéma de principe du multiplexeur. Celui-ci est raccordé aux lignes DIO du bus; sur les 3 bits de poids faible est raccordé un décodeur "3 vers 8" dont les sorties sont raccordées à des relais commandant les bornes d'entrées/sorties numérotées de Ø à 7: le relais est fermé lorsque le bit qui le commande est à 1, sinon il est ouvert. Lorsqu'un relais est fermé, la borne correspondante est mise en communication soit avec l'entrée du convertisseur A/D, soit avec la sortie du convertisseur D/A.
Les 5 bits de poids fort de la donnée présente sur les DIO commandent les bornes numérotées de 8 à 12.
Nous pouvons établir un tableau reprenant les différents cas de sélection de bornes d'entrées/sorties.

Données sur DIO		Borne
binaire	décimal	sélectionnée
00000000	0	0
00000001	1	1
00000010	2	2
00000011	3	3
00000100	4	4
00000101	5	5
00000110	6	6
00000111	7	7
00001000	8	8
00010000	16	9
00100000	32	10
01000000	64	11
10000000	128	12

Les valeurs intermédiaires des données non reprises dans le tableau peuvent être utilisées pour sélectionner plusieurs bornes d'entrées/ sorties.
Par exemple, les valeurs de données, sur la ligne DIO, comprises entre 8 et 15 permettent de sélectionner la borne 8 et l'une quelconque des bornes 0 à 7, en fonction des 3 bits de poids faible de cette donnée.
Supposons maintenant que la donnée présente sur le bus de données soit égale à 193, soit11000001, en binaire; les bornes sélectionnées sont alors celles portant le numéro 12, 11 et 1.

I.2.6. Déroulement d'un transfert de données.

ATN	DIO	Commentaires
		- Conditions initiales: les 3 périphériques ne sont pas adressés
		- première phase : configuration du multiplexeur. Cette phase permet de choisir l'appareil de mesure avec lequel l'acquisition va se dérouler.
vrai	Adresse MPX	
faux	Donnée de configuration	- le multiplexeur est adressé comme listener; le contrôleur prend l'état de talker. - le multiplexeur reçoit du contrôleur une donnée de configuration qui détermine quel instrument de mesure va être sélectionné
vrai	UNL	- le contrôleur désactive le multiplexeur.
		- deuxième phase: acquisition des données de l'appareil de mesure. Cette phase permet au contrôleur d'acquérir les données en provenance de l'appareil de mesure, via le A/D.

ATN	DIO	Commentaires
vrai	Adresse A/D	
		- le convertisseur A/D est adressé comme talker; le contrôleur prend l'état de listener.
faux		
	Donnée 1	- au moment où ATN devient faux, les données sont transférées vers le contrôleur: A/D est talker et le micro-ordinateur contrôleur est listener
	⋮	
	Donnée n	
vrai	UNT	- lorsque le contrôleur a reçu le nombre de données voulu, il fait taire le A/D en envoyant la commande UNT
		- troisième phase: envoi des données à l'instrument de visualisation. Les données envoyées lors de la deuxième phase par le A/D sont transférées vers l'organe de visualisation, via le convertisseur D/A.
vrai	Adresse D/A	
		- le convertisseur D/A est adressé comme listener
faux		
	Donnée 1	- transfert des données vers le D/A.
	⋮	
	Donnée n.	
vrai	UNL	- Lorsque le contrôleur a terminé le transfert, il désactive le D/A par la commande UNL: la configuration est ainsi remise dans sa situation initiale.

I.2.7. Liaison micro-ordinateur et DEC-2Ø

La liaison entre le micro-ordinateur et le DEC-2Ø est assurée par une ligne téléphonique permanente, installée au Département de Chimie (cfr figure I.11). Cette ligne est raccordée sur le port A du circuit d'entrée/ sortie série SIO. Il s'agit d'une ligne dont la vitesse de transmission est de 2400 bauds.

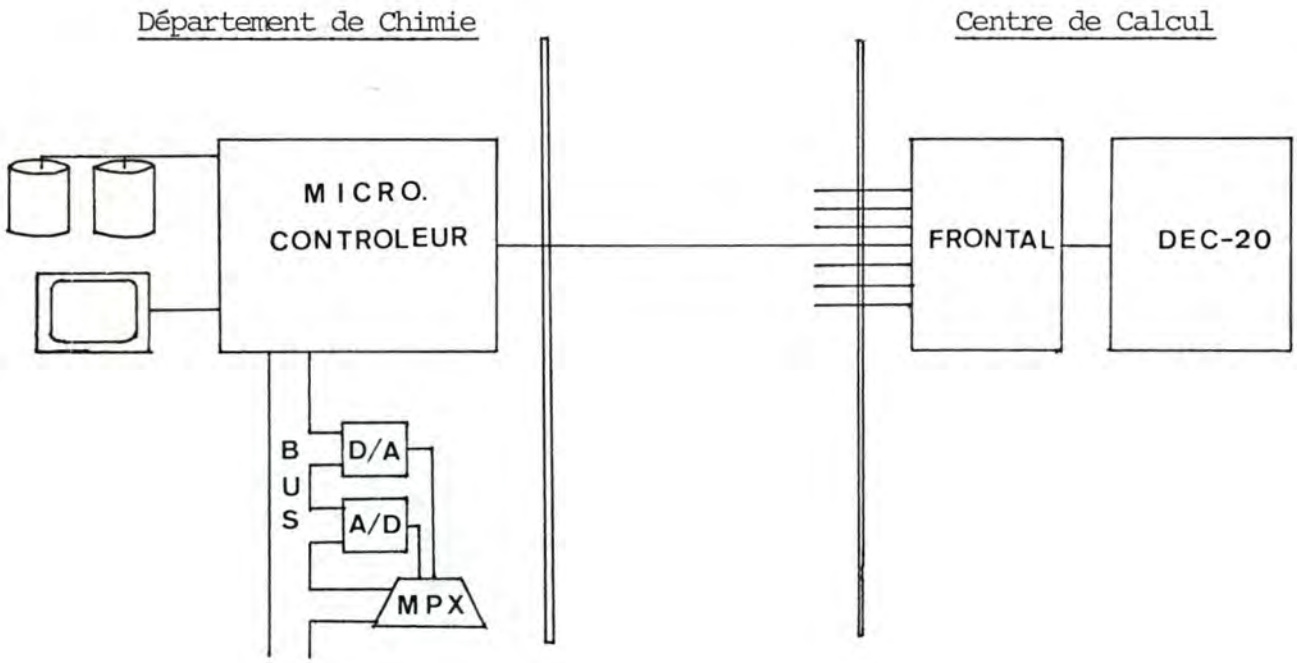


Figure I.11: configuration générale du matériel.

I.2.8. Définitions

Nous avons regroupé ici un ensemble de définitions de termes utilisés dans le texte du paragraphe consacré à la description du matériel.
Les mots suivis d'une astérisque sont définis tels que dans la norme IEEE-488.

1. Adresse: information transférée sur le bus de données entre le contrôleur et l'ensemble des périphériques connectés au bus, lorsque la ligne ATN est au niveau logique vrai: les valeursASCII des adresses vont de 32 à 62 (LAG) et de 64 à 94 (TAG).

2. Appareil: instrument de mesure ou de visualisation traitant des signaux de nature analogique.

3. Bus (*): ligne ou ensemble de lignes, utilisé par un système d'interfaçage, qui permet la connexion de plusieurs périphériques et par lequel des informations peuvent être transférées.

4. Bus bidirectionnel (*): Bus utilisé par des périphériques pour des transfert d'informations en lecture et en écriture.

5. Commande: information transférée sur le bus de données entre le contrôleur et soit l'ensemble des périphériques (commandes universelles), soit les périphériques adressés en mode listener (commandes adressées).

6. Contrôleur: dispositif capable de gérer la ligne ATN, d'envoyer des commandes sur le bus et d'adresser les périphériques comme talker et listener.

7. Données: informations échangées entre un talker et un ou plusieurs listeners, c'est-à-dire lorsque ATN est faux.

8. Emetteur: dispositif dont la fonction est d'envoyer des informations sur le bus de données, que la ligne ATN soit vraie ou fausse.

9. Informations: caractère transféré sur les lignes de données: une information est donc soit une adresse ou une commande (ATN vrai), soit une donnée (ATN faux).

10. Interface (*): frontière commune entre deux systèmes ou entre les parties d'un système, à travers laquelle l'information peut transiter d'un système à l'autre.

11. Listener (*): périphérique qui peut être adressé de façon à pouvoir recevoir des informations en provenance d'un autre périphérique connecté au système d'interfaçage.

12. Périphérique: tout dispositif connecté au bus, à l'exception du contrôleur.

13. PIO: abréviation de "Parallel Input Output".
organe d'entrée/sortie parallèle. Le PIO Zilog comporte 2 ports de 8 bits parallèle chacun (cfr.I.2.1.2.).

14. Port: organe d'entrée/sortie consistant en un ensemble de registres (registre d'entrée, de sortie, de contrôle), ayant chacun leur propre adresse, et permettant à un processeur d'effectuer des entrées/sorties avec des organes périphériques.

15. Récepteur: dispositif dont la fonction est de recevoir des informations sur le bus de données, que la ligne ATN soit vraie ou fausse.

16. SIO: abréviation de "Serial Input Output".
organe d'entrée/sortie série.
Le SIO de Zilog comporte 2 ports séries programmables (cfr. I.2.1.3.).

17. Système d'interfaçage (*): ensemble des éléments mécaniques, électriques et fonctionnels d'une interface, nécessaire à la communication entre divers périphériques.

18. Talker (*): périphérique qui peut être adressé de façon à pouvoir envoyer des informations à un autre périphérique connecté au système d'interfaçage.

1.3. DESCRIPTION DES LOGICIELS DE BASE

Ce paragraphe présente succinctement les logiciels que nous avons utilisé pour la réalisation de ce mémoire.

Système d'exploitation: CP/M version 2.2.

Il s'agit d'un système extrêmement répandu, qui tend à devenir un standard pour les machines à architecture 8 bits.
Outre les commandes de gestion de fichiers que comporte CP/M, celui-ci est doté d'un éditeur de textes (ED), d'un assembleur (ASM), utilisant les mnémoniques du microprocesseur Intel 8080, ainsi qu'un système de mise au point DDT de programmes écrits en langage machine.

Editeur de texte: MINCE

Il s'agit du même éditeur que celui tournant sous UNIX. Sa particularité est d'avoir un grand nombre de commandes identiques à celles de EMACS.
Nous avons préféré cet éditeur à ED de CP/M, ce dernier étant peu performant et difficile d'utilisation.

Assembleur:

L'assembleur utilisé est M 80 diffusé par MICROSOFT: il s'agit d'un assembleur relativement complet puisqu'il possède de nombreuses pseudo-instructions, des instructions d'assemblage conditionnel. Il permet de travailler soit avec les mnémoniques du microprocesseur Intel 8080, soit avec ceux du Zilog Z-80. Il est également possible de définir des macro-instructions.
Les routines d'acquisitions de données, ainsi que certaines routines du programme de connexion avec le DEC-2Ø, sont écrites en assembleur Z-8Ø, ceci pour plusieurs raisons: une des principales est le fait que les langages de haut niveau, tels que PASCAL, BASIC ou FORTRAN, ne permettent pas d'accéder aux ports d'entrée/sortie du système.
Lelangage "C" constitue cependant une exception. Une autre raison est la rapidité d'exécution, qu'un compilateur ou un interpréteur de langage évolué ne permet pas d'atteindre.

Compilateur:

Deux compilateurs différents ont été utilisés. L'un est un compilateur FORTRAN appelé F8O et diffusé par Microsoft, qui nous a servi à développer des programmes de test d'acquisition de données, utilisant nos routines écrites en assembleur.
L'interaction entre programmes écrits en FORTRAN et les routines assembleur est aisée: le passage des paramètres est réalisé grâce aux registres internes du microprocesseur Z-8O. De plus, il est possible à une routine assembleur de faire appel à des routines appartenant à la librairie FORTRAN (routines d'entrée/sortie, arithmétiques). L'édition des liens se fait grâce à un utilitaire appelé L8Ø.

L'autre compilateur est JRT PASCAL, de Microsoft. C'est à l'aide de ce compilateur qu'a été développé le programme de connexion avec le DEC-2Ø.
Le logiciel JRT comporte, outre le compilateur, un éditeur de liens (LINKER), un utilitaire (CONVERIM) permettant de convertir un fichier objet créé par un autre compilateur, en un fichier compatible avec l'éditeur de liens de JRT.

CHAPITRE II

II.1. SPECIFICATIONS DES ROUTINES D'ECHANGES DE DONNEES

II.1.1. INTRODUCTION

Les échanges de données entre un dispositif expérimental et le système informatique de contrôle (cfr I.2.1.), sont assurés d'une part par un dispositif, constitué par un convertisseur D/A, un convertisseur A/D et un multiplexeur, et d'autre part par un ensemble de routines dont le rôle est de constituer une "interface" entre un programme d'application, rédigé en FORTRAN, et le dispositif expérimental.

Ce chapitre est consacré à la description de ces routines. Après avoir donné un aperçu général des différentes routines, nous expliquerons les principes de base de ces routines, et ensuite nous donnerons une description plus détaillée des routines avec le nombre, le nom et le type des paramètres nécessaires à ces routines.

II.1.2. APERCU GENERAL DES ROUTINES

La description du dispositif d'échange de données, étudié dans le précédent chapitre, montre la présence de:

- un convertisseur D/A qui a pour rôle de transformer les données binaires prélevées sur le bus en un signal analogique, que le multiplexeur se charge de diriger vers un ou plusieurs appareils.

- un convertisseur A/D qui permet de transformer une tension venant d'un appareil en une valeur binaire qui sera ensuite chargée sur le bus, à destination de l'ordinateur de contrôle.

- un multiplexeur dont le rôle est de sélectionner parmi l'ensemble des appareils qui y sont reliés, celui ou ceux avec lesquels les échanges de données vont se dérouler.

Ces périphériques réalisent respectivement les fonctions suivantes:

- envoi de données, venant du système informatique, vers un ou plusieurs appareils

- envoi de données, venant d'un appareil, et destinées au système informatique

- sélection d'un ou plusieurs appareils avec lesquels le système informatique va réaliser des échanges de données.

A chacune de ces fonctions est associée une routine:

- la routine DATSND (Data Send) réalise la fonction d'envoi de données du système informatique vers le convertisseur D/A

- la routine DATRCV (Data Receive) assure la réception par le système informatique de données émises par le convertisseur A/D

- la routine MXSND (Multiplexer Send) permet la sélection d'un ou plusieurs appareils avec lesquels les échanges de données vont se dérouler.

Une routine d'initialisation a également été élaborée: elle s'appelle GPBINI (General Purpose Bus Initialisation) et permet de configurer le port B du circuit PIO en mode contrôle (cfr I.2.4.2.)

et d'envoyer les commandes UNL et UNT de façon à désactiver les trois périphériques du bus (D/A, A/D et multiplexeur).

Une dernière routine, appelée AFFMES (Affichage Message), est chargée d'afficher, à l'écran du système informatique de contrôle, des messages d'erreurs signalant à l'utilisateur qu'un problème s'est produit sur le bus.

II.1.3. PRINCIPES DE BASE DES ROUTINES

Les transferts d'informations se déroulent sur le bus, conformément au protocole défini par la norme IEEE-488 (cfr. figure I.6.). Il peut cependant arriver que des erreurs se produisent au cours d'un transfert. Mis à part une erreur consistant en une modification de l'information transférée sur le bus de données, dont nous ne tiendrons pas compte dans la suite, on peut mettre en évidence deux autres types d'erreur:

- erreur d'adressage, qui consiste dans le fait que le contrôleur tente d'effectuer un transfert avec un périphérique qui n'a pas été adressé, parce que le contrôleur a envoyé au cours de la phase d'adressage, une adresse qui ne correspond à aucune de celles des périphériques de la configuration.

- erreur due à une déficience matérielle, que l'on rencontre lorsque le périphérique adressé par le contrôleur n'est pas en état d'assurer sa tâche (par exemple parce qu'il n'est plus alimenté), ou lorsque une ou plusieurs lignes du bus de synchronisation sont hors d'usage, rendant les transferts impossibles.

Dans ces deux situations, l'échange de données ne peut se dérouler normalement. La routine qui, dans un tel cas, tenterait un transfert sur le bus, se trouverait bloquée sur une opération d'entrée/sortie impossible,

avec comme conséquence un blocage du programme d'application qui a fait appel à cette routine.

II.1.3.1. Erreur due à une déficience matérielle

La solution que nous avons adoptée pour résoudre ce type d'erreur, et qui est utilisée sur certains micro-ordinateurs munis d'interface IEEE-488 consiste à travailler par "time-out".
En effet, le protocole de transfert comporte des points d'attente qui sont pour la phase émetteur:
"Le récepteur est-il prêt pour recevoir une information?" ($\overline{NRFD}$)

"Le récepteur a-t-il accepté l'information?" ($\overline{NDAC}$)

et pour la phase récepteur:
"L'émetteur a-t-il validé l'information?" (DAV)

"L'émetteur a-t-il invalidé l'information?" ($\overline{DAV}$).

Pour la phase émetteur comme pour la phase récepteur, on attend que la condition soit vérifiée. Dans le cas où le (les) périphérique(s) impliqué(s) dans la communication se trouve(nt) dans l'incapacité de modifier l'état des lignes de synchronisation pour poursuivre le transfert, les points d'attente deviennent des points de blocage.

Par conséquent, pour éviter le blocage du contrôleur, il lui suffit que, passé un certain délai d'attente, il arrête le transfert et positionne un code erreur qui permettra à l'utilisateur d'être averti de l'arrêt de l'opération de transfert.
Le délai d'attente a été, dans notre application, empiriquement fixé à 100 millisecondes.

II.1.3.2. Erreur d'adressage

En ce qui concerne l'erreur d'adressage, nous allons distinguer deux cas: 1) erreur d'adressage sur un périphérique talker
2) erreur d'adressage sur un périphérique listener.

Dans le premier cas, lorsque le contrôleur envoie sur le bus une adresse de talker, par convention,tout autre talker présent se désadresse automatiquement. Ceci est une conséquence du fait que à tout moment, il ne peut y avoir qu'un seul talker en activité.
Par conséquent, si l'on adresse un talker inexistant, tout autre talker est désactivé.
Dans notre application le transfert de données se faisant toujours entre le contrôleur et un (ou plusieurs) périphérique, nous pouvons alors éviter le blocage du contrôleur par la méthode décrite précédemment (cfr II.1.3.1.), le contrôleur va, au cours de l'échange de la première donnée, attendre que le talker (ici le A/D) valide la donnée; comme cette validation ne pourra se produire, puisqu'il n'y a pas de talker adressé, l'opération d'échange sera suspendue après un certain délai. Le contrôle est alors rendu au programme principal.

Dans le deuxième cas, pour éviter le blocage du contrôleur par adressage d'un périphérique listener inexistant, on procède comme suit: étant donné que tout listener doit, lors de l'initialisation du protocole de transfert, placer les lignes NRFD et NDAC au niveau logique vrai, il suffit au contrôleur de contrôler la validité de ces signaux: si NRFD et NDAC sont vrais, l'échange peut avoir lieu; dans les autres cas, il n'y a pas de listener adressé et on arrête l'opération d'échange.

II.1.4. Spécifications des routines

Nous allons présenter chacune des routines, en spécifiant pour chacune d'elles, leur nom, les paramètres en entrée et en sortie, ainsi que la fonction de la routine.

Les routines GPBINI, MXSND, DATSND et DATRCV réalisent des transferts d'informations sur le bus; c'est pourquoi elles ont toutes un paramètre d'erreur, appelé ERROR, qui est un nombre entier dont la valeur permet d'identifier l'erreur. Les valeurs possibles de ERROR sont:

0 si il n'y a pas eu d'erreur pendant le transfert
1 si un des récepteurs n'est pas prêt pour recevoir une information
2 si l'information n'est pas acceptée par un des récepteurs
3 si la donnée envoyée par le talker (A/D) n'est pas validée par celui-ci
4 si la donnée envoyée par le talker n'est pas invalidée par celui-ci
5 si le périphérique adressé n'existe pas.

II.1.4.1. Routine GPBINI (ERROR): initialisation du bus

paramètre en entrée: - -
paramètre en sortie: ERROR nombre entier déclaré en INTEGER*1 (codé sur un octet).
ERROR est un paramètre d'erreur, permettant de diagnostiquer le déroulement d'un transfert sur le bus.
fonction: La routine GPBINI a deux fonctions: la première est de configurer le port B du PIO en mode contrôle (cfr.I.2.4.).
La deuxième fonction est d'envoyer sur le bus les commandes UNL et UNT.
En cas d'erreur, ERROR est positionné à une valeur non nulle.
Les valeurs possibles sont 0 (pas d'erreur), 1, 2 ou 5.

Il est à remarquer que, de façon à pouvoir envoyer une donnée sur le bus, la routine doit configurer les transceivers du port A en mode sortie, ainsi que le port A lui-même.

II.1.4.2. Routine MXSND (LSTNR, BYTE, ERROR): configuration du multiplexeur.

paramètre en entrée: LSTNR: entier déclaré en INTEGER*1.
LSTNR représente l'adresse logique du multiplexeur sur le bus.
BYTE: entier déclaré en INTEGER*1.
Byte représente une donnée que l'on envoye au multiplexeur de façon à le configurer.

paramètre en sortie: ERROR: nombre entier déclaré en INTEGER*1.
ERROR est un paramètre d'erreur permettant de diagnostiquer le déroulement d'un transfert sur le bus.

fonction: la routine MXSND permet de programmer le multiplexeur, dont l'adresse est LSTNR, en lui envoyant une donnée BYTE, de façon à choisir une ou plusieurs des bornes d'entrées/sorties du multiplexeur (cfr.I.2.5.5.).
Le paramètre ERROR est positionné à une valeur non nulle en cas d'erreur dans la phase d'adressage (envoi de l'adresse LSTNR) ou dans la phase d'échange (envoi de BYTE).
Les valeurs possibles pour ERROR sont 0 (pas d'erreur), 1, 2 ou 5.

II.1.4.3. Routine DATSND (LSTNR, ARRAY, LENGTH, ERROR): envoi de données.

paramètres en entrée: LSTNR: entier déclaré en INTEGER*1.
LSTNR représente l'adresse logique du convertisseur D/A.
ARRAY: tableau de nombres réels déclaré en REAL (codés sur 4 octets).

ARRAY est un tableau de données que l'on veut envoyer au convertisseur D/A.
LENGTH: entier déclaré en INTEGER*2 (codé sur 2 octets). LENGTH représente le nombre d'éléments du tableau ARRAY que l'on désire envoyer au convertisseur.

paramètres en sortie: ERROR: entier déclaré en INTEGER*1.
ERROR représente un paramètre d'erreur permettant de diagnostiquer le déroulement d'un transfert sur le bus.
LENGTH: entier déclaré en INTEGER*2.
Après exécution de la routine, LENGTH donne le nombre de données du tableau ARRAY qui n'ont pas été envoyées.

fonction: La routine DATSND permet d'envoyer au convertisseur D/A, d'adresse LSTNR, un bloc de données ARRAY de longueur LENGTH.
Le paramètre LENGTH est décrémenté à chaque fois qu'un élément du tableau est envoyé. Si tous les éléments du tableau ont pu être envoyés, LENGTH est nul: il ne reste plus d'élément à envoyer.
Si par contre, une erreur se produit, la routine DATSND est interrompue et LENGTH donne alors le nombre d'éléments non envoyés au convertisseur.
En cas d'erreur, ERROR est mis à une valeur non nulle, et les valeurs possibles sont 0 (pas d'erreur), 1, 2, 5.

Les éléments de ARRAY sont envoyés séquentiellement, à partir de l'indice 1 vers le convertisseur. Le signal analogique résultant de la conversion est dirigé vers l'(les) appareil(s) choisi(s) grâce à la programmation du multiplexeur.

Le convertisseur D/A est caractérisé par une fonction de conversion et par le format des données binaires qu'il traite.

L'utilisateur se voit donc contraint de transformer les éléments du tableau de données suivant la transformation inverse de celle que leD/A effectue.
Ce travail se révélant fastidieux, il nous a semblé intéressant que la routine DATSND réalise elle-même ces opérations de formatage et de conversion. Ainsi, le tableau ARRAY contient simplement des nombres réels représentant les tensions que l'utilisateur veut envoyer au convertisseur, et, par là, aux appareils choisis par la configuration du multiplexeur (voir figure II.1).

Les valeurs des éléments de ARRAY doivent se situer dans l'intervalle [-10, +10] puisque le convertisseur est cablé suivant cette échelle de travail (cfr. I.2.5.4.).

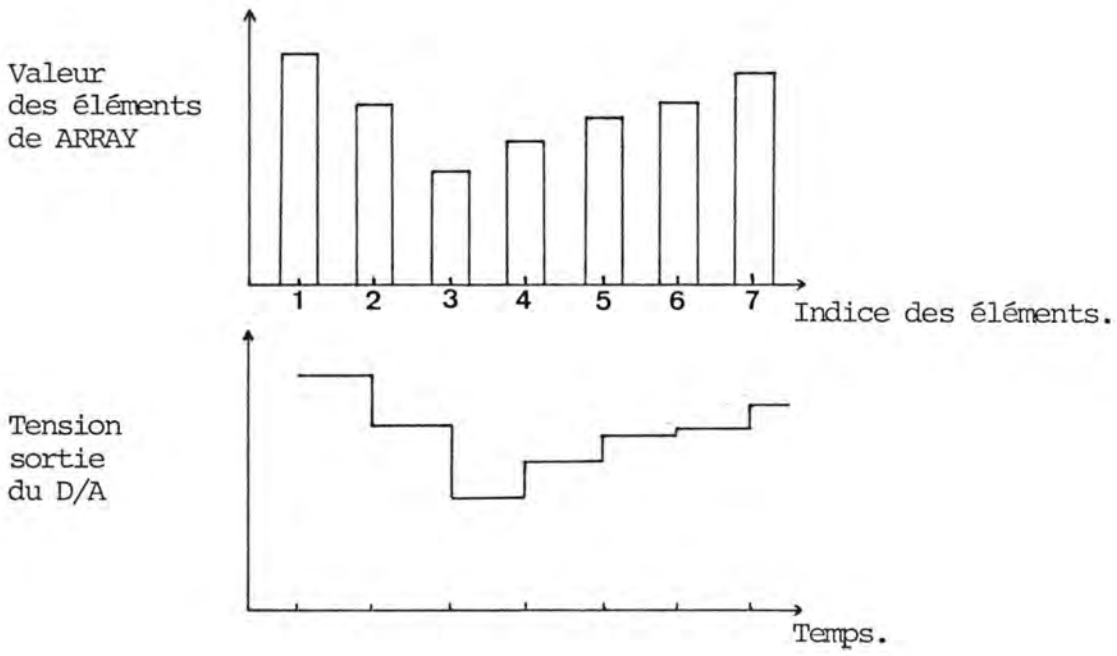


Figure II.1.: échange avec D/A.

II.1.4.4. routine DATRCV (TALKER, ARRAY, LENGTH, ERROR): réception de données.

paramètres en entrée: TALKER: entier déclaré en INTEGER*1
TALKER représente l'adresse logique du convertisseur A/D.
LENGTH: entier déclaré en INTEGER*2
LENGTH représente le nombre de données que l'on veut recevoir du convertisseur A/D.

paramètres en sortie: ARRAY: tableau de réels déclarés en REAL.
Les éléments de ARRAY contiennent les données venant du convertisseur A/D.
LENGTH: entier déclaré en INTEGER*2.
Après exécution de la procédure, LENGTH donne le nombre de données non envoyées par le convertisseur.
ERROR: entier déclaré en INTEGER*1.
ERROR représente un paramètre d'erreur permettant de diagnostiquer le déroulement d'un transfert sur le bus.

fonction: La routine DATRCV permet de recevoir, du convertisseur A/D d'adresse TALKER, un bloc de données ARRAY de longueur LENGTH.
Si toutes les données demandées ont été reçues et stockées dans le tableau, le paramètre LENGTH, après exécution de la routine, est nul. Par contre, si une erreur se produit au cours du transfert, l'exécution de DATRCV s'interrompt, et ERROR est mis à une valeur différente de zéro.
LENGTH donne alors le nombre de données non reçeus. Les valeurs possibles de ERROR sont 3 ou bien 4.(0 si pas d'erreur).
Les données reçues du convertisseur sont stockées séquentiellement dans le tableau à partir de l'indice 1.
Comme pour la routine DATSND, la routine DATRCV se charge de la conversion et du formatage des données prélevées sur le bus. (cfr.I.2.5.3.), avant de les stocker dans le tableau ARRAY: ce dernier contient donc des nombres réels, situés dans l'intervalle [-10, +10] et représentant physiquement des tensions.

II.1.4.5. routine AFFMES (ERROR): affichage des messages d'erreur.

paramètre en entrée: ERROR: entier déclaré en INTEGER*1.
ERROR représente un paramètre d'erreur permettant de diagnostiquer le déroulement d'un transfert sur le bus.

paramètre en sortie: -

fonction: La routine AFFMES permet d'afficher les messages d'erreur correspondant aux différentes valeurs du paramètre ERROR, renvoyé par les routines GPBINI, MXSND, DATSND et DATRCV.

Ces messages sont:

ERROR	Messages
0	** Communication OK **
1	** listener not ready to receive data **
2	** data not accepted by listener **
3	** data received not validated by talker **
4	** data received not unvalidated by talker **
5	** device not present **

II.1.5. PROBLEME DE L'ECHANTILLONNAGE

Grâce à la routine d'acquisition de données DATRCV, il est possible d'effectuer des mesures sur un système physique. On se heurte cependant à un problème typique de ce genre d'expérience, qui est celui de la fréquence d'échantillonnage. On sait, par le théorème de Nyquist qu'il est nécessaire pour reconstituer un signal, de l'échantillonner à une fréquence double de la plus haute fréquence du spectre de ce signal.

Or, la vitesse ou fréquence à laquelle la routine DATRCV va échantillonner un signal va être limitée par différents facteurs.
Le convertisseur A/D a besoin d'un certain délai pour convertir une tension en une donnée binaire. A la durée de conversion vient s'ajouter la durée de l'échange des 2 bytes représentant la donnée, entre le convertisseur et le micro.
Enfin, il faut encore tenir compte du temps nécessaire au micro pour convertir les 2 octets en un nombre réel et le stocker dans le tableau en mémoire centrale. Appelons T l'intervalle de temps séparant le début de la conversion du signal par le A/D et le stockage de la donnée en mémoire; la fréquence d'échantillonnage est donc égale à $1/T$, avec $T = t_1 + 2t_2 + t_3 + t_4$ dans laquelle t_1 représente la durée de conversion, t_2 la durée du transfert d'un octet entre le A/D et le micro, t_3 le temps de conversion des deux octets par le micro, et t_4 le temps de stockage de la donnée en mémoire.

La fréquence d'échantillonnage est difficile à calculer théoriquement, principalement à cause des facteurs t_2 et t_3.
On peut cependant la mesurer en chronométrant le temps nécessaire pour acquérir un tableau de, par exemple, 1000 mesures: ce temps est de 10 secondes environ, ce qui donne donc une fréquence d'échantillonnage de 100 Hertzs.
La fréquence maximale du spectre du signal est ainsi de 50 Hertzs .

Suite à des problèmes techniques de dernière minute, il ne nous a pas été possible d'effectuer une mesure de la fréquence d'échantillonnage lorsque t_3 est nul.

Malgré la faiblesse de la fréquence d'échantillonnage, celle-ci s'avère suffisante pour les applications envisagées au Département de Chimie: il s'agit notamment d'expériences de résonance magnétique nucléaire et de spectrométrie de masse.

II.1.6. SPECIFICATIONS DES TRAITEMENTS

Pour chaque routine élaborée, nous pouvons procéder à une décomposition de celles-ci en fonctions, dont nous allons donner les spécifications.

II.1.6.1. routine GPBINI

La routine GPBINI se compose de quatre fonctions, qui consistent à:

- configurer le port B du PIO en mode contrôle.
- configurer le port A et ses buffers en mode sortie.
- envoyer la commande UNL.
- envoyer la commande UNT.

II.1.6.2. routine MXSND

La routine MXSND comporte trois fonctions:

- programmer le port A et ses buffers en mode sortie.
- adressage du périphérique multiplexeur.
- configuration du multiplexeur: cette fonction consiste à envoyer une donnée pour sélectionner une ou plusieurs des bornes d'entrée/sortie du multiplexeur.

II.1.6.3. routine DATSND

La routine DATSND consiste en cinq fonctions:

- programmer le port A et ses buffers en mode sortie.
- adressage du convertisseur D/A.
- conversion d'un élément du tableau à envoyer: chaque élément de ce tableau est converti suivant la formule du paragraphe I.2.5.4.
- formatage de l'élément converti, en deux données de 8 bits (voir paragraphe I.2.5.4.).

- envoi d'une donnée sur le bus: chacune de ces deux données est envoyée sur le bus.

II.1.6.4. routine DATRCV

La routine DATRCV comporte six fonctions:

- programmer le port A et ses buffers en mode sortie.
- adressage du convertisseur A/D.
- programmation du port A et ses buffers en entrée.
- réception d'une donnée envoyée sur le bus par le convertisseur A/D.
- formatage de deux données lues successivement sur le bus en une seule donnée (voir paragraphe I.2.5.3.).
- conversion de cette donnée en une valeur réelle et stockage de cette valeur dans un tableau (cfr. I.2.5.3.).

II.1.6.5. routine AFFMES

La routine AFFMES ne comporte qu'une seule fonction qui consiste à imprimer à l'écran du terminal un message de diagnostic d'un transfert.

II.2. SPECIFICATIONS DU PROGRAMME DE CONNEXION AVEC LE DEC-20

II.2.1. INTRODUCTION

Le programme de liaison avec le DEC-2Ø est destiné à remplir deux fonctions: la première consiste en des opérations de transfert de fichiers entre le DEC-2Ø et le micro-ordinateur, décrit au chapitre précédent. La deuxième fonction est l'utilisation de ce micro-ordinateur comme terminal du DEC-2Ø.

Les opérations de transferts de fichiers portent sur des fichiers de données, codées en ASCII, résultant de mesures expérimentales, réalisées à partir des routines d'acquisition de données.
Une fois que celles-ci sont transférées sur le DEC-2Ø, elles peuvent être traitées par des programmes de traitement des signaux.
Une fois le traitement terminé, les résultats sont retransférés vers le micro-ordinateur, pour être analysés par les utilisateurs.
Il est possible de transférer des fichiers de texte - source de programmes, édités sur le micro, et dont l'exécution doit se dérouler sur le DEC.
Ceci présente l'avantage de délivrer le DEC des travaux de dactylographie.
Les fichiers ainsi transférés peuvent également être imprimés par les imprimantes rapides du Centre de Calcul.

L'utilisation du BigBoard en terminal du DEC permet d'assurer différents services. Par exemple, on peut s'assurer que le transfert d'un fichier vers le DEC s'est correctement déroulé ou encore qu'un fichier que l'on veut amener du DEC vers le micro se trouve bien dans le directoire auquel on est connecté.
Un autre avantage est que l'on peut, grâce à l'interface graphique du Bigboard, exécuter sur celui-ci des logiciels graphiques du DEC-2Ø.

II.2.2. PRINCIPES DE LA CONNEXION

II.2.2.1. Introduction

La liaison entre le micro-ordinateur et le DEC-2Ø est assurée, comme nous l'avons vu dans le précédent chapitre, par une ligne téléphonique permanente installée entre le Centre de Calcul et le Département de Chimie.

Il nous faut cependant établir une procédure de dialogue entre ces deux machines. C'est ce à quoi nous allons nous intéresser dans ce paragraphe.

En général, une procédure peut se décomposer en cinq phases:

- phase d'établissement de la liaison, qui consiste, dans notre cas, à savoir comment appeler le DEC-2Ø pour lui signaler que l'on veut dialoguer avec lui;
- phase d'initialisation de la liaison, qui consiste à envoyer au DEC soit une invitation à émettre, soit une invitation à recevoir;
- phase de transfert;
- phase de terminaison de la liaison, qui consiste à clore le transfert des informations entre les deux machines;
- phase de libération, qui consiste à interrompre la communication entre les machines et à libérer la ligne qui les relie.

Nous allons passer en revue chacune de ces phases, en montrant comment elles ont été implémentées, et en essayant de justifier nos choix.

II.2.2.2. Phase d'établissement de la liaison

Cette phase consiste donc à signaler au DEC-2Ø que l'on désire entrer en communication avec lui.

Pour ce faire, nous devons:

a) - initialiser les registres de contrôle du port A du circuit SIO; c'est en effet à ce port qu'est raccordé la ligne du DEC-2Ø.
Les registres sont initialisés de façon à répondre aux caractéristiques de la transmission, à savoir:
- vitesse de lecture/écriture des caractères sur le port série = 2400 bauds puisque, par défaut, le DEC-2Ø travaille à cette vitesse.
- mode de transmission = asynchrone
- nombre de bit stop = 1.
- nombre de bits par caractère = 8.

b) - nous connecter au directoire de fichiers, sur le DEC, appartenant à l'utilisateur. Ceci se fait en deux temps: il faut, en premier lieu, envoyer un caractère Control/C au DEC de façon à appeler l'interpréteur de commande; ensuite, on exécute la commande LOGIN : elle nécessite deux paramètres: - le nom de l'utilisateur
- le mot de passe.

Il nous a semblé intéressant d'automatiser complètement la procédure de LOGIN: c'est le programme de connexion qui appelle l'interpréteur de commande puis qui envoie la commande LOGIN et les coordonnées de l'utilisateur (nom et mot de passe) au DEC-2Ø.
Le programme s'assure que le LOGIN a été correctement effectué, de façon à éviter des problèmes lors des opérations de transfert de fichiers.
Le fait que ce soit le programme et non l'utilisateur qui effectue le LOGIN permet une plus grande sécurité de travail, et assure que la phase d'établissement de la liaison soit menée à bien en vue d'effectuer les opérations de transfert ou le travail en terminal.

II.2.2.3. Phase d'initialisation de la liaison

Cette phase consiste à envoyer au DEC-2Ø soit une invitation à émettre, soit une invitation à recevoir, selon que le DEC est soit émetteur, soit récepteur d'informations.

Cette invitation consiste en une commande qui va faire en sorte que le DEC lance l'exécution d'un programme de réception ou d'émission d'informations. Ce programme peut être soit un programme utilisateur, soit un programme système.
Nous avons tiré profit de l'existence de tels programmes-systèmes sur le DEC, principalement pour des raisons de manque de temps, mais également pour des raisons de facilité.
L'invitation à émettre est donnée par la commande TYPE suivie du nom du fichier du DEC à transférer: l'effet de cette commande est de transférer le fichier spécifié sur la ligne utilisée.
L'invitation à recevoir est la commande COPY suivie de deux noms de fichiers: COPY ⟨nom fichier source⟩ ⟨nom fichier destination⟩ .
L'effet de la commande est de copier le fichier ⟨nom fichier source⟩ sur le fichier ⟨nom fichier destination⟩ . Lorsque l'on attribue le nom "TTY:"= au nom de fichier source, alors l'effet de la commande est de lire tous les caractères envoyés sur la ligne assignée à l'utilisateur, et de les sauvegarder dans le fichier ⟨nom fichier destination⟩ .
La fin du fichier est signalée au DEC par l'envoi d'un caractère Control/Z.

Lors de l'initialisation de la liaison et avant d'envoyer une invitation à émettre, il est indispensable de procéder à l'envoi de deux commandes au DEC-2Ø: celles-ci sont:

- TERMINAL PAUSE END-OF-PAGE: elle signale au DEC que lors de l'exécution d'un TYPE d'un fichier, lorsque 24 lignes (d'au plus 80 caractères) sont envoyées (ce qui correspond à une page d'écran d'un terminal), le DEC attend un caractère Control/Q pour passer aux 24 lignes suivantes. Nous étudierons cette commande TYPE plus en détail dans la phase de transfert.
- TERMINAL PAGE x: l'effet de cette commande est de modifier le nombre de lignes d'écran du terminal : l'écran comporte alors x lignes au lieu de 24 qui est la valeur par défaut.
  L'utilité de cette commande apparaîtra plus clairement dans la phase de transfert.

II.2.2.4. Phase de transfert d'informations

II.2.2.4.1. Principes des opérations de transfert.

Les opérations de transfert de fichiers sont basées sur un contrôle de flux élémentaire reposant sur le principe du "stop and wait": nous pensons que c'est le plus facile à implémenter, compte tenu des délais impartis pour réaliser l'ensemble de l'application (cfr chapitre V: problèmes rencontrés).
Rappelons brièvement en quoi consiste ce protocole: l'émetteur envoie ses données en bloc; lorsqu'un bloc est envoyé, l'émetteur attend l'accusé de réception du récepteur. Lorsque l'émetteur reçoit l'accusé de réception, il peut transférer le bloc suivant, et ainsi de suite jusqu'à la fin du transfert.

Par ailleurs, nous n'avons pas jugé indispensable d'implémenter un contrôle d'erreur. Ceci est justifié compte tenu de la distance relativement courte séparant les deux machines et par le fait que les transferts d'information entre le DEC et les terminaux ne sont pas soumis à un contrôle d'erreur.

II.2.2.4.2. Transfert du micro-ordinateur vers le DEC.

Au paragraphe précédent, nous avons annoncé que les opérations de transfert du micro-ordinateur vers le DEC-2Ø se déroulait grâce à la commande système COPY TTY: ⟨nom fichier⟩ qui permet d'accepter les caractères venant d'un terminal (ici le Bigboard) et de les stocker dans un fichier sur le DEC.

Le déroulement du transfert est implémenté de la façon suivante: le Bigboard envoie le fichier caractère par caractère; après chaque caractère envoyé, il attend l'écho de ce caractère, avant de procéder à l'envoi du caractère suivant.

Si à première vue, ce protocole peut paraître peu performant au point de vue vitesse de transfert, nous estimons qu'il est suffisant pour l'application envisagée, compte tenu du fait que le DEC-2Ø est une machine travaillant en time-sharing et étant la plupart du temps extrêmement "chargée".

II.2.2.4.3. Transfert du DEC vers le micro-ordinateur.

Le transfert est initialisé par trois commandes (cfr II.2.2.3.)

TERMINAL PAUSE END-OF-PAGE
TERMINAL PAGE 12
TYPE ‹non fichier› (invitation à émettre).

Le contrôle de flux est basé sur le principe du "Stop and Wait". Les blocs de données envoyées par le DEC comportent 12 lignes d'au plus 80 caractères: nous verrons au paragraphe II.2.3. pourquoi nous avons travaillé avec des blocs de 12 lignes.

Le DEC, après transfert d'un bloc, attend le caractère d'accusé de réception pour envoyer le bloc suivant : ce caractère est le control/Q. Le Bigboard envoie ce caractère après avoir traité le bloc de données reçu. Les caractères qui composent le bloc sont stockés dans un buffer de 1024 caractères; quand le buffer est plein, le Bigboard procède à une écriture sur disque.
Les blocs de caractères envoyés par la commande TYPE sont encadrés par divers caractères de contrôle.
Chaque bloc est précédé par un caractère "Cursor Home" qui, lorsque l'on travaille sur un terminal "classique" positionne le curseur dans le coin supérieur gauche de l'écran; le caractère suivant est le "Clear Screen" qui efface complètement l'écran. Chaque ligne que comporte le bloc est terminée par les caractères "Carriage Return" et "Line Feed", à l'exception de la dernière ligne qui est terminée par un "Carriage Return" seul.
Le dernier caractère du bloc est un caractère "Bell".

II.2.2.5. Phase de terminaison de la liaison

II.2.2.5.1. Terminaison d'un transfert du micro-ordinateur vers le DEC.

La phase de terminaison consiste à signaler au DEC-2Ø que le transfert du fichier est terminé parce qu'on est arrivé à la fin du fichier ou simplement parce que l'utilisateur a décidé d'avorter le transfert.
Il s'agit donc d'interrompre la commande COPY: ceci se fait en envoyant un caractère control/Z au DEC, ce qui a pour effet de clore le fichier sur le DEC puis d'appeler l'interpréteur de commande.

II.2.2.5.2. Terminaison d'un transfert du DEC vers le micro-ordinateur.

La fin du transfert est signalée par le retour du DEC, après exécution de la commande TYPE, à l'interpréteur de commande.
Il est possible d'avorter un transfert en envoyant deux caractères CTRL/C au DEC, ce qui a pour effet d'arrêter la commande TYPE et ensuite d'appeler l'interpréteur de commandes.

II.2.2.5.3. Terminaison du mode terminal.

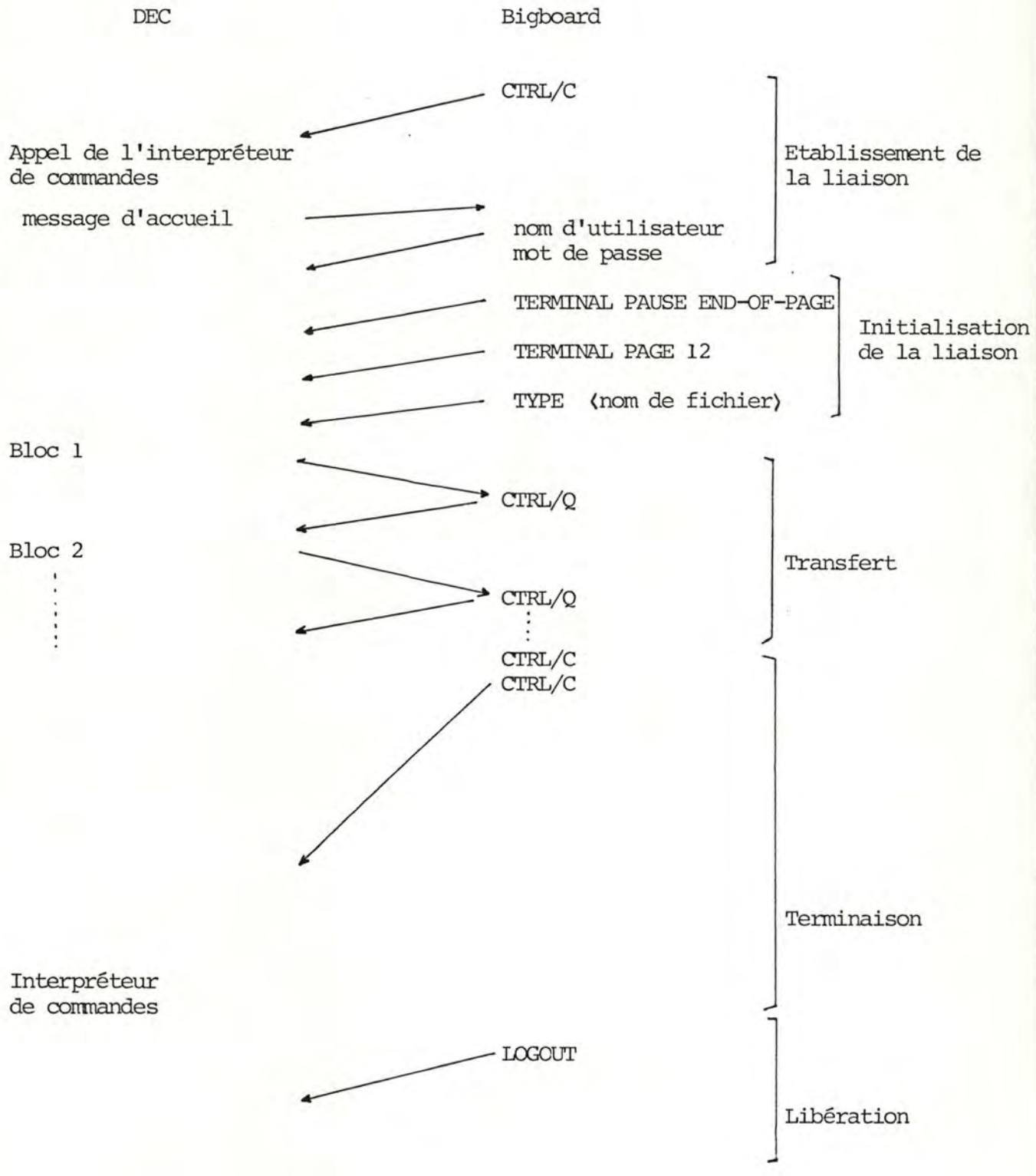
L'utilisateur peut à tout moment interrompre le travail en mode terminal en tapant au clavier le caractère "Backspace".
Le choix de cette touche n'est pas le fait du hasard: en effet, le caractère "Backspace" n'est pas reconnu par l'interpréteur de commandes du DEC ni par ses utilitaires , tels que, par exemple, les éditeurs de texte.

II.2.2.6. Phase de libération de la liaison

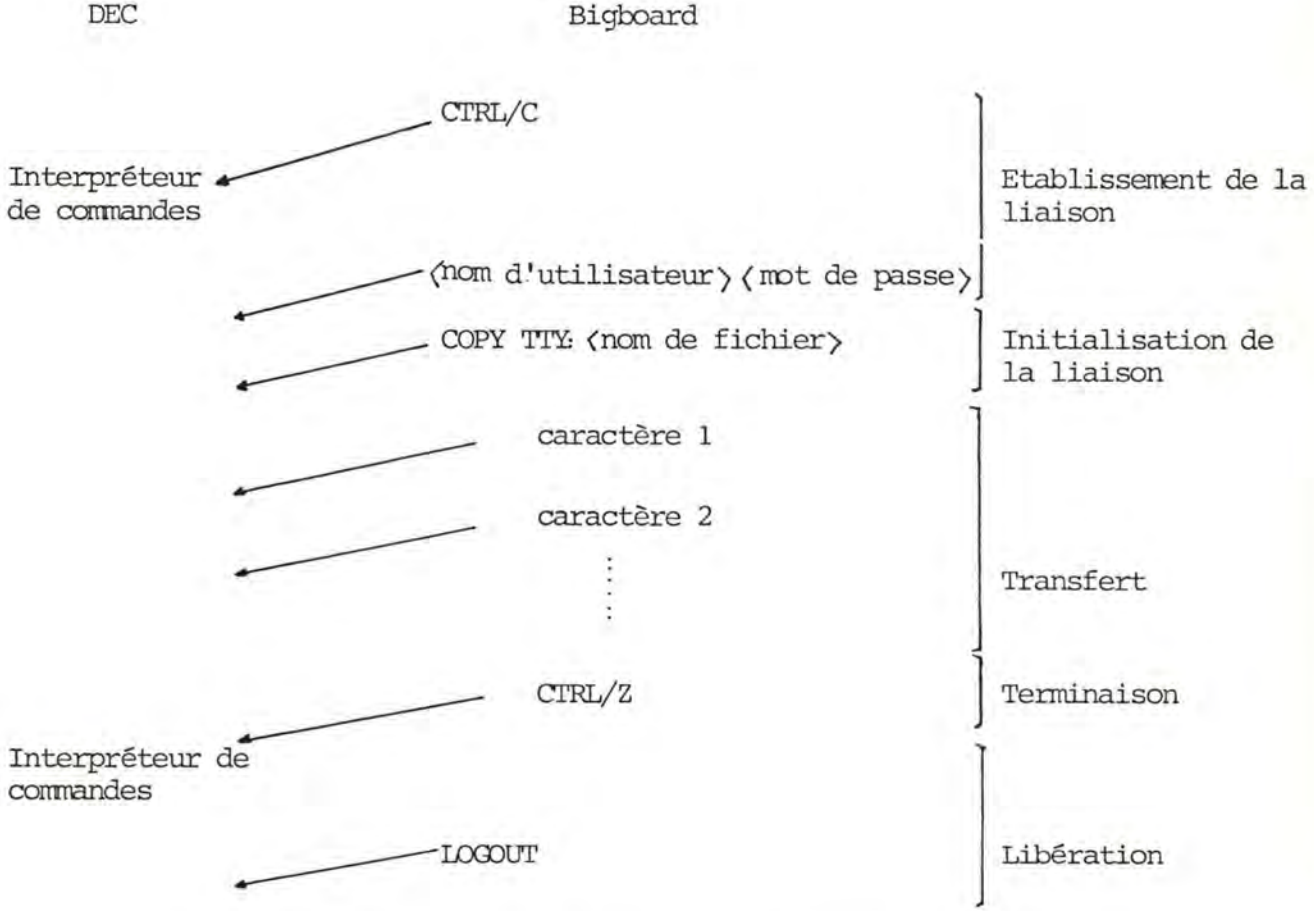
La libération de la liaison s'effectue par l'envoi, au DEC, de la commande LOGOUT qui arrête ainsi la session de travail de l'utilisateur sur le DEC et libère la ligne.

II.2.3. SYNTHESE SUR LES PRINCIPES DES TRANSFERTS

II.2.3.1. Transfert du DEC vers le micro-ordinateur.



II.2.3.2. Transfert du micro-ordinateur vers le DEC.



II.2.4. Description du dialogue et possibilités offertes

Nous allons décrire dans ce paragraphe comment se déroule une session de travail, en présentant toutes les grilles et les messages que l'utilisateur va rencontrer. Nous montrerons également les limites du produit que nous avons développé, et justifierons les choix effectués. Lors du lancement de l'exécution (voir à cet effet le mode d'emploi en annexe), l'écran du micro. s'efface complètement, et une première question s'affiche sur la deuxième ligne de l'écran:

DONNEZ VOTRE NOM D'UTILISATEUR:

On demande ensuite le mot de passe, par une question affichée à la quatrième ligne:

DONNEZ VOTRE MOT DE PASSE:

Une fois le mot de passe introduit, celui-ci est immédiatement effacé.

```
  0 1 2 3 4 5 6 7 8 ...
0 ┌──────────────────────────────────────
1 │
2 │  DONNEZ VOTRE NOM D'UTILISATEUR:
3 │
4 │  DONNEZ VOTRE MOT DE PASSE:
: │
```

Figure II.1.: grille d'écran pour le LOGIN

A partir des coordonnées de l'utilisateur, le programme effectue le LOGIN sur le DEC, tout en s'assurant également, en fonction de la réponse du DEC, que les coordonnées sont correctes. Si ce n'est pas le cas, c'est-à-dire si le nom d'utilisateur et/ou le mot de passe sont invalides, le programme pose à nouveau les 2 questions précédentes.

Une fois que le LOGIN est correctement effectué, une nouvelle grille d'écran apparaît.

ZONE EN VIDEO-INVERSE

```
x x x x x x x x x x x x x CONNEXION BB <=> DEC x x x x x x x x x x x x x
x x MENU  x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x
x TERMINAL - - - - - - - - - - -<1>  | NOM DU FICHIER SUR DEC x x x x x
x                                    |──────────────────────────────── x
x TRANSFERT VERS MICRO- - - - -<2>    |                                 x
x                                    |──────────────────────────────── x
x TRANSFERT VERS DEC - - - - - -<3>   | NOM DU FICHIER SUR MICRO        x
x                                    |──────────────────────────────── x
x FIN- - - - - - - - - - - - - -<4>   |                                 x
x──────────────────────────────────────────────────────────────────────x
x QUELLE OPTION CHOISISSEZ-VOUS ?                                       x
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x
```

Figure II.2.: grille d'écran du menu général

L'écran se compose de deux parties différentes:

- la première, constituée par les 12 lignes supérieures de l'écran et affichée en vidéo inverse, comporte les différentes options du programme (menu), deux cases dans lesquelles seront affichés les noms des fichiers à transférer, et une zone (11e ligne) dans laquelle seront effectuées les diverses questions auxquelles l'utilisateur devra répondre;
- la deuxième partie, constituée par les 12 lignes inférieures et dans laquelle seront affichés les fichiers au fur et à mesure de leur transfert.

Le programme affiche la question:

QUELLE OPTION CHOISISSEZ-VOUS ?

auquel l'utilisateur répond par 1, 2, 3 ou 4 suivi de l'appui sur la touche ‹RETURN›. En fonction de l'option choisie, une flèche "<= =" s'affiche dans le menu, en face du numéro correspondant.
En cas de réponse erronée, la question reste affichée.

Nous allons maintenant décrire chacune des options autorisées par le programme.

II.2.4.1. Option terminal.

Cette option du programme permet, à l'utilisateur, de travailler au clavier du BigBoard comme sur un simple terminal du DEC-2Ø.

Losque l'on passe en mode terminal, l'écran s'efface complètement, et, sur la ligne supérieure de l'écran s'affiche le message suivant:

‹mode TERMINAL = tapez sur ‹BACKSPACE› pour sortir ›

A la ligne suivante apparaît le prompt ? du DEC-2Ø.
Pour sortir de ce mode et revenir au menu principal (cfr figure II.2), l'utilisateur doit simplement appuyer sur la touche ‹ BACKSPACE › , située en haut et à l'extrême droite du clavier.

Le choix de cette touche n'est pas le fait du hasard: en effet, le caractère Backspace n'est pas reconnu par l'interpréteur de commande du DEC, ni par ses utilitaires, tels que, par exemple, les éditeurs de texte.

Lorsque l'on quitte le mode terminal par appui sur⟨Backspace⟩, l'écran décrit précédemment dans la figure II.2. réapparaît, et le choix d'une nouvelle option est offert à l'utilisateur. Lorsque l'utilisateur quitte le mode terminal, la liaison avec le DEC est toujours établie.
Le DEC peut être à ce moment en train d'exécuter une commande que l'utilisateur a lancé, telle que par exemple une compilation.

II.2.4.2. Option de transfert vers le micro-ordinateur.

Cette option permet de transférer un fichier du DEC-2Ø sur une des disquettes du micro-ordinateur.
Le programme demande en premier lieu, le nom du fichier sur le DEC.
Le nom de ce fichier fait l'objet d'une validation: si le fichier n'existe pas, le message " ** FICHIER INEXISTANT ** " est affiché dans la case supérieure droite de l'écran (cfr figure II.3.), puis on demande à nouveau le nom du fichier.

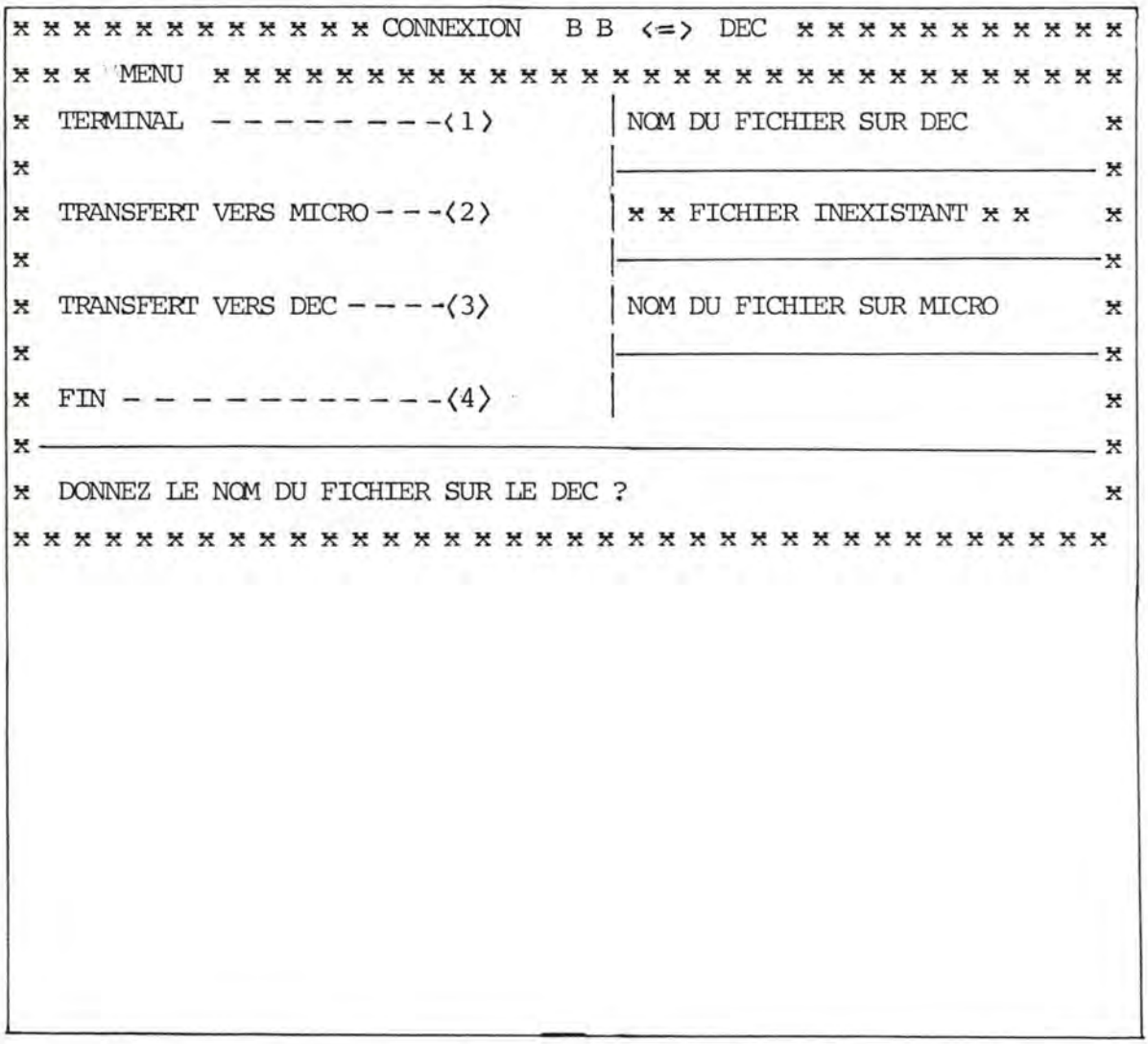


Figure II.3. : option de transfert vers le micro.

Lorsque le nom de fichier est correct, il est affiché dans la rubrique correspondante. Ensuite, le programme demande le nom du fichier sur le micro-ordinateur: la question est affichée à la 11e ligne de l'écran comme dans la figure II.3.
L'utilisateur peut y répondre de deux façons différentes: il peut appuyer directement sur la touche (RETURN) (option par défaut); dans ce cas, le nom du fichier sur le micro sera le même que le nom du fichier sur le DEC. Il peut aussi donner un nom en toute lettre. Si le nom du fichier existe déjà, la précédente version sera détruite et remplacée par la nouvelle copie que l'on va transférer.

Dans les deux cas, le nom du fichier est affiché dans la case "NOM DU FICHIER SUR LE MICRO"; puis le message "TAPEZ<CTRL/Z>POUR ARRETER LE TRANSFERT" s'affiche sur la 11e ligne.

L'opération de transfert peut alors commencer: la partie inférieure de l'écran est réservée pour l'affichage du fichier au fur et à mesure de sa réception par le micro. Lorsque le transfert est terminé, le prompt "ↂ " du DEC apparaît à l'écran juste sous la dernière ligne du fichier transféré: cela signifie donc que le DEC a terminé l'envoi; l'utilisateur doit alors taper CTRL/Z pour clore le transfert.
On est obligé de procéder de cette façon car il n'est pas possible de connaître à quel moment le transfert est terminé: il est évident qu'il ne suffit pas de détecter le caractère "ↂ " pendant le transfert pour en conclure que ce transfert est terminé, puisque le caractère "ↂ " peut faire lui-même partie du fichier.
L'utilisateur a la liberté d'arrêter le transfert quand il le désire en tapant CTRL/Z au clavier.
Une fois le CTRL/Z tapé, le programme demande:

VOULEZ-VOUS EFFECTUER UN NOUVEAU TRANSFERT VERS LE MICRO (O/N) ?

En répondant par "O" (majuscule ou minuscule), on peut entamer un nouveau transfert vers le micro; tout autre caractère ramène l'utilisateur au menu principal (cfr figure II.2.).

Signalons pour terminer que le programme ne s'occupe pas de savoir s'il reste suffisamment de place sur la disquette pour recevoir le fichier transféré: c'est à l'utilisateur a s'assurer avant le transfert qu'il reste suffisamment de place; il peut, pour ce faire, connaitre la taille du fichier qu'il va transférer en choisissant l'option terminal et en consultant le directoire de ses fichiers sur le DEC.

II.2.4.3. Option de transfert vers le DEC.

Cette option permet de transférer un fichier venant du micro vers le DEC-2Ø. Le programme demande tout d'abord le nom du fichier sur le micro. Sous CP/M, le nom du fichier peut être précédé par le nom de l'unité de disquettes (A: ou B:): par défaut, ce nom d'unité est le nom de l'unité sous laquelle on a lancé l'exécution du programme de transfert.
Le programme n'effectue pas de validation sur l'existence du fichier à envoyer au DEC.

Cette limitation tient au fait suivant: nous avions l'intention de développer le programme de connexion en Pascal MT+ car ce langage présentait l'avantage de posséder des primitives d'accès aux fichiers élaborées; il est par exemple possible de détecter une tentative d'ouverture d'un fichier inexistant, ce qui permettait la validation du nom du fichier.

Suite à des problèmes de délai pour l'obtention d'une licence d'exploitation du MT+, nous avons dû nous rabattre sur une version moins performante, le JRT Pascal, qui n'offrait pas ce genre de possibilités.

Lorsque le nom est donné, il est affiché dans la zone d'écran réservée à cet effet.
Ensuite, c'est le nom du fichier sur le DEC qui est demandé (cfr Option 2). Par défaut, ce nom est le même que celui du fichier sur le micro.
Si ce fichier existe déjà sur le DEC, cette version est détruite et est remplacée par la nouvelle venant du micro.
Le transfert peut alors avoir lieu: pour l'interrompre, il suffit, comme pour l'option 2, de taper un CTRL/Z.
Le fichier, pendant son transfert, s'affiche sur la partie inférieure de l'écran. Lorsque cette partie est remplie, elle est effacée, puis l'affichage reprend à partir de la 13e ligne de l'écran.

Lorsque le fichier est transféré, le programme demande:

VOULEZ-VOUS EFFECTUER UN NOUVEAU TRANSFERT VERS LE DEC (O/N) ?

Comme pour l'option 2, il suffit de taper sur tout caractère différent de "O" (majuscule ou minuscule) si l'on veut dire non.

II.2.4.4. Option fin de session.

Cette option a pour but de clore la liaison avec le DEC-2Ø, en effectuant la procédure de LOGOUT. Avant de terminer la liaison, il est indispensable de s'assurer que le directoire, sur le DEC, ne se trouve pas en état de dépassement de capacité, en passant en mode terminal, sans quoi la connexion ne pourrait être rompue normalement.
Une fois le LOGOUT effectué, le programme de connexion s'arrête et repasse la main à CP/M.

II.2.5. Spécifications des traitements.

De façon à spécifier les traitements effectués par le logiciel de connexion, nous allons procéder à une décomposition de notre application en terme de phases et de fonctions.

A partir de ce que nous avons vu dans le paragraphe II.2.2. , nous pouvons envisager la découpe de notre application en trois phases distinctes:
- phase TERMINAL,
- phase TRANSFERT VERS MICRO,
- phase TRANSFERT VERS DEC.

Nous allons décrire chacune de ces phases, et les décomposer en fonctions dont nous expliquerons les spécifications.

II.2.5.1. TERMINAL

Cette phase permet de travailler sur le BigBoard comme sur un terminal "classique" du DEC. Compte tenu des principes de base sur lesquels repose la connexion, nous pouvons décomposer la phase terminal en 5 fonctions:

- initialisation du SIO
- exécution du LOGIN sur le DEC
- initialisation de la ligne
- mode terminal
- fin de session.

- initialisation du SIO.

Cette fonction consiste à programmer le port du circuit SIO sur lequel est raccordé la ligne du DEC de façon à ce qu'il réponde aux caractéristiques de la transmission, à savoir (cfr II.2.2.2.):

- vitesse de lecture/écriture des caractères sur port série: 2400 bauds
- mode de transmission asynchrone
- nombre de bit stop = 1
- nombre de bits par caractère = 8.

- exécution du LOGIN sur le DEC.

Cette fonction consiste, moyennant connaissance des coordonnées de l'utilisateur (nom et mot de passe), à appeler l'interpréteur de commande du DEC-2Ø, puis à envoyer la commande du LOGIN.

Dans le cas où le LOGIN n'est pas effectué (nom et mot de passe erroné), de nouvelles coordonnées sont demandées à l'utilisateur.

- initialisation de la ligne.

Cette fonction permet d'initialiser la ligne entre le DEC et le BigBoard, en envoyant la commande TERMINAL PAUSE END-OF-PAGE au DEC. (cfr fonction de transfert de fichiers du DEC vers le micro).

- mode terminal.

Cette fonction permet à l'utilisateur de travailler, sur le micro-ordinateur, en terminal du DEC. Elle consiste simplement à, alternativement, scruter le clavier pour envoyer le caractère lu vers le DEC, et à lire un caractère envoyé par le DEC pour l'afficher à l'écran du micro-ordinateur.

- fin de session.

Cette fonction consiste à envoyer la commande LOGOUT au DEC-2Ø de façon à arrêter la session de travail de l'utilisateur.

II.2.5.2. TRANSFERT VERS le MICRO-ORDINATEUR

La phase de transfert consiste àn les fonctions suivantes:
- initialisation du SIO
- exécution du LOGIN sur le DEC
- initialisation de la ligne
- acquisition des noms de fichiers
- réception d'un fichier
- fin de session.

Les fonctions d'initialisation du SIO, d'exécution du LOGIN sur le DEC, d'initialisation de la ligne et de fin de session sont identiques à celles de la phase TERMINAL.

- acquisition des noms de fichiers.

Cette fonction consiste à acquérir, à partir du clavier du micro, le nom du fichier sur le DEC ainsi que le nom du fichier sur le micro, vers lequel on va le transférer. Elle vérifie l'existence du fichier sur le DEC. Dans le cas où ce fichier n'existe pas, un message d'erreur est envoyé à l'écran (cfr figure II.3.).

- réception d'un fichier.

Cette fonction consiste à transférer un fichier, envoyé par le DEC, vers le micro; le fichier est celui que l'utilisateur a désigné lors de l'acquisition des noms de fichiers. Pendant le transfert, le fichier est affiché sur les 12 lignes inférieures de l'écran. Le transfert peut être interrompu à tout moment en tapant un CTRL/Z au clavier.

II.2.5.3. TRANSFERT VERS LE DEC

La phase de transfert de fichiers vers le DEC se compose des fonctions suivantes:
- initialisation du SIO
- exécution du LOGIN sur le DEC
- initialisation de la ligne
- acquisition des noms de fichiers
- envoi d'un fichier
- fin de session.

Il nous reste à décrire les fonctions d'acquisition des noms de fichiers et la fonction d'envoi d'un fichier.

- acquisition des noms de fichiers.

Cette fonction sert à acquérir le nom du fichier, appartenant à l'une des disquettes, que l'on désire transférer sur le DEC. Ainsi qu'il a déjà été expliqué, il n'y a pas de validation de l'existence de ce fichier (cfr II.2.4.3.).

- envoi d'un fichier.

Cette fonction consiste à transférer un fichier du BigBoard vers le DEC. Le fichier concerné est celui choisi lors de l'acquisition du nom des fichiers. Pendant le transfert, le fichier est affiché sur les 12 lignes inférieures de l'écran. A tout moment, il est possible d'interrompre le transfert en tapant CTRL/Z au clavier.

CHAPITRE III

CHAPITRE III. DEMARCHE DE CONCEPTION

Ce chapitre présente la démarche que nous avons adoptée pour concevoir l'architecture logicielle des programmes.

Nous donnons également dans la partie annexe de ce travail les spécifications des modules composant l'architecture.

III.1. DEMARCHE DE CONCEPTION DES ROUTINES D'ECHANGES DE DONNEES

Partant des spécifications des routines (paragraphe II.1.4.), il apparait que ces routines comportent deux classes de modules: il s'agit tout d'abord des modules d'entrée/sortie qui ont trait aux échanges de données sur le bus, à la gestion des circuits d'interface (PIO, buffers), les autres modules sont ceux relatifs aux traitements des données qui sont responsables des opérations de conversion de données que nécessitent les convertisseurs.

III.1.1. Modules d'entrée/sortie

Les routines MXSND, DATSND et DATRCV ont une structure identique: elles se composent en effet de:

1. phase d'adressage: MXSND doit adresser le multiplexeur, tandis que DATSND adresse le convertisseur D/A, et DATRCV le convertisseur A/D.

2. phase d'échange: MXSND effectue un échange avec le multiplexeur de façon à le programmer. DATSND (resp. DATRCV) réalise l'échange d'un ensemble de données avec le convertisseur D/A (resp. A/D).

La routine GPBINI se compose uniquement d'une phase d'adressage, au cours de laquelle elle envoie les commandes UNL et UNT.

Ces considérations font ressortir le fait que chaque routine (sauf AFFMES) utilise le protocole de transfert d'informations, défini dans la norme IEEE-488.
Ceci nous a amené à élaborer deux modules de transfert d'informations:

1. module BYTSND (Byte Send): qui assure le transfert d'une information, sur le bus, du contrôleur vers les périphériques (phase émetteur).

2. Module BYTRCV (Byte Receive): qui assure le transfert d'une information d'un périphérique vers le contrôleur (phase récepteur).

Ces modules sont invoqués un certain nombre de fois au cours de l'exécution des routines:

GPBINI : invoque deux fois BYTSND pour l'envoi de la commande UNT et de la commande UNL.

MXSND : invoque deux fois BYTSND pour l'envoi de l'adresse du multiplexeur, puis de la donnée nécessaire à la programmation du multiplexeur.

DATSND : invoque BYTSND une fois pour l'envoi de l'adresse du convertisseur D/A. Ensuite, pour chaque élément du tableau de données (appelé ARRAY dans le précédent chapitre), DATSND invoque deux fois BYTSND pour envoyer au convertisseur les 2 octets de données nécessaires à la conversion.

DATRCV : invoque BYTSND une seule fois, lors de l'envoi de l'adresse du convertisseur A/D. A partir de deux données consécutives issues d'une opération de conversion du A/D et fournies par invoquation du module BYTRCV, DATRCV compose une nouvelle donnée qu'il peut stocker dans un tableau.

Les phases d'adressage des routines MXSND, DATSND et DATRCV sont assurées par un module ADRESS (Adressage) qui place la ligne ATN au niveau vrai, puis procède à l'envoi de l'adresse grâce au module BYTSND, et ensuite replace la ligne ATN au niveau faux.

Les derniers modules d'entrée/sortie sont les modules OUTPOR, INPOR et PIOPGM.
OUTPOR permet de configurer en mode de sortie le port A du PIO, ainsi que les buffers du port A sur lesquels est raccordé le bus des lignes de données.
Le module INPOR configure le port A et ses buffers en mode d'entrée.
Le module PIOPGM sert à programmer le port B du PIO en mode de contrôle.

OUTPOR est invoqué:
- par GPBINI avant d'envoyer sur le bus, les commandes UNL et UNT.
- par MXSND avant d'envoyer d'une part l'adresse du multiplexeur et d'autre part la donnée servant à le configurer.
- par DATSND avant d'envoyer l'adresse du convertisseur D/A et de réaliser l'envoi de chaque donnée au convertisseur.
- par DATRCV avant d'envoyer l'adresse du convertisseur A/D.

INPOR est invoqué par DATRCV avant la réception des données envoyées par le convertisseur A/D. Quant à la routine AFFMES, elle utilise les routines d'entrées/sorties de chaine de caractères du BDOS(*)de CPM

(*) BDOS: Basic Disk Operating System. Le BDOS est la partie de CP/M renfermant toutes les procédures d'entrée/sortie avec les périphériques.

III.1.2. Modules de traitement des données.

Les modules de traitement des données servent aux opérations de conversion de données, que nécessitent les convertisseurs utilisés (cfr. I.2.5.3. et I.2.5.4.).

La routine DATSND comporte deux modules de traitement:

1. module DATCVR (Data Conversion) dont la fonction est de convertir un élément du tableau de données que l'on désire envoyer au convertisseur, en une donnée binaire par application de la formule vue au paragraphe I.2.5.4.

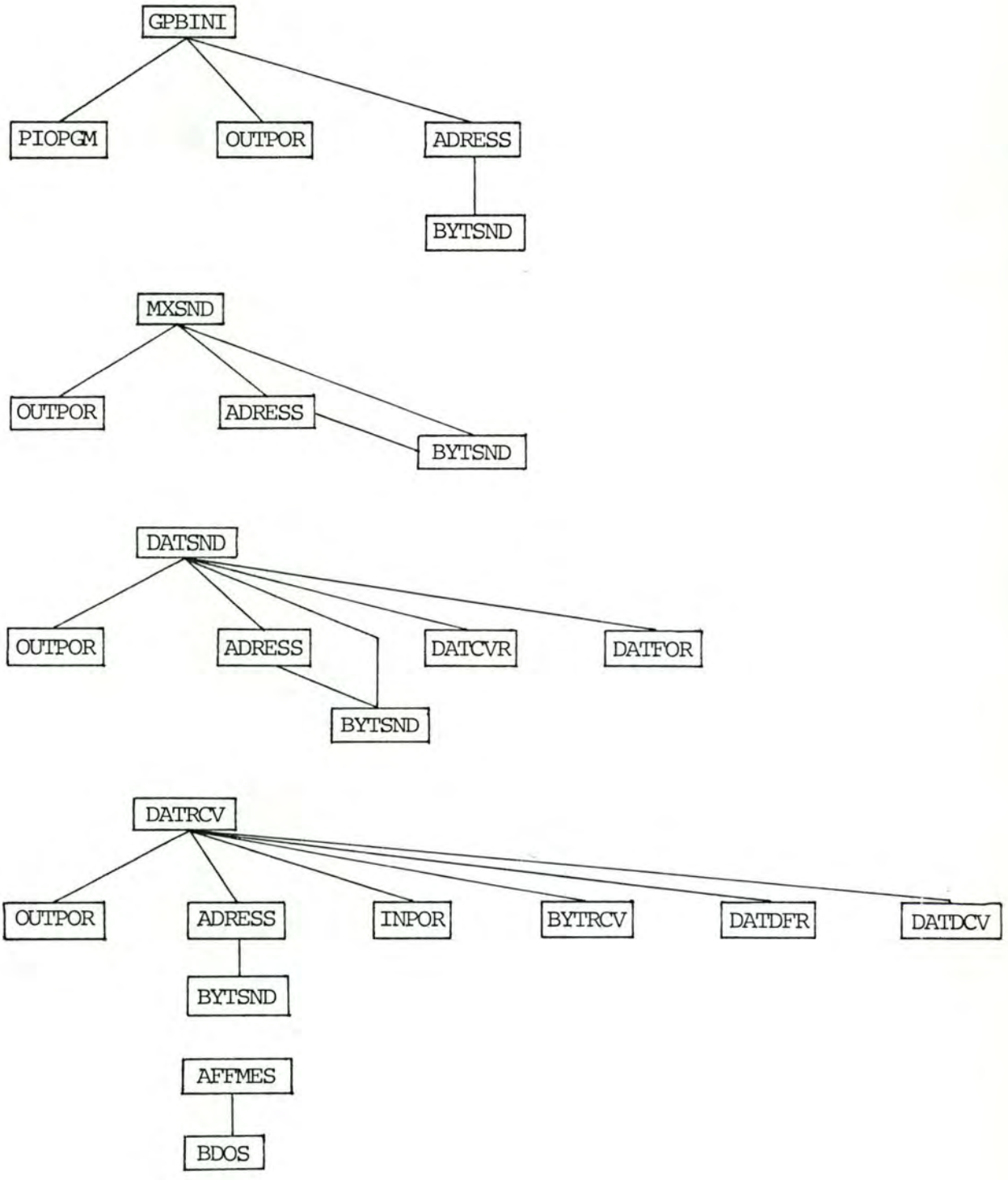
2. module DATFOR (Data Formatter) dont la fonction est de transformer la donnée binaire venant du module DATCVR en deux octets dont l'un est constitué par les 6 bits de poids faibles, de la donnée binaire, précédés par les bits 10 et l'autre est constitué par les 6 bits suivants, mais, cette fois, précédés par les bits 01 (cfr. paragraphe I.2.5.4.).

La routine DATRCV comporte elle-aussi deux modules de traitement:

1. module DATDFR (Data "De Formatter") dont la fonction est de reconstituer à partir de deux octets envoyés par le convertisseur A/D, une donnée selon les principes donnés au paragraphe I.2.5.3.

2. module DATDCV (Data "De Conversion") dont le rôle est de convertir la donnée sortant du module DATDFR en une valeur réelle, d'après la formule du paragraphe I.2.5.4.

III.1.3. Architecture générale des routines.



III.1.4. Choix d'un langage

L'entièreté des routines d'échanges de données a été développée en langage assembleur Z-80. Les raisons de ce choix sont les suivantes: les langages de haut niveau (BASIC, FORTRAN, PASCAL) ne permettent pas d'accéder aux ports d'entrée/sortie du système; de plus, le langage assembleur assure une plus grande rapidité d'exécution qu'un langage de haut niveau, même compilé. Enfin, les routines ainsi développées peuvent être utilisées à partir de programmes d'application écrits dans différents langages.

III.2. DEMARCHE DE CONCEPTION DU LOGICIEL DE CONNEXION AVEC LE DEC-20

Les principes de bases de la connexion avec le DEC-20, étudiés en II.2.2., ainsi que les spécifications des traitements, vus au paragraphe II.2.5., nous ont amené à la conception d'un ensemble de modules d'entrée/sortie, responsables de la gestion du port série A du micro-ordinateur.

Vu les possibilités offertes par le programme de connexion au niveau du dialogue avec l'utilisateur, un ensemble de modules de gestion d'écran a été conçu.

Nous commencerons par présenter les modules dérivés directement des spécifications fonctionnelles du logiciel.

III.2.1. Modules dérivés de l'analyse fonctionnelle

Les spécifications des traitements font apparaître la présence d'un certain nombre de fonctions communes aux trois phases mises en évidence.

Ces fonctions sont:
- initialisation du SIO
- exécution du LOGIN sur le DEC
- initialisation de la ligne
- fin de session.

Chacune de ces fonctions est assurée par un module: il s'agit, dans l'ordre, des modules DECINI, LOGINDEC, TERMINI, LOGOUT.

Les autres fonctions dégagées au chapitre précédent sont:

- mode terminal
- acquisition des noms de fichiers (transfert vers le micro)
- réception d'un fichier par le micro-ordinateur
- acquisition des noms de fichiers (transfert vers le DEC).
- envoi d'un fichier, vers le DEC.

A chacune de ces fonctions est associée un module, sauf pour les deux fonctions d'acquisitions des noms de fichiers qui sont regroupées en un seul module, à cause de leur similitude.

Les modules associés sont dans l'ordre:

TERMINAL	: mode terminal
QUERYFILENAME	: acquisition des noms de fichiers concernés par le transfert
FILESEND	: envoi d'un fichier
FILERECEIVE	: réception d'un fichier.

Le module TERMINAL, qui donne la possibilité au micro de travailler comme terminal du DEC-20, renferme un module, appelé TERM et écrit en langage machine, qui assure le dialogue entre l'utilisateur et le DEC-20 via l'interface série.

Etant donné que l'utilisateur peut souhaiter transférer plusieurs fichiers l'un à la suite de l'autre, soit du micro vers le DEC, soit du DEC vers le micro, deux modules supplémentaires ont été élaborés.

Il s'agit des modules:

TRANSDEC: transfert de plusieurs fichiers vers le DEC

TRANSMIC: transfert de plusieurs fichiers vers le micro.

Ces modules ne font qu'utiliser les modules précédemment définis, à savoir pour TRANSDEC, les modules FILESEND et QUERYFILENAME, et pour TRANSMIC, les modules FILERECEIVE et QUERYFILENAME.

Le module TERMINAL permet de travailler comme sur un terminal du DEC. Après affichage du message signifiant le passage en mode terminal (cfr II.2.4.1.), tous les caractères que l'utilisateur tape au clavier sont envoyés au DEC; leurs échos ainsi que les caractères renvoyés par le DEC sont affichés à l'écran du micro.
On quitte le mode terminal par appui sur la touche ‹Backspace› .

Le module QUERYFILENAME est chargé de demander à l'utilisateur les noms des fichiers qu'il désire transférer: ces noms sont un nom de fichier sur le DEC (FILENAMEDEC) et un nom de fichier sur le micro (FILENAMEMIC). Ces noms vont servir aux modules de transfert FILESEND et FILERECEIVE.

Le module FILESEND procède au transfert d'un fichier du micro, dont le nom est FILENAMEMIC, vers un fichier FILENAMEDEC situé sur le DEC (cfr II.2.3.2.).

Le module FILERECEIVE procède au transfert d'un fichier du DEC, appelé FILENAMEDEC, vers un fichier du micro FILENAMEMIC (cfr II.2.3.1.).

Le module TRANSDEC permet de transférer plusieurs fichiers vers le DEC l'un à la suite de l'autre, sans repasser par le menu principal (cfr figure II.2.).
Après avoir demandé à l'utilisateur les noms des fichiers à transférer (QUERYFILENAME), et avoir procédé au transfert (FILESEND), on demande à l'utilisateur s'il faut encore transférer un nouveau fichier (cfr page 67). Si oui, on demande des nouveaux noms et on recommence un transfert, si non on retourne au menu principal.

Le module TRANSMIC agit de la même façon, mais ici les transferts se déroulent du DEC vers le micro.

III.2.2. Modules d'entrée/sorite

Les modules d'entrée/sortie sont responsables de la gestion du port série A du circuit SIO. Ces modules sont tous implémentés en langage-machine Z-80 pour deux raisons:

la première tient au fait que le JRT PASCAL, qui a servi à l'implémentation du logiciel de connexion, ne permet pas d'effectuer des entrées/sorties sur le port série du micro. La deuxième raison vient de contraintes de vitesse lors d'échange de caractères dans le sens DEC vers le micro.

Examinons en premier lieu les modules d'entrée de caractères sur le port série A.
Ils sont au nombre de deux:

PAGERD: module de lecture d'une "page"; c'est ce module qui est chargé de lire les caractères envoyés, sur la ligne, par le DEC, lorsqu'il exécute la commande TYPE, envoyée par le micro. Ce module est donc utilisé par le module FILERECEIVE décrit au paragraphe précédent. Les caractères lus sont stockés par PAGERD, dans un buffer, en mémoire centrale.

READPR: Ce module lit tous les caractères envoyés par le DEC, jusqu'à ce que ce dernier envoie le caractère "ↂ", qui est le prompt du TOPS-20. READPR n'affiche aucun des caractères lus: il se contente de positionner une variable booléenne en entrée à la valeur vraie, si un des caractères est soit "?", soit "%".
Il est utilisé à deux moments dans le module LOGINDEC: le premier lors de l'envoi au DEC du caractère Control/C, de façon à lire le message d'accueil du DEC, avant d'envoyer la commande LOGIN; le deuxième lors de l'envoi des coordonnées de l'utilisateur, de façon à détecter la validité de celles-ci: en effet, le module analyse chaque caractère lu; si, lors du LOGIN, le module READPR détecte le caractère "?", c'est que les

coordonnées de l'utilisateur sont invalides et que, par conséquent, il est nécessaire de recommencer le LOGIN.
Le module READPR permet également de valider un nom de fichier, qui se trouverait dans le directoire de l'utilisateur, sur le DEC.
On procède de la façon suivante: le programme de connexion envoie la commande DIRECTORY suivie du nom de fichier dont on désire vérifier l'existence.
Ensuite, on appelle le module READPR. De deux choses l'une: ou bien le fichier existe et, dans ce cas, le DEC renvoie simplement le nom du fichier puis repasse en mode commande, ou bien le fichier n'existe pas et le DEC renvoie alors le message:

%FILE NOT FOUND

Le module READPR analyse ces caractères: en se basant sur le caractère "%", on peut donc savoir si, oui ou non, le fichier existe bien. Le module READPR est donc utilisé dans le module QUERYFILENAME, pour la validation des noms de fichiers.

Passons maintenant aux modules de sortie sur le port série A. Ils sont également au nombre de deux:

STROUT: envoi d'un caractère ou d'une chaine de caractères sur le port série A avec lecture de l'écho de chaque caractère envoyé.

CHROUT: envoi d'un caractère ou d'une chaine de caractère sans lecture de l'écho.

Le module STROUT est utilisé par tous les modules voulant envoyer des commandes au DEC, et des caractères lors des transferts de fichiers (module FILESEND).

Le module CHROUT est destiné plus spécialement à l'envoi de caractères de contrôle pour lesquels le DEC ne renvoie pas l'écho, tels que Control/C et Control/Q. CHROUT permet également d'envoyer le mot de passe de l'utilisateur au DEC: on sait en effet, que sur le DEC, lorsque l'utilisateur tape son mot de passe, ce dernier reste invisible à l'écran, car le DEC ne renvoie aucun écho.

III.2.3. Modules de gestion d'écran.

Les modules de gestion d'écran ont comme rôle de permettre de constituer des grilles d'écran agréables pour l'utilisateur, en tirant parti des possibilités offertes par le terminal utilisé.

Ces modules sont également implémentés en langage-machine Z-80: ils pourront ainsi être utilisés à partir de programmes écrits en divers langages de haut niveau.

Les modules de gestion d'écran sont les suivants:

HOME : amène le curseur dans le coin supérieur gauche de l'écran.
BLANK : définit le début d'une zone d'écran en vidéo-inverse.
ENDBL : définit la fin d'une zone en vidéo-inverse.
CLREOL : efface une ligne à partir de la position courante du curseur.
CLREOS : efface l'écran à partir de la position courante du curseur.
CLRSCR : efface l'entièreté de l'écran.
GOTOXY (x, y): placer le curseur en position de coordonnées (x,y) sur l'écran: x désigne le numéro de colonne (entre 0 et 79), et y le numéro de la ligne (entre 0 et 23); le coin supérieur gauche de l'écran est le point de coordonnées (0,0).

Il reste à présenter deux modules qui échappe à la classification présentée. Il s'agit des modules:

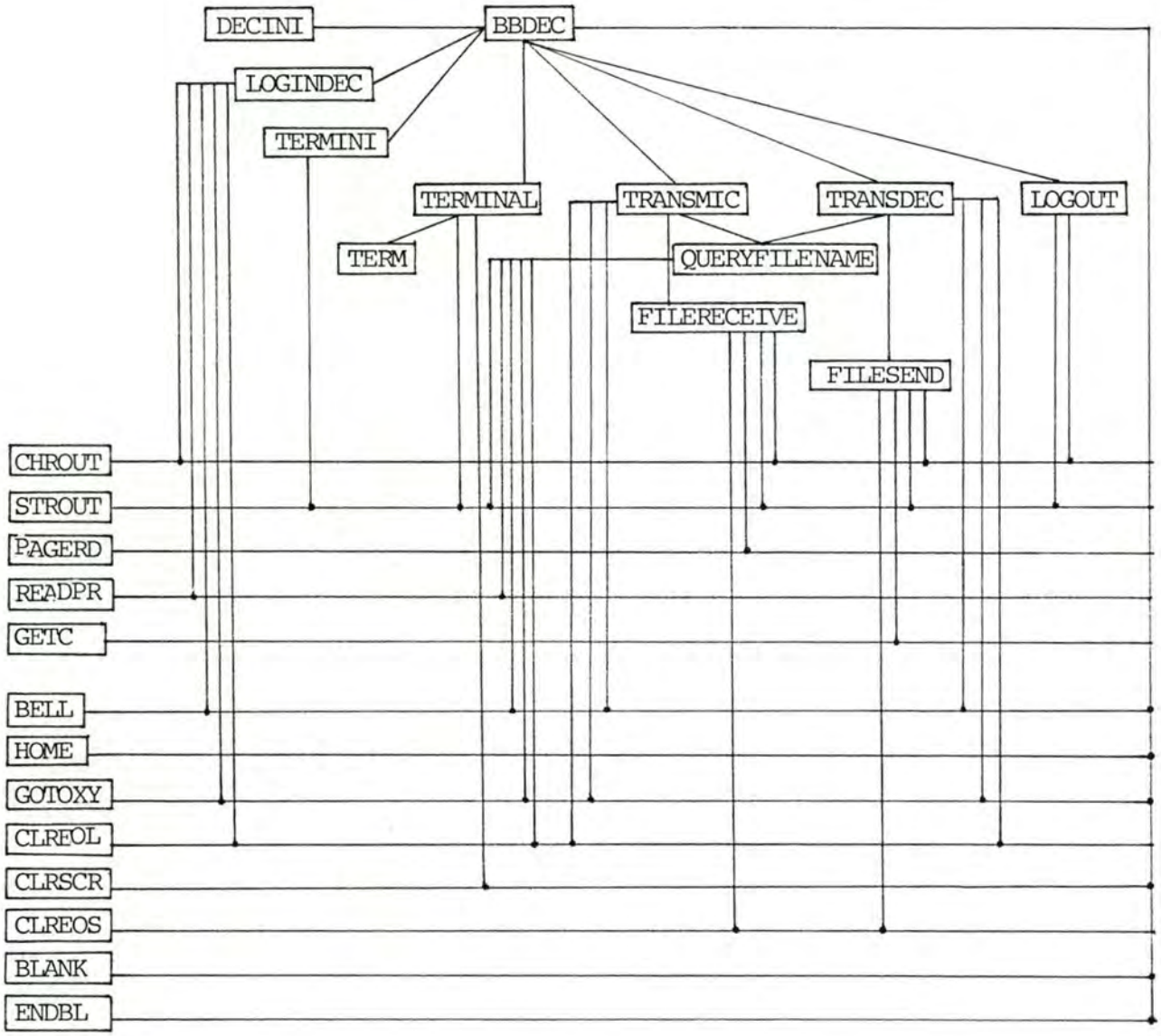
GETC : saisie au vol d'un caractère tapé au clavier.

BELL : émission d'un "beep" sonore.

Le module GETC permet une scrutation du clavier: ce module, utilisé dans le module FILESEND autorise la détection, pendant l'envoi d'un fichier, du caractère Control/Z tapé par l'utilisateur, lorsque celui-ci a décidé d'arrêter l'opération de transfert.

Le module BELL a comme rôle d'émettre un signal sonore de façon à attirer l'attention de l'utilisateur lorsqu'une question lui est posée, ou encore lors de l'affichage d'un message d'erreur.

III.2.4. ARCHITECTURE GENERALE DU LOGICIEL DE CONNEXION



III.2.5. CHOIX D'UN LANGAGE

Mis à part les modules d'entrée/sortie et de gestion d'écran, développés en langage assembleur Z-80 pour les raisons décrites plus haut (cfr III.2.2. et III.2.3.), les autres modules, c'est-à-dire ceux dérivés de l'analyse fonctionnelle, sont écrits en langage JRT PASCAL, principalement en raison de la souplesse et à la rapidité de développement qu'offre un langage de haut niveau.
L'utilisation d'un tel langage assure en plus une portabilité accrue du logiciel ainsi implémenté.

CHAPITRE IV

CHAPITRE IV. PHILOSOPHIE DE TEST DES PROGRAMMES

IV.1. TEST DES ROUTINES D'ECHANGES DE DONNEES.

La phase de test des routines a nécessité la conception de nombreux programmes, écrits en FORTRAN, utilisant nos routines et permettant ainsi d'exploiter et de tester toutes leurs possibilités (voir par exemple le programme en annexe A.I.1.3.) afin de mettre les erreurs en évidence.

Ce sont principalement les modules d'entrée/sortie BYTSND et BYTRCV qui ont posé le plus de problèmes. Le moyen le plus efficace pour détecter des erreurs dans les protocoles de transmission sur un bus consiste à utiliser un analyseur logique raccordé sur ce bus.
Ne disposant pas d'un tel appareil, nous avons simplement utilisé un oscilloscope à double trace, connecté sur au plus deux fils du bus d'interfaçage; pendant l'exécution en pas à pas des modules de transfert, il était possible de visualiser les transitions des signaux sur les différentes lignes.
La validité des effets des modules BYTSND et BYTRCV démontre à postériori celle des modules de configuration des ports du PIO (INPOR, OUTPOR et PIOPGM).

A un niveau plus élevé, nous avons également procédé à de nouveaux tests des modules BYTSND et BYTRCV appelés directement à partir d'un programme de test écrit en FORTRAN.

En particulier, nous avons rédigé un programme permettant de réaliser l'adressage d'un périphérique du bus. Chaque périphérique est équipé d'une diode à émission lumineuse signalant à l'utilisateur son état d'adressage: lorsque la diode est allumée, le périphérique est adressé. Par conséquent, il était facile de constater de visu la réussite d'une tentative d'adressage.
Ce même programme permettait également de tester l'effet des commandes UNL et UNT, puisque, par envoi de ces commandes, on pouvait constater la désactivation des périphériques par extinction de leur diode.
La robustesse de ces modules BYTSND et BYTRCV vis-à-vis des erreurs a été mise à l'épreuve simplement en court-circuitant certains fils du bus de synchronisation, et en tentant également d'adresser des périphériques inexistants.
Il suffisait alors de vérifier que le code-erreur était positionné à des valeurs non nulles.

Chaque module de traitement de données (DATCVR, DATFOR, DATDCV, DATDFR) a été testé isolément, en leur fournissant les paramètres d'entrée et en comparant les résultats fournis par ces modules avec la réponse exacte calculée à la main.

Ensuite, nous avons procédé au test de la routine DATSND de la façon suivante: nous avons connecté un voltmètre à la sortie du convertisseur D/A. En envoyant différentes valeurs de tensions au D/A (balayage entre - 10 et + 10), grâce à DATSND, on pouvait constater sur le voltmètre d'une part que l'envoi de données fonctionnait normalement, mais également que les opérations de formatage et de conversion étaient correctes.
Le test de la routine DATRCV s'est déroulé en connectant la sortie du D/A avec l'entrée du A/D; grâce à DATSND, on envoie une donnée au D/A; celle-ci se retrouve donc à l'entrée du A/D. La routine DATRCV permet alors de recevoir la donnée en sortie du A/D.
L'égalité entre la donnée envoyée par DATSND et celle reçue par DATRCV, permet de s'assurer du bon fonctionnement de DATRCV.

IV.2. TEST DU PROGRAMME DE CONNEXION AVEC LE DEC-20

La phase de test du programme de connexion portait essentiellement sur les modules d'entrée/sortie du SIO.

Les premiers modules que nous avons testés sont les modules DECINI et TERM qui permettent d'initialiser le SIO puis de travailler à partir du micro, en terminal du DEC.

Des essais ont ensuite été réalisés pour envoyer des commandes au DEC, directement à partir d'un programme écrit en PASCAL, grâce aux modules STROUT et CHROUT. En passant en mode terminal immédiatement après l'envoi de la commande, on pouvait constater l'effet de cette dernière. Grâce aux modules STROUT et CHROUT, nous avons pu d'une part réaliser un LOGIN automatique sur le DEC (module LOGINDEC), et, d'autre part, transférer un fichier du micro vers le DEC après envoi de la commande COPY (module FILESEND).

Nous avons ensuite procédé au transfert de fichier du DEC vers le micro, par la commande-système TYPE, grâce au module PAGERD.

Lorsque le programme a été complètement rédigé, nous avons testé toutes ces possibilités en effectuant de nombreux transferts entre les deux machines: tous les fichiers-sources des programmes que nous avons développés, ont été transférés sur le DEC, dans le but de les archiver et de tester le programme de connexion, mais aussi pour les imprimer grâce aux imprimantes rapides du Centre de Calcul: les listes des programmes sont présentées dans la partie annexe de ce travail.

Enfin, la conclusion de la phase de test a été assurée par les utilisateurs eux-mêmes, qui ont pu juger de la validité des opérations de transfert et de la qualité conversationnelle du produit développé.

CHAPITRE V

CHAPITRE V. PROBLEMES RENCONTRES

Le principal élément que nous avons eu à affronter au cours de ce travail, fut le facteur temps.
L'interface permettant de relier le bus au micro-ordinateur n'a été fournie qu'au début du mois d'octobre de l'année 1983. Le JRT PASCAL n'a pu être obtenu que le mois suivant.

Le temps disponible pour l'implémentation des deux produits a de plus été réduit par le fait de mon entrée au service militaire, le 1er décembre de la même année.
Ceci nous a amené à réduire nos ambitions quant à la sophistication des logiciels développés, et, principalement, du logiciel de connexion. De plus, la ligne, installée au Département de Chimie, est utilisée pour le raccordement au DEC du seul terminal graphique que compte le bâtiment de Chimie. La ligne est donc très souvent monopolisée par les utilisateurs chimistes, avec les désagréments que cela implique pour mener à terme l'implémentation et les tests du logiciels.

CHAPITRE VI

CHAPITRE VI. ETAT FINAL DU PROJET ET SES LIMITATIONS

Malgré les problèmes rencontrés au cours de ce travail, nous pensons avoir répondu aux exigences des utilisateurs, malgré quelques limitations que nous allons rappeler maintenant.

La principale limitation relative aux routines d'échange de données réside dans la faiblesse de la fréquence d'échantillonnage (routine DATRCV): ceci s'explique d'une part par le fait que les opérations de conversion sont couteuses au point de vue temps de calcul, et, d'autre part, à cause de l'architecture adoptée pour l'interface, qui nous a obligés à gérer le protocole d'échanges sur le bus par logiciel, plutôt que par matériel.

Le logiciel de connexion avec le DEC-20 renferme quelques restrictions. Il n'est pas possible de vérifier l'existence d'un fichier sur le micro-ordinateur avant de procéder à son transfert.
De plus, l'utilisateur doit s'assurer lui-même qu'avant toute opération de transfert, il dispose d'une mémoire de masse suffisante aussi bien sur les disquettes que sur les disques du DEC.
Enfin, on peut regretter la lenteur relative des opérations de transfert, due au fait que la transmission des caractères se déroule de façon asynchrone.

CHAPITRE VII

CHAPITRE VII. CONCLUSIONS

Dans ce travail, nous nous sommes proposé de mettre au point un dispositif d'acquisitions de données, à partir d'une configuration matérielle mise à notre disposition par le Département de Chimie.

Notre contribution a consisté en l'élaboration de deux logiciels: l'un est destiné à effectuer des mesures ainsi que le contrôle des paramètres sur un dispositif expérimental. L'autre offre la possibilité de transférer ces mesures vers un ordinateur de plus grande puissance en vue du traitement et de l'archivage de ces mesures.
Les capacités de ce dernier logiciel ont par ailleurs été étendues afin d'offrir à l'utilisateur la possibilité de travailler à partir du système informatique de contrôle, comme terminal de l'ordinateur de traitement; cette option permet notamment à l'utilisateur de bénéficier des logiciels de cet ordinateur.

Ce travail a permis de mettre en évidence et de résoudre les difficultés posées d'une part par la mesure de grandeurs analogiques à partir d'un système informatique, et d'autre part par les échanges de données entre des machines de puissance différente.

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

1. BASTIDE G., VELLAS J.R., Mise en oeuvre du bus I.E.E.E.-488 - utilisation et réalisation d'appareils, Editests, Paris 1982

2. TOWNSEND CARL, How to get started with CP/M, Dilithium Press. 1980.

3. TUTORIAL DESCRIPTION OF THE HEWLETT-PACKARD INTERFACE BUS, document H.P. 1980.

4. VANOBBERGHEN W., Conception et réalisation d'un contrôleur de bus I.E.E.E.-488, mémoire de fin d'études 1983-1984. Institut d'informatique, F.U.N-D.P. Namur.

5. ZAKS RODNAY, How to program Z-80 . Sybex, Paris, 1980.

6. ZAKS R., LESEA A., Techniques d'interfaçage au microprocesseur, Sy bex, Paris, 1978.

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique

Année Académique 1983-1984.

CONTRIBUTION A LA MISE AU POINT D'UN SYSTEME D'ACQUISITION DE DONNEES

ANNEXES

DEBRUN Jacques

Mémoire présenté en vue de l'obtention du grade de Licencié et Maître en Informatique.
Promoteur de mémoire: Monsieur le Professeur Van Bastelaer.

ANNEXES

ANNEXE I

A.I. MODE D'EMPLOI DES PROGRAMMES

A.I.1. Mode d'emploi des routines d'échanges de données.

Nous consacrerons ce paragraphe à la façon d'utiliser les routines d'échanges de données, à partir de programmes écrits en FORTRAN.

A.I.1.1. Assemblage des routines

L'assemblage des routines est effectué grâce à l'assembleur M80 de Microsoft. Chacune des routines occupe un fichier, dont le nom est celui de la routine et l'extension est MAC.

L'assemblage des routines se fait, sous CP/M, grâce à la commande:

M 80 ‹nom fichier 1› = ‹nom fichier 2›

où ‹nom fichier 2› est le nom du fichier contenant la routine à assembler et ‹nom fichier 1› est le nom du fichier contenant la routine assemblée. L'extension de ce fichier est REL.

A.I.1.2. Edition des liens

Cette phase va donner lieu à la création d'un fichier, dont l'extension est COM, contenant le programme d'échange de données relié avec les routines.

L'édition des liens se fait grâce à la commande:

L 80

A ce moment, on se retrouve en mode commande dans l'éditeur de liens. Ceci est signalé par une astérisque, en première colonne de l'écran. Il faut alors donner la liste des fichiers que l'on veut relier. Par exemple, supposons que le programme-utilisateur s'appelle TEST 1 et que l'on désire le relier avec l'ensemble des routines d'échanges de données; on tape pour ce faire la ligne suivante, à la suite du "prompt" de L 80:

```
* GPBINI, MXSND, DATSND, DATRCV, AFFMES, TEST 1, TEST 2 / N / E.
```

La commande comporte la liste des fichiers qu'occupent les routines, le nom du fichier contenant le programme-utilisateur, puis le nom du fichier exécutable (ici TEST 2) que L 80 doit créer.
La commande est terminée par deux "switches" qui ont pour effet de créer un fichier exécutable et de le sauver sur disquette.

A.I.1.3. Exemple de programme-utilisateur

L'exemple que nous présentons ici montre la façon d'utiliser les routines; on remarquera aussi la déclaration des paramètres des routines.

```
      program acq
      dimension array(20)
      integer length
      integer*1 lstnr,byte,talker,error
      real array,val


6     write(5,10)
10    format(' adresse du multiplexeur:')
      read(5,15)lstnr
15    format(i3)
      write(5,20)
20    format(' numero de l''appareil a selectionner:')
      read(5,25)byte
25    format(i3)
      call gpbini(error)
      call gftmes(error)
      if (error.gt.0) goto 6
      call gxspo(lstnr,byte,error)
      call gftmes(error)
      if (error.gt.0) goto 6
      write(5,89)
89    format(' valeur a envoyer au convertisseur D ==> A :')
      read(5,120)val
120   format(f7.3)
      do 90 i=1,10
      array(i) = val
90    continue
20    write(5,30)
30    format(' adresse du convertisseur D ==> A :')
      read(5,35)lstnr
35    format(i3)
      write(5,38)
38    format('++++++  envoi  ++++++')
      call gpbini(error)
      call gftmes(error)
      if (error.gt.0) goto 26
      length=10
      call gatsnd(lstnr,array,length,error)
      call gftmes(error)
      if (error.gt.0) goto 26

40    write(5,300)
300   format(' adresse du convertisseur A ==> D :')
      read(5,350)talker
350   format(i3)
      write(5,45)
45    format(' +++++ acquisition +++++')
      do 303 i=1,10
      array(i)=0.
303   continue
      length=10
      call gpbini(error)
      call gftmes(error)
      if (error.gt.0) goto 46
      call gatrcv(talker,array,length,error)
      call gftmes(error)
      if (error.gt.0) goto 46
```

```
      do 500 i=1,10
      write(5,550)array(i)
550   format(f8.2)
500   continue
      write(5,600)
600   format(' pour un autre essai ,tapez un nombre >= 0 :')
      read(5,605)i
605   format(i3)
      if (i.ge.0) goto 6
      stop
      end
```

Le programme commence par la déclaration des variables:

- un tableau de réels ARRAY qui va contenir les données soit que l'on va envoyer soit que l'on va recevoir.
- une variable entière LENGTH (2 octets) qui sera le nombre de données à recevoir ou à envoyer.
- quatre variables entières (un seul octet par variable):
    - LSTNR représente l'adresse soit du multiplexeur, soit du convertisseur D/A
    - BYTE représente la donnée pour configurer le multiplexeur
    - TALKER est l'adresse du convertisseur A/D
    - ERROR est le paramètre d'erreur servant à diagnostiquer les opérations de transfert.

Le programme commence par demander, à l'utilisateur, l'adresse du multiplexeur (lignes 6 à 15), puis le numéro de l'appareil à sélectionner (lignes 15 à 25). Vient ensuite l'initialisation du bus (call GPBINI) et l'affichage d'un message d'erreur (call AFFMES).
Si une erreur s'est produite, on recommence au début du programme, sinon, on procède à la configuration du multiplexeur, et on affiche ensuite un message de diagnostic du transfert. On demande ensuite (lignes 89 à 120) quelle valeur on veut envoyer au convertisseur D/A. Cette valeur sert à remplir un tableau de 10 éléments (lignes 120 à 90).
Après avoir donné l'adresse du convertisseur D/A (lignes 26 à 35), le bus est réinitialisé de façon à désactiver le multiplexeur précédemment adressé. Puis le transfert des données du tableau se déroule, grâce à l'appel de la routine DATSND. On procède, ensuite, à l'acquisition de données, en demandant l'adresse du A/D (lignes 46 à 350), en remettant à zéro le tableau qui va recevoir les données envoyées par le A/D, en initialisant le bus pour désactiver le convertisseur D/A, et, enfin, à la réception des données par appel de DATRCV. Le tableau contenant les données est affiché à l'écran. Le programme se termine en proposant une nouvelle exécution

A.I.2. Mode d'emploi du programme de connexion avec le DEC-2Ø

L'exécution du programme de connexion se fait, à partir de CP/M, en donnant la commande:

EXEC BBDEC

où EXEC est le nom d'un fichier appartenant au système JRT, et, dont l'extension est COM, contenant les routines-systèmes nécessaires à l'exécution. BBDEC est le nom d'un fichier avec toutes les routines (entrée/sortie sur le SIO, gestion d'écran) déclarées en EXTERN
Si un problème se produit pendant l'exécution de BBDEC, provoquant un retour à CP/M, on peut se tirer d'affaire grâce à un petit programme écrit en assembleur Z-80 et qui permet de se reconnecter au DEC en transformant le micro en terminal. Ce programme s'exécute, sous CP/M, en donnant le nom du programme, à savoir DECLNK.

En cas de modification à apporter au programme BBDEC ou à l'une de ses routines, il est bon de savoir comment s'y prendre pour compiler et exécuter la nouvelle version du programme.

A.I.2.1. Assemblage des routines externes.

Le programme BBDEC comporte un certain nombre de routines externes, écrites en assembleur Z-80.

L'assemblage de ces routines est effectué par M 80 (voir A.I.1.1.). M 80 crée un fichier d'extension REL, qui n'est pas compatible avec les fichiers résultant d'une compilation en JRT PASCAL.

Un utilitaire, fourni avec le logiciel JRT, permet de convertir les fichiers créés par M 80 en fichiers compatibles avec ceux créés par JRT. Cet utilitaire est appelé par la commande:

EXEC CONVERTM.

A ce moment, l'utilitaire CONVERTM demande à l'utilisateur les noms des fichiers à convertir. Pour chaque fichier en entrée, CONVERTM crée un fichier de même nom, mais dont l'extension est INT, et est compatible, c'est-à-dire reliable, avec les programmes développés en JRT.

A.I.2.2. Edition des liens.

L'édition des liens en JRT se passe grâce à l'utilitaire LINKER. La commande est:

EXEC LINKER

LINKER demande alors le nom du programme et le nom des routines à relier. Une fois l'opération effectuée, il en ressort un seul fichier dont le nom et l'extension sont identiques à ceux du fichier contenant le programme non relié; l'extension de ce dernier est par ailleurs modifié en IN2, pour ne pas confondre les fichiers.

A.I.2.3. Exécution.

L'exécution a lieu en tapant la commande:

EXEC <nom fichier>

où <nom fichier> est le nom d'un fichier résultant de l'édition des liens.

ANNEXE II. SPECIFICATIONS DES MODULES.

A.II.1. MODULES DES ROUTINES D'ECHANGES DE DONNEES

1. Module AFFMES

Entrée: ERROR: - paramètre d'erreur permettant de diagnostiquer un transfert sur le bus.

pré-condition: - ERROR doit être déclaré en INTEGER*1 et sa valeur est comprise entre 0 et 5.

Sortie: -

post-condition: -

fonction: Le module AFFMES a comme rôle d'afficher à l'écran du terminal, selon la valeur de ERROR, un message d'erreur.

ERROR	Message
0	** communication OK **
1	** listener not ready to receive data **
2	** data not accepted by listener **
3	** data received not validated by talker **
4	** data not unvalidated by talker **
5	** device not present **

Appelle: - routine d'impression de chaîne de caractères, du BDOS
- routine d'impression de caractères, du BDOS.

2. Module GPBINI

entrée: -

pré-condition: -

sortie: - ERROR paramètre d'erreur

post-condition: - ERROR a comme seule valeur possible 0, 1, 2 ou 5

fonction: Le module GPBINI a comme fonction de configurer le port B du PIO en mode de contrôle et de désactiver les listeners et le talker par envoi des commandes UNL et UNT.

Appelle: PIOPGM
OUTPOR
BYTSND

3. Module MXSND

entrée: - LSTNR: adresse du multiplexeur
- BYTE: donnée servant à configurer le multiplexeur

pré-conditions: - LSTNR doit être une adresse appartenant au LAG; elle est déclarée en INTEGER*1
- BYTE peut avoir toute valeur comprise entre 0 et 255; elle est déclarée en INTEGER*1.

sortie: - ERROR paramètre d'erreur

post-conditions: ERROR a comme seule valeur possible 0, 1, 2 ou 5

fonction: - Le module MXSND procède à la configuration du multiplexeur d'adresse LSTNR, en lui envoyant la variable BYTE; si une erreur se produit pendant la transmission, ERROR est positionné à une valeur non nulle.

Appelle: MXPGM
ADRESS
OUTPOR

4. Module DATSND

entrée: - LSTNR: adresse du convertisseur D/A
- ARRAY: tableau de données à envoyer au D/A
- LENGTH: nombre de données à envoyer au D/A

pré-conditions: - LSTNR doit être une adresse appartenant au LAG; elle est déclarée en INTEGER*1
- ARRAY est un tableau de réels, codés sur 4 octets
- LENGTH est un nombre entier déclaré en INTEGER

sortie: - ERROR: paramètre d'erreur permettant de diagnostiquer les opérations de transfert
- LENGTH: donne le nombre de données non envoyées au convertisseur D/A

post-conditions: - ERROR a une valeur comprise entre 0 et 5
- LENGTH a une valeur comprise entre sa valeur initiale (avant exécution de DATSND) et 0

fonction: Le module DATSND permet d'adresser le convertisseur D/A d'adresse LSTNR, et ensuite de lui envoyer un ensemble de LENGTH données stockées dans le tableau ARRAY.
Le déroulement de chaque opération de transfert est diagnostiquée grâce à un code erreur.
Après exécution de DATSND, le paramètre LENGTH donne le nombre de données qui n'ont pas pu être envoyées.
Les éléments de ARRAY sont envoyés séquentiellement à partir de l'élément d'indice 1.

Appelle: DATFOR
DATCVR
BYTSND
ADRESS
OUTPOR

5. Module DATRCV

entrée: - TALKER: adresse du convertisseur A/D
- LENGTH: nombre de données que l'on veut recevoir et stocker dans ARRAY.

pré-conditions: - TALKER est une adresse appartenant au TAG et est déclarée en INTEGER*1
- LENGTH est un nombre entier déclaré en INTEGER

sortie: - ARRAY: tableau dans lequel vont être stockées les données envoyées par le A/D
- LENGTH: nombre de données non reçues
- ERROR: paramètre d'erreur permettant de diagnostiquer les opérations de transfert

post-conditions: - ERROR a une valeur comprise entre 0 et 5
- LENGTH a une valeur comprise entre sa valeur initiale (avant exécution de DATRCV) et 0
- les valeurs des éléments de ARRAY sont compris entre + 10 et - 10

fonction: Le module DATRCV est chargé de recevoir un ensemble de LENGTH données, envoyées par le convertisseur A/D d'adresse TALKER, et de les stocker dans un tableau ARRAY.
Les éléments sont placés séquentiellement dans le tableau à partir de l'indice 1.

Appelle: DATDFR
DATDCV
BYTRCV
ADRESS
INPOR et OUTPOR

6. Module BYTRCV

entrée: - pointeur vers un paramètre d'erreur ERROR

pré-condition: - ce pointeur est passé par la paire de registres HL

fonction: Le module BYTRCV procède à la réception d'une donnée, de 8 bits, envoyée par un des périphériques situés sur le bus.
La réception se déroule conformément à l'organigramme présenté à la figure I.6.
En cas d'erreur pendant le transfert, un code erreur est positionné à une valeur non nulle (cfr II.1.3.).

sortie: - donnée de 8 bits reçue

post-condition: - la donnée est stockée dans l'accumulateur A

Appelle: -

7. Module BYTSND

entrée: - pointeur vers un paramètre d'erreur ERROR
- donnée de 8 bits à envoyer

pré-conditions: - le pointeur est passé par la paire de registres HL
- la donnée est passée par l'accumulateur

fonction: Le module BYTSND procède à l'envoi d'une donnée de 8 bits, sur le bus. L'émission de la donnée se déroule conformément à l'organigramme présenté à la figure I.6.
En cas d'erreur pendant le transfert, un code-erreur est positionné à une valeur non nulle (cfr II.1.3.).

sortie: -

post-condition: -

Appelle: -

8. Module DATCVR

entrée: - pointeur vers le nombre réel à convertir

pré-condition: - le pointeur est passé par la paire de registres HL du microprocesseur

fonction: Le module DATCVR convertit le nombre réel en entrée en un nombre entier par la formule:

$$B = \text{partie entière} \left(2047 - \frac{2048 * V}{10} \right)$$

où B est un nombre entier, codé sur deux octets et V le nombre réel passé par référence au module

sortie: B nombre entier calculé par la formule précédente

post-condition: B est stocké sur deux octets dans la paire de registres HL

Appelle: routines arithmétiques de la bibliothèque FORTRAN, appelée FORLIB

Ces routines sont:

$L1: chargement de l'accumulateur flottant
$DA: division d'un réel par un entier
$MA: multiplication d'un réel par un entier
$AA: addition d'un réel et d'un entier
$CH: conversion d'un réel en entier.

9. Module DATFOR

entrée: donnée sur deux octets, représentant une donnée à formatter

pré-condition: la donnée est passée par le registre HL

fonction: Le module DATFOR formatte la donnée en deux octets dont la structure est:
10 - - - - - - où les - représentent les 6 bits de poids faible de la donnée
01 + + + + + + où les + représentent les 6 bits suivants de la donnée
Les deux octets sont ainsi formattés en vue de leur envoi au convertisseur D/A

sortie: - les deux octets formattés 10 - - - - - - et 01 + + + + + +

post-condition: l'octet précédé par 10 est stocké dans le registre H et l'octet précédé par 01 est stocké dans le registre L

10. Module DATDCV

entrée: - pointeur vers un élément du tableau ARRAY
- mot de 16 bits, venant du module DATDFR

pré-condition: - le mot de 16 bits est passé par la paire de registres HL
Les 4 bits de poids fort du mot sont nuls
- le pointeur est passé par la paire de registres BC

fonction: Le module DATDCV convertit le mot de 16 bits en entrée en un nombre réel par la formule:

$$V = \frac{10}{2048} \ (2047 - B)$$

dans laquelle B représente le mot de 16 bits et V le nombre réel. V est alors stocké à l'adresse donnée par la paire de registres BC.

sortie: - le nombre réel V

post-condition: - V est compris entre - 10 et + 10

Appelle: routines arithmétiques de la bibliothèque FORTRAN, appelée FORLIB.
Ces routines sont:

- $DA: division d'un réel par un entier
- $MA: multiplication d'un réel par un entier
- $AA: addition d'un réel et d'un entier
- $L1: stockage de l'accumulateur flottant
- $CA: conversion d'un entier en réel

11. Module DATDFR

entrée: - deux octets envoyés successivement sur le bus par le convertisseur A/D

pré-condition: - le premier octet reçu comporte les 8 bits de poids fort d'un mot de 12 bits présent à la sortie du convertisseur
- le deuxième octet reçu comporte les 4 bits de poids faible de ce même mot, cadrés à gauche; les 4 bits restant sont nuls

fonction: Le module DATDFR formatte les deux octets en un mot de 16 bits, dont les 4 bits de poids fort sont nuls et les 12 bits de poids faible représentent le mot de 12 bits présent à la sortie du convertisseur.

sortie: - mot de 16 bits résultant de l'opération de formattage

post-condition: - les 4 bits de poids fort sont nuls
- le mot est passé par la paire de registres HL

Appelle: -

12. Module INPOR

entrée: -

pré-condition: -

fonction: Le module INPOR permet de programmer le port A du PIO et les buffers raccordés au port A en mode d'entrée de données

sortie: -

post-condition: -

Appelle: -

13. Module OUTPOR

entrée: -

pré-condition: -

fonction: Le module OUTPOR permet de programmer le port A du PIO et les buffers raccordés au port A en mode de sortie de données

sortie: -

post-condition: -

Appelle: -

14. Module MXPGM

entrée: - donnée d'un octet servant à configurer le multiplexeur
- pointeur vers un paramètre d'erreur ERROR

pré-condition: - le pointeur est passé par la paire de registres HL
- la donnée est passée par l'accumulateur A

fonction: Le module MXPGM permet de configurer le multiplexeur en lui envoyant une donnée d'un octet, suivant le protocole défini par la norme IEEE-488.
En cas d'erreur pendant le transfert, ERROR est positionné à une valeur non nulle.

sortie: -

post-condition: -

Appelle: BYTSND

15. Module ADRESS

entrée: - adresse ou commande
- pointeur vers un paramètre d'erreur ERROR

pré-condition: - les adresses ou commandes sont formées d'un seul octet, passé par l'accumulateur A

fonction: Le module ADRESS procède à l'envoi d'adresses ou de commandes sur le bus. Pour ce faire, il place la ligne ATN au niveau vrai, envoie la commande ou la donnée, suivant le protocole IEEE-488, puis replace la ligne au niveau faux.

sortie: -

post-condition: -

Appelle: BYTSND

16. Module PIOPGM

entrée: -

pré-condition: -

fonction: Le module PIOPGM permet de programmer le port B du PIO en mode de contrôle et de définir les lignes d'entrée et les lignes de sortie de ce port.

sortie: -

post-condition: -

A.II.2. MODULES DU PROGRAMME DE CONNEXION AVEC LE DEC-2Ø

1. Module BLANK

entrée: -

pré-condition: -

fonction: Le module BLANK permet de commencer une zone d'écran en vidéo-inverse, en envoyant, au terminal, les caractères ESC et j.

sortie: -

post-condition: -

Appelle: CONOUT: routine d'affichage de caractères, appartenant au moniteur.

2. Module CHROUT

entrée: chaîne de caractères

pré-conditions: - la chaîne a une longueur maximale de 255 caractères
- la chaîne est de type STRING

fonction: Le module CHROUT permet d'envoyer une chaîne de caractères sur le port A du circuit SIO: les caractères sont envoyés un par un sans attendre leur écho.
Ceci est utile pour envoyer des caractères de contrôle (CTRL/Q, CTRL/C), ou encore pour envoyer le mot de passe de l'utilisateur, pour lesquels le DEC ne renvoit pas d'écho.

sortie: -

post-condition: -

Appelle: -

3. Module CLREOL

entrée: -

pré-condition: -

fonction: Le module CLREOL permet d'effacer une ligne d'écran à partir de la position courante du curseur, jusqu'à la fin de la ligne, en envoyant au terminal les caractères ESC et T.

sortie: -

post-condition: -

Appelle: CONOUT: routine d'affichage de caractères, appartenant au moniteur.

4. Module CLREOS

entrée: -

pré-condition: -

fonction: Le module CLREOS permet d'effacer l'écran à partir de la position courante du curseur. Les caractères réalisant cette fonction sont ESC et Y.

sortie: -

post-condition: -

Appelle: CONOUT: routine d'affichage de caractères, appartenant au moniteur.

5. Module CLRSCR

entrée: -

pré-condition: -

fonction: Le module CLRSCR permet d'effacer l'entièreté de l'écran. Le caractère réalisant cette fonction est CTRL/Z.

sortie: -

post-condition: -

Appelle: CONOUT: routine d'affichage de caractères, appartenant au moniteur.

6. Module DECINI

entrée: -

pré-condition: -

fonction: Le module DECINI permet d'initialiser le port A du SIO sur lequel est raccordé la ligne téléphonique du DEC-20. L'initialisation consiste à charger les registres de contrôle du SIO de façon à définir les caractéristiques de la transmission:

- Vitesse de la transmission: 2400 bauds
- Nombre de bit stop: 1
- Nombre de bits/caractères: 8
- Mode de transmission asynchrone

sortie: -

post-condition: -

Appelle: -

7. Module ENDBL

entrée: -

pré-condition: -

fonction: Le module ENDBL permet de délimiter la fin de zone d'écran en vidéo-inverse. Les caractères à envoyer à la console sont ESC et k.

sortie: -

post-condition: -

Appelle: CONOUT: routine d'affichage de caractères, appartenant au moniteur.

8. Module GETC

entrée: -

pré-condition: -

fonction: Le module GETC permet d'effectuer une scrutation du clavier afin de détecter si un caractère CTRL/Z a été tapé. Si c'est le cas, une variable booléenne, appelée EOT, est mise à la valeur vraie.

sortie: variable booléenne

post-condition: la variable booléenne est vraie, si on a tapé CTRL/Z au clavier

Appelle: CONST: routine de test de l'état du clavier, appartenant au moniteur.

CONIN: routine de lecture du clavier, appartenant au moniteur.

9. Module GOTOXY

entrée: x, y représentant des coordonnées d'un point à l'écran du terminal

pré-conditions: - x et y sont des nombres entiers
- x est compris entre 0 et 79 (nombre de colonnes)
- y est compris entre 0 et 23 (nombre de lignes)

fonction: Le module GOTOXY permet de positionner le curseur à la colonne x de la ligne y.
Le coin supérieur gauche de l'écran a pour coordonnées (0,0).

sortie: -

post-condition: -

Appelle: CONOUT: routine d'affichage de caractères, appartenant au moniteur.

10. Module HOME

entrée: -

pré-condition: -

fonction: Le module HOME permet de positionner le curseur dans le coin supérieur gauche de l'écran.
Le caractère réalisant cette fonction est CTRL/↑ .

sortie: -

post-condition: -

Appelle: CONOUT: routine d'affichage de caractères, appartenant au moniteur.

11. Module PAGERD

entrée: -

pré-condition: -

fonction: Le module PAGERD lit une "page" de caractères, envoyés vers le port série du micro par le DEC. Cette "page" est envoyée via la commande-système TYPE. Les caractères sont stockés dans un buffer.
Les caractères reçus sont également affichés à l'écran; mis à part les caractères CR et LF, les autres caractères de contrôle ne sont pas envoyés à l'écran.
En cours de réception, on teste si l'utilisateur a tapé le caractère CTRL/Z, pour demander l'arrêt de la réception.

sortie: - buffer contenant les caractères reçus
- variable donnant le nombre de caractères présents dans le buffer
- variable booléenne qui est vraie si l'utilisateur a tapé CTRL/Z.

post-condition: - le buffer a une taille maximale de 65536 caractères

Appelle: - CONIN: routine de lecture du clavier, appartenant au moniteur.
- CONST: routine de test de l'état du clavier, appartenant au moniteur
- CONOUT: routine d'affichage de caractères, appartenant au moniteur.

12. Module READPR

entrée: -

pré-condition: -

fonction: Le module READPR permet de lire, sur le port série A, les caractères envoyés par le DEC; si, parmi le flot des caractères, sont détectés "%" ou "?", alors une variable booléenne est mise à la valeur vraie. L'exécution de READPR s'arrête lorsque l'on rencontre le prompt du DEC = "∂".
Ce module permet de détecter:
1. une erreur dans le mot de passe, lors du LOGIN
2. une erreur dans le nom d'utilisateur
3. la non-existence d'un fichier sur le DEC.

sortie: booléen

post-condition: le booléen est mis à la valeur vraie, si l'on a rencontré le caractère "%" ou "?".

Appelle: -

13. Module STROUT

entrée: chaîne de caractères à envoyer sur la ligne, via le port série A.

pré-conditions: - la longueur de la chaîne est limitée à 255 caractères
- la chaîne est de type STRING

fonction: Le module STROUT permet d'envoyer une chaîne de caractères sur le port série A: les caractères sont envoyés un par un, et on attend leur écho, après chaque caractère envoyé.

sortie: -

post-condition: -

Appelle: -

14. Module TERM

entrée: -

pré-condition: -

fonction: Le module TERM permet de transformer le Bigboard en un terminal du DEC.
Pour quitter le mode terminal, il suffit d'appuyer sur la touche BACKSPACE.

sortie: -

post-condition: -

Appelle: CONST: routine de test de l'état du clavier, appartenant au moniteur
CONIN: routine de lecture d'un caractère tapé au clavier, appartenant au moniteur
CONOUT: routine d'affichage d'un caractère à l'écran, appartenant au moniteur.

15. Module TERMINI

entrée: -

pré-condition: -

fonction: Le module TERMINI permet d'initialiser la ligne raccordant le Bigboard au DEC, pour la réception de fichiers. Ceci se fait en envoyant la commande: TERMINAL PAUSE END-OF-PAGE.

sortie: -

post-condition: -

Appelle: STROUT.

16. Module BELL

entrée: -

pré-condition: -

fonction: Le module BELL sert à émettre un "beep" sonore au terminal.

sortie: -

post-condition: -

Appelle: -

17. Module TERMINAL

entrée: -

pré-condition: -

fonction: Le module TERMINAL permet au micro-ordinateur de travailler comme terminal du DEC. Il envoie préalablement la commande TERMINAL PAGE 24, qui permet de travailler avec 24 lignes d'écran, puis ensuite affiche à l'écran le message:

⟨mode TERMINAL: tapez sur ⟨BACKSPACE⟩ pour sortir⟩

Le passage en mode terminal se fait par appel du module TERM.

sortie: -

post-condition: -

Appelle: TERM
CLRSCR
STROUT

18. Module QUERYFILENAME

entrée: - variable booléenne, appelée TODEC, donnant le sens du transfert des fichiers qui va se dérouler.

pré-condition: - TODEC est vraie si le sens du transfert est du Bigboard vers le DEC; elle est fausse si le sens est du DEC vers le Bigboard.

fonction: Le module QUERYFILENAME permet d'interroger l'utilisateur pour savoir quel fichier il veut transférer.
Considérons tout d'abord le cas où le transfert doit se dérouler vers le DEC; le module affiche à l'écran (10e ligne):

DONNEZ LE NOM DU FICHIER SUR LE MICRO:

L'utilisateur répond en donnant le nom du fichier qu'il veut transférer vers le DEC. Le nom s'inscrit dans la case correspondante, à l'écran.
Ensuite, on affiche, au même endroit, la question:

DONNEZ LE NOM DU FICHIER SUR LE DEC:

L'utilisateur peut soit donner un nom, soit taper sur RETURN. Dans ce dernier cas, le nom du fichier transféré sera le même sur le DEC.
Finalement, s'affiche sur la 10e ligne de l'écran, le message:

TAPEZ < CTRL/Z > POUR ARRETER LE TRANSFERT

Dans le cas où le transfert se déroule dans le sens DEC vers micro, la première question qui s'affiche est:

DONNEZ LE NOM DU FICHIER SUR LE DEC:

Le nom de fichier que donne l'utilisateur est alors validé; s'il n'existe pas, le message xx FICHIER INEXISTANT xx s'inscrit à l'écran, dans la case correspondante (cfr chapitre II). S'il existe, le nom est affiché.
La question suivante apparaît alors:

DONNEZ LE NOM DU FICHIER SUR LE MICRO:

En tapant directement sur la touche RETURN, le nom de fichier sera identique à celui du fichier sur le DEC; sinon, un autre nom est donné au fichier.
Pour terminer, le message suivant est affiché:

TAPEZ <CTRL/Z> POUR ARRETER LE TRANSFERT

sortie: - FILENAMEDEC: nom du fichier sur le DEC
- FILENAMEMIC: nom du fichier sur le micro.

post-condition: - FILENAMEMIC et FILENAMEDEC sont des chaînes de caractères (STRING).

Appelle: - BELL
- GOTOXY
- CLREOL
- STROUT
- READPR

19. Module FILESEND

entrée: - FILENAMEMIC: nom du fichier du micro, que l'on va transférer
- FILENAMEDEC: nom du fichier sur le DEC dans lequel le fichier FILENAMEMIC est transféré.

pré-condition: - FILENAMEMIC et FILENAMEDEC sont des chaînes de caractères (STRING)

fonction: Le module FILESEND procède à l'envoi d'un fichier de nom FILENAMEMIC, appartenant au micro, vers un fichier FILENAMEDEC situé sur le DEC, grâce à la commande COPY TTY: ⟨FILENAMEDEC⟩ .
Pendant le transfert, le contenu du fichier est affiché sur les 12 lignes inférieures de l'écran.
Pour arrêter le transfert quand l'utilisateur le désire, il lui suffit de taper CTRL/Z.

sortie: -

post-condition: -

Appelle: - STROUT
- GOTOXY
- GETC
- CHROUT
- CLREOS

20. Module FILERECEIVE

entrée: - FILENAMEDEC: nom du fichier, du DEC, que l'on va transférer
- FILENAMEMIC: nom du fichier, sur le micro, dans lequel le fichier FILENAMEDEC est transféré.

pré-condition: - FILENAMEMIC et FILENAMEDEC sont des chaînes de caractères (STRING).

fonction: Le module FILERECEIVE procède à la réception d'un fichier de nom FILENAMEDEC, appartenant au DEC, vers un fichier FILENAMEMIC, du micro; l'opération se déroule grâce à la commande TYPE ⟨ FILENAMEDEC ⟩ .
Pendant le transfert, le contenu du fichier est affiché sur les 12 lignes inférieures de l'écran.
Pour arrêter le transfert quand l'utilisateur le désire, il lui suffit de taper CTRL/Z.

sortie: -

post-condition: -

Appelle: - GOTOXY
- PAGERD
- CLREOS
- CHROUT

21. Module LOGINDEC

entrée: -

pré-condition: -

fonction: Le module LOGINDEC procède à l'exécution du LOGIN sur le DEC. Il demande les coordonnées (nom d'utilisateur et mot de passe), puis, après envoi d'un CTRL/Z pour appeler l'interpréteur de commandes, il envoie les coordonnées; celles-ci sont validées. Dans le cas où les coordonnées sont incorrectes, on demande des nouvelles coordonnées à l'utilisateur.

sortie: -

post-condition: -

Appelle: - GOTOXY
- BELL
- CLREOL
- CHROUT
- READPR

22. Module TRANSDEC

entrée: -

pré-condition: -

fonction: Le module TRANSDEC permet d'envoyer successivement plusieurs fichiers vers le DEC. Après avoir demandé les noms des fichiers grâce au module QUERYFILENAME, il procède au transfert par le module FILESEND.

Le transfert effectué, le message suivant apparaît sur la ligne 10 de l'écran.

VOULEZ-VOUS EFFECTUER UN NOUVEAU TRANSFERT VERS LE DEC (O/N)

En répondant par "O" (majuscule ou minuscule), on peut effectuer un nouveau transfert. L'appui sur tout autre touche provoque le retour au menu principal.

sortie: -

post-condition: -

Appelle: - QUERYFILENAME
- FILESEND
- BELL
- GOTOXY
- CLREOL

23. Module LOGOUT

entrée: -

pré-condition: -

fonction: Le module LOGOUT permet d'envoyer la commande LOGOUT au DEC afin de libérer la ligne.

sortie: -

post-condition: -

Appelle: STROUT
CHROUT

24. Module BBDEC (programme principal)

entrée: -

pré-condition: -

fonction: Le module BBDEC a pour but d'effectuer les opérations d'initialisation de la communication entre le micro et le DEC, d'afficher le menu principal, et de demander quelle option l'utilisateur désire prendre.

sortie; -

post-condition: -

Appelle: - CLRSCR
- DECINI
- LOGINDEC
- TERMINI
- BLANK
- GOTOXY
- ENDBL
- TERMINAL
- TRANSMIC
- LOGOUT

25. MODULE TRANSMIC

entrée: -

pré-condition: -

fonction: Le module TRANSMIC permet de recevoir successivement plusieurs fichiers, en provenance du DEC. Après avoir demandé les noms des fichiers, grâce au module QUERYFILENAME, TRANSMIC procède au transfert par le module FILERECEIVE.
Le transfert terminé, le message suivant apparaît sur la ligne 10 de l'écran.
VOULEZ-VOUS EFFECTUER UN NOUVEAU TRANSFERT VERS LE MICRO (O/N)?

En répondant par "O" (majuscule ou minuscule), on peut effectuer un nouveau transfert.
L'appui sur toute autre touche provoque le retour au menu principal.

sortie: -

post-condition: -

Appelle: - QUERYFILENAME

- FILERECEIVE

- BELL

- GOTOXY

- CLREOL

ANNEXE III. LISTE DES PROGRAMMES

A.III.1. Liste des routines d'échanges de données.

```
;-----------------------------------------------------------------------
;
;                        routine AFFMES
;
;   entree : - ERROR : code erreur utilise par les routines GPBINI,MXSND,
;                      DATSND,DATRCV
;
;   sortie : ---
;
;   fonction :
;              La routine AFFMES affiche un message selon la valeur du
;              code erreur renvoye par les routines GPBINI,MXSND,DATSND,
;              DATRCV.Les messages sont enonces ci-dessous (MSG1,MSG2,MSG3,
;              MSG4,MSG5,MSG6)
;
;   assemblee avec M80
;   link avec programme FORTRAN ,grace a L80 (switches :/N/E)
;
;-----------------------------------------------------------------------

        .z80
        dseg

        error:  defs    2       ;pointeur vers parametre

;table des messages

        msg1:   defb    '** communication OK **                         $'
        msg2:   defb    '** listener not ready to receive data **       $'
        msg3:   defb    '** data not accepted by listener **            $'
        msg4:   defb    '** data received not validated by talker **$'
        msg5:   defb    '** data not unvalidated by talker **           $'
        msg6:   defb    '** device not present **                       $'

        cr      equ     0dh     ;carriage return
        lf      equ     0ah     ;line feed
        bdos    equ     05h     ;point d'entree CPM
        print   equ     09h     ;impression chaine de CPM
        conout  equ     02h     ;impression caractere de CPM
        length  equ     43      ;longueur message
        normes  equ     5       ;valeur max de ERROR

;point d'entree de AFFMES

        entry   affmes

;sauvetage des registres

affmes:

        push    af
        push    bc
        push    de
        push    hl
        push    ix
        push    iy
        ld      (error),hl

;calcul de l'adresse du message
```

```
        ld      hl,(error)
        ld      a,(hl)          ;place la valeur de ERROR dans A
        cp      0
        ld      c,0
        jr      z,lprint
        ld      b,a             ;ERROR dans B
        ld      a,0
bcl:
        add     a,length
        djnz    bcl
        ld      c,a             ;offset dans C
lprint:
        ld      b,0
        ld      hl,msg1
        add     hl,bc
        push    hl
        pop     de              ;adresse message dans DE
        ld      c,print
        call    bdos            ;impression message
        ld      c,conout
        ld      e,cr
        call    bdos
        ld      c,conout
        ld      e,lf
        call    bdos

end:
        pop     iy
        pop     ix
        pop     hl
        pop     de
        pop     bc
        pop     af
        ret
        end
```

```
;-----------------------------------------------------------------------
;
;                          routine DATRCV
;
;   entree : - TALKER : adresse du convertisseur A/D
;                       declaree en INTEGER*1
;            - LENGTH : nombre de donnees a recevoir du A/D
;                       declaree en INTEGER
;   sortie : - ARRAY : tableau de donnees envoyees par le A/D
;                      declare en REAL
;            - LENGTH : nombre de donnees restant a recevoir
;                       declare en INTEGER
;            - ERROR : code erreur
;                      declare en INTEGER*1
;
;   fonction :
;              La routine DATRCV procede a la reception de LENGTH donnees
;              envoyees par le convertisseur A/D,d'adresse TALKER ,et stocke
;              ces donnees dans un tableau ARRAY ;si une erreur se produit
;              pendant l'echange ,ERROR est positionne a une valeur non nulle;
;              LENGTH donne alors le nombre de donnees restant a recevoir.
;
;   assemblee avec M80
;   link avec programme FORTRAN ,grace a L80 (switches :/N/E)
;
;-----------------------------------------------------------------------

        .z80
        dseg

        ext     bytsnd,adress,outpor
        ext     $ca,$d9,$ma,$aa,$t1,$da  ;routines arithmetiques de FORLIB

        pioda   equ     08h
        piodb   equ     0ah
        pioca   equ     09h
        piocb   equ     0bh

        ptr:    defs    2
        talker: defs    2
        array:  defs    2
        length: defs    2
        error:  defs    2
        lsbyte: defs    1
        msbyte: defs    1

        typdat  equ     4           ;longueur des donnees du tableau ARRAY
        atnmsk  equ     10111111b

;point d'entree de DATRCV

        entry datrcv

;sauvetage des registres dans un bloc de donnees local a DATRCV

datrcv:
        push    af
        push    ix
        push    iy
        ld      (talker),hl
```

```
        ld      (array),de
        ld      (ptr),bc

;stockage de l'adresse de ERROR et de LENGTH

        ld      ix,(ptr)
        ld      c,(ix+0)
        ld      b,(ix+1)
        ld      (length),bc
        ld      c,(ix+2)
        ld      b,(ix+3)
        ld      (error),bc

;configuration port A en sortie

        call    outpor

;adressage du talker par appel du module ADRESS(l'adresse est passee par
;l'accumulateur

        ld      bc,(talker)         ;adresse le talker
        ld      a,(bc)
        ld      hl,(error)
        call    adress
        ld      a,0
        ld      hl,(error)
        cp      (hl)
        jr      nz,end

;boucle principale pour echange des donnees
;IX sert de pointeur vers LENGTH
;LENGTH est decremente a chaque execution de la boucle

        ld      ix,(length)

;si LENGTH = 0,alors on sort de la boucle et du programme
;on compare les deux bytes de LENGTH

        ld      a,0
        cp      (ix+1)
        jr      nz,trait
        cp      (ix+0)
        jr      z,end

;programme port A en entree de donnees

        call    inpor

;reception du byte de poids faible
;stockage du byte dans LSBYTE

trait:
        ld      hl,(error)
        call    bytrcv
        ld      (msbyte),a

;teste si la reception du byte s'est deroulee sans erreur
;si ERROR<>0,alors on quitte DATRCV

        ld      a,0
```

```
        ld      hl,(error)
        cp      (hl)
        jr      nz,end

;reception du byte de poids fort

        ld      hl,(error)
        call    bytrcv
        ld      (lsbyte),a

;teste si la reception du byte s'est deroulee sans erreur
;si ERROR<>0,alors on quitte DATRCV

        ld      a,0
        ld      hl,(error)
        cp      (hl)
        jr      nz,end

;deformatage des donnees recues

        ld      a,(lsbyte)
        ld      h,a
        ld      a,(msbyte)
        ld      l,a
        call    datdfr

;le resultat de la conversion est dans HL
;on convertit HL en un nombre reel a stocker dans ARRAY

        ld      bc,(array)
        call    datdcv

;le resultat de la conversion est un nombre reel (4 octets)
;on passe a la donnee suivante

        ld      hl,(array)
        ld      bc,typdat
        add     hl,bc
        ld      (array),hl

;a-t'on encore de donnees a envoyer

        ld      a,0
        ld      c,(ix+0)
        ld      b,(ix+1)
        dec     bc
        cp      c
        jr      nz,stock
        cp      b
        jr      z,etiq
stock:
        ld      (ix+0),c
        ld      (ix+1),b
        jr      trait

;fin de DATRCV,restauration des registres

etiq:
        ld      (ix+0),c
        ld      (ix+1),b
```

```
end:
        pop     iy
        pop     ix
        pop     af
        ret

;-------------------------------------------------------------------------
;       routine DATDCV
;       on passe la donnee a convertir par HL
;       le nombre converti est stocke a l'adresse donnee par PTRARR
;-------------------------------------------------------------------------

        ten     equ     10
        one     equ     1
        minus1  equ     -1
        range   equ     2047    ;2 ^ 11 -1

        ptrarr: defs    2

;conversion de l'entier en reel

        entry   datdcv

datdcv:

        push    hl
        ld      (ptrarr),bc
        call    $ca

;on divise le nombre a convertir par 2047

        ld      hl,range
        call    $da

;on multiplie le resultat par -1

        ld      hl,minus1
        call    $ma

;on ajoute 1

        ld      hl,one
        call    $aa

;on multiplie par 10

        ld      hl,ten
        call    $ma

;on stocke l'accumulateur flottant dans ARRAY

        ld      hl,(ptrarr)
        call    $ti
        pop     hl

        ret

;-------------------------------------------------------------------------
;       routine DATDFR
;       formatte les donnees envoyees par le A/D
```

```
;-----------------------------------------------------------------------

        lsb:    defs    1
        msb:    defs    1

        entry   datdfr

datdfr:

        push    af
        push    bc
        push    de

        ld      a,h
        ld      (lsb),a
        ld      a,l
        ld      (msb),a

        ld      a,(lsb)
        srl     a
        srl     a
        srl     a
        srl     a
        ld      (lsb),a

        ld      a,(msb)
        sla     a
        sla     a
        sla     a
        sla     a
        ld      b,a
        ld      a,(lsb)
        add     a,b
        ld      l,a

        ld      a,(msb)
        srl     a
        srl     a
        srl     a
        srl     a
        ld      h,a

;restauration des registres

        pop     de
        pop     bc
        pop     af

        ret

;=======================================================================
;
;                       routine BYTRCV
;             envoie du byte passe par l'accumulateur
;
;=======================================================================

        cptr    equ     3000    ;compteur pour delai 100 msec
        mskfdl  equ     11110111b
```

```
mnrfdh  equ     00001000b
mndach  equ     00010000b
mndacl  equ     11101111b
minit   equ     11100111b
davin   equ     0               ;no ligne DAV in

char:   defs    1               ;zone sauvetage caractere
err:    defs    2               ;adresse du parametre d'erreur

entry   bytrcv

bytrcv:

        ld      (err),hl
        push    ix
        push    iy
        push    bc
        push    de
        push    hl
        push    af

;mettre NRFD et NDAC bas

        in      a,(piodb)
        and     minit
        out     (piodb),a

;mise a zero de ERROR (pas d'erreur)

        ld      hl,(err)
        ld      a,0
        ld      (hl),a

;mettre NRFD (en sortie) a 1 = faux

        in      a,(piodb)
        or      mnrfdh
        out     (piodb),a

;attendre que le talker signale que les donnees sont disponibles

        ld      bc,cptr
dav:
        in      a,(piodb)
        bit     davin,a
        jr      z,lbl1
        ld      a,0
        dec     bc
        cp      b               ;teste le lsb du compteur
        jr      nz,dav
        cp      c               ;teste le msb du compteur
        jr      nz,dav
        ld      hl,(err)        ;si DAV bas apres 100 msec,ERROR=3
        ld      a,3
        ld      (hl),a
        jr      endrcv          ;et on sort de BYTRCV

;placer NRFD (en entree) bas = vrai

lbl1:
```

```
        in      a,(piodb)
        and     mnrfdl
        out     (piodb),a

;lire la donnee sur le port A et la stocker dans CHAR

        in      a,(pioda)
        ld      (char),a

;placer NDAC a 1 = faux

        in      a,(piodb)
        or      mndach
        out     (piodb),a

;DAV est-il passe haut?

        ld      bc,cptr
ndav:
        in      a,(piodb)
        bit     davin,a
        jr      nz,lbl2
        ld      a,0
        dec     bc
        cp      c               ;teste le lsb du compteur
        jr      nz,ndav
        cp      b               ;tests le msb du compteur
        jr      nz,ndav
        ld      hl,(err)        ;si DAV bas apres 100 msec
        ld      a,4             ;alors ERREUR=4
        ld      (hl),a          ;et on sort de bytrcv
        jr      endrcv

;placer NDAC bas = vrai

lbl2:
        in      a,(piodb)
        and     mndacl
        out     (piodb),a

;fin de la routine

endrcv:
        pop     af
        pop     hl
        pop     de
        pop     bc
        pop     iy
        pop     ix
        ld      a,(char) ;passe le byte recu par l'accumulateur
        ret

;-------------------------------------------------------------------------
;       module INPOR
;       programme le port A en entree
;-------------------------------------------------------------------------

        entry   inpor

inpor:
```

```
        push    af
;positionner port A en entree de donnees
        ld      a,0ffh
        out     (pioca),a
        out     (pioca),a
;positionner buffer en entree
        in      a,(piodb)
        or      080h
        out     (piodb),a

        pop     af
        ret

        END
```

```
;-------------------------------------------------------------------------
;
;                          routine DATSND
;
;  entree : - LSTNR : adresse du convertisseur D/A
;                     declare en INTEGER*1
;           - ARRAY : tableau de donnees a envoyer au D/A
;                     declare en REAL
;           - LENGTH : nombre de donnees a envoyer
;                      declare en INTEGER
;  sortie : - ERROR : code erreur
;                     declare en INTEGER*1
;           - LENGTH : nombre de donnees restant a envoyer
;                      declare en INTEGER
;
;  fonction :
;             La routine DATSND procede a l'envoi d'un tableau de LENGTH
;             donnees ARRAY ,au convertisseur D/A d'adresse LSTNR ;si une
;             erreur se produit pendant l'echange ,ERROR est mis a une
;             valeur non nulle.LENGTH est decremente a chaque envoi d'une
;             donnees :il donne  ainsi a la sortie de DATSND ,le nombre
;             de donnees non envoyees.
;
;  assemblee avec M80
;  link avec programmes FORTRAN grace a L80 (switches :/N/E)
;
;-------------------------------------------------------------------------

        .Z80
        dseg

        ext     $ac,$da,$ma,$aa,$ch,$11,$t1  ;routines math de FORLIB
        ext     adress,bytsnd,outpor

        pioda   equ     08h
        piodb   equ     0ah
        pioca   equ     09h
        piocb   equ     0bh

        ptr:    defs    2
        lstnr:  defs    2
        array:  defs    2
        length: defs    2
        error:  defs    2
        lsbyte: defs    1
        msbyte: defs    1
        typdat  equ     4;longueur des donnees du tableau ARRAY
        atnmsk  equ     10111111b

;point d'entree de la routine

        entry   datsnd

;sauvetage des registres:les registres BC,DE et HL sont sauves dans un bloc
;de donnees local

datsnd:

        push    af
        push    ix
```

```
        push    iy
        ld      (lstnr),hl
        ld      (array),de
        ld      (ptr),bc

;stockage de l'adresse de ERROR et de LENGTH

        ld      ix,(ptr)
        ld      c,(ix+0)
        ld      b,(ix+1)
        ld      (length),bc
        ld      c,(ix+2)
        ld      b,(ix+3)
        ld      (error),bc

;adressage du listener par appel de ADRESS (adresse passee par accumulateur)
;on envoie seulement le LSB de LSTNR

        ld      bc,(lstnr)
        ld      a,(bc)
        ld      hl,(error)
        call    adress
        ld      a,0
        ld      hl,(error)
        cp      (hl)
        jr      nz,end

;boucle principale pour echange des donnees
;IX sert de pointeur vers LENGTH
; LENGTH est decremente a chaque execution de la boucle
;ceci pour,apres execution de DATSND,savoir si toutes les donnees ont
;ete envoyees ou combien n'ont pas ete envoyees

        ld      ix,(length)

;si LENGTH=0 alors,on sort de la routine DATSND(on compare les deux bytes
;de la variable LENGTH

        ld      a,0
        cp      (ix+1)
        jr      nz,trait
        cp      (ix+0)
        jr      z,end

;passage de l'adresse(de l'element a envoyer)a la routine de conversion

trait:
        ld      hl,(array)
        call    datcvr

;le resultat de la conversion est un entier de 2 octets stockes dans HL
;formatage de la donnee

        call    datfor
        ld      a,h
        ld      (lsbyte),a
        ld      a,l
        ld      (msbyte),a

;envoi du byte de poids faible
```

```
        ld      a,(lsbyte)
        ld      hl,(error)
        call    bytsnd

;teste de la reussite de l'envoi du byte de poids faible

        ld      a,0
        ld      hl,(error)
        cp      (hl)
        jr      nz,end

;envoi du byte de poids fort

        ld      a,(msbyte)
        ld      hl,(error)
        call    bytsnd

;test de la reussite de l'envoi du byte de poids fort

        ld      a,0
        ld      hl,(error)
        cp      (hl)
        jr      nz,end

;on passe a la donnee suivante en incrementant de 4 le contenu de ARRAY

        ld      hl,(array)
        ld      bc,typdat
        add     hl,bc
        ld      (array),hl

;a-t'on encore des donnees a envoyer?
;on transfere LENGTH dans BC ,on le decremente puis on le compare byte par byte
;ensuite,on le stocke a la place qu'il occupe en memoire

        ld      a,0
        ld      c,(ix+0)
        ld      b,(ix+1)
        dec     bc
        cp      c
        jr      nz,stock
        cp      b
        jr      z,etiq
stock:
        ld      (ix+0),c
        ld      (ix+1),b
        jr      trait

;fin de la routine,restauration des registres
etiq:
        ld      (ix+0),c
        ld      (ix+1),b
end:
        pop     iy
        pop     ix
        pop     af
        ret
```

```
;-----------------------------------------------------------------------
;       routine DATFOR:formatte les donnees pour le D/A
;       donnee passee par HL
;-----------------------------------------------------------------------

        lbyte:  defs    1
        rbyte:  defs    1
        lsb:    defs    1
        msb:    defs    1
        octet:  defs    1

        entry   datfor

datfor:

        push    af
        push    bc
        push    de

        ld      a,h
        ld      (lbyte),a
        ld      a,l
        ld      (rbyte),a

;msbyte := lbyte shl 4

        ld      a,(lbyte)
        ld      b,4
sl1:
        sla     a
        djnz    sl1

;msbyte := msbyte shr 2

        srl     a
        srl     a

;msbyte := msbyte + 40h

        add     a,40h
        ld      (msb),a

;octet := rbyte shr 6

        ld      b,6
        ld      a,(rbyte)
sr1:
        srl     a
        djnz    sr1

;msbyte := msbyte + octet

        ld      b,a
        ld      a,(msb)
        add     a,b
        ld      (msb),a

;rbyte := rbyte shl 2

        ld      a,(rbyte)
```

```
        sla     a
        sla     a
;rbyte := rbyte shr 2
        srl     a
        srl     a
;rbyte := rbyte + 80h
        add     a,80h
;lsbyte := rbyte
        ld      (lsb),a
;fin routine DATFDR
        ld      a,(lsb)
        ld      h,a
        ld      a,(msb)
        ld      l,a
        pop     de
        pop     bc
        pop     af
        ret
;-----------------------------------------------------------------------
;       routine  DATCVR
;       on passe a DATCVR l'adresse du nombre a convertir par HL
;-----------------------------------------------------------------------
ten     equ     10      ;l'echelle va de -10 a +10
one     equ     1
minus1  equ     -1
range   equ     2047    ; 2 ^ 11 - 1
;on charge l'accumulateur flottant avec le nombre a convertir
        entry   datcvr
datcvr:
        call    $l1
;on charge la deuxieme operande dans le registre HL
        ld      hl,ten
;division reel par entier
        call    $da
;changer le signe du resultat
        ld      hl,minus1
        call    $ma
;ajouter 1
```

```
        ld      hl,one
        call    $aa

;multiplier le tout par 2047

        ld      hl,range
        call    $ma

;prendre la partie entiere du resultat:le resultat final(entier sur 2 octet)
;est stocke dans le registre HL

        call    $ch
        ret

        END     ;FIN D'ASSEMBLAGE
```

```
;-------------------------------------------------------------------------
;
;                        ROUTINE GPBINI
;
;  entree : ---
;  sortie : ERROR entier declare en integer*1
;
;  fonction  : - configuration du PIO B en mode controle
;              - initialisation du bus
;              - envoi de UNT et de UNL
;
;  remarque :
;              - Le raccordement du bus au PIO B est :
;
;                bit 7 ==> data direction
;                bit 6 ==> atn
;                bit 5 ==> ndac in
;                bit 4 ==> ndac out
;                bit 3 ==> nrfd out
;                bit 2 ==> nrfd in
;                bit 1 ==> dav out
;                bit 0 ==> dav in
;
;   routine assemblee avec M80
;   link avec programme ecrit en FORTRAN,grace a L80 (switches:/N/E)
;
;-------------------------------------------------------------------------

                .z80
                dseg

        outdat  equ     0fh             ;code PIO pour sortie de donnees
        ctrl    equ     11001111b       ;code PIO pour controle port B
        inout   equ     00100101b       ;mot de controle port B
        atnmsk  equ     10111111b       ;masque pour activer ATN
        noatn   equ     6               ;numero de la ligne ATN (bit 7=poids fort)
        outmsk  equ     7fh             ;masque buffer sortie
        pioca   equ     09h             ;adr registre controle du port A
        piocb   equ     0bh             ;adr registre controle du port B
        pioda   equ     08h             ;adr registre donnee du port A
        piodb   equ     0ah             ;adr registre donnee du port B
        unl     equ     063             ;commande UNL
        unt     equ     095             ;commande UNT
        error:  defs    2               ;pointeur vers parametre d'erreur

;point d'entree de la routine et sauvetage des registres

                entry   gpbini
      gpbini:
                push    af
                push    bc
                push    de
                push    hl
                push    ix
                push    iy
                ld      (error),hl

                call    piopgm  ;programme le port B
```

```
                call    inhib   ;inhibition des listeners et talkers
;restauration des registres
                pop     iy
                pop     ix
                pop     hl
                pop     de
                pop     bc
                pop     af
;fin de GPBINI
                ret
;-------------------------------------------------------------------------
;               module INHIB
;               inhibition des listeners par UNL
;-------------------------------------------------------------------------
                entry   inhib
        inhib:
                push    af
                in      a,(piodb)
                and     atnmsk          ;place ATN vrai(= 0)
                out     (piodb),a
                call    outpor          ;port A en mode sortie
                ld      a,unl           ;envoi de UNL
                ld      hl,(error)
                call    bytsnd
                ld      a,0
                ld      hl,(error)
                cp      (hl)
                jr      nz,stop         ;si erreur,on sort
                ld      a,unt           ;envoi de UNT
                ld      hl,(error)
                call    bytsnd
         stop:
                pop     af
                ret
;-------------------------------------------------------------------------
;               module OUTPOR
;               bascule les buffers en sortie
;                programme port A en sortie
;-------------------------------------------------------------------------
                entry   outpor
        outpor:
                push    af
                in      a,(piodb)       ;faire basculer les buffers
                and     outmsk          ;en mode sortie
                out     (piodb),a

                ld      a,ctrl          ;programme port A en
                out     (pioca),a       ;mode sortie
                ld      a,0
                out     (pioca),a
```

```
                pop     af

                ret

;-------------------------------------------------------------------------------
;               module PIOPGM:programmation du PIO
;               programmation du port B en mode controle
;               positionner port A en entree
;-------------------------------------------------------------------------------

        piopgm:

                push    af
                ld      a,07h           ;desactive les interruptions
                out     (piocb),a
                out     (pioca),a
                ld      a,ctrl
                out     (piocb),a

;definition des lignes d'entree et de sortie pour le port B

                ld      a,inout
                out     (piocb),a

;place tous les signaux a faux

                ld      a,0ffh
                out     (piodb),a
                pop     af

                ret


;-------------------------------------------------------------------------------
;
;                       routine BYTSND
;
;  entree : - byte a envoyer a passer par l'accumulateur A
;           - adresse de ERROR passe par HL
;  sortie : ---
;
;  fonction :
;             -  La routine BYTSND procede a l'envoi d'un byte sur le bus
;                raccorde au PIO ,conformement au protocole IEEE-488.
;                En cas d'erreur ,ERROR est positionne a une valeur <> 0
;
;-------------------------------------------------------------------------------

        cptr    equ     3000    ;compteur pour delai 100 msec
        nrfdin  equ     2       ;no ligne NRFD IN
        ndavh   equ     00000010b
        ndavl   equ     11111101b
        ndacin  equ     5       ;no ligne NDAC IN
        char:   defs    1       ;mot reserve pour stocker le caractere
        err:    defs    2       ;pointeur vers ERROR de routine appelante

;point d'entree de la routine BYTSND
;on sauve le caractere a l'adresse char
;on sauve tous les registres
```

```
                entry   bytsnd
     bytsnd:

                ld      (err),hl
                ld      (char),a
                push    ix
                push    iy
                push    bc
                push    de
                push    hl
                push    af

;on initialise ERROR a 0

                ld      hl,(err)        ;initialisation de ERROR a 0
                ld      a,0
                ld      (hl),a

;la donnee est indisponible
;on place DAV haut = 1

                in      a,(piodb)       ;lit le port B
                or      mdavh           ;on met DAV haut
                out     (piodb),a

;tester NRFD haut

                in      a,(piodb)
                bit     nrfdin,a
                jr      z,cont

;tester NDAC haut

                bit     ndacin,a
                jr      z,cont
                ld      a,5             ;NRDF + NDAC haut =>device not present
                ld      hl,(err)
                ld      (hl),a
                jr      endsnd

;placer les donnees sur le port A

     cont:
                ld      a,(char)
                out     (pioda),a


;tester si NRFD est haut
;si,apres 100 msec de delai,NRFD est toujours bas,il y a erreur de type 1

                ld      bc,cptr         ;initialise le compteur
      nrfd:
                in      a,(piodb)       ;lit le port B
                bit     nrfdin,a        ;teste l'etat de NRFD
                jr      nz,lbli         ;si NRFD haut,on continue
                dec     bc              ;decremente le compteur
                ld      a,0             ;et teste s'il est nul
                cp      b               ;si cptr <>0,on teste nrfd
                jr      nz,nrfd
                cp      c
```

```
                jr      nz,nrfd
                ld      hl,(err)        ;sinon,positionner ERROR a 1
                ld      a,1
                ld      (hl),a
                jr      endsnd

;avertir que les donnees sont valides
;en placant DAV bas = 0

        lbl1:
                in      a,(piodb)
                and     mdavl
                out     (piodb),a

;attendre que la donnee soit acceptee
;si pas acceptee dans un delai de 100 msec ==> ERROR = 2

                ld      bc,cotr
        ndac:
                in      a,(piodb)
                bit     ndacin,a
                jr      nz,lbl3
                dec     bc
                ld      a,0
                cp      b
                jr      nz,ndac
                cp      c
                jr      nz,ndac
                ld      hl,(err)
                ld      a,2
                ld      (hl),a
                jr      endsnd

;les donnees ne sont plus valides ==> placer DAV haut

        lbl3:
                in      a,(piodb)
                or      mdavh
                out     (piodb),a

;fin de la routine

        endsnd:
                pop     af
                pop     hl
                pop     de
                pop     bc
                pop     iy
                pop     ix
                ret

                END     ;FIN D'ASSEMBLAGE
```

```
;-----------------------------------------------------------------------
;
;                   routine MXSND
;
;  entree : - LSTNR : adresse du multiplexeur
;                     declare en integer*1
;           - BYTE :  donnee de configuration du multiplexeur
;                     declare en integer*1
;
;  sortie : - ERROR : code erreur
;                     declare en integer*1
;
;  fonction :
;            - La routine MXSND procede a la configuration du multiplexeur
;              d'adresse LSTNR ,en lui envoyant la variable BYTE ;si une
;              erreur se produit pendant la transmission ,ERROR est
;              positionne a une valeur non nulle
;
;  assemblee avec M80
;  link avec programmes ecrits en FORTRAN ,grace a L80 (switches:/N/E)
;
;-----------------------------------------------------------------------

                .z80
                dseg

        ext     bytsnd,outpor
        atnmsk  equ     10111111b
        inmsk   equ     80h
        matn    equ     01000000b
        pioca   equ     09h
        piocb   equ     0bh
        pioda   equ     08h
        piodb   equ     0ah

        lstnr:  defs    2       ;pointeur vers le parametre 1
        byte:   defs    2       ;pointeur vers le parametre 2
        error:  defs    2       ;pointeur vers le parametre 3

;point d'entree de la routine
;sauvetage des adresses des parametres dans un bloc local a la routine
;sauvetage des registres

                entry   mxsnd
        mxsnd:
                ld      (lstnr),hl
                ld      (byte),de
                ld      (error),bc
                push    ix
                push    iy
                push    af

;configuration port A en sortie

                call    outpor

;appel de la routine d'adressage
;on adresse le multiplexeur en lui envoyant le byte de poids faible de la
;variable LSTNR
```

```
                ld      hl,(lstnr)
                ld      a,(hl)
                ld      hl,(error)
                call    adress
                ld      a,0
                ld      hl,(error)
                cp      (hl)
                jr      nz,end

;appel de la routine de programmation du multiplexeur

                ld      hl,(error)
                call    mxpgm

;fin de MXSND
;on restaure les registres

          end:
                pop     af
                pop     iy
                pop     ix
                ret

;-------------------------------------------------------------------------------
;               routine MXPGM
;               programmation du multiplexeur
;-------------------------------------------------------------------------------

                err:    defs    2
          mxpgm:
                ld      (err),hl
                ld      hl,(byte)       ;passe le byte a envoyer par accu.
                ld      a,(hl)          ;on n'envoie que le LSB de BYTE
                ld      hl,(err)
                call    bytsnd          ;envoi du byte
                ret

;-------------------------------------------------------------------------------
;               routine ADRESS
;               adresse le listener/talker sur le bus
;               on passe l'adresse par l'accumulateur
;-------------------------------------------------------------------------------

;point d'entree de la routine ADRESS

                entry   adress

          adress:

                push    af
                push    af
                in      a,(piodb)
                and     atnmsk
                out     (piodb),a       ;place ATN vrai
                pop     af
                call    bytsnd
                in      a,(piodb)       ;lit port B
                or      matn
                out     (piodb),a       ;place ATN haut(faux = 1)
                pop     af
```

```
ret
END                    ;FIN D'ASSEMBLAGE
```

A.III.2. Liste du programme de connexion avec le DEC-20

```
program BBDEC;

const   prompt='a';

type block = array[1..2048] of char;

var
     x,y        : integer;
     loggedout  : boolean;
     ch,reponse    : char;
     str        : string;

(*  ++++++++++++     procedures ecrites en ml Z-80     ++++++++++++++++++  *)

procedure TERM;extern;(* transforme le BBOARD en terminal *)
procedure DECINI;extern;(* initialise la ligne et SIO *)
procedure STROUT(var str : string);extern;(*envoi  string sur SIO + lit echo*)
procedure CHROUT(var str : string);extern;(*envoi string sans lire echo *)
procedure PAGERD(var buffer : block;var size : integer;var endofile:boolean);
                                                                    extern;
procedure HOME;extern;(* curseur coin sup gauche de l'ecran *)
procedure GOTOXY(x,y: integer);extern;(* adressage curseur *)
procedure CLREOL;extern;(* efface jusque fin ligne *)
procedure CLRSCR;extern;(* efface l'ecran *)
procedure CLREOS;extern;(* efface jusque fin ecran *)
procedure BLANK;extern;(* video inverse *)
procedure ENDBL;extern;(* fin video inverse *)
procedure READPR(var notok : boolean);extern;(* lit car. sur SIO => @ *)
procedure GETC(var eot : boolean);extern;(*lit clavier,eot true si ctrl/z*)

(*-----------------------------------------------------------------------*)

procedure TERMINI;
(* initialise terminal *)
begin

    str := 'ter speed 2400';
    strout(str);
    str := chr(10);
    strout(str);
    str := 'ter  pau end';
    strout(str);
    str := chr(10);
    strout(str);

end;

procedure BELL;
(* beep sonore *)

begin
    write(chr(07h));
end; (* bell *)

procedure TERMINAL;
(* transforme le micro en terminal *)

begin

    str := 'ter page 24';
```

```
    strout(str);
    str := chr(10);
    strout(str);
    clrscr;
    writeln('< mode TERMINAL:tapez sur <BACKSPACE> pour sortir >');
    term;

end;

procedure QUERYFILENAME(var filenamedec:string;var filenamemic:string;var todec:boolean);
(* demande les noms des fichiers a transferer *)
(* sortie : filenamedec : nom du fichier sur le DEC *)
(*          filenamemic : nom du fichier sur le micro *)
(* entree : todec = true si transfert du BB vers DEC *)
(*                  false si transfert du DEC vers BB *)

var notok : boolean;

begin
if (todec) then
begin

  filenamemic:='';
  repeat

    bell;
    gotoxy(1,10);write(' DONNEZ LE NOM DU FICHIER SUR LE MICRO:');
    clreol;
    gotoxy(77,10);write('*');
    gotoxy(40,10);readln(filenamemic);

  until (filenamemic <> '');(*attendre un nom de fichier <> vide*)
  gotoxy(50,8);clreol;
  gotoxy(77,8);write('*');
  gotoxy(50,8);write(filenamemic);
  bell;
  gotoxy(1,10);clreol;
  gotoxy(77,10);write('*');
  gotoxy(1,10);write(' DONNEZ LE NOM DU FICHIER SUR LE DEC:');
  readln(filenamedec);

  if (filenamedec = '') then
     begin
       if (filenamemic[2] = ':') then
          filenamedec:=copy(filenamemic,3,(length(filenamemic)-2))
       else filenamedec:=filenamemic;
     end;(* if filenamedec='' *)

  gotoxy(48,4);clreol;
  gotoxy(77,4);write('*');
  gotoxy(50,4);write(filenamedec);
  bell;
  gotoxy(1,10);write('TAPEZ <ctrl/Z> POUR ARRETER LE TRANSFERT');clreol;
  gotoxy(77,10);write('*');

end

else

begin
```

```
  filenamedec := '';
  repeat

  repeat

    bell;
    gotoxy(1,10);write(' DONNEZ LE NOM DU FICHIER SUR LE DEC:');clreol;
    gotoxy(77,10);write('*');
    gotoxy(38,10);readln(filenamedec);

  until (filenamedec <> '');

  str := 'dir ';strout(str);
  str:=filenamedec;strout(str);
  str := chr(10);strout(str);
  notok := false;
  readpr(notok);

  if (notok) then
    begin
      bell;
      gotoxy(48,4);write('**FICHIER INEXISTANT**');
    end;

  until(filenamedec <>'') and (not(notok));

  gotoxy(48,4);clreol;
  gotoxy(77,4);write('*');
  gotoxy(50,4);write(filenamedec);
  bell;
  gotoxy(1,10);clreol;
  gotoxy(77,10);write('*');
  gotoxy(1,10);write(' DONNEZ LE NOM DU FICHIER SUR LE MICRO:');
  readln(filenamemic);

  if (filenamemic = '') then filenamemic := filenamedec;

  gotoxy(1,10);clreol;
  gotoxy(77,10);write('*');
  gotoxy(50,8);clreol;
  gotoxy(77,8);write('*');
  gotoxy(50,8);write(filenamemic);
  bell;
  gotoxy(1,10);write('TAPEZ <ctrl/Z> POUR ARRETER LE TRANSFERT');clreol;
  gotoxy(77,10);write('*');

end;(*else *)

end;(* queryfilename *)


procedure FILESEND(var fnamemic:string;var fnamedec:string);
(* envoi d'un fichier de nom FNAMEMIC vers un fichier FNAMEDEC sur DEC  *)
(* entree : FNAMEMIC,FNAMEDEC :noms des fichiers concernes par le transfert*)
(* attend l'echo de chaque caractere envoye *)

var
      i,j,line: integer;
      eot     : boolean;
```

```
        ch     : char;
        buffer : block;
        fileout: file of char;


begin

        (* initialise le DEC pour recevoir un fichier *)
        str := 'ter page 12 ';strout(str);
        str := chr(10);strout(str);
        str := 'copy tty: ';strout(str);
        str := fnamedec;strout(str);
        str := chr(10);strout(str);

        reset(fileout,fnamemic,binary,1024);

        read(fileout;ch);
        gotoxy(0,12);clreos;
        line := 0;
        while (ch <> chr(1ah)) and (not eot) do (* ctrl/Z = fin fichier *)
           begin
             str := ch;
             if (ch = chr(10)) then line := line + 1;
             if (line = 11) then
                begin
                  line := 0;
                  gotoxy(0,12);
                  clreos;
                end;
             if (line <> 11) or (ch <> chr(10)) then write(ch);(* avoid last LF*)
             (* ne pas envoyer les crtl/J au dec *)
             if (ch <> chr(10)) then strout(str);
             read(fileout;ch);
             getc(eot);
           end;
         str := chr(26); (* arrete le COPY et ferme le fichier sur DEC *)
         chrout(str);
         close(fileout);

end;(* filesend *)


procedure FILERECEIVE(var fnamemic:string;var fnamedec:string);
(* reception d'un fichier de nom FNAMEDEC et stockage de ce fichier dans *)
(* FNAMEMIC *)
(* entree : FNAMEMIC,FNAMEDEC noms des fichiers concernes par le transfert *)


var
        i,j,size: integer;
        endofile : boolean;
        ch     : char;
        buffer : block;
        filein : file of char;

begin

        str := 'ter page 12 ';strout(str);
        str := chr(10);strout(str);
```

```
        rewrite(filein,fnamemic,binary,1024);

        endofile := false;
        str := 'type ';strout(str);
        str := fnamedec;strout(str);
        str := chr(10);strout(str);

        repeat

             gotoxy(0,12);clreos;
             pagerd(buffer,size,endofile);
             str := chr(19);chrout(str);
             if (endofile) then size := size-1;(* avoid last char = @ *)
             for i:=4 to size-1 do(* dec send garbage : cursor nbme,... *)
                 begin
                   write(filein,buffer[i]);
                 end;
             str := chr(17);chrout(str);
        until (endofile);
        close(filein);


end;(* filereceive *)


procedure LOGINDEC;
(* effectue le LOGIN sur DEC *)
(* verifie validite password et username *)


var
        i          : integer;
        notok,loggedin : boolean;
        ch,rep     :char;
        username   : string;
        password   : string;


begin

    loggedin := false;

    while (not loggedin) do
       begin

         gotoxy(1,2);clreol;bell;write(' DONNEZ VOTRE NOM D''UTILISATEUR :');
         readln(username);
         gotoxy(1,4);clreol;write(' DONNEZ VOTRE MOT DE PASSE :');
         readln(password);
         gotoxy(29,4);clreol;(*efface le mot de passe*)

         str := chr(3);chrout(str);(* envoi d'un CTRL/C au DEC *)

         readpr(notok);(* cherche prompt *)

         str := username;chrout(str);
         str := ' ';chrout(str);
         str := password;chrout(str);
         str := chr(10);chrout(str);(* LOGIN *)
```

```
        notok := false;
        reader(notok);
        loggedin := not (notok);

    end;(* while not loggedin *)

end;(* procedure LOGINDEC *)


procedure TRANSDEC;
(* phase de transfert d'un ou plusieurs fichiers vers le DEC *)

var     todec,fini : boolean;
        rep  : char;
        fnamedec,fnamemic : string;

begin

    rep:=' ';
    fini := false;
    while (not fini) do
      begin
        todec := true;
        queryfilename(fnamedec,fnamemic,todec);
        filesend(fnamemic,fnamedec);
        bell;gotoxy(1,10);
        write(' VOULEZ-VOUS EFFECTUER EN NOUVEAU TRANSFERT VERS LE DEC (O/N) ?');
        readln(rep);
        if (rep = 'O') or (rep = 'o') then fini := false else fini:=true;
        gotoxy(1,10);clreol;gotoxy(77,10);write('*');
      end;

end;(* transdec *)


procedure TRANSMIC;
(* transfere un ou plusieurs fichiers vers le BB *)

var
        todec,fini : boolean;
        rep  : char;
        fnamedec,fnamemic : string;

begin

    rep:=' ';
    fini := false;
    while (not fini) do
      begin
        todec := false;
        queryfilename(fnamedec,fnamemic,todec);
        filereceive(fnamemic,fnamedec);
        bell;gotoxy(1,10);
        write(' VOULEZ-VOUS EFFECTUER UN NOUVEAU TRANSFERT VERS LE MICRO (O/N) ?');
        readln(rep);
        if (rep = 'O') or (rep = 'o') then fini :=false else fini:=true;
        gotoxy(1,10);clreol;gotoxy(77,10);write('*');
      end;
```

```
end;(* transmic *)

procedure LOGOUT;
(* effectue le LOGOUT sur le DEC *)

var
    i       : integer;
    ch      : char;
    command : string;

begin

    str := 'logout';strout(str);
    str := chr(10);chrout(str);
    loggedout := true;

end;(* logout *)

(*---------------------- PROGRAMME PRINCIPAL ----------------------------*)
begin
clrscr;
decini;
logindec;
termini;
loggedout:=false;
while (not loggedout) do
begin
  clrscr;
  gotoxy(79,11);endbl;
  home;blank;
  write('************************   CONNEXION BB <=> DEC-20   ***************
  ********');
  gotoxy(0,1);
  write('**************  MENU  ***********************************************
  *********');
  gotoxy(0,2);
  write('*  TERMINAL ..................  <1>          |      NOM DU FICHIER SUR DE
  C       *');
  gotoxy(0,3);
  write('*                                            |-------------------------
  --------*');
  gotoxy(0,4);
  write('*  TRANSFERT VERS MICRO ......  <2>          |
          *');
  gotoxy(0,5);
  write('*                                            |-------------------------
  --------*');
  gotoxy(0,6);
  write('*  TRANSFERT VERS DEC ........  <3>          |      NOM DU FICHIER SUR MI
  CRO     *');
  gotoxy(0,7);
  write('*                                            |-------------------------
  --------*');
  gotoxy(0,8);
  write('*  FIN DE SESSION ............  <4>          |
          *');
  gotoxy(0,9);
  write('*------------------------------------------|-------------------------
```

```
        --------*');
        gotoxy(0,10);
        write('*
                *');
        gotoxy(0,11);
        write('*********************************************************************
        *********');

        y := 8;(* initialise position fleche *)
        gotoxy(0,10);write('* QUELLE OPTION CHOISISSEZ-VOUS ?');clreol;
        gotoxy(77,10);write('*');
        gotoxy(33,10);read(ch);
        gotoxy(34,y);write('    ');
        while (ch<>'1') and (ch <> '2') and (ch <> '3') and (ch <> '4') do
          begin
            gotoxy(1,10);clreol;
            gotoxy(77,10);write('*');
            bell;
            gotoxy(0,10);write('* QUELLE OPTION CHOISISSEZ-VOUS ?');read(ch);
            gotoxy(34,y);write('    ');
          end;
        y := (ord(ch)-48)*2;
        gotoxy(34,y);write(' <==');

        case ch of

              '1' : terminal;
              '2' : transmic;
              '3' : transdec;
              '4' : logout

        end;(* end case *)

      end;(* not logged out *)

      gotoxy(0,13);

      end.
```

```
;-----------------------------------------------------------------------
;
;       routine BLANK
;
;       fonction : La routine BLANK permet de commencer une zone d'ecran
;                  en video-inverse,en envoyant au terminal les
;                  caracteres ESC j
;
;       assemblee avec M80
;       convertie en *.INT par CONVERTM
;       link avec JRT PASCAL
;
;-----------------------------------------------------------------------

        .z80
        dseg

conout  equ     0f00ch      ;routine affiche caractere BB

        entry   blank
        defb    95,6,0
        defb    92
        defb    0           ;entry code for JRT
blank:
        push    af
        ld      a,01bh
        call    conout      ;escape
        ld      a,06ah
        call    conout      ; j
        pop     af
        ret
        end
```

```
;-------------------------------------------------------------------------------
;
;       routine CHROUT : sortie d'une chaine de caractere sur SIO A
;
;       sortie : ---
;       entree : STR chaine de caractere declaree par STRING en JRT Pascal
;
;       fonction :- La routine CHROUT permet d'envoyer une chaine de
;                   caracteres sur le port serie A:les caracteres sont
;                   envoyes un par un ,sans attendre d'echo:ceci est utile
;                   pour envoyer par exemple des caracteres de controle
;                   pour lesquels le DEC ne fait pas d'echo:c'est le cas
;                   pour ctrl/Q et ctrl/S
;
;       remarque :- la longueur de la chaine ,qu'il est possible d'envoyer
;                   est limitee a 256 caracteres.
;                 - un STRING est represente en memoire par:
;                           - 2 octets de longueur
;                           - la chaine elle-meme
;                   En JRT,le parametre passe a CHROUT (par variable),est
;                   l'adresse du premier octet du STRING,i.e.l'octet de
;                   poids faible du double octet de longueur.
;
;       assemblee avec M80
;       convertie en *.INT par CONVERTM (cfr document JRT page 90)
;       link avec JRT PASCAL (cfr doc. JRT   page 87)
;
;-------------------------------------------------------------------------------
        .Z80
        dseg
siodat  equ     04h     ;registre donnee SIO A
siоctl  equ     06h     ;registre controle SIO A
adrbd   equ     00h     ;registre baud rate (de 00h a 03h)
bdrate  equ     0ah     ;vitesse transfert(2400 bds),07h pour 1200

        entry   CHROUT
        defb    95,6,0
        defb    92
        defb    0       ;entry code for JRT
CHROUT:
        push    af
        push    bc
        push    de
        push    hl
        push    ix
        push    iy

        push    de
        pop     ix
        ld      l,(ix-4)
        ld      h,(ix-3)
        ld      b,(hl)  ;B est la longueur du string
        inc     hl
        inc     hl      ;HL est le ptr vers le STRING
debut:
        ld      a,(hl)  ;place le caractere dans A
        push    af
        push    bc
        push    hl
        ld      c,a     ;sortir le caractere present dans A vers le SIO
```

```
lp:     ld      a,10h
        out     (sioctl),a
        in      a,(sioctl)
        and     04h
        cp      04h
        jr      nz,lp
        ld      a,c
        out     (siodat),a
        pop     hl
        pop     bc
        pop     af

        inc     hl          ;on passe au caractere suivant
        djnz    debut       ;a-t'on encore des donnees?
fin:
        pop     iy
        pop     ix
        pop     hl
        pop     de
        pop     bc
        pop     af
        ret

        end
```

```
;-----------------------------------------------------------------------
;
;       routine CLREOL
;
;       fonction : La routine CLREOL permet d'effacer une ligne d'ecran
;                  a partir de la position courante du curseur,jusqu'a la
;                  fin de la ligne.
;                  Les caracteres qui realisent cette fonction sont ESC T
;
;       appelle CONOUT routine impression caractere presente en ROM
;       assemblee avec M80
;       convertie en *.INT par CONVERTM
;       link avec JRT PASCAL
;
;-----------------------------------------------------------------------

        .Z80
        dseg

conout  equ     0f00ch    ;routine affiche caractere BB

        entry   clreol
        defb    95,6,0
        defb    92
        defb    0     ;entry code for JRT
clreol:
        push    af
        ld      a,01bh
        call    conout
        ld      a,054h
        call    conout
        pop     af
        ret
        end
```

```
;-----------------------------------------------------------------------
;
;       routine CLREOS
;
;       fonction : La routine CLREOS permet d'effacer l'ecran a partir de
;                  la position courante du curseur.
;                  Les caracteres realisant cette fonction sont ESC Y
;
;       appelle CONOUT routine impression caractere presente en ROM
;       assemblee avec M80
;       convertie en *.INT par CONVERTM
;       link avec JRT PASCAL
;
;-----------------------------------------------------------------------

        .z80
        dseg
esc     equ     01bh        ;escape
eep     equ     059h        ;erase to end of page
conout  equ     0f00ch      ;routine affiche caractere BB

        entry   clreos
        defb    95,6,0
        defb    92
        defb    0     ;entry code for JRT
clreos:
        push    af
        ld      a,esc
        call    conout
        ld      a,eep
        call    conout
        pop     af
        ret
        end
```

```
;-----------------------------------------------------------------------
;
;       routine CLRSCR
;
;       fonction : La routine CLRSCR permet d'effacer tout l'ecran a partir
;                  de la position courante du curseur.
;                  Le caractere realisant cette fonction est ctrl/Z
;       appelle CONOUT routine impression caractere presente en ROM
;       assemblee avec M80
;       convertie en *.INT par CONVERTM
;       link avec JRT PASCAL
;
;-----------------------------------------------------------------------

        .Z80
        dseg
clear   equ     01ah        ;ctrl/Z
conout  equ     0f00ch      ;routine affiche caractere BB

        entry   clrscr
        defb    95,6,0
        defb    92
        defb    0     ;entry code for JRT

clrscr:
        push    af
        ld      a,clear
        call    conout
        pop     af
        ret
        end
```

```
;-----------------------------------------------------------------------
;
;       routine DECINI
;
;       fonction : La routine DECINI permet d'initialiser le port A du
;                  SIO avec les caracteristiques de la transmission
;       assemblee avec M80
;       convertie en *.INT par CONVERTM
;       link avec JRT PASCAL
;
;-----------------------------------------------------------------------

        .z80
        dseg

        siodat  equ     04h     ;registre donnee SIO A
        sioctl  equ     06h     ;registre controle SIO A
        adrbd   equ     00h     ;registre baud rate (de 00h a 03h)
        bdrate  equ     0ah     ;vitesse transfert(2400 bds),07h pour 1200
        bdos    equ     05h

        entry   decini          ;initialise le SIO
        defb    95,6,0
        defb    92
        defb    0    ;entry code for JRT
decini:
        push    af
        push    bc
        push    de
        push    hl
        push    ix
        push    iy
        ld      a,bdrate        ;vitesse 2400 bauds
        out     (adrbd),a
        ld      a,04h           ;registre 4
        out     (sioctl),a
        ld      a,044h          ;1 stop bit
        out     (sioctl),a
        ld      a,03h           ;registre 3
        out     (sioctl),a
        ld      a,0c1h          ;8 bits/caractere
        out     (sioctl),a
        ld      a,05h           ;registre 5
        out     (sioctl),a
        ld      a,0eah
        out     (sioctl),a
        ld      a,11h           ;registre 1
        out     (sioctl),a
        ld      a,0h            ;disable interrupt
        out     (sioctl),a

        pop     iy
        pop     ix
        pop     hl
        pop     de
        pop     bc
        pop     af
        ret
        end
```

```
;-------------------------------------------------------------------------
;
;       routine ENDBL
;
;       fonction : La routine ENDBL permet de delimiter la fin de zone
;                  d'ecran en video-inverse.
;                  Les caracteres a envoyes a la console sont ESC k
;       appelle CONOUT routine impression caractere en ROM
;       assemblee avec M80.
;       convertie en *.INT avec CONVERTM
;       link avec JRT PASCAL
;
;-------------------------------------------------------------------------

        .z80
        dseg

conout  equ     0f00ch      ;routine affiche caractere BB

        entry   endbl
        defb    95,6,0
        defb    92
        defb    0           ;entry code for JRT
endbl:
        push    af
        ld      a,01bh
        call    conout      ;escape
        ld      a,06bh
        call    conout      ; k
        pop     af
        ret
        end
```

```
;-----------------------------------------------------------------------
;
;       routine GETC
;
;       sortie : EOT variable booleenne
;       entree : ---
;
;       fonction :
;                 La routine GETC permet la scrutation du clavier;si
;                 un ctrl/Z a ete tape ,alors EOT est mis a 'true'.
;       appelle: CONST routine test etat console (en ROM)
;                CONIN routine lecture caractere console (en ROM)
;       assemblee avec M80
;       convertie en *.INT par CONVERTM
;       link avec JRT PASCAL
;
;-----------------------------------------------------------------------
        .Z80
        dseg

        conin   equ     0f009h ;routine lecture console
        const   equ     0f006h ;routine test etat console

        entry   getc
        defb    95,6,0
        defb    92
        defb    0
getc:
        push    af
        push    bc
        push    de

        push    de
        pop     ix

bcl:    call    const           ;teste l'etat de la console
        or      a
        jr      z,end           ;si pas de caractere tape,on quitte GETC
        call    conin           ;si caractere tape,le placer dans A
        cp      01ah            ;teste si ctrl/Z
        jr      nz,false
        ld      c,(ix-4)        ;si ctrl/z,eot = true
        ld      b,(ix-3)
        ld      a,01h
        ld      (bc),a
        jr      end

false:
        ld      c,(ix-4)        ;si <> ctrl/z,eot = false
        ld      b,(ix-3)
        ld      a,00h
        ld      (bc),a

end:
        pop     de
        pop     bc
        pop     af
        ret
        end
```

```
;-----------------------------------------------------------------------
;
;       routine GOTOXY :(equivalent a GOTOXY en UCSD Pascal)
;
;       sortie : ---
;       entree : x,y : entier passe par valeur
;
;       fonction : La routine GOTOXY permet de positionner le curseur
;                  a la colonne x de la ligne y.
;
;                  (0,0)              (79,0)
;                    ------------------
;                    |                |
;                    |     ECRAN      |
;                    |                |
;                    ------------------
;                  (0,23)             (79,23)
;
;       appelle CONOUT routine impression caractere en ROM
;       assemblee avec M80
;       convertie en *.INT par CONVERTM
;       link avec JRT PASCAL
;-----------------------------------------------------------------------

        .Z80
        dseg
offset  equ     020h        ;origine coordonnees pour TVI
conout  equ     0f00ch      ;routine affiche caractere BB
        entry   gotoxy
        defb    95,6,0
        defb    92
        defb    0           ;entry code for JRT

gotoxy:
        push    af
        push    bc
        push    de
        push    hl
        push    ix
        push    iy

        push    de
        pop     ix
        ld      a,01bh      ;esc
        call    conout
        ld      a,03dh      ;=
        call    conout
        ld      a,offset
        add     a,(ix-4)    ;ajoute OFFSET a y
        call    conout
        ld      a,offset
        add     a,(ix-6)    ;ajoute OFFSET a x
        call    conout

        pop     iy
        pop     ix
        pop     hl
        pop     de
        pop     bc
```

```
pop     af
ret
end
```

```
;-----------------------------------------------------------------------
;
;       routine HOME
;
;       fonction : La routine HOME permet de placer le curseur dans
;                  le coin superieur gauche de l'ecran
;
;       appelle CONOUT routine impression caractere en ROM
;       assemblee avec M80
;       convertie en *.INT par CONVERTM
;       link avec JRT PASCAL
;
;-----------------------------------------------------------------------

        .z80
        dseg
curhom  equ     01eh        ;ctrl/^
conout  equ     0f00ch      ;routine affiche caractere BB

        entry   home
        defb    95,6,0
        defb    92
        defb    0     ;entry code for JRT

home:
        push    af
        ld      a,curhom
        call    conout
        pop     af
        ret
        end
```

```
;-----------------------------------------------------------------------
;    routine PAGERD : lit une page d'ecran sur port A sio
;
;    sortie : buffer,size,endofile
;    entree : ---
;
;    fonction:-lit une "page" sur le port serie A du BB;la page est stockee
;              dans BUFFER,et la taille de la page est donnee par SIZE
;             -teste si l'utilisateur a taper sur ctrl/z pour
;              sortir de la procedure;dans ce cas,on quitte PAGERD
;              avec ENDOFILE vrai.
;
;    remarque:les caracteres indesirables envoyes par le DEC sont
;             filtres:cursor home,clear screen,bell...
;             le dernier CR n'etant pas suivi par un LF,PAGERD l'ajoute.
;
;-----------------------------------------------------------------------
        .Z80
        dseg

        siodat  equ     04h     ;registre donnee SIO A
        sioctl  equ     06h     ;registre controle SIO A
        conin   equ     0f009h ;routine lecture console
        const   equ     0f006h ;routine test etat console
        conout  equ     0f00ch ;routine affichage console

        entry   pagerd
        defb    95,6,0
        defb    92
        defb    0
pagerd:
        push    af
        push    bc
        push    de
        push    hl
        push    ix
        push    iy


        push    de
        pop     ix

        ld      c,(ix-8)
        ld      b,(ix-7)  ;BC pointe sur BUFFER
        ld      hl,1      ;HL compteur

idle:   call    const
        or      a
        jr      z,idle1
        call    conin
        cp      01ah        ;tests si ctrl/z
        jr      z,cont

idle1:
        ld      a,010h
        out     (sioctl),a
        in      a,(sioctl)
        and     01h
        cp      01h
        jr      nz,idle
```

```
            in      a,(siodat)
            or      a
            jr      z,idle
            cp      01eh            ;si ctrl/^ ne pas l'afficher
            jr      z,lbl2
            cp      087h            ;si ctrl/G,envoyer un LF
            jr      nz,next
            ld      a,0ah           ;envoyer un lf
            ld      (bc),a
            inc     hl
            inc     bc
            jr      end

;probleme avec ctrl/G => envoyer un ctrl/J
; 3 caracteres en trop apres le ctrl/M et
; 1 caractere en trop avant le ctrl/M

next:
            and     07fh            ;MASQUE PARITE
            push    af
            call    conout          ;affiche le caractere recu
            pop     af
lbl2:
            ld      (bc),a
            inc     bc
            inc     hl
            jr      idle
cont:
            ld      c,(ix-4)    ;fin fichier vraie
            ld      b,(ix-3)
            ld      a,01h
            ld      (bc),a
bcl:        ld      a,10h
            out     (sioctl),a
            in      a,(sioctl)
            and     04h
            cp      04h
            jr      nz,bcl
            ld      a,03h
            out     (siodat),a
bcl2:       ld      a,10h
            out     (sioctl),a
            in      a,(sioctl)
            and     04h
            cp      04h
            jr      nz,bcl2
            ld      a,03h
            out     (siodat),a
end:
            ld      c,(ix-6)    ;stocke le nombre de caracteres dans SIZE
            ld      b,(ix-5)
            ld      a,l
            ld      (bc),a
            inc     bc
            ld      a,h
            ld      (bc),a

            pop     iy
            pop     ix
            pop     hl
```

```
pop     de
pop     bc
pop     af
ret
end
```

```
;-----------------------------------------------------------------------
;
;       routine READPR
;
;       sortie : NOTOK  : booleen
;       entree : ---
;
;       fonction : La routine READPR permet de lire sur le port serie A
;                  les caracteres envoyes par le DEC;si,parmi ces
;                  caracteres ,sont detectes '%' ou '?',alors le
;                  booleen NOTOK est mis a la valeur true;sinon,NOTOK
;                  est laisse a la valeur qu'il avait a l'entree de
;                  la procedure.On sort de la routine des que l'on
;                  trouve le prompt du DEC = @.
;
;       remarque :-cette routine a ete ecrite pour detecter :
;                  1.une erreur dans le mot de passe sur le DEC
;                  2.une erreur dans le username sur le DEC
;                  3.DIR d'un fichier inexistant sur le DEC
;
;                 -un booleen true est egal a 1 en JRT,et 0 si false.
;
;       assemblee avec M80
;       convertie en *.INT par CONVERTM
;       link avec JRT PASCAL
;
;-----------------------------------------------------------------------

        .Z80
        dseg

        siodat  equ     04h     ;registre donnee SIO A
        sioctl  equ     06h     ;registre controle SIO A


        entry   readpr
        defb    95,6,0
        defb    92
        defb    0               ;entry code for JRT

readpr:
        push    af
        push    bc
        push    de
        push    hl
        push    ix
        push    iy

lp:     ld      a,10h
        out     (sioctl),a
        in      a,(sioctl)
        and     01h
        cp      01h
        jr      nz,lp
        in      a,(siodat)
        or      a
        jr      z,lp
        and     07fh            ;masque la parite
        ld      b,025h          ; %
        cp      b
```

```
        jr      z,true
        ld      b,03fh          ;?
        cp      b
        jr      z,true
        ld      b,040h          ;si @ alors on sort
        cp      b
        jr      nz,lp
        jr      end
true:
        push    de              ;renvoie un booleen true si ? ou %
        pop     ix
        ld      c,(ix-4)
        ld      b,(ix-3)
        ld      a,0ih
        ld      (bc),a
        jr      lp
end:
        pop     iy
        pop     ix
        pop     hl
        pop     de
        pop     bc
        pop     af
        ret

        end
```

```
;-----------------------------------------------------------------------
;
;       routine STROUT : sortie d'une chaine de caractere sur SIO A
;
;       sortie : ---
;       entree : STR chaine de caractere declaree par STRING en JRT Pascal
;
;       fonction :- La routine STROUT permet d'envoyer une chaine de
;                   caracteres sur le port serie A:les caracteres sont
;                   envoyes un par un,et on attend leur echo ,apres chaque
;                   caractere envoye.
;
;       remarque :- la longueur de la chaine ,qu'il est possible d'envoyer
;                   est limitee a 256 caracteres.
;                 - un STRING est represente en memoire par:
;                           - 2 octets de longueur
;                           - la chaine elle-meme
;                   En JRT,le parametre passe a STROUT (par variable),est
;                   l'adresse du premier octet du STRING,i.e. l'octet de
;                   poids faible du double octet de longueur.
;
;       assemblee avec : M80
;       link avec : JRT PASCAL
;       convertie en fichier *.INT par CONVERTM (cfr document JRT page 90)
;
;-----------------------------------------------------------------------

        .Z80
        dseg
siodat  equ     04h     ;registre donnee SIO A
sioctl  equ     06h     ;registre controle SIO A
adrbd   equ     00h     ;registre baud rate (de 00h a 03h)
bdrate  equ     0ah     ;vitesse transfert(2400 bds),07h pour 1200

        entry   strout
        defb    95,6,0
        defb    92
        defb    0       ;entry code for JRT
strout:
        push    af
        push    bc
        push    de
        push    hl
        push    ix
        push    iy

        push    de
        pop     ix
        ld      l,(ix-4)
        ld      h,(ix-3)
        ld      b,(hl)          ;B est la longueur du string
        inc     hl
        inc     hl              ;HL est le ptr vers le STRING
debut:
        ld      a,(hl)          ;place le caractere dans A
        push    af
        push    bc
        push    hl
        ld      c,a             ;sortir le caractere lu vers le SIO
lp:     ld      a,10h
```

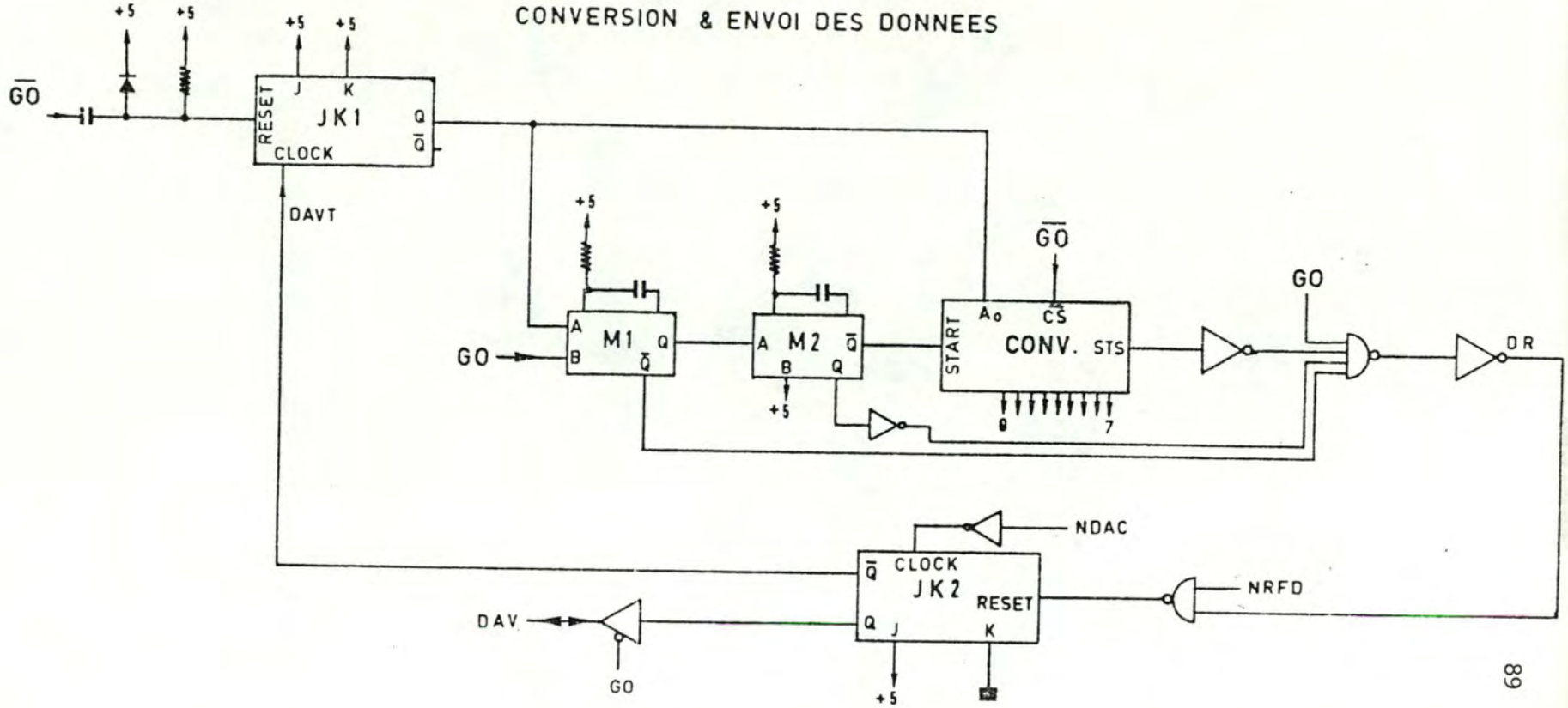
```
        out     (sioctl),a
        in      a,(sioctl)
        and     04h
        cp      04h
        jr      nz,lp
        ld      a,c
        out     (siodat),a
        pop     hl
        pop     bc
        pop     af
        push    af                  ;attendre l'echo du caractere
bcl:    ld      a,10h
        out     (sioctl),a
        in      a,(sioctl)
        and     01h
        cp      01h
        jr      nz,bcl
        in      a,(siodat)
        or      a
        jr      z,bcl
        pop     af
        inc     hl                  ;on passe au suivant
        djnz    debut               ;a-t'on encore des donnees?
fin:
        pop     iy
        pop     ix
        pop     hl
        pop     de
        pop     bc
        pop     af
        ret

        end
```

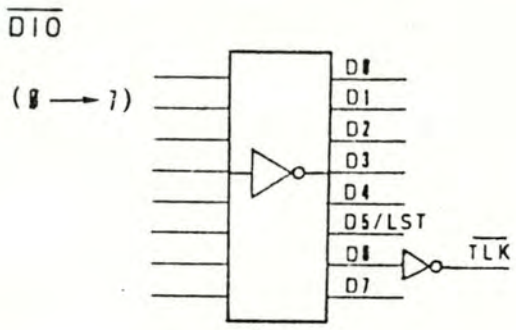
```
;-----------------------------------------------------------------------
;
;       routine TERM
;
;       fonction : La routine TERM permet de se connecter sur le DEC :
;                  le BB est ainsi transforme en terminal ;le retour
;                  au programme appelant se fait en tapant sur la
;                  touche <backspace>
;
;       assemblee avec M80
;       convertie en *.INT par CONVERTM
;       link avec JRT PASCAL
;
;-----------------------------------------------------------------------

        .Z80
        dseg

        siodat  equ     04h     ;registre donnee SIO A
        sioctl  equ     06h     ;registre controle SIO A

        conin   equ     0f009h ;routine lecture console
        const   equ     0f006h ;routine test etat console
        conout  equ     0f00ch ;routine affichage console

        bs      equ     08h     ;caractere backspace


        entry   term
        defb    95,6,0
        defb    92
        defb    0               ;entry code for JRT

term:
        push    af
        push    bc
        push    de
        push    hl
        push    ix
        push    iy
idle:   call    const           ;teste console status
        or      a               ;donnee disponible?
        jr      z,idle1         ;non,alors teste SIO
        call    conin           ;oui,lire le caractere
        cp      bs
        jr      z,out           ;si backspace,retour a CPM
        ld      c,a             ;sortir le caractere lu vers le SIO
loop:   ld      a,10h
        out     (sioctl),a
        in      a,(sioctl)
        and     04h
        cp      04h
        jr      nz,loop
        ld      a,c
        out     (siodat),a
idle1:  ld      a,010h
        out     (sioctl),a
        in      a,(sioctl)      ;lire le caractere sur SIO
        and     01h
        cp      01h
```

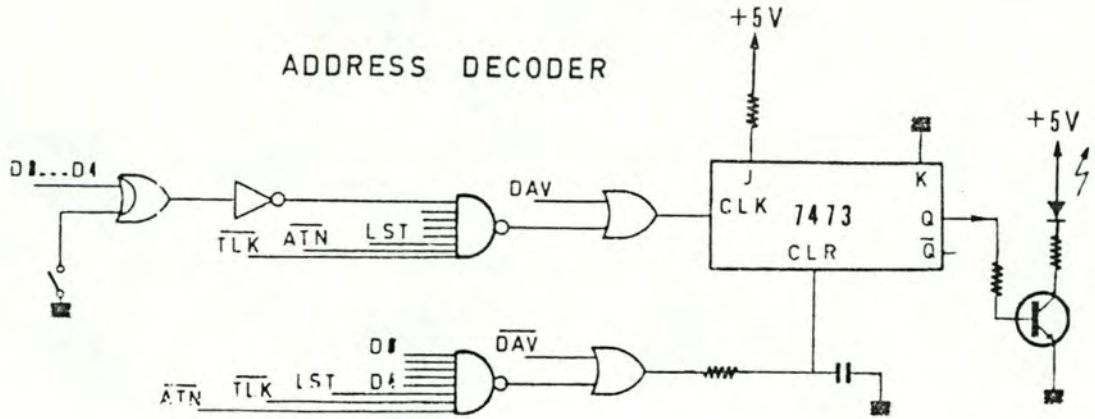
```
        jr      nz,idle
        in      a,(siodat)
        or      a               ;si caractere nul,..
        jr      z,idle          ;on ignore
        call    conout          ;sinon,on l'imprime
        jr      idle            ;boucle
out:    pop     iy
        pop     ix
        pop     hl
        pop     de
        pop     bc
        pop     af
        ret
        end
```

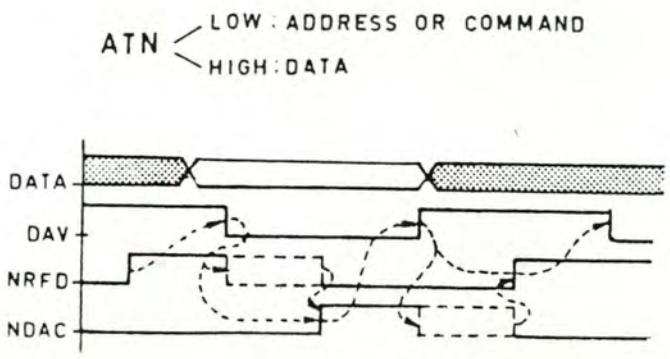
ANNEXE IV. SCHEMA DES CIRCUITS DE CONVERSION DE DONNEES.

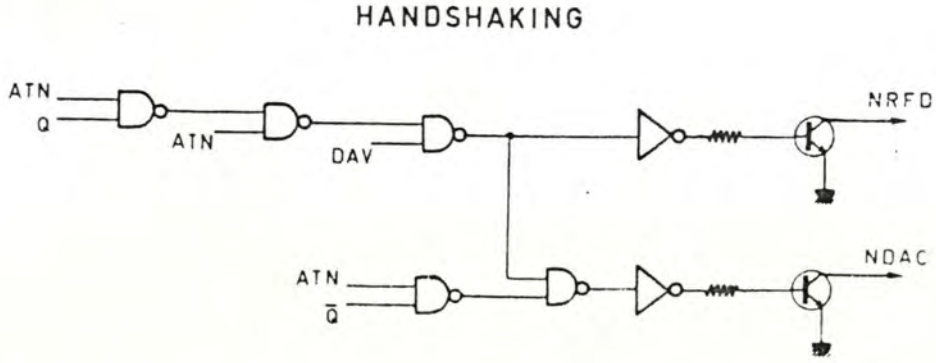
CONVERSION & ENVOI DES DONNEES

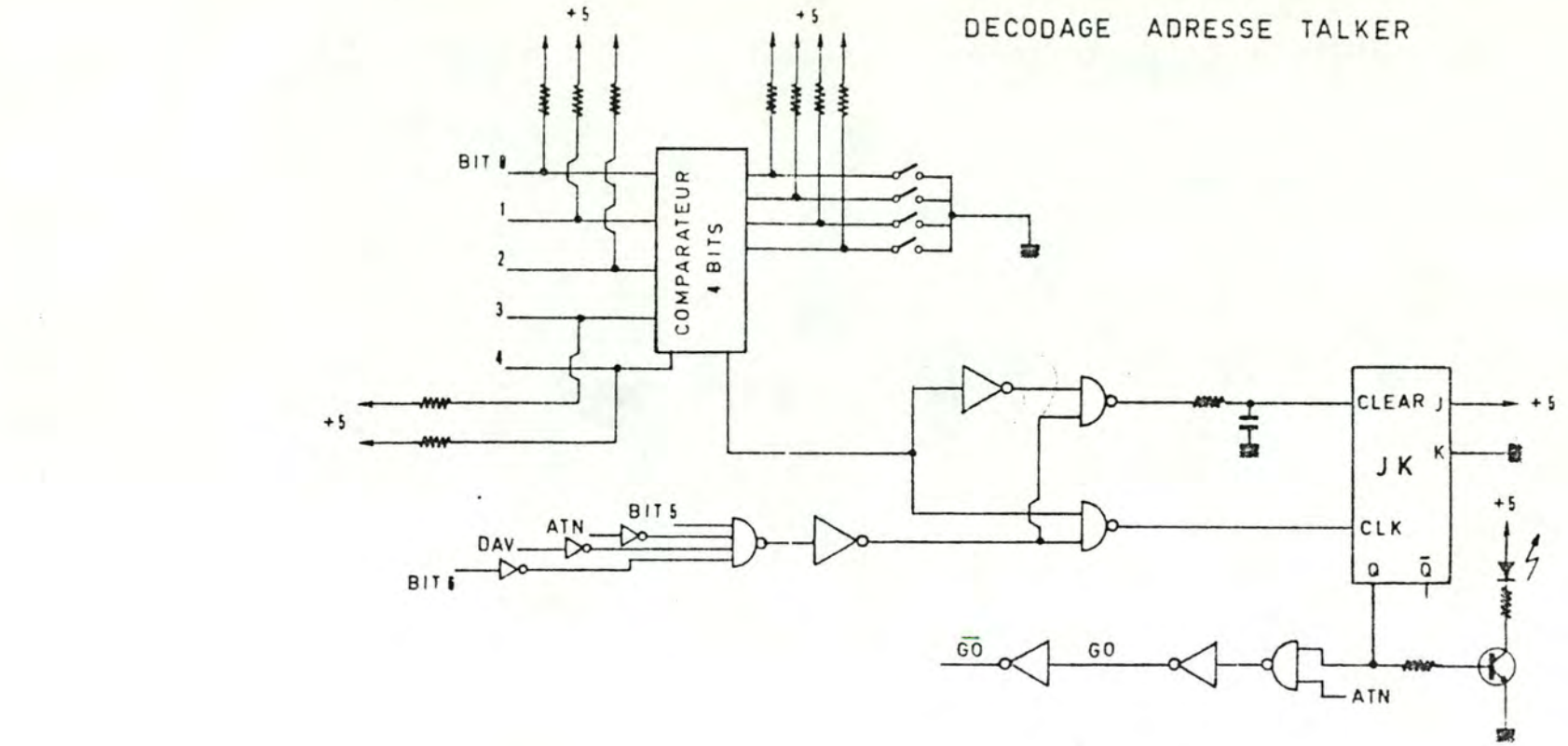


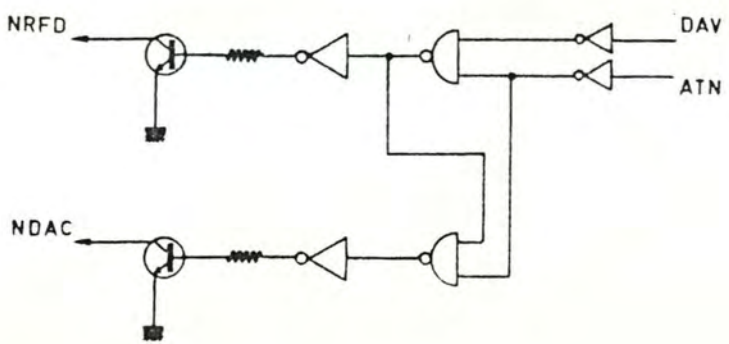

D10
(8 → 7)
D0
D1
D2
D3
D4
D5/LST
D6
D7
TLK
ATN
ATN


ADDRESS DECODER
+5V
+5V
D8...D4
TLK ATN LST
DAV
J
K
CLK
7473
CLR
Q
Q
D8
D4
ATN TLK LST
DAV


ATN
LOW : ADDRESS OR COMMAND
HIGH : DATA
DATA
DAV
NRFD
NDAC


HANDSHAKING
ATN
Q
ATN
DAV
NRFD
ATN
Q
NDAC


DECODAGE ADRESSE TALKER
+5
+5
BIT 0
1
2
3
4
COMPARATEUR 4 BITS
+5
CLEAR
J
K
JK
CLK
Q
Q̄
+5
+5
ATN
BIT 5
DAV
BIT 6
GO
GO
ATN


NRFD
DAV
ATN
NDAC

SEQUENCE DE CONVERSION DU A/D