



DISQUETTE ET DISQUE DUR

ATARI

EDITIONS MICRO APPLICATION



LIVRE DATA BECKER

ST DISQUETTE ET DISQUE DUR

ATARI

EDITIONS MICRO APPLICATION



LIVRE DATA BECKER

Distribué par :

MICRO APPLICATION

58, Rue du Faubourg Poissonnière
75010 PARIS

EDITIONS RADIO

189, Rue Saint Jacques
75005 PARIS

(c) Reproduction interdite sans l'autorisation de
MICRO APPLICATION

'Toute représentation ou reproduction, intégrale ou partielle, faite sans le consentement de MICRO APPLICATION est illicite (Loi du 11 Mars 1957, article 40, 1er alinéa).

Cette représentation ou reproduction illicite, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles 425 et suivants du Code Pénal.

La Loi du 11 Mars 1957 n'autorise, aux termes des alinéas 2 et 3 de l'article 41, que les copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à l'utilisation collective d'une part, et d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration'.

ISBN : 2-86899-087-8

Code ER : 213

(c) 1989 DATA BECKER

Merowingerstrasse, 30
4000 Düsseldorf - RFA

Auteurs : Mr. BRAUN, Mr. DITTRICH, Mr. SCHRAMM

Traduction française assurée par Olivier POLETTE

(c) 1989 MICRO APPLICATION

58 Rue du Faubourg Poissonnière
75010 PARIS

Collection dirigée par Mr Philippe OLIVIER
Edition réalisée par Frédérique BEAUDONNET

IBM PC, IBM PC XT, IBM PC AT sont des modèles déposés de International Business Machines.

ATARI, ST, Mega ST, GDOS sont des marques déposées et SH104, SH204, SH205, SF354, SF314, SMM804 et SLM804 sont des modèles déposés par Atari Corp.

Motorola est une marque déposée et MC68000 est un modèle de Motorola Inc.

MS DOS est une marque déposée de Microsoft Corporation.

UNIX est une marque déposée de Bell Laboratories.

INTEL est une marque déposée de Intel Corporation.

8086 est un modèle déposé de Intel.

SOMMAIRE

<u>1.</u>	<u>Introduction</u>	1
<u>2.</u>	<u>Programmes et Fichiers</u>	3
2.1	Structure des fichiers et modes d'accès avec différents langages évolués	6
2.1.1	Les fonctions de GEMDOS : généralités	7
2.2	Modes d'accès aux fichiers en BASIC	9
2.2.1	Les fichiers séquentiels en BASIC	10
2.2.2	Les fichiers RANDOM en BASIC	12
2.3	Gestion de fichiers en PASCAL	15
2.3.1	Les fichiers séquentiels en PASCAL	15
2.3.2	Les fichiers RANDOM en PASCAL	19
2.4	Modes d'accès aux fichiers en C	20
2.4.1	Les fichiers séquentiels en C	24
2.4.2	Les fichiers RANDOM en C	25
2.5	La gestion de fichiers en FORTRAN	28
2.5.1	Les fichiers séquentiels en FORTRAN	29
2.5.2	Les fichiers RANDOM en FORTRAN	30
2.6	Une base de données simplifiée	31
<u>3.</u>	<u>Structure des données</u>	41
3.1	Le format des disquettes	41
3.2	Le BOOT secteur	43
3.2.1	Un programme de formatage	46
3.2.2	Le BIOS-Parameter-Block BPB	57
3.3	Le catalogue	66
3.4	La FAT	70
3.5	Structure du programme	71
3.5.1	Le program-header	72
3.5.2	La table de relocation	74
3.6	Le format du disque dur	76

4.	<u>Les lecteurs de disquette</u>	79
4.1	Fonctions et structure	79
4.2	Le circuit DMA	81
4.2.1	Le contrôleur de disque	82
4.2.1.1	Connexions	86
4.2.1.2	Organisation	92
4.2.1.3	Description des commandes	101
4.2.1.4	Interprétation de l'état	138
4.2.2	L'interface avec le lecteur de disquette	143
4.3	Branchement du lecteur de disquette	146
5.	<u>Le disque dur SH204</u>	149
5.1	Fonctions et structure	150
5.1.1	Le contrôleur de disque dur	151
5.1.1.1	Structures d'instructions	154
5.1.1.2	Liste des instructions	162
5.1.1.3	HDC TOOLS	169
5.1.1.4	Analyseur de partition	174
5.2	Branchement du disque dur	185
5.3	Impression de tout le catalogue	186
6.	<u>Le RAM disk</u>	197
6.1	Un programme de RAM disk très pratique	199
6.2	Copie d'une disquette vers le RAM disk	214
7.	<u>Exemple de programmation en langage machine avec un éditeur de disquette</u>	221
7.1	Les fonctions du TOS pour l'accès au lecteur de disquette	221
7.2	Listing et mode d'emploi du moniteur de disquette	231
7.2.1	Le menu principal	349
7.2.2	Le menu "Track"	350
7.2.3	Le menu "Track with Syncs"	351
7.2.4	Le menu "Sector"	351

7.2.5	Le menu "Cluster"	352
7.2.6	Le menu "Format"	353
7.2.7	Le menu "GAP"	353
7.2.8	Le menu "Options"	354
7.3	Exemples d'utilisation de l'éditeur de disquettes	354
7.3.1	File Allocation Table	359
7.3.2	Dossiers et sous-catalogues sur la disquette	361
7.3.3	Formatage en format non standard (non ATARI)	363
7.4	L'assemblage avec différents assembleurs	365
8.	<u>Programmes utilitaires en BASIC</u>	367
8.1	Mise en route et paramétrage	367
8.2	Quelques programmes en exemple	368
8.2.1	Interface BASIC/TOS	369
8.2.2	Lecture du catalogue	371
8.2.3	Lecture/Ecriture de secteurs	376
8.2.4	Différents types de formatage de disquettes	378
8.2.5	Recherche de données	384
8.2.6	Tri de données	386
8.2.7	Lecture de la date et de l'heure de façon formatée	389
8.3	La programmation du FDC en BASIC	392
8.3.1	Un programme d'interfaçage BASIC/FDC	393
8.3.2	Demo 1 - Accès à toutes les commandes du FDC	418
8.3.3	Demo 2 - Copie de disquette	428
8.3.4	Demo 3 - Création de formats standards et spéciaux	434
8.3.5	Demo 4 - Conversion de disquettes simple face en disquettes double face	441
8.4	Création de chargeurs BASIC	
 <u>Annexes</u>		
I.	Chargeur BASIC de editor.tos	452
II.	Tableau des codes ASCII	479
III.	Index	481

1. Introduction

Compte tenu de leur processeur 16/32 bits très rapide et de leur grande capacité mémoire, les micro-ordinateurs de la série ATARI ST sont destinés à des applications professionnelles. Mais plus encore que la mémoire interne, c'est la mémoire externe qui est importante. Les lecteurs de disquette et le disque dur sont des mémoires de masse très intéressantes offrant de nombreuses possibilités qui ne sont pas évoquées dans le manuel.

Pour utiliser pleinement un tel système, il est important de connaître les possibilités des composants. C'est l'objet de ce livre. Cet ouvrage vous ouvre une fenêtre sur le monde de la mémoire de masse et vous montrera comment programmer des logiciels d'application. Puis nous découvrirons les secrets des lecteurs de disquette de l'ATARI ST, du disque dur et des RAM disks.

Cette connaissance du soft et du hardware vous met en mesure de réaliser des choses intéressantes avec ces mémoires. Vous pourrez augmenter la capacité des disquettes, créer vos propres protections de programmes, réaliser des RAM disks adaptés à vos besoins et accélérer sensiblement l'accès aux données sur disquette ou sur disque dur à l'aide des utilitaires que nous vous donnons.

De plus, vous trouverez des programmes très utiles, comme par exemple un logiciel d'impression du catalogue (avec le contenu des dossiers !), ou un analyseur de disquettes ou de disque dur.

En outre, nous avons conçu pour vous un moniteur de disquettes très complet qui vous permettra d'accéder directement aux données sur disquette et d'exploiter vos connaissances nouvellement acquises. Ainsi, vous pourrez récupérer des fichiers écrasés, lire des disquettes d'autres types de micro-ordinateurs, etc...

Vous trouverez dans ce livre des trucs qui n'existent dans aucun autre manuel. Nous avons retrouvé ces commandes et ces informations grâce à un patient travail de recherche et d'exploration. C'est pourquoi cet ouvrage est publié un peu plus tardivement que prévu, mais sous une forme plus complète. Vous vous rendrez compte que le lecteur de disquette et le disque dur peuvent faire beaucoup plus de choses que vous n'aviez imaginé !

Nous espérons que ce livre répondra à toutes vos questions concernant les mémoires de masses et nous vous souhaitons bonne lecture !

Les auteurs,

Octobre 1986

2. Programmes et fichiers

Ce titre présente deux concepts qui en réalité n'en forment qu'un. En effet, il s'agit tout simplement de données informatiques sauvegardées sur mémoire de masse. Bien que la mémoire centrale des ordinateurs soit toujours plus grande (l'ATARI 1040 a en fin de compte 1 méga-octet de RAM), les données qui ne sont pas utilisées par l'ordinateur doivent être sauvegardées sur mémoire de masse externe afin de ne pas les perdre lorsqu'on éteint l'appareil. Peu importe la qualité de ces données : il peut s'agir d'un programme de traitement de texte, du nombre d'habitants à Nantes ou bien de l'évolution du prix du cacao durant les cinquantes dernières années.

Les mémoires de masse utilisées aujourd'hui sont les bandes magnétiques, les disquettes, les disques durs et dernièrement, les CD-ROMs. Avec tous ces systèmes, les données sont tout d'abord écrites sur le média puis relues en mémoire grâce à un procédé de lecture électronique. Quel que soit le support utilisé, on appelle tout groupe de données ayant le même nom un "Fichier".

L'utilisateur s'intéresse peu au fait que ces données soient des adresses, des lettres, des programmes sources ou des codes exécutables. Mais cela est très important pour l'ordinateur. Lorsqu'on sauvegarde du code exécutable, il ne doit pas y avoir de caractère de séparation entre chaque donnée. Il en va différemment avec les textes pour lesquels le caractère de séparation normal est inscrit à la fin de chaque phrase (le code correspondant à la touche RETURN par exemple).

Les utilisateurs reconnaissent le type d'un fichier à son extension, c'est à dire aux trois dernières lettres de son nom. Le système d'exploitation de l'ATARI ST ne différencie les fichiers qu'à leur extension. Si on renomme un programme par exemple (PRG) et qu'on met l'extension .DAT à la place de .PRG, il ne sera plus exécutable. Par contre, on pourra afficher les données du fichier qui le contient.

Voici les extensions utilisées par l'ATARI ST :

.PRG correspond à un programme en langage machine exécutable. Ce programme tourne sous GEM.

- .TOS** correspond à un programme en langage machine mais qui ne fonctionnera pas sous GEM.
- .TTP** correspond à .TOS mais une petite fenêtre s'ouvre à la mise en route du programme permettant de saisir des paramètres.
- .ACC** Il s'agit de programmes en langage machine qui sont chargés lors de la mise en route de l'ordinateur. Ces programmes restent en mémoire et peuvent être appelés à partir du menu BUREAU du DESKTOP.
- .INF** Sert au DESKTOP dans DESKTOP.INF. Ce type de fichier contient les informations sur les positions et la taille des fenêtres, les valeurs se trouvant dans le champ de contrôle, etc... Ce fichier est modifié lorsqu'on sélectionne l'option "*Sauvegarder bureau*" du menu OPTIONS.

Les autres fichiers, comme par exemple les programmes BASIC avec l'extension .BAS, ont une extension standard mais ne servent à rien pour le système d'exploitation. De plus, on peut charger un fichier ayant pour extension .TXT avec l'interpréteur BASIC si ce fichier contient un programme en BASIC. Les autres extensions n'ont donc pas de signification précise pour le système d'exploitation mais elles sont très utiles pour l'utilisateur afin de déterminer le type du fichier.

Les différences entre les types de fichiers résident dans leurs structures. La plupart des langages évolués font la distinction entre différentes formes de fichiers, c'est à dire fichiers avec ou sans caractères de séparation entre les lettres ou les chiffres, types de textes spéciaux, etc... Commençons par les fichiers qui ne contiennent que des chiffres ou du texte. Il existe différentes méthodes pour retrouver et traiter des données particulières.

La vitesse d'accès à certaines données sur disquette ou sur disque dur tient essentiellement à "l'intelligence" du système de gestion de fichier. Nous allons vous le prouver à l'aide d'un petit exemple :

Supposons un fichier qui contienne les adresses de tous les habitants de Nantes.

On nomme *ENREGISTREMENT* l'ensemble des données correspondant à une adresse, c'est-à-dire par exemple le nom, le prénom, la rue, le code postal, la ville. Chaque information prise séparément (le nom par exemple) est nommée *CHAMP*. Le type de gestion de fichier le plus simple est le fichier *SEQUENTIEL* : toutes les données sont enregistrées les unes à la suite des autres. Le programme qui lira de telles données doit reconnaître la fin de chaque enregistrement grâce à un caractère de séparation.

Le plus souvent, tous les enregistrements ont une longueur différente. Si on veut accéder au 10^{ième} enregistrement, il faut lire le fichier depuis le début jusqu'au 10^{ième} enregistrement. Ce processus est acceptable pour de petits fichiers mais, imaginez qu'on ait à rechercher l'adresse de Monsieur Dugus Dupont dans un fichier contenant les adresses de tous les français !

Si on travaille avec de nombreuses données, on garde le plus souvent une longueur d'enregistrement fixe et on utilise des fichiers *RANDOM* (*RANDOM* = aléatoire) à accès direct. Dans ces fichiers, chaque champ de chaque enregistrement a une longueur déterminée, soit par exemple 13 caractères pour le nom, 13 caractères pour le prénom, 5 pour le code postal, 14 pour la ville, 15 pour le nom de la rue et 4 pour le numéro du bâtiment, soit en tout 64 caractères par enregistrement.

Si on veut accéder au 10^{ième} enregistrement, il suffit de calculer le début du 10^{ième} enregistrement relativement au début du fichier, soit dans notre exemple $10 \times 64 = 640$. Il suffit alors de commencer la lecture au 640^{ième} octet du fichier pour accéder directement au 10^{ième} enregistrement. Bien entendu, ce calcul n'est valable que si l'on a un enregistrement 0.

Ce truc fonctionne seulement si l'on peut avoir accès directement à n'importe quel emplacement de la mémoire de masse. Cela n'est pas possible avec une cassette magnétique. Par contre, on peut le faire avec une disquette ou un disque dur car ils sont divisés en secteurs numérotés.

Revenons à notre exemple de gestion d'adresses. Si nous savons sur quel secteur commence l'enregistrement zéro, on peut facilement calculer sur quel secteur est situé le 640^{ième} octet du fichier.

Supposons que notre fichier commence au secteur 10. Avec l'ATARI ST, chaque secteur contient 512 octets donc le 640^{ième} octet de notre fichier se trouvera dans le secteur 11 au $640-512=128$ ^{ième} emplacement.

Le programmeur n'a pas à s'occuper de ce calcul lorsqu'il travaille avec un langage évolué. On appelle langage évolué n'importe quel langage de programmation en dehors du langage machine également nommé assembleur. Les programmeurs en assembleur peuvent eux aussi laisser ce travail de recalcul des secteurs au système d'exploitation de l'ATARI ST qui offre cette fonction. Nous en reparlerons plus longuement dans le chapitre 7.

Il existe d'autres principes d'organisation de fichiers bâtis sur ces techniques de base. Ainsi, on peut trier le fichier entier en fonction d'un champ donné (le nom par exemple), écrire tous les noms dans un fichier séparé avec pour chaque nom un index renvoyant vers l'enregistrement correspondant. On nomme ce type de fichier un "fichier-index". L'ensemble des deux fichiers constitue un fichier en "séquentiel indexé" pour lequel il existe des procédures de recherche très raffinées. Il existe des moyens très rapides pour retrouver un enregistrement et pour le lire.

2.1 STRUCTURE DES FICHIERS ET MODES D'ACCES

AVEC DIFFERENTS LANGAGES EVOLUES.

Le système d'exploitation d'un ordinateur offre les opérations de base pour le traitement de fichiers. Les langages évolués exploitent ces opérations pour réaliser leurs propres formes de fichiers. Comme nous l'avons indiqué, le système d'exploitation de l'ATARI ST nommé TOS, ou GEMDOS, permet de traiter des fichiers en RANDOM-ACCESS. Nous allons vous présenter ces fonctions du GEMDOS et leurs équivalents en langage évolué. Les programmes que nous vous donnons en BASIC, PASCAL, C et FORTRAN font exactement la même chose : ils créent et lisent un fichier séquentiel et un fichier à accès direct.

2.1.1 Les fonctions de GEMDOS : généralités.

Chaque fichier possède un nom que lui a donné l'utilisateur. Le nom de fichier tient sur 11 caractères au plus. Les 8 premiers caractères correspondent au nom de fichier proprement dit et sont séparés par un point des 3 derniers caractères qui forment l'extension et qui sont une information supplémentaire sur le fichier. Lorsqu'on utilise des compilateurs en langage évolué, c'est à dire des programmes qui traduisent un source en code exécutable, on peut définir l'extension des fichiers objets. En effet, on peut créer jusqu'à 4 fichiers intermédiaires entre le fichier source et le fichier exécutable. Ces 4 fichiers peuvent avoir le même nom mais doivent posséder une extension différente. On créera par exemple un fichier texte "test1.c" en C avec un éditeur ou un traitement de texte. Lorsqu'on compilera le fichier, on obtiendra un fichier nommé "test1.o" et un fichier "test1.prg".

GEMDOS dispose de la fonction CREATE (n° de fonction : \$3C) pour créer un nouveau fichier. Il faut indiquer à cette fonction le nom du fichier et un numéro de mode qui donne des informations sur le type du fichier. Lorsqu'on appelle Create, on crée sur le lecteur courant un fichier vide dans lequel on pourra inscrire des données. De nombreux langages évolués utilisent la fonction CREATE dans leur commande OPEN. Lorsqu'on utilise OPEN pour inscrire des données dans un fichier, la commande crée automatiquement un nouveau fichier s'il n'en existe aucun avec le nom indiqué.

Pour écrire dans un fichier, il faut utiliser la fonction WRITE (\$40) et lui indiquer le numéro de fichier utilisé par OPEN ou CREATE (le "handle"), le nombre de caractères à écrire et les caractères eux-mêmes bien entendu. Une fois que toutes les données ont été inscrites dans le fichier et si on doit par la suite pouvoir y accéder, il faut auparavant fermer le fichier. On utilise pour cela la fonction CLOSE (\$3E). Si on oublie d'appeler cette fonction, le fichier risque d'être perdu. En effet, la structure du fichier n'est alors pas correctement définie sur la disquette.

Une fois qu'on a créé le fichier avec CREATE, qu'on y a inscrit des données avec WRITE et qu'on l'a fermé avec CLOSE, il faudra l'ouvrir de nouveau avec OPEN pour pouvoir lire ce qu'il contient. Comme pour CREATE, il faut préciser le nom de fichier et un numéro de mode entre 0 et 2 pour la fonction OPEN (\$3D).

Si on inscrit la valeur 0 pour le numéro de mode, le fichier peut seulement être lu, c'est à dire que le numéro de fichier (handle) autorise la lecture exclusivement. Toute tentative d'écriture se solde par un message d'erreur. En mode 1, le fichier autorise seulement l'accès en écriture, en mode 2, le fichier peut être lu ou écrit. Il faut utiliser la fonction READ (\$3F) et lui indiquer comme pour la fonction WRITE, le numéro de fichier et le nombre de caractères à lire.

L'accès aux fichiers avec READ et WRITE s'effectue séquentiellement, c'est à dire que le système d'exploitation crée un pointeur pour chaque fichier qui est remis à zéro à chaque fois qu'on ouvre le fichier. Ce pointeur est positionné sur la donnée en cours de traitement.

Si on écrit par exemple 14 caractères dans un fichier à l'aide de WRITE, le système d'exploitation augmente ce pointeur de 14 positions de manière à ce que le prochain caractère envoyé vers le fichier soit ajouté à la suite des autres. L'utilisateur doit donc, soit définir un nombre de caractères déterminé pour chaque champ, soit inscrire un code de séparation entre chaque champ afin d'en marquer la fin.

Comme pour l'écriture, le pointeur est positionné sur le caractère courant lors de la lecture. On peut donc lire n'importe quel caractère mais il faudra lire tous les caractères précédents avant d'y avoir accès. La fonction GEMDOS LSEEK (\$42) permet le positionnement du pointeur sur n'importe quel caractère relativement au début du fichier, à la fin du fichier ou à la position courante du pointeur. La fonction LSEEK permet de programmer un fichier à accès direct si on emploie des champs de longueur définie, soit par exemple 13 caractères pour le nom, 64 caractères pour un enregistrement complet, etc... On saura ainsi de combien de caractères il faut déplacer le pointeur interne pour accéder à l'enregistrement ou au champ suivant.

Il manque encore trois fonctions GEMDOS très utiles pour la manipulation de fichiers :

SETDTA (\$1A) crée un tampon pour les fonctions SFIRST (\$4E) et SNEXT (\$4F), qui servent à lire tous les fichiers d'une disquette à partir du catalogue et à déterminer leur taille.

Après cette vue d'ensemble du système d'exploitation de l'ATARI ST, nous pouvons étudier certains langages évolués et voir la manière de manipuler des fichiers avec ces langages. Ces exemples représentent une introduction aux langages concernés mais ne constituent pas une gestion de fichier complète. Ils ont pour seul objectif de vous expliquer la création et la manipulation de fichiers. Il existe de nombreux ouvrages qui vous permettront d'apprendre ces langages et les différentes organisations de fichiers.

Après une étude "semi-théorique" des modes d'accès aux fichiers, vous trouverez au chapitre 2.6 un programme BASIC de gestion des données simple mais complet qui constituera une application pratique des connaissances acquises.

2.2 MODES D'ACCES AUX FICHIERS EN BASIC

Le ST BASIC livré avec l'ATARI ST offre l'accès séquentiel et l'accès direct. De plus, on pourra utiliser le programme avec GfA BASIC en transférant le fichier grâce au logiciel de conversion livré avec la disquette GfA BASIC.

Instructions BASIC

Pour créer un fichier sur disquette, il faut utiliser la fonction OPEN qui offre trois options. Voici la syntaxe de l'instruction :

```
OPEN "Mode", #numéro_de_fichier, "Nom du fichier",
longueur_d'enregistrement
```

Voici les options possibles pour le mode (en majuscule obligatoirement) :

```
"I"  = Fichier ouvert en lecture séquentielle
"O"  = Fichier ouvert en écriture séquentielle
"R"  = Fichier ouvert en accès direct.
```

Le numéro de fichier est un nombre variant entre 1 et 15. Le nom de fichier est composé de 8 caractères suivis d'un point puis de 3 caractères. La longueur de l'enregistrement n'a de signification que pour l'ouverture d'un fichier en accès direct (mode "R"), et doit être donnée en octets.

Contrairement à la fonction du système d'exploitation, il faut indiquer dès le départ s'il s'agit d'un fichier en accès séquentiel ou en accès direct.

L'utilisation des fichiers séquentiels est très limitée avec ce BASIC car il n'est pas possible d'ajouter des données une fois que le fichier a été fermé. Le seul moyen est d'utiliser un petit truc : si on a sauvegardé un fichier séquentiel avec 100 adresses et qu'on veut y ajouter tante Octavia, il faut lire les 100 adresses, les mémoriser, y ajouter la nouvelle adresse et sauvegarder les 101 adresses de nouveau.

Chaque OPEN "O" efface tout fichier déjà existant avec le nom donné et crée un nouveau fichier vide. Compte tenu de ces limitations rédhibitoires et étant donné que la taille maximale d'un fichier séquentiel dépend de la taille mémoire de l'ATARI ST (ce qui n'est pas gênant pour les petits fichiers), je n'insisterai pas sur les fichiers séquentiels en BASIC.

2.2.1 Les fichiers séquentiels en BASIC

Un fichier séquentiel peut contenir des chaînes de caractères (en ASCII) ou des chiffres. Les caractères spéciaux peuvent poser des problèmes car la fin des champs risque d'être introuvable. On utilisera par exemple la commande suivante pour ouvrir un tel fichier :

OPEN "O",#1,"TEST1.DAT"

Pour écrire des données dans ce fichier, il faut employer les instructions *WRITE#1* et *PRINT#1*. *WRITE* insère une virgule entre chaque donnée écrite et *PRINT* inscrit les données telles qu'elles apparaissent à l'écran (avec par exemple un espace après les virgules.).

PRINT# et *WRITE#* ont la même syntaxe :

PRINT#numéro_de_fichier,donnée[,donnée,...]

WRITE#numéro_de_fichier,donnée[,donnée,...]

On pourra créer le fichier "TEST1.DAT" de la manière suivante :

```
10 OPEN "O",#1,"A:TEST1.DAT"  
20 A$="JEAN"  
30 B$="DUPONT"  
40 FOR I=1 TO 10  
50 WRITE#1,A$  
60 WRITE#1,B$  
70 NEXT I  
80 CLOSE #1
```

Ce petit programme crée un fichier nommé "TEST1.DAT" sur la disquette se trouvant dans le lecteur A et y inscrit 10 fois les données "JEAN" et "DUPONT".

La fonction *WRITE* écrit le texte (chaîne de caractères) entre guillemets suivi du code \$0D (CR = Carriage Return) et \$0A (LF = Line Feed). Le caractère \$1A sert de code de fin de fichier (EOF = End Of File) en BASIC. Il permet au programmeur de déterminer la fin du fichier.

Il existe deux instructions pour lire un fichier séquentiel en ST BASIC. Ces deux commandes se différencient uniquement lors de la lecture des caractères spéciaux.

La fonction *INPUT#* ignore les espacements, les CR, les LF et les caractères spéciaux et lit tous les caractères normaux jusqu'à rencontrer le caractère de fin de ligne (EOL : End Of Line constitué de \$0A et \$0D, soit LF et CR), une virgule, le caractère EOF ou une fois atteint 255 caractères. La fonction *LINE INPUT#* lit tous les caractères du premier jusqu'au caractère de fin de ligne ou jusqu'à 254 caractères. Pour chaque fonction, il faut préciser une variable dans laquelle seront inscrites les données et le numéro du fichier. *INPUT #1, A\$* lit une chaîne de caractères dans le fichier ayant le numéro 1 et l'inscrit dans la variable A\$.

Le petit programme qui suit ouvre le fichier "TEST1.DAT" déjà existant et lit toutes les chaînes de caractères jusqu'à la fin (EOF = \$1A). La fonction *EOF* sert à reconnaître cette fin. Elle renvoie *TRUE* lorsque la fin a été rencontrée et *FALSE* lorsque ce n'est pas le cas.

```
10 OPEN "I",#1,"A:TEST1.DAT"  
20 IF EOF(1) GOTO 100  
30 INPUT #1,A$  
40 PRINT A$  
50 GOTO 20  
100 CLOSE #1
```

2.2.2 Les fichiers à accès direct en BASIC

La manipulation des fichiers à accès direct est mieux implémentée en ST BASIC que celle des fichiers séquentiels. Nous allons d'abord vous présenter quelques instructions de manipulation de fichiers car l'accès direct est plus difficile à utiliser que l'accès séquentiel.

L'ouverture et la création de fichiers se font sans différence notable. `OPEN "R",#1,"TEST2.DAT",65` ouvre le fichier `"TEST2.DAT"` en accès direct et définit une longueur d'enregistrements de 64 caractères (octets). Si vous accédez au fichier avec `GET#` et `PUT#`, ces accès se feront toujours par portions de 64 caractères.

Les seuls caractères autorisés sont les codes ASCII. Ainsi, toutes les valeurs qu'on inscrit dans un fichier à accès direct doivent auparavant être transformées en caractères. Ces caractères sont transformés en chiffres lors de la lecture. Mais n'ayez crainte : il existe de nombreuses fonctions BASIC pour cela.

La plupart du temps, on utilise plusieurs champs par enregistrement, c'est à dire un pour le nom, un pour le prénom, etc.. C'est la fonction `FIELD #` qui s'occupe de cette division de l'enregistrement en champs.

La commande

```
FIELD #1, 13 AS A$, 13 AS B$, 5 AS C$, 14 AS D$, 15 AS E$, 4 AS F$
```

réserve 13 caractères pour le champ #1 avec A\$ (prénom), 13 caractères avec B\$ (Nom), 5 caractères avec C\$ (code postal), 14 caractères avec D\$ (ville), 15 caractères avec E\$ (rue), 4 caractère avec F\$ (numéro du bâtiment). Il ne faut pas manipuler ces chaînes de caractères directement.

On doit passer par les fonctions *LSET* et *RSET*. *LSET A\$ = "Dupont"* inscrit la chaîne "Dupont" dans la variable A\$. Cette chaîne est ajustée à gauche, les caractères restants pour remplir le champ sont remplacés par des espaces : \$20 (le champ fait 13 caractères dans notre exemple). L'instruction *RSET A\$ = "Dupont"* inscrit "Dupont" dans la variable tampon comme *LSET* mais justifié à droite.

Si on veut écrire des chiffres dans un fichier à accès direct, il faut d'abord les transformer en caractères. On fait cela grâce aux fonctions *MKIS* et *MKSS* :

MKIS(nombre)

donne une chaîne de caractères sur deux octets
(variables INTEGER).

MKSS(nombre)

donne une chaîne de caractères sur quatre octets
(variables REAL).

MKDS(nombre)

donne une chaîne de caractères sur huit octets
(variables REAL en double précision).

On adapte donc les nombres dans une chaîne de caractères appropriée avant de les transférer vers le fichier à accès direct. On pourra les relire par la suite grâce à des fonctions d'adaptation dans l'autre sens (CVI, CVS, CVD).

Après avoir défini les variables tampons avec *FIELD*, les chaînes de caractères avec *LSET*, après avoir adapté les nombres avec *MKD\$*, *MKSS* (etc..) et les avoir inscrits dans les variables tampon grâce à *LSET*, on peut inscrire l'enregistrement complet dans un fichier avec une seule instruction : *PUT*. *PUT #5, 1* inscrit les données de la variable tampon dans le fichier numéro 5 en tant qu'enregistrement numéro 1.

Le petit programme BASIC qui suit crée un fichier à accès direct dont le nom est "TEST3.DAT" sur la disquette située dans le lecteur A, spécifie 6 champs pour la variable tampon, inscrit des valeurs dans cette variable et inscrit deux enregistrements dans le fichier à accès direct.

```
10 OPEN "R",#1,"A:TEST3.DAT",64
20 FIELD #1,13 as A$,13 as B$,5 as C$,14 as D$,15 as E$,4 as F$
30 LSET A$ = "JEAN"
40 LSET B$ = "Dupont"
50 a = 12345
60 LSET C$ = MKS$(a)
70 LSET D$ = "Nantes"
80 LSET E$ = "chemin chem"
90 b = 235
100 LSET F$ = MKS$(b)
110 PUT #1, 1
120 PUT #1, 2
130 CLOSE #1
```

En ligne 60, le nombre 12345 est transformé en chaîne CS sur 4 octets par la commande *MKS\$*.

Si on veut relire les données du fichier à accès séquentiel, on suit le même processus que pour l'écriture : on ouvre le fichier, on définit la variable tampon et on lit un enregistrement complet avec *GET #1*. On a accès directement aux enregistrements grâce à la variable tampon.

En ce qui concerne les nombres, il ne faut pas oublier de les transformer dans le sens contraire. En effet, les nombres sont sauvegardés sous forme de chaîne en ASCII. Le petit programme BASIC qui suit ouvre un fichier, lit tous ses enregistrements et les affiche à l'écran :

```
10 OPEN "R",#1,"A:TEST3.DAT",64
20 FIELD #1, 13 as A$,13 as B$,4 as C$,14 as D$,15 as E$,4 as F$
30 GET #1,1
40 PRINT A$,B$
50 PRINT CVS(C$),D$
60 PRINT E$,CVS(F$)
70 CLOSE 1
```

La taille des champs ne doit pas différer lors de l'écriture et lors de la lecture. En clair, si la variable *A\$* fait 13 caractères lors de l'écriture, il faut réserver 13 caractères dans la variable tampon qui reprendra le champ correspondant lors de la lecture.

2.3 LA GESTION DE FICHIERS EN PASCAL

La description des fonctions de fichiers en PASCAL que nous allons faire correspond au compilateur PASCAL ST-PASCAL+ de CCD. Ce compilateur est une très bonne implémentation du langage PASCAL qui dépasse de loin le standard PASCAL. ST-PASCAL+ dispose de fichiers à accès séquentiel et à accès direct.

Les fichiers sont définis par le type "file of" ou bien par le type prédéfini TEXT qui ne sert qu'aux fichiers séquentiels et qui correspond au type "packed array of char". Par exemple,

```
var dat: file of integer
```

décrit un fichier qui peut enregistrer des nombres du type INTEGER dont le pointeur est la variable dat qui pointe sur le caractère courant.

2.3.1 Le fichier séquentiel en PASCAL.

Après la déclaration d'une variable-fichier du type "file of", on peut créer un nouveau fichier avec la fonction rewrite (nom de fichier interne, 'nom de fichier externe'). Cette fonction correspond à l'instruction BASIC OPEN "O". Cette commande crée un fichier avec un nom donné et un nom externe. Le nom de fichier doit être déclaré comme variable du type "file of". On peut ensuite avoir accès au fichier à partir du nom de fichier ou de la variable tampon éventuellement définie par rewrite (même nom, mais suivi d'une flèche vers le haut).

Le nom de fichier interne représente le fichier à l'intérieur du programme PASCAL et le nom externe représente le même fichier sur mémoire de masse. Si on déclare par exemple le fichier "dat" avec :

```
var dat : file of integer;
```

et qu'on l'ouvre pour un accès séquentiel par rewrite (dat, "a:sfichier.dat"), une variable tampon dat[^] est automatiquement définie. Cette variable peut saisir des valeurs du type INTEGER et pointe pour l'instant sur le premier élément du fichier. De plus, le fichier "sfichier.dat" est créé sur le lecteur A et ouvert pour l'écriture.

Toutes les entrées/sorties suivantes se feront sur ce fichier.

Pour lire un fichier déjà existant, il faut l'ouvrir avec `reset` (nom de fichier interne, 'nom externe'). Cette instruction ouvre un fichier pour la lecture et charge le premier enregistrement dans la variable tampon. Si on essaie d'ouvrir un fichier inexistant, `EOF()` devient vrai.

La fonction `EOF(nom de fichier interne)` renvoie une valeur du type boolean (`TRUE,FALSE`). `EOF()` renvoie `TRUE` lorsque le pointeur est positionné sur la fin de fichier.

`EOL(variable-fichier)` est également du type boolean mais elle ne peut être utilisée que sur les fichiers du type "packed file of char" ou `TEXT`. Elle renvoie `TRUE` lorsqu'on atteint la fin d'une ligne.

L'accès aux données d'un fichier se fait avec `put` (nom de fichier interne) pour l'accès en écriture et `get` (nom de fichier interne) pour l'accès en lecture.

`put(dat)` inscrit la valeur de la variable tampon `dat^` dans le fichier. La variable tampon représente un pointeur sur le fichier qui est remis à zéro après chaque `rewrite` ou `reset` et qui est augmenté de un pour pointer sur l'élément suivant après chaque `get` ou `put`. Après l'ouverture d'un fichier pour la lecture avec `reset` (`dat,"nom"`), la variable tampon `dat^` prend pour valeur le premier élément du fichier. Si on refait un `get` (`dat`), le pointeur est augmenté de 1 et la variable `dat^` prend pour valeur l'élément sur lequel est positionné le pointeur. La fin du fichier se reconnaît à la fonction `eof` (variable fichier) qui renvoie une valeur en boolean (`TRUE,FALSE`). Avant l'accès avec `GET`, il faut tester si on a atteint la fin de fichier car `get` augmente le pointeur de un et met l'élément suivant dans la variable tampon. Avec des fichiers du type `TEXT`, on a la possibilité de reconnaître la fin de la ligne grâce à la fonction `eol` (variable fichier). Cette fonction renvoie également une valeur du type boolean.

Tous les types de données en `PASCAL` peuvent être des éléments de fichiers. Cela est vrai également pour les `RECORDS`. Dès qu'on a ouvert un fichier avec `rewrite`, la variable tampon peut prendre une valeur qui sera inscrite dans le fichier grâce à `put`.

Dans les fichiers-texte c'est à dire les fichiers du type "file of char" (TEXT) on peut résumer en une seule instruction les opérations d'inscription d'une valeur dans la variable tampon par `dat^:=valeur` et d'écriture de cette valeur dans le fichier par `put (dat)`. Cette instruction est `write (dat, valeur)`. De même, la commande `read (dat, valeur)` lit une valeur dans un fichier et remplace les commandes `valeur := dat^` et `get (dat)`.

Le petit programme PASCAL qui suit crée un fichier sur disquette dans le lecteur A et y inscrit 20 chaînes de caractères. En CCD PASCAL, l'instruction `string[20]` définit une variable du type "packed array of char" qui peut contenir 21 caractères. Le compilateur PASCAL garde la valeur de cette chaîne en l'inscrivant au tout début c'est-à-dire dans le caractère 0 de la chaîne.

(* écriture d'un fichier séquentiel en PASCAL U.B. 9.86 *)

```
program sfichier ;

var dat1    : file of string[20] ;
    t1, t2  : string [20] ;
    i       : integer ;

begin
  rewrite (dat1, 'a:seqfich.dat');
  t1:='jean' ;
  t2:='Lacoste' ;

  for i:=1 to 10 do
    begin
      dat1^:= t1;
      put (dat1);
      dat1^:= t2;
      put (dat1);
    end; (* boucle for *)

  end. (* programme *)
```

Si vous analysez le fichier "seqfich.dat" avec le moniteur de disque présenté dans le chapitre 7, vous reconnaîtrez le type d'organisation d'un fichier séquentiel en PASCAL avec les variables `string` (21 caractères par chaîne, longueur de la chaîne au début).

Voici un petit programme pour lire le fichier crée :

(* lecture d'un fichier séquentiel en PASCAL U.B. 9.86 *)

```
program lirefichier ;

var   dat1      : file of string[20] ;
      t1, t2    : string[20] ;
      i         : integer ;

begin
  writeln (' lecture de fichier ');
  reset (dat1, 'a:seqfich.dat');

  while not eof(dat1) do
    begin
      t1 := dat1^;
      get (dat1);
      writeln (t1);
    end; (* boucle while *)
  writeln;
  writeln (' presser la touche return ');
  readln (t2);

end. (* programme *)
```

Après avoir ouvert le fichier par `reset (dat1, 'a:seqfich.dat')`, la variable tampon `dat1^` prend pour valeur le premier élément du fichier. Ainsi, on peut utiliser cette variable dès l'ouverture du fichier. Bien entendu, cette variable doit être du même type que la variable tampon définie avec la déclaration de la variable fichier, sous peine d'erreurs. D'autre part, il ne faut pas essayer de lire des données après la fin de fichier. La fonction `eof (dat1)` teste si la fin de fichier a été atteinte et dans ce cas, la boucle de lecture est quittée.

En PASCAL comme en BASIC, il n'existe aucune possibilité d'ajouter des données à la suite d'un fichier déjà existant. Si vous voulez étendre un fichier déjà existant, la seule possibilité est de lire le fichier en entier, d'ajouter le nouvel élément et de créer un nouveau fichier.

La création et l'accès aux fichiers d'un autre type (file of integer, file of real) se fait de la même manière que dans l'exemple précédent.

2.3.2 Fichiers à accès aléatoire en PASCAL

La création et l'ouverture de fichiers à accès aléatoire s'effectuent avec les mêmes instructions que pour le fichier séquentiel (rewrite et reset). L'accès aux données se fait de la même manière. Cependant, il y a un autre paramètre pour les fonctions get et put, c'est à dire le numéro de l'enregistrement qui doit être écrit ou lu. La numérotation des enregistrements commence à 0 et il faut d'abord créer tous les enregistrements entre 0 et le nombre maximum. Si le dernier enregistrement porte le numéro 8 par exemple, on ne peut pas créer d'enregistrement ayant le numéro 10. Il faut d'abord écrire un enregistrement numéro 9. Voici un petit programme qui montre la flexibilité de ce type de fichiers. Il crée un fichier d'adresses dans lequel est inscrit 10 fois la même adresse.

(* écriture d'un fichier à accès direct en PASCAL U.B. 9.86 *)

```
program ranfich ;
```

```
type adres =
```

```
  record
```

```
    prenom : string[12];
```

```
    nom    : string[12];
```

```
    code   : integer;
```

```
    ville  : string[14];
```

```
    rue    : string[14];
```

```
    numero : integer;  end; (* record *)
```

```
var  dat1 : file of adres;
```

```
    t1, t2 : adres;
```

```
    i      : integer;
```

```
begin
```

```
  rewrite (dat1, 'a:random1.dat');
```

```
  t1.prenom := 'Olivier';
```

```
  t1.nom := 'Polette';
```

```
  t1.code := 2222;
```

```
tl.ville := 'Nantes';  
tl.rue := 'duchemin';  
tl.numero := 245;
```

```
for i:=0 to 9 do  
begin  
    dat1^ := tl;  
    put (dat1,i);  
end; (* boucle for *)
```

```
end. (* programme *)
```

L'instruction `dat1^ := tl` écrit (en CCD PASCAL) la structure de l'adresse (avec le nom, le prénom, etc...) dans la variable tampon afin de l'écrire dans le fichier en tant qu'enregistrement numéro `i` avec `put (dat1,i)`.

Comme vous pouvez le constater, le nombre de caractères de chaque chaîne est mémorisé avant le premier caractère de la chaîne considérée. Les Integer sont sauvegardés en tant que nombre décimaux sur 2 octets. PASCAL utilise le caractère `$F5` comme caractère de fin de fichier.

2.4 MODES D'ACCES AUX FICHIERS EN C.

Le langage C est pour ainsi dire la langue maternelle de l'ATARI ST. Une grande partie de son système d'exploitation a été écrite dans ce langage. Il n'est donc pas étonnant de rencontrer par la suite (partiellement modifiées) les fonctions GEMDOS décrites dans l'introduction à ce chapitre.

Le langage C est le moins pratique du point de vue utilisateur. En effet, beaucoup de fonctions doivent être écrites par le programmeur. Il en va de même pour les fonctions de gestion de fichiers. Tous les compilateurs C offrent les fonctions de base décrites par Kernighan & Ritchie dans le fichier include "STDIO.H". Pour utiliser ces fonctions, il faut les intégrer au début du programme avec l'instruction `#include <stdio.h>`.

Le gros ennui pour les débutants en C, outre les abréviations illisibles (&,<=,>||), est de devoir utiliser le compilateur C de Digital Research (tout au moins les premières versions). Un débutant inexpérimenté se demandera toujours devant une erreur : *"Est-ce une erreur de ma part ou une bogue du compilateur ?"*. Il arrive d'ailleurs que le programme se plante durant ou après la compilation. Pour cette raison, nous avons compilé les petits programmes que nous vous présentons avec le compilateur LATTICE C de Metacomco. L'adaptation à d'autres compilateurs ne devrait pas poser de problèmes car nous utilisons seulement les fonctions standards de la bibliothèque STDIO.H (les programmes fonctionnent sans modifications avec MEGAMAX).

La communication avec les fichiers se fait au travers d'une structure de données du type FILE. Cette commande et les fonctions d'accès aux fichiers sont incluses dans la bibliothèque STDIO.H. Les différents paramètres du fichier comme l'adresse du tampon ou le pointeur sur ce tampon sont mémorisés. Voici un rapide résumé des fonctions d'accès avec le type de leurs paramètres.

pointer = fopen (nom, mode)

FILE *fopen()

FILE *pointer

char *nom

char *mode

Les modes possibles sont :

"w" : création et ouverture d'un fichier pour l'écriture.

"a" : ouverture d'un fichier pour ajouter des données.

"r" : ouverture d'un fichier pour lire des données.

Outre ceux-ci, il existe d'autres modes qui ont différentes fonctions selon les compilateurs et qui ne présentent pas grand intérêt pour ce que nous allons faire.

La fonction ouvre un fichier pour le mode d'accès prévu. Lorsque survient une erreur, pointer = zéro, sinon, pointer prend pour valeur le pointeur sur le fichier.

code = fclose (pointer)

int code
FILE *pointer

ferme le fichier sur lequel pointe pointer.

fprintf (pointer, format, arguments)

FILE *pointer
char *format
char *arguments

écrit dans un fichier autant d'arguments (chaînes de caractères) que désirés, avec le format défini par le paramètre 'format'. Le paramètre format correspond à celui de l'instruction printf normale.

code = fscanf (pointer, format, chpointer)

FILE *pointer
char *format
char *chpointer
int code

lit une chaîne de caractères dans le fichier spécifié par pointer et l'inscrit dans la variable chpointer avec le format spécifié. Les options format sont les mêmes que celles de la fonction scanf.

code = fputs (buffer, pointer)

FILE *pointer
char *buffer
int code

écrit la chaîne de caractères sur laquelle pointe buffer (jusqu'à l'octet nul) dans le fichier sur lequel pointe pointer. Si une erreur survient, code = EOF. L'octet nul qui termine toute chaîne de caractères en C n'est pas écrit dans le fichier. Il est remplacé par un NEWLINE.

code = fgets (buffer, nombre, pointer)

FILE *pointer
char *chpoint
char *buffer
int nombre
int code

lit le nombre de caractères du fichier sur lequel pointe pointer, et l'inscrit dans le tampon sur lequel pointe buffer. La lecture est interrompue lorsque survient un caractère EOL (End Of Line). Un octet nul est ajouté à la chaîne de caractères et le pointeur sur le tampon de chpoint est renvoyé. Après un accès sans erreurs, cvhpoint pointe sur buffer, sinon chpoint contient un 0 ce qui correspond à un nul en C.

code = fputc (chpoint, pointer)

FILE *pointer
char *chpoint
int code

écrit un seul caractère, celui sur lequel pointe chpoint, dans le fichier sur lequel pointe pointer. Après une erreur, code = EOF sinon, cette variable prend pour valeur le code du caractère écrit.

code = fgetc (pointer)

FILE *pointer
int code

lit un caractère dans le fichier sur lequel pointe pointer. Le code du caractère lu est inscrit dans code ou dans EOF si la fin du fichier est atteinte.

code = fseek (pointer, position, mode)

FILE *pointer
long position
int mode
int code

positionne le pointeur du fichier (sur lequel pointe pointer) sur une nouvelle valeur. Le paramètre mode spécifie la nouvelle position du pointeur et peut prendre les valeurs suivantes :

- 0 : nouvelle position relativement au début du fichier.
- 1 : nouvelle position relativement à la position courante.
- 2 : nouvelle position relativement à la fin du fichier.

2.4.1 Les fichiers séquentiels en C

Le programme en C qui suit ouvre le fichier 'seqfich.dat' pour l'écriture et y inscrit 10 fois "Jean Dupont".

/ écriture d'un fichier séquentiel en C U.B. 9.86. */*

```
#include <math.h>
#include <stdio.h>
```

```
main ()
```

```
{
```

```
    int i, k ;
    FILE *dat1, *fopen()
```

```
    char *t1 = "Jean" ;
    char *t2 = "Dupont" ;
```

```
    dat1 = fopen ("a:seqfich.dat","w") ;
```

```
    for ( k=0; k<10; k++)
```

```
    {
        fprintf (dat1, "%13s", t1);
        fprintf (dat1, "%13s", t2);
```

```
    } /* fin de la boucle for */
```

```
    i = fclose (dat1);
    printf (" pressez une touche \n ");
```

```
    getchar();
```

```
} /* fin de la fonction main */
```

Pour lire le fichier, voici un petit programme qui affiche tout le contenu du fichier à l'écran :

```
/* lecture d'un fichier séquentiel en C U.B. 9.86. */
```

```
#include <stdio.h>
```

```
main()
```

```
{
```

```
    int i, k ;
```

```
    FILE *dat1, *fopen() ;
```

```
    char placel [14] ;
```

```
    char *p ;
```

```
    dat1 = fopen("a:seqfich.dat","r") ;
```

```
    while (p = fgets (placel, 14, dat1) != NULL)
```

```
    {
```

```
        printf ("%s\n", placel);
```

```
    } /* fin de la boucle while */
```

```
    i = fclose (dat1) ;
```

```
    printf ("\n\n") ;
```

```
    printf (" pressez une touche \n ") ;
```

```
    getchar () ;
```

```
} /* fin de la fonction main */
```

2.4.2 Les fichiers à accès direct en C.

Pour obtenir un accès direct en C, il faut utiliser la fonction `fseek()` qui autorise le positionnement du pointeur sur un caractère du fichier. Chaque enregistrement a une longueur définie grâce à l'écriture formatée de `fprintf()`, par exemple 13 caractères pour le prénom, etc... Chaque enregistrement a une longueur définie (dans ce fichier, 64 caractères).

Pour écrire le i ème enregistrement, il suffit de multiplier la longueur d'un enregistrement par le numéro de l'enregistrement désiré. Il faut ensuite positionner le pointeur de fichier sur la valeur obtenue. On peut alors traiter l'enregistrement désiré. En C, la numérotation des enregistrements commence à zéro.

/ écriture d'un fichier à accès direct en C U.B. 9.86 */*

```
#include <math.h>
```

```
#include <stdio.h>
```

```
char *prenom = "Jean" ;  
char *nom = "Dupont" ;  
char *ville = "Nantes" ;  
char *rue = "duchemin" ;  
int code = 47000 ;  
int numero = 264 ;
```

```
main ()
```

```
{  
    int i, k ;  
    FILE *dat1, *fopen() ;  
  
    dat1 = fopen("a:random2.dat","w");  
  
    for (k=1; k<11; k++);  
    {  
        fprintf (dat1,"%13s",prenom);  
        fprintf (dat1,"%13s",nom);  
        fprintf (dat1,"%4d",code);  
        fprintf (dat1,"%15s",ville);  
        fprintf (dat1,"%15s",rue);  
        fprintf (dat1,"%4d",numero);  
  
    } /* fin de la boucle for */  
  
    i = fclose (dat1);  
    printf (" pressez une touche\n ");  
  
    getchar();  
} /* fin de la fonction main */
```


Le programme qui suit lit toutes les données du fichier et les affiche à l'écran avec le numéro d'enregistrement et la position relative dans le fichier.

/ lecture d'un fichier à accès direct en C U.B. 9.86 */*

```
#include <math.h>
#include <stdio.h>

#define longueur 64L

main ()
{
    int k, il, i ;
    FILE *dat1, *fopen() ;
    long pos ;

    char placel[80], *p ;

    dat1 = fopen ("a:random2.dat","r") ;

    k=0;
    pos = k*longueur ;

    while ((i = fgetc(dat1)) != EOF)
    {
        i = fseek(dat1,pos,0) ;

        printf(" numero d'enregistrement = %8d\n",k) ;
        printf(" position de l'octet dans le fichier = %8d\n", pos) ;
        printf("\n") ;

        p = fgets(placel,14,dat1);
        printf(" prenom = %s\n",placel) ;

        p = fgets(placel,14,dat1);
        printf(" nom = %s\n",placel);

        p = fgets(placel,5,dat1);
        il = atoi (placel) ;
        printf(" code postal = %8d\n",il);
```

```
p = fgets(placel,16,dat1);
printf(" ville = %S\n",placel) ;

p = fgets(placel,16,dat1) ;
printf(" rue = %s\n", placel);

p = fgets(placel,5,dat1);
il = atoi(placel) ;
printf(" numero du batiment = %8d\n",il) ;

k+=1 ;
pos = k*longueur ;

printf("*****\n\n");

} /* fin de la boucle while */

i = fclose (dat1) ;
printf("\n\n");

printf (" pressez une touche\n ") ;

getchar() ;

} /* fin de la fonction main */
```

2.5 LA GESTION DE FICHIERS EN FORTRAN

Toutes les explications que nous allons donner sur ce langage sont basées sur le compilateur PRO FORTRAN 77 de PROSPERO qui est distribué par la société FOCUS. Comme le CCD PASCAL, ce FORTRAN autorise les fichiers séquentiels et les fichiers à accès direct. L'implémentation sur ATARI ST est très convenable car toutes les définitions du standard 77 ont été conservées. Mais en ce qui concerne la rapidité également, ce compilateur dépasse de loin (tout au moins pour les calculs mathématiques) les compilateurs C et le compilateur CCD PASCAL.

2.5.1 Les fichiers séquentiels en FORTRAN

Pour ouvrir ou créer un fichier séquentiel, on utilise la fonction `open` qui possède de nombreux paramètres optionnels. `Open (5, file = 'a:fdatl.dat')` ouvre le fichier "fdatl.dat" sur le canal numéro 5 et le lecteur A. Si ce fichier n'existe pas, il est créé.

Pour écrire dans ce fichier, on peut utiliser les instructions `read` et `write` avec des paramètres optionnels. `write(5) 'truc'` écrit 'truc' dans le fichier sur le canal 5. La sortie avec `write` n'exploite pas toutes les possibilités de FORTRAN qui dépassent de loin le cadre de cet ouvrage.

Voici notre programme en FORTRAN qui crée un fichier séquentiel dans lequel il inscrit 10 fois le nom "Jean Dupont".

```

program seq1

character*13 nom, prenom

prenom = 'Jean'
nom = 'Dupont'

open (2, file = 'a:fsequel.dat', form = 'unformatted')

do 100 n = 1,10
write (2) prenom
write (2) nom
100 continue
close (2)
end

```

Pour lire ce fichier séquentiel, on utilise le programme suivant :

```

program seq2
character*2 t1
character*13 text

open (2, file='a:fsequel.dat', form='unformatted', status='old')

100 continue
read (2,end=200) text

```

```
      write (*,*) text  
      goto 100  
  
200  continue  
      close (2)  
      end
```

2.5.2 Les fichiers à accès direct en FORTRAN.

Voici notre programme standard pour les fichiers à accès direct :

C écriture d'un fichier à accès direct en FORTRAN U.B. 9.86

```
      program randl  
      integer*5 code, numero  
  
      character*13 nom, prenom  
      character*15 rue, ville  
  
      prenom = 'Jean'  
      nom = 'Dupont'  
      code = 44000  
      ville = 'Nantes'  
      rue = 'duchemin'  
      numero = 264  
  
      open (2,file = 'a:\frandl.dat', recl = 64, access='direct')  
      do 100 n = 1,10  
        write (2,rec= n) prenom, nom, code, ville, rue, numero  
100  continue  
      close (2)  
      end
```

Voici un petit programme en FORTRAN pour lire ce fichier :

C lecture d'un fichier à accès direct en FORTRAN U.B. 9.86

```
      program randl  
      integer*5 code, numero, stat  
  
      character*13 nom, prenom
```

```

character*15 rue, ville

open (2, file = 'a:\frand1.dat', recl = 64, access='direct',
status = 'old')

n = 1
10 continue

read (2,rec=n, iostat = stat ) prenom, nom, code, ville, rue,
numero
if (stat .eq. 0) then

write (*,*) 'numero enregistrement :',n
write (*,*)
write (*,*) 'prenom :', prenom
write (*,*) 'nom :', nom
write (*,*(a,i6)) 'numero du batiment :', numero
write (*,*)
write (*,*)
n = n+1
goto 10

else
write (*,*)
write (*,*)
write (*,*) 'pressez une touche'
close (2)
endif
end

```

2.6 UNE BASE DE DONNEES SIMPLIFIEE

Pour mettre en pratique toute cette théorie, nous allons appliquer nos connaissances avec une petite base de données. Ce programme n'est pas destiné à la gestion de stock d'un grand magasin, mais il suffit largement pour la gestion de numéros de téléphone ou d'une collection de disques.

Le programme est écrit en BASIC avec l'interpréteur ST BASIC. Ce BASIC est rempli de bogues et il sera préférable d'adapter le programme en GfA BASIC. Cependant, il fonctionne tel quel avec le ST BASIC.

Avant de créer un tel programme, il faut savoir quelles doivent être les possibilités d'une gestion de base de données. Nous avons inclu les fonctions les plus importantes dans notre programme :

- Création d'une nouvelle base de données.
- Saisie de nouvelles données ou correction des anciens enregistrements.
- Chargement d'une base de données déjà existante sur disquette.
- Affichage des données à l'écran ou sur imprimante.
- Recherche par critères.
- Tri des données selon un champ déterminé.
- Interruption du programme.

Ces fonctions sont disponibles à partir d'un menu qui est affiché à l'écran. Il suffit de saisir le numéro de la fonction et de confirmer par RETURN.

Avant d'étudier les fonctions de manière plus approfondie, voyons le programme :

```
10   '### Mini-gestion de données S.D. ###
20   dim d$(5), i$(5), l(5), p$(500), r(500)
30   for i=1 to 500: r(i)=i: next i
40   for i=1 to 5: d$(i)=space$(100)
50   i$(i)="": next i
60   start:
70   fullw 2: clearw 2: gotoxy 0,0
80   ? " Mini-Gestion de données du : livre du lecteur de disquette ST S
.D."
```



```

90  ? : ? d ; " Enregistrements dans le fichier " ; f$
100 for i=1 to 5
110 gotoxy 22,1+i: ? i ; " ) " ; i$(i)
120 next i
130 if so then gotoxy 21,1+so: ? ">"
140 gotoxy 0,5
150 ? : ? "1) Création d'une base de données"
160 ? "2) Saisie des données"
170 ? "3) Chargement d'une base de données"
180 ? "4) Tri des données"
190 ? "5) Recherche"
200 ? "6) Impression des données"
210 ? "7) Fin"
220 ? : input "Votre choix " ; w
230 on w gosub creer,saisie,charger,trier,rechercher,impression,fin
240 goto start
250 '
260 '## Creation d'une base de données ##
270 creer:
280 ? " ## creation de la base de données : 500 enregistrements de 5 champs##"
290 sum=0
300 ? : for i=1 to 5
310 ? i ; ". nom du champ, Longueur ";
320 input i$(i),l(i)
330 sum=sum+l(i)
340 next i
350 ? : input "OK " ; o$
360 if o$="n" or o$="N" then creer
370 gosub getfn
380 open "D",#1,f$
390 for i=1 to 5
400 print#1,i$(i)
410 print#1,l(i)
420 d$(i)=space$(l(i))
430 next i
440 close #1
450 open "R",#1,f$d,sum
460 field #1, l(1) as d$(1), l(2) as d$(2), l(3) as d$(3),
    l(4) as d$(4), l(5) as d$(5)

```

```
470 d=0
480 return
490 '
500 '## Saisie des données ##
510 saisie:
520 clearw 2: gotoxy 0,0: ? " ## Saisie des données ##
530 ? d;" Enregistrements mémorisés"
540 gotoxy 0,3: ? "Numéro ";d+1
550 gotoxy 0,3: input "Numéro ";d$
560 if len(d$)>0 then d1=val(d$) else d1=d+1
570 if d1=0 then return
580 if d1>d+1 then saisie
590 if d1<d+1 then gotoxy 0,5: o$="b": gosub impression1
600 for i=1 to 5
610 gotoxy 0,4+i
620 ?i$(i);: gotoxy 20,4+i
630 input d$
640 if len(d$)>0 then lset d$(i)=d$
650 next i
660 ?: input "OK (o/n) ";o$
670 if o$="n" or o$="N" then saisie
680 if d1=d+1 then d=d+1
690 put #1,r(d1)
700 goto saisie
710 '
720 '## Chargement de la base de données ##
730 charger:
740 gosub getfn
750 close #1
760 sum=0
770 open "I",#1,f,$
780 for i=1 to 5
790 input#1,i$(i)
800 input#1,l(i)
810 sum=sum+l(i)
820 d$(i)=space$(l(i))
830 next i
840 close #1
850 open "R",#1,f,d$,sum
860 field #1, 1(1) as d$(1), 1(2) as d$(2), 1(3) as d$(3),
```

```
      l(4) as d$(4), l(5) as d$(5)
870  d=0
880  while not eof(1)
890  d=d+1
900  get #1,d
910  wend
920  return
930  '
940  '## Impression des données ##
950  impression:
960  if d=0 then ? "Aucune donnée n'est mémorisée !": goto waitkey
970  ? " ## Impression des données ##"
980  input "E)cran ou I)mpimante ";o$
990  for d1=1 to d
1000 gosub impression1
1010 if o$="I" or o$="i" then lprint else ?
1020 next d1
1030 waitkey:
1040 ?: input "-Veuillez presser 'Return' svp-",w$
1050 return
1060 impression1:
1070 get #1,r(d1)
1080 for j=1 to 5
1090 if o$="i" or o$="I" then lprint i$(j),d$(j)
      else ? i$(j),d$(j)
1100 next j
1110 return
1120 '
1130 '## Recherche ##
1140 rechercher:
1150 if d=0 then ? "Aucune donnée n'est mémorisée !": goto waitkey
1160 ?: input "Numéro du champ, Texte ";f,t$
1170 for d1=1 to d
1180 get #1,d1
1190 if instr(d$(f),t$) then gosub impression1: ?
1200 next d1
1210 goto waitkey
1220 '
1230 '## Tri ##
1240 trier:
```

```
1250 if d=0 then ? "Aucune donnée n'est mémorisée !": goto waitkey
1260 ?: input " Tri selon quel champ ";so
1270 if so=0 or so>5 then return
1280 for i=1 to d
1290 get #1,i
1300 p$(i)=d$(so)
1310 next i
1320 for i=1 to d
1330 for j=i to d
1340 if p$(r(i))>p$(r(j)) then swap r(i),r(j)
1350 next j
1360 next i
1370 return
1380 '
1390 '## Fin ##
1400 fin:
1410 close #1
1420 ?: ? "### Fin. Tchao! ###"
1430 end
1440 '
1450 '## Sous-programmes ##
1460 getfn:
1470 ?: input "Nom du fichier ";f$
1480 fi$=f$+".idx"
1490 fd$=f$+".dat"
1500 return
```

Voici maintenant les différentes fonctions :

1) Création d'une base de données

Après avoir appelé cette fonction, le programme vous demande de saisir deux données cinq fois de suite. Il faut inscrire le nom de l'enregistrement et (séparé du nom par une virgule) la longueur maximale de cette donnée en nombre de caractères. Pour une gestion d'adresses, on peut saisir par exemple :

Nom,15
Prenom,12
Ville,16
Rue,16
Tel,11

Une fois qu'on a saisi ces données, le programme vous demande "OK ?" pour confirmation. Dès que les données sont bien saisies, entrez simplement O (O pour 'oui').

Ensuite, le programme vous demande le nom du fichier dans lequel sera enregistrée la base de données. On peut saisir le nom du lecteur : A:test. Il ne faut pas donner d'extension car le programme crée deux fichiers du même nom mais avec une extension différente. Vous trouverez par la suite deux fichiers sur la disquette : un avec l'extension .IDX qui contient les noms et les longueurs des champs et un avec l'extension .DAT qui contient les enregistrements proprement dits.

Une fois que les données ont été saisies et que le fichier est créé sur disquette, vous revenez au menu principal.

2) Saisie de données

Lorsque vous avez choisi cette fonction, vous obtenez le nombre d'enregistrements déjà créés et le programme vous demande le numéro de l'enregistrement à modifier. Le numéro de l'enregistrement vide suivant est inscrit directement après le point d'interrogation. Ainsi, il vous suffit de presser RETURN pour traiter cet enregistrement.

Si vous voulez modifier un enregistrement, saisissez le numéro correspondant. Le programme affiche immédiatement le contenu de l'enregistrement et un point d'interrogation. Vous pouvez commencer à saisir l'enregistrement suivant. Si vous ne voulez pas modifier l'enregistrement en question, pressez simplement RETURN.

La saisie de toutes les autres données se fait de la même manière. Si vous voulez quitter la saisie, il suffit d'inscrire un 0 après la question vous demandant le numéro de l'enregistrement. Vous revenez alors immédiatement au menu principal.

3) Chargement d'une base de données

Cette option vous demande tout d'abord le nom de la base de données à charger. Vous pouvez également inscrire le nom du lecteur avant le nom du fichier. Dès que votre base de données est chargée, le programme affiche le menu principal avec le nom du fichier, le nombre d'enregistrements et les noms des champs avec leurs numéros.

4) Tri des données

Si vous voulez trier les enregistrements selon un champ donné, sélectionnez cette fonction. Le programme vous demande le numéro du champ selon lequel le tri doit être effectué. Ainsi, vous pouvez trier votre fichier d'adresse en fonction du nom, l'imprimer, puis le trier en fonction de la ville, l'imprimer, etc...

La fonction de tri ne dispose d'aucune option d'impression. Si vous voulez retrouver le champ selon lequel est effectué le tri, revenez au menu principal : c'est le champ devant lequel est inscrit un '>'.
>

5) Recherche

Cette fonction vous demande de saisir un numéro de champ suivi d'une chaîne de caractères. Si vous voulez imprimer toutes les adresses de Nantes par exemple, saisissez '3,Nantes'. Dans ce cas, tous les champs correspondants au numéro sélectionné sont analysés et le programme vous affiche les enregistrements dont le champ choisi contient la chaîne de caractère indiquée. Vous pouvez également inscrire comme critère de recherche la chaîne 'Nan' dans notre exemple.

Ce programme n'étant pas une gestion de fichier très puissante, et après un tri, la recherche ne donne pas toujours le bon résultat.

6) Impression de données

Cette fonction autorise l'impression de tous les enregistrements à l'écran ou sur l'imprimante. Le programme vous demande quel périphérique vous voulez utiliser. Répondez 'I' pour l'imprimante. Dans tous les autres cas, l'impression s'effectuera sur l'écran.

L'impression est faite dans l'ordre dans lequel les enregistrements ont été saisis, à moins que vous ayez utilisé la fonction de tri auparavant.

7) Fin

Cette option ferme le canal de données (CLOSE#1) et vous ramène à la fenêtre de commandes.

Comme vous l'avez constaté, ce programme exploite à la fois l'accès séquentiel et l'accès direct. Les noms de champ et la longueur des champs sont sauvegardés séquentiellement (nom.IDX) et les enregistrements sont sauvegardés dans un fichier à accès direct (nom.DAT). Bien entendu, avec une petite base de données, on peut sauvegarder les données séquentiellement et toutes les charger en mémoire (Cela est possible avec l'ATARI ST qui dispose d'une importante mémoire centrale). Mais cela perd du temps en chargement et ne fonctionne que si l'on n'oublie pas de sauvegarder les données après les avoir traitées (ce qui peut arriver !).

De plus, nous avons utilisé ce système afin de vous montrer l'emploi de différentes formes de fichiers. J'espère que ce programme vous permettra de créer des bases de données qui répondront à vos désirs.

3. Structure des données

On prend une grande masse de données et on la copie sur disquette : cela paraît très simple mais lorsqu'on analyse plus précisément ce processus, on se rend compte de certains problèmes.

Le gestion de la disquette doit être faite de telle manière qu'on puisse retrouver les données à tout moment. Pour cela il faut les préparer. Cela ne concerne pas l'utilisateur mais le système d'exploitation et le hardware de l'ordinateur (voire le lecteur de disquettes ou le disque dur) doivent le prendre en charge.

La disquette doit en premier lieu être formatée. Dans cette opération, la surface de la disquette est divisée en plusieurs secteurs dont la position est définie par le format utilisé.

Ce format doit être également défini pour l'ordinateur car il doit pouvoir travailler avec différents formats. Dans ce cas, le nombre de faces de la disquette est aussi important que le nombre de secteurs et leur taille. Cette information est contenue dans le boot-sector que nous allons analyser.

Par la suite, il faut affecter et marquer tous les secteurs correspondant à un fichier (ou un programme) sur la disquette. Cette information se trouve dans le catalogue de la disquette et dans la FAT, "File Allocation Table". Nous en parlerons également dans ce chapitre.

Commençons donc par le commencement : le formatage.

3.1 LE FORMAT DES DISQUETTES

Lorsqu'on formate une disquette, elle est divisée en plusieurs parties. Cette division générale s'appelle la création de pistes. Ces pistes se présentent comme des cercles concentriques et sont numérotés de l'extérieur vers l'intérieur. Une disquette formatée normalement contient 80 pistes (ou Tracks) numérotés de 0 à 79. Mais on peut formater jusqu'à 82 pistes. Cependant, en raison du manque de sécurité des pistes 80 à 82 qui réduisent d'autant la surface utilisable, le formatage standard ne les utilise pas. Elles peuvent être utilisées si on les formate convenablement.

Les pistes sont alors divisées en secteurs qui représentent chacun une portion de cercle. Ces secteurs sont regroupés eux-mêmes en clusters à raison de deux secteurs par cluster. La signification de ces clusters n'étant pas indispensable, nous n'étudierons que les secteurs.

Dans le format normal, chaque piste contient 9 secteurs de 512 octets. Une disquette simple face contient donc $80 \times 9 \times 512 = 368640$ octets. Cela ne correspond pas au nombre de données sauvegardées sur disquette. Lors du formatage, certaines données sont écrites pour chaque piste et pour chaque secteur.

Ces données sont utilisées par le contrôleur de disquettes (le circuit gérant les floppys) pour retrouver le bon secteur dans une piste. Voyons la structure d'une piste normale :

Nombre	Octets	Remarque
60	\$4E	début de la piste
<i>Par secteur</i>		
12	\$00	
3	\$F5	prennent pour valeur \$A1
1	\$FE	ID adress mark
1	piste	numéro de piste de 0 à 79
1	face	numéro de face 0 ou 1
1	secteur	numéro de secteur de 1 à 9
1	\$02	*\$100=512 octets par secteur
1	\$F7	Ecrire la somme-test CRC (prennent 2 octets)
22	\$4E	octets de remplissage
12	\$00	" " "
3	\$F5	prennent pour valeur \$A1
1	\$FB	marquage (Data Adress Mark)
512	données	les données se trouvent ici
1	\$F7	Ecrire la somme-test CRC
40	\$4E	octets de remplissage
etc...		
fin de la piste		
1401	\$4E	octets de remplissage

Si on compte le nombre d'octets, on obtient 6969 octets par piste, ce qui donne une capacité de disquette (non formatée) de 557520 octets. Malheureusement, on ne peut pas utiliser cette capacité pour les données car le contrôleur ne pourrait pas les retrouver (comment reconnaîtrait-il la fin d'un secteur ?). Cependant, il est possible d'utiliser les 14401 octets restants pour créer un secteur. Cela augmentera la capacité utilisable à 409600 octets. Si on ajoute 3 nouvelles pistes (pistes 80 à 82), on obtient 424960 octets. Mais comme nous l'avons déjà dit, la sécurité n'est plus aussi fiable.

Pour créer un tel format, il faut un petit programme. Avant de le voir, nous devons d'abord étudier de plus près le formatage de la disquette. Le formatage des pistes ne suffit pas. Il faut également écrire les paramètres utilisés comme le nombre de pistes et de secteurs afin que l'ordinateur puisse déterminer si la disquette a été formatée. C'est à cela que sert le boot secteur.

3.2 LE BOOT SECTEUR

Le boot secteur se trouve toujours au début d'une disquette ou d'un disque dur, c'est-à-dire sur la piste 0, face 0, secteur 1 ou sur le secteur 0 du disque dur. Comme tous les autres secteurs, il fait 512 octets et il est testé par le système d'exploitation dès qu'on change de disquette. De plus, il est indispensable pour "booter" une disquette. "Booter" signifie charger un système d'exploitation à la mise en route de l'ordinateur. Dans cette opération, le bootsecteur de la disquette se trouvant dans le lecteur A est chargé puis le système teste si la disquette contient un système d'exploitation. Le boot secteur contient également d'autres informations.

Il contient le numéro de série de la disquette, un bloc de paramètres pour le BIOS et éventuellement, un programme de chargement avec les paramètres de chargement. S'il contient un tel programme, la somme de tous les octets du secteur (somme-test) doit donner le nombre "magique" \$1234. Si ce nombre est atteint, le programme est exécuté dès le début du secteur où se trouve une instruction BRA (Branch always). Le programme doit être conçu de manière à pouvoir fonctionner à n'importe quelle adresse mémoire car le boot secteur est chargé n'importe où.

Habituellement, le boot secteur ne contient pas de boot-programme. Les paramètres qui se trouvent sur le secteur sont plus importants. Ces paramètres sont chargés par le système d'exploitation à l'aide d'un appel "Get BPB". Ils sont inscrits dans le BPB (Bios Parameter Block). Si ces paramètres ne conviennent pas, la fonction Get BPB ne renvoie pas l'adresse du BPB mais un 0.

L'information suivante est le numéro de série de la disquette. Il s'agit d'un nombre sur 24 bits qui est créé lors du formatage. Ce nombre sert à reconnaître les changements de disquette.

Voici la structure complète du boot secteur : 482

Octet	Nom	Signification	(normal 1/2 faces)
\$00	BRA	Instruction de branchement dans le boot-programme (event.)	
\$02	Remp.	Octets de remplissage réservés ou "loader"	
\$08	Série	Numéro de série	
* \$0B	BPS	Nombre d'octets par secteur (512)	
* \$0D	SPC	Nombre de secteurs par cluster (2)	
* \$0E	RES	Nombre de secteurs réservés (1)	
* \$10	NFATS	Nombre de FATs (File Allocation Tables) (2)	
* \$11	NDIRS	Nombre de Données par directory (112)	
* \$13	NSECTS	Nombre de secteurs par disquette (720/1440)	
* \$15	MEDIA	Description du support (inutilisé)	
* \$16	SPF	Nombre de secteurs de la FAT (5)	
* \$18	SPT	Nombre de secteurs par piste (9)	
* \$1A	NSIDES	Nombre de faces de la disquette (1/2)	
* \$1C	NHID	Nombre de secteurs cachés (0)	
\$1E	EXECFLG	Flag pour COMMAND.PRG	
\$20	LDMODE	Flag pour le boot secteur ou le boot file	
\$22	SSECT	Premier secteur à charger	
\$24	SECTCNT	Nombre de fichiers à charger	
\$26	LDAADR	Adresse de chargement	
\$2A	FATBUF	Adresse de la FAT	
\$2E	FNAME	Nom du fichier (la plupart du temps TOS.IMG)	
\$39	RES	Réserve	
\$3A	BOOTIT	Programme boot	482
- \$1FD			
\$1FE		Mot de comparaison pour la somme-test	

Les données marquées par une astérisque (*) correspondent au BPB de la disquette. Ce tableau est le même que celui de MS-DOS, le système d'exploitation de l'IBM PC. Remarquez également qu'un mot de 16 bit est inscrit sous la forme mot de poids faible/mot de poids fort (p. ex. BPS = \$00 \$02 indique \$200 octets par secteur). Ainsi, l'ATARI ST est en mesure de lire les disquettes IBM. Mais il ne peut pas les convertir car la gestion des données est différente sur les disquettes de l'IBM PC.

Encore quelques remarques sur les données du Boot secteur :

- Les nombres entre parenthèses indiquent les valeurs possibles des paramètres pour une disquette en simple face.
- NHID, le nombre de secteurs cachés n'est pas utilisé par le BIOS de l'ATARI ST.

Les données situées à partir de \$1E ne sont utiles que lorsque la disquette peut être "bootée". Une telle disquette contient normalement le système d'exploitation sous la forme d'un fichier nommé fichier-image (.IMG). On reconnaît un boot secteur exécutable au fait qu'il contient le texte 'loader' à partir du troisième octet. Le bootprogramme qui est situé dans les deux ROMs de l'ancien ATARI ST reconnaît un boot secteur exécutable à la somme test qui doit être de \$1234. Si c'est le cas, les autres données du boot secteur ont la signification suivante :

EXECFLG est copié dans la variable système 'cmdload'. Ce flag détermine si il faut charger ou non le programme COMMAND.PRG après le chargement du système d'exploitation.

LMODE détermine le mode de chargement. Si ce flag est à zéro, le fichier indiqué par *FNAME* est recherché puis chargé. Ce fichier s'appelle habituellement TOS.IMG. Si *LDMODE* est différent de zéro, les secteurs sont chargés en fonction de *SECTCNT* et *SSECT*.

SSECT est le secteur logique à partir duquel on boote. Cette variable n'est utilisable que lorsque *LDMODE* est différent de zéro.

SECTCNT indique le nombre des secteurs à booter. Cela n'est valable qu'avec *LDMODE* différent de zéro.

LDADDR est l'adresse-mémoire à partir de laquelle le fichier ou les secteurs sont chargés.

FATBUF indique l'adresse à laquelle les secteurs FAT et les secteurs de catalogue doivent être chargés.

FNAME est le nom de fichier du fichier-image qui doit être chargé (*LDMODE* = 0). Il a exactement la même structure qu'un nom de fichier normal, c'est à dire 8 caractères pour le nom et 3 caractères pour l'extension.

BOOTIT est un programme-boot qui peut être éventuellement chargé après le chargement du boot secteur.

Nous venons donc de voir la structure du boot secteur. Avec le format de la disquette, vous possédez maintenant suffisamment de connaissances pour en venir à la pratique. Nous allons l'appliquer avec un programme pour formater une disquette.

Le menu 'Fichiers' permet de formater des disquettes. Mais comme nous l'avons déjà indiqué, le format utilisé par le TOS d'ATARI est limité à 80 pistes de 9 secteurs. Mais en réalité, on peut mettre beaucoup plus de pistes et de secteurs sur une disquette.

3.2.1 Un programme de formatage

Le programme qui suit offre quelques possibilités pour augmenter la capacité d'une disquette. Les paramètres peuvent être modifiés à l'aide d'un petit menu. On choisit l'état de fonctionnement grâce aux touches de fonction.

Le menu qui apparaît après la mise en route du programme est le suivant :

*** programme de formatage S.D ***

[F1]	face(s)	:	2
[F2]	pistes	:	80
[F3]	secteurs/piste	:	9
[F4]	lecteur	:	A
[F8]	formatage		
[F10]	Quit !		

Lorsqu'on presse une touche de fonction, soit un paramètre est modifié, soit une fonction est exécutée. Les paramètres sont les suivants :

- F1 : Cette touche de fonction permet de choisir entre une ou deux faces. Si vous utilisez un lecteur simple face, choisissez l'option '1 face'.
- F2 : Cette touche permet de sélectionner de 80 (normal) à 82 pistes. Il est possible de formater 83 pistes mais je n'ai pas prévu cette possibilité car cela réduirait considérablement la sécurité des données. Mais vous pouvez travailler avec 83 pistes en modifiant très légèrement le programme.
- F3 : La touche de fonction F3 permet de choisir 9 ou 10 secteurs par piste.
- F4 : Avec cette option, vous pouvez sélectionner le lecteur A ou le lecteur B. Il faut contrôler ce point avant le formatage de manière à ne pas formater une disquette par mégarde.
- F8 : Cette touche effectue le formatage proprement dit. Le processus commence dès que vous avez pressé la touche et il est indiqué par le message "formatage en cours, veuillez patienter...". Si une erreur survient le message "*** Une erreur est survenue ***" apparaît. Dans ce cas, vérifiez que vous avez bien déprotégé la disquette. Le message d'erreur reste à l'écran tant que vous n'avez pas pressé de touche. Ainsi, vous n'avez pas besoin de rester présent durant le formatage.
- F10 : Une fois que vous avez fini de formater les disquettes, pressez la touche F10 pour quitter le programme.

La flexibilité de ce programme vous permet de déterminer toutes les possibilités de formatage. Voici quelques capacités de disquettes simple face :

Pistes	Secteurs/piste	Capacité en octets
80	9 (normal)	357376
82	9	366592
80	10	398336
82	10	408576

Comme vous le constatez au tableau précédent, on peut gagner jusqu'à 51 200 octets pour une disquette simple face. Avec les disquettes double face, cette capacité est doublée si bien qu'on peut gagner 100 Koctets.

Voici maintenant le programme. Il a été créé avec l'assembleur SEKA qui présente quelques différences avec l'assembleur de DRI. Ainsi, les lignes de commentaires commencent avec une astérisque (*) et les commandes 'blk.b' doivent être transformées en 'ds.b'.

;; Programme de formatage S.D. ;;

run:

```

move.l #menu,d0
bsr    print      ;Afficher menu
bsr    getkey
cmp.b  #$3b,d0
blt    run        ;Mauvaise touche
cmp.b  #$44,d0
bgt    run        ;Mauvaise touche

cmp.b  #$3b,d0    ;F1 ?
bne    notf1
eor    #3,sds     ;1/2 faces
eor    #1,sdsf
bra    run

```

notf1:

```

cmp.b  #$3c,d0    ;F2 ?
bne    notf2

```

```

eor    #2,trs      ;B0/82 Tracks
eor    #2,trs
bra    run

```

```

notf2:
  cmp.b  #3d,d0      ;F3 ?
  bne    notf3
  eor    #3,sptf
  eor    #1109,spt    ;9/10 Secteurs par piste
  bra    run

```

```

notf3:
  cmp.b  #3e,d0      ;F4 ?
  bne    notf4
  eor    #3,lw
  eor    #1,lwf       ;Lecteur A/B
  bra    run

```

```

notf4:
  cmp.b  #42,d0      ;FB ?
  bne    notf8
  bsr    format       ;=> Formatage
  bra    run

```

```

notf8:
  cmp.b  #44,d0      ;F10 ?
  bne    run
  clr    -(sp)
  trap   #1           ;Quit, retour au Desktop

```

```

format:
  move.l  #wait,d0    ;# Formatage #
  bsr     print        ;"Formatage en cours.."
  move    trsf,trsf1
  subq    #1,trsf1

```

```

floop:
  move    sdsf,face    ;Déterminer la face
floop1:
  bsr     fattr        ;formatage d'une piste

```

```

    bne    error
    subq   #1,face      ;encore l'autre face
    bpl    floopl       ;formatage
    subq   #1,trsf1
    bpl    floop        ;Piste suivante

setboot:                ;Création du Boot-Sector
    clr    -(sp)        ;Execute-Flag: Inexécutable
    moveq  #2,d0
    or     sdsf,d0
    move   d0,-(sp)      ;Choix du type de disque et du nbre de face
    move.l #$1000000,-(sp) ;Création du n° de série
    pea    tampon        ;buffer-Adress
    move   #$12,-(sp)
    trap   #14           ;Création du Boot-Sector
    add.l  #14,sp

    lea    tampon,a0     ;Pointeur sur Boot-Sector-buffer
    clr.l  d0
    cmp    #9,sptf       ;9 Secteurs par piste ?
    beq    sok           ;oui
    move.b #10,24(a0,d0) ;sinon mettre 10 SPT
    move   trsf,d1        ;Nombre de pistes dans D1
    tst    sdsf           ;1 face ?
    beq    sd11          ;oui
    lsl    #1,d1          ;sinon doubler
sd11:
    bsr    addsec        ;SEC + nombre de pistes (D1)

sok:
    cmp    #80,trsf      ;80 Tracks ?
    beq    trak          ;oui
    move   #18,d1
    tst    sdsf           ;1 face ?
    beq    sd12          ;oui
    lsl    #1,d1          ;sinon doubler
sd12:
    bsr    addsec        ;SEC + 2*9 ou 4*9

trak:

```

```

move    #1,-(sp)      ;1 secteur
clr.l   -(sp)         ;face 0, Track 0
move    #1,-(sp)      ;Secteur 1
move    lwf,-(sp)      ;Lecteur
clr.l   -(sp)
pea     tampon         ;Buffer
move    #9,-(sp)
trap    #14            ;flopwr, écrire Boot-Sector
add.l   #20,sp
tst     d0             ;Erreur survenue ?
bne     error          ;oui: Message d'erreur

bra     run            ;Redémarrage

addsec:                ;SEC = SEC + D1
move.b  20(a0,d0),d2   ;HI
lsl     #8,d2
move.b  19(a0,d0),d2   ;LO
add     d1,d2
move.b  d2,19(a0,d0)   ;set LO
lsr     #8,d2
move.b  d2,20(a0,d0)   ;set HI
rts

error:
move.l   #errtxt,d0
bsr     print          ;Afficher message d'erreur
bsr     getkey         ;Attendre touche
bra     run            ;et redémarrer

fmttr:                ;Formater une piste
clr      -(sp)         ;Données vierges
move.l   #4B7654321,-(sp) ;Chiffre magique
move     #1,-(sp)      ;interleave
move     face,-(sp)    ;face
move     trsf1,-(sp)   ;Track
move     sptf,-(sp)    ;Secteurs/Track
move     lwf,-(sp)     ;Lecteur
clr.l   -(sp)
pea     tampon         ;Track-buffer

```

```

move    #10,-(sp)
trap    #14          ;flopfmt, Formater piste
add.l   #26,sp
tst     d0           ;Teste si erreur
rts

```

```

print:                                     ;Afficher le texte à partir de (D0)

```

```

move.l  d0,-(sp)
move    #9,-(sp)
trap    #1
addq.l  #6,sp
rts

```

```

getkey:                                   ;Attendre pression sur une touche

```

```

move.w  #1,-(sp)
trap    #1
addq.l  #2,sp
swap    d0          ;Code de la touche dans D0.b
rts

```

```

; Texte et variables:

```

```

even

```

```

menu:   dc.b $1b,"E$$$ Programme de formatage S.D. $$$"
        dc.b 10,13,10,13
        dc.b " [F1]  face(s) .....: "
sds:    dc.b " 2",10,13
        dc.b " [F2]  Tracks .....: "
trs:    dc.b "80",10,13
        dc.b " [F3]  Secteur/piste ....: "
spt:    dc.b " 9",10,13
        dc.b " [F4]  Lecteur .....: "
lw:     dc.b " A",10,13
        dc.b " [F8]  Formater .....",10,13
        dc.b "[F10]  Quit !",10,13,10,13,0

```

```

wait:   dc.b " Formatage en cours, patientez...",10,13,0

```

```

errtxt: dc.b " $$ Une erreur est survenue !! $$",10,13,0

```

```

even

```

```
sdsf: dc.w 1  
trsf: dc.w 80  
trsf1: dc.w 80  
sptf: dc.w 9  
lwf: dc.w 0  
face: dc.w 0
```

```
data
```

```
tampon: blk.b 8000
```

Le programme est divisé de la façon suivante :

1) Gestion des menus : le menu est affiché (on efface d'abord l'écran) et le programme attend qu'une touche soit pressée. Après la saisie, le code de la touche pressée, inscrit dans D0, est converti. Si une comparaison `CMP.B#$xx,D0` est valable, la fonction est exécutée. Avec les fonctions flip-flop (F1-F4), le changement de fonction est pris en compte aussi bien dans le texte du menu que dans la ligne de paramètres correspondante grâce à l'instruction `EOR`. Après une conversion réussie, le programme se branche au début (run) sauf si on a pressé la touche F10 qui permet de quitter le programme grâce à la fonction `TERM`.

2) Formatage : après l'affichage du message "Formatage en cours...", la disquette est formatée à partir de la piste maximale-1 jusqu'à la piste 0. Si vous avez sélectionné un formatage en double face, chaque piste est d'abord formatée sur la face 1 puis sur la face 0.

3) Création du boot secteur : la routine crée d'abord un boot secteur normal avec la fonction 'protobt' du XBIOS. Elle tient compte pour cela du nombre de faces.

4) Correction du boot secteur : s'il faut faire des modifications par rapport au boot secteur normal (10 secteurs par piste, 82 pistes), le boot secteur est corrigé. Le programme teste d'abord le nombre de secteurs par piste. S'il est égal à 10, il est inscrit dans la case `SPT` du boot secteur puis le nombre de pistes est additionné au nombre de secteurs sur la disquette. Ensuite, le programme vérifie le nombre de piste choisi et additionne si nécessaire le nombre de secteurs.

5) Sauvegarde du boot secteur : le nouveau boot secteur est sauvegardé sur la face 0, piste 0 secteur 1 avec la fonction 'flopwr' du XBIOS. Si une erreur survient, elle est affichée.

6) Secteur de données : cette partie contient les textes des menus et des messages ainsi que les variables. Bien que la longueur du tampon soit définie, il n'est pas inscrit sur disquette car il se situe dans la zone ".bss". Avec l'assembleur DRI, remplacez "data" par ".bss".

Voici maintenant un programme BASIC qui crée le programme de formatage sur disquette. Ce programme porte le nom "BIGFORMAT.PRГ".

```

10 '##### File-Maker A.S. #####
15 '
20 ? :fullw 2:clearw 2:gotoxy 0,0
25 ? "Le fichier >> bigformat.prГ << est crГe":? :? :?
30 dim c%( 415):cs#=0
35 for i=0 to 415
40 read a$:c%(i)=val("&H"+a$)
45 check#=check#+(c%(i))
50 next i
55 if check#= 4116448.96 then 70
60 ? "Ca ne va pas encore, car quelque chose ne convient pas dans les DATAs
  "
65 goto 80
70 bsave "bigformat.prГ",varptr(c%(0)), 831
75 ? "Le programme >> bigformat.prГ << est Гcrit."
80 ? :? :? :? "Pressez une touche":a=inp(2):end
85 '
90 '##### DATA's pour bigformat.prГ #####
95 '
100 DATA 601A,0000,01DA,0000,0124,0000,1F40,0000
101 DATA 0000,0000,0000,0000,0000,0000,203C,0000
102 DATA 01DA,6100,01BA,6100,01C2,B03C,003B,6DEC
103 DATA B03C,0044,6EE6,B03C,003B,6612,0A79,0003
104 DATA 0000,0220,0A79,0001,0000,02F2,60CE,B03C
105 DATA 003C,6612,0A79,0002,0000,023E,0A79,0002
106 DATA 0000,02F4,60B6,B03C,003D,6612,0A79,0003
107 DATA 0000,02F8,0A79,1109,0000,025C,609E,B03C
108 DATA 003E,6612,0A79,0003,0000,027A,0A79,0001
109 DATA 0000,02FA,6086,B03C,0042,6606,6110,6000
110 DATA FF7C,B03C,0044,6600,FF74,4267,4E41,203C
111 DATA 0000,02AB,6100,012B,33F9,0000,02F4,0000
112 DATA 02F6,5379,0000,02F6,33F9,0000,02F2,0000
113 DATA 02FC,6100,00CE,6600,00BC,5379,0000,02FC
114 DATA 6AF0,5379,0000,02F6,6ADE,4267,7002,8079
115 DATA 0000,02F2,3F00,2F3C,0100,0000,4879,0000
116 DATA 02FE,3F3C,0012,4E4E,DFFC,0000,000E,41F9

```

117 DATA 0000,02FE,4280,0C79,0009,0000,02F8,6718
118 DATA 11BC,000A,0018,3239,0000,02F4,4A79,0000
119 DATA 02F2,6702,E349,6144,0C79,0050,0000,02F4
120 DATA 670E,7212,4A79,0000,02F2,6702,E349,612C
121 DATA 3F3C,0001,42A7,3F3C,0001,3F39,0000,02FA
122 DATA 42A7,4879,0000,02FE,3F3C,0009,4E4E,0FFC
123 DATA 0000,0014,4A40,661C,6000,FEA2,1430,0014
124 DATA E14A,1430,0013,D441,1182,0013,E04A,1182
125 DATA 0014,4E75,203C,0000,02CF,6142,614C,6000
126 DATA FE7C,4267,2F3C,8765,4321,3F3C,0001,3F39
127 DATA 0000,02FC,3F39,0000,02F6,3F39,0000,02F8
128 DATA 3F39,0000,02FA,42A7,4879,0000,02FE,3F3C
129 DATA 000A,4E4E,0FFC,0000,001A,4A40,4E75,2F00
130 DATA 3F3C,0009,4E41,5C8F,4E75,3F3C,0001,4E41
131 DATA 548F,4840,4E75,1B45,2A2A,2A20,2050,726F
132 DATA 6772,616D,6D65,2064,6520,666F,726D,6174
133 DATA 6167,6520,2020,532E,442E,202A,2A2A,0A0D
134 DATA 0A0D,205B,4631,5D20,2066,6163,6528,7329
135 DATA 202E,2E2E,2E2E,2E2E,2E2E,3A20,2032,0A0D
136 DATA 205B,4632,5D20,2054,7261,636B,7320,2E2E
137 DATA 2E2E,2E2E,2E2E,2E2E,3A20,3B30,0A0D,205B
138 DATA 4633,5D20,2053,6563,7465,7572,732F,5069
139 DATA 7374,652E,2E2E,3A20,2039,0A0D,205B,4634
140 DATA 5D20,204C,6563,7465,7572,202E,2E2E,2E2E
141 DATA 2E2E,2E2E,3A20,2041,0A0D,205B,463B,5D20
142 DATA 2046,6F72,6D61,7465,7220,2E2E,2E2E,2E2E
143 DATA 2E2E,0A0D,205B,4631,305D,2020,5175,6974
144 DATA 2021,0A0D,0A0D,0020,466F,726D,6174,6167
145 DATA 6520,656E,2063,6F75,7273,2C20,7061,7469
146 DATA 656E,7465,7A2E,2E2E,0A0D,002A,2A20,556E
147 DATA 6520,6572,7265,7572,2065,7374,2073,7572
148 DATA 7665,6E75,6520,2121,202A,2A0A,0D00,0001
149 DATA 0050,0050,0009,0000,0000,0000,0002,220B
150 DATA 100B,100B,100B,200A,0406,0604,0E08,0C0E
151 DATA 120A,0E06,0E0A,160B,321A,0606,0608,0000

Encore quelques remarques au sujet du programme :

- La copie d'une disquette normale sur une disquette étendue n'est possible que par fichiers. En effet, le système d'exploitation ne copie pas directement la disquette car il reconnaît un format différent.
- Une disquette étendue ne peut pas être utilisée directement comme disquette TOS car le boot secteur ne contient pas de **LOADER**. Pour cela, il faudra copier tout d'abord le boot secteur d'une disquette normale puis modifier les paramètres de ce boot secteur avec un moniteur de disquettes, de manière à les adapter à la disquette étendue.
- N'utilisez pas la disquette étendue pour y inscrire des données importantes que vous ne possédez qu'en un seul exemplaire. Si les disquettes que vous employez ne sont pas très fiables, vous risquez de perdre un secteur dans les pistes supérieures...

Revenons à la théorie : comme nous l'avons déjà indiqué, le Bios Parameter Block est créé à partir d'informations diverses. Analysons ce BPB de plus près.

3.2.2 Le BIOS Parameter Block BPB

Nous connaissons déjà quelques données de ce BIOS Parameter Block car elles existent dans le boot secteur. Le BPB est créé grâce à la fonction 'Get BPB' (N°7) de la BIOS, lorsqu'on change de disquette. Contrairement au boot secteur, le BPB contient des données dans le format 16 bits normal dans l'ordre suivant :

recsize	- taille du secteur en octets	(512)
clziz	- taille du cluster en secteurs	(2)
clsizb	- taille du cluster en octets	(1024)
rdlen	- nombre des secteurs du catalogue	(7)
fsiz	- taille de la FAT en secteurs	(5)
fatrec	- secteur de départ de la deuxième FAT	(6)
datrec	- premier secteur de données (rdlen+fsiz+fatrec=18)	
numcl	- Nombre de clusters de données	(711)
bflags	- Taille des données de la FAT en bits (0 = 12 bits, 1 = 16 bits)	(0)

Les nombres entre parenthèses indiquent la valeur standard des paramètres pour une disquette double-face.

Nous allons maintenant étudier un nouveau programme permettant de lire et d'analyser ce BIOS parameter block BPB. Le programme est de structure très simple. Il affiche d'abord un prompt qui contient le titre. Ce prompt demande la saisie d'une lettre. Il faut entrer soit un code de lecteur (a,b,c ou d) soit la lettre 'q'. 'q' permet de quitter le programme et de retourner au DESKTOP.

Après la saisie, le programme teste si la lettre saisie convient. Si ce n'est pas le cas, on recommence au début, si on a saisi un "q", on quitte le programme.

La lettre est ensuite convertie dans la valeur nécessaire pour la fonction GETBPB (0 à 3). Cette valeur est obtenue par soustraction de "a". Le programme appelle ensuite la fonction GETBPB. On obtient l'adresse du BPB dans le registre D0.

Le programme lit ensuite les paramètres du BPB les uns à la suite des autres et les affiche en hexadécimal avec le texte correspondant. Vous avez donc d'un seul coup d'oeil toutes les informations importantes sur la disquette.

Voici le programme qui a été écrit lui aussi avec l'assembleur SEKA.

;; BPB-Analysator S.D. ;;

run:

```
move.l #prompt,d0
bsr     pmsg      ;Afficher Prompt
bsr     getkey    ;saisie des lecteurs A-D
cmp.b   #'q',d0   ;Quit ?
beq     quit      ;oui => Desktop
move    d0,d6     ;Sauvegarder caractère
```

```
bsr    pcr1f      ;Afficher CR

sub.b   #'a',d6    ;Transformer valeur
bmi     run        ;Mauvaise saisie
cmp.b   #3,d6
bgt     run        ;mauvaise saisie

move    d6,-(sp)   ;Device-Nr.
move    #7,-(sp)
trap    #13        ;GETBPB-Function
addq.l   #4,sp

tst.l   d0
beq     run        ;Erreur !
move.l   d0,a5     ;Sauvegarder BPB-Adress

bsr     pnext
move.l   #bps,d0
bsr     pline      ;"octets par secteur"

bsr     pnext
move.l   #spc,d0
bsr     pline      ;"Secteurs par Cluster"

bsr     pnext
move.l   #bpc,d0
bsr     pline      ;"Octets par Cluster"

bsr     pnext
move.l   #dirsec,d0
bsr     pline      ;"Secteur-Directory"

bsr     pnext
move.l   #fatsec,d0
bsr     pline      ;"Secteurs-FAT"

bsr     pnext
move.l   #fat2s,d0
bsr     pline      ;"Start-Sector du 2. FAT"
```

```
    bsr    pnext
    move.l  #datsec,d0
    bsr    pline    ;"Start-Sector des données"

    bsr    pnext
    move.l  #datc,d0
    bsr    pline    ;"cluster de données"

    move    #'$',d0
    bsr    pchar    ;"$" Afficher
    move    #12,d0  ;Récupérer 12 Bit
    btst    #0,(a5) ;correct ?
    beq     bits12   ;oui
    move    #16,d0   ;sinon 16 Bit
bits12:
    bsr    phexbyt
    move.l  #fatbit,d0
    bsr    pline    ;"Bits par donnée FAT"

    bra     run      ;fini => redémarrage

quit:                                ; Retour au Desktop
    clr     -(sp)
    trap    #1

getkey:                                ;Get Key -> D0
    move    #1,-(sp)
    trap    #1
    and.l   #$ff,d0
    addq.l  #2,sp
    rts

pline:                                ;Print Line/CR
    bsr    pmsg

pcrlf:                                ;Print CR,LF
    move    #10,d0
    bsr    pchar
    move    #13,d0

pchar:                                ;Print Character D0
    move    d0,-(sp)
```

```

    move    #2,-(sp)
    trap    #1
    addq.l   #4,sp
    rts

pmsg:                                ;Print Line (D0)
    move.l   d0,-(sp)
    move     #9,-(sp)
    trap     #1
    addq     #6,sp
    rts

pnext:                                ;Récupérer mot suivant et l'afficher
    move     #'$',d0
    bsr      pchar                    ;Afficher "$"
    move     (a5)+,d0
phexword:                                ;Print Hex-Word D0
    moveq    #3,d1
    bra      phex1
phexbyt:                                ;Print Hex-Byte
    moveq    #1,d1
    rol.l    #8,d0
phex1:
    rol.l    #4,d0
    move.l    d0,-(sp)
    move.l    d1,-(sp)
    bsr      phexnib                  ;Afficher un nibble (0-F)
    move.l    (sp)+,d1
    move.l    (sp)+,d0
    dbra     d1,phex1
    rts

phexnib:
    and.l     #$7f,d0
    swap      d0
    and.l     #$0f,d0
    add.b     #$30,d0
    cmp.b     #$3a,d0
    bcs       phexn
    add.b     #7,d0

```

phexn:

bra pchar ;Afficher Nibble

prompt: dc.b "### BPD-Analyseur S.D. ###",10,13

dc.b "Saisissez le lecteur (a-d) ou",10,13

dc.b "'q' pour Quitter : ",0

bps: dc.b " Bytes par Secteur",0

spc: dc.b " Secteurs par Cluster",0

bpc: dc.b " Bytes par Cluster",0

dirsec: dc.b " Secteurs-Directory",0

fatsec: dc.b " Secteurs-FAT",0

fat2s: dc.b ": Start-Sector 2.FAT",0

datsec: dc.b ": Start-Sector des données",0

datc: dc.b " Cluster de données",0

fatbit: dc.b " Bits par donnée de la FAT",10,13,0


```
10 '***** File-Maker A.S. *****
15 '
20 ? :fullw 2:clearw 2:gotoxy 0,0
25 ? "Le fichier >> bpbanaly.prg << est crée":??:?
30 dim c%( 310):cs#=0
35 for i=0 to 310
40 read a$:c%(i)=val("%H"+a$)
45 check#=check#+(c%(i))
50 next i
55 if check#= 4451543.04 then 70
60 ?"Ca ne va pas encore, car quelque chose ne convient pas dans les DATAs
."
65 goto 80
70 bsave "bpbanaly.prg",varptr(c%(0)), 622
75 ? "Le programme >> bpbanaly.prg << est écrit."
80 ??:??:?"Pressez une touche":a=inp(2):end
85 '
90 '***** DATA's pour bpbanaly.prg *****
95 '
```

```
100 DATA 601A,0000,0244,0000,0000,0000,0000,0000
101 DATA 0000,0000,0000,0000,0000,0000,203C,0000
102 DATA 012A,6100,00DB,6100,00B0,B03C,0071,6700
103 DATA 00A4,3C00,6100,00B4,9C3C,0061,6BDE,BC3C
104 DATA 0003,6EDB,3F06,3F3C,0007,4E4D,5BBF,4A80
105 DATA 67CA,2A40,6100,00B2,203C,0000,017C,6100
106 DATA 00B8,6100,00A4,203C,0000,0190,6100,007A
107 DATA 6100,0096,203C,0000,01A7,6100,006C,6100
108 DATA 00B8,203C,0000,01BB,615E,6100,007C,203C
109 DATA 0000,01D0,6152,6100,0070,203C,0000,01DF
110 DATA 6146,6164,203C,0000,01F4,613C,615A,203C
111 DATA 0000,020F,6132,303C,2400,6134,700C,0B15
112 DATA 0000,6702,7010,614C,203C,0000,0224,611B
113 DATA 6000,FF4A,4267,4E41,3F3C,0001,4E41,C0BC
114 DATA 0000,00FF,54BF,4E75,6112,700A,6102,700D
115 DATA 3F00,3F3C,0002,4E41,5BBF,4E75,2F00,3F3C
116 DATA 0009,4E41,5C4F,4E75,303C,2400,61E2,301D
117 DATA 7203,6004,7201,E19B,E99B,2F00,2F01,610A
118 DATA 221F,201F,51C9,FFF2,4E75,CCBC,0000,007F
119 DATA 4B40,C0BC,0000,000F,D03C,0030,B03C,003A
120 DATA 6502,5E00,60AA,2A2A,2A20,2042,5042,2D41
121 DATA 6E61,6C79,7365,7572,2020,532E,442E,2020
122 DATA 2A2A,2A0A,0B53,6169,7369,7373,657A,206C
123 DATA 6520,6C65,6374,6575,7220,2B61,2D64,2920
124 DATA 6F75,0A0D,2771,2720,706F,7572,2051,7569
125 DATA 7474,6572,203A,2000,2020,4279,7465,7320
126 DATA 7061,7220,5365,6374,6575,7200,2020,5365
127 DATA 6374,6575,7273,2070,6172,2043,6C75,7374
128 DATA 6572,0020,2042,7974,6573,2070,6172,2043
129 DATA 6C75,7374,6572,0020,2053,6563,7465,7572
130 DATA 732D,4469,7265,6374,6F72,7900,2020,5365
131 DATA 6374,6575,7273,2D46,4154,003A,2053,7461
132 DATA 7274,2053,6563,746F,7220,322E,4641,5400
133 DATA 3A20,5374,6172,742D,5365,6374,6F72,2064
134 DATA 6573,2064,6F6E,6E82,6573,0020,2043,6C75
135 DATA 7374,6572,2064,6520,646F,6E6E,8265,7300
136 DATA 2020,2020,4269,7473,2070,6172,2064,6F6E
137 DATA 6E82,6520,6465,206C,6120,4641,540A,0D00
138 DATA 0000,0002,3C0E,0E0E,0C0C,0A0A,1A00
```

A la mise en route de l'ordinateur, les données du BPB ne sont pas mémorisées. Le système d'exploitation crée le BPB seulement après avoir booté, c'est-à-dire après être en possession du nombre des lecteurs et de leurs caractéristiques.

Mais pour cela, il faut bien entendu qu'il y ait un système d'exploitation en mémoire.

Si le TOS n'est pas en ROM, il faut d'abord le charger. L'ordinateur boote sur la disquette même s'il a le TOS en ROM, si cette disquette a un système d'exploitation qui peut être booté et un boot secteur.

Le processus du "Boot" se fait en quatre parties :

1. Le boot secteur est chargé et le boot programme qu'il contient est exécuté.
2. La FAT et le catalogue sont chargés de la disquette. Le chargeur recherche le nom de fichier indiqué (le plus souvent TOS.IMG). S'il ne le trouve pas, il renvoie un message d'erreur.
3. TOS.IMG est chargé à partir de l'adresse mémoire \$40000.
4. Le programme chargé est exécuté au début.

Le TOS.IMG est lui même constitué de trois parties :

- Un relocateur, c'est à dire un programme qui relog le système d'exploitation à l'adresse prévue (\$6100). Ce programme efface l'écran, déplace le TOS imagebloc à son adresse permanente et le met en route.
- Les données du système d'exploitation (BIOS, XBIOS).
- Les données de GEM et du programme DESKTOP.

Comme on le voit, la structure du système d'exploitation du fichier TOS.IMG est assez compliqué. Le TOS en ROM qui tient dans 6 PROM (Programmable Read Only Memory) est naturellement un peu plus court car il contient seulement le TOS et GEM sans relocateur.

Venons-en à la partie suivante concernant les structures de données sur disquette et analysons la structure et la gestion du catalogue.

3.3 LE CATALOGUE

Le catalogue commence à la piste 1, secteur 3 et prend 7 secteurs sur une disquette formatée en simple face. Outre les noms de fichiers et leur extension, il contient d'autres données qui sont plus ou moins utiles pour la gestion de la disquette.

Chaque donnée du catalogue est constituée de 32 octets qui contiennent toutes les informations utiles au système d'exploitation concernant le fichier. Ces 32 octets se divisent en 8 champs qui ont la structure suivante :

1) Nom de fichier	8 octets
2) Type du fichier (extension)	3 octets
3) Attribut	1 octet
4) Réserve	10 octets
5) Heure	2 octets
6) Date	2 octets
7) Premier cluster	2 octets
8) Taille du fichier	4 octets

Le premier champ contient donc le nom de fichier. Ce nom est constitué de caractères ASCII, c'est-à-dire seulement des lettres et des chiffres. Seules les majuscules sont utilisées. Le nom prend au plus 8 caractères, s'il a moins de 8 caractères, le reste est rempli avec des espaces.

Si le premier octet du nom est un zéro, cela signifie que le champ n'a pas encore été utilisé. Si le fichier a déjà été utilisé puis effacé, on trouve un 229 à cet endroit (\$E5).

Si le premier caractère du nom est un point (.), cela correspond à un sous-catalogue, c'est à dire un dossier.

Le champ qui suit contient le type du fichier, c'est à dire l'extension. Ce type est limité à 3 lettres (p. ex. PRG, TOS, BAS etc...) et peut être rempli éventuellement par des espaces. Ici aussi, seules les majuscules sont utilisées.

A la suite se trouve l'octet d'attribut de fichier qui contient, codé en binaire, le statut de ce fichier. Voici la signification de ces bits :

Bit	Signification si activé (1)
0	lecture uniquement
1	fichier caché
2	fichier système
3	la donnée est le nom du lecteur
4	la donnée est un dossier
5	le fichier a été modifié

Après cet octet, on trouve 10 octets qui n'ont aucune signification. Il servent d'octets de réserve qui pourront être utilisés pour des applications ultérieures.

A la suite se trouvent deux octets qui contiennent l'heure à laquelle a été effectuée la dernière modification du fichier. Pour gagner de la place, l'heure a été codée de manière un peu spéciale.

Les seize bits de l'heure sont divisés en 3 parties : les heures, les minutes et les secondes. Cette séparation est faite de la manière suivante :

Exemple : 19:21:34

heure	minutes	secondes/2
10011	010101	10001

Les secondes sont décomptées deux à deux, c'est pourquoi les 5 derniers bits de l'heure contiennent un 17.

Le champ suivant du catalogue contient la date de la dernière modification du fichier. La division en année, mois et jour se fait de la même manière qu'avec l'heure. Il n'y a que 7 bit de réservés pour l'année et il faut additionner 1980 au numéro indiqué. Ainsi, la valeur suivante donne :

Exemple : 12.05.1986

Année	Mois	Jour
0000110	0101	01100

Le septième champ du catalogue contient le numéro du premier cluster de la disquette qui correspond à un fichier.

Le fichier est mémorisé en premier dans ce cluster (qui est divisé en deux secteurs). Nous verrons dans le prochain chapitre sur la FAT comment la suite est structurée.

Le dernier champ contient la longueur du fichier en octets. Il faut remarquer qu'on pourra lire moins d'octets que ce qui est indiqué. Cela dépend de la FAT. La longueur du fichier doit donc être considérée comme la longueur maximale.

Avec ces connaissances sur la structure du catalogue de la disquette, vous êtes maintenant en mesure d'étudier la structure de la disquette à l'aide d'un moniteur de disquette. On peut faire de nombreuses manipulations en modifiant certains paramètres. Leur effet peut parfois être très désagréable. Pour ces raisons, il est conseillé de faire une copie de la disquette avant les manipulations.

Si on veut écrire un programme qui puisse lire le catalogue d'une disquette, il faut préparer un tampon qui contienne les données nécessaires avant d'appeler la fonction correspondante. Le début de ce tampon est appelé Disk Transfert Address (DTA).

Ce tampon fait 44 octets de long et doit être précisé au système d'exploitation en appelant une fonction particulière. Par la suite, la recherche des données du catalogue peut commencer. Pour cela, on cherche la première donnée avec la fonction SFIRST (Search First) puis les données suivantes avec SNEXT (Search Next) et on les charge dans le tampon à la DTA.

Le tampon contient après l'appel toutes les informations qui se trouvent dans la fenêtre-catalogue du DESKTOP. La structure des données est la suivante :

Octet(s)	Contenu
0...20	réservé
21	attribut de fichier
22,23	heure
24,25	date
26...29	taille du fichier en octets (LO,HI)
30...43	nom du fichier et extension

Venons-en maintenant au programme. Cette routine en langage machine fixe d'abord la DTA et recherche le nom de fichier dans le catalogue. Si le nom indiqué est seulement " *.*", la première donnée correspondant à l'attribut indiqué est chargée. Si aucune donnée convenable n'existe, la fonction renvoie la valeur -33 dans le registre de données D0 : fichier introuvable. Sinon, ce registre est à zéro.

MOVE.L	#tampon,-(sp)	* indiquer la DTA
MOVE	#\$1A,-(sp)	* numéro de fonction SETDTA
TRAP	#1	* appel du système d'exploitation
ADDQ.L	#6,sp	* restauration de la pile
MOVE	##%11001,-(sp)	* type de fichier : tous les fichiers
MOVE.L	#nom,-(sp)	* adresse du nom de fichier
MOVE	#\$4E,-(sp)	* numéro de fonction SFIRST
TRAP	#1	* Appel du système d'exploitation
ADDQ.L	#8,sp	* restauration de la pile
TST	D0	* trouvé
BNE	nyestpas	* non
etc...		
.		
.		
tampon:	.ds.b 44	* place pour les données
nom:	.dc.b " *.*",0	* Tous les noms sont autorisés

Pour rechercher la donnée suivante, la partie de programme que voici suffit :

MOVE	#\$4F,-(sp)	* Numéro de fonction SNEXT
TRAP	#1	* Appel du système d'exploitation
ADDQ.L	#2,SP	* restauration de la pile
TST	D0	* trouvé
BNE	nyestpas	* non

De cette manière, on peut très facilement réaliser un programme qui imprime le catalogue sur imprimante. Vous trouverez un tel programme qui imprime le catalogue complet, dossiers inclus au chapitre 5.3.

Si on affiche le catalogue dans le desktop, on a le nom, l'extension, la date, l'heure et la longueur du fichier.

Si vous cliquez sur un programme, le système d'exploitation doit non seulement savoir où il se trouve sur la disquette mais également où se trouvent les données suivantes. Cette information se trouve dans la FAT que nous allons analyser.

3.4 LA FAT

La FAT (File Allocation Table) prend normalement 5 secteurs sur une disquette simple face et commence normalement à la piste 0, secteur 2 de la face 0. La taille de cette table varie en fonction du format utilisé. Elle sert à mémoriser la structure des fichiers sur disquette.

La raison est qu'un fichier ne couvre pas obligatoirement des secteurs situés les uns derrière les autres. En effet, les secteurs qui avaient servi pour mémoriser un fichier maintenant effacé sont de nouveau libres pour d'autres données. Si on écrit un nouveau fichier sur la disquette, il pourra utiliser ces secteurs. Les secteurs occupés seront simplement ignorés.

A chaque secteur doit donc correspondre une donnée de la FAT indiquant s'il est libre ou occupé. Pour garder le début de la FAT aussi petit que possible, on regroupe deux secteurs ensemble en un cluster. Chaque cluster est numéroté de 2 à la fin de la disquette. La FAT contient donc des données sur deux secteurs à la fois.

Chaque donnée de la FAT fait normalement 12 bits. Certains formats utilisent des données de 16 bits, mais nous les ignorerons. Etant donné que les indications sont codées sur 12 bits, 2 données de la FAT prennent 3 octets.

Les deux premières indications de la FAT contiennent les informations sur le format. C'est pourquoi la numérotation commence à 2.

Chaque indication suivante concerne un cluster. Un zéro indique que le cluster correspondant est libre. Cela ne signifie pas bien entendu que les secteurs ne contiennent pas de données car un fichier écrasé n'est jamais réellement effacé sur la disquette. La suppression d'un fichier consiste simplement à remplacer la première lettre de son nom dans le catalogue par un \$E5 et à mettre un zéro dans la FAT à l'endroit correspondant. Les données du fichier demeurent en mémoire mais sont difficiles à trouver.

Si une donnée de la FAT contient un \$FF7, cela signifie que le cluster est inutilisable. De tels cluster sont reconnus et marqués lors du formatage. Si de telles erreurs surviennent sur la disquette, par exemple à cause d'une rayure, on le remarque à la capacité réduite de la disquette après le formatage. Mais si une telle erreur se produit sur la piste 0 ou 1, la disquette est inutilisable car ces pistes contiennent le boot secteur, la FAT et le catalogue.

Si vous voulez charger un fichier, le système d'exploitation récupère le numéro du premier cluster dans le catalogue. Puis il va rechercher dans la FAT le numéro du cluster suivant qui se trouve dans le champ du premier cluster. Le cluster suivant est indiqué dans le champ du second cluster, et ainsi de suite jusqu'à ce qu'on rencontre un \$FFF. Cela signifie que le cluster est le dernier du fichier.

On peut également modifier la FAT avec un moniteur de disquettes. Dans ce cas, la probabilité de perte des données est si grande qu'il faut d'abord faire une copie de la disquette.

3.5 STRUCTURE DU PROGRAMME

L' ATARI ST dispose d'une grande mémoire qui peut contenir de nombreux programmes. En fait, il est possible de mettre plusieurs programmes simultanément en mémoire et de les exécuter. Les accessoires sont un bon exemple car ils fonctionnent derrière le programme principal.

Cette structure ouverte de la mémoire amène cependant un problème. Avec les ordinateurs 8 bits, on est habitué à ce qu'un programme tienne à une place mémoire définie.

Cela tient au fait que les programmes en langage machine adressent directement la mémoire et qu'il fonctionnent à certaines places mémoires auxquelles il faut les charger.

Mais avec l'ATARI ST, cela n'est pas possible. Comment un programmeur peut-il en effet savoir à l'avance à quel endroit son programme sera chargé et être sûr qu'il n'y aura pas d'autre programme à cet endroit ?

De plus, le système d'exploitation doit connaître la taille du programme chargé et la mémoire nécessaire pour son fonctionnement. Si un programme a besoin de mémoire en plus pour sauvegarder du texte par exemple, il ne faut pas que cette mémoire soit recouverte par un autre programme.

Comme on le voit, il ne suffit pas qu'un fichier programme sur disquette contienne uniquement les données du programme. Nous allons expliquer la structure d'un tel fichier dans le chapitre que voici.

Un programme exécutable sur disquette, c'est à dire un fichier du type .PRG, .TOS ou .TTP est divisé en 4 parties. Ces parties sont les suivantes :

<i>Début du fichier :</i>	File Header Programme avec données Tableau symbolique (s'il y en a un) Données de relocation (si elles existent)
---------------------------	---

Fin du fichier

Etudions d'abord la première partie : le File Header.

3.5.1 Le program Header

Le Header est un accessoire au programme qui fait 14 mots de long et qui contient la taille des segments. Voici la structure du Header :

Octet N°	Contenu
\$00,\$01	\$601A, l'instruction assembleur BRA *+\$1A
\$02-\$05	Longueur du segment de programme (text)
\$06-\$09	Longueur du segment de données (data)
\$0A-\$0D	Longueur du segment de mémoire additionnelle (bss)
\$12-\$1B	00, réservé

La première donnée est une instruction en langage machine qui détermine le déroulement du programme au début du segment de programme.

A la suite, on trouve la longueur du segment de programme, appelé segment "TEXT". Ce segment contient le programme proprement dit. Toutes les adresses utilisées par le programme sont déterminées de telle manière que le début du programme soit à l'adresse 0. Les données que contient ce segment ne sont pas modifiées.

La donnée suivante est la longueur du segment de données, nommé segment "DATA". Ce segment doit toujours être relié au programme. Dans un programme en langage machine, c'est la commande DATA qui fait la division entre le segment text et le segment data. Les données sont des données initialisées comme par exemple du texte ou des tableaux. Les données non initialisées comme le tampon de données pour les opérations avec les disquettes ou les mémoires temporaires se trouvent dans le segment suivant.

La longueur de la mémoire additionnelle est indiquée par la donnée suivante du Header. Cette partie de la mémoire est nommée "bss". Après le chargement du programme, cette place mémoire est mise à sa disposition et gardée pour d'autres applications. Son contenu n'est pas défini car il sera modifié par le programme. L'avantage du segment bss par rapport au segment data est que ce secteur n'est pas nécessairement inclus dans le fichier sur disquette. Ainsi, un fichier programme prendra seulement la place mémoire nécessaire.

La cinquième donnée est la longueur de la table symbolique. Cette table est rarement présente car elle ne présente aucune fonction nécessaire au déroulement du programme. Un tableau symbolique est ajouté au programme assemblé ou compilé par l'assembleur ou le compilateur. Les symboles correspondent aux labels utilisés dans le programme source par les routines ou les données.

L'avantage d'un tel tableau est par exemple la recherche d'erreurs car un debugger symbolique comme SID met à la place de chaque adresse utilisée le nom symbolique (ou label). Dès qu'on a terminé la phase de test et de debuggage après le développement du programme, il est préférable de supprimer le tableau symbolique afin que le fichier programme ne soit pas excessivement long.

Chaque donnée du tableau symbolique est longue de 7 mots et contient le nom, le type et la valeur du label :

Octet	Contenu
\$0-\$7	Nom symbolique, se termine avec un zéro
\$8-\$9	Type symbolique (relogeable, global ou externe)
\$A-\$C	Valeur (Adresse, n° de registre, valeur directe, etc...).

Le tableau symbolique d'un programme peut être lu et imprimé avec le programme NM68. Il faut d'abord entrer la commande

NM68 nomdefichier

Si on ajoute >prn: la sortie se fait sur imprimante, sinon elle se fait à l'écran.

Revenons à la structure du program Header. Les octets restants de \$12 à \$1B sont réservés pour des applications ultérieures et restent pour l'instant à zéro.

A la suite du Header, on trouve le programme lui même qui doit pouvoir fonctionner à l'adresse \$0000. Pour que ce programme puisse tourner à l'adresse à laquelle il sera chargé, toutes les adresses absolues qui apparaissent dans le programme seront modifiées. Il suffit pour cela d'ajouter l'adresse de chargement réelle aux adresses se trouvant dans le programme. Mais comment le système d'exploitation peut-il savoir que ces modifications doivent être faites aux adresses absolues du programme ? La réponse se trouve dans la table de relocation.

3.5.2 Le tableau de relocation

Après le tableau symbolique, on trouve le tableau de relocation. Ce tableau contient les différences numériques entre les mots longs qui doivent être relogés. Le premier mot long de ce tableau indique le premier mot long à changer à partir du début du programme. Ensuite, on utilise des octets qui indiquent la différence entre un mot long et le suivant. Si la distance entre deux mots longs est supérieure à 254, l'octet prend pour valeur 1, et cela, tant qu'on n'a pas une valeur inférieure à 255.

Le premier octet qui contient un 0 indique la fin du tableau de relocation. Il indique également la fin du fichier programme.

Quand on charge un programme, le système d'exploitation met ce programme à une place libre et le reloge. La structure du programme en mémoire est un peu différente de celle qu'il avait sur disquette. Avant le programme proprement dit qui suit le segment data et bss, se trouve ce qu'on appelle une page de base. Cette basepage de 256 octets de long est un ensemble d'informations sur la structure réelle du programme en mémoire.

La basepage est organisée comme suit :

Octet	Longueur	Contenu
00	4	adresse de départ de la mémoire de travail
04	4	adresse haute de la mémoire de travail + 1
08	4	adresse de départ du programme
0C	4	longueur du segment de programme en octets
10	4	adresse de départ du segment data
14	4	longueur du segment data en octets
18	4	adresse de départ du segment bss
1C	4	longueur du segment bss en octets
2C	4	Pointeur sur l'environnement string
80	80	Texte de la ligne de commande (P. ex. .TTP)

Toutes les autres données de la basepage que nous n'avons pas indiquées sont réservées.

Il n'y a pas que l'ordinateur à utiliser les données de ce tableau. Un programme peut en tirer un parti intéressant. Le meilleur exemple pour cela est la ligne de commande. Si on utilise le type .TTP pour son programme, le système d'exploitation affiche une zone de dialogue lors du chargement du programme. On peut inscrire des données dans cette fenêtre et ces paramètres seront pris en compte par le programme.

Pour accéder à l'adresse de la ligne de commande, il faut mettre la séquence suivante au début de son programme :

```
run:  MOVE.L 4(sp),A0    ; adresse de la basepage
      LEA   $80(A0),A0
```

A0 contient alors l'adresse de la ligne de commande. On peut continuer à la traiter.

3.6 LE FORMAT DU DISQUE DUR

Intéressons-nous maintenant au disque dur. En raison de sa grande capacité mémoire, il n'est pas structuré aussi simplement qu'une disquette. Un disque dur est divisé en quatre parties au plus qui contiennent chacune un boot secteur. Ces parties sont nommées partitions.

La premier secteur d'un disque dur (secteur logique 0) contient les informations sur la structure du disque. Ces informations se présentent comme suit :

Octet	Nom	Signification
\$1C2	hd_siz	taille totale du disque dur en secteurs logiques
\$1C6	p0_flg	la partition 0 existe si p0_flg>0 si le bit 7 est positionné, le bootage se fait ici.
\$1C7	p0_id	identificateur de la partition
\$1CA	p0_st	numéro de secteur logique du premier secteur de la partition
\$1CE	p0_siz	taille de la partition en secteurs
\$1D2	p1_flg	idem, pour partition 1
\$1D3	p1_id	
\$1D6	p1_st	
\$1DA	p1_siz	
\$1DE	p2_flg	idem, pour partition 2
\$1DF	p2_id	
\$1E2	p2_st	
\$1E6	p2_siz	
\$1EA	p3_flg	idem, pour partition 3
\$1EB	p3_id	
\$1EE	p3_st	
\$1F2	p3_siz	
\$1F6	bsl_st	secteur de départ de la "bad sector list"
\$1FA	bsl_cnt	nombre de secteurs défectueux

La "bad sector list" est inscrite sur le disque dur lors du formatage. Elle contient la liste des secteurs défectueux qui n'ont pas pu être formatés. Le tableau se trouve la plupart du temps à la fin du disque dur.

Avec la variable `p*_flg`, le système d'exploitation se rend compte si la partition existe (`p*_flg` différent de zéro). Le premier secteur d'une partition contient un boot secteur dans lequel se trouve la BPB. Le système d'exploitation boote à partir du premier boot secteur dont `p*_flg` ait le bit 7 positionné.

Un conseil : vous trouverez un programme pour analyser et afficher les paramètres de partition au chapitre 5.1.1.4 : partition analysator.

4. Les lecteurs de disquette

Le mode de mémorisation des données le plus connu est sans doute la disquette. Cette mémoire de masse de 3 pouce 1/2 ou de 5 pouces 1/4 (il existe également des disquettes 3 et 8 pouces mais elles ne servent à rien pour les possesseurs d'ATARI ST) présente certains avantages.

D'abord le prix : étant donné qu'une disquette 3 pouce 1/2 coûte 20 FF environ (simple face), cela fait pour une capacité de 360 Ko, à peine plus de 5 centimes du Koctet. Avec les disquettes 5 pouces 1/4, c'est encore moins cher. Il n'y a aucun problème pour brancher un lecteur de disquettes 5 pouces 1/4 à l'ATARI ST. Cet avantage de prix peut donc intéresser certains utilisateurs et plusieurs personnes possédant à la fois un lecteur 3 pouces 1/2 et un lecteur 5 pouces 1/4.

Un autre avantage par rapport au disque dur est la possibilité de changer de disquettes. Ainsi, avec un seul lecteur, on peut utiliser un nombre illimité de disquettes et donc gérer un nombre illimité de données. D'autre part, les disquettes sont très pratiques pour échanger des programmes ou des données.

Mais il y a bien sûr un inconvénient à cela. Si l'on excepte les cassettes, les disquettes sont le support de masse le plus lent. Les lecteurs de disquette de l'ATARI ST supportent aisément la comparaison avec leurs concurrents car ils offrent un transfert des données très rapide en raison de nombreux trucs employés par l'ATARI ST.

Avançons plus profondément dans le monde des disquettes.

4.1 FONCTIONS ET STRUCTURE

Lorsque l'ordinateur doit lire des données sur disquettes, le lecteur de disquette exécute certaines opérations.

Tout d'abord, le moteur du lecteur est mis en route. On remarque d'ailleurs qu'avec deux lecteurs de disquette, les deux moteurs se mettent en route. La raison est que le signal correspondant est envoyé aux deux lecteurs à la fois par l'ordinateur.

Cela a l'avantage que lors de la copie, les deux lecteurs tournent en même temps à la vitesse optimale, si bien qu'on gagne du temps.

Dans le processus suivant, il faut cependant travailler avec un seul lecteur. Cela est spécifié grâce à la sortie "drive-select". Lorsqu'un lecteur est adressé, sa lampe "busy" s'allume, ce qui indique qu'il est en route.

Il faut ensuite décider quelles données doivent être lues. L'ordinateur doit indiquer très précisément sur quelle piste se trouvent les données. Ces pistes sont des anneaux virtuels inscrits de manière concentriques sur la disquette. La tête de lecture/écriture qui est positionnée à un endroit donné de la disquette avec un petit bras suit exactement chacune de ces pistes.

Les données sauvegardées sont divisées en unités magnétiques sur cette piste à l'aide d'un système particulier. Pour que les données puissent être manipulées plus facilement, chaque piste est divisée en secteurs. Chaque piste comporte 9 secteurs qui contiennent chacun 512 octets de données effectives. Par "donnée effective" nous entendons les données utilisables car le secteur comporte d'autres données qui ne sont pas accessibles normalement. Nous décrirons ces données dans un autre chapitre.

La tête de lecture/écriture qui glisse sur la disquette en rotation comporte une petite bobine. Cette bobine sert de capteur magnétique qui reconnaît les bits mémorisés sous forme d'impulsions magnétiques. Ce principe ressemble à celui des bandes magnétiques, mais avec les disquettes, il faut une meilleure précision. En effet, il est possible de mettre environ 3 million de bits sur une surface de 30 cm² (disquettes 3 pouces 1/2). Ainsi, un octet (qui est constitué de 8 bits) couvre une surface de 0,008 mm² !

Si l'ordinateur a besoin de données, il doit accéder à un secteur donné. Par un processus assez compliqué, le contrôleur de disquette, qui est un circuit de l'ordinateur, distingue quel est le secteur désiré parmi les données qui viennent de la tête de lecture. Les bits correspondants sont lus et envoyés par paquets de 512 Koctets à l'ordinateur.

Tous ces processus représentent un gros problème pour le constructeur de lecteurs de disquette.

La partie mécanique qui positionne la tête de lecture doit accéder exactement au secteur désiré (qui a une largeur de 0,2 mm environ). Ensuite, les impulsions magnétiques qui viennent de la surface de la disquette doivent être reconnues très précisément comme étant des oui ou des non, ce qui fait à peu près 0,5 microseconde par bit pour une vitesse de 300 rotations par minute.

L'électronique doit ensuite distinguer les données recherchées à partir de tous ces bits. Cela est réalisé avec ce qu'on appelle l'octet de synchronisation qui se trouve au début de chaque secteur pour une disquette. Nous étudierons la structure des données dans la partie suivante, mais nous allons pour l'instant nous intéresser au hardware.

Un lecteur de disquettes est donc un système assez compliqué comme nous l'avons vu. Mais nous étudierons simplement la structure générale du système nécessaire à l'utilisation des disquettes.

4.2.1 Le circuit DMA

Commençons par l'ordinateur lui même. Le lecteur de disquette envoie les données par le câble de liaison sous la forme d'un flux d'octets. Ces données doivent alors être entreposées quelque part dans la mémoire afin de pouvoir être utilisées. La plupart des ordinateurs font ainsi : l'unité centrale (Central Processing Unit = CPU) récupère les données et les entrepose en mémoire. Cela signifie cependant que durant le transfert des données, l'ordinateur est complètement occupé.

L'ATARI ST, lui, fonctionne différemment. La réception et la gestion des données en mémoire est prise en charge par un composant spécial qui dispose comme le CPU d'un accès direct à la mémoire. Ce circuit s'appelle DMA (DMA = Direct Memory Access). Il fonctionne de façon indépendante (bien entendu avec une instruction du CPU) si bien que le CPU peut être utilisé pour un tout autre travail lors du transfert de données. De plus, le circuit DMA peut effectuer le travail beaucoup plus rapidement que le CPU.

Grâce à ce système technique employé par les constructeurs de l'ATARI, la vitesse de transfert est très élevée, ce qui est sensible avec un lecteur de disquettes mais plus encore avec un disque dur.

Le circuit DMA occupe les places mémoire suivantes dans l'ATARI ST :

- \$FF8604 FDC access/ sector count.* Cette adresse sert à l'accès aux registres du DMA ou du FDC qui est sélectionné par :
- \$FF8606 DMA mode/ status.* Les bits 0-2 indiquent l'état du circuit DMA ou FDC lors de la lecture. Si on écrit dans ce registre 16 bits, le mode DMA est activé.
- \$FF8609 Vecteur mémoire du DMA. Octet de poids fort.*
\$FF860B Vecteur mémoire du DMA. Octet de poids moyen.
\$FF860D Vecteur mémoire du DMA. Octet de poids faible.

Ces trois octets donnent l'adresse (sur 24 bits) à laquelle (ou à partir de laquelle) doivent être transférées les données par le DMA. Ces octets doivent être spécifiés dans l'ordre LO,MID,HI.

Le circuit DMA utilisé dans l'ATARI ST est juste à côté de l'interface servant au transfert de données avec le disque dur. Les lecteurs de disquettes n'y sont pas directement reliés. Entre la prise des lecteurs de disquette et le DMA se trouve un circuit qui prépare les données en série de manière à les réceptionner ou à les envoyer. Ce circuit est nommé le contrôleur de disquettes et il contrôle les fonctions du lecteur de disquettes. La programmation de ces deux composants est tellement semblable que nous l'évoquerons dans le chapitre suivant.

4.2.2 Le contrôleur de disquettes

Ce gros chapitre traite du contrôleur de disquette WD1772 (que nous appellerons FDC, Floppy Disk Controller, ou contrôleur, par la suite) utilisé dans l'ATARI ST. Dans quel autre livre pourrait-on en effet parler aussi longuement d'un seul composant ?

Nous avons réuni toutes les informations disponibles sur ce circuit. Mais elles ne suffisaient pas bien entendu pour une description complète du circuit : la théorie et la pratique sont souvent très différentes. Ainsi, il était nécessaire d'acquérir une expérience du WD 1772 pour vérifier la véracité des informations et pour découvrir des différences éventuelles.

Le chapitre qui en résulte contient une masse d'informations dépassant de loin les désirs des gens qui veulent avoir seulement une vue globale du FDC. Par contre, le programmeur qui désire contrôler directement le FDC et qui ne se plaindrait que du manque d'informations à son sujet trouvera ici tout le savoir nécessaire sur ce circuit.

Pour le transfert de données normal entre les lecteur de disquette et l'ATARI ST, il n'est pas nécessaire de programmer le FDC par soi-même. On peut utiliser les fonctions du BIOS ou du XBIOS du système d'exploitation.

Mais le système d'exploitation n'exploite pas toutes les possibilités du FDC. L'utilisateur qui désire par exemple développer un programme de copie rapide ou bien une protection aura besoin des fonctions manquantes. Si vous voulez créer des formats de disquette particuliers, il ne faudra pas utiliser les routines du système d'exploitation pour formater les pistes mais travailler par vous même.

Pour relier toutes ces fonctions à un programme d'application, il faut utiliser la programmation directe du contrôleur. Mais pour cela, il faut connaître parfaitement les commandes du FDC et leur fonctionnement. Ces connaissances évitent une expérimentation inutile conduisant, après plusieurs heures de programmation, à la conclusion que l'on ne peut pas mettre son projet en pratique.

Voici un exemple d'un tel projet : "Si je charge toutes les pistes d'une disquette en mémoire à l'aide de la commande READ TRACK et que je les écris sur une autre disquette avec la commande WRITE TRACK, j'ai le programme de copie le plus rapide possible. De plus, j'ai la possibilité de faire un BACKUP de mes programmes protégés. La commande READ TRACK lit en effet toutes les informations d'une piste (donc la protection incluse) et la commande WRITE TRACK les réécrit totalement !"

Si vous écrivez un programme qui fonctionne selon ce principe, vous constaterez que les copies sont inutilisables. Il n'est pas possible de copier une disquette protégée par ce moyen.

Si vous étudiez ce chapitre et si vous comprenez l'effet des commandes du FDC, vous comprendrez pourquoi le "programme de copie" décrit plus haut ne peut pas fonctionner comme prévu. Nous sommes convaincus que notre description du contrôleur est suffisamment complète pour prévenir des "projets" de ce type.

Venons-en enfin à la description du WD 1772. Ce circuit développé par WESTERN DIGITAL contient toutes les fonctions utiles à la gestion d'un lecteur de disquettes 5 pouces 1/4. Bien entendu, les disquettes 3 pouce 1/2 peuvent être contrôlées sans problème avec ce circuit comme le prouve l'ATARI ST. Mais cette possibilité n'est pas due particulièrement au WD 1772 mais aux développeurs du lecteur 3 pouce 1/2, la firme SONY. Elle a constaté effectivement que la pénétration du marché serait meilleure avec une interface compatible avec les lecteurs 5 pouce 1/4. en ce qui concerne l'utilisateur d'ATARI ST, cela signifie qu'il peut utiliser des lecteurs 5 pouce 1/4.

Mais méfiez-vous des anciens modèles que vous possédez peut-être ou que vous pouvez récupérer. Si vous tentez de brancher un tel modèle, voici les problèmes que vous pouvez rencontrer :

Le WD 1772 dont la version standard WD 1770 est compatible de façon logicielle avec les séries WD 179x et WD 279x dispose de "stepping rates" plus courts. ces "stepping rates" sont les délais laissés par le contrôleur au lecteur de disquettes pour déplacer la tête de lecture d'une piste en avant ou en arrière.

Avec le WD 1772, les quatre délais programmables sont de 2, 3, 5 et 6 ms alors que le WD 1770 avait des délais de 6, 12, 20 et 30 ms.

Cela signifie que les lecteurs doivent être en mesure de déplacer la tête de lecture dans un temps maximum de 6 ms. Vérifiez dans la notice technique du lecteur de disquettes. Le délai est indiqué dans la rubrique "TRACK TO TRACK".

Après ce conseil accessoire, intéressons-nous au FDC et commençons avec un petit résumé des caractéristiques de ce circuit.

Les caractéristiques du WD 1772 sont :

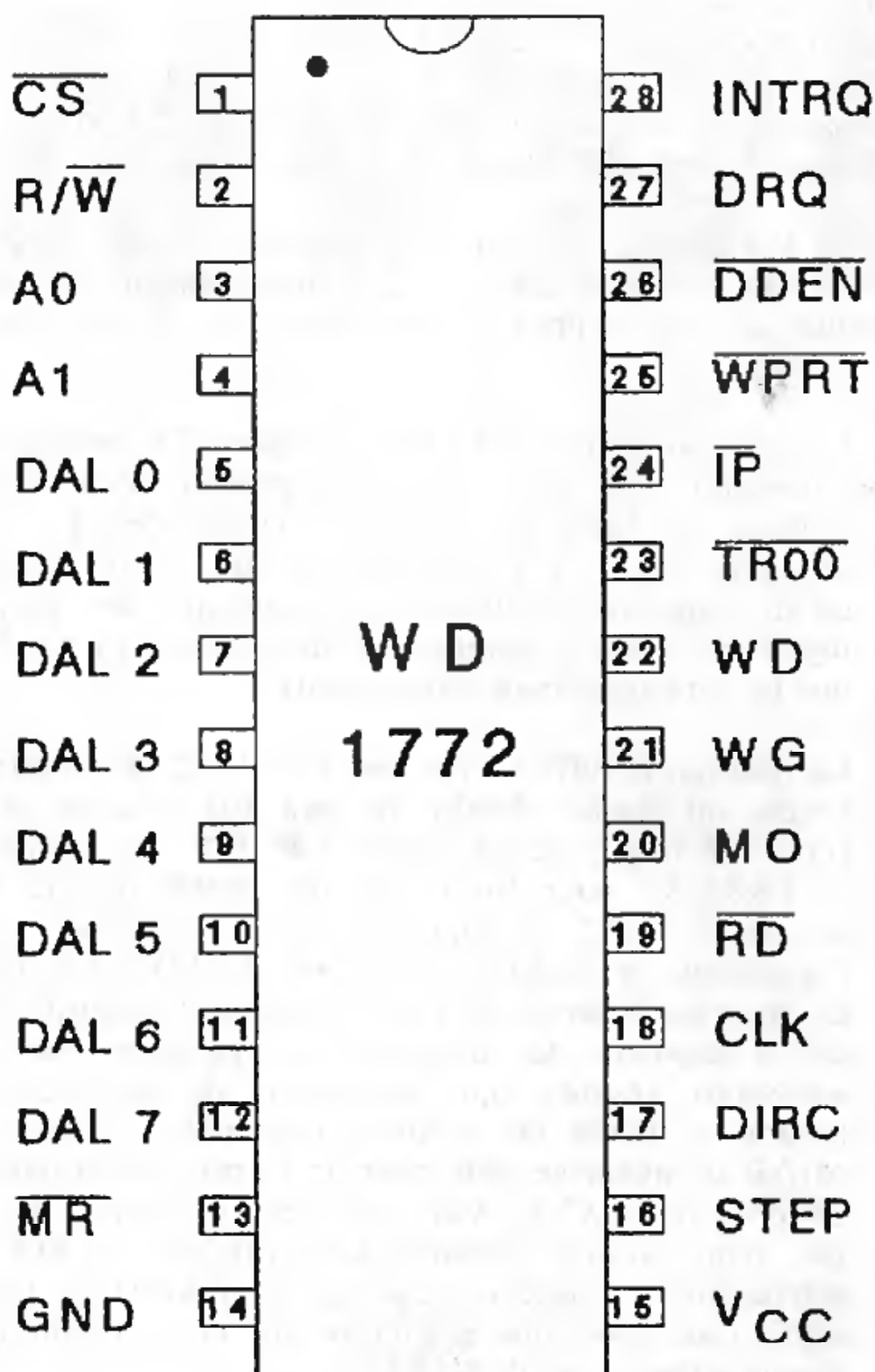
- Circuit 28 broches dual-in-line.
- Alimentation 5 V.
- Séparateur de données digital inclus.
- Précompensation en écriture intégrée.
- Écriture en simple et double densité.
- Contrôle du moteur intégré.
- Longueur des secteurs de 128, 256, 512 ou 1024 octets.
- "Stepping rates" plus rapides (2, 3, 5 et 6 ms).

Comme nous l'avons dit, il s'agit seulement des caractéristiques en gros. Nous allons expliquer deux points immédiatement. Les autres seront évoqués dans le chapitre suivant concernant les fonctions du FDC.

1. Le fait que le circuit du FDC comporte 28 broches n'est intéressant que pour le développement d'un système utilisant un FDC. Le fait qu'un circuit de 28 broches soit plus facile à positionner sur un circuit imprimé qu'un composant de 40 broches par exemple sera un point important pour le développeur de systèmes (à condition que les caractéristiques conviennent).
2. La possibilité offerte par le WD 1772 de traiter en simple ou double densité ne sera pas évoquée par la suite. La raison en est simple : le FDC est utilisé par l'ATARI ST pour fonctionner en double densité. Pour utiliser le FDC en simple densité, il faudrait ouvrir l'ordinateur et modifier le câblage du FDC. Le résultat de ce procédé serait de pouvoir utiliser seulement 50 % de la capacité des disquettes. A première vue, cela semblerait absurde (qui accepterait de bon cœur de perdre la moitié de la place disponible ?) mais c'est parfois un avantage pour créer un format compatible avec un ordinateur XYZ. Mais de telles applications ne sont pas d'une utilité extrême. Le principe du FDC est suffisamment complexe pour qu'on n'alourdisse pas les explications avec une possibilité qui ne sera sans doute jamais utilisée avec l'ATARI ST.

4.2.1.1 Connexions

Après cette vue globale du WD 1772, nous allons vous le décrire en détail avec le brochage.



PIN 1 CS (CHIP SELECT)

Un signal LOW sur cette broche sélectionne le circuit et autorise l'accès à ses registres. La broche CHIP SELECT existe sur tous les circuits périphériques (dont font partie naturellement les circuits de mémoire). Etant donné qu'ils sont tous reliés au bus de données du processeur, il y aurait un certain cafouillage s'ils participaient tous au transfert de données en même temps. C'est pourquoi la broche CHIP SELECT permet de définir un seul circuit avec lequel les transferts de données vont être effectués. Dans l'ATARI ST, le FDC est sélectionné à partir du contrôleur DMA.

PIN 2 R/W (READ/WRITE)

Cette broche contrôle le sens des données. Avec un signal HIGH, le contenu du registre sélectionné est envoyé vers DAL0-DAL7 alors qu'avec un signal LOW, les données de DAL0-DAL7 sont récupérées dans le registre sélectionné.

PIN 3,4 A0,A1 (ADRESS 0,1)

Ces deux entrées sélectionnent le registre du FDC. Le WD 1772 contient 5 registres adressables par l'ordinateur. Mais comme on ne peut sélectionner que 4 registres avec 2 broches, on a mis deux registres dans une adresse (A0=0 et A1=0). Pour distinguer ce registre, on inverse la valeur de la broche R/W.

				READ	WRITE
CS	A1	A0	R/W = 1	R/W = 0	
0	0	0	○	○	registre d'état
0	0	1	○	○	registre de commandes
0	1	0	○	○	registre de pistes
0	1	1	○	○	registre de secteurs
0	1	1	○	○	registre de données

Il en résulte que le registre de commandes ne peut pas être lu et que le registre d'état ne peut pas être écrit.

Ces broches ne sont pas directement reliées au bus d'adresses du processeur. Elles sont reliées au contrôleur DMA. Les registres du FDC sont sélectionnés à partir d'un registre de contrôle dans le contrôleur DMA.

PIN 5-12 DAL0-DAL7 (DATA ACCESS LINE 0-7)

Ces 8 broches représentent le bus de données bi-directionnel. Ce bus sert à transférer les données entre l'ordinateur et les registres du FDC. Ces broches sont reliées au circuit DMA comme les broches d'adresse. On utilise le registre de données du DMA pour accéder aux registres du FDC. On sélectionne les registres du FDC avec le registre de contrôle du DMA.

PIN 13 MR (MASTER RESET)

Etant donné que le contenu des registres du FDC est aléatoire à la mise en route, il faut d'abord les mettre dans un état prédéfini. On obtient cela en envoyant une impulsion LOW (d'au moins 50 us) sur cette broche.

Cela se fait à la mise en route de l'ATARI ST. L'instruction RESET du 68000 permet bien entendu de remettre à zéro le FDC à tout moment. Mais il faut faire attention au fait que tous les autres composants qui sont reliés à la broche RESET subiront le même sort et seront initialisés avec des valeurs prédéfinies.

PIN 14 GND (GROUND)

Masse.

PIN 15 Vcc (POWER SUPPLY)

Entrée d'alimentation à +5V.

PIN 16 STEP (STEP)

Cette broche de sortie envoie une impulsion indiquant que la tête de lecture/écriture doit être déplacée d'un pas.

PIN 17 DIRC (DIRECTION)

Cette broche précise au FDC du lecteur connecté dans quelle direction doit se déplacer la tête de lecture/écriture après un signal STEP. Si le signal à cette broche est HIGH, l'impulsion STEP indique un déplacement vers le centre de la disquette alors qu'un signal LOW indique un déplacement vers l'extérieur.

PIN 18 CLK (CLOCK)

Après une instruction, il y a, tout comme dans un microprocesseur, des microprogrammes qui fonctionnent dans le FDC. Pour cette raison, le FDC est muni d'une fréquence contrôlant l'exécution comme pour un microprocesseur. Il utilise également cette fréquence d'impulsion pour le timing des données séries. L'impulsion n'est pas créée dans le FDC lui-même mais exécutée à partir de l'extérieur par la broche CLK. La fréquence est de 8 MHz.

PIN 19 RD (READ DATA)

Le signal envoyé par la tête de lecture/écriture du lecteur est dirigé vers cette broche. Le séparateur de données du FDC sépare les impulsions de fréquence et de données qui sont toutes les deux contenues dans le signal.

PIN 20 MO (MOTOR ON)

Cette sortie est utilisée pour le contrôle du moteur. Avec un signal HIGH, les moteurs des lecteurs sont automatiquement mis en route pour les opérations d'écriture, de lecture et de recherche.

PIN 21 WG (WRITE GATE)

Les impulsions de données que le FDC envoie vers le lecteur ne vont pas directement à la tête de lecture/écriture mais d'abord vers un amplificateur d'écriture. Si aucune donnée ne provient du FDC, l'entrée de cet amplificateur est ouverte. Mais les amplificateurs dont l'entrée est libre ont la particularité désagréable d'être sensibles aux courants parasites qui peuvent venir par exemple du câble de liaison. Pour éviter que de tels courants parasites n'atteignent la tête de lecture/écriture et ne détruisent les données, les lecteurs sont pourvus d'un interrupteur qui règle les amplificateurs. Lors des opérations d'écriture, c'est-à-dire lorsqu'on écrit seulement les données, la WRITE GATE du FDC est mise au niveau HIGH. Ainsi, la régulation de l'amplificateur est supprimée et les données disponibles à partir de WRITE DATA peuvent être traitées.

PIN 22 WD (WRITE DATA)

Cette sortie envoie les informations à écrire vers le lecteur. Ces informations sont constituées d'impulsions de données et d'impulsions de fréquence.

PIN 23 TR00 (TRACK 00)

Les lecteurs disposent d'un rayon optique qui est coupé par le bras sur lequel se trouve la tête de lecture/écriture dès que la tête est positionnée sur la piste zéro. Dans ce cas, l'entrée TR00 du FDC du lecteur est mise à l'état bas (LOW).

PIN 24 IP (INDEX PULSE)

Le lecteur envoie une impulsion à chaque tour du moteur. Cette impulsion est convertie par le contrôleur lors de ses opérations. Ainsi, il reconnaît lors de la lecture ou écriture le début d'une piste (L'index-impuls est la réponse pratique à la question : où commence une piste circulaire ?).

De même, en comptant les index-impuls, il peut se rendre compte à quel moment le moteur a atteint la vitesse de croisière et éventuellement attendre avant le transfert de données.

L'index-impuls est indépendant de la disquette sur un lecteur 3 pouces 1/2. Il n'y a donc pas de trou dans la disquette (comme avec les disquettes 5 pouces 1/4) pour permettre un test opto électronique.

PIN 25 WPRT (WRITE PROTECT)

Si le FDC doit effectuer une opération d'écriture, il teste d'abord cette broche. Si elle est à l'état LOW (disquettes protégées), le contrôleur interrompt l'opération d'écriture.

PIN 26 DDEN (DOUBLE DENSITY ENABLE)

L'état de cette broche spécifie le format avec lequel le FDC travaille. Avec un LOW, le contrôleur est en mode DOUBLE DENSITY (double densité) alors qu'un HIGH indique qu'il fonctionne en SINGLE DENSITY (simple densité).

Sur l'ATARI ST, le WD 1772 est mis en mode double densité car DDEN est relié à la masse et donc au signal LOW.

PIN 27 DRQ (DATA REQUEST)

Cette sortie dont l'état est spécifié par un bit dans le registre d'état a la signification suivante lorsqu'elle est mise à l'état HIGH par le FDC :

- a) Lors d'une opération de lecture, le registre de données contient un octet qui doit être lu (Registre de données plein).
- b) Lors d'une opération en écriture, le registre de données est vide et il faut charger à l'intérieur l'octet suivant à écrire.

Une lecture ou une écriture dans le registre de données remet à zéro la sortie DRQ et le bit d'état DRQ.

La possibilité de DMA du WD 1772 facilite la manipulation de cette sortie. Elle est reliée au contrôleur DMA dans l'ATARI ST. Alors qu'il faudrait normalement tester le bit d'état DRQ à chaque opération de lecture ou d'écriture, cette opération est effectuée par le DMA qui est contrôlé à partir de la sortie DRQ.

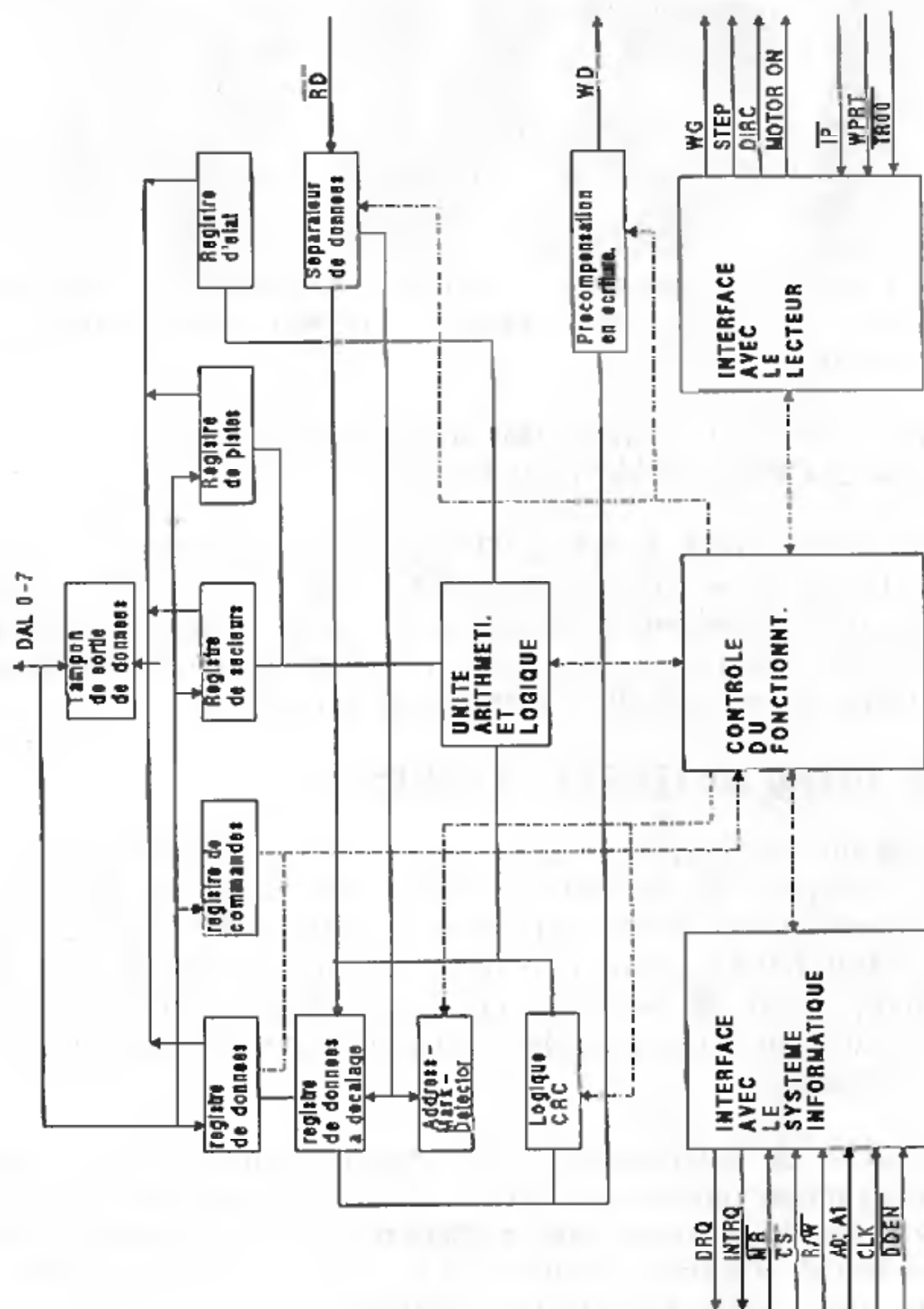
PIN 28 INTRQ (INTERRUPT REQUEST)

Après chaque commande exécutée, cette broche est mise à l'état HIGH. Lorsqu'on lit le registre d'état, cette sortie est remise à l'état normal. Cette broche est reliée au port d'entrée/sortie (bit 5) du MFP 68901. Pour reconnaître quand le FDC a fini sa commande, ce bit est testé dans une boucle. Il faut remarquer que ce bit est inversé. La commande est donc terminée lorsque le bit du port est effacé.

Il est possible de programmer le MFP de telle manière qu'il exécute une interruption lorsque la sortie INTRQ est à l'état HIGH. On évite ainsi de tester continuellement le bit de port et on peut effectuer d'autres opérations. Le contrôle des interruptions n'est pas utilisé par le système d'exploitation.

4.2.1.2 Organisation

Pour mieux comprendre la programmation du FDC que nous expliquerons en détail ultérieurement, il est utile d'analyser le schéma synoptique du DW 1772 et d'expliquer les différents blocs de fonctions.



Data Shift Register (DSR)

Ce registre à décalage de 8 bits sert à récupérer les données série qui proviennent de l'entrée READ DATA lors d'une opération de lecture. Lors d'une opération d'écriture, le contenu de ce registre est envoyé en série par la broche WRITE DATA. Dans certaines opérations, le data shift register est utilisé comme mémoire temporaire.

Data Register (DR)

Dans les opérations de lecture/écriture, ce registre est utilisé comme mémoire tampon. Lorsque le DSR a récupéré 8 bits après une opération de lecture, cette information est mise dans le registre de données. Dans une opération d'écriture, l'octet suivant du registre de données est transféré dans le DSR à chaque fois que le DSR vient d'envoyer un octet.

Dans une opération de recherche, (voir commande SEEK), le registre de données contient le numéro de la piste désirée.

Track register (TR)

Ce registre contient normalement le numéro de la piste sur laquelle se trouve la tête de lecture/écriture. Comme le laisse supposer le "normalement", il y a ici aussi des exceptions. Pour que le Track Register soit maintenu à l'état courant, il est incrémenté de 1 lorsque la tête se déplace vers l'intérieur et diminué de 1 lorsqu'elle se déplace vers l'extérieur.

Alors que cela se fait automatiquement avec les commandes RESTORE et SEEK, ce n'est vrai pour les commandes STEP, STEP-IN et STEP-OUT que lorsque le update-flag est positionné.

Dans les opérations de lecture, écriture et verify, le contenu de registre de piste est comparé avec le numéro de piste contenu dans l'ID.

Le registre de piste peut être lu et écrit mais il ne doit pas être écrit durant une opération.

Sector Register (SR)

Lors des opérations de lecture et d'écriture, ce registre contient le numéro du secteur désiré qui est comparé avec le numéro de secteur décrit dans l'ID. Après une instruction READ ADDRESS, le numéro de piste indiqué dans l'ID est inscrit dans ce registre.

Le registre de secteur peut être lu et écrit mais il ne faut pas y écrire pendant une opération.

Command Register (CR)

Ce registre contient la commande en cours d'exécution. Il peut seulement être écrit car une lecture sélectionne automatiquement le registre d'état. Le registre de commande ne doit pas être écrit pendant une opération sauf si la commande à écrire est l'instruction FORCE INTERRUPT.

Status register (STR)

Les informations se trouvant dans ce registre indiquent l'état du FDC ou du lecteur. Les différents bits de ce registre sont partiellement indépendants de la commande en cours d'exécution. Le registre d'état peut seulement être lu et l'octet lu a les significations suivantes :

Bit 7 MOTOR ON

Ce bit reflète l'état de la sortie MOTOR ON. Il reste positionné pendant 1 à 2 s après une commande, c'est à dire après que le bit busy soit effacé.

Bit 6 WRITE PROTECT

Ce bit indique après chaque opération d'écriture si le lecteur contient une disquette protégée. S'il est positionné, cela signifie que l'opération d'écriture n'a pas été effectuée. Le bit WPRT est éventuellement positionné après une commande du type 1 (dans le cas d'une disquette protégée).

Ce bit est effacé lorsqu'une opération a été effectuée avec une disquette non protégée.

Bit 5 SPIN UP/RECORD TYPE

SPIN UP : Avec les commandes du type 1, ce bit est positionné à la fin de la séquence spin-up. On voit ainsi que le moteur du lecteur de disquette a (sans doute) atteint sa vitesse de rotation optimale.

RECORD TYPE : Après une instruction READ SECTOR, on voit si le champ de données commence avec une marque normale ou effacée.

Bit 5 = 0, Data-mark normale (\$FB)

Bit 5 = 1, Data-mark effacée (\$F8)

Bit 4 RECORD NOT FOUND (RNF)

Lorsqu'un champ d'ID correct a été rencontré, ce bit est positionné. Cela peut être le cas lors d'une commande READ ADDRESS, READ SECTOR ou WRITE SECTOR.

Après une instruction READ SECTOR, le bit RNF peut être positionné malgré un ID correct. C'est le cas lorsqu'aucune data-mark n'a été rencontrée parmi les 43 octets qui suivaient le dernier octet de CRC de l'ID.

Bit 3 CRC ERROR

Ce bit est positionné lorsque le contenu du champ CRC (dans le champ de donnée ou l'ID) ne correspond pas au contenu du registre CRC.

Bit 2 LOST DATA/TRACK 00

LOST DATA : Lorsqu'il n'y a pas de réaction à une commande du type 2 ou du type 3 dans le temps accordé pour une demande de données (indiqué par la sortie DRQ ou le bit d'état DRQ), ce bit est positionné.

TRACK 00 : Avec une commande du type 1, ce bit est positionné lorsque la tête de lecture/écriture se trouve sur la piste zéro.

Bit 1 DATA REQUEST/INDEX

DATA REQUEST : Avec les commandes du type 2 ou du type 3, ce bit est positionné lorsque des données sont disponibles ou sont utilisées. Il est effacé par la lecture ou l'écriture du registre de données.

INDEX : Avec les commandes du type 1, ce bit est positionné lorsqu'un index-impuls est rencontré.

Bit 0 BUSY

Ce bit est positionné durant l'exécution d'une commande.

CRC LOGIC

Pour éviter des erreurs de lecture, il faut utiliser un processus qui assure une grande sécurité des données. Le processus dont nous voulons parler fonctionne de telle manière qu'une somme-test sur 16 bit est établie à partir d'un algorithme défini. Cette somme est inscrite sur la disquette derrière les données.

Lorsqu'on relit ces données, le même algorithme permet de redéterminer la somme. Si cette somme correspond à celle inscrite sur la disquette, on est assuré presque à 100% que les données ont été bien lues. En raison de la relative complexité de cet algorithme, il est très invraisemblable qu'on ait de mauvaises données malgré une somme-test égale.

Cette somme test sur 16 bit se nomme "Cyclic Redundancy Check (CRC)". C'est la logique CRC qui est responsable de la création et du contrôle de la somme-test.

Pour des raisons de rapidité, le calcul est effectué par le HARWARE à partir du polynôme : $CRC(x) = x^{16} + x^{12} + x^5 + 1$.

ARITHMETIC/LOGIC UNIT (ALU)

L'ALU est utilisée d'une part pour la modification des registres (augmentation, réduction) et d'autre part pour la comparaison entre les registres et les informations contenues sur disquette dans les champs d'ID.

ADDRESS MARK DETECTOR

La partie la plus importante du FDC est sans doute le détecteur. Comme l'indique le nom, cette partie a la faculté de pouvoir reconnaître une address mark. Une telle marque distingue le début du champ d'ID (Index Address Mark) ou le début d'un champ de données (Data Address Mark).

Mais pourquoi utilise-t-on un détecteur spécial ? Prenons pour exemple la valeur \$FE que le contrôleur interprète comme Index Address Mark. Même sans détecteur particulier, il n'est pas difficile de retrouver ce \$FE.

Mais le problème devient clair lorsqu'on pense que cette valeur peut apparaître n'importe où dans un champ de données. Mais il ne faudra pas toujours la considérer comme Index Address Mark. Comment est-il donc possible de distinguer un \$FE d'un autre \$FE ? Hé bien, c'est justement ce à quoi sert l'address-mark-detector.

Comme nous l'avons déjà indiqué, l'information écrite n'est pas seulement constituée d'impulsions de données. Elle contient également des impulsions de fréquence. Ces impulsions sont séparées des autres par le séparateur de données lors des opérations de lectures et envoyées vers l'address-mark detector.

Avec une commande WRITE TRACK, les valeurs supérieures à \$F4 sont traitées de manière particulière. Ces valeurs sont écrites sans impulsion de fréquence. Elles sont donc constituées exclusivement d'impulsions de données.

Lorsque cette information est lue, le séparateur de donnée n'envoie aucune impulsion de fréquence car il n'y en a aucune. L'address-mark-detector n'est activé que lorsque l'impulsion de fréquence est absente. Il ne peut donc reconnaître une address mark que si elle est écrite sans impulsion de fréquence.

Mais il y a un autre problème que vous ignorez sans doute car nous n'avons parlé pour l'instant que d'octets complets. Mais les octets sont inscrits en série et le début d'un octet n'est pas marqué.

Lorsqu'on récupère 8 bits dans le DSR après une opération de lecture, on ne peut pas savoir s'il s'agit d'un octet. Il peut s'agir de deux séries de 4 bits provenant de deux octets différents.

L'address mark detector ne reconnaîtrait une address mark que si les bits de données étaient par hasard synchronisés avec les octets. Etant donné qu'il ne faut rien laisser au hasard, il faut avoir une possibilité de reconnaître le début d'un octet. Cela est obtenu grâce aux octets de synchronisation. Ces (ils sont 3) octets sont écrits avant chaque address mark lors du formatage de chaque piste. Les "SYNC Bytes" ne contiennent pas d'impulsion de fréquence comme les Address marks et activent donc l'Address mark detector.

Le contrôleur qui connaît déjà l'état de l'address mark detector lit alors tous les bits de données jusqu'à ce que le DSR contienne un octet de synchronisation dont il connaît la valeur. A partir de ce moment, le bit suivant est obligatoirement le premier bit de l'octet suivant.

Mais il faut remarquer quelque chose. Pendant que le détecteur est activé, les octets lus peuvent être falsifiés (voir commande READ TRACK). Pourquoi et quand ? Lorsque le détecteur est activé par le manque d'impulsion de fréquence, le FDC considère que les octets de synchronisation sont suivis d'une address mark. Dans la phase de synchronisation, c'est-à-dire durant la reconstruction de l'octet de synchronisation, certains bits de données sont ignorés. Si le détecteur se trompe dans la recherche de l'address mark dans un champ de données, les données sont naturellement modifiées jusqu'à ce qu'on reconnaisse que "l'alarme" était fausse. Etant donné que l'address mark detector ne doit pas être trop sensible sur le manque d'impulsion de fréquence, il est désactivé par avance (lorsque les champs d'ID et de données sont lus).

DATA SEPARATOR

Le séparateur de données a déjà été décrit dans les explications concernant l'address mark detector.

C'est pourquoi nous indiquerons seulement rapidement qu'il a pour objet de séparer l'impulsion de fréquence de l'impulsion de données dans le signal reçu. Cette impulsion de fréquence est utilisée pour contrôler l'address mark detector.

L'interface avec le système

L'interface avec le processeur est constituée de 8 ports bi-directionnels, des deux broches d'adresse (A0,A1), de l'indicateur de données (DRQ), de l'indicateur d'interruption (INTRQ), du chip selcet (CS), de la broche de lecture/écriture (R/W), de l'entrée clock (CLK) et de l'entrée MASTER RESET (MR). Ces broches permettent de transférer des données et des signaux de contrôle entre le processeur et le FDC.

L'interface avec le lecteur

Les informations que le FDC récupère à partir du lecteur sont :

- a) Si la disquette est protégée (WPRT=1)
- b) Si la tête de lecture/écriture se trouve sur la piste 0 (TR00 = 0)
- c) Si la disquette a fait un tour complet (IP=0)
- d) Les données séries sont lues (RD)

Les signaux que le contrôleur envoie au lecteur sont :

- a) Mise en route du moteur du lecteur (MOTOR ON = 1)
- b) Déplacement de la tête de lecture (STEP = 1)
- c) La direction de déplacement (DIRC = 0 ou 1)
- d) Activation de la logique d'écriture (WG = 1)
- e) Les données série doivent être écrites (WD)

Le contrôle du déroulement

Lorsqu'une commande du FDC est exécutée, les différentes fonctions qui la représentent doivent bien entendu être activées et désactivées selon un ordre déterminé.

De plus, des données sont transférées entre les différents registres du FDC, des calculs sont effectués, des broches sont testées, et l'état des broches de sortie est modifié.

Le temps de déroulement qui dépend de la commande est défini par le contrôle du déroulement.

Opérations de lecture

Les opérations de lecture n'ont lieu qu'avec la commande READ SECTOR. Les autres instructions qui lisent des données sur la disquette n'ont qu'un rôle de diagnostic et n'ont pas d'utilité pour les applications normales.

Les secteurs peuvent avoir une longueur de 128, 256, 512 ou 1024 octets. La longueur des secteurs est définie par le champ de longueur (le quatrième octet du champ d'ID) lors du formatage.

Si on doit lire un secteur, le contrôleur reconnaît le nombre d'octets de données qui doit être lu à partir de la data address mark grâce au champ de longueur. Pour que les octets de données soient lus sans erreur, il faut que l'ADDRESS MARK DETECTOR soit désactivé durant ce temps sinon, il peut produire des erreurs de lecture.

Opérations d'écriture

Avant de pouvoir écrire sur la disquette avec une instruction d'écriture, il faut activer la sortie WRITE GATE du FDC. Par mesure de sécurité contre les écritures non prévues, cela ne se fait qu'après que le registre de données soit écrit dès que la sortie DRQ a été activée. Si cela ne se passe pas ainsi, l'instruction est bloquée, INTRQD est activé, le bit d'état LOST DATA est positionné et le bit d'état BUSY est effacé.

S'il y a une réaction à la première demande de données, l'instruction d'écriture est exécutée. Si le FDC ne reçoit aucun autre octet de données après une demande de données, l'instruction n'est pas stoppée mais un octet nul est inscrit. Dans ce cas également, le bit LOST DATA est activé après l'instruction. Si on n'a par exemple écrit que 112 octets avec une instruction WRITE SECTOR, le contrôleur remplit les 400 octets restants (avec un secteur de 512 octets) avec des nuls. Il faut éviter d'utiliser ce processus pour effacer partiellement un secteur. Etant donné que le bit LOST DATA est positionné, on ne sait pas s'il y a eu une erreur lors de l'écriture du premier octet.

En général, l'écriture est interdite lorsque l'entrée WRITE PROTECT est à l'état bas. Dans ce cas, toute commande d'écriture est interrompue, INTRQ est mis à l'état HIGH, le bit d'état WRITE PROTECT est positionné et le bit d'état BUSY est effacé.

Pour augmenter la sécurité des données lorsque la densité est supérieure, comme sur les pistes intérieures, il est possible d'activer la précompensation en écriture. Si le bit de précompensation en écriture est effacé lors de la commande d'écriture, le flux de données sur WRITE DATA est envoyé 125 ns plus tôt ou plus tard, indépendamment des bits à envoyer. Le tableau qui suit montre quel cas se produit quand :

x	1	1	0	Plus tôt
x	0	1	1	Plus tard
0	0	0	1	Plus tôt
1	0	0	0	Plus tard

				Prochain bit à transférer
				Bit qui vient d'être envoyé
				Bits envoyés précédemment

La précompensation en écriture est activée normalement sur les pistes intérieures qui présentent une densité plus forte sur les disquettes 5 pouces 1/4. Sur les disquettes 3 pouces 1/2 dont la densité des pistes extérieures est la même que celle des pistes du milieu d'une disquette 5 pouces 1/4, la précompensation est toujours activée.

4.2.1.3 Description des commandes

Maintenant que nous avons expliqué la structure interne du FDC et quelques processus de base dans les opérations de lecture et d'écriture, venons-en aux commandes.

Le WD 1772 connaît 11 instructions différentes (commandes) qui peuvent être divisées en 4 groupes ou types.

Le tableau qui suit montre ces commandes :

multiple sector

φ = 1
1 = 77

Type	Commande	7	6	5	4	3	2	1	0
I	Restore	0	0	0	0	h	V	rl	r0
I	Seek	0	0	0	1	h	V	rl	r0
I	Step	0	0	1	u	h	V	rl	r0
I	Step-in	0	1	0	u	h	V	rl	r0
I	Step-out	0	1	1	u	h	V	rl	r0
II	Read sector	1	0	0	m	h	E	0	0
II	Write sector	1	0	1	m	h	E	P	a0
III	Read address	1	1	0	0	h	E	0	0
III	Read Track	1	1	1	0	h	E	0	0
III	Write Track	1	1	1	1	h	E	P	0
IV	Force Inter.	1	1	0	1	I3	I2	I1	I0

Motor on FLAG
V = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, 12, 13, 14, 15

Ces commandes ont plusieurs bits de label qui ont les significations suivantes :

h = Motor on flag

h = 0, activation du motor on - Test

h = 1, désactivation du motor on- Test

Lorsque le moteur du lecteur est mis en route, il faut attendre un certain temps avant qu'il ait atteint une vitesse optimale. Cela est géré par le WD 1772 de telle manière qu'il attend 6 impulsions d'index après la mise en route du moteur. Avec une vitesse optimale de 300 TPM, ce temps d'attente est de une seconde au moins. Ce processus (nommé spin up sequence) assure que les moteurs ont bien atteint la vitesse de croisière avant de commencer les opérations de lecture/écriture.

A la fin d'une commande, les moteurs du lecteur sont arrêtés après 10 rotations de la disquette (environ 2 s.). Si une autre commande survient durant la phase de fin de rotation, il serait bien sûr superflu de reprendre la phase de spin up. C'est pourquoi le contrôleur inclu un test motor-on.

Lorsque ce test est activé ($h=0$), la sortie motor-on est d'abord testée. Si motor-on est à l'état LOW, le FDC exécute la séquence de spin-up. Si motor-on est à l'état HIGH, le contrôleur considère que les moteurs ont atteint la vitesse de croisière et il exécute l'instruction.

Lorsque le FDC reçoit une commande (avec le bit h positionné), il met d'abord les moteurs en route en mettant la sortie motor-on à l'état HIGH. Cela est indépendant de l'état de la sortie MOTOR-ON. Ensuite, il commence l'exécution de la commande. Il n'attend donc pas que 6 impulsions d'index soient atteintes.

V = Verify Flag

V = 0, activer Verify

V = 1, désactiver Verify

Ce flag-bit existe seulement dans le groupe des commandes de type I. S'il est positionné, le contrôleur exécute une vérification de piste après une commande STEP ou après le dernier STEP, dans une instruction RESTORE ou SEEK. Cela se fait comme suit : un champ d'ID correct dont le numéro de piste correspond au contenu du registre de piste est recherché après le STEP.

Si une instruction READ ou WRITE-SECTOR se déroule alors que la tête de lecture/écriture n'est pas sur la piste désirée, on ne lit ou écrit dans aucun cas un mauvais secteur car un verify est exécuté avec cette instruction. Dans ce cas, l'instruction STEP est exécutée avec un Verify.

Un Verify est absolument sans utilité lorsqu'on formate une nouvelle disquette par exemple. Il serait donc absurde d'exécuter une commande Verify après chaque instruction step. Comme le contrôleur recherche un champ d'ID correct en un temps de 5 rotations environ, on perd environ 1 minute 1/2 par formatage de 80 pistes.

Si on formate une seule piste d'une disquette portant déjà des données, il est préférable d'exécuter le verify. En effet, la commande WRITE-TRACK qui sert au formatage d'une piste ne fait aucun test avant son exécution. On formate donc la piste sur laquelle se trouve la tête de lecture écriture.

R1,R0 = Stepping Rate

0	0	2ms
0	1	3ms
1	0	5ms
1	1	6ms

Ces deux bits servent à programmer le stepping rate. Il s'agit du temps d'attente entre deux impulsions step d'une commande SEEK ou RESTORE. On a ainsi la possibilité d'adapter dans certaines limites le FDC aux possibilités mécaniques du lecteur.

Prenons pour exemple un lecteur dont la tête nécessite un temps de déplacement de 6 ms. Lorsqu'on programme le stepping rate à 3ms et qu'on déplace la tête de la piste 0 à la piste 40 avec une commande SEEK (Ce qui correspond à 39 impulsions), la tête n'atteindrait que la piste 20 car en raison des contraintes mécaniques, une impulsion sur deux est ignorée.

Avec des impulsions uniques (step, step-in, step-out), la tête sera certainement correctement contrôlée car la durée des impulsions step ne sera pas influencée par le stepping rate. Avec le timing interne du FDC, l'unité est de 4 us.

u = Update Flag

u = 0, pas de sauvegarde du registre de piste

u = 1, sauvegarde du registre de piste.

Si le bit-u est positionné durant une commande STEP, STEP-IN ou STEP-OUT, le registre de piste est incrémenté de un ou décrémenté de un après chaque opération. Cela ne signifie pas que le contenu du registre de piste corresponde à la piste courante. Il faut préciser deux choses :

- 1) Le numéro de piste doit correspondre avant le step au contenu du registre de piste.
- 2) Il ne faut pas essayer de contrôler un numéro de piste supérieur à 82. Si la tête de lecture/écriture se trouve par exemple sur la piste 82 (la dernière piste qui peut être atteinte par le lecteur) et si le registre de piste

contient 82, le registre de piste contiendra la valeur 87 après 5 nouvelles instructions STEP-IN alors que la tête se trouve toujours sur la piste 82.

m = Multiple sector

m = 0, écriture ou lecture de secteur

m = 1, écriture ou lecture de plusieurs secteurs

Avec ce bit, on a la possibilité de lire ou écrire au maximum tous les secteurs d'une piste avec une seule instruction READ- ou WRITE-SECTOR. Mais il faut que les numéros de secteurs forment une série continue. Le numéro du premier secteur à lire ou à écrire est d'abord inscrit dans le registre de secteur. Dès que le FDC a écrit ou lu ce secteur, il incrémente le registre de secteurs et tente de lire ou d'écrire le secteur suivant. Cela se poursuit tant que l'on trouve un autre secteur ou que la commande n'est pas interrompue par une instruction FORCE INTERRUPT.

a0 = Data Address Mark

a0 = 0, écriture d'une data mark normale (\$FB)

a0 = 1, écriture d'une data mark effacée (\$F8)

La data address Mark indique le début du champ de données. Etant donné qu'on peut écrire plusieurs data address mark avec le bit a0 (suivant qu'il est positionné ou effacé), on est en mesure de reconnaître simplement un secteur. Après une commande READ SECTOR, le type de la data address mark est inscrit dans le bit d'état RECORD TYPE.

E = 30 ms setting delay

E = 0, aucun temps d'attente pour la tête

E = 1, 30 ms de temps d'attente pour la tête.

Il existe certains types de lecteurs de disquette pour lesquels la tête de lecture n'est pas continuellement posée sur le support magnétique. Dans ces lecteurs, un processus magnétique soulève ou repose la tête de lecture/écriture. Ce processus nommé Head Load réduit l'usure de la disquette car on ne pose la tête que pour les opérations de lecture ou d'écriture.

Mais des contraintes mécaniques font que durant un certain temps, la tête n'est pas positionnée convenablement sur la disquette et donc que le contact optimal entre la tête et la disquette n'est pas possible durant un temps donné. Si on positionne le bit E, un délai est prévu pour couvrir ce temps.

P = Write Precompensation

p = 0, activation de la précompensation en écriture

p = 1, désactivation de la précompensation en écriture.

10-13 = Conditions Interrupt

I0 = 1, pas de signification

I1 = 1, pas de signification

I2 = 1, interruption à chaque impulsion d'index

I3 = 1, interruption immédiate

10-13 = 0, exécution de la commande en cours sans interruption

LES COMMANDES DE TYPE 1

Les commandes de type 2 et de type 3 qui servent à la lecture et à l'écriture de données concernent toujours la piste sur laquelle se trouve momentanément la tête de lecture/écriture. C'est le groupe des commandes RESTORE, SEEK, STEP, STEP-IN, STEP-OUT qui sert à son positionnement.

RESTORE (recherche de la piste 0)

Mot de commande :	7	6	5	4	3	2	1	0
	0	0	0	0	h	V	r1	r0

Lorsque le FDC reçoit cette commande, il teste tout d'abord l'entrée TR00. Si elle est à l'état LOW car la tête se trouve sur la piste zéro, le registre de piste est mis à zéro.

Si la tête de lecture/écriture n'est pas sur la piste zéro, les impulsions STEP-OUT sont envoyées jusqu'à ce que l'entrée TR00 soit à l'état bas. Si ce n'est pas encore le cas après 255 impulsions STEP, la commande est interrompue. L'interruption de la commande se voit à la sortie INTRQ et l'effacement de bit d'état BUSY.

SEEK (Recherche de piste)

Commande :	7	6	5	4	3	2	1	0
	0	0	0	1	h	V	r1	r0

Cette instruction sert à positionner directement la tête de lecture/écriture sur une piste déterminée. Le numéro de la piste désirée est mis dans le registre de données. Pour que cette commande fonctionne sans problèmes, il faut que le numéro de la piste courante se trouve dans le registre de piste. Si on utilise plusieurs lecteurs, il faut éventuellement mettre le numéro de la piste courante dans le registre de piste.

Si le FDC rencontre une commande SEEK, il compare le registre de piste avec le registre de données et détermine si les impulsions STEP-IN ou STEP-OUT sont possibles. Ensuite, il envoie les impulsions STEP pour la direction désirée.

Le registre de piste est remis à jour à chaque impulsion STEP. Dès que les contenus du registre de piste et du registre de données sont égaux, la piste-objet est atteinte et la commande est interrompue. Cela est visible au fait que la sortie INTRQ est activée et que le bit d'état BUSY est effacé.

STEP

Commande :	7	6	5	4	3	2	1	0
	0	0	1	u	h	V	r1	r0

Cette instruction indique au FDC qu'il faut envoyer une instruction STEP. La direction est la même que celle de la dernière instruction STEP car l'état de la sortie DIRECTION n'est pas modifié. La sortie INTRQ est mise à l'état HIGH et le bit d'état BUSY est effacé.

STEP IN

Commande :	7	6	5	4	3	2	1	0
	0	1	0	u	h	V	r1	r0

Dans une commande STEP IN , la sortie DIRECTION est mise à l'état HIGH, quelque soit l'état précédent et une instruction STEP est envoyée. Elle sert à déplacer la tête vers le centre de la disquette. La sortie INTRQ est mise à l'état HIGH et le bit d'état BUSY est effacé.

STEP OUT

Commande :	7	6	5	4	3	2	1	0
	<hr/>							
	0	1	1	u	h	V	r1	r0

Avec cette instruction, la sortie DIRECTION est mise à l'état LOW, quelque soit son état précédent et une impulsion STEP est envoyée. Ainsi, la tête de lecture/écriture est déplacée d'un pas vers l'extérieur de la disquette. La sortie INTRQ est mise à l'état HIGH et le bit d'état BUSY est effacé.

La séquence VERIFY dans les commandes de type I

Si on a positionné le bit V dans la commande, la commande suivante est exécutée après un délai de 30 ms lorsqu'on a atteint la piste désirée :

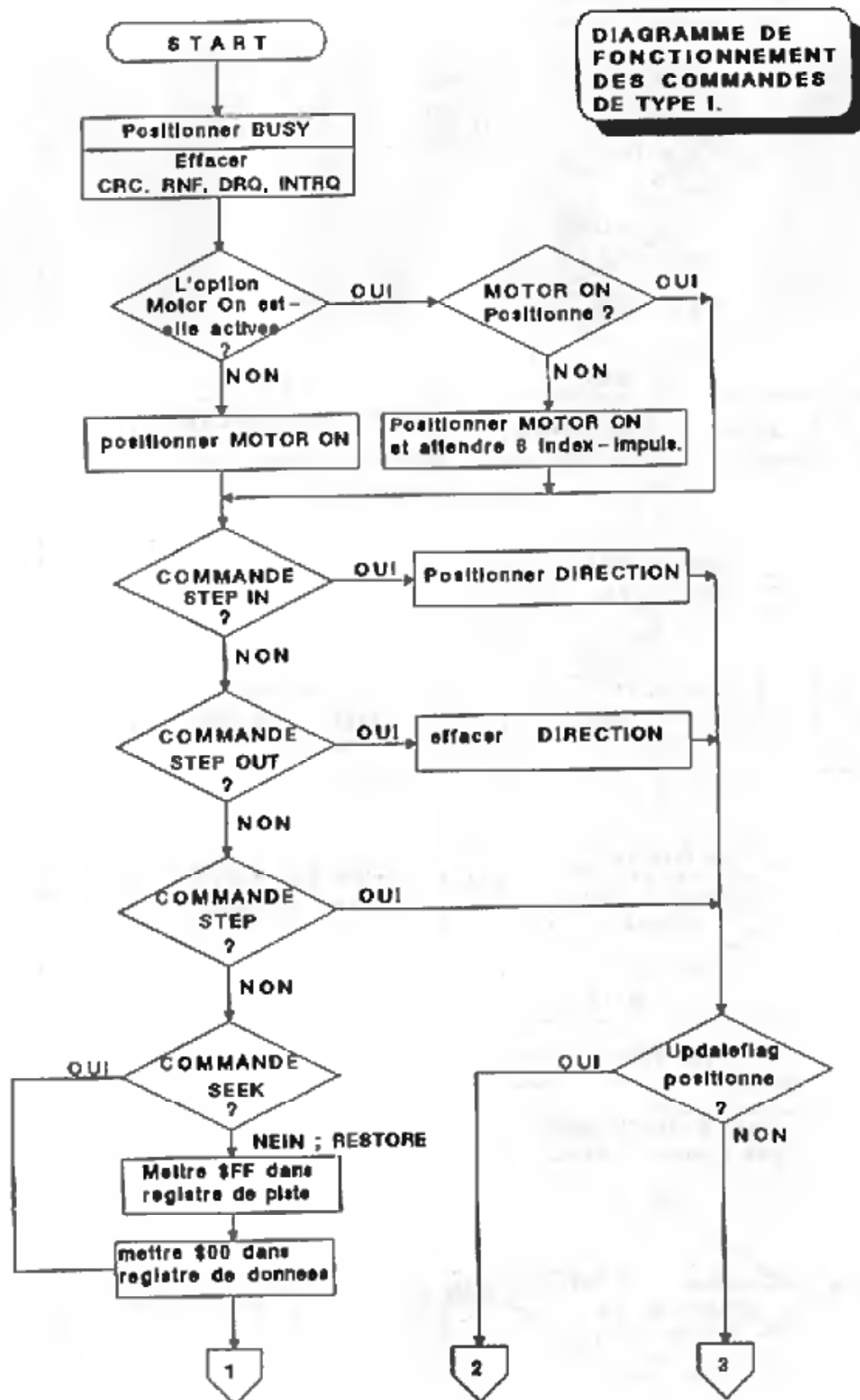
Le numéro de la piste du premier champ d'ID lu est comparé avec le contenu du registre de piste. S'ils correspondent, l'octet de CRC du champ d'ID est testé. S'il est égal à l'octet calculé avec la logique CRC, la séquence VERIFY s'est déroulée sans erreur. La sortie INTRQ est mise à l'état HIGH et le bit d'état BUSY est effacé.

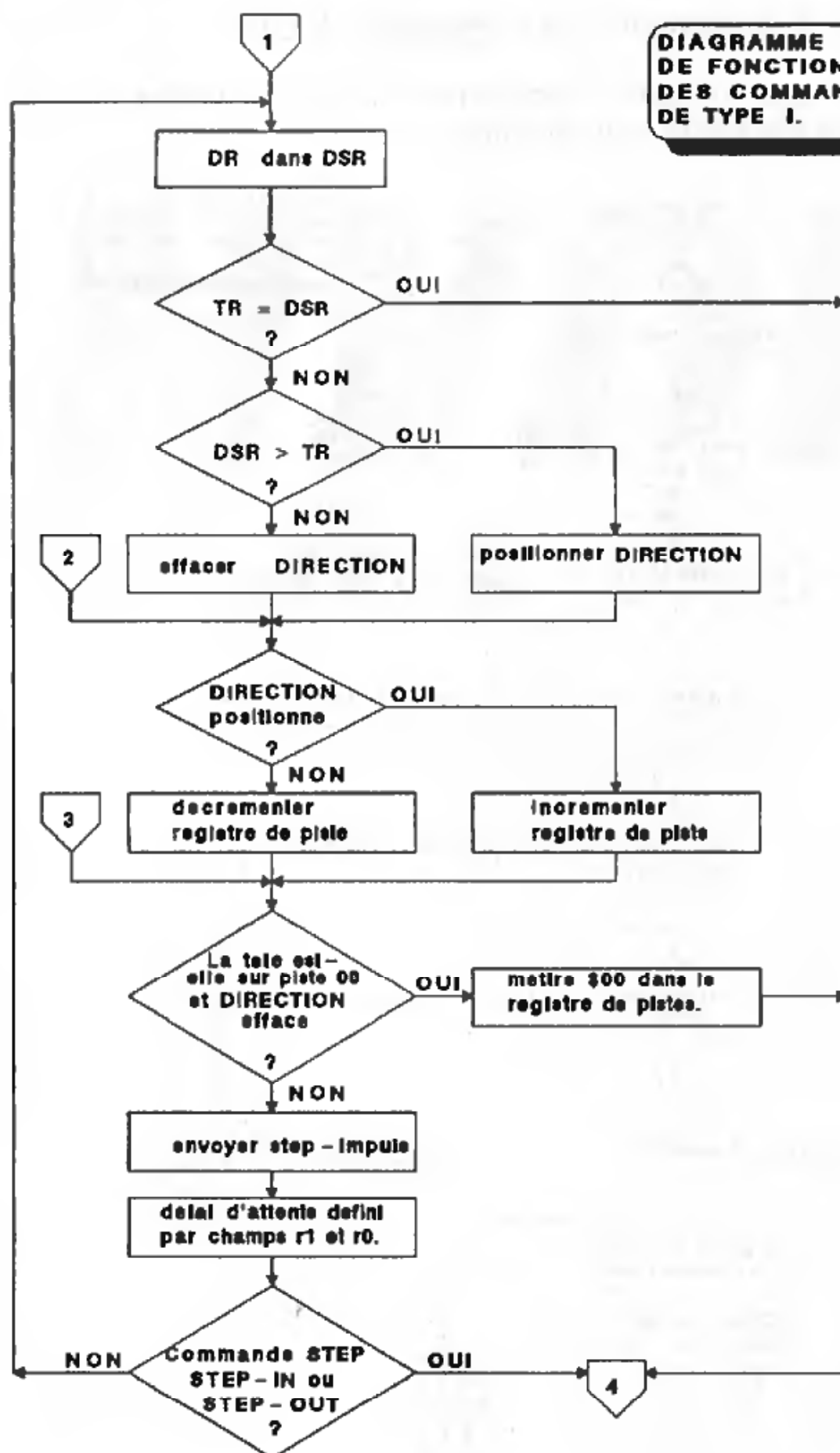
Si dans le champ d'ID le numéro de piste est faux ou si la somme CRC n'est pas correcte, le champ suivant est lu et le test est repris.

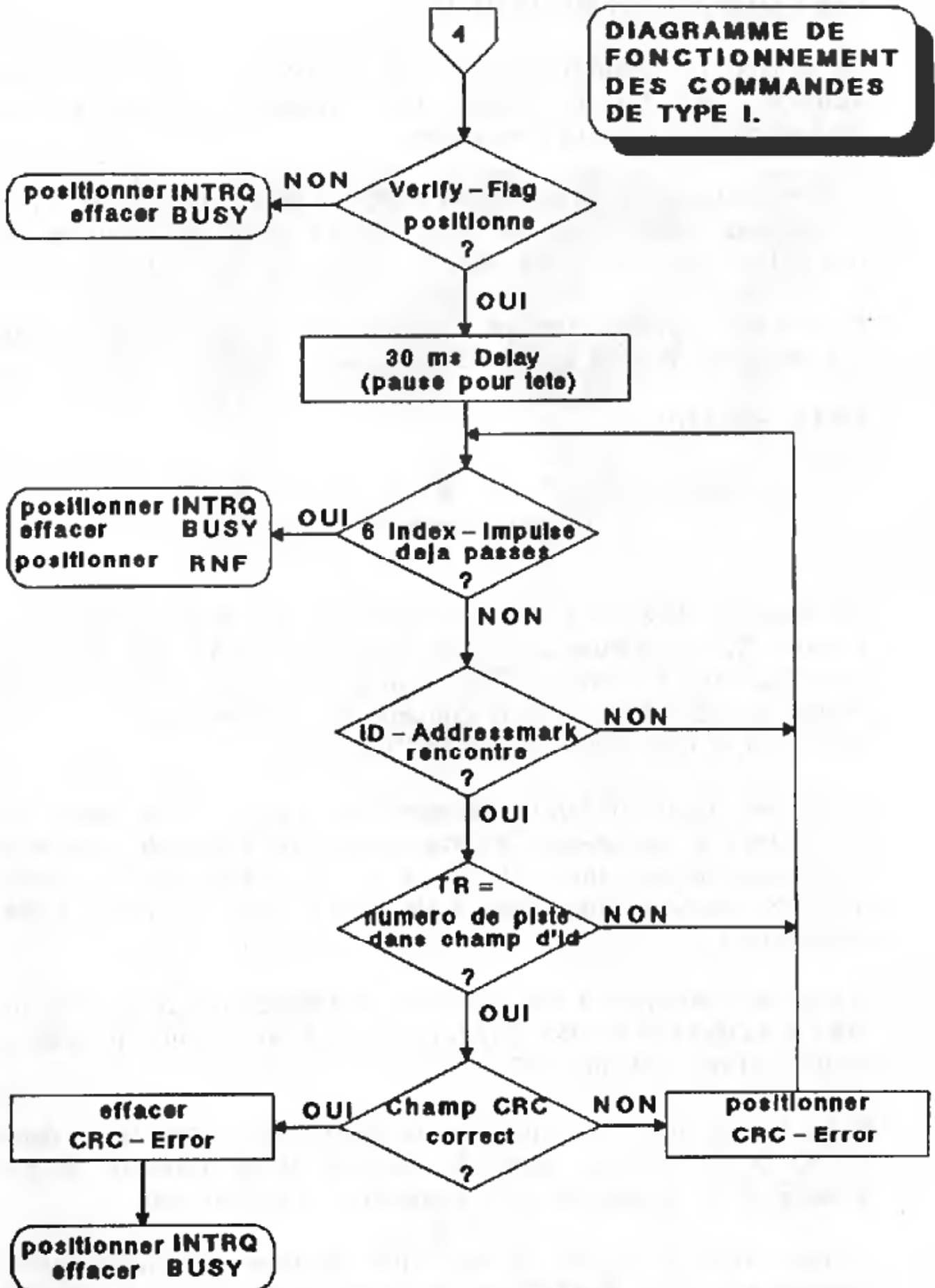
Si on n'a pas trouvé de champ d'ID avec un numéro de piste et un checksum correct après 5 rotations de la disquette, le bit RECORD NOT FOUND du registre d'état est positionné. La commande est interrompue, ce qui est visible à l'état de la sortie INTRQ et au fait que le bit d'état BUSY est effacé.

Diagramme de fonctionnement des commandes de type I

Pour rendre plus explicite le fonctionnement des commandes de type I, nous vous présentons le diagramme suivant :



**DIAGRAMME
DE FONCTION
DES COMMANDES
DE TYPE I.**


**DIAGRAMME DE
FONCTIONNEMENT
DES COMMANDES
DE TYPE I.**


LES COMMANDES DE TYPE II

Ce groupe de commande sert à la lecture et à l'écriture des secteurs (c'est l'unité logique de données). Le transfert de données est fait avec ces commandes.

Comme toutes les commandes qui servent à la lecture et à l'écriture de données, elles prennent effet sur la piste sur laquelle se trouve la tête de lecture/écriture.

C'est le rôle du programmeur de positionner auparavant la tête sur la bonne piste avec les commandes de type I.

READ SECTOR

Commande :	7	6	5	4	3	2	1	0
	<hr/>							
	1	0	0	m	h	E	0	0

On donne au FDC le numéro du secteur à lire, dans le registre de secteurs. Si on exécute alors une commande READ SECTOR, le contrôleur lit un champ d'ID et teste si son numéro de piste et de secteur correspondent avec le contenu des registres de piste et de secteur. Si ce n'est pas le cas, on lit l'ID suivant.

Si les deux correspondaient, on mémorise la longueur du secteur et on compare la somme-test CRC du champ d'ID avec celle calculée à partir des données lues. S'il n'y a aucune différence, le champ d'ID correspondant au secteur a été trouvé, sinon, on lit un autre champ d'ID.

Avant de commencer à lire le champ de données, il faut trouver un DATA ADDRESS MARK (DAM) dans les 43 octets suivants. Sinon, on doit relire un champ d'ID.

Si on n'a pas trouvé de champ d'ID correct, suivi d'un DAM dans les 43 octets suivants, après 5 rotations de la disquette, le bit d'état RNF est positionné et la commande est interrompue.

Comme vous le voyez, il faut que quelques conditions soient remplies avant que le FDC lise un champ de données qui contienne un secteur.

Si tout s'est bien déroulé jusque là, voici la suite des événements :

Le type du DATA ADDRESS MARK (normal = 0, effacé = 1) est mis dans le bit d'état numéro 5.

Le détecteur d'address mark est désactivé et le nombre d'octets correspondant est lu (ce nombre est calculé à partir de la valeur dans le champ de longueur de secteur).

Pour chaque octet, un DRQ est exécuté. S'il n'y a aucune réaction à ce DRQ, le FDC positionne le bit d'état LOST DATA.

Dès que tous les octets sont lus, une CRC est testée. Elle se trouve dans les deux octets suivant le secteur et elle est comparée avec la CRC calculée à partir des octets lus. Si elles diffèrent, le bit d'état CRC ERROR est positionné.

READ SECTOR (avec bit m positionné)

Si on a positionné le Bit m dans la commande READ SECTOR, le FDC essaie de lire plusieurs secteurs (la piste entière au maximum). Le nombre des premiers secteurs à lire est mis dans le registre des secteurs du contrôleur. Le déroulement de la commande est le même que ce qu'on vient de décrire. Une fois que le secteur a été lu, le contrôleur incrémente automatiquement le registre de secteur et exécute un nouveau READ SECTOR.

Cela est fait tant qu'on peut trouver un autre secteur. En d'autres mots, cette commande est interrompue avec une RECORD NOT FOUND ERROR.

Etant donné qu'on ne peut pas indiquer au contrôleur le nombre de secteurs à lire, il faut avoir une autre possibilité pour lire plusieurs secteurs. On utilise pour cela la commande FORCE INTERRUPT. On n'attend donc pas que le FDC interrompe la commande de lui même, mais on l'interrompt à un moment donné.

L'exemple qui suit va vous expliquer comment :

Supposons que la tête de lecture se trouve sur une piste quelconque au format ATARI.

Les secteurs de cette piste sont numérotés de 1 à 9 et nous voulons lire les secteurs 3-7, c'est-à-dire 5 secteurs.

Le secteur numéro 3 est mis dans le registre de secteurs et l'opération de lecture est activée avec la commande.

Si on ne fait rien d'autre, le FDC va lire 7 secteurs (secteurs 3 à 9) et interrompre la commande avec une RNF ERROR après 5 rotations (car le secteur 0 ne peut pas être trouvé).

Mais on veut lire seulement les secteurs 3-7 (et sans message d'erreur). C'est pourquoi on suit le processus suivant :

Etant donné que le transfert de données est effectué par le contrôleur DMA, il est initialisé avant l'exécution de la commande FDC par une adresse de début de DMA. Pendant que le FDC envoie les données vers le contrôleur DMA, cette adresse est incrémentée. On peut donc connaître l'adresse DMA courante en lisant le registre d'adresses DMA.

On teste donc ce registre jusqu'à ce que son contenu soit de \$A00 (5 * \$200) supérieur à l'adresse de départ. Dès que c'est le cas, la commande READ SECTOR est interrompue avec un FORCE INTERRUPT.

WRITE SECTOR

Commande : 7 6 5 4 3 2 1 0

1 0 1 m h E P a0

Nous ne décrivons dans cette partie que les différences avec la commande READ SECTOR. En effet, la plus grande partie du processus est la même.

Au début de la commande, le FDC teste si l'entrée WRITE PROTECT est à l'état LOW. Si c'est le cas (disquette protégée), le bit d'état WPRT est positionné et la commande est interrompue.

Si la disquette n'est pas protégée, le FDC recherche le champ d'ID.

Dès qu'un champ d'ID correct a été trouvé, un délai correspondant à 23 octets est accordé puis 12 octets nuls et une DATA ADDRESS MARK sont écrites (cela est indépendant du bit a0). Puis il y a une information sur le secteur qui est donnée par la CRC. Enfin, la commande écrit un octet \$FF.

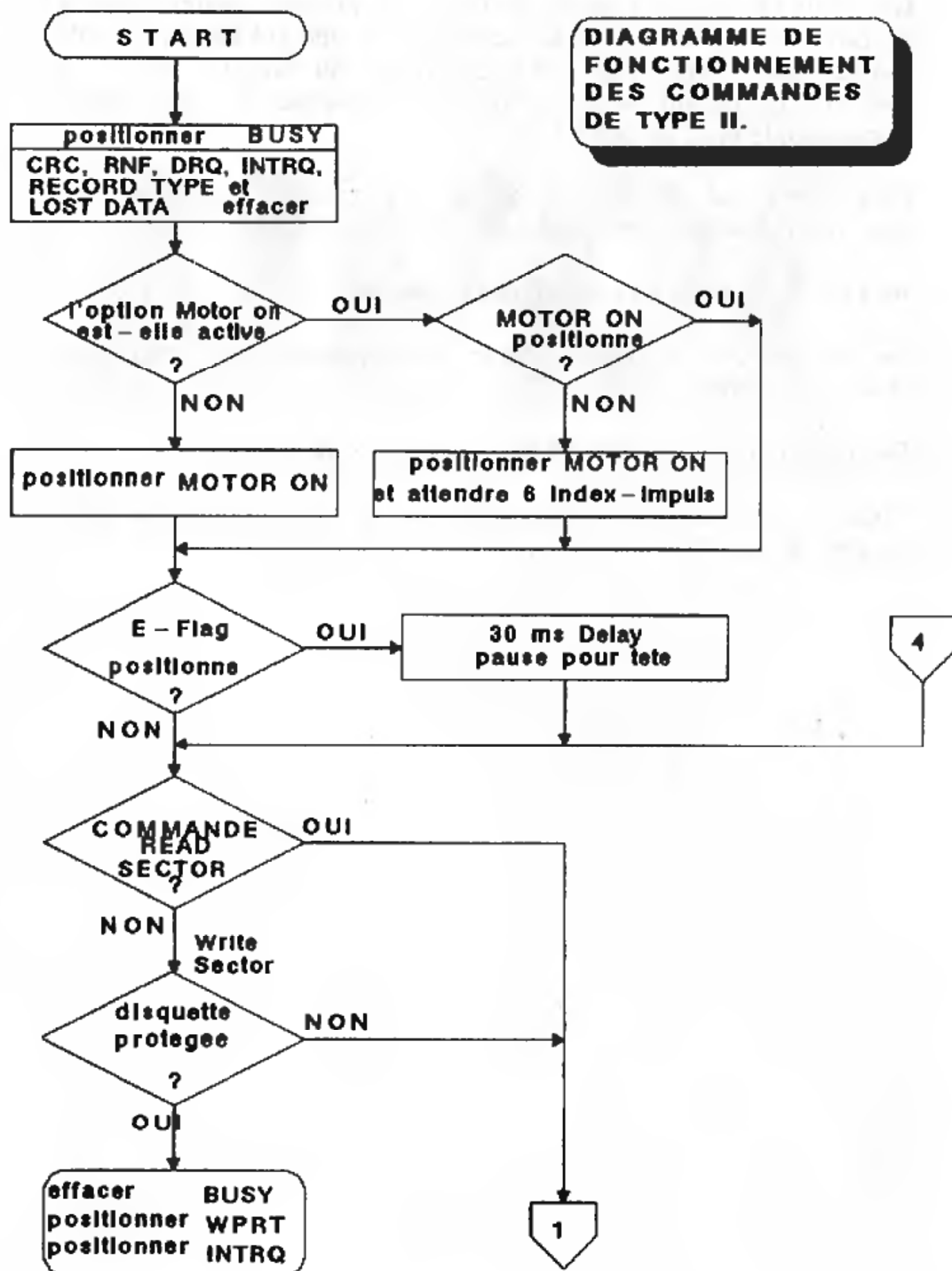
Pour savoir si le secteur a été correctement écrit, il faut absolument exécuter une commande READ SECTOR.

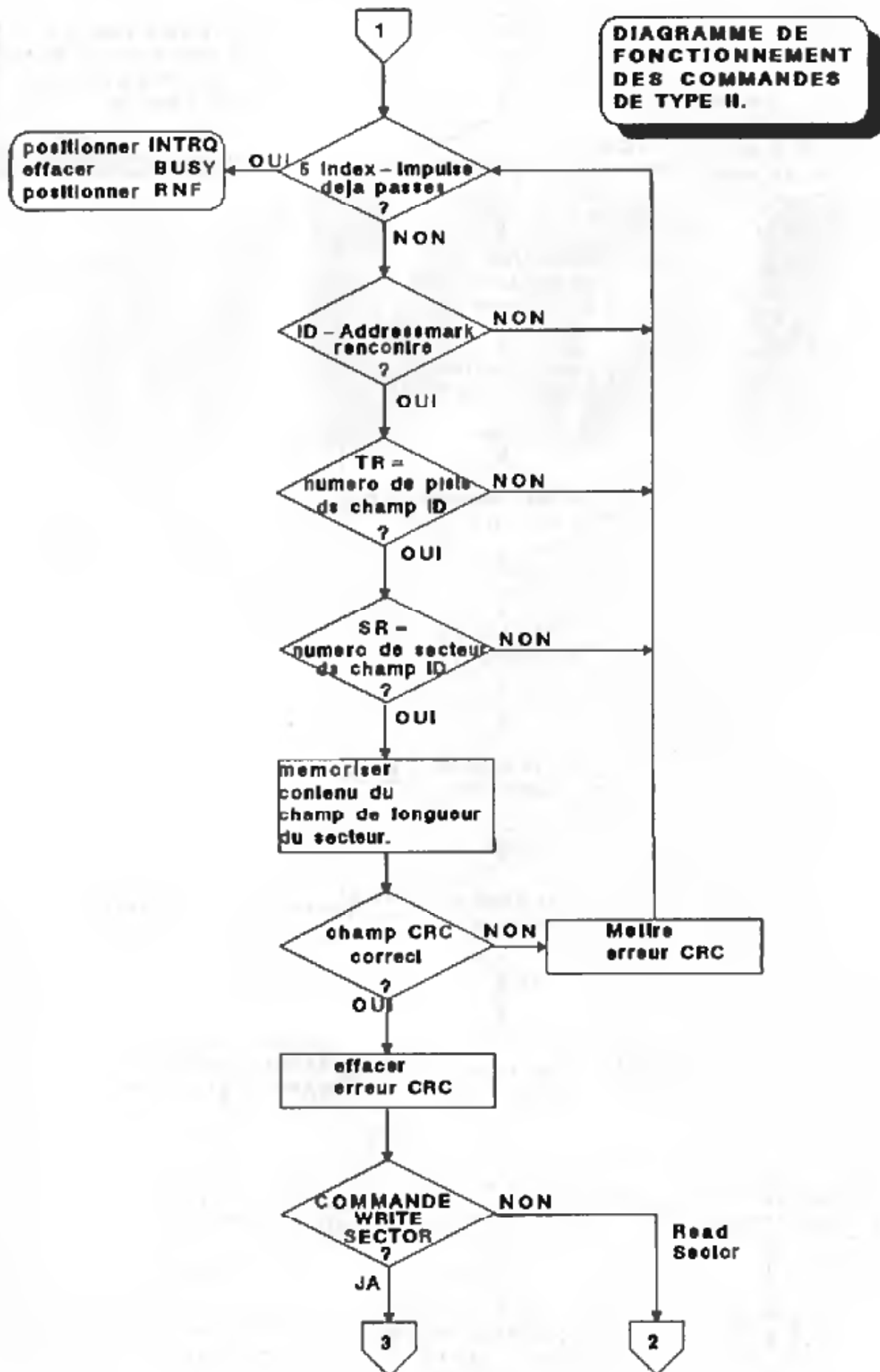
WRITE SECTOR (avec le bit m positionné)

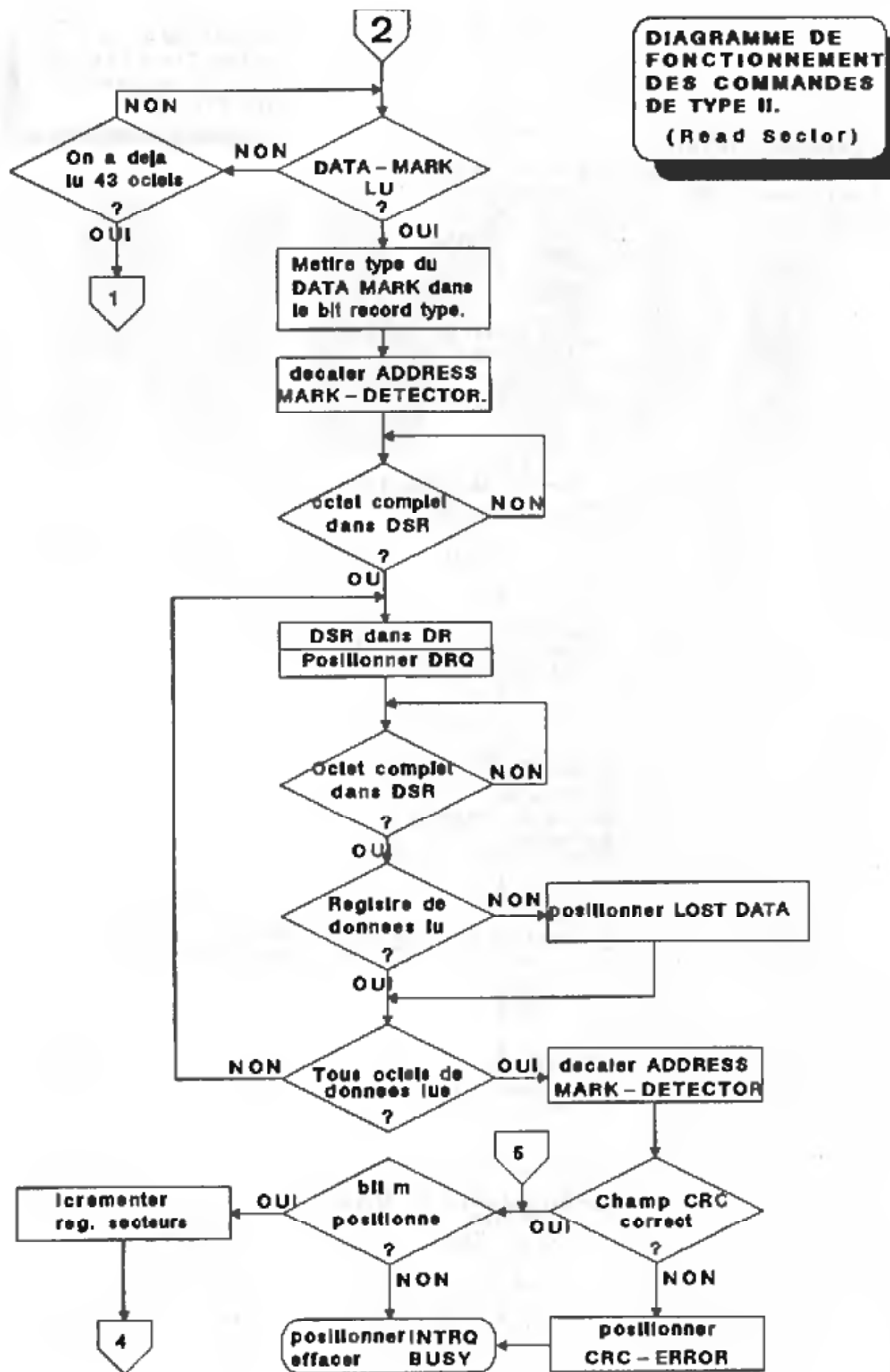
Le processus est le même qu'avec la commande READ SECTOR avec le bit m positionné.

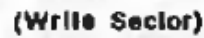
Diagramme de fonctionnement des commandes de type II

Voici un diagramme de fonctionnement qui correspond aux commandes de type II.









LES COMMANDES DE TYPE III

La commande **WRITE TRACK** de ce groupe d'instruction sert au formatage d'une piste alors que **READ TRACK** et **READ ADDRESS** peuvent servir à analyser le format d'une piste.

READ ADDRESS

Commande :	7	6	5	4	3	2	1	0
	<hr/>							
	1	1	0	0	h	E	0	0

Avant cette commande, il est nécessaire de dire quelques mots concernant le contrôleur DMA. Sinon, vous risquez de frôler la folie durant la programmation de l'instruction **READ ADDRESS**.

Dès que la commande **READ ADDRESS** est exécutée, on se rend compte avec étonnement que le champ d'ID n'est pas en RAM.

Mais si on teste le registre d'état (DMA et DFC), il n'y a aucune erreur. Si l'on soustrait l'adresse de départ du DMA de l'adresse de fin, on obtient zéro. Donc Zéro octets sont transférés. C'est bien sûr un peu faible !

Où sont donc passés les 6 octets de champ d'ID ? Mais c'est très simple : dans le contrôleur DMA ! En effet ce contrôleur ne transfère pas les octets un à un, mais il attend d'en avoir 16. Il envoie alors un signal bus vers le 68000 pour pouvoir mettre ces 16 octets dans la RAM.

Comment avoir accès au champ d'ID qui se trouve dans le contrôleur DMA ? Il n'y a qu'une possibilité. Il faut lire d'autres données. Cela se fait en lisant plusieurs champs d'ID les uns derrière les autres.

Après 3 instructions **READ ADDRESS** par exemple (18 octets) on a 16 octets en RAM et 2 octets dans le contrôleur DMA. Afin de mettre tous les octets lus dans la RAM, il faut que leur nombre soit un multiple entier de 16.

Mais il y a un autre truc. Il consiste en l'effacement du registre d'état du DMA, ce qui est obtenu en faisant un 'toggle' (c'est à dire en activant et désactivant) sur la broche de lecture/écriture. Si vous réactivez ce registre avant chaque transfert DMA par mesure de sécurité, vous risquez d'être surpris. En effet, non seulement le registre d'état est effacé, mais également tous les octets qui se trouvaient dans le contrôleur DMA. Dans ce cas, on peut lire autant de champ DMA qu'il y en a sur la disquette. Le contrôleur DMA ne mettra pas un seul octet en mémoire. Il faut donc effacer le registre d'état du DMA seulement avant la première commande READ ADDRESS.

Venons-en maintenant à la commande READ ADDRESS proprement dite.

Cette commande sert à lire le champ d'ID suivant. Elle peut être utilisée en liaison avec la commande READ TRACK pour analyser le format d'une piste. De plus, il est possible d'exécuter une vérification de piste sans quitter la piste.

La commande READ ADDRESS lit un champ d'ID sans tester s'il existe un champ de données correspondant. Elle lit 6 octets qui ont les significations suivantes :

<u>Octet n°</u>	<u>Signification</u>
-----------------	----------------------

1	numéro de piste
2	numéro de face
3	numéro de secteur
4	taille du secteur
5	octet CRC 1
6	octet CRC 2

Le numéro de piste (octet 1) est inscrit dans le registre de secteur. On peut donc effectuer un verify sur la piste sans utiliser les données lues.

Cela peut paraître inutile à première vue car peu importe qu'on compare le contenu du registre de secteurs ou le premier octet transféré avec le contenu du registre de piste pour effectuer un verify de piste. De plus, il est plus simple d'utiliser l'octet transféré car dans ce cas, il n'est pas besoin de sélectionner auparavant le registre de secteur.

Mais après une instruction READ ADDRESS, il n'y pas encore d'informations en RAM car, comme nous l'avons indiqué plus haut, le transfert n'est effectuée qu'à partir du moment où le contrôleur a reçu 16 octets. Mais étant donné que le FDC inscrit le premier octet du champ d'ID également dans le registre de secteur, on a la possibilité de faire un verify sur une piste avec une seule commande READ ADDRESS.

Lors de la lecture du champ d'ID, un checksum est construit pour être comparé avec les octets de CRC 1 et 2. S'ils ne sont pas identiques, le bit d'état CRC ERROR est positionné.

Si on n'a pas trouvé de champ d'ID après 6 impulsions d'index (cela correspond à 5 rotations minimum), le bit d'état RNF est positionné (Record not found).

Dès que le contrôleur a traité la commande, il l'indique en activant la sortie INTRQ et en effaçant le bit d'état BUSY.

READ TRACK

Commande :	7	6	5	4	3	2	1	0
	1	1	1	0	h	E	0	0

La commande READ TRACK sert uniquement à vérifier la piste. Elle lit une piste complète, avec les octets de GAP, de SYNC et de données.

La lecture commence avec l'impulsion d'index suivante que le contrôleur reçoit du lecteur. Il lit les données tant qu'il n'a pas reçu d'autre impulsion d'index. La fin de l'opération est indiquée comme d'habitude par l'activation de la sortie INTRQ et par l'effacement du bit d'état BUSY.

Pour chaque bit lu, un DRQ est créé. Comme dans toutes les commandes de transfert de données, le bit d'état LOST DATA est positionné s'il n'y a aucune réaction au DRQ.

C'est d'après notre expérience la raison la plus fréquente pour laquelle le bit LOST DATA puisse être positionné.

Cela a pour effet que ce bit ne permet pas de conclure que des données ont été perdues lors d'une commande READ TRACK. De tels cas sont apparus avec des READ TRACK sur une disquette non formatée.

La lecture des bits de données est synchronisée avec chaque address mark. L'ADDRESS MARK DETECTOR qui sert pour cela n'est pas désactivé et reste activé durant toute la lecture (à la différence de la commande READ SECTOR). Il recherche continuellement un address mark et évite ainsi les erreurs de lecture.

D'après le constructeur, toutes les données, à l'exception des octets de GAP doivent être lues correctement. Mais nos essais ont conduit à d'autres constatations. En pratique, seul le champ d'ID est lu correctement. Mais même avec le test CRC du champ d'ID, il arrive parfois des erreurs.

On se pose donc la question à quoi peut servir la commande READ TRACK. En raison d'erreurs de lectures possibles, les données sont d'une valeur peu sûre et on obtient plus facilement des champs d'ID corrects avec la commande READ ADDRESS. En effet, la recherche d'un champ d'ID sur toute la piste pose un autre problème. Ainsi, avec une suite d'octets comme FE-01-00-01-02-BC-DB on ne sait pas s'il s'agit d'un champ d'ID. Cette suite d'octets peut très bien survenir dans un champ de données. Un vrai champ d'ID est seulement un champ reconnu comme tel par le contrôleur.

L'instruction READ TRACK n'est pas très intéressante, utilisée seule. Mais, utilisée conjointement avec la commande READ ADDRESS, elle permet d'analyser assez précisément une piste. Alors que les données en elles-mêmes n'ont pas une importance considérable, en revanche, leur nombre est significatif. On peut calculer très précisément les écarts entre les points "marquants", ce qui est important pour l'analyse des pistes. On évite ainsi l'inconvénient de la commande READ ADDRESS qui lit seulement un champ d'ID sans tester s'il existe un champ de données correspondant.

Comme exemple, nous allons prendre le déroulement d'une analyse de piste :

- a) Tous les champs d'ID de la piste sont lus avec la commande READ ADDRESS.
- b) L'ensemble des informations de la piste est lu avec la commande READ TRACK.
- c) Tous les secteurs de la piste (on retrouve les numéros de piste et de secteurs à partir du champ d'ID sont lus avec la commande READ SECTOR.

Si la piste ne contient aucun champ d'ID, notre analyse est terminée car nous avons affaire à un format illisible.

On recherche donc le premier champ d'ID de la piste. Il faut s'orienter vers les sus-nommés points "marquants".

Le premier de ces points est une suite d'octets de \$A1,\$FE ou \$C2,\$FE, c'est-à-dire octet SYNC et ID-ADDRESS-MARK. L'ID se trouve obligatoirement derrière un tel point.

Dès qu'il a été retrouvé, on recherche le second point "marquant". Il faut trouver un octet SYNC suivi d'une DATA-ADDRESS-MARK. S'il n'y en a pas, il n'existe aucun champ de données valable pour l'ID trouvé.

Il y a un autre test concernant la plausibilité du champ d'ID. Si ce champ montre une taille de secteur de 512 octets et que le champ d'ID suivant est à 200 octets de distance, quelque chose ne va pas.

La lecture des secteurs se fait ainsi pour deux raisons. Premièrement, les informations sur le secteur ne peuvent pas être récupérées à partir des données lues par la commande READ TRACK (à cause des erreurs de données dont nous avons parlé plus haut). D'autre part, seule la commande READ SECTOR permet un test de la CRC.

De cette manière ou d'une manière équivalente, on peut analyser tout le contenu de la piste. La conversion des informations obtenues offre la possibilité soit de reproduire la piste soit de reconnaître que la piste ne peut pas être reproduite avec les possibilités du WD 1772.

WRITE TRACK (FORMAT TRACK)

Commande :	7	6	5	4	3	2	1	0
	1	1	1	1	h	E	P	0

Avant d'utiliser une disquette pour sauvegarder des données, il faut la formater. Mais que se passe-t-il dans ce cas ? En principe, cela est très simple à expliquer.

L'unité logique dans le transfert de données entre le lecteur et le contrôleur est le secteur. Mais comme il n'y a aucune information concernant un secteur sur une nouvelle disquette, il faut le créer.

A chaque secteur correspond un champ qui contient les informations le concernant. Les données sont sécurisées avec des checksums. Cela aussi n'existe pas sur une nouvelle disquette et doit être écrit. Les octets de synchronisation ne s'y trouvent pas non plus. Or il sont d'une importance extrême pour reconnaître le début d'un octet qui sera quelque part dans le flux de bits lors d'une opération de lecture.

L'objectif du formatage est donc d'écrire toutes les informations et les marquages sur disquette. Mais cela se passe selon des règles précises sinon aucun transfert de secteur ne serait possible.

Si on se limite à ces règles, on peut créer de nombreux formats qui fonctionnent et qui s'écartent du standard.

On envoie un octet de donnée au FDC, c'est-à-dire un octet dont la valeur oscille entre \$00 et \$FF. Mais pour écrire le marquage nécessaire, qui se différencie d'un octet de données "normal", sur la disquette, il faut qu'il soit possible de contrôler le contrôleur afin d'écrire non un octet de données mais un marquage sur la disquette.

Nous allons donc traiter des octets de contrôle qui sont représentés par des octets de \$F5 à \$FF.

Contrairement aux commandes **READ SECTOR** et **WRITE SECTOR** pour lesquelles ils sont décrits comme des octets de données "normaux", la commande **WRITE TRACK** indique au

contrôleur qu'il faut inscrire des marquages spéciaux. Cela signifie qu'ils sont écrits sans impulsion de fréquence et donc qu'ils pourront être distingués des octets de données qui possèdent une impulsion de fréquence (voir Address Mark Detector). Voici un schéma qui vous montre l'effet de ces octets de contrôle avec une instruction WRITE TRACK et leur signification lors d'une opération de lecture ultérieure.

Octets envoyés au FDC	Octets reçus du FDC	Description
\$F5	\$A1	octets SYNC, effacer registre CRC
\$F6	\$C2	octet SYNC
\$F7	\$XX,\$XX	2 octets de CRC
\$F8	\$F8	data address mark "effacé"
\$F9	\$F9	data mark
\$FA	\$FA	data mark
\$FB	\$FB	data address mark "normale"
\$FC	\$FC	data mark
\$FD	\$FD	data mark
\$FE	\$FE	index address mrk
\$FF	\$FF	

Nous devons maintenant expliquer comment on peut créer un format quelconque avec ces octets de contrôle. Comme des valeurs devront être données en exemple, nous utiliserons celles nécessaires au format ATARI. Nous vous dirons en temps voulu dans quelle mesure on peut modifier les valeurs.

Commençons avec le tampon qui contient toutes les informations sur une piste complète devant être écrite avec la commande WRITE TRACK. Sa taille doit être au minimum de 6250 octets.

Il faut remplir ce tampon de données qui correspondent à un format de 9 secteurs faisant chacun 512 octets de long.

Divisons notre tampon en deux composantes. Nous obtenons le schéma suivant qui est valable pour tous les formats. Le nombre de RECORDs peut naturellement varier.

GAP1	RECORD 1	RECORD 2	...//...	RECORD 9	GAP 5
------	----------	----------	----------	----------	-------

Une piste commence et finit avec un bloc nommé GAP. Comme nous le verrons plus tard, ces GAP sont aussi inclus dans les RECORDs. Un GAP est un espace séparant les différentes composantes d'une piste. Il ne contient aucune information utile mais seulement des octets de remplissage ou des octets de SYNC s'il se trouve avant un champ d'ID ou de données. Le FDC dispose donc d'un temps variant selon le GAP durant lequel il peut déterminer ses fonctions en rapport avec les composantes qui suivent.

Description	Valeur	Long. du GAP format ATARI	Long. du GAP autre format
GAP1 (Pre-piste)	\$4E	60 octets	min. 32 octets
GAP5 (Post-piste)	\$4E	env. 664 octets	min. 16 octets

La longueur du GAP5 est pour l'instant sans importance. LE GAP5, autrement dit est ce qui reste. Mais dans notre calcul de structure du tampon, il faut réserver au moins 16 octets pour le GAP5.

Si nous soustrayons de notre tampon le nombre d'octets nécessaires au GAP1, nous disposons de 6190 octets pour les records. Mais cette structure ne peut pas encore être déterminée car nous ne connaissons pas la taille d'un record.

Comme vous l'avez certainement déjà pensé, chacun de nos records contient un secteur parmi les 9. Nous sommes donc sûrs que les records sont au moins aussi longs que les secteurs. Si nous divisons encore un record, nous obtenons le schéma suivant :

GAP2	champ d'index	GAP3	champ de données	GAP4
------	---------------	------	------------------	------

Vous voyez les autres Gaps dont les noms sont pre-record, inter-record et post-record.

Description	Valeur	Longueur du GAP format ATARI	Long. du GAP autre format
CAP2	\$00	12 octets	min. 8 octets
SYNC	\$F5	3 octets	3 octets
GAP3	\$4E	22 octets	22 octets
	\$00	12 octets	12 octets
SYNC	\$F5	3 octets	3 octets
GAP4	\$4E	40 octets	min. 24 octets
Somme octets de GAP/record:		92 octets	min. 72 octets

Les octets de synchronisation (\$F5) des GAP2 et GAP3 servent à synchroniser la lecture des bits arrivant en série avec le début de l'octet. De plus, ils ont pour rôle d'indiquer au FDC l'existence d'une address mark et d'initialiser sa logique CRC. Pour écrire un octet de synchronisation, il faut envoyer la valeur \$F5 au FDC qui écrit alors un octet \$A1 sans impulsion de fréquence.

Le champ de données

L'utilité des GAPs qui se trouvent dans un record est donc claire. Voyons de plus près le champ de données dans lequel se trouve d'ailleurs notre secteur.

DAM	secteur	CRC
\$FB	512 octets de données	\$F7

Le champ de données commence par un DATA ADDRESS MARK qui indique le début du secteur. La valeur \$FB sera interprétée par une commande READ SECTOR ultérieure comme une DATA ADDRESS MARK "normale" alors que la valeur \$F8 qui peut être inscrite à la place du \$FB indique une DAM "effacée".

Le champ de secteur est rempli par des octets sans importance lors du formatage. Les valeurs peuvent être librement définies mais ne doivent en aucun cas dépasser \$F4.

Leur nombre peut être de 128, 256, 512 ou 1024. Nous expliquons comment le FDC reconnaît les différentes longueurs de secteurs dans la partie concernant le champ d'index.

La valeur \$F7 indique au FDC d'inscrire le contenu de son registre de CRC 16 bits (qui contient un checksum) sur disquette. Bien qu'on n'envoie qu'un octet, le contrôleur écrit deux octets.

La longueur totale du champ de données est de 515 octets avec une longueur de secteur de 512 octets.

Le champ d'index

Le champ d'index (ou champ d'ID) contient les informations sur le champ de données suivant.

ID-AM	piste	face	secteur	longueur	CRC
\$FE	00-79	00-01	00-09	00-03	\$F7

L'index Address Mark (ID-AM) est le marquage du début du champ d'ID. Si le contrôleur rencontre un ID-AM lors d'opérations de lecture ultérieures, il lit les 6 octets suivant si son address-mark-detector est activé. Le ID-AM est écrit sans impulsion de fréquence.

Les trois octets qui terminent l'ID-AM décrivent l'état du Record.

Il y a d'abord le numéro de la piste sur laquelle il se trouve. Dans notre cas, c'est une valeur entre 0 et 79, indépendamment de la piste formatée.

Le champ "face" indique si le record se trouve sur la face recto ou verso. Cet octet n'est jamais utilisé par le contrôleur lui-même dans quelque opération que ce soit.

Le champ de secteur contient le numéro de secteur (1-9). Etant donné que le FDC gère des secteurs de plusieurs tailles, il faut lui indiquer le nombre d'octets qui se trouvent dans le secteur suivant. Cela est inscrit dans le champ de longueur.

Tableau des longueurs de secteurs

<i>Champ de longueur</i>	<i>Octets par secteur</i>
00	128
01	256
02	512
03	1024

Avec un secteur de 512 octets, on a donc "02" dans ce champ.

Il manque seulement le checksum. Cette somme est écrite tout comme le champ de données en envoyant la valeur SF7. De plus, la longueur d'un champ d'ID est toujours de 7 octets.

Maintenant que nous avons expliqué toutes les composantes d'une piste et leur ordre, nous pouvons calculer la taille d'un record.

Champ de données		515 octets
Champ d'ID	+	7 octets
GAP2-GAP4	+	92 octets
<hr/>		
Longueur du record	=	614 octets

C'est la taille effective du record. Mais notre tampon ne comprendra que 612 octets car on n'envoie qu'un seul octet pour chaque CRC (qui prend deux octets sur disquette), soit deux octets en tout.

Mais dans notre calcul, c'est la place que prend un record sur disquette qui est déterminante, soit 614 octets.

Pour 9 records, nous avons donc besoin de $9 * 614 = 5526$ octets. Si nous soustrayons cette valeur des 6190 octets disponibles, il nous reste 664 octets pour le GAP5, post-piste.

C'est plus que suffisant quand on pense qu'il n'a besoin que de 16 octets. Même un format sur 10 secteurs de 512 octets laisse 50 octets pour le GAP5, ce qui est bien au dessus de la limite.

En conclusion, voyons les données nécessaires à la préparation de notre tampon. Précisons certaines données concernant le tableau que nous avons préparé :

Les données du GAP2 au GAP4 inclus (un record complet) sont répétées pour chaque secteur. Ainsi, pour un format utilisant par exemple 29 secteurs, il faudra écrire dans le tampon 29 fois le même bloc à la suite.

Les valeurs décrites par \$XX doivent être définies par vous-même. Ce n'est pas très compliqué. Si vous voulez formater la piste 54 par exemple, inscrivez un 54 pour le numéro de la piste.

La valeur de la face est soit 0 pour le recto, soit 1 pour le verso.

Le numéro de secteur commencera normalement par 1 et ira en ordre croissant. Mais la suite est variable et on peut avoir pour un format sur 9 secteurs la suite 3,6,9,1,4,7,2,5,8. Il est seulement nécessaire que la suite soit complète.

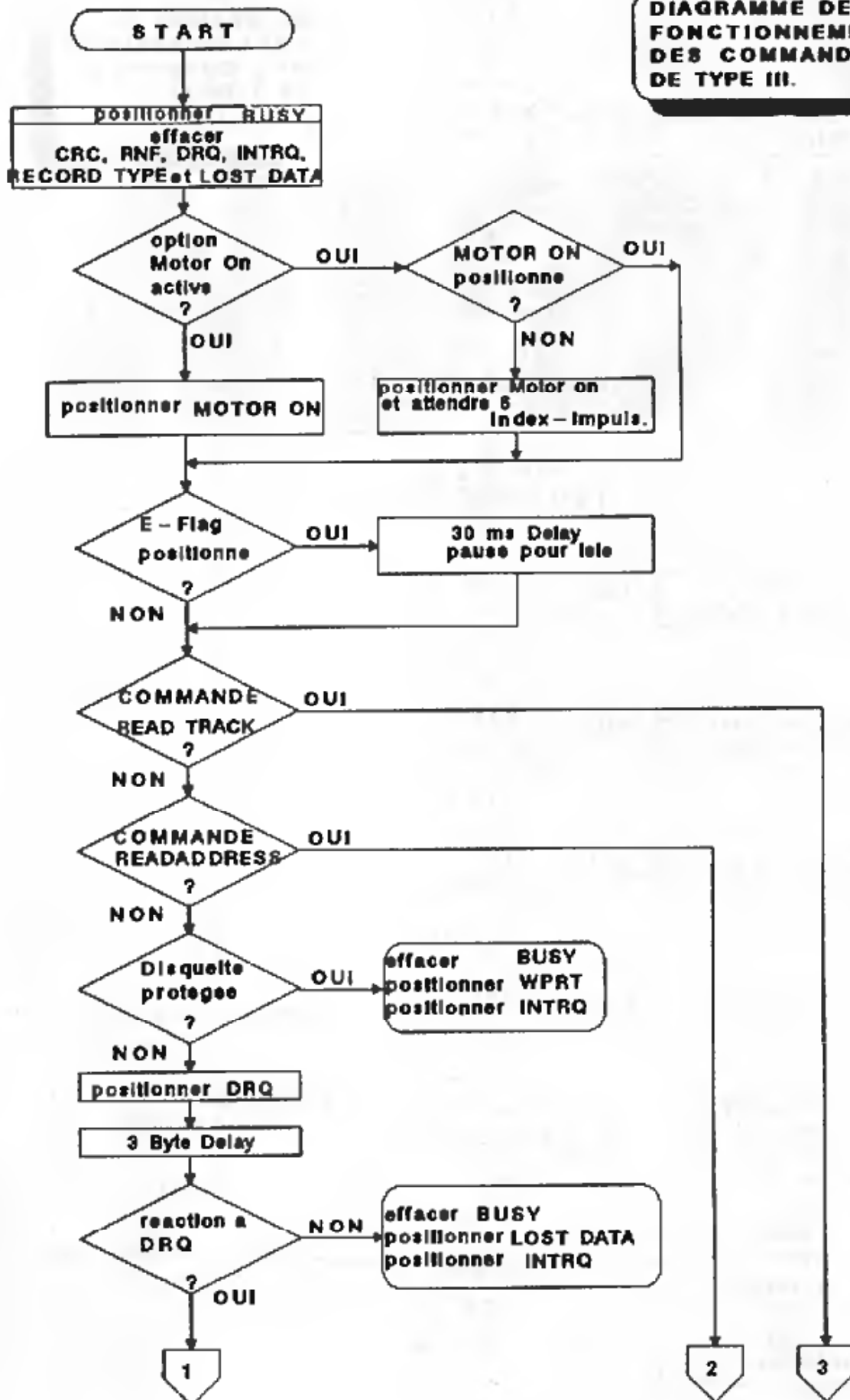
Si on décrit les secteurs par 1,2,3,5,6,7,8,9 par exemple, il se produira une erreur sur la piste 3 lors d'un READ SECTOR ou d'un WRITE SECTOR avec le bit *m* positionné. En effet, le FDC ne pourra pas trouver de secteur avec le numéro 4.

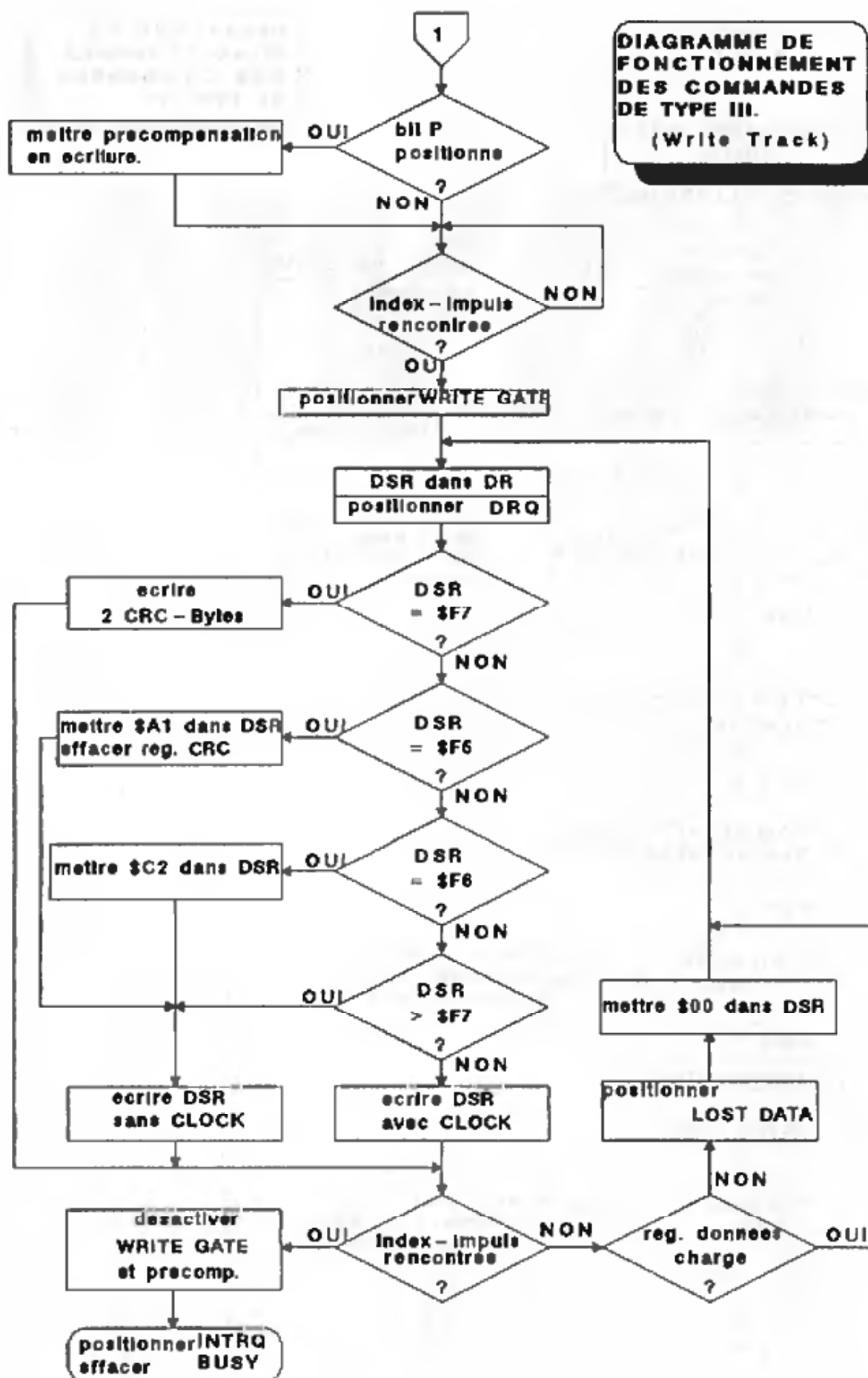
Le format ATARI est présenté en première colonne. Si vous jetez un oeil sur "A Hitchhiker's Guide to the BIOS", vous constaterez une différence avec notre tableau. La longueur du post-piste GAP5 n'est pas de 664 octets mais de 1401 octets. Mais si on additionne tous les octets, il apparaît que la longueur de la piste serait de 7000 octets, ce qui n'est pourtant pas le cas. En fait, on prépare un tampon qui est plus grand que la piste réelle. Mais on ne peut pas inscrire plus de 6250 octets sur une piste. Même un tampon de 50 Koctets ne pourra rien changer à ce fait.

	Nombres de secteurs / Taille des secteurs			
	9 / 512	18 / 256	29 / 128	5 / 1024
GAP1	60 * \$4E	42 * \$4E	40 * \$4E	60 * \$4E
GAP2	12 * \$00	11 * \$00	10 * \$00	40 * \$00
SYNC	3 * \$F5	3 * \$F5	3 * \$F5	3 * \$F5
ID-AM	1 * \$FE	1 * \$FE	1 * \$FE	1 * \$FE
Piste numéro	1 * \$XX	1 * \$XX	1 * \$XX	1 * \$XX
Face numéro	1 * \$XX	1 * \$XX	1 * \$XX	1 * \$XX
Secteur numéro	1 * \$XX	1 * \$XX	1 * \$XX	1 * \$XX
Longueur de secteur	1 * \$02	1 * \$01	1 * \$00	1 * \$03
ID-CRC	1 * \$F7	1 * \$F7	1 * \$F7	1 * \$F7
GAP3	22 * \$4E	22 * \$4E	22 * \$4E	22 * \$4E
	12 * \$00	12 * \$00	12 * \$00	12 * \$00
SYNC	3 * \$F5	3 * \$F5	3 * \$F5	3 * \$F5
OAH	1 * \$FB	1 * \$FB	1 * \$FB	1 * \$FB
Données	512 * \$E5	256 * \$E5	128 * \$E5	1024 * \$E5
DATA-CRC	1 * \$F7	1 * \$F7	1 * \$F7	1 * \$F7
GAP4	40 * \$4E	26 * \$4E	25 * \$4E	40 * \$4E
GAP5	664 * \$4E	36 * \$4E	33 * \$4E	420 * \$4E

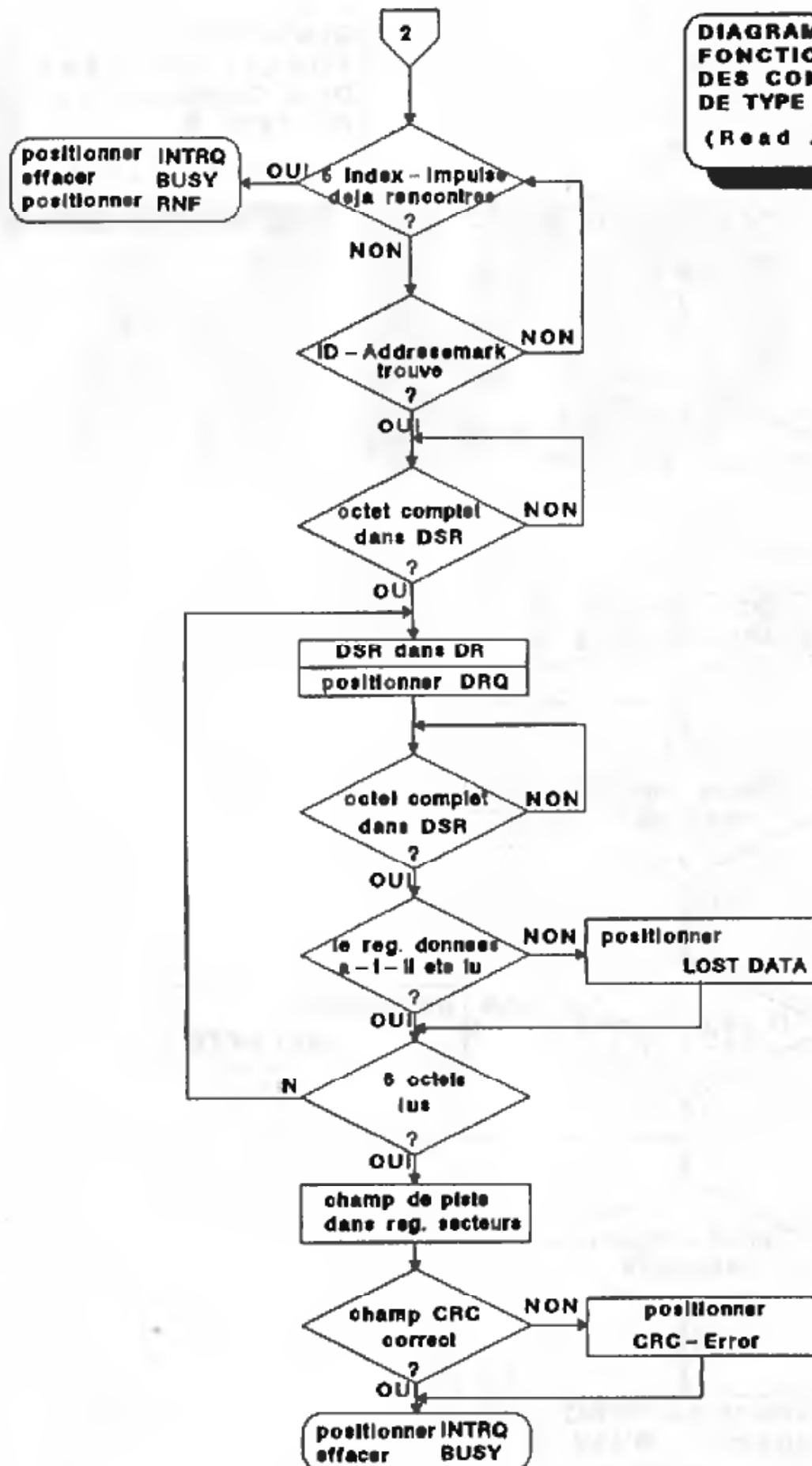
Diagramme synoptique des commandes de type 3.

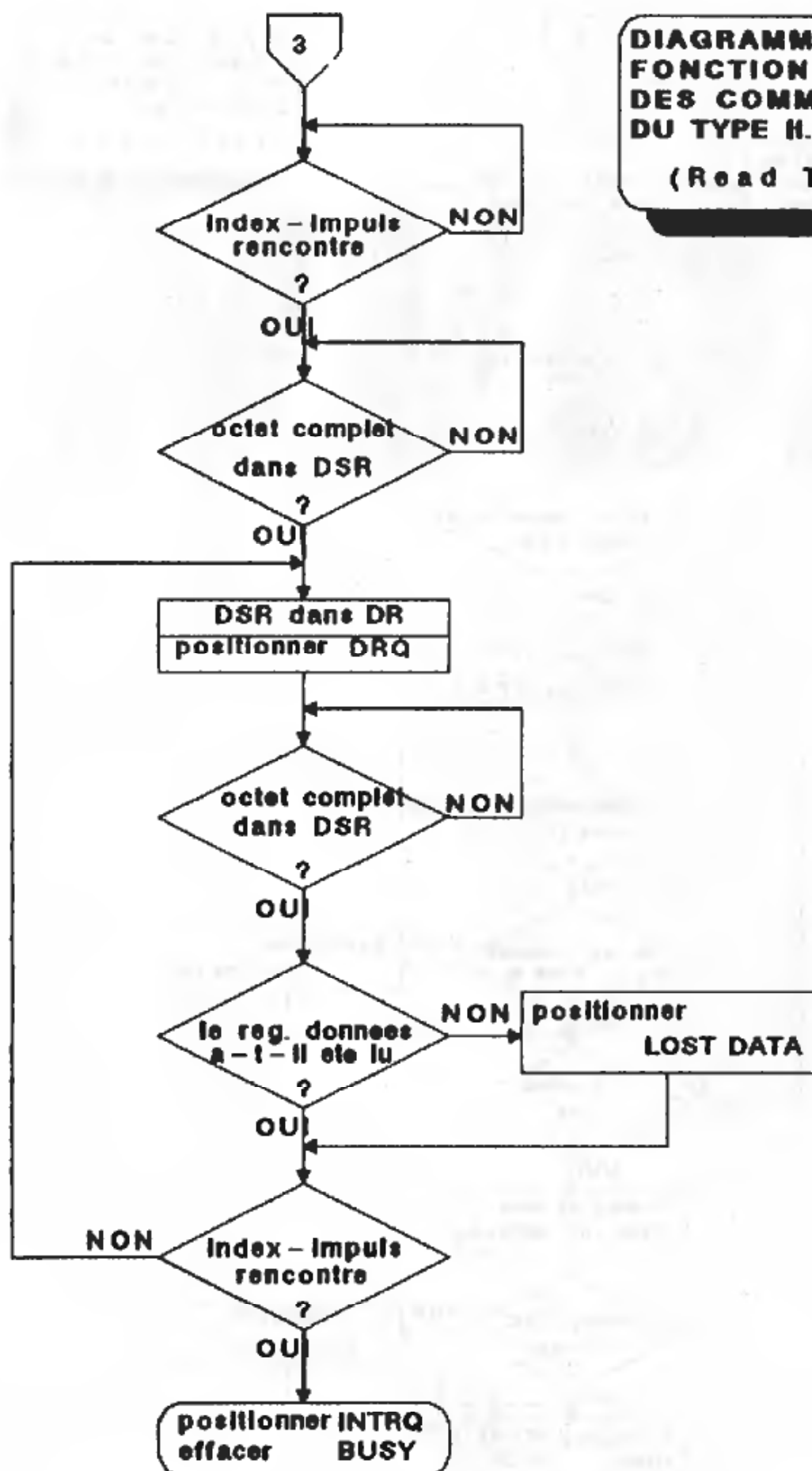
Nous avons réalisé un schéma de fonctionnement pour les commandes de type 3 également.

**DIAGRAMME DE
FONCTIONNEMENT
DES COMMANDES
DE TYPE III.**




**DIAGRAMME DE
FONCTIONNEMENT
DES COMMANDES
DE TYPE III.
(Read Address)**





LA COMMANDE DE TYPE IV

FORCE INTERRUPT

Commande :	7	6	5	4	3	2	1	0
	1	1	0	1	I3	I2	I1	I0

Cette commande est la seule qui peut être envoyée vers le contrôleur lorsqu'il exécute une autre commande. Elle sert principalement à interrompre une commande READ SECTOR ou WRITE SECTOR avec le bit *m* positionné.

Il existe trois possibilités pour l'interruption. Elles sont déterminées par les bits de condition (I0-I3) du mot de commande. I0 et I1 n'ont aucune signification et doivent rester effacés. Les bits I2 et I3 servent à définir le type de l'interruption comme suit :

(\$D4)	I2	= 1, interruption à chaque impulsion d'index.
(\$D8)	I3	= 1, interruption de l'exécution en cours.
(\$D0)	I2-I3	= 0, arrêt de la commande en cours, sans interruption.

L'interruption à chaque impulsion d'index (\$D4) peut servir par exemple à définir la vitesse de rotation du lecteur. Une autre application est la synchronisation de la commande READ ADDRESS avec le début de la piste. Avec les commandes WRITE TRACK et READ TRACK, cela n'est pas nécessaire car elles sont toujours exécutées avec l'impulsion d'index.

Les interruptions \$D8 et \$D0 permettent d'interrompre une commande en cours d'exécution. Il faut faire attention à ce que la sortie INTRQ ne soit pas repositionnée par une lecture ou une écriture après une interruption \$D8. Cela arrive seulement lorsque le \$D8 est suivi d'un \$D0 et que le registre d'état est lu ensuite.

Si on envoie une commande FORCE INTERRUPT au FDC, il faut garder un délai de 16 us avant la commande suivante. Sinon, la commande d'interruption n'est pas exécutée.

4.2.1.4 Interprétation de l'état

Il apparaît que la programmation de circuits périphériques destinés au transfert de données est presque entièrement prise en charge par le système d'exploitation. Mais le plus souvent, ces circuits offrent beaucoup plus de possibilité que nécessaire pour le système d'exploitation. C'est la raison pour laquelle le système d'exploitation n'utilise pas les possibilités les moins employées.

Mais dans certains cas, les talents inexploités du circuit périphérique peuvent fortement simplifier voire résoudre un problème de programmation. Pourquoi les programmeurs qui les exploitent sont-ils donc si peu nombreux ? Serait-ce parce qu'il n'existe aucun vecteur du système d'exploitation pour les appeler ? Non, la crainte de ce type de programmation tient plutôt à "la peur de l'état".

Cela signifie que l'on sait bien quel type de commande on envoie au circuit mais qu'on ne sait pas interpréter l'état qu'il renvoie. Souvent, on ne sait pas quel état on obtient après une exécution sans erreur parce que certains bits sont presque toujours positionnés. Un état OK peut donc varier. Si on ne sait pas cela, comment continuer le codage après un état sans erreur ?

Cela prend du temps d'essayer toutes les possibilités et de faire des expériences pour déterminer dans quel cas on obtient quel état. Mais ce n'est pas la peine. Nous avons déjà fait ce travail pour vous, en ce qui concerne le contrôleur de disquettes.

Nous vous montrons une fois de plus la signification des bits d'états que nous avons décrite dans le chapitre 4.2.1.2.

Registre d'état du FDC

Bit	Dénomination	Bit = 1 signifie
7	Motor On (MO)	Moteur en route
6	Write Protect (WPRT)	Disquette protégée
5	Record Type/ Spin Up ou Spin Up	DATA MARK effacé Rotation Optimale
4	Record Not Found (RNF)	Secteur pas trouvé
3	CRC Error (CRC)	Erreur de checksum
2	Piste 0 ou Lost Data	Tête sur la piste 0 Perte de données
1	Impulsion d'index ou Data Request	état de l'impul. d'id Prêt au transfert
0	Busy	Commande active

Il est utile de savoir que dès le positionnement de la disquette on sait s'il y a une disquette protégée (ou aucune disquette).

Pour toutes les commandes (Type 1 et Type 3), le mot d'état qui est lu juste après une commande a les caractéristiques suivantes : le bit 7 est positionné (le moteur n'est pas immédiatement arrêté) et le bit 0 est effacé (le FDC a fini la commande). De plus, le bit 5 est positionné après une commande du type 1 (Spin Up).

L'état après une commande de type 1.

Commençons avec l'état correct après une commande du type 1. Dans les mots de commande utilisés, les "Stepping-rate-bits" sont positionnés pour 3 ms "Track to Track" (r0=1, r1=0).

Commande RESTORE	Normal	Protégé contre écriture
01 (avec option MO, sans Verify)	A4	E4
01 (voir (1))	A6	E6
05 (avec option MO, avec Verify)	A4	E4
09 (sans option MO, sans Verify)	A4	E4
0D (sans option MO, avec Verify)	A4	E4

Commande SEEK	Normal	Protégé contre écriture
11 (avec option MO, sans Verify)	A0	E0
11 (voir (3))	A2	E2
11 (voir (2))	A4	E4
11 (voir (1))	A6	E6
15 (avec option MO, avec Verify)	A0	E0
15 (voir (2))	A4	E4
19 (sans option MO, sans Verify)	A0	E0
19 (voir (2))	A4	E4
1D (sans option MO, avec Verify)	A0	E0
1D (voir (2))	A4	E4

STEP, STEP-IN, STEP-OUT	Normal	Protégé contre écriture
x1 (avec option MO, sans Verify)	A0	E0
x1 (voir (2))	A4	E4
x5 (avec option MO, avec Verify)	A0	E0
x5 (voir (2))	A4	E4
x9 (sans option MO, sans Verify)	A0	E0
x9 (voir (2))	A4	E4
xD (sans option MO, avec Verify)	A0	E0
xD (voir (2))	A4	E4

- (1) Cette valeur est valable lorsque la tête de lecture/écriture se trouvait déjà sur la piste 0 avant une commande RESTORE ou SEEK vers la piste 0. A coté du bit de piste 0, le bit IP est positionné. Cela tient au fait que l'on attend 6 impulsions d'index lorsque l'option Motor On est activée. Ainsi, le FDC détermine pendant une impulsion d'index si la piste désirée a été atteinte et interrompt la commande.
- (2) On obtient cet état avec une commande SEEK, STEP ou STEP-OUT lorsque la tête de lecture/écriture est positionnée sur la piste 0.

- (3) Si la tête de lecture/écriture se trouve déjà sur la bonne piste (en dehors de la piste 0) lors d'une commande SEEK, le bit IP du mot d'état est positionné. Il s'agit du même fait que ce qui est décrit en (1).

On n'obtiendra un état d'erreur après une commande du type 1 que si le bit Verify du mot de commande est positionné. Dans le mot d'état, on a en outre :

- (a) Au cas où on n'a trouvé aucun champ d'ID, le bit RNF est positionné.

et

- (b) Au cas où on n'a trouvé aucun champ d'ID correct, les bits RNF et CRC sont positionnés.

Ainsi, avec une disquette protégée, on obtient un état de B2, F2 ou FA.

Si cela se passe sur la piste 0, le bit de la piste 0 est bien sûr lui aussi positionné. Le mot d'état a alors pour valeur soit B6 ou BE soit F6 ou FE avec une disquette protégée.

Il apparaît que dans le cas d'une erreur, le bit IP est positionné. Mais dans ce cas, cela n'a rien à voir avec l'option Motor-On mais s'explique par le fait que la recherche du champ d'ID est interrompue par la sixième impulsion d'index.

L'état après une commande du type 2

Dans une commande du type 2, l'interprétation de l'état est plus simple.

Après une commande WRITE SECTOR réussie, le registre d'état contient toujours la valeur 80.

Après une commande READ SECTOR, le mot d'état peut contenir A0 lorsqu'un secteur a été lu avec une DATA MARK effacée. Sinon, on trouvera un 80.

Si la commande n'a pas été réussie, l'état après une commande WRITE SECTOR est soit :

- (a) C0 après avoir tenté d'écrire sur une disquette protégée contre l'écriture.
- (b) 90 si le champ d'ID correspondant au secteur désiré n'a pas été trouvé.
- (c) 88 si la checksum (CRC) du champ d'ID n'était pas correcte.

ou

- (d) 84 s'il n'y a eu aucune réaction à un DATA REQUEST du FDC.

Après une commande READ SECTOR sans erreur :

- (a) 90 si le champ d'ID du secteur désiré ou la DATA MARK n'a pas été trouvé.
- (b) 98 si la checksum (CRC) du champ d'ID n'était pas correcte.
- (b) 88 si la checksum (CRC) du champ de données retournait une erreur.

ou

- (c) 84 s'il n'y a eu aucune réaction à un DATA REQUEST du FDC.

L'état après une commande de type 3

L'état après une commande de type 3 est encore plus simple. La valeur 80 indique une exécution sans erreur.

Dans le cas d'une erreur après une commande WRITE TRACK, on a :

- (a) C0 avec une disquette protégée contre l'écriture.

ou

- (b) 84 s'il n'y a eu aucune réaction à un DRQ du FDC.

Il n'y a pas d'exécution sans erreur de la commande READ TRACK. Le FDC lit simplement l'entrée RD entre deux impulsions d'index. Peu importe qu'il y ait une disquette dans le lecteur ou non.

La seule erreur qui peut être représentée, un LOST DATA (état 84), peut ne pas être interprétée à cause d'une erreur de logiciel dans un microprogramme du FDC. Le LOST DATA bit est positionné indépendamment du format lu, sauf en cas de perte de données.

Ainsi, il n'est pas possible de déterminer dans une commande READ TRACK si un bit LOST DATA indique une perte de données ou autre chose.

Il reste la commande READ ADDRESS. Dans ce cas également, un état de 80 est correct. De plus, on peut obtenir :

- (a) 90 si aucun champ d'ID n'a été trouvé.
- (b) 88 si le FDC a reconnu une erreur de checksum dans le champ d'ID.

ou

- (c) 84 s'il n'y a eu aucune réaction à une impulsion de transfert de données.

4.2.2 L'interface avec le lecteur de disquette

La prise assez visible située à l'arrière du ST possède 14 broches. Ces 14 broches servent à contrôler le lecteur et le transfert de données. Le fonctionnement de ce contrôle est très simple à décrire car le lecteur de disquette n'a pas d'intelligence propre.

Cela présente un grand avantage. L'interface avec de tels lecteurs de disquette est normalisée. Il s'agit d'une interface SHUGART qui se trouve sur de nombreux lecteurs de disquettes. C'est pourquoi il est très simple d'adapter d'autres lecteurs à l'ATARI.

Cette interface possède une connexion avec 34 broches qui est le plus souvent prévue pour des cables plats. La moitié de ces 34 broches est reliée ensemble et porte le pôle moins, c'est-à-dire la masse. Avec un cable plat, ces cordons sont toujours situés au milieu des autres car les broches impaires sont toutes à la masse.

Cela tient simplement au fait qu'il y a toujours une masse entre deux cables. On obtient ainsi une sorte de blindage entre les différents signaux, ce qui est très important pour la fréquence très élevée des signaux.

Sur les 18 broches qui restent, 14 sont reliées à l'ATARI. Voyons ces signaux sur le bus SHUGART :

Broche 2 : Head Load

Un signal nul sur cette broche a pour effet de laisser la tête sur la disquette. Cette mesure sert à protéger les disquettes car la tête ne se trouve sur les disquettes que lorsque c'est nécessaire. Malheureusement ce signal n'est pas utilisé par l'ATARI ST car le contrôleur de lecteur WD 1772 ne dispose pas de cette broche. Mais cette broche est souvent reliée à MOTOR ON.

Broche 3 : Masse

A partir de celle-ci, toutes les broches impaires jusqu'à la 33^{ième} sont branchées à la masse. Ce moins est également utilisé comme blindage.

Broche 4 : in Use

Ce signal indique au lecteur de disquette qu'il est branché et en cours d'utilisation. Cette broche n'est pas reliée à l'ATARI ST.

Broche 6 : Drive Select 3

Un signal nul sur cette broche signifie que seul le lecteur 3 est concerné. Seul le lecteur qui est défini comme étant le lecteur 3 grâce à ce qu'on appelle un JUMPER (une petite prise dans le lecteur) réagit aux instructions suivantes. Tous les autres lecteurs restent sans réaction. Ce signal n'est pas utilisé par l'ATARI ST car on peut brancher au plus deux lecteurs (0 et 1 ou A et B).

Broche 8 : Index

Le lecteur envoie un signal nul sur cette broche à chaque tour de la disquette. Ce signal signifie pour le contrôleur que les données suivantes se trouvent tout au début de la piste courante. On peut ainsi synchroniser le contrôleur.

Broche 10 : Drive select 0

Ce signal correspond à celui de la broche 6 mais c'est le lecteur 0 qui est concerné (lecteur A).

Broche 12 : Drive Select 1

Comme plus haut mais pour le lecteur 1 (disque B).

Broche 14 : Drive Select 2

Comme plus haut mais avec le lecteur 2. Cela n'est pas possible sur le ST car on ne peut brancher que deux lecteurs.

Broche 16 : Motor On

Un signal 1 sur cette broche met en route les moteurs de tous les lecteurs branchés. Un 0 les arrête de nouveau.

Broche 18 : Direction

Ce signal indique la direction dans laquelle va se déplacer la tête de lecture/écriture. Avec un nul, elle va vers le centre de la disquette, c'est-à-dire vers la piste 79 alors qu'un 1 la conduit vers l'extérieur, c'est à dire vers la piste 0.

Broche 20 : Step

Une impulsion nulle indique au moteur pas à pas de déplacer la tête de lecture/écriture d'un pas dans la direction indiquée.

Broche 22 : Write Data

Cette broche transporte les données séries qui doivent être écrites sur disquette.

Broche 24 : Write Gate

Ce signal sélectionne la direction des données. S'il y a un nul, on écrit sur la disquette, s'il y un 1, on lit les données. Si la protection de la disquette contre l'écriture est positionnée, il n'est pas possible d'écrire dessus.

Broche 26 : Piste 0

Si la tête de lecture/écriture se trouve sur la piste 0, on a un signal 0 sur cette broche.

Broche 28 : Write Protect

Un 0 sur cette broche signifie que la disquette est protégée.

Broche 30 : Read Data

Cette broche transporte les données lues vers l'ordinateur.

Broche 32 : Side Select

Cette broche sert à sélectionner la face de la disquette. Un nul sélectionne la face 1 et un 1 la face 0. Si on utilise un lecteur simple face, cette broche est inutilisée.

Broche 34 : Ready

Un nul sur cette broche indique qu'il y a une disquette dans le lecteur et qu'elle est en rotation. Cette broche permet à l'ordinateur de déterminer si on a changé de disquette. Cette broche n'est pas utilisée par l'ATARI ST.

Tous ces signaux correspondent au standard TTL, c'est-à-dire qu'un 0-0,4 Volt signifie LO (nul), 2,5-5,25 signifie HI (un). Pour assurer ces signaux, la plupart des lecteurs a une série de résistances pull-up.

Si on branche plusieurs lecteurs en parallèle, il est utile d'enlever ces résistances jusqu'au dernier lecteur (sur l'ATARI, le deuxième) afin de ne pas surcharger la sortie de l'ATARI. Pour certains lecteurs (l'EPSON par exemple), ces résistances sont regroupées en un seul composant, si bien qu'il est très facile de les enlever. Mais dans les lecteurs originaux d'ATARI, c'est-à-dire des lecteurs EPSON, ce n'est pas le cas.

4.3 BRANCHEMENT DU LECTEUR DE DISQUETTES

Le lecteur de disquettes qui est vendu par ATARI se branche très facilement : il suffit de connecter le câble. Il est plus compliqué d'adapter un autre lecteur de disquettes. Le problème tient en premier lieu à la difficulté de se procurer une prise convenante.

On peut résoudre ce problème en soudant des broches sur un circuit imprimé ou une plaque de plastique.

Si on a réussi à se procurer une prise adaptée, on peut s'attaquer au problème du branchement. Il est préférable d'utiliser un câble blindé si la longueur dépasse le 1 mètre. En raison de la haute vitesse de transfert, des effets électriques comme l'inductivité et la capacité apparaissent et peuvent influencer sur la qualité du transfert. Au mieux, on utilisera un câble dans lequel tous les cordons sont isolés.

Une fois qu'on a tous les composants nécessaires, il faut souder le câble. Voici encore une fois les broches :

ATARI ST	Connexion	Bus SHUGART
1	Read Data	30
2	Side 0 select	32
3	Ground	toutes les broches impaires
4	Index impulse	8
5	Drive 0 Select	10
6	Drive 1 Select	12
7	Ground	cf. plus haut
8	Motor On	16
9	Direction In	18
10	Step	20
11	Write Data	22
12	Write Gate	24
13	Track 00	26
14	Write Protect	28

Si on branche deux lecteurs, toutes les broches doivent être connectées en parallèle. La sélection du lecteur (A ou B) se fait directement par le lecteur. Il faut pour cela débrancher des JUMPERs, c'est-à-dire des petites prises qui relient différents contacts. Pour savoir où se trouvent ces JUMPERs, voyez dans le manuel de votre lecteur.

Il est possible de créer une disquette de 5.25" de diamètre, mais elle ne sera pas compatible avec les lecteurs de disquette ATARI ST.

Il est possible de créer une disquette de 3.5" de diamètre, mais elle ne sera pas compatible avec les lecteurs de disquette ATARI ST.

Il est possible de créer une disquette de 5.25" de diamètre, mais elle ne sera pas compatible avec les lecteurs de disquette ATARI ST.

Il est possible de créer une disquette de 3.5" de diamètre, mais elle ne sera pas compatible avec les lecteurs de disquette ATARI ST.

Il est possible de créer une disquette de 5.25" de diamètre, mais elle ne sera pas compatible avec les lecteurs de disquette ATARI ST.

Il est possible de créer une disquette de 3.5" de diamètre, mais elle ne sera pas compatible avec les lecteurs de disquette ATARI ST.

Il est possible de créer une disquette de 5.25" de diamètre, mais elle ne sera pas compatible avec les lecteurs de disquette ATARI ST.

Il est possible de créer une disquette de 3.5" de diamètre, mais elle ne sera pas compatible avec les lecteurs de disquette ATARI ST.

Il est possible de créer une disquette de 5.25" de diamètre, mais elle ne sera pas compatible avec les lecteurs de disquette ATARI ST.

Il est possible de créer une disquette de 3.5" de diamètre, mais elle ne sera pas compatible avec les lecteurs de disquette ATARI ST.

Il est possible de créer une disquette de 5.25" de diamètre, mais elle ne sera pas compatible avec les lecteurs de disquette ATARI ST.

Il est possible de créer une disquette de 3.5" de diamètre, mais elle ne sera pas compatible avec les lecteurs de disquette ATARI ST.

Il est possible de créer une disquette de 5.25" de diamètre, mais elle ne sera pas compatible avec les lecteurs de disquette ATARI ST.

Il est possible de créer une disquette de 3.5" de diamètre, mais elle ne sera pas compatible avec les lecteurs de disquette ATARI ST.

5. Le disque dur SH204

Venons-en à quelque chose de plus cher mais de beaucoup plus rapide dans le domaine de la mémorisation de données : le disque dur. Il existe un disque dur pour l'ATARI ST, et ce à un prix assez intéressant.

Quels sont donc les avantages et les inconvénients du disque dur ? Le premier inconvénient est évident : un disque dur coûte sensiblement plus cher qu'un lecteur de disquette. D'autre part, il n'est pas possible d'échanger des programmes sur disque et de se créer une bibliothèque comme avec les disquettes.

Mais si l'on considère les avantages du disque dur, l'investissement s'avère intéressant. D'une part, il y a la rapidité de transfert de données entre le disque et l'ATARI ST. Elle est jusqu'à 10 fois plus rapide qu'avec un lecteur de disquette.

Un autre avantage est la capacité du disque dur qui atteint pour l'instant 20 Mégaoctets. Un tel disque peut contenir par exemple l'ensemble des programmes et des fichiers d'un compilateur complet, avec les sources C ou PASCAL. Etant donné que ce type de compilateur fonctionne la plupart du temps en faisant de nombreux accès sur disquette (ou sur disque), cela présente un gros avantage. On évite d'avoir à changer de disquette à chaque instant.

On utilise également beaucoup les disques durs dans la gestion de données. Dans ce domaine, il serait impensable de changer de disquette à chaque opération ! Imaginez que les employés d'une banque soient obligés d'insérer une disquette à chaque nouveau compte.

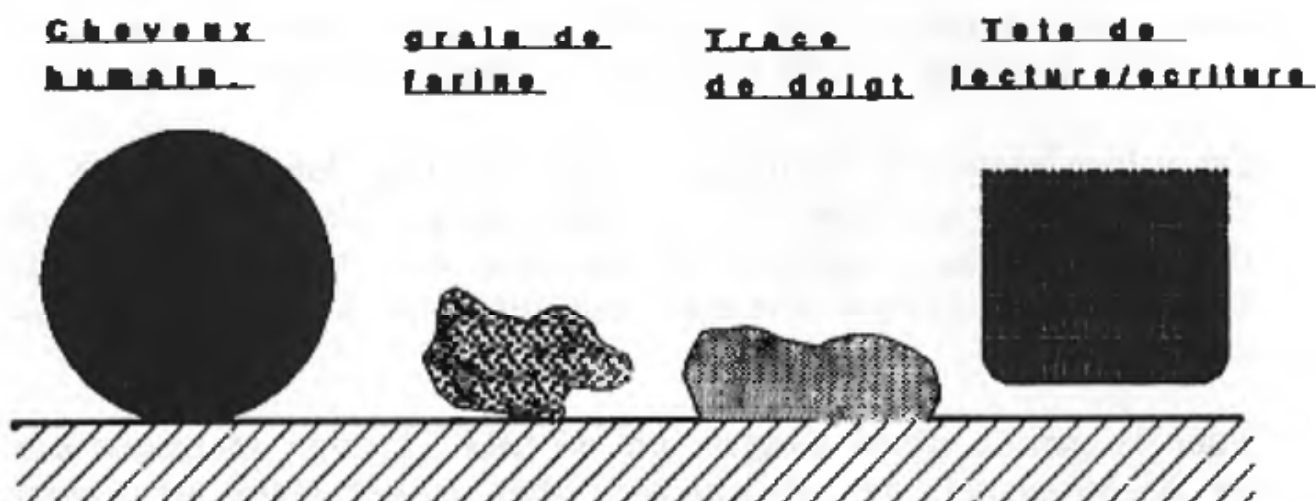
Pour la gestion d'une banque, un ATARI ST avec un disque dur de 20 Mégaoctets ne convient pas. Mais on peut gérer un secteur restreint comme la gestion de stock ou la gestion de personnel. Cela représente l'utilisation principale des disques durs.

Voyons maintenant comment on peut gérer une telle masse de données avec un disque dur et naturellement avec l'ordinateur qui y est relié.

5.1 FONCTION ET STRUCTURE

La fonction d'un disque dur est très semblable à celle d'un lecteur de disquette. Dans ce cas également un ou plusieurs disques (sur l'ATARI, un seul) tournent avec une vitesse constante et sont parcourus par une tête de lecture/écriture. Mais il y a des différences importantes avec un lecteur de disquette. La vitesse de rotation du disque dur est considérablement plus importante que celle des disquettes pour permettre une vitesse de transfert plus élevée.

En ce qui concerne la tête de lecture/écriture qui vole au dessus de ce disque (qui tourne environ 10 fois plus vite qu'une disquette), elle ne repose pas sur le disque. Grâce à un chef d'oeuvre de technique, cette tête reste à une légère distance de la surface du disque. Cet écart est tellement minime qu'une allumette ressemblerait à un tronc d'arbre à côté.



Si une allumette se trouvait sur le disque en rotation et si elle rencontrait la tête de lecture, cela pourrait provoquer des dommages sur le disque et la tête de lecture/écriture. On nomme ce type d'accident un "HEADCRAASH". Ces ennuis sont craints par tous car ils entraînent souvent des effets très coûteux.

Pour éviter un headcrash, le disque dur est enfermé avec la tête de lecture dans un coffret sous vide. On comprend alors pourquoi les disques durs ne peuvent pas être échangés comme les disquettes. Il existe sur le marché des disques durs qui peuvent être échangés mais ils coûtent très chers. De plus, il n'en existe aucun pour l'ATARI ST. Aussi, nous ne nous y intéresserons pas plus.

Une autre différence entre un lecteur de disquette et un lecteur de disque dur réside dans le contrôleur. Le contrôleur de disquette intégré à l'ATARI ST ne convient que pour les lecteurs de disquette comme son nom l'indique. Par contre, le disque dur a son propre contrôleur qui est intégré dans le boîtier. Ainsi, il n'est plus aussi facile d'adapter un autre disque à l'ATARI ST.

Voyons maintenant de plus près ce contrôleur.

5.1.1 Le contrôleur de disque dur

Le contrôleur utilisé dans le disque de l'ATARI ST est un appareil très puissant. Ce contrôleur autorise une vitesse de transfert de données atteignant les 8 Mbit par seconde, c'est à dire 1 Mégaoctet/seconde. Malheureusement, ce chiffre n'est pas valable pour le véritable transfert de données.

Un premier frein important est constitué par la partie mécanique du disque dur. Il faut d'abord que soit atteinte la vitesse optimale de rotation du disque et du moteur, que la tête de lecture/écriture soit à la bonne place, c'est à dire sur la bonne piste. Tous ces points diminuent considérablement la vitesse du transfert de données qui est déjà très haute.

Le contrôleur a une structure très simple. Mais sa police d'instruction est tellement plurifacettes qu'il dispose même d'une correction d'erreurs.

Le hardware du contrôleur est constitué essentiellement d'un contrôleur de disque, d'un codeur/décodeur et d'un microcontrôleur. Ces différentes parties ont les fonctions suivantes :

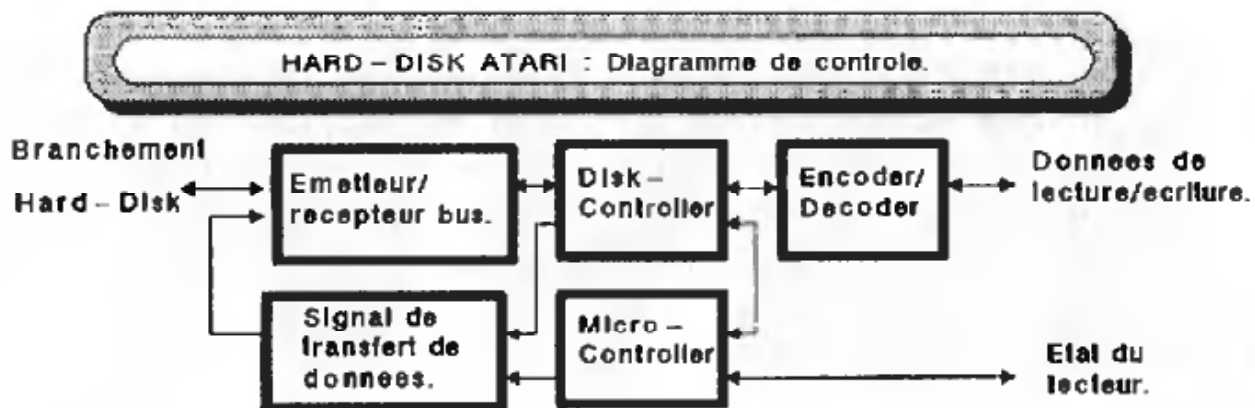
Le contrôleur de disque transforme les données du format série en format parallèle et vice versa. D'autre part, il transforme les données proprement dites en motif de bits qui est écrit sur le disque. Ce nouveau format permet de reconnaître les erreurs simples grâce à un truc.

Le codeur/décodeur transforme les données obtenues du contrôleur de disque en signaux électriques qui contrôlent la tête de lecture/écriture. Et à l'inverse, il transforme les signaux électriques en provenance de la tête en bit. Il sert en même temps de séparateur de données (voir contrôleur de disquettes).

Le microcontrôleur travaille comme un contrôleur normal. Ses fonctions sont :

- Interprétation des commandes venant de l'ordinateur.
- Sélection du lecteur adressé (normalement il n'y en a qu'un).
- Sélection de la tête dans le lecteur (face supérieure et inférieure du disque dur).
- Contrôle du moteur pas à pas qui conduit la tête de lecture/écriture dans la position correcte.
- Création de l'état.

Voici ici un simple diagramme du disque dur de l'ATARI :



Les opérations qu'on exécute avec le disque dur au travers du bus DMA sont divisées en 5 phases. Ces phases sont définies comme suit :

Phase de Reset

Elle se produit lorsque l'on presse la touche RESET de l'ordinateur ou lorsqu'on exécute la commande RESET du langage machine 68000. Le bus et le HDC sont mis à l'état de base.

Phase de bus libéré

Elle se produit lorsqu'aucun appareil ne fait d'accès au bus.

Phase de sélection de l'objet

Elle commence avec l'appel d'un périphérique en activant la sortie SEL. L'adressage du périphérique désiré se fait en positionnant un bit de données du port 8 connecteurs. Le périphérique adressé (ici HDC) répond avec un signal Busy avec lequel la sortie SEL est de nouveau activée. Ensuite commence la...

Phase d'information de transfert.

Durant cette phase, sont transférés :

- Le bloc de commandes, 6 octets du ST vers le HDC.
- Le ou les blocs de données si la commande l'exige.
- L'octet d'état du HDC vers le ST qui indique une opération réussie ou une erreur. Cet octet est toujours nul si bien qu'on ne peut obtenir l'état qu'avec un Timeout éventuel.
- L'octet de completion du HDC vers le ST est un octet nul qui signale la fin de toute l'opération.

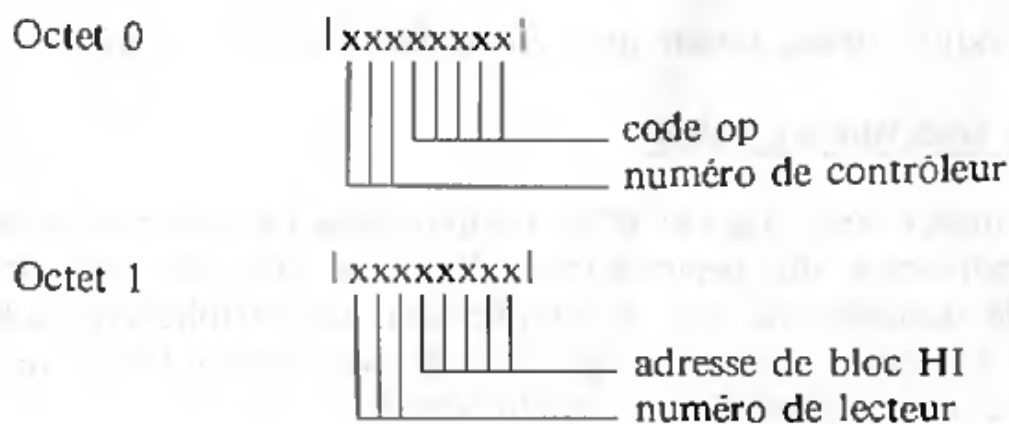
Phase de libération de bus

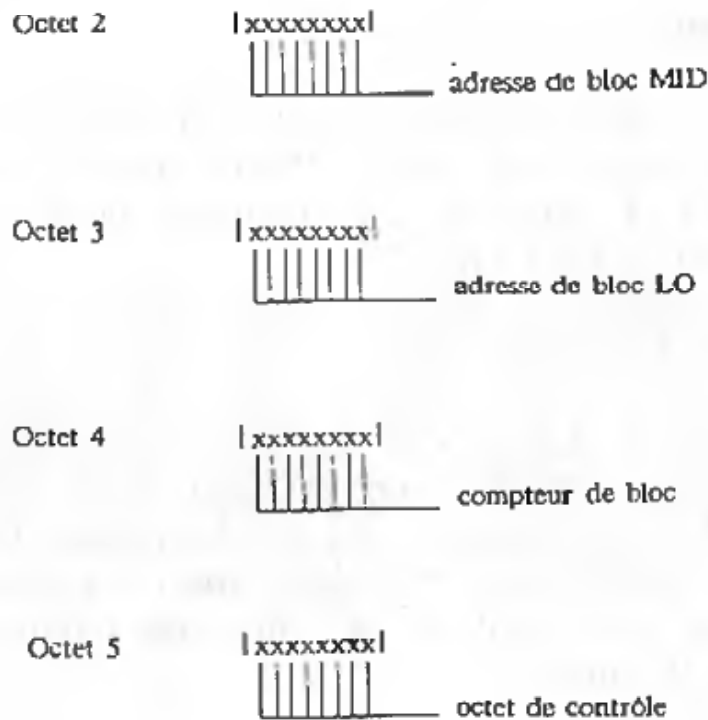
Elle est réalisée par l'activation de la broche Busy et elle signifie que le bus est libre pour l'opération suivante. Le bus se trouve alors dans la phase de bus libéré.

5.1.1.1 Structure d'instructions

Le transfert d'instruction vers le Hard Disk Controller est parfaitement défini. Chaque commande est envoyée sous la forme d'un bloc de 6 octets, le 'command descriptor bloc'. Si le contrôleur reçoit une telle commande, il l'indique à l'ATARI ST avec une interruption. Si la commande contient une instruction de recherche d'une piste donnée (Verify, Format Track, Read, Write), celle-ci est exécutée automatiquement. Le bloc de données logique désiré est alors divisé en grandeurs physiques comme la face, le numéro de piste et transformé par le contrôleur.

Le diagramme suivant montre la structure d'un bloc de commandes :





Numéro de contrôleur :

C'est une valeur sur 3 bits (0-7) qui représente le numéro du contrôleur sélectionné. Ainsi, il est possible de brancher jusqu'à 8 contrôleurs. Le numéro de contrôleur est déterminé par 3 micro-interrupteurs sur la plaque de circuit imprimé du disque. Si une commande arrive au bus du contrôleur, il teste s'il est concerné. Sinon, il se comporte comme s'il n'existait pas. Si oui, la communication entre l'ordinateur (l'initiateur) et le contrôleur concerné commence. Si aucun contrôleur ne répond à la commande, l'ordinateur envoie un TIMEOUT après environ 4 secondes.

Il faut remarquer une chose : les rôles de chaque appareil en tant qu'initiateur ou objet sont définis. Ainsi, il n'est pas possible d'avoir une communication entre deux appareils du même type.

Codeop :

Ce code contient la commande sur 5 bits. Ainsi, on ne peut avoir que des commandes de 0 à 31.

Numéro de lecteur :

Comme le numéro de contrôleur, il s'agit d'un nombre sur 3 bits qui décrit le lecteur sélectionné. Ainsi, chacun des 8 contrôleurs peut contrôler jusqu'à 8 lecteurs. On pourrait donc théoriquement brancher 8 lecteurs à l'ATARI.

Adresse de bloc :

Ce nombre sur 21 bits décrit le secteur de données logique sélectionné. La transformation de ce nombre en valeur physique (jusqu'à 2097151) est autorisée par le contrôleur. Le disque dur ATARI contient 41616 secteurs si bien que les adresses de bloc de ces secteurs ne se recouvrent pas et comme on commence à 0, elles ne dépassent pas la limite.

Compteur de bloc :

Ce compteur définit le nombre des secteurs à lire ou à écrire. Le compteur doit contenir une valeur différente de 0 (1-255).

Octet de contrôle :

Cet octet contient différentes données suivant la commande utilisée.

Pour envoyer un bloc de commande de ce type au HDC, il faut suivre la procédure que voici :

D'abord, il faut mettre le CPU en mode superviseur avec la fonction SUPER (\$20) du GEMDOS (TRAP #1) car le hardware effectuera certains accès privilégiés.

Ensuite, on désactive les routines de traitement du lecteur de disquette en positionnant la variable système FLOCK (\$43E). Cela est indispensable car le HDC et le FDC sont contrôlés par les mêmes registres HARD. Cependant, afin qu'il n'y ait aucune interférence comme par exemple un OK d'un contrôleur alors qu'on attend le OK de l'autre contrôleur, le FDC est pour ainsi dire écarté du système et ne peut plus interférer.

Lorsqu'on inscrit la valeur \$88 dans le registre \$FF8606 (appelé WDL par la suite), les bits 7 et 3 sont positionnés, les autres bits sont effacés. Ainsi, on sélectionne d'une part le HDC et d'autre part la broche A1 qui est adressé par le BIT 1 en mettant ce bit à 0.

Cette broche A1 sert à indiquer au HDC qu'un seul octet de commande est envoyé (le premier octet d'un bloc de commande).

Ensuite, on envoie l'octet de commande dans le registre \$FF8604 que nous appelons WDC par la suite. Le HDC prend cet octet et envoie un signal 0 sur la broche d'interruption du HDC. Cette broche est sur le bit 5 du port I/O du multi-fonction-chip MFP et se trouve à l'adresse \$FFFA01.

Cette interruption se produit également après chaque octet transféré. Si elle ne survient pas, soit l'octet n'a pas été reconnu, soit le HDC n'est pas prêt à recevoir les données.

Pendant le transfert de l'octet de commande, l'interruption est attendue pendant au plus 100 millisecondes et même jusqu'à 3 secondes après le transfert complet du bloc de commande. En effet, il faut que cette commande soit exécutée totalement pour que le HDC puisse envoyer un OK. S'il n'y a pas eu d'interruption après ce délai, le transfert de l'octet de commande est interrompu et un TIMEOUT est envoyé.

Si l'octet de commande a été transféré et si l'interruption s'est produite normalement, un \$8A est inscrit dans le WDL, ce qui met la broche A1 à 1.

Les 5 octets restants dans le bloc de commande sont transférés d'après le même schéma mais avec le Bit 1 positionné (A1). L'octet est inscrit dans le WDC avec un \$8A dans le WDL, l'interruption est attendue au plus 100 millisecondes (sinon timeout) et l'octet suivant est transféré.

Une fois que le dernier octet a été transféré (octet 5), l'interruption est attendue pendant un temps plus long (Max. 3 secondes). Ainsi, le HDC a suffisamment de temps pour exécuter la commande.

Si l'interruption s'est produite dans le temps, un \$80 est inscrit dans le registre WDL, ce qui repositionne le bit 3 et "désélectionne" le HDC. Ainsi, le FDC, le floppy disk controller, est de nouveau libre.

Enfin, la variable système FLOCK (\$43E) est repositionnée à 0 pour autoriser de nouveau les opérations avec le lecteur de disquette. Enfin, le processeur peut être mis de nouveau en mode USER.

Voici maintenant un petit programme qui exécute les opérations décrites ci-dessus et qui envoie un bloc de commande au HDC. Remarquez qu'il ne fonctionne parfaitement que si la commande ne fait aucun transfert de données avec le DMA (p.ex. READ, WRITE) car il faudrait programmer le DMA pour cela. Nous reviendrons sur ce point ultérieurement.

```

; ** Accès au Hard-Disk S.D. **
; * Envoie l'octet de commande du champ COM au HDC *

wdc      = $ff8604
wdl      = $ff8606
wdcwdl   = wdc
port     = $fffa01
flock    = $43e

run:
    move.b    #'0',num      ;Préparer message Timeout
    clr.l     -(sp)
    move      #$20,-(sp)
    trap      #1            ;Mettre en mode Super
    addq.l    #6,sp
    move.l    d0,spsave     ;memoriser ancien Stackpointer

    lea       com,a0        ;Pointeur sur bloc de commandes
    bsr       send          ;Envoie bloc de commandes au HDC
    bra       exit          ;fini

send:      ; * Envoyer bloc de commande au HDC *
    st       flock         ;Définir Floppy

```



```

    move    #$B8,wdl        ;Sélectionner HDC A1=0
    clr.l   d0
    moveq   #5,d2           ;compteur: 6 Bytes
loop:
    clr.l   d0
    move.b  (a0)+,d0        ;Récupérer octet
    bsr     send_byte       ;Envoyer octet au HDC
    bmi     error           ;Timeout !
    dbra    d2,loop         ;continuer

cont:
    move    #$8a,wdl
    bsr     waitl           ;Attente d'interruption max. 3s
    bni     error           ;Timeout !
    move    #$8a,wdl
    move    wdc,d0          ;Récupérer Status-Byte
    move    #$80,wdl        ;"Désélectionner" HDC
    move    wdc,d1          ;Récupérer Completion-Byte
    clr     flock           ;Libérer Floppy
    rts
    ;fini

exit:
    move.l  spsave,-(sp)
    move    #$20,-(sp)
    trap    #1              ;Remettre en mode User
    addq.l  #6,sp
    clr     -(sp)           ;fin
    trap    #1

error:
    ;Afficher Erreur
    clr     flock           ;Libérer Floppy
    move.l  #senderr,d0
    bsr     pline           ;Afficher message d'erreur
    bra     exit            ;et fin

send_byte:
    ;# Envoyer un octet au HDC #
    swap    d0              ;Byte dans HI-Wort
    move     #$8a,d0        ;$8A dans LO-Wort
    move.l   d0,wdcwdl      ;Positionner WDC et WDL
    bra     wait            ;Attendre OK (Interruption)

```

```
wait1:
    add.b    #1,num          ;Numéro courant+1
    move.l   #45000,d3       ;Timeout après 3 secondes
    bra      wait1           ;attente...

wait:
    add.b    #1,num          ;Numéro courant+1
    move.l   #15000,d3       ;Timeout après 100 ms

wait1:
    subq.l   #1,d3           ;Compteur Timeout - 1
    bmi      timeout         ;Timeout !
    move.b    port,d0        ;Charger I/O-Port
    and.b     #$20,d0        ;Effacer Bit 5
    bne      wait1           ;Encore positionné, continuer
    moveq     #0,d3          ;Envoyer OK
    rts                          ;fini

timeout:
    moveq     #-1,d3         ;Ne pas envoyer OK
    rts

pline:      ;$ Afficher ligne à l'écran $
    move.l    d0,-(sp)
    move      #9,-(sp)
    trap      #1
    addq.l     #6,sp
    rts

spsave:     dc.l 0
senderr:     dc.b "ERROR avec send_byte "
num:         dc.b "1..fois !",10,13,0
com:         dc.b $b,$0,$0,0,0,$0
even
```

L'octet du bloc de commande transféré ici positionne la tête de lecture/écriture sur la piste 8 (\$B=SEEK). Pour permettre des tests, le programme contient également un affichage d'erreurs qui affiche un timeout à l'écran avec l'indication du moment. Cette partie peu bien sûr être supprimée car elle ne sert qu'au contrôle du transfert correct du bloc de commande.

L'envoi d'instruction de lecture ou d'écriture au HDC est un peu plus compliqué. Il faut en effet programmer le DMA pour le transfert du bloc de commandes. Ce DMA (Direct Memory ACCESS) sert au transfert de données entre la mémoire de l'ordinateur et le disque dur.

Le DMA a besoin des informations suivantes :

- L'adresse de la mémoire dans laquelle ou vers laquelle sont lus ou envoyés les octets de données. Cette adresse est inscrite dans les registres \$FF8609, \$FF860B et \$FF860D dans l'ordre octets LO, MID et HI. Etant donné qu'il manque un octet pour que ce soit une adresse sur 32 bits, on ne peut mettre cette adresse que dans le secteur 0-\$FFFFFF (voir programmation du FDC).
- Le sens de transfert des données, c'est-à-dire lecture ou écriture. Cette information se trouve dans le bit 8 du mot WDL. Un 0 signifie lecture-dans-la et un 1 écriture-à-partir-de-la mémoire.
- L'état indique si le DMA est activé. Le DMA obtient cette information dans le bit 6 du registre WDL \$FF8606. Normalement, le DMA est toujours activé, le bit 6 est donc à 0.

Il est également très important de savoir à quel moment le DMA doit effectuer ces opérations afin qu'il n'y ait pas de collision avec les opérations précédentes. S'il s'agit d'une lecture du disque dur, on envoie d'abord l'octet de commande au HDC puis on définit l'adresse DMA. Ainsi, on évite que le DMA charge des données indésirables en mémoire car le HDC attend les octets suivant du bloc de commande dès qu'il a reçu l'octet de commande.

Pour écrire sur le disque dur, on définit d'abord l'adresse de DMA et on transfère ensuite l'octet de commande. Pour savoir comment cela se fait en pratique, vous pouvez étudier le programme HDC TOOLS du chapitre 5.1.1.3. Mais nous allons continuer pour l'instant avec la théorie et regarder les commandes du HDC.

5.1.1.2 Liste des instructions

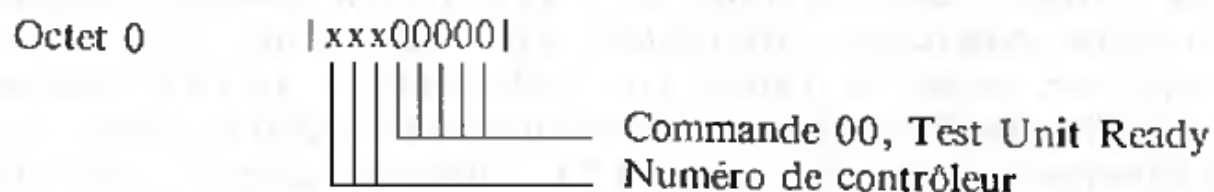
La police d'instruction utilisée par l'ATARI ST ne contient que 9 commandes. Les différents manuels sur le disque dur indiquent parfois d'autres commandes mais en fait, elle ne fonctionnent pas comme indiqué. Voici donc un résumé des commandes qui fonctionnent avec le codeop en hexa.

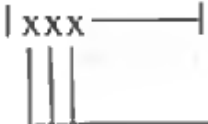
CodeOp	Commande
00	Test Unit Ready
01	Restore
03	Request Sense
04	Format Drive
08	Read
0A	Write
15	Mode Select
1B	Seek To Shipping-Position

Voici une explication des commandes avec leurs octets de paramètre. Le caractère "-" signifie que le bit n'a aucune signification. Ces octets doivent alors être mis à 0.

Test Unit Ready (00)

Cette commande permet à l'ordinateur de contrôler le bus et de déterminer quels sont les périphériques branchés.




Octet 1 :  Numéro de lecteur

Octets 2 à 5 : 

Une fois que le lecteur indiqué est activé et prêt, un 0 est envoyé vers l'octet d'état. Sinon, le check-condition-bit est positionné.

Restore (01)

Cette commande met le HDC à l'état de base et remet la tête de lecture/écriture sur la piste 0.

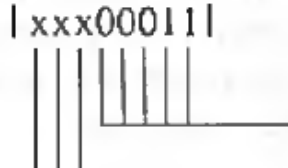
Octet 0 :  Commande 01, Restore
Numéro de contrôleur

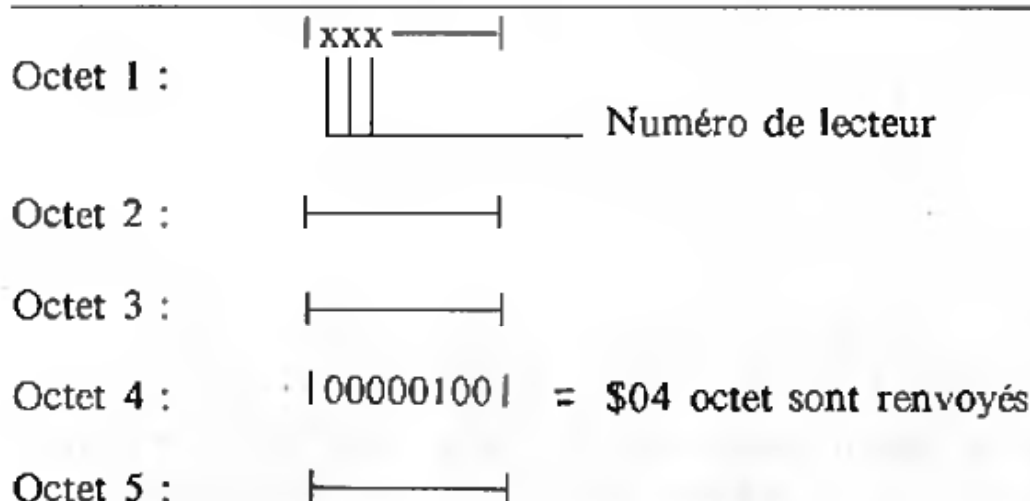
Octet 1 :  Numéro de lecteur

Octets 2 à 5 : 

Request Sense (03)

Cette commande renvoie 4 octets (Lire 4 fois le WDC), et seul le premier octet a une signification. Il contient le code de l'erreur de la dernière commande exécutée. S'il n'y avait aucune commande, il contient un 0.

Octet 0 :  Commande 03, Request Sense
Numéro de contrôleur



Format Drive (04)

Cette commande indique au HDC qu'il faut formater le disque dur en entier. Il n'est pas conseillé d'expérimenter cette instruction pour voir !

On envoie certains paramètres à cette commande :

- le DATA PATTERN FLAG

qui est constitué de 2 octets et qui détermine quelles données doivent être écrites sur les secteurs vides. Si les bits ne sont pas positionnés (0), tous les secteurs sont remplis avec \$6C. Si les bits sont positionnés, c'est le 2 ième octet des octets de commande qui est écrit.

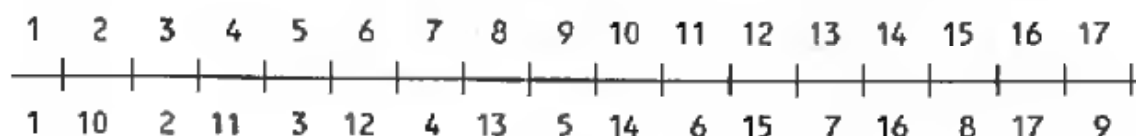
- DATA PATTERN.

Ici se trouve l'octet avec lequel on va remplir les secteurs si le DATA PATTERN FLAG est positionné. Si le flag n'est pas positionné, cet octet n'a aucune signification.

- INTERLEAVE FACTOR.

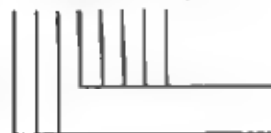
Cette valeur indique l'écart entre deux numéros de secteurs se suivant. Si le facteur est 1, les secteurs sont inscrits à la suite sur les pistes. S'il est de deux par exemple, un autre secteur sera inscrit entre les secteurs 1 et 2. La suite des 17 secteurs d'une piste serait alors la suivante :

Numéro courant :



Ainsi, on a besoin de deux tours du disque pour lire une piste entière. Le processus est alors plus lent mais plus sûr car il y a une petite pause entre chaque lecture de secteur. Normalement ce facteur est à 1.

Octet 0 : |xxx00100|



Commande 04, Format Drive
Numéro de contrôleur

Octet 1 : |xxx—xx—|



Data Pattern Flag
Numéro de lecteur

Octet 2 : |xxxxxxxx| Data Pattern

Octet 3 : |xxxxxxxx| Interleave Factor HI (Devrait être 0)

Octet 4 : |xxxxxxxx| Interleave Factor LO (Normalement 1)

Octet 5 : |————|

Read Sectors (08)

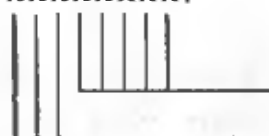
Cette commande indique au contrôleur qu'il faut positionner la tête de lecture/écriture sur la piste du secteur désiré, qu'il faut lire le nombre de secteurs indiqué et les envoyer vers l'ordinateur. De plus, le DMA doit être programmé pour le transfert du bloc de commande afin que les données lues puissent être inscrites dans la place mémoire désirée.

Octet 0 : |xxx01000|



Commande 08, Read Sectors
Numéro de contrôle

Octet 1 : |xxxxxxxx|



Numéro de secteur HI
Numéro de lecteur

Octet 2 : |xxxxxxxx|

Numéro de secteur MID

Octet 3 : |xxxxxxxx|

Numéro de secteur LO

Octet 4 : |xxxxxxxx|

Nombre de secteurs à lire

Octet 5 : |-----|

Write Sectors (0A)

Cette commande sert à écrire dans des secteurs. La tête est positionnée sur la piste considérée, les données en provenance du DMA sont reçues et écrites dans les secteurs. Il faut bien sûr programmer le DMA pour le transfert du bloc de commande.

Octet 0 : |xxx0101|



Commande 0A, Write Sectors
Numéro de contrôle

Octet 1 : |xxxxxxxx|



Numéro de secteur HI
Numéro de lecteur

Octet 2 : |xxxxxxxx|

Numéro de secteur MID

Octet 3 : |xxxxxxxx|

Numéro de secteur LO

Octet 4 : |xxxxxxxx| Nombre des secteurs à écrire

Octet 5 : |————|

Seek (0B)

Cette commande positionne la tête de lecture/écriture du lecteur. Le contrôleur calcule la piste à partir du numéro de secteur indiqué dans la commande et y positionne la tête.

Octet 0 : |xxx01011|



Commande 0B, Seek
Numéro de contrôleur

Octet 1 : |xxxxxxxx|



Numéro de secteur HI
Numéro de lecteur

Octet 2 : |xxxxxxxx|

Numéro de secteur MID

Octet 3 : |xxxxxxxx|

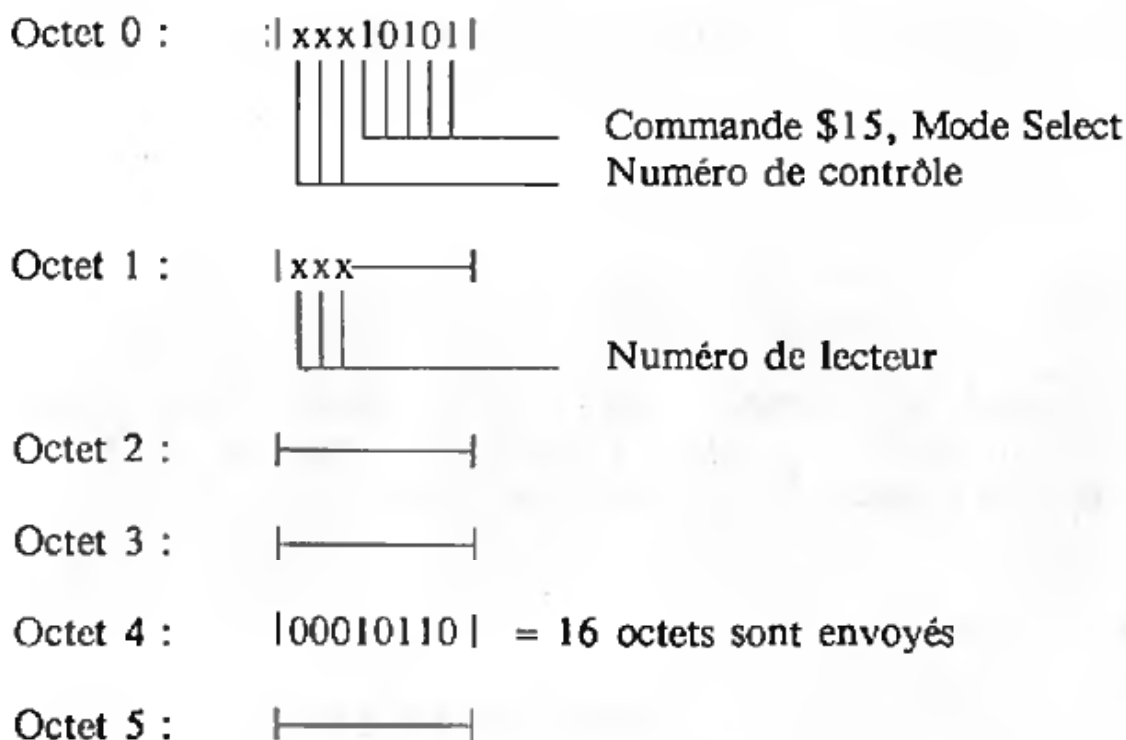
Numéro de secteur LO

Octet 4 : |————|

Octet 5 : |————|

Mode Select (15)

Cette commande sert à définir les paramètres pour le formatage du disque dur. Après la commande, un bloc de 16 octets est envoyé au HDC (avec la programmation du DMA !).

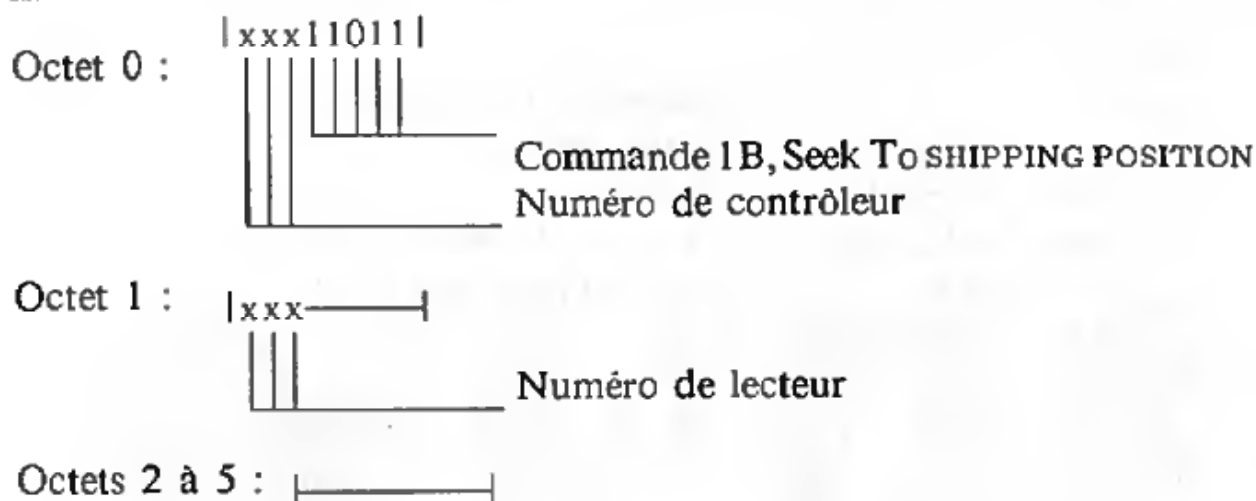


Seek To Shipping Position (1B)

Cette commande positionne la tête de lecture/écriture à un endroit déterminé qui est prévu pour éviter les chocs. On appelle cette position Shipping Position car elle est prévue pour le transport du lecteur.

Le programme SHIP.PRG, qui se trouve sur la disquette livrée avec le disque dur ATARI, met tous les lecteurs branchés à cette position. Il faut l'appeler avant tout transport du disque dur. Faites attention au fait qu'aucune fenêtre de catalogue ne soit affichée durant l'appel du programme. En effet, dans ce cas, le catalogue est de nouveau appelé après l'exécution du programme et la tête de lecture est alors déplacée hors de la position de sécurité.

D'autre part, la première commande ne fonctionne pas correctement après une commande 1B car la tête doit d'abord être sortie de sa position de sécurité.



5.1.1.3 HDC TOOLS

Pour démontrer les accès en lecture et écriture au disque dur, voici un programme qui lit au choix un ou plusieurs secteurs et les inscrit en mémoire ou qui au contraire inscrit des secteurs sur disque à partir de données en mémoire. Dans l'exemple, on charge 8 secteurs à partir du secteur 132. C'est à cet endroit que se trouve le catalogue de la première partition du disque dur.

D'autre part, ce programme intègre le transfert de bloc de commandes, comme dans l'exemple du chapitre 5.1.1.1.

;; LIRE/ECRIRE Harddisk-Sector, envoyer commandes ;;

```
wdc      = $ffB604      ;FDC/HDC-Access, DMA-Sector-Count
wdl      = wdc+2        ;DMA-Mode/Status
dma      = $ffB609      ;DMA-Adress HI
flock    = $43e         ;Floppy-VBL-Flag
port     = $ffa01       ;Parallel-Port, Bit 5=HDC-IRQ
```

run:

```
clr.l    -(sp)
move     #$20,-(sp)
trap     #1             ;Mettre en Supervisor-Mode
addq.l   #6,sp
move.l   d0,spsave      ;Mémoriser User-Stackpointer
```

```

bp1:
; bra put                ; Seulement pour transferts
    pea    buffer        ; Buffer-Adress
    move   #8, -(sp)      ; 8 Secteurs
    move.l #132, -(sp)    ; à partir de secteur 132
    bsr    read           ; Lire secteurs dans buffer
    bra    bp2

put:
    bsr    send           ; Envoyer bloc de commandes
bp2:
    move.l spsave, -(sp)
    move   #20, -(sp)
    trap   #1             ; Mettre en User-Mode
    addq.l #6, sp

    rts                  ; Retour à l'appel
                        ; ou

    clr    -(sp)
    trap   #1             ; Retour au Desktop

send:                ; * Transfert de bloc de commandes *
    lea    wdc, a0
    lea    com, a1        ; Pointeur sur bloc de commandes
    st     flock          ; Mémoriser Floppy
    move   #$BB, wdl      ; Sélectionner HDC A1=0
    clr.l   d0
    moveq   #5, d1
loop:
    clr.l   d0
    move.b  (a1)+, d0
    bsr    send_byte      ; Envoyer octet au HDC
    bmi    tout           ; Timeout !
    dbra   d1, loop       ; Sinon continuer

    bsr    waitl          ; Attendre max. 3 secondes
    bmi    tout           ; Timeout !
    move   wdc, d6
    move   #$B0, wdl      ; sinon
    clr    flock          ; Libérer Floppy
    rts                  ; fini

```

```

read:                ; * Lire secteurs *
    lea    wdc,a0
    st     flock      ;Mémoriser routine Floppy-VBL
    move   #$88,2(a0) ;Accès HDC, A1=0
    nop
    move.l #$08008a,(a0) ;Commande READ

    move.l 10(sp),-(sp) ;buffer-Adress
    bsr    setdma      ;Positionner DMA
    addq.l #4,sp

    bsr    set_parameters ;Nombre et numéros de secteurs
    bmi    tout         ;Timeout !

    move   #$190,2(a0)
    nop
    move   #$90,2(a0)    ;Remettre en READ
    nop
    move   8(sp),(a0)    ;Envoyer Sector-Count au DMA
    nop
    move   #$8a,2(a0)
    nop
    move.l #0,(a0)      ;Commencer transfert
    bsr    waitl        ;Attendre max. 3 secondes
    bmi    tout         ;Timeout !
    move   #$8a,2(a0)
    bra    exec

```

```

write:               ; * Ecrire secteurs *
    lea    wdc,a0
    st     flock      ;Mémoriser Floppy-VBL
    move.l 10(sp),-(sp)
    bsr    setdma      ;Positionner Adresse DMA
    addq.l #4,sp

    move   #$88,2(a0)    ;Accès HDC, A1=0
    nop
    move.l #$0a00Ba,(a0) ;Commande WRITE

```

```

    bsr    set_parameters    ;Nombre et numéros de secteurs
    bmi    tout              ;Timeout !

    move   #$90,2(a0)
    nop
    move   #$190,2(a0)      ;Mettre sur WRITE
    nop
    move   8(sp), (a0)      ;Envoyer Sector-Count au DMA
    nop
    move   #$18a,2(a0)
    nop
    move.l #$100, (a0)      ;Commencer transfert
    bsr    waitl            ;Attendre max. 3 secondes
    bmi    tout            ;Timeout !
    move   #$18a,2(a0)

exec:
    nop
    move.l (a0), d6         ;Recupérer HDC/DMA-Status dans D6
    and.l  #$ff00ff, d6     ;HI=HDC, LO=DMA
tout:
    move   #$80,2(a0)      ;Mettre sur FDC
    nop
    move.l (a0), d7         ;Récupérer Completion-Byte
    and.l  #$ff00ff, d7     ;HI=HDC (0), LO=DMA
    clr    flock           ;Libérer routine Floppy-VBL
    rts                    ;fini

set_parameters:            ;Définir nombre de secteurs et Sector-Count
    move   #$8a,2(a0)
    bsr    wait            ;Attendre que HDC=OK
    bmi    setpx           ;Timeout !

    clr    d0
    move.b 4+5(sp), d0      ;Sectornr. HI
    bsr    send_byte
    bmi    setpx

    move.b 4+6(sp), d0      ;Sectornr. MID
    bsr    send_byte

```

```

bmi    setpx

move.b 4+7(sp),d0      ;Sectornr. LO
bsr    send_byte
bmi    setpx

move    4+8(sp),d0      ;Nombre de secteurs
bsr    send_byte
setpx:
rts                    ;fin

send_byte:              ; * Envoyer 1 octet au HDC *
swap    d0
move    %%B0,d0
move.l  d0,(a0)
bra     wait

waitl:                  ;Attendre OK max. 3 secondes
move.l  #450000,count
bra     waitl

wait:                  ;Attendre OK max. 100 ms
move.l  #15000,count
waitl:
subq.l  #1,count
bmi     timeout
move.b  port,d0
and.b   %%20,d0          ;HDC-Interrupt ?
bne     waitl            ;non
moveq   #0,d0            ;oui => OK
rts

timeout:
move.l  %errline,d0
bsr     pline            ; Afficher 'Timeout'
moveq   #-1,d0           ;Afficher Timeout
rts

setdma:                  ; * Positionner adresse DMA *
move.b  7(sp),dma+4      ;LO
move.b  6(sp),dma+2      ;MID
move.b  5(sp),dma        ;HI

```

```
    rts

pline:                ; * Afficher ligne à l'écran *
    move.l    d0,-(sp)
    move      #9,-(sp)
    trap      #1
    addq.l    #6,sp
    rts

errline: dc.b    "Timeout !",10,13,0
com:      dc.b    $b,0,0,132,0,0
even
count:    dc.l    1                ;Timeout-Counter
spsave:   dc.l    0                ;User-Stackpointer
buffer:   blk.b    512,$FF        ;buffer pour B secteurs
```

Avec ce programme, on est en mesure de charger ou d'écrire directement des secteurs sur disque dur. L'information d'état qui suit est inscrite dans le registre D6 qui peut alors être lu avec un moniteur debugger (SID ou KSEKA).

La différence avec la fonction de lecture/écriture de secteurs du système d'exploitation est qu'on ne peut accéder qu'à la partition désirée du disque dur. Mais si on veut lire le secteur 0 du disque dur, il faut le faire avec le programme ci-dessus.

5.1.1.4 Analyseur de partition

Le secteur 0 dont nous venons de parler est très intéressant car il contient les informations sur les partitions et le disque dur. Pour pouvoir lire et décoder ces informations, voici un petit programme. Il contient entre autres des parties du programme précédent (read) que vous pouvez éventuellement reprendre directement.

Le programme lit le secteur 0 sur disque dur et interprète les données obtenues. Elles sont alors affichées à l'écran et tous les chiffres sont donnés en hexadécimal.


```
;** Partitions-Analysator S.D. **
```

```
wdc = $ff8604      ;FDC/HDC-Access, DMA-Sector-Count
wdl = wdc+2        ;DMA-Mode/Status
dna = $ff8609      ;DMA-Adress HI
flock= $43e        ;Floppy-VBL-Flag
port = $ffa01      ;Parallel-Port, Bit 5=HDC-IRQ
```

```
run:
```

```
    lea    stp,sp
    clr.l  -(sp)
    move   #$20,-(sp)
    trap   #1          ;Mettre en Supervisor-Mode
    addq.l #6,sp
    move.l d0,spsave    ;Sauvegarde du User-Stackpointer

    pea    puf          ;Buffer-Adress
    move   #1,-(sp)      ;1 Secteur
    move.l #0,-(sp)      ;A partir de Secteur 0
    bsr    read          ;Lire les secteurs dans le tampon

    move.l spsave,-(sp)
    move   #$20,-(sp)
    trap   #1          ;Mettre en User-Mode
    addq.l #6,sp

    move.l #head,d0
    bsr    pline        ;Imprimer intitulé

    move.l #hi_cc,d0
    bsr    pmsg
    move   puf+$1b6,d0
    bsr    pword        ;Imprimer nombre-cylindre
    bsr    pcrLf

    move.l #hi_dhc,d0
    bsr    pmsg
    move.b puf+$1b8,d0
    bsr    pbyt        ;Imprimer le nombre de têtes
    bsr    pcrLf
```

```
move.l #hi_lz,d0
bsr     pmsg
move.b puf+$1be,d0
bsr     pbyt           ;Imprimer position d'atterrissage
bsr     pcrLf

move.l #hi_rt,d0
bsr     pmsg
move.b puf+$1bf,d0
bsr     pbyt           ;Imprimer Seek-Rate
bsr     pcrLf

move.l #hi_in,d0
bsr     pmsg
move.b puf+$1c0,d0
bsr     pbyt           ;Imprimer Interleave-Factor
bsr     pcrLf

move.l #hi_spt,d0
bsr     pmsg
move.b puf+$1c1,d0
bsr     pbyt           ;Imprimer sectors/Track
bsr     pcrLf

move.l #hd_size,d0
bsr     pmsg
move.l puf+$1c2,d0
bsr     plong          ;Imprimer secteurs sur Harddisk
bsr     pcrLf

move.l #bsl_count,d0
bsr     pmsg
move.l puf+$1fa,d0
bsr     plong          ;Afficher # secteurs morts
bsr     pcrLf

clr     d5
clr.l   d6
lea     puf+$1c6,a6    ;Champ de Partition 0
```

```

loop:
    bsr    pcr1f
    move.b d5,px_on
    add.b  #'0',px_on
    cmp.b  #0,0(a6,d6)      ;Partition active ?
    bne    pon ;ja
    move.l #' off',px_on+14 ;Sinon afficher 'OFF'
    move.l #px_on,d0
    bsr    pline
    bra    nextp

pon:
    move.l #' on ',px_on+14
    move.l #px_on,d0
    bsr    pline            ;Afficher 'Partition on'
    and.b  #$B0,0(a6,d6)    ;Boot-bar ?
    beq    noboot           ;non
    move.l #boot,d0
    bsr    pline            ;Sinon afficher 'Boot-bar'

noboot:
    move.b 1(a6,d6),px_id+18
    move    2(a6,d6),px_id+19
    move.l  #px_id,d0
    bsr     pline

    move.l  #px_start,d0
    bsr     pmsg
    move.l  4(a6,d6),d0
    bsr     plong          ;Afficher secteur de départ
    bsr     pcr1f

    move.l  #px_size,d0
    bsr     pmsg
    move.l  8(a6,d6),d0
    bsr     plong          ;Afficher Sectors/Track
    bsr     pcr1f

nextp:
    addq    #1,d5
    add     #12,d6
    cmp     #4#12,d6

```

```

blt      loop

move     #1,-(sp)
trap     #1                ;Attente de pression d'une touche
addq     #2,sp
clr      -(sp)
trap     #1                ;Retour au Desktop

read:     ; Lire les secteurs (comme plus haut !)
lea      wdc,a0
st        flock            ;Mémoriser la routine Floppy-VBL
move     #$88,2(a0)        ;Accès au HDC, A1=0
nop
move.l   #$800Ba,(a0)      ;read-Command

move.l   10(sp),-(sp)      ;buffer-Adress
bsr      setdma            ;Positionner DMA
addq.l   #4,sp

bsr      set_parameters    ;Mettre nombre et numéros de secteurs
bmi      tout             ;Timeout !

move     #$190,2(a0)
nop
move     #$90,2(a0)        ;Mettre sur READ
nop
move     8(sp),(a0)        ;Envoyer Sector-Count au DMA
nop
move     #$Ba,2(a0)
nop
move.l   #0,(a0)          ;Commencer le transfert
bsr      waitl
bmi      tout
move     #$Ba,2(a0)
move.l   (a0),d6           ;Récupérer HDC/DMA-Status
and.l    #$ff00ff,d6       ;HI=HDC, LO=DMA

tout:
move     #$80,2(a0)        ;Réactiver FDC
nop
move.l   (a0),d7           ;Récupérer Completion-Byte

```

```

and.l    #$ff00ff,d7      ;HI=FDC, LD=DMA
clr      flock            ;Quitter routine Floppy-VBL
rts                               ;fini

set_parameters:           ;Définir le nombre de secteurs et le Sector-Count
    move    #$8a,2(a0)
    bsr     wait          ;Attendre que HDC-OK
    bni     setpx         ;Timeout !

    clr     d0
    move.b  4+5(sp),d0     ;Sectornr. HI
    bsr     send_byte
    bni     setpx
    move.b  4+6(sp),d0     ;Sectornr. MID
    bsr     send_byte
    bni     setpx
    move.b  4+7(sp),d0     ;Sectornr. LD
    bsr     send_byte
    bni     setpx
    move    4+8(sp),d0     ;Nombre de secteurs
    bsr     send_byte

setpx:
    rts

send_byte:                ;Envoyer 1 octet vers le HDC
    swap    d0
    move    #$8a,d0
    move.l  d0,(a0)
    bra     wait

wait1:                    ;Attendre max. 3 Secondes un OK
    move.l  #450000,count
    bra     wait1

wait:                      ;Attendre max. 100 ms un OK
    move.l  #15000,count

wait1:
    subq.l  #1,count
    bmi     timeout
    move.b  port,d0
    and.b   #$20,d0       ;HDC-Interrupt ?

```

```
    bne    waitl            ;non
    moveq  #0,d0            ;oui => OK
    rts

timeout:
    move.l #errline,d0
    bsr    pline
    moveq  #-1,d0           ;Afficher Timeout
    rts

setdma:    ;Positionner DMA-Adress
    move.b 7(sp),dma+4      ;LO
    move.b 6(sp),dma+2      ;MID
    move.b 5(sp),dma        ;HI
    rts

; ** Autre routines **

pline:      ; Print Line/CR
    bsr    pmsg
pcrlf:      ;Print CR,LF
    move   #10,d0
    bsr    pchar
    move   #13,d0
pchar:      ;Print Character D0
    move   d0,-(sp)
    move   #2,-(sp)
    trap   #1
    addq.l #4,sp
    rts

pmsg:       ;Print Line (D0)
    move.l d0,-(sp)
    move   #9,-(sp)
    trap   #1
    addq   #6,sp
    rts

plong:      ;Afficher D0 en hexa sur 8 chiffres
    moveq  #7,d1
    bra    phexw11
pword:      ;Print Hex-Word D0
```

```

        swap    d0
        moveq   #3,d1
        bra     phexw11
pbyt:      ; Print Hex-Byte D0
        moveq   #1,d1
        ror.l   #8,d0
phexw11:
        rol.l   #4,d0
        move.l   d0,-(sp)
        move.l   d1,-(sp)
        bsr     phexnib
        move.l   (sp)+,d1
        move.l   (sp)+,d0
        dbra    d1,phexw11
        rts
phexnib:
        and.l   #$0f,d0
        add.b    #$30,d0
        cmp.b    #$3a,d0
        bcs     phexn1
        add.b    #7,d0
phexn1:
        bra     pchar      ;Afficher caractère

head:      dc.b  "## Harddisk-Analyse 8/86 S.D. ##",0
hi_cc:     dc.b  "cylindre      :",0
hi_dhc:    dc.b  "Têtes         :",0
hi_lz:     dc.b  "Park-Position :",0
hi_rt:     dc.b  "Seek-Rate     :",0
hi_in:     dc.b  "Interleave    :",0
hi_spt:    dc.b  "Sectors/Track :",0
hd_size:   dc.b  "Total secteurs :",0
bsl_count: dc.b  "Secteurs morts :",0
even
px_on:     dc.b  "1. Partition : ",0
boot:      dc.b  "Boot-bar ",0
px_id:     dc.b  "Partition-ID  : ",0
px_start:  dc.b  "Start-Sector  :",0
px_size:   dc.b  "Nbre de secteurs:",0

```

```
errline:    dc.b "Timeout !",10,13,0
even
data
count:      dc.l 1      ;Timeout-Counter
spsave:     dc.l 0      ;User-Stackpointer
            blk.l 200
stp:        blk.l 1
puf:        blk.b 512   ;Buffer pour un secteur
```

```
10 '***** File-Maker      A.S.  *****
15 '
20 ?:fullw 2:clearw 2:gotoxy 0,0
25 ? "Le fichier >> anapart.tos << est crée":?::??
30 dim c%( 575):cs#=0
35 for i=0 to 575
40 read a$:c%(i)=val("%H"+a$)
45 check#=check#+(c%(i))
50 next i
55 if check#= 6185822.08 then 70
60 ?"Ca ne va pas encore, car quelque chose ne convient pas dans les DATAs
."
65 goto B0
70 bsave "anapart.tos",varptr(c%(0)), 1152
75 ? "Le programme >> anapart.tos << est écrit."
80 ?::?::?"Pressez une touche":a=inp(2):end
85 '
90 '***** DATA's pour anapart.tos *****
95 '
100 DATA 601A,0000,043A,0000,0000,0000,0528,0000
101 DATA 0000,0000,0000,0000,0000,0000,4FF9,0000
102 DATA 075E,42A7,3F3C,0020,4E41,5CBF,23C0,0000
103 DATA 043E,4B79,0000,0762,3F3C,0001,2F3C,0000
104 DATA 0000,6100,019A,2F39,0000,043E,3F3C,0020
105 DATA 4E41,5CBF,203C,0000,0316,6100,0282,203C
106 DATA 0000,0339,6100,028C,3039,0000,0918,6100
```


107 DATA 0292,6100,026C,203C,0000,034C,6100,0274
108 DATA 1039,0000,091A,6100,0280,6100,0254,203C
109 DATA 0000,035F,6100,025C,1039,0000,092C,6100
110 DATA 0268,6100,023C,203C,0000,0372,6100,0244
111 DATA 1039,0000,0921,6100,0250,6100,0224,203C
112 DATA 0000,0385,6100,022C,1039,0000,0922,6100
113 DATA 0238,6100,020C,203C,0000,0398,6100,0214
114 DATA 1039,0000,0923,6100,0220,6100,01F4,203C
115 DATA 0000,03AB,6100,01FC,2039,0000,0924,6100
116 DATA 01FE,6100,01DC,203C,0000,03BE,6100,01E4
117 DATA 2039,0000,095C,6100,01E6,6100,01C4,4245
118 DATA 42B6,4DF9,0000,0928,6100,01B6,13C5,0000
119 DATA 03D2,0639,0030,0000,03D2,0C36,0000,6000
120 DATA 6616,23FC,206F,6666,0000,03E0,203C,0000
121 DATA 03D2,6100,018A,606C,23FC,206F,6E20,0000
122 DATA 03E0,203C,0000,03D2,6100,0174,0236,0080
123 DATA 6000,670A,203C,0000,03E5,6100,0162,13F6
124 DATA 6001,0000,0401,33F6,6002,0000,0402,203C
125 DATA 0000,03EF,6100,0148,203C,0000,0407,6100
126 DATA 0152,2036,6004,6100,0156,6100,0134,203C
127 DATA 0000,041A,6100,013C,2036,6008,6100,0140
128 DATA 6100,011E,5245,DC7C,000C,BC7C,0030,6D00
129 DATA FF5B,3F3C,0001,4E41,544F,4267,4E41,41F9
130 DATA 00FF,8604,50F9,0000,043E,317C,008B,0002
131 DATA 4E71,20BC,000B,008A,2F2F,000A,6100,00C6
132 DATA 588F,6150,6B36,317C,0190,0002,4E71,317C
133 DATA 0090,0002,4E71,30AF,000B,4E71,317C,008A
134 DATA 0002,4E71,20BC,0000,0000,615E,6B0E,317C
135 DATA 008A,0002,2C10,CCBC,00FF,00FF,317C,0080
136 DATA 0002,4E71,2E10,CEBC,00FF,00FF,4279,0000
137 DATA 043E,4E75,317C,008A,0002,613A,6B20,4240
138 DATA 102F,0009,611A,6B16,102F,000A,6112,6B0E
139 DATA 102F,000B,610A,6B06,302F,000C,6102,4E75
140 DATA 4B40,303C,008A,20B0,600C,23FC,0006,DD00
141 DATA 0000,043A,600A,23FC,0000,3A98,0000,043A
142 DATA 53B9,0000,043A,6B10,1039,00FF,FA01,C03C
143 DATA 0020,66EC,7000,4E75,203C,0000,042D,611E
144 DATA 70FF,4E75,13EF,0007,00FF,860D,13EF,0006
145 DATA 00FF,860B,13EF,0005,00FF,8609,4E75,6112
146 DATA 700A,6102,700D,3F00,3F3C,0002,4E41,588F

147 DATA 4E75,2F00,3F3C,0009,4E41,5C4F,4E75,7207
148 DATA 600A,4840,7203,6004,7201,E098,E99B,2F00
149 DATA 2F01,610A,221F,201F,51C9,FFF2,4E75,C0BC
150 DATA 0000,000F,D03C,0030,B03C,003A,6502,5E00
151 DATA 60B4,2A2A,204B,6172,6464,6973,6B2D,416E
152 DATA 616C,7973,6520,2038,2F3B,3620,2053,2E44
153 DATA 2E20,2A2A,0063,796C,696E,6472,6520,2020
154 DATA 2020,2020,203A,2000,548B,7465,7320,2020
155 DATA 2020,2020,2020,2020,3A20,0050,6172,6B2D
156 DATA 506F,7369,7469,6F6E,2020,203A,2000,5365
157 DATA 656B,2D52,6174,6520,2020,2020,2020,3A20
158 DATA 0049,6E74,6572,6C65,6176,6520,2020,2020
159 DATA 203A,2000,5365,6374,6F72,732F,5472,6163
160 DATA 6B20,2020,3A20,0054,6F74,616C,2073,6563
161 DATA 7465,7572,7320,203A,2000,5365,6374,6575
162 DATA 7273,206D,6F72,7473,2020,3A20,0000,312E
163 DATA 2050,6172,7469,7469,6F6E,203A,2020,2020
164 DATA 0042,6F6F,742D,6261,7220,0050,6172,7469
165 DATA 7469,6F6E,2D49,4420,2020,203A,2020,2020
166 DATA 2020,0053,7461,7274,2D53,6563,746F,7220
167 DATA 2020,203A,2000,4E62,7265,2064,6520,7365
168 DATA 6374,6575,7273,3A20,0054,696D,656F,7574
169 DATA 2021,0A0D,0000,0000,0002,1006,140E,0A0A
170 DATA 0E0A,0E0A,0E0A,0E0A,0E0A,0E0A,0E0A,120A
171 DATA 0B12,0610,0612,0C0B,060A,16E0,0C06,1800

Vous avez également le programme en BASIC qui crée ANAPART.TOS sur disquette.

Comme nous l'avons expliqué dans le chapitre sur le boot secteur, `px_flag` indique pour chaque partition (4 au plus) si elle est active et si elle peut être bootée. Le disque dur de l'ATARI ST ne possède au départ aucune partition bootable car le ST ne peut pas booter à partir du disque dur sans driver de disque (AHD1.PRG).

La description Seek-Rate envoie normalement un 2, ce qui signifie 3 millisecondes par pas (Step). L'intervalle peut être de 1 à 16 (secteurs/piste - 1) mais il contient normalement 1. Cela indique l'écart entre deux numéros de secteurs consécutifs, comme sur les disquettes.

La valeur indiquée derrière 'secteurs morts' indique le nombre de secteurs défectueux sur le disque dur complet. Ces secteurs sont reconnus par le programme HDX.PRG et marqués. Un 0 signifie que le disque dur est en état de fonctionnement. De plus, un secteur défectueux par Mégaoctet est quelque chose de tout-à-fait normal.

5.2 BRANCHEMENT DU DISQUE DUR

La prise de 19 broches à l'arrière du ST représente l'interface DMA. Le disque dur est branché à cette prise avec le câble livré (câble très court !), ce qui le connecte directement au travers du DMA à la mémoire du ST. La raison de la taille du câble tient à la vitesse de transfert élevée. Si le câble était plus long, il aurait l'effet d'une antenne, ce qui pourrait détruire complètement les signaux à cause des parasites herziens.

Le transfert de données se fait parallèlement par un câble de 8 conducteurs (broches 1-8), ce qui fait qu'on peut transférer un octet complet à chaque fois.

De plus, cette interface dispose de broches de service comme le RESET (broche 12), ce qui permet de remettre le disque à l'état de base avec un reset du ST. On a également la broche d'interruption (broche 10) qui permet au harddisk de se faire connaître du ST et d'acquiescer les données reçues.

L'ATARI ST et le disque dur sont donc uniquement reliés au travers de ce connecteur. En théorie, on peut brancher jusqu'à 8 disques durs avec 8 lecteurs, mais il faudrait pour cela avoir un connecteur sur le disque dur lui même.

Pour pouvoir communiquer avec le disque dur, l'ordinateur doit envoyer ses désirs sous la forme de blocs de commande au travers du câble. Nous avons déjà décrits ces blocs de commande. Ils sont envoyés par le câble sous la forme de 8 bits, comme les données à écrire ou à lire. Le programme HDCTOOL effectue cela par une simple sélection des registres correspondants et inscription des octets de commande. L'octet est donc disponible par le câble et peut être saisi par le disque dur. Cela est interrompu par la broche interrupt.

Pour autoriser l'échange de données, il faut charger le programme AHDI.PRGM (ATARI HARD DISK INTERFACE). Mais ce programme et le programme HDX ne fonctionnent qu si le TOS est en ROM. Le driver fonctionne bien quelque soit le type du TOS mais pas le programme HDX, ce qui fait qu'il est impossible de travailler avec le disque dur. Avant l'utilisation, le disque dur doit être formaté et partitionné même si vous avez déjà une capacité de 20 mégaoctet avec un disque neuf. Le contrôleur ne peut gérer que 16 mégaoctets par partition !

5.3 IMPRESSION COMPLETE DU CATALOGUE

On utilise souvent des dossiers pour ordonner les nombreux fichiers d'un disque dur. Ces dossiers contiennent parfois également des dossiers et ainsi de suite. Cela améliore sensiblement la clarté mais on ne se souvient parfois plus très bien dans quel dossier ou sous-dossier se trouve tel ou tel fichier.

Pour le retrouver, il faut ouvrir les dossiers afin de voir leur contenu. Il serait beaucoup plus pratique d'imprimer le contenu complet du disque dur. Mais cela n'est pas possible directement avec le système d'exploitation de l'ATARI ST.

Nous allons vous présenter un programme qui exécute ce travail. Après la mise en route, il demande le lecteur concerné (a-f) puis envoie vers l'imprimante tous les fichiers et les dossiers qui se trouvent sur ce disque ou cette disquette. Les contenus des dossiers sont décalés de deux caractères vers la droite à chaque fois si bien que l'imbrication est très claire.

De plus, cela vous permet de savoir si vous avez un programme en double ou en triple sur le disque dur, ce qui prend de la place inutilement.

Voici enfin le programme écrit en langage machine avec l'assembleur SEKA. Si vous utilisez un autre assembleur, il faut commencer les commentaires avec une astérisque au lieu d'un point virgule et remplacer les commandes BLK.X par DS.X.

;; Affichage du catalogue complet 8/86 S.D. ;;

```

run:
    lea    stp,sp
    move.l #menu,d0
    bsr    pmsg
    bsr    getkey           ;Saisie du lecteur
    cmp    #'a',d0
    blt    run              ;mauvais lecteur
    cmp    #'f',d0
    bgt    run              ;mauvais lecteur

    move.b d0,fname
    bsr    pcr1f
    lea    fname+7,a6       ;Pointeur sur fin du nom de fichier + 1
    pea    dta
    move    #$1a,-(sp)
    trap    #1              ;SETDTA
    addq.l #6,sp

    clr    d4               ;Profondeur 0
    lea    profond,a4       ;Pointeur sur compteur()-Array
    move.b #0,(a4)          ;Compteur=0
    bsr    sfirst
    bra    test

sfirst:
    move    #$10,-(sp)
    pea    fname
    move    #$4e,-(sp)
    trap    #1              ;SFIRST
    addq.l #8,sp

sea:
    cmp.b  #'.' ,dta+30     ;Subdir ?
    bne    seax
    bsr    snext1
    tst    d0
    bne    seax
    bra    sea

seax:

```

```

    rts

snext:
    add.b    #1,(a4,d4)
snext1:
    move     #$4f,-(sp)
    trap     #1                ;SNEXT
    addq.l   #2,sp
    rts

next:
    bsr      snext
test:
    tst      d0
    bne      up                ;Un degré plus haut
    cmp.b    #$10,dta+21      ;Subdirectory ?
    bne      output           ;non : afficher la donnée
    bra      down

up:
    subq     #1,d4             ;Profondeur - 1
    bmi      fini              ;Fin !
    sub      #6,a6
mlop:
    cmp.b    #'\'',-(a6)
    bne      mlop
    bsr      addwc              ;Ajouter ".!",0
    bsr      sfirst
    clr      d7
    move.b   (a4,d4),d7        ;Compteur (profondeur) dans D7
    addq     #1,d7             ;Compteur+1
    move.b   #0,(a4,d4)
selop:
    subq     #1,d7
    beq      next              ;Fin avec ce niveau
    bsr      snext             ;Recherche de la donnée n°Compteur (profondeur
)
    bra      selop

down:

```

```

    move.l #sub,a5
    bsr    impline
    move.l #dta+30,a5
    bsr    impline
    bsr    imprcr      ;Imprimer CR
    addq   #1,d4       ;Profondeur+1
    move.b #0,{a4,d4}
    subq.l #4,a6
    move   #13,d7
    lea    dta+30,a3
flop:
    move.b (a3)+,d0
    beq    flop%
    move.b d0,{a6}+    ;Ajouter le nom de fichier au Path
    dbra   d7,flop
flop%:
    bsr    addwc       ; Ajouter "\t.1",0
bp:
    bsr    sfirst
    bra    test        ;Recherche du niveau suivant

addwc:
    move.b #'\\',{a6}+
    move.b #'1',{a6}+
    move.b #'.',{a6}+
    move.b #'1',{a6}+
    move.b #0,{a6}+
    rts

output:      ;Afficher donnée
    cmp.b  #8,$e1b    ;Touche Alternate pressée ?
    bne    out1        ;non
    bra    fini        ;sinon fin
out1:
    lea    dta+30,a0
    lea    outln,a5     ;Ligne d'impression
    move   d4,d5
blop:
    move   #' ',(a5)+
    dbra   d5,blop

```



```

blcp1:
    move.b (a0)+,d0
    beq    blcp1x
    move.b d0,(a5)+
    bra    blcp1
blcp1x:
    move.b #' ',(a5)+
    cmp.l  #outln+26,a5
    blt    blcp1x
    move.l dta+26,d0
    bsr    pdez8
    move.b #0,(a5)
    move.l #outln,a5
    bsr    impline
    bsr    imprcr
    bra    next

fini:      ;c'est fini
    clr    -(sp)
    trap   #1                ;Exit => Desktop

menu:      dc.b  "## Affichage du directory S.D. ##",10,13
           dc.b  "Indiquez le lecteur (a-f) :",0
sub:        dc.b  "Sub-Directory : ",0
fname:      dc.b  "a:\\$.#",0,"
even

; ## Sous programmes ##

getkey:     ;Get Key -> D0
    move    #1,-(sp)
    trap    #1
    and.l   #$ff,d0
    addq.l  #2,sp
    rts

pline:      ; Print Line/CR
    bsr     pmsg
pcrlf:      ;Print CR,LF
    move    #10,d0

```

```
    bsr    pchar
    move   #13,d0
pchar:    ;Print Character D0
    move   d0,-(sp)
    move   #2,-(sp)
    trap   #1
    addq.l #4,sp
    rts

impline:    ;Imprimer la ligne à partir de (a5)
    move.b (a5)+,d0
    beq    impx
    bsr    impchr    ;Imprimer caractère
    bra    impline
impx:
    rts

imprcr:    ; Imprimer CR/LF
    move   #10,d0
    bsr    impchr
    move   #13,d0
impchr:
    move   d0,-(sp)
    move   #5,-(sp)
    trap   #1 ;Imprimer caractère
    addq.l #4,sp
    rts

pmsg:      ;Print Line (D0)
    move.l d0,-(sp)
    move   #9,-(sp)
    trap   #1
    addq   #6,sp
    rts

pdez8:     ;Imprimer D0 en décimal 8 chiffres
    divu   #10000,d0
    swap   d0
    move   d0,-(sp) ;Reste
    swap   d0
```

```

    and.l    #$ffff,d0
    move.l   #1000,d1
    bsr      dez1
    move     (sp)+,d0
pdez4:      ;Imprimer D0 en décimal avec 4 chiffres
    move.l   #1000,d1
dez1:
    divu     d1,d0
    move.l   d0,-(sp)
    add      #'0',d0
    move.b   d0,(a5)+ ;Caractère dans la ligne d'affichage
    move.l   (sp)+,d0
    swap     d0
    and.l    #$ffff,d0
    divu     #10,d1
    bne      dez1
    rts

```

```

data
dta:   blk.b 44
temp:  blk.l 0
profond: blk.b 10
outln: blk.b 80
       blk.l 200
stp:   blk.l 1

```

```

10 '***** File-Maker      A.S. *****
15 '
20 ?:fullw 2:clearw 2:gotoxy 0,0
25 ? "Le fichier >> alldir.tos << est crée":?::?
30 dim c%( 334):cs#=0
35 for i=0 to 334

```

```
40 read a$:c%(i)=val("&H"+a$)
45 check#=check#+(c%(i))
50 next i
55 if check#= 3715982.08 then 70
60 ?"Ca ne va pas encore, car quelque chose ne convient pas dans les DATAs
."
65 goto 80
70 bsave "alldir.tos",varptr(c%(0)), 669
75 ? "Le programme >> alldir.tos << est écrit."
80 ?"?:?:?:?"Pressez une touche":a=inp(2):end
85 '
90 '***** DATA's pour alldir.tos *****
95 '
100 DATA 601A,0000,026C,0000,0000,0000,03AA,0000
101 DATA 0000,0060,0000,0000,0000,0000,4FF9,0000
102 DATA 0612,203C,0000,016A,6100,0216,6100,01D2
103 DATA B03C,0061,6DE6,B03C,0066,6EE0,13C0,0000
104 DATA 01BB,6100,01CE,4DF9,0000,01C2,4B79,0000
105 DATA 026C,3F3C,001A,4E41,5C8F,4244,49F9,0000
106 DATA 029B,18BC,0000,6102,6036,3F3C,0010,4B79
107 DATA 0000,01BB,3F3C,004E,4E41,508F,0C39,002E
108 DATA 0000,028A,6608,610C,4A40,6602,60EE,4E75
109 DATA 5234,4000,3F3C,004F,4E41,548F,4E75,61F0
110 DATA 4A40,660E,0C39,0010,0000,0281,6600,007E
111 DATA 602B,5344,6B00,00DC,5D4E,0C26,005C,66FA
112 DATA 6154,61A6,4247,1E34,4000,5247,19BC,0000
113 DATA 4000,5347,67C8,61B8,60FB,2A7C,0000,01AA
114 DATA 6100,0142,2A7C,0000,02BA,6100,0138,6100
115 DATA 013E,5244,19BC,0000,4000,598E,7E0D,47F9
116 DATA 0000,02BA,101B,6706,1CC0,51CF,FFF8,6106
117 DATA 6100,FF5B,608A,1CFC,005C,1CFC,002A,1CFC
118 DATA 002E,1CFC,002A,1CFC,0000,4E75,0C39,0008
119 DATA 0000,0E1B,6602,604A,41F9,0000,028A,4BF9
120 DATA 0000,02A2,3A04,3AFC,2020,51CD,FFFA,101B
121 DATA 6704,1AC0,60FB,1AFC,0020,BBFC,0000,02BC
122 DATA 6DF4,2039,0000,02B6,6100,00E2,1ABC,0000
123 DATA 2A7C,0000,02A2,6100,00AC,6100,00B2,6000
124 DATA FF1E,4267,4E41,2A2A,2041,6666,6963,6861
125 DATA 6765,2064,7520,6469,7265,6374,6F72,7920
126 DATA 2053,2E44,2E20,2A2A,0A0D,496E,6469,7175
```

127 DATA 657A,206C,6520,6C65,6374,6575,7220,2861
128 DATA 2D66,2920,3A00,5375,622D,4469,7265,6374
129 DATA 6F72,7920,3A20,0061,3A5C,2A2E,2A00,2020
130 DATA 2020,2020,2020,2020,2020,2020,2020,2020
131 DATA 2020,2020,2020,2020,2020,2020,2020,0000
132 DATA 3F3C,0001,4E41,C0BC,0000,00FF,54BF,4E75
133 DATA 612E,700A,6102,700D,3F00,3F3C,0002,4E41
134 DATA 5B8F,4E75,101D,6704,610A,60FB,4E75,700A
135 DATA 6102,700D,3F00,3F3C,0005,4E41,5B8F,4E75
136 DATA 2F00,3F3C,0009,4E41,5C4F,4E75,80FC,2710
137 DATA 4B40,3F00,4B40,C0BC,0000,FFFF,223C,0000
138 DATA 03EB,610B,301F,223C,0000,03EB,80C1,2F00
139 DATA D07C,3000,1AC0,201F,4B40,C0BC,0000,FFFF
140 DATA 82FC,000A,66E6,4E75,0000,0002,061A,0A06
141 DATA 1012,1028,340A,1A3A,061C,0B0E,0000

AVERTISSEMENT

Les programmes RAM-DISK et DISK-TO-RAM-DISK présentés dans le chapitre 6 sont exclusivement réservés aux systèmes NON équipés de disque dur.

Ne les utilisez en aucun cas si vous possédez un disque dur.

6. Le RAM DISK

Le troisième type de mémoire de masse pour l'ATARI ST est le RAM DISK. Un tel lecteur de disquette virtuel qui réside en mémoire représente une possibilité intéressante et surtout rapide de mémorisation des données. Comment cela fonctionne t-il ?

Tout d'abord il faut une place mémoire qui n'est utilisée par aucune autre application de l'ordinateur. Nous y inscrirons les données au lieu de les mettre sur une disquette. L'avantage est évident : le 68000 peut très facilement inscrire et lire des données en mémoire et le fait avec une vitesse énorme. De plus, tous les phénomènes mécaniques qui ralentissent un lecteur normal (tête, mise en route du moteur, etc...) sont évités. Le résultat : un RAM DISK est très rapide.

Ce dont nous avons encore besoin c'est d'un programme. Ce programme doit prendre en charge la gestion de la mémoire RAM DISK et déplacer à volonté les données en mémoire. Il existe déjà de nombreux programmes de ce type sur le marché et dans les livres (*P. ex. Trucs et Astuces chez MICRO APPLICATION.*). Ils travaillent tous selon le même principe que nous allons étudier maintenant.

En premier lieu, il faut initialiser la place mémoire qui servira au RAM DISK. On crée un boot secteur qui contiendra toutes les informations sur le type, la structure et la taille du "disque". Sur les disquettes normales, ce secteur est toujours en tout début. Nous allons donc le placer au début du disque virtuel.

Ensuite, il faut que le programme s'installe, c'est-à-dire qu'il doit effectuer certaines préparations lui permettant de savoir à tout moment si un transfert de données est en cours et si oui, où et dans quelle direction. On réalise cela en positionnant trois pointeurs du système d'exploitation sur ses propres routines. Ces pointeurs sont des places mémoire dans lesquelles se trouvent des adresses de programmes. Si le système d'exploitation veut appeler un tel programme, le pointeur correspondant est lu et renvoie à la routine correspondante.

Les pointeurs qui sont utilisés pour l'installation d'un RAM DISK sont prévus pour le disque dur.

Ils se trouvent aux adresses \$472 à \$47E et pointent sur des routines qui ont les significations suivantes :

Adresse	Nom	Signification
\$472	hdv_bpb	Définition et envoi du bloc de paramètre qui contient des données sur la disquette ou le disque dur.
\$476	hdv_rw	Routine de lecture/écriture pour le disque dur. Le transfert de données est effectué à cet endroit.
\$47A	hdv_boot	Routine de boot pour le disque dur. Elle n'est pas utilisée par le ram disk car il n'est pas possible de booter à partir d'un ram disk.
\$47E	hdv_mediach	Détermine si le support de masse (disquette) a été changé.

Une fois que les pointeurs ont été repositionnés et que leur contenu a été mémorisé, le programme peut être mis en route. Pour cela, on appelle une adresse particulière de la BIOS qui permet de réserver une place mémoire donnée. Ainsi, le RAM DISK est installé.

Il faut alors préparer une icône de lecteur pour le desktop. Pour cela, il suffit de cliquer sur une des icônes existantes, sélectionner l'option 'définir des préférences' et modifier la lettre du lecteur de disquette. Dès qu'on a cliqué sur "confirmer", le nouveau symbole de disquette est affiché à l'écran. Il permet dès lors de charger et de sauvegarder des données en cliquant dessus comme d'habitude. Cependant, les fonctions de formatage et de copie de disquette en entier ne fonctionnent pas et on ne peut que traiter des fichiers un à un.

Si le système d'exploitation veut avoir accès au disque dur ou au ram disk, il se branche sur le programme de ram disk au travers des pointeurs dont nous avons parlé plus haut. Le programme teste alors s'il s'agit d'un adressage du ram disk. Sinon, il repositionne les pointeurs dont il avait mémorisé les valeurs initiales.

Si le disque virtuel était concerné, le programme commence son travail. Dans un accès en lecture/écriture, les paramètres comme le secteur, le nombre de secteurs et la direction de transfert de données sont lus sur la pile et les données correspondantes sont copiées en mémoire.

S'il s'agit d'un test media-change qui teste un changement éventuel de mémoire de masse, le programme de ram disk renvoie un 0. Cela signifie que rien n'a été changé, cela n'est bien sûr pas possible avec un ram disk.

Le troisième type d'appel signifie que le système d'exploitation désire connaître l'adresse mémoire du bloc de paramètres. L'adresse désirée est renvoyée dans D0.

C'étaient les tâches du disque virtuel. Par contre, il ne peut pas conserver les données en mémoire après extinction de l'ordinateur. C'est le problème : les données ne sont pas réellement sauvegardées. Pour cette raison, il ne faut pas omettre de sauvegarder les données sur disquette avant d'éteindre l'ordinateur.

C'est fini pour la théorie. Nous allons maintenant étudier un programme de disque virtuel qui effectue tout cela.

6.1 UN PROGRAMME DE RAM DISK TRES PRATIQUE

Le programme présenté dans ce chapitre contient quelques possibilités qui ne sont indispensables pour l'utilisation de base d'un ram disk. Mais comme elles sont très pratiques, le programme est un peu plus long mais plus confortable. Il est prévu pour être utilisé en tant que disque C mais il peut facilement être adapté à un autre numéro de lecteur.

Le programme est un accessoire qui est mis dans le menu "bureau" sous la dénomination "ram disk". Si on sélectionne cette option, une petite fenêtre de dialogue vous offre la possibilité de définir certains paramètres.

Le premier paramètre qui est encadré en gras offre l'option "exit". Lorsqu'on sélectionne cette option ou qu'on presse la touche RETURN, on revient au desktop. Rien ne s'est passé.

Cette option est prévue pour les cas où on a choisi l'option RAM DISK par erreur.

Au milieu de la zone de dialogue se trouve l'option "plus". Lorsqu'on clique sur ce bouton, le numéro indiqué dans la petite boîte à droite est modifié. Il s'agit de la taille du ram disk. Lorsqu'on clique sur "plus", cette valeur est augmentée 100 par 100 et revient à 0 une fois qu'on a atteint 800.

Une fois qu'on a défini la capacité du ram disk, on sélectionne l'option contenant le nombre. Etant donné que le contenu du ram disk va être effacé pour installer une nouvelle place mémoire, une fenêtre de dialogue s'affiche pour confirmation. Il suffit de répondre par "oui" à la question qu'elle pose. Sinon, le ram disk n'est pas modifié.

Une fois qu'on a sélectionné "oui", le programme fait son travail et installe les modifications. D'abord, il renvoie la place mémoire utilisée par le premier ram disk au système d'exploitation.

Ensuite, le programme essaie de réserver la place mémoire désirée. S'il n'y a pas suffisamment de place mémoire, on obtient le message "ram insuffisante". Lorsqu'on quitte ce message, le ram disk a disparu. Il faut choisir une nouvelle taille mémoire plus petite en sélectionnant de nouveau l'option ram disk et en cliquant sur l'option "plus".

Si on choisit 0 comme taille, le ram disk disparaît et on a libéré toute la mémoire. Ainsi, on est en mesure de définir et de modifier le ram disk à volonté. La plupart des ram disk sur le marché n'ont pas cette possibilité. Vous vous rendrez compte des avantages lorsque vous aurez utilisé intensivement le ram disk.

Encore un point avant de voir le programme : étant donné qu'il n'est pas possible de formater le ram disk (n'essayez pas car cela peut se faire sur les disquettes !), il faut effacer tous les fichiers si on veut vider le ram disk. Dans le programme que nous vous proposons, il suffit de sélectionner la même capacité et le ram disque sera automatiquement vidé.

Le programme :

```
;**** RAM-Disk avec confort S.D. ****
```

```
hdv_bpb      = $472
```

```
hdv_rw       = $476
```

```
hdv_mediach  = $47e
```

```
drvbits      = $4c2
```

```
start:
```

```
    move.l #nstapel,a7 ;Définir nouvelle pile
```

```
    move    #10,opcode    ;appl_init
```

```
    move    #0,sintin
```

```
    move    #1,sintout
```

```
    move    #0,saddrin
```

```
    move    #0,saddrout
```

```
    bsr     aes
```

```
    move    intout,appid ;Application-ID
```

```
    move    #77,opcode    ;graf_handle
```

```
    move    #5,sintout
```

```
    move    #0,saddrin
```

```
    move    #0,saddrout
```

```
    bsr     aes
```

```
    move    intout,grhandle ;Graphic-Handle
```

```
    move    #35,opcode    ;Menu_Register
```

```
    move    #1,sintin
```

```
    move    #1,sintout
```

```
    move    #1,saddrin
```

```
    move    appid,intin
```

```
    move.l #accname,addrin
```

```
    bsr     aes
```

```
    move    intout,accid ;Accessory-Number
```

```
;** A partir d'ici se trouve la boucle de préparation **
```

```
loop: bsr     event        ;Event_Multi
```

```
    cmp     #40,msgbuff    ;Acc_open ?
```

```
    bne     loop          ;non
```

```

move    msgbuff+8,d0
cmp     accid,d0      ;Notre Accessory-Number ?
bne     loop          ;non
bsr     run           ;Représenter menu
bra     loop          ;Recommencer...

```

```

; ** Choix **

```

```

run:

```

```

move.l  #combien,addrin
bsr     formalert     ;Représenter choix
move    intout,choice
cmp     #1,choice     ;Exit?
beq     end           ;oui => fin
cmp     #3,choice     ;OK ?
beq     ok            ;oui
addq    #2,size       ;Proposer autre taille
cmp     #10,size      ;Supérieur à 800 KByte?
blt     more          ;non
clr     size          ;oui, retour à 0 KByte

```

```

more:

```

```

lea     sizes,a0
clr.l   d0
move    size,d0
move    0(a0,d0),capaci ;Installer nouvelle taille
lsl     #1,d0
lea     dezi,a0
move.l  0(a0,d0),offer ;Afficher nouvelle taille
bra     run           ;Recommencer

```

```

; * Réserver mémoire *

```

```

ok:

```

```

move.l  #clear,addrin
bsr     formalert     ;Vraiment effacer?
cmp     #2,intout
beq     okx           ;non => Fin
bsr     mfree         ;Libérer mémoire
tst     size          ;0 KByte ?
bne     ok1           ;non

```

```

okk:
    rts                ;0 Kbyte: fini
okl:
    move #2,changed    ;'Changement de disquette'
    clr.l d7
    move capaci,d7      ;Capacité en KByte
    add.l #9,d7         ;plus 9K pour la gestion
    asl.l #5,d7
    asl.l #5,d7         ;fois 1024: capacité en octets
    move.l d7,-(sp)     ;Secteur RAM à utiliser
    move #48,-(sp)      ;MALLOC-Fonction
    trap #1
    addq.l #6,sp
    tst.l d0            ;Erreur ?
    beq erreur         ;oui => Message d'erreur
    move.l d0,buffer    ;Mémoriser Startadress du RAM-Disk

    move.l #init,-(sp)
    move #38,-(sp)      ;Initialisation en Supervisor
    trap #14
    addq.l #6,sp
    rts

erreur:
    move.l #error,addrin
    bsr formalert      ;'RAM insuffisante !'
    bra end           ;Quitter

init:
    move.l hdv_bpb,bpb_save ;Mémoriser anciens vecteurs
    move.l #bpb,hdv_bpb
    move.l hdv_rw,rw_save ;Positionner vecteurs sur nouvelles routines
    move.l #rw,hdv_rw
    move.l hdv_mediach,media_save
    move.l #media,hdv_mediach

    move.l buffer,a0
    move.l #10240/4,d0

iloop1:
    clr.l (a0)+        ;Effacer Boot-Sector et FATs

```

```

    dbra    d0,iloop1

; * Génération de Boot-Sector *
    move.l  buffer,a0
    add.l   #11,a0      ;ab Puffer+11
    lea     boottab,a1
    move.l  #tabend,d0
    sub.l   #boottab,d0
    sub.l   #1,d0
bloop:
    move.b  (a1)+,(a0)+ ;Copier données dans
    dbra    d0,bloop

    move     capaci,d7
    move     d7,numcl    ;Capacité en Koctets dans le BPB

    lsl     #1,d7        ;capacité en secteurs
    add     #18,d7        ;plus 18 Secteurs
    move.l  buffer,a0
    add.l   #19,a0        ;Dans buffer+19 et +20
    move.b  d7,(a0)+      ;LD
    lsr     #8,d7
    move.b  d7,(a0)       ;HI

    bset    #2,drvbits+3 ;Afficher Drive C
    rts

; * Fonction: Get BPB *

bpb:  cmp    #2,4(sp)     ;Drive C ?
      beq    bpb1         ;oui
      move.l  bpbsave,a0   ;ancienne Routine
      jmp     (a0)         ;Appeler

bpb1: move.l  #bpbtav,d0   ;Pointeur sur BIOS-Parameter-Bloc
      rts

; * Fonction: Read/Write *

```

```

rw:   cmp     #2,14(sp)    ;Drive C ?
      beq     rw1          ;oui
      move.l  rwsave,a0    ;ancienne Routine
      jmp     (a0)         ;appeler

rw1:  move    12(sp),d0     ;recno, Numéro de secteur logique
      ext.l   d0
      lsl.l   #8,d0
      lsl.l   #1,d0        ;fois 512

      move.l  6(sp),a0      ;buffer-Adress
      move    10(sp),d1     ;Nombre de secteurs
      subq    #1,d1
      move.l  buffer,a1     ;Basis-Adress
      add.l   d0,a1         ;plus adresse relative dans RAM-Disk

      move    4(sp),d0      ;rwflag
      btst    #0,d0         ;lecture ?
      beq     rloop0        ;oui
      exg     a0,a1         ;Echanger objet et source

rloop0: move.l #511,d0      ;un secteur
rloop: move.b (a1)+,(a0)+   ;Copier buffer
      dbra    d0,rloop
      dbra    d1,rloop0     ;Secteur suivant
      clr     d0            ;OK
      rts

```

;* Fonction: Media-Change *

```

media: cmp     #2,4(sp)    ;Drive C ?
      beq     media1       ;oui
      move.l  mediasave,a0 ;ancienne routine
      jmp     (a0)         ;appeler

media1: move    changed,d0  ;Eventuellement disquette changée
      clr     changed      ;mais seulement une fois
      rts

```

event:

```

move    #25,opcode    ;Event_Multi, Déterminer événement GEM
move    #16,sintin
move    #7,sintout
move    #1,saddrin
move.l  #msgbuff,addrin
lea     table,a1
lea     intin,a2
moveq   #15,d0
lop1:
move    (a1)+,(a2)+    ;Définir parametre
dbra    d0,lop1
bsr     aes
rts

aes:
; Appel AES
move.l  #aespb,d1
move    #3c8,d0
trap    #2
rts

mfree:
; Libérer mémoire
tst.l   buffer
beq     end             ;est déjà fait

move.l  #reinit,-(sp)
move    #38,-(sp)      ;Reinitialisation
trap    #14            ;en mode Superviseur
addq.l  #6,sp

move.l  buffer,-(sp)
move    #49,-(sp)      ;MFREE-Fonction, libérer mémoire
trap    #1
addq.l  #6,sp
tst.l   d0              ;Error?
beq     end             ;non
move.l  #error1,addrin
bsr     formalert       ;Message d'erreur
end:
clr.l   buffer          ;Plus de mémoire réservée
rts

```


reinit:

```

move.l bpb save, hdy_bpb
move.l rwsave, hdy_rw ;Positionner vecteurs sur anciennes routines
move.l mediasave, hdy_mediasave
bclr #2, drvbits+3 ;Accès au Drive C
rts

```

formalert:

```

move #52, contrl ;form_alert, Représenter zone d'alarme
move #1, contrl+2
move #1, contrl+4
move #1, contrl+6
move #0, contrl+8
move #1, intin
bsr aes
rts

```

table: dc.w \$13,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0

accname: dc.b " RAM-Disk ",0

even

combien: dc.b "[1][Taille du RAM-Disk en Koctets ?]"

dc.b "[Exit! plus !]"

offer: dc.b " 100]",0,0

clear: dc.b "[1][Effacer ancien contenu du RAM-Disk ?]"

dc.b "[oui ! | Non]",0,0

error: dc.b "[2][RAM insuffisante !]"

dc.b "[Alors non..]",0,0

error1: dc.b "[2][Erreur avec MFREE !]"

dc.b "[Alors non..]",0,0

even

capaci: dc.w 100

size: dc.w 2

sizes: dc.w 0,100,200,300,400,500,600,700,800

dezi: dc.b ' 0 100 200 300 400 500 600 700 800'

buffer: dc.l 0 ;Adresse du tampon RAM-Disk

changed: dc.w 0 ;Flag pour 'disquette changée'

bphtab:

```

recsiz:  dc.w $200      ;Taille du secteur
clsiz:   dc.w 2         ;Taille du cluster en secteurs
clsizb:  dc.w $400      ;Taille du cluster en octets
rdlen:   dc.w 7         ;Taille du catalogue en secteurs
fsiz:    dc.w 5         ;Taille de la FAT
fatrec:  dc.w 6         ;Secteurs de la FAT
datrec:  dc.w 18        ;Secteurs pour la gestion
numcl:   blk.w 1        ;Capacité en Koctets
flags:   dc.l 0,0,0,0

```

```

boottab: ; Données en format B086
         dc.b 0,2       ;Octets par secteur
         dc.b 2         ;Secteurs par cluster
         dc.b 1,0       ;reserved Sectors
         dc.b 2         ;FATs
         dc.b 112,0     ;Directory entries
         blk.b 2        ;Sectors on media
         dc.b 0         ;media descriptor
         dc.b 5,0       ;Sectors pro FAT
         dc.b 9,0       ;Sectors pro Track
         dc.b 1,0       ;Sides
         dc.b 0         ;hidden
tabend:  dc.b 0

```

even

```

bpbsave: blk.l 1        ;Place pour anciens vecteurs
rwsave:  blk.l 1
mediasave: blk.l 1

```

```

aespb:   dc.l contrl,global,intin,intout,addrin,addrout

```

data

```

choice:  blk.w 1        ;Choix
grhandle: blk.w 1
appid:   blk.w 1        ;Application-ID
accid:   blk.w 1        ;Accessory-Unit
msgbuff: blk.b 16
         blk.l 128      ;nouvelle pile

```

nstapel: blk.l 1

contrl: ;GEM-Parameter-Bloc

opcode: blk.w 1

sintin: blk.w 1

sintout: blk.w 1

saddrin: blk.w 1

saddrout: blk.l 1

blk.w 5

global: blk.l 8

intin: blk.w 80

ptsin: blk.w 80

intout: blk.w 80

ptsout: blk.w 80

addrin: blk.w 80

addrout: blk.w 80

Ce programme a été créé avec le macro assembleur SEKA et certains points diffèrent de l'assembleur DRI qui est inclus dans le système de développement ATARI. Pour le modifier, il faut changer les lignes de commentaires qui doivent commencer par un "*" et remplacer les commandes "blk" par "ds".

Le programme est divisé en plusieurs parties :

1. Appel de l'accessoire.
2. Boucle d'attente qui tourne constamment en fond lors du fonctionnement normal de l'ATARI ST et qui ne doit donc pas avoir de fin.
3. Fenêtre de dialogue pour sélectionner la capacité.
4. Test de confirmation.
5. Renvoi de la mémoire utilisée précédemment (MFREE).
6. Réservation de la nouvelle mémoire ou affichage d'un message d'erreur.
7. Mémorisation des vecteurs BIOS pour les routines de disk et installation de nouveaux vecteurs.
8. Fonction GET BPB
9. Fonction READ/WRITE
10. Fonction MEDIA CHANGE
11. Champ de données pour les blocs de paramètres.

Les points 7 à 10 ont déjà été traités dans le chapitre précédent. Nous ne décrirons pas les fonctions 1 à 6 car elles sont très complexes. Vous trouverez leur description dans les livres "La bible de l'ATARI ST" et "Le livre de GEM" chez MICRO APPLICATION.

Voici le programme en BASIC qui crée un programme RAMDISK.ACC sur disquette :

```

10 '***** File-Maker      A.S.  *****
15 '
20 ? :fullw 2:clearw 2:gotoxy 0,0
25 ? "Le fichier >> ramdisk.acc << est crée":?::?
30 dim c%( 741):cs#=0
35 for i=0 to 741
40 read a$:c%(i)=val("&H"+a$)
45 check#=check#+(c%(i))
50 next i
55 if check#= 524757b then 70
60 ? "Ca ne va pas encore, car quelque chose ne convient pas dans les DATAs
."
65 goto B0
70 bsave "ramdisk.acc",varptr(c%(0)), 1483
75 ? "Le programme >> ramdisk.acc << est écrit."
80 ?::?::?"Pressez une touche":a=inp(2):end
85 '
90 '***** DATA's pour ramdisk.acc *****
95 '
100 DATA 601A,0000,0544,0000,0000,0000,0612,0000
101 DATA 0000,0000,0000,0000,0000,0000,2E7C,0000
102 DATA 075C,33FC,000A,0000,0760,33FC,0000,0000
103 DATA 0762,33FC,0001,0000,0764,33FC,0000,0000
104 DATA 0766,33FC,0000,0000,0768,6100,02E4,33F9
105 DATA 0000,08D6,0000,0548,33FC,004D,0000,0760
106 DATA 33FC,0005,0000,0764,33FC,0000,0000,0766
107 DATA 33FC,0000,0000,0768,6100,02B6,33F9,0000
108 DATA 08D6,0000,0546,33FC,0023,0000,0760,33FC
109 DATA 0001,0000,0762,33FC,0001,0000,0764,33FC
110 DATA 0001,0000,0766,33F9,0000,0548,0000,0796
111 DATA 23FC,0000,03DC,0000,0A16,6100,0274,33F9
112 DATA 0000,08D6,0000,054A,6100,0224,0C79,002B
113 DATA 0000,054C,66F2,3039,0000,0554,B079,0000
114 DATA 054A,66E4,6102,60E0,23FC,0000,03E8,0000
115 DATA 0A16,6100,02AE,33F9,0000,08D6,0000,0544
116 DATA 0C79,0001,0000,0544,6700,0268,0C79,0003
117 DATA 0000,0544,673E,5479,0000,04AE,0C79,0012
118 DATA 0000,04AE,6D06,4279,0000,04AE,41F9,0000
119 DATA 04B0,4280,3039,0000,04AE,33F0,0000,0000

```

120 DATA 04AC,E348,41F9,0000,04C2,23F0,0000,0000
121 DATA 041A,6094,23FC,0000,0422,0000,0A16,6100
122 DATA 0242,0C79,0002,0000,09D6,670C,6100,01D0
123 DATA 4A79,0000,04AE,6602,4E75,33FC,0002,0000
124 DATA 04EA,42B7,3E39,0000,04AC,DEBC,0000,0009
125 DATA EBB7,EBB7,2F07,3F3C,0048,4E41,5C8F,4A80
126 DATA 6716,23C0,0000,04E6,2F3C,0000,01AE,3F3C
127 DATA 0026,4E4E,5C8F,4E75,23FC,0000,045E,0000
128 DATA 0A16,6100,01DE,6000,01AA,23F9,0000,0472
129 DATA 0000,0520,23FC,0000,0250,0000,0472,23F9
130 DATA 0000,0476,0000,0524,23FC,0000,026B,0000
131 DATA 0476,23F9,0000,047E,0000,0528,23FC,0000
132 DATA 02B4,0000,047E,2079,0000,04E6,203C,0000
133 DATA 0A00,429B,51C8,FFFC,2079,0000,04E6,D1FC
134 DATA 0000,000B,43F9,0000,050C,203C,0000,051E
135 DATA 90BC,0000,050C,53B0,10D9,51C8,FFFC,3E39
136 DATA 0000,04AC,33C7,0000,04FA,E34F,DE7C,0012
137 DATA 2079,0000,04E6,D1FC,0000,0013,10C7,E04F
138 DATA 10B7,0BF9,0002,0000,04C5,4E75,0C6F,0002
139 DATA 0004,670B,2079,0000,0520,4ED0,203C,0000
140 DATA 04EC,4E75,0C6F,0002,000E,670B,2079,0000
141 DATA 0524,4ED0,302F,000C,48C0,E18B,E38B,206F
142 DATA 0006,322F,000A,5341,2279,0000,04E6,D3C0
143 DATA 302F,0004,0B00,0000,6702,C149,203C,0000
144 DATA 01FF,10D9,51C8,FFFC,51C9,FFF2,4240,4E75
145 DATA 0C6F,0002,0004,670B,2079,0000,0528,4ED0
146 DATA 3039,0000,04EA,4279,0000,04EA,4E75,33FC
147 DATA 0019,0000,0760,33FC,0010,0000,0762,33FC
148 DATA 0007,0000,0764,33FC,0001,0000,0766,23FC
149 DATA 0000,054C,0000,0A16,43F9,0000,03BC,45F9
150 DATA 0000,0796,700F,34D9,51CB,FFFC,6102,4E75
151 DATA 223C,0000,052C,303C,00C8,4E42,4E75,4AB9
152 DATA 0000,04E6,672C,2F3C,0000,035E,3F3C,0026
153 DATA 4E4E,5C8F,2F39,0000,04E6,3F3C,0049,4E41
154 DATA 5C8F,4A80,670C,23FC,0000,04B5,0000,0A16
155 DATA 6130,42B9,0000,04E6,4E75,23F9,0000,0520
156 DATA 0000,0472,23F9,0000,0524,0000,0476,23F9
157 DATA 0000,052B,0000,047E,0BB9,0002,0000,04C5
158 DATA 4E75,33FC,0034,0000,0760,33FC,0001,0000
159 DATA 0762,33FC,0001,0000,0764,33FC,0001,0000

160 DATA 0766,33FE,0000,0000,0768,33FC,0001,0000
161 DATA 0796,6100,FF5C,4E75,0013,0001,0001,0001
162 DATA 0000,0000,0000,0000,0000,0000,0000,0000
163 DATA 0000,0000,0000,0000,2020,5241,4D2D,4469
164 DATA 736B,2000,5B31,5D5E,5461,696C,6C65,2064
165 DATA 7520,5241,4D2D,4469,736E,2065,6E20,4B6F
166 DATA 6374,6574,7320,3F5D,5B20,457B,6974,7C20
167 DATA 706C,7573,207C,2031,3030,205D,0000,5B31
168 DATA 5D5B,4566,6661,6365,7220,616E,6369,656E
169 DATA 7C20,636F,6E74,656E,7520,6475,2052,414D
170 DATA 2D44,6973,6B20,203F,5D5E,206F,7569,2021
171 DATA 207C,204E,6F6E,205D,0000,5B32,5D5E,2052
172 DATA 414D,2069,6E73,7566,6669,7361,6E74,6520
173 DATA 215D,5B41,6C6F,7273,206E,6F6E,2E2E,5D00
174 DATA 005B,325D,5B45,7272,6575,7220,6176,6563
175 DATA 204D,4652,4545,2021,5D5B,416C,6F72,7320
176 DATA 6E6F,6E2E,2E5D,0000,0064,0002,0000,0064
177 DATA 00C8,012C,0190,01F4,0258,02BC,0320,2020
178 DATA 3020,2031,3030,2032,3030,2033,3030,2034
179 DATA 3030,2035,3030,2036,3030,2037,3030,2038
180 DATA 3030,0000,0000,0000,0200,0002,0400,0007
181 DATA 0005,0006,0012,0000,0000,0000,0000,0000
182 DATA 0000,0000,0000,0000,0002,0201,0002,7000
183 DATA 0000,0005,0009,0001,0000,0000,0000,0000
184 DATA 0000,0000,0000,0000,0000,0760,0000,0776
185 DATA 0000,0796,0000,03D6,0000,0A16,0000,0AB6
186 DATA 0000,0002,080B,080B,080A,0408,080B,080A
187 DATA 0408,080B,0806,0406,040A,040C,0806,0C04
188 DATA 0A04,080C,080B,0806,0B0B,0B0B,0804,0C0C
189 DATA 0C0B,1E06,1004,1206,0E06,0E06,0A12,0C06
190 DATA 060E,060C,240B,101C,300B,060A,080B,080B
191 DATA 0406,0612,0E0B,0E12,0408,080A,0A16,0B0B
192 DATA 0B0B,0B01,7C04,0404,0404,0000

Si vous mettez en route votre ordinateur et vous installez le RAM DISK, il vous arrivera sans doute fréquemment de copier en premier lieu de nombreux programmes sur ce RAM DISK. Cette copie prend du temps et ne constitue pas un travail très exaltant. Mais on peut résoudre ce problème.

6.2 COPIE D'UNE DISQUETTE VERS LE RAM DISK

Le programme qui suit copie simplement le contenu d'une disquette simple face sur le ram disk C. Les secteurs 0 (numéro de secteur logique) à 9*80-1, soit 719, sont lus et les secteurs sont copiés sur ram disk. Il faut bien sûr faire attention à ce que la capacité du ram disk soit au moins de 400 Ko car le secteur 719 existe lui aussi.

Pour que ce programme soit le plus rapide possible, on lit 9 secteurs à chaque appel de la fonction de lecture/écriture FLOPRW. Cet avantage de copie est dû au circuit DMA qui peut lire 9 secteurs à la suite grâce à une programmation adaptée et les renvoyer à l'ordinateur. Cela n'améliore pas la vitesse considérablement mais apporte un petit plus. Il serait plus rapide de charger tous les secteurs d'un seul coup mais on pourrait rencontrer des problèmes de place mémoire.

Si on met une disquette double face dans le lecteur à lire, le ram disk contiendra bien entendu le catalogue de tous les fichiers de cette disquette. Le catalogue a été bien entendu copié mais il manque la moitié inférieure des secteurs. Si la disquette originale est remplie au delà de sa moitié, les programmes qui s'y trouvent ne pourront pas être chargés dans le ram disk. En dehors de cette limitation, le programme fonctionne avec les disquettes double face.

La structure très simple du programme vous permettra de le modifier facilement. Vous pouvez par exemple modifier la condition de la commande `CMP #9*80,SECTOR` pour pouvoir copier les disquettes de 800 Koctets en mettant tout simplement à la place `#9*80*2`.

Voici ce programme ainsi que le chargeur BASIC qui crée le programme DTRCOPY.PRГ :

*** Disk - to - RAM-Disk - Copy S.D. ***

run:

```

clr.l  ap1rsv
clr.l  ap2rsv
clr.l  ap3rsv
clr.l  ap4rsv
move   #10,opcode    ;appi_init
move   #0,sintin
move   #1,sintout
move   #0,saddrin
move   #0,sintin
jsr    aes
move   #77,opcode    ;graf_handle
move   #5,sintout
move   #0,saddrin
move   #0,saddrout
jsr    aes
move   intout,grhandle

move.l #alarm,d0
bsr    formalert      ;Affichage de zone de dialogue
subq   #2,d0          ;Correction du numéro de lecteur
tst    d0
bmi    quit           ;Quitter

move   d0,drive       ;Mémorise numéro de lecteur
clr    sector         ;Commence au secteur 0

```

loop:

```

move   drive,d1       ;Disquette sélectionnée
move   #2,d0           ;Read
bsr    floprw          ;Lecture de 9 Secteurs
bne    readerr         ;Erreur de lecture !
move   #2,d1           ;Drive C = RAM-Disk
move   #1,d0           ;Write
bsr    floprw          ;Ecriture de 9 secteurs
bne    wrerr           ;Erreur d'écriture !
add    #9,sector       ;numéro de secteur + 9
cmp    #9*80,sector    ;Fin ?

```

```
        blt     loop      ;non

quit:
        clr -(sp)
        trap #1           ;Exit => Desktop

floprw:
        ;Disquette Read/Write
        move    d1, -(sp) ;Lecteur
        move    sector, -(sp) ;Start-Sector
        move    #9, -(sp) ;Lecture/écriture de 9 secteurs
        pea     buffer    ;Tampon
        move    d0, -(sp) ;Read/Write
        move    #4, -(sp)
        trap    #13       ;rwabs-fonction
        add.l   #14, sp
        tst     d0        ;Teste si erreur
        rts

readerr:
        move.l   #reer, d0
        bsr     formalert ;"Erreur de lecture !"
        bra     quit

wrerr:
        move.l   #wrer, d0
        bsr     formalert ;"Erreur d'écriture"
        bra     quit

aes:
        ;Appel-AES
        move.l   #aesp, d1
        move     #$c8, d0
        trap    #2
        rts

formalert:
        move     #52, contrl ;form_alert
        move     #1, contrl+2
        move     #1, contrl+4
        move     #1, contrl+6
        move     #0, contrl+8
```

```

move    #1,intin
move.l  d0,addrin
jsr     aes
move    intout,d0
rts

```

```

alarm:   dc.b "[1][Copie à partir de: quel lecteur ?]"
         dc.b "[Exit: A : B ]",0,0
reer:    dc.b "[2][Erreur de lecture !][Quit]",0,0
wrrer:   dc.b "[2][Erreur d'écriture !][Quit]",0,0

```

```

even
aespb:   dc.l contrl,global,intin,intout,addrin,addrout

```

```

data
contrl:                                     ;divers champs pour l'AES

```

```

opcode:  blk.w 1
sintin:  blk.w 1
sintout: blk.w 1
saddrin: blk.w 1
saddrout: blk.l 1
         blk.w 5

```

```

global:  blk.w 7
ap1rsv:  blk.l 1
ap2rsv:  blk.l 1
ap3rsv:  blk.l 1
ap4rsv:  blk.l 1

```

```

intin:   blk.w 128
ptsin:   blk.w 128
intout:  blk.w 128
ptsout:  blk.w 128
addrin:  blk.w 128
addrout: blk.w 128

```

```

grhandle: blk.w 1
drive:    blk.w 1      ;Numéro de lecteur
sector:   blk.w 1      ;Compteur de secteurs
buffer:   blk.b 9*512  ;Tampon pour 9 secteurs

```

```

10 '***** File-Maker      A.S.  *****
15 '
20 ? :fullw 2:clearw 2:gotoxy 0,0
25 ? "Le fichier >> dtrcopy.prg << est crée":?::?
30 dim c%( 265):cs#=0
35 for i=0 to 265
40 read a$:c%(i)=val("&H"+a$)
45 check#=check#+(c%(i))
50 next i
55 if check#= 2461714.08 then 70
60 ? "Ca ne va pas encore, car quelque chose ne convient pas dans les DATAs
."
65 goto 80
70 bsave "dtrcopy.prg",varptr(c%(0)), 532
75 ? "Le programme >> dtrcopy.prg << est écrit."
80 ?::?::?"Pressez une touche":a=inp(2):end
85 '
90 '***** DATA's pour dtrcopy.prg *****
95 '
100 DATA 601A,0000,01CC,0000,0000,0000,183A,0000
101 DATA 0000,0000,0000,0000,0000,0000,42B9,0000
102 DATA 01F0,42B9,0000,01F4,42B9,0000,01F9,42B9
103 DATA 0000,01FC,33FC,000A,0000,01CC,33FC,0000
104 DATA 0000,01CE,33FC,0001,0000,01D0,33FC,0000
105 DATA 0000,01D2,33FC,0000,0000,01CE,6100,00AE
106 DATA 33FC,004D,0000,01CC,33FC,0005,0000,01D0
107 DATA 33FC,0000,0000,01D2,33FC,0000,0000,01D4
108 DATA 6100,00BA,33F9,0000,0400,0000,0800,203C
109 DATA 0000,013E,6100,00B4,5540,4A40,6B32,33C0
110 DATA 0000,0802,4279,0000,0B04,3239,0000,0B02
111 DATA 7002,6120,6642,7202,7001,611B,6644,0679
112 DATA 0009,0000,0B04,0C79,02D0,0000,0B04,6DDA
113 DATA 4257,4E41,3F01,3F39,0000,0B04,3F3C,0009
114 DATA 4B79,0000,0B06,3F00,3F3C,0004,4E4D,DEFC
115 DATA 0000,000E,4A40,4E75,203C,0000,0174,611A
116 DATA 60CE,203C,0000,0194,6110,60C4,223C,0000
117 DATA 01B4,303C,00C8,4E42,4E75,33FC,0034,0000
118 DATA 01CC,33FC,0001,0000,01CE,33FC,0001,0000

```

119 DATA 01D0,33FC,0001,0000,01D2,33FC,0000,0000
120 DATA 01D4,33FC,0001,0000,0200,23C0,0000,0600
121 DATA 61E8,3039,0000,0400,4E75,5B31,5D59,436F
122 DATA 7069,6520,8520,7061,7274,6972,2064,657C
123 DATA 2071,7565,6C20,6C65,6374,6575,7220,3F5D
124 DATA 5B45,7869,747C,2041,207C,2042,205D,0000
125 DATA 5E32,505B,4572,7265,7572,2064,6520,6C65
126 DATA 6374,7572,6520,215D,5E51,7569,745D,0000
127 DATA 5E32,505B,4572,7265,7572,2064,2782,6372
128 DATA 6974,7572,6520,215D,5E51,7569,745D,0000
129 DATA 0000,01CC,0000,01E2,0000,0200,0000,0400
130 DATA 0000,0600,0000,0700,0000,0002,0606,0608
131 DATA 080B,080B,0C0B,0E0B,0A04,0610,0606,1608
132 DATA 0E0A,1E0A,0A10,0B0B,080B,0E06,067C,0404
133 DATA 0404,0400

7. Exemple de programmation en langage machine avec un éditeur de disquettes.

Les programmes que vous avez rencontrés dans cet ouvrage vous permettent déjà de lire et de modifier certaines données de la disquette mais que serait un livre du lecteur de disquette sans éditeur de disquettes permettant de modifier toutes les données d'une disquette ? Etant donné que je dispose déjà d'une expérience de 7 ans de saisie de programmes sur les livres et les magazines, j'aimerais vous éviter cette frustration et vous proposer un programme dont la structure quasi modulaire vous conduira à un résultat sans trop de saisie.

Le listing `edit.s` contient toutes les options, les données et les sous-programmes de l'éditeur de disquettes. Cependant, seules les routines du menu sector ont été écrites. Les autres routines ne contiennent qu'un RTS et reviennent donc immédiatement au point de départ. Le listing `subrout.s` contient toutes les routines sous le même nom. Vous pourrez alors les insérer dans le programme `edit.s` et en faire un programme complet.

Le plus gros problème lors de la saisie tiendra sans doute à l'éditeur : le nom des labels et des variables. Dans un tel projet en assembleur, 8 lettres sont insuffisantes pour formuler des noms significatifs.

Si vous voulez utiliser immédiatement l'éditeur, vous avez également la possibilité d'acheter la disquette qui contient les sources et les codes exécutables des programmes de ce livre.

7.1 LES FONCTIONS TOS POUR L'ACCES AU LECTEUR DE DISQUETTE.

Les fonctions de l'éditeur sont pour la plus grande part construites sur les fonctions du système d'exploitation (TOS ou GEMDOS). Seule une petite partie utilise directement le DMA et le contrôleur. L'accès au DMA et au contrôleur de l'ATARI ST est possible à partir de plusieurs plans du TOS qui est construit de manière hiérarchique.

Les langages évolués comme PASCAL, C, FORTRAN et BASIC permettent le traitement de fichiers séquentiels et directs. Ils ne divisent donc pas la disquette en pistes et en secteurs mais en fichiers. Leur accès est donc limité à ces fichiers.

Si on fait un pas de plus dans la hiérarchie du système d'exploitation, on se rend compte des fonctions GEMDOS utilisables par les langages évolués avec l'ATARI ST. Ces fonctions GEMDOS représentent les fonctions du système d'exploitation pour les langages évolués. Ces fonctions sont donc également orientées fichiers et permettent l'accès direct ou séquentiel aux fichiers de la disquette.

Avec le pas suivant, on entre dans les fonctions de la BIOS qui permettent le premier contact physique avec la disquette car elles divisent la disquette en secteurs (1440 pour les double face et 720 pour les simple face). Les fonctions de la BIOS autorisent l'accès à tous les secteurs de la disquette. Mais on ne sait pas sur quelle piste et quelle face se trouve le secteur logique lu. Mais on peut retrouver cela à partir des données du bios parameterbloc. Si on utilise les fonctions de la XBIOS, l'accès aux pistes, aux faces et aux secteurs physiques, est possible. Mais l'utilisateur doit savoir combien il y a de secteurs par piste, etc. Les fonctions de la XBIOS offrent la possibilité de définir de telles particularités. Ainsi, on peut formater des secteurs définis et un nombre donné de secteurs par piste.

Le point central de toutes les commandes est constitué par le DMA et le contrôleur WD 1772 qui prennent certaines adresses dans le secteur I/O de l'ATARI ST entre \$FF800 à \$FFFFFF. Ces adresses permettent de contrôler certaines fonctions de ces circuits.

En clair : la fonction WRITE# de BASIC utilise la fonction GEMDOS WRITE qui appelle elle-même la fonction RWABS de la BIOS qui utilise la fonction FLOPWR de la XBIOS qui elle-même indique au DMA et au contrôleur où et comment écrire ces données.

Toutes les fonctions du système d'exploitation sont décrites en détail dans le livre "La bible de l'ATARI ST" chez MICRO APPLICATION.

Je ne m'intéresserai ici qu'aux 8 fonctions de la BIOS et de la XBIOS qui servent à la communication directe avec la disquette. Tous les appels récupèrent les paramètres sur la pile et renvoient les résultats ou un code d'erreur négatif sur le registre D0. Dans la plupart des cas, les registres D0-D2 et A0-A2 sont modifiés après un appel et doivent éventuellement être mémorisés. Les deux fonctions RWASB et GETBPB sont appelées avec le TRAP #13 spécifique à la BIOS et ont les significations suivantes :

RWABS : fonction BIOS numéro 4

Cette fonction très flexible sert aussi bien à la lecture qu'à l'écriture d'un ou de plusieurs secteurs logiques. Ces secteurs peuvent se trouver aussi bien sur disquette que sur disque dur ou sur ram disk. Voici les paramètres utilisés :

Device : détermine le lecteur auquel on a accès. La numérotation commence à 0 pour le lecteur A et n'est pas limitée vers le haut. Le ram disk présenté dans ce livre au chapitre 6.1 qui est nommé C a donc pour numéro de lecteur 2.

Recnr : indique le numéro logique du secteur à traiter. Le début est 0. Le nombre maximum de secteur varie selon l'appareil. Ainsi, une disquette ATARI simple face avec 80 pistes contient 720 secteurs logiques dont seulement 702 restent à disponibilité de l'utilisateur. Le TOS utilise les 18 secteurs restants pour gérer les données utilisateur à l'aide du catalogue et de la FAT (File Allocation Table).

nombre : définit le nombre des secteurs logiques à traiter.

Buffer : c'est une adresse dans laquelle on écrit ou à partir de laquelle on lit les données pour le lecteur. Si vous voulez lire par exemple 4 secteurs logiques au format ATARI, (512 octets/secteur), il faut avoir $4 \times 512 = 2048$ octets libres à partir de l'adresse Buffer.

rwflag : détermine enfin s'il faut lire ou écrire. Il y a 4 valeurs possibles :

rwflag	signification
0	lire secteur.
1	écrire secteur.
2	lire le secteur quoiqu'il arrive (même changement de disquette).
3	écrire le secteur quoiqu'il arrive (même changement de disquette).

Voici comment pourrait apparaître un appel en langage machine :

move.w	#0,-(a7)	* lecteur A (device)
move.w	#11,-(a7)	* recnr commence au secteur logique 11
		secteurs
move.l	#buffer,-(a7)	* adresse de la place mémoire
		disponible
move.w	#2,-(A7)	* RWFLAG, lire quoiqu'il arrive
move.w	#4,-(a7)	* numéro de fonction BIOS
trap	#13	* appel BIOS
add.l	#14,a7	* restauration de la pile
tst.w	d0	* teste s'il y eu une erreur
bmi	error	* valeur négative égale erreur
...		
...		* Normalement, on trouve la suite ici.
...		* Les adresses lues se trouvent en ram
...		* à l'adresse buffer

Getbpb fonction BIOS numéro 7

Le biosparameterbloc contient les données de la disquette. Ces données se trouvent dans le boot secteur de la disquette et sont inscrites dans le BPB se trouvant en RAM. Appel en assembleur :

move.w	device,-(a7)	* numéro du lecteur 0=A
move.w	#7,-(a7)	* numéro BIOS
trap	#13	
addq.l	#4,a7	* restauration de la pile
tst.w	d0	
bmi	error	* en cas d'erreur, D0 est négatif
...		

...
...
...
...
...

- * sinon, D0 contient l'adresse du BPB.
- * normalement \$4DCE correspond au
- * lecteur A et \$4DEE au lecteur B.

Les données sont en mémoire sous la forme de mots (2 octets) à partir de l'adresse dans D0. On a :

Lecteur A

Adresse	Nom	Signification	SS	DS
\$4DCE	recsiz	taille de secteur en octets	512	512
\$4DD0	clsiz	taille de cluster en secteurs	2	2
\$4DD2	clsizb	taille de cluster en octets	1024	1024
\$4DD4	rdlen	taille de catalogue en secteurs	7	7
\$4DD6	fsiz	taille de la FAT en secteurs	5	5
\$4DD8	fatrec	numéro de secteur de la 2ième FAT	6	6
\$4DDA	datrec	numéro secteur du 1er cluster data	18	18
\$4DDC	numcl	nombre de cluster sur disque	351	711
\$4DDE	bflags	divers flags		
\$4DE0	inconnu			
\$4DE2	nside	nombre de face du disque	1	2

Les exemples de données sont valables pour le format ATARI avec 80 pistes (SS= single sided, DS= double sided).

Mediach fonction BIOS numéro 9

Cette fonction teste si la disquette a été changée à l'aide du nom de disquette. Le paramètre utilisé est le numéro de lecteur.

```

move.w device, -(a7)    * Numéro de lecteur (p.ex. 0 pour A)
move.w #9, -(a7)        * Numéro de fonction BIOS
trap #13                * Appel
addq.l #4, a7           * Restauration de la pile

```

...

Les valeurs rendues dans D0 vont de 0 à 2 et signifient :

Numéro	Signification
0	La disquette n'a pas été changée
1	La disquette a été éventuellement changée
2	La disquette a été changée

Voici les 4 fonctions de la XBIOS que nous allons utiliser :

Floprd fonction numéro 8

Cette fonction permet de lire un ou plusieurs secteurs situés les uns à la suite des autres. Les paramètres à envoyer sont :

nombre : détermine le nombre de secteurs à lire. Les valeurs possibles avec les disquettes au format ATARI vont de 1 à 10. Mais on ne peut lire 10 secteurs que si on les a créés au moyen d'un programme de formatage spécial. En effet le programme de formatage ATARI n'inscrit que 9 secteurs par piste.

face : indique la face de la disquette, 0 ou 1.

Track : détermine la piste sur laquelle se trouve le secteur à lire.

sector : le secteur physique lui même.

device : le paramètre de lecteur déjà connu (0=A)

filler : un mot long qui n'a aucune signification et qui est certainement prévu pour une extension de l'instruction.

Buffer : l'adresse à laquelle les données doivent être mises.

move.w	#1, -(a7)	* nombre, un secteur
move.w	#0, -(a7)	* face
move.w	#0, -(a7)	* PISTE zéro
move.w	#1, -(a7)	* secteur 1 = boot secteur
move.w	#0, -(a7)	* lecteur A
move.l	#0, -(a7)	* filler, mot long inutile
move.l	#buffer, -(a7)	* adresse de l'endroit où inscrire les données
move.w	#9, -(a7)	* numéro de fonction XBIOS

```

trap      #14
add.l     #20,a7
tst.w     d0
bmi       error
...
buffer :  ds.b      512

```

Flopwr : Fonction XBIOS numéro 9

De même qu'avec la fonction précédente, flopwr permet d'inscrire des secteurs sur disquette. Les paramètres à donner sont les mêmes que pour floprd.

```

move.w    #4,{a7}      * nombre, quatre secteurs
move.w    #0,{a7}      * face
move.w    #5,{a7}      * PISTE cinq
move.w    #1,{a7}      * secteur 1 = secteur de début
                        d'écriture
move.w    #0,{a7}      * lecteur A
move.l    #0,{a7}      * filler, mot long inutile
move.l    #buffer,{a7} * adresse de l'endroit où inscrire les
                        données
move.w    #9,{a7}      * numéro de fonction XBIOS
trap      #14
add.l     #20,a7
tst.w     d0
bmi       error
...
buffer :  ds.b      4*512

```

Cet appel écrit les 2048 octets de données qui se trouvent en mémoire à partir de l'adresse "buffer", sur les secteurs 1,2,3,4 de la piste 5 face 0 de la disquette.

Flopfmt : fonction XBIOS numéro 10

Cette routine autorise le formatage d'une piste avec 1-10 secteurs par piste. Les paramètres sont :

virgin : ce mot définit les nouveaux contenus de secteurs, c'est-à-dire les données qui seront inscrites dans les secteurs. Il faut prendre la même valeur que le TOS (\$E5E5) car les valeurs d'octets

supérieures à \$EF ne sont pas inscrites sous forme d'octets mais représentent des fonctions particulières et écrivent l'address mark ou la checksum de l'octet précédent (voyez le chapitre concernant le contrôleur de disquettes).

magic : c'est la constante magique \$87654321

Interleave : détermine l'ordonnancement des secteurs sur la disquette. Les ordinateurs sans DMA et avec des contrôleurs moins intelligents doivent transformer les données en provenance de la disquette, ce qui prend du temps. Il peut arriver que le secteur suivant soit déjà lu lorsque le CPU a terminé son travail. Ainsi, on n'écrit pas les secteurs dans l'ordre 1 2 3 4 5 6 7 8 9 mais dans l'ordre 1 6 2 7 3 8 4 9 5. Cela améliore la lecture car il faudrait attendre un tour complet de la disquette pour lire le secteur suivant. Dans ce cas, on passe par dessus un secteur seulement. Mais avec l'ATARI ST, la conversion est effectuée par le contrôleur si bien qu'on peut écrire les secteurs les uns derrière les autres. Il faut donc inscrire une valeur de 1.

Face : face de la disquette

track : piste objet

spt : nombre de secteurs par piste. La disquette autorise 10 secteurs par piste mais le TOS ne travaille qu'avec 9 secteurs.

device : numéro de lecteur

filler : encore un mot sans importance pour des extensions ultérieures.

buffer : détermine l'adresse à laquelle la XBIOS construit la piste complète (avec octets SYNC et champs d'adresse). Elle utilise environ 8 Koctets.

Le numéro de la fonction est 10.

move.w	#\$e5e5, -(a7)	* virgin
move.l	#\$87654321, -(a7)	* magic
move.w	#1, -(a7)	* interleave égale 1
move.w	#0, -(a7)	* face 0

```

move.w    #5,-(a7)      * piste 5
move.w    #9,-(a7)      * spt, 9 secteurs par piste
move.w    #0,-(a7)      * device 0 égale lecteur A
move.l    #0,-(a7)      * filler, mot long inutile
move.l    #buffer,-(a7) * adresse de la mémoire libre
move.w    #10,-(a7)     * numéro de fonction XBIOS
trap      #14
add.l     #26,a7
tst.w     d0             * erreur ?
bmi       error          * oui
...
...
buffer :   ds.b    8*1024    * place réservée

```

Protobt : Fonction XBIOS numéro 18

Cette fonction facilite la création de boot secteur pour différents formats de disquettes. On lit d'abord le secteur 1 de la piste 0 face 0 d'une disquette formatée, on appelle alors protobt et on écrit le boot secteur modifié par protobt sur le secteur 1 piste 0 face 0 sur la disquette sur laquelle il faut écrire le boot secteur. Voici les paramètres à donner :

execflag : indique si le boot secteur est exécutable c'est-à-dire s'il y a un programme exécutable au 30 ième octet à partir du début du secteur. Les valeurs possibles sont :

Valeur	Signification
0	inexécutable
1	exécutable
-1	le buffer reste comme il était

disktyp :	Type de la disquette
0	40 pistes, single sided (SS, SD 180 Koctets)
1	40 pistes, double sided (DS, SD 360 koctets)
2	80 pistes, single sided (SS, DD 360 koctets)
3	80 pistes, double sided (DS, DD 720 koctets)
-1	le type de disquette demeure inchangé

Les formats ATARI sont comme vous le savez déjà du type double densité format 2 et 3.

serialnr : le numéro de série est un nombre de 24 bits qui est inscrit dans le boot secteur et qui permet au système d'exploitation de déterminer un éventuel changement de disquette. Si le numéro de série indiqué est supérieur à 24 bits (p.ex. \$01000000), le système d'exploitation inscrit un nombre aléatoire, s'il est -1, le numéro de série du tampon n'est pas modifié.

buffer : indique l'adresse à laquelle se trouve le boot secteur à modifier (512 octets).

```

move.w    #-1,-(a7)          * execflag, exécutabilité non modifiée
                                double sided
move.l     #$04000000,-(a7)  * numéro de série aléatoire
move.l     #buffer,-(a7)    * endroit où se trouve le secteur
move.w     #18,-(a7)        * numéro de fonction XBIOS
trap       #14,-(a7)
add.l      #14,a7
tst.w      d0
bmi        error            * cela n'a pas marché !
...
...
buffer :   ds.b   512

```

Ce fragment de programme transforme un boot secteur de disquette simple face en boot secteur pour une disquette double face qui peut alors être inscrit sur une disquette formatée en double face. Ainsi, on pourra charger le système d'exploitation à partir d'une disquette double face.

Voyons maintenant les informations qui se trouvent dans le boot secteur.

Les données sous 16 bit sont inscrites sous le format INTEL (low byte / high byte) et on reconnaît un boot secteur exécutable à une checksum de \$1234.

Octet	40	pistes SS	40 pistes DS	80 pistes SS	80 pistes DS
0,1	bra 30 branchement à \$30 si le boot secteur est exécutable				
2-7	text : 'loader'				
8-10	serialnr				
11-12	bps	512	512	512	512
13	spc	2	2	2	2
14-15	res	1	1	1	1
16	fat	2	2	2	2
17-18	dir	64	112	112	112
19-20	sec	360	720	720	1440
21	media	252	253	248	249
22-23	spf	2	2	5	5
24-25	spt	9	9	9	9
26-27	side	1	2	1	2
28-29	hide	0	0	0	0
30	bootcode : à partir d'ici se trouve le boot code si le boot secteur est exécutable				
..					
..					
510-511	Checksum du boot sector de l'octet 0 à 509.				

7.2 LISTING ET MODE D'EMPLOI DU

MONITEUR DE DISQUETTES

Comme nous vous l'avons dit plus haut, voici la première partie de l'éditeur de disquettes. Ce listing est prévu pour l'assembleur de DIGITAL RESEARCH et peut être traité tel quel. Une fois que vous avez saisi et assemblé le listing, seul le menu secteur fonctionnera. Pour que le programme soit complet, il faudra ajouter les sous-programmes accessoires.

```

*****
*
*       The little Diskeditor, U. Braun , August 1986
*
*
*       DATA BECKER & MICRO APPLICATION
*
*****

```

text

```

*****
*
* Branchement après chargement, calcul longueur, réservation de place
*
*****

```

```

sstart: move.l  a7,a5      * la Basepageadress est sur la pile
        move.l  4(a5),a5   * basepage address = début du programme - $100
        move.l  $c(a5),d0  * Longueur du programme
        add.l   $14(a5),d0 * Longueur du secteur de données initialisé
        add.l   $1c(a5),d0 * Longueur du secteur de données non initialisé
        add.l   #$1100,d0  * 4 K-Byte Userstack=place restante
        move.l  a5,d1      * Startadress du programme
        add.l   d0,d1      * Plus nombre d'octets occupés = besoin en place
        and.l   #-2,d1     * adresse paire pour la pile
        move.l  d1,a7      * Userstackpointer sur derniers 4K- Byte
        move.l  d0,-(sp)   * Longueur du secteur réservé
        move.l  a5,-(sp)   * Adresse de début du secteur réservé
        move.w  d0,-(sp)   * Dummy-Word
        move.w  #$4a,-(sp) * GEM Dos Fonction SETBLOCK
        trap    #1
        add.l   #12,sp     * restauration de l'ancienne pile
        move.w  #3,-(a7)   * ralentir Repeat-Rate car dépassement
        move.w  #$b,-(a7)  * du tampon clavier si plus grand
        move.w  #35,-(a7)  * Repeat-Rate
        trap    #14
        addq.l  #6,a7

        jsr     main       * Branchement au programme principal . ( créer User )
        move.l  #0,-(a7)   * Fin du programme en cours

```

```
trap #1          ‡ Retour au Gem-Desktop
```

```

#####
#####
‡
‡   Le programme proprement dit commence ici
‡
#####

```

```

main:  jsr      start1          ‡ Initialiser Line-A
        jsr      videbuff       ‡ Vider tampon clavier
        jsr      clear         ‡ Effacer écran
        jsr      init          ‡ Positionner paramètres par défaut
        jsr      gomain         ‡ Créer menu principal
        jsr      menul1        ‡ Envoyer exécution au Menu-Handler
endmain: rts                  ‡ Retour au Desktop

```

```
#####
```

```

init:  jsr      cursoff        ‡ Désactiver curseur
        jsr      clear         ‡ Effacer écran
        move.w   #0,wtrack     ‡ Piste 0 face 0
        move.w   #0,wside
        move.w   #0,wdrive     ‡ drive zéro, secteur un
        move.w   #1,wsector    ‡
        move.w   #0,d0
        move.w   #6,maxdriv     ‡ nombre maximal de lecteurs
        move.w   #1,maxside     ‡ nombre maximal de faces - 1
        move.w   #79,maxtrack   ‡ nombre maximal de pistes par défaut
        move.w   #9,maxsect     ‡ " maximal " Sectors "
        move.w   #9,asector     ‡ nombre maximal de Sect/Track
        move.b   #'0',setrack   ‡ Donner les bonnes valeurs
        move.b   #'9',setrack+1 ‡ à la chaîne de menu
        move.w   #1500,maxclust ‡ nombre maximal de Cluster
        move.l   #platztr,editptr ‡ Buffer
        jsr      prmessag       ‡ pub pour livre
        rts                    ‡ et retour

```

```

*****
* Affiche un texte de pub avec copyright
*****

```

```

prmessag: jsr    videbuff        # vider tampon de clavier
           move.w #20,column      # positionner curseur
           move.w #10,ligne
           jsr    loccurs
           move.l #haques1,a0     # Afficher Message partie 1
           jsr    printf
           move.w #20,column
           move.w #12,ligne
           jsr    loccurs        # positionner curseur
           move.l #haques2,a0     # Message partie 2
           jsr    printf
           move.w #20,column
           move.w #14,ligne
           jsr    loccurs        # positionner curseur
           move.l #haques3,a0     # Message partie 3
           jsr    printf
           jsr    wtastr          # Attendre une pression de touche
           jsr    clear          # Effacer écran
           jsr    videbuff       # Effacer tampon de clavier
           rts                  # et retour

```

```

*****
#
# C'est la boucle des menus, la partie de programme qui
# contrôle le programme en entier. Elle détermine si on a
# exécuté une autre option de menu avec curseur-gauche ou
# curseur droit ou si on a sélectionné une option de menu avec
# curseur-haut ou curseur-bas. caselect sert alors à envoyer
# ces contrôles à l'option.
#
*****

```

```

menu1: jsr    touch    # Test du clavier

```

```

tst.l    d0          * Pas de saisie, continuer à attendre
beq      menu1
swap     d0          * Sinon, tester plusieurs touches
cmp.b    #$44,d0     * F-10 = fin, NOTSTOP, pour Debug
beq      menuende

menu2:    cmp.b    #$4b,d0    * Curseur gauche
        bne      menu3
        jsr      curleft
        bra      menu1

menu3:    cmp.b    #$4d,d0    * Curseur droit
        bne      menu4
        jsr      curright
        bra      menu1

menu4:    cmp.b    #$50,d0    * Curseur bas
        bne      menu5
        jsr      cursdown
        bra      menu1

menu5:    cmp.b    #$48,d0    * Curseur-haut
        bne      menu6
        jsr      cursup
menu6:    bra      menu1

endmain2: add.l    #8,a7      * Ici, il y a encore deux adresse de
menuende: rts              * retour sur la pile (supprimer)

#####
*   Curseur-up sélectionne une option. Le bloc d'adresse de branchem. *
*   est chargé après jmpable, et indiqué à meselect.                  *
*                                                                       *
#####

cursup:   move.l    invar,jmpable    * Table de branchement par
        jsr        meselect        * Curseur haut, exécution de routine

```

* et retour à la boucle de menu

```

*****
* Une option de menu a été sélectionnée par curseur-bas. Dans
* certains menus, les sous-programmes exécutés diffèrent suivant
* la touche sélectionné (haut ou bas) p. ex inctrack et dectrack
* dans le menu sector.
*****

```

```

cursdown: move.l    decvar,jmptable    * Table de branchement pour
        jsr        meselect          * Curseur bas
        rts

```

[illegible]

```

curleft: move.l    revnum,d0          ‡ Sélection de l'option du menu
        sub.l      #1,d0
        beq        laround           ‡ Ecrire la même chose
        move.l     d0,revnum         ‡ en inversé
        bra        curlend
laround: move.l    ganz,revnum        ‡ swap around
curlend: jsr       dispmen           ‡ Affichage du menu
        rts

```

```

*****
*  comme avec curleft, mais c'est Curseur droit qui a été pressé  *
*****

```

```
currright: move.l    revnum,d0          ‡ Comme avec l'urgauche
            add.l     #1,d0
            cmp.l     ganz,d0
            bgt       raround
```

```

        move.l    d0,revnum
        bra       current
raround: move.l    #1,revnum
current: jsr      dispnen
        rts

```

```

#####
* Exécute la sous-routine
#####

```

```

meselect: jsr      videbuff      * Appel de la routine
          move.l    jumptable,a0  * l'adresse de début
          move.l    revnum,d0     * du bloc d'adresse de branchement
          subq.l    #1,d0         * fois quatre est dans jumptable
          lsl.l     #2,d0         * une adresse
          move.l    (a0,d0.l),a1   * fait quatre octets. Chargement
          jmp       (a1)          * et exécution de la routine

```

```

#####
* Traite l'erreur survenue en affichant une chaîne d'erreur
* à partir du numéro d'erreur négatif sur la pile.
*
#####

```

```

errhand: move.w    #10,column    * Le numéro d'erreur est envoyé sur
          move.w    #2,ligne     * la pile (not)
          jsr       loccurs      * Positionner le curseur en ligne 2
          jsr       delline     * Effacer ligne
          move.w    4(a7),d0     * Récupérer numéro d'erreur
          neg.w     d0           * Mettre en positif
          cmp.w     #29,d0       * Comparer avec l'erreur maximale
          blt       errhand1
          move.w    #29,d0       * Numéro d'erreur par défaut

```

```

errhand1: lsl.w     #2,d0        * Utiliser comme pointeur dans
          move.l    #errtab,a1   * le tableau d'erreur
          move.l    0(a1,d0.w),a0 * Récupérer la chaîne d'erreur
          jsr       printf      * imprimer

```

```

jsr      wtast           ‡ Attendre pression d'une touche
jsr      delline        ‡ Effacer de nouveau la ligne
jsr      cursbuf        ‡ Curseur de nouveau en ligne 4
move.l   (a7)+,a0       ‡ Récupérer adresse de retour
addq.l   #2,a7          ‡ Corriger la pile (erreurs.)
jmp      (a0)           ‡ Retour à l'appelant

```

```

#####
#####
‡ Envoie les Parametres pour le menu principal a divers variables ‡
‡ menuadr, incvar, decvar, ganz, revnum ‡
#####

```

```

gomain: jsr      clear           ‡ Effacer écran
        move.l   #7,ganz        ‡ Setp options dans menu principal
        move.l   #1,revnum      ‡ Première option inversée
        move.l   #menmain,menuadr ‡ Adresses de la chaîne de menu
        move.l   #haincjmp,incvar ‡ Adresses des routines de menu
        move.l   #haincjmp,decvar ‡ pour Curseur haut et bas
        jsr      dispmen        ‡ égalé, afficher Menu
        rts                  ‡ et retour

```

```

#####
#####
‡ Voici les routines du menu principal ‡
#####
#####

```

```

#####
‡ Met dans mes variables du système Menuselect, les adresses ‡
‡ pour l'option TRACK ‡
#####

```

```

gotrack: jsr      clear           ‡ Effacer écran
        move.l   #mentrack,menuadr ‡ Adresses des chaînes de menu
        move.l   #8,ganz        ‡ Le menu track a 8 options
        move.l   #5,revnum      ‡ Représenter le 5. point en inversé

```



```

move.l #trincjmp,incvar  * Cursorup-Jumptable
move.l #trdecjmp,decvar  * Cursor-down-Jumptable
jsr     dispmen          † Afficher menu
jsr     cursmess         † et un message
move.l #trques1,a0       † "TRACK MODE"
jsr     printf
rts                      † Retour au Menuhandler

```

```

#####
‡ Met dans les variables du système Menuselect, les adresses de ‡
‡ l'option TRACK with SYNCs ‡
#####

```

```

gosync: move.l #6,ganz      * Le menu Track mit Sync a
      move.l #4,revnum      * six option
      move.l #syincjmp,incvar * up-Jumptable
      move.l #sydecjmp,decvar * down-Jumptable
      move.l #amensync,menuadr * Adresses des chaînes de menu
      jsr     dispmen        † Affichage du menu
      jsr     cursmess       † Positionner curseur
      move.l #trques2,a0     † "Track with Syncs"
      jsr     printf         † imprimer
      rts

```

```

#####
‡ Met dans les variables du système Menuselect, les adresses de ‡
‡ L'option SECTOR ‡
#####

```

```

gosector: jsr     clear
      move.l #mensect,menuadr * Option du menu secteur
      move.l #seincjmp,incvar
      move.l #sedecjmp,decvar
      move.l #platztr,editptr
      move.l #8,ganz          * 8 options
      move.l #5,revnum        † Représenter 5. en inversé

```

```

    jsr    dispmen
    jsr    cursmess
    move.l #seques1,a0
    jsr    printf
    rts

```

```

#####
* Met dans les variables du système Menuselect, les adresses de      *
* l'option CLUSTER                                                    *
#####

```

```

goclust: jsr    initdrv          * menu cluster, initialiser
          jsr    rdfat           * premier lecteur puis lire la FAT
          move.l #8,ganz         * Le menu a 8 sous-options
          move.l #3,revnum       * read = inversé
          move.l #menclust,mnuadr * Adresse de la chaîne de menu
          move.l #clincjmp,incvar * Jumptable
          move.l #cldecjmp,decvar
          jsr    cursmess
          move.l #clques1,a0     * "cluster mode"
          jsr    printf          * Ecrire
          jsr    dispmen         * Afficher menu
          rts                   * et retour

```

```

#####
* Met dans les variables du système Menuselect, les adresses de      *
* L'option FORMAT                                                      *
#####

```

```

goformat: jsr    clear          * Menu Format
          move.l #formmen,mnuadr * Adresse de la chaîne de menu
          move.l #8,ganz         * 8 options de menu
          move.l #3,revnum       * troisième inversé
          move.l #foincjmp,incvar *
          move.l #fodecjmp,decvar
          jsr    dispmen
          jsr    cursmess
          move.l #drques1,a0
          jsr    printf

```

rts

```

#####
*   Le sous-menu du menu format met dans les variables, les adresses   *
*   de menu GAP                                                         *
#####

```

```

gogaps: jsr      clear
        move.l   #mengap,menuadr
        move.l   #7,ganz          * Sept options de menu
        move.l   #1,revnum
        move.l   #gpincjmp,incvar
        move.l   #gpdecjmp,decvar
        jsr      dispmen
        jsr      cursmess
        move.l   #gpques1,a0
        jsr      printf
        rts

```

```

#####
*   Met dans les variable du système Menuselect, les adresses de      *
*   l'option OPTION                                                    *
#####

```

```

goinit: move.l   #6,ganz          * Le menu Init a
        move.l   #4,revnum        * six options
        move.l   #inincjmp,incvar
        move.l   #indecjmp,decvar
        move.l   #meninit,menuadr
        jsr      dispmen
        jsr      cursmess
        move.l   #driques1,a0
        jsr      printf
        rts

```

```

#####
#####

```

```

* voici les premières routines du menu SECTOR
*
*
*

```

```

*
*
* Incrémentation du numéro de lecteur dans l'option
*
*

```

```

incdrive: move.w    wdrive,d0      * Lecteur actif
          cmp.w     maxdriv,d0    * Comparer avec maxdrive
          blt       incdr1        * Si inférieur alors incrémenter
          move.w    #0,d0         * Sinon lecteur actif, nul
          bra       incdr2
incdr1:   addq.w    #1,d0
incdr2:   move.w    d0,wdrive      * mémoriser de nouveau
          add.b     #'0',d0       * et mettre dans le menu
          move.b    d0,wdrive     *
          jsr       dispmen       * L'afficher
          rts                * et retour

```

```

*
*
* Décrémente le numéro de lecteur dans le menu, les sous-programmes *
* qui suivent fonctionnent de la même manière   inctrack, incside *
*

```

```

decdrive: move.w    wdrive,d0      * Décrémenter lecteur courant
          cmp.w     #0,d0
          ble       decdr1
          subq.w    #1,d0
          bra       decdr2
decdr1:   move.w    maxdriv,d0
decdr2:   move.w    d0,wdrive
          add.b     #'0',d0
          move.b    d0,wdrive
          jsr       dispmen
          rts

```

```

*

```

```

incside:  move.w    wside,d0      * Face courante
          cmp.w     #1,d0         * Egalé un ?
          blt       incsi1        * Si oui alors
          move.w    #0,d0         * Mettre face 0
          bra       incsi2
incsi1:   move.w    #1,d0         * Sinon face 1
incsi2:   move.w    d0,wside      * Et mémoriser
          add.b     #'0',d0       * Et mettre dans chaîne de menu
          move.b    d0,mside
          jsr       dispmen       * Afficher menu
          rts                 * et retour

```

```

decside:  move.w    wside,d0      * Décrémenter face
          cmp.w     #0,d0
          ble       decsi1
          move.w    #0,d0
          bra       decsi2
decsi1:   move.w    #1,d0
decsi2:   move.w    d0,wside
          add.b     #'0',d0
          move.b    d0,mside
          jsr       dispmen
          rts

```

#####

```

inctrack: move.w    wtrack,d0     * Incrémenter piste. Comparer piste
          cmp.w     maxtrack,d0  * courante avec maxtrack
          blt       inctr1       * Si inférieure alors continuer
          move.w    #0,d0        * Sinon piste courante sur 0
          bra       inctr2
inctr1:   addq.w    #1,d0         * Additionner 1
inctr2:   move.w    d0,wtrack     * Et mémoriser
          ext.l     d0
          divu      #10,d0        * Mettre dans menu
          add.b     #'0',d0       * binaire -> ascii
          move.b    d0,mtrack     * High-Byte
          swap      d0
          add.b     #'0',d0

```

```

        move.b    d0,mtrack+1    * Low-Byte
        jsr      dispmen        * Afficher
        rts

dectrack: move.w    wtrack,d0    * decremonter piste
        cmp.w     #0,d0        * Piste courante égale 0
        ble      dectr1
        subq.w    #1,d0
        bra      dectr2
dectr1:  move.w     maxtrack,d0   * Alors piste courante = maxtrack
dectr2:  move.w     d0,wtrack
        ext.l     d0
        divu     #10,d0
        add.b     #'0',d0
        move.b    d0,mtrack
        swap      d0
        add.b     #'0',d0
        move.b    d0,mtrack+1   * Mettre dans chaîne de menus
        jsr      dispmen        * Afficher et retour
        rts

```

#####

```

incsect: move.w     wsector,d0   * incrementer secteur courant
        cmp.w     maxsect,d0    * voir inctrack
        blt      incsel
        move.w    #0,d0
        bra      incse2
incsel:  addq.w     #1,d0
incse2:  move.w     d0,wsector
        ext.l     d0
        divu     #10,d0
        add.b     #'0',d0
        move.b    d0,msector
        swap      d0
        add.b     #'0',d0
        move.b    d0,msector+1
        jsr      dispmen
        rts

```

```

decsect: move.w    wsector,d0      * decramenter secteur courant
        cmp.w     #0,d0
        ble       decsel
        subq.w    #1,d0
        bra       decse2
decsel:  move.w    maxsect,d0
decse2:  move.w    d0,wsector
        ext.l     d0
        divu     #10,d0
        add.b     #'0',d0
        move.b    d0,wsector
        swap     d0
        add.b     #'0',d0
        move.b    d0,wsector+1
        jsr      dispmen
        rts

```

```

#####
* Lit le secteur courant dans wsector, si drbyte = 1024 alors      *
* lecture de 1024 octets car le système d'exploitation ne calcule *
* avec le nombre de secteurs, que le nombre d'octets à lire. Pour *
* cela, il multiplie le nombre de secteurs par 512.              *
* Si on indique 2 Secteurs, il lit 1024 octets, peut importe qu'ils *
* soient dans un secteur de 1024 octets ou dans 2 secteurs de 512 *
* ou dans 4 de 256 octets.                                         *
* Voir LA BIBLE DE L'ATARI ST                                     *
#####

```

```

readsec: move.w    drbyte,d0      * Nombre Byte/Sector
        move.w    #1,d1          * 1 secteur par défaut
        cmp.w     #1024,d0       * si drbyte = 1024, alors
        bne       readsuit       * lire un secteur de 1024
        move.w    #2,d1          * octets
readsuit: move.w    d1,-(a7)      * Nombre de secteurs
        move.w    wside,-(a7)    * face
        move.w    wtrack,-(a7)  * Track
        move.w    wsector,-(a7) * Secteur ou Startsector

```

```

        move.w    wdrive,-(a7)      * Drive
        clr.l     -(a7)             * Met long dummy
        move.l    #platctr,-(a7)    * bufferadress
        move.w    #8,-(a7)          * floprd
        trap      #14               * XBIOS-Call
        add.l     #20,a7            * Restaurer pile
        tst.w     d0                * Y-a-t-il eu une erreur ?
        bmi       readser           * Si oui, alors message
        jsr       showsec           * Sinon, afficher secteur lu
        rts                          * et retour

```

```

readser: move.w    d0,-(a7)          * numéro d'Erreur sur pile
        jsr       errhand           * Traiter erreur
        jsr       cursmess          *
        move.l    #seques!,a0
        jsr       printf
        rts                          * Et retour

```

```

#####
* Affiche le secteur à l'écran. Utilise la routine showit      *
* Qui affiche tout ce qui lui est envoyé. Voir aussi          *
* editsec                                                       *
#####

```

```

showsec: move.w    #0,head2          * Pointeur dans secteur
        move.l     editptr,topptr    * Pointeur sur début de secteur
        move.w     #31,prcount       * Compteur pour l'impression
        move.w     #18,ligcount      * Nombre de lignes affichées
        move.w     #0,maxdown        * Scrolldown-Flag
        move.w     #208,maxup        * Scrollup-Flag
        move.w     drbyte,d0         * Bytes in Sector/ du menu Gap
        cmp.w      #1024,d0          * Si il y en a 1024 alors
        bne        showse2
        move.w     #512,maxdown      * positionner Scrollup Scrolldown-Flag
5
        move.w     #720,maxup        * En fonction
        move.w     #63,prcount       * Et modifier compteur d'impression
showse2: jsr       showit
        rts

```



```

#####
*
* Routine d'affichage universelle qui saisit le tampon clavier,
* effectue un scrolling up et down et teste la touche 'p' en vue
* d'une impression. Un pointeur sur le début du secteur mémoire à
* afficher est envoyé, ainsi que les limites supérieure et inférieure
*
#####

```

```

showit:  jsr      cursbuf
         jsr      videbuff      * Vider tampon clavier
         move.w   #0,head2      * Pointeur sur secteur
showit3:  move.w   head2,head1   * Ecrire ce buffer en fonction
         jsr      dispbuff      * du pointeur sur routine dispbuff
         jsr      videbuff      * Vider tampon clavier
         jsr      cursbuf
showit4:  jsr      touch         * Test du clavier
         swap     d0
         cmp.b    #$19,d0       * Teste si touche 'p' pressée
         beq      printit       * si oui, impression sur imprimante
         cmp.b    #$48,d0       * Teste si Cursor-up
         beq      upper         * Si oui, traitement
         cmp.b    #$50,d0       * Teste si Cursor-down
         beq      lower         * Si oui, traitement
         cmp.b    #$1c,d0       * Teste si 'RETURN'
         beq      shsecli
         cmp.b    #$4b,d0       * Teste si Cursor-left
         beq      shsecli       * Si oui
         cmp.b    #$4d,d0       * Teste si Cursor-right
         bne      showit4       * sinon, retour à boucle de test
         jsr      curright      * sinon Cursor-right, enfin retour
         bra      showiten      * A la routine d'appel
shsecli: jsr      curleft       * Appeler Cursor-left et retour
showiten: rts                 * a la routine d'appel

```

```

upper:   move.w   head2,d0      * Traite Cursor-up

```

```

        cmp.w    #0,d0          * Met le pointeur sur début du secteur
        beq      uppend         * Puis ne fait rien
        cmp.w    maxup,d0       * S'il pointe sur la limite supérieure
        beq      upper1         * alors soustraire 208 pour comparer
        sub.w    #256,head2     * sinon soustraire 256
        sub.l    #256,topptr    * du pointeur secteur et du compteur
        bra      uppend         * et retour
upper1:  sub.w    #208,head2     * soustraire 208 du pointeur sur
        sub.l    #208,topptr    * secteur et comparer au secteur
uppend:  bra      showit3       * Puis retour pour affichage

```

```

lower:   move.w   head2,d0      * Traitement du Cursor-down
        cmp.w    maxup,d0      * Comme ci dessus mais additionner
        beq      lowend        * Au pointeur et au compteur
        cmp.w    maxdown,d0
        bne      lower1
        add.w    #208,head2
        add.l    #208,topptr
        bra      lowend
lower1:  add.w    #256,head2
        add.l    #256,topptr
lowend:  bra      showit3       * Continuer affichage

```

```

#####
*
* Imprime le contenu du tampon sur topptr en tant que 16 nombres hexa
* sur 2 chiffres et 16 chiffres ASCII . Le nombre
* de lignes à imprimer est mis dans prcount
*
#####

```

```

printit: move.w   #0,device     * conout sur imprimante
        movem.l   a3-a5/d3-d7,savereg * Sauvegarder Register
        move.l    #m1secta,a5   * Imprimer
        move.w    #45,d7        * Track, Sector et face
printit0: move.b   (a5)+,d0
        move.w    d0,-(a7)
        jsr      conout         * Imprimer

```

```

        dbra      d7,printit0
        move.l    #miclusa,a5          * Imprimer Clusternumber
        move.w    #13,d7
printit1: move.b   (a5)+,d0
        move.w    d0,-(a7)
        jsr       conout
        dbra      d7,printit1
        jsr       crlinef             * Carriage-Ret. + Line-Feed
        jsr       crlinef             * 2 fois
        move.l    topptr,a4           * Mémoriser Pointeur sur secteur
        move.l    a4,a5               *
        move.w    head2,head1         * Compteur courant
        move.w    #15,d3              * Compteur de colonnes correspond à 16
        move.w    d3,d4               * mémoriser
        move.w    prcount,d5          * Nombre de lignes à imprimer
printit2: move.w   d4,d3              * Nombre de lignes
        jsr       header              * Impression de l'octet de comptage
        jsr       hex16               * Impression de 16 octets hexa
        move.w    d4,d3               * restaurer pile
        move.l    a5,a4               * Pointeur de nouveau sur début secteur
        move.w    #5,d7               * Insérer 5 espaces
printit3: move.w   #$20,-(a7)
        jsr       conout              * Impression
        dbra      d7,printit3
        jsr       char16              * Impression de 16 caractères ASCII
        add.l     #16,a5               * Adapter pointeur sur secteur
        add.w     #16,head1
        jsr       crlinef             * Nouvelle ligne
        dbra      d5,printit2         * Jusqu'à ce que toutes les lignes
        jsr       videbuff            * aient été imprimées
        move.l    savereg,a3-a5/d3-d7 * Rappel du Registre
        move.w    #2,device           * Impression de nouveau sur écran
        bra       showit4             * Et affichage du secteur

```

```

printerr: rts

```

```

#####
* Met en Sector Mode sous Edit Mode, en appelant la routine          *
* editit très flexible. On donne à cette routine seulement un        *
* pointeur sur le début du secteur mémoire à éditer et deux          *

```

```

* variables de limites.
*****

editsec: jsr      cursmess
         move.l    #edques1,a0      * AFFICHER MESSAGE D'ÉDITION
         jsr      printf
         move.w    #0,column
         move.w    #4,ligne
         jsr      loccurs
         jsr      clrest             * Effacer reste de l'écran
         move.w    drbyte,d0
         cmp.w     #1024,d0          * si 1024 Bytes/Sector alors
         bne      edsewei1
         move.w    #512,maxdown      * Sélectionner limite sup
         move.w    #720,maxup        * afin de pouvoir éditer également
         bra      edsewei2           * ces 1024 octets

edsewei1: move.w   #0,maxdown         * Sinon sélectionner limit inf.
         move.w   #208,maxup          *
edsewei2: move.w   #18,ligcount       * Afficher 19 lignes
         move.l   #platztr,editptr    * bufferadress
         jsr      editit              * et éditer
         jsr      curleft             * Enfin menu sector
         jsr      curleft             * Remettre sur lecture
         move.w   #2,ligne
         jsr      loccurs
         jsr      delline             * Effacer ligne
         jsr      cursmess
         move.l   #seques1,a0         * Afficher message
         jsr      printf
         jsr      cursoff             * désactiver curseur
         jsr      showsec             * Et afficher de nouveau le secteur
         jsr      videbuff            * vider tampon clavier
         rts                          * et retour

```

```

*****
* Voici la routine d'édition qui peut éditer autant de lignes de
* 16 Byte que voulu et afficher ces lignes sous forme hexadécimale
* ou ascii
*****

```

```

editit: movem.l    a3-a6/d3-d7,-(a7)    * Mémoriser registres
        move.l     editptr,topptr      * bufferadress
        move.w     #0,head2            * initialiser pointeur pour tampon
        move.w     #0,head1
        jsr        dispbuf             * Afficher première face de tampon
        jsr        videbuff            * vider tampon clavier
        move.w     #7,column           * Edit commence en colonne 7
        move.w     #4,ligne
        jsr        loccurs              * positioner curseur
edit0:  jsr        curson                * Activer curseur clignotant
        move.l     #retw1,-(a7)         * Donner adresse d'une variable à
        jsr        hexon                * hexon Le chiffre indiqué est dans
        jsr        cursoff              * cette adresse.
        tst.w     retw1                 * Si le nombre était négatif
        bmi        otherkey            * Alors presser autre touche
        move.w     ligne,d0             * Ligne courante
        subq.w     #4,d0                 * Startoffset de la première ligne
        lsl.w     #4,d0                 * 4016 caractères/ligne
        move.w     column,d2            * plus colonne - Startoffset
        sub.w     #7,d2
        ext.l     d2                    * divisé par 3 caractères par
        divu     #3,d2                  * Byte (1 Space + 2 Digits)
        add.w     d2,d0                 * plus fin de ligne
        move.w     retw1,d1             * Chiffre hexa indiqué
        move.l     topptr,a3            * Startadress du tampon
        move.b     d1,0(a3,d0.w)        * Mettre chiffre dans tampon
        jsr        displine             * avec fin comme pointeur. Afficher lig

        cmp.w     #52,column           * Etait-ce le dernier Editbyte
        blt       edits1                * de la ligne ?
        move.w     #4,column            * Si oui, retour en début de ligne
edit1:  addq.w     #3,column             * sinon additionner 3 caractères / octe
t
        jsr        loccurs              * positioner curseur
        bra       edits0                * Et continuer édition

```

```

otherkey: move.l    var11,d0            * On détermine si aucun chiffre

```

```

swap      d0                * valable n'a été donné
cmp.b     #$4b,d0           * Cursor left?
beq        oleft            * oui traiter
cmp.b     #$4d,d0           * cursor right
beq        oright
cmp.b     #$50,d0           * Cursor down
beq        odown
cmp.b     #$48,d0           * cursor up
beq        oup
cmp.b     #$52,d0           * Insert
beq        edend1
cmp.b     #$72,d0           * Enter
beq        edend1           * Quitter mode edit
cmp.b     #$1c,d0           * Return quitte le mode
beq        edend1           * Edit
jsr        displine         * Sinon afficher ligne
jsr        loccurs           * positionner curseur
bra        edits0           * Et continuer edition
oleft:     move.w    column,d0 * Cursor left
cmp.w     #7,d0             * Cursor déjà à gauche
bgt        oleft1           * sinon, continuer
move.w    #55,column        * si oui, wrap around
oleft1:    subq.w     #3,column * sinon, 3 caractères/octet
jsr        loccurs           * soustraire, positionner curseur
jsr        videbuff         * Vider tampon clavier
bra        edits0           * Et continuer edition

oright:    move.w    column,d0 * La même chose en vert pour cursor
cmp.w     #52,d0            * right
blt        oright1
move.w    #4,column
oright1:   addq.w     #3,column
jsr        loccurs
jsr        videbuff
bra        edits0

*****

odown:     jsr        cursoff    * Cursor down
move.w    ligne,d0            * Ligne courante

```

```

      cmp.w    #22,d0          * inférieure à 22
      blt      odown2         * Si oui continuer
      move.w   head2,d0       * Sinon comparer compteur avec
      cmp.w    maxdown,d0     * Limite
      bne      odown1         * Si différents continuer
      add.w    #208,head2      * si égaux traiter
      add.l    #208,topptr     * traiter le reste du tampon.
      move.w   column,oldspal  * Donc additionner 208 au lieu de 256
      jsr      dispbuf        * Afficher d'abord tampon
      move.w   oldspal,column  * Cursor dans ancienne colonne
      move.w   #5,ligne       * Offset dans Buffer
      jsr      loccurs        * positionner curseur
      bra      odownend       * et retour
odown1: cmp.w   maxup,d0      * Si égal limite sup.
      beq      odownend       * Puis ne rien faire
      add.w    #256,head2      * sinon additionner 256
      add.l    #256,topptr     * aux pointeurs
      move.w   column,oldspal
      jsr      dispbuf        * Et afficher tampon
      move.w   oldspal,column
      move.w   #6,ligne
      jsr      loccurs
      bra      odownend       * et retour
odown2: addq.w  #1,ligne      * si pas en ligne 22, alors
      jsr      loccurs        * incrémenter ligne courante de 1
odownend: jsr   videbuff      * vider tampon clavier et
      bra      edits0         * continuer edition

```

```

oup:   jsr      cursoff       * idem que pour Cursor-down
      move.w   ligne,d0      * mais pour Cursor up
      cmp.w    #4,d0         * Ligne courante = ligne 4
      bne      oup2          * Sinon continuer
      move.w   head2,d0      * Si oui charger compteur
      cmp.w    #0,d0         * Limite sup. du tampon Edit?
      beq      oupend        * Si oui ne rien faire
      cmp.w    maxup,d0      * Si non comparer avec limite
      beq      oup1          * Si égal soustraire seulement 208
      sub.w    #256,head2     * sinon soustraire 256 des

```

```

sub.l    #256,topptr    * pointeurs
move.w   column,oldspal * Charger ancienne colonne
jsr      dispbuf        * afficher tampon
move.w   oldspal,column * Curseur dans ancienne colonne
move.w   #19,ligne      * Offset dans ligne
jsr      loccurs        * positionner curseur
bra      oupend         * Et fin
oup1:    sub.w   #208,head2 * Afficher reste sup. du tampon
sub.l    #208,topptr
move.w   column,oldspal
jsr      dispbuf
move.w   oldspal,column
move.w   #19,ligne
jsr      loccurs
oup2:    subq.w   #1,ligne    * Si pas dans ligne sup.
jsr      loccurs            * decrements ligne
oupend:  jsr      videbuff   * Vider tampon clavier
bra      edits0            * Et continuer edition
edend!:  move.w   #0,column  * Arrêter edition
move.w   #4,ligne          * positionner curseur en ligne 4
jsr      loccurs
movem.l  (a7)+,a3-a6/d3-d7 * retour du Registre
rts      * Et retour

```

```

writsec: movem.l  a3-a6/d3-d7,-(a7) * Ecrit un secteur sur disquette
move.w   #0,column    * Demander s'il doit vraiment
move.w   #2,ligne     * être écrit
jsr      loccurs
move.l   #nrques1,a0
jsr      printf
move.l   #n1secta,a3  * Track courante, Sector etc.
move.w   #45,d3       * Afficher à l'écran
writl1:  move.b   (a3)+,d0
move.w   d0,-(a7)
jsr      conout
dbra     d3,writl1

```



```

        move.l    #wrques2,a0
        jsr      printf
        jsr      videbuff      * Vider tampon clavier et
        jsr      wtast         * Attendre touche
        cmp.b    #'y',d0       * Si ni 'y' ni 'Y'
        beq      writit        * Alors ne pas écrire
        cmp.b    #'Y',d0
        bne      wrend1        * Aller à la fin

writit: move.w    drbyte,d0     * si drbyte = 1024, alors utiliser
        cmp.w    #1024,d0      * la routine personnelle writesec-
        beq      selfsect      *
        move.w    #1,d1        * Sinon écrire un secteur
writit: move.w    d1,-(a7)      * Nombre
        move.w    wside,-(a7)  * face
        move.w    wtrack,-(a7) * Track
        move.w    wsector,-(a7) * Sector
        move.w    wdrive,-(a7) * Drive
        clr.l     -(a7)        * Mot long Dummy
        move.l    #platztr,-(a7) * bufferadress
        move.w    #9,-(a7)     * flopwr
        trap      #14          * XBIOS-Call
        add.l     #20,a7
        tst.w     d0            * Teste si erreur
        bni      writerr       * Traiter erreur

wrend2: jsr      delline
        jsr      videbuff
        jsr      cursness      * Afficher Mode
        move.l    #seques1,a0
        jsr      printf
        movem.l   (a7)+,a3-a6/d3-d7
        rts              * Et retour

wrend1: jsr      delline
        move.l    #wrques3,a0   * message = "not written"
        jsr      printf
        jsr      videbuff
        jsr      wtast
        bra      wrend2

```

```

writerr: move.w    d0, -(a7)          * Traiter erreur
        jsr      errhand
        bra      wrend2

```

```

#####
#####
*   Début de la programmation assembleur modulaire. Saisissez      *
*   les noms de routines avec RTS Seul le menu sector fonctionnera *
*   Dans le programme. Vous pouvez donc lire et écrire des      *
*   secteurs. Si ce menu fonctionne sans erreur, vous pouvez      *
*   Intégrer les autres sous-programmes complets dans le listing  *
*   Intégrez de préférence le bloc de commande complet correspondant *
*   à une option de menu car la plupart des options font accès    *
*   aux routines des autres options. Il faut faire un peu        *
*   attention à l'ordre de l'implémentation                        *
*                                                                    *
*   Je vous conseille l'ordre suivant : d'abord OPTION, puis      *
*   TRACK, Enfin TRACK with SYNCs, FORMAT et CLUSTER             *
#####
#####

```

```

#####
#####
*   Subroutines de l'option INIT, doivent d'abord être implémentées *
*   car elles permettent de lire le 10ième Secteur sur la piste    *
*   B2 Track etc. De plus, certaines routines sont appelés par    *
*   d'autres parties de programme.                                  *
#####
#####

```

```
incmaxtr:  rts
```

```
decmaxtr:  rts
```

```
incmaxse:  rts
```

decmaxse: rts

dodrivin: rts

showbpb: rts

initdriv: rts

rdfat: rts

selfsect: rts

readltr: rts

incstra: rts

decstra: rts

edittr: rts

showtr: rts

writltr: rts

rdtracks: rts

shtracks: rts

readadr: rts

showadr: rts

edclust: rts

decclust: rts

incclust: rts

nextclst: rts

wrclust: rts

rdclust: rts

stclust: rts

format1: rts

xformat: rts

incgap1: rts

incgap2: rts

incgap3: rts

incgap4: rts

incgap5: rts

decgap1: rts

decgap2: rts

decgap3: rts

decgap4: rts

decgap5: rts

incbyte: rts

decbyte: rts

```

#####
#####
*
* Voici quelques sous programmes souvent utilisés
*
#####
#####

```

```

#####
* dessine une ligne hizintale de 0,20 à 639,20
*
#####

```

```

hline:  move.l  lineavar,a0      # pointeur sur variables du Line-A
        move.w  #0,38(a0)      # X1
        move.w  #20,40(a0)     # Y1
        move.w  #639,42(a0)
        move.w  #1,24(a0)      # couleur
        move.w  #0,36(a0)      # Write-Mode
        move.l  #pattern,46(a0) # motif + nombre de mots de motif
        move.w  #0,50(a0)      # Horizontal Line
        dc.w    $a004
        rts

```

```

#####
* Ecrit une chaîne sur l'écran
*
#####

```

```

printf: move.l  a0,-(a7)        # Ecrit la chaîne dont l'adresse
        move.w  #9,-(a7)        # de début se trouve dans le
        trap    #1              # registre d'adresse A0
        addq.l  #6,a7           # La chaîne doit se terminer par
        rts                    # un 0

```

```

#####
* Initialise les variables du Line-A et mémorise l'adresse du bloc
* dans variables dans "lineavar".
*
#####

```

```

inlinea: dc.w    $a000          # Initialise les variables du Line-A

```

```

move.l    a0,lineavar    * Mémoriser Adresse
move.w    #0,32(a0)
move.w    #$fff,34(a0)   * Motif de la ligne
move.w    #0,36(a0)      * Writing mode = replace
move.w    #1,24(a0)      * Couleur d'écriture
rts

```

```

*****
* Initialiser Line-A-
*****

```

```

start1:   jsr      inlinea      * Initialiser Line A
          rts

```

```

*****
*
* Ecrit un nombre sur la pile sous forme de mot et sur l'écran
* en hexadécimal sur 2 chiffres ou sur le périphérique défini
* par le contenu de device
*
*****

```

```

hexpr:    move.w    4(a7),d1      * Récupérer Argument sur la pile
          and.w     #$00ff,d1     * Masquer Highword
          move.w    d1,varw1      * Mémoriser Byte
          lsr.w     #4,d1         * Décaler nibble inférieur
          ext.w     d1            * Mettre sous forme de mot
          and.w     #$00ff,d1     * Masquer high-Byte
          cmp.w     #9,d1         * Supérieur à 9 donc
          bgt       ischar1       * Imprimer un caractère 'A'-'F'
          jsr       hexdig        * Sinon un chiffre '0'-'9'
          bra       secdig1
ischar1:  jsr       hexchar
secdig1:  move.w    varw1,d1      * Puis transformer le Nibble inf, low-Byte
          and.w     #$000f,d1     * Masquer le supérieur,
          cmp.w     #9,d1         * voir plus haut
          bgt       ischar2

```

```

        jsr      hexdig
        bra      hexpren      * Vers la fin

ischar2: jsr      hexchar
hexpren: move.l   (a7)+,a0      * Adresse de retour sur la pile
        add.l    #2,a7         * Libérer la pile
        jmp      (a0)         * et retour à l'appel comme rts

hexdig:  add.w    #48,d1        * additionner valeur ASCII de '0'
        move.w   d1,-(a7)      *
        move.w   device,-(a7)  * et imprimer
        move.w   #3,-(a7)      * conout
        trap     #13           * BIOS-TRAP
        addq.l   #6,a7
        rts

hexchar: sub.w    #10,d1        * Soustraire 10 et additionner valeur ASCII
        add.w    #65,d1        * de 'A'
        move.w   d1,-(a7)
        move.w   device,-(a7)  * et imprimer
        move.w   #3,-(a7)
        trap     #13
        addq.l   #6,a7
        rts

#####
*   Impression d'un INTEGER sur 2 octets en décimal   *
#####

dezpr:  move.w   #0,dflag      * Impression d'un INTEGER sur 2 octets
        move.w   4(a7),d3      * Les 0 sont ignorés et affiché sous
        ext.l    d3            * la forme d'espaces
        divs     #10000,d3
        beq      dezpr1       *
        move.w   #-1,dflag     * Flag sortie, pas
dezpr1: jsr      deznum        * d'impression
        swap     d3            * diviser reste de la lière division

```

```

        ext.l    d3          * par 1000
        divs     #1000,d3    * Et imprimer à la place des 1000 ièmes
        beq      dezpr3
dezpr2: move.w   #-1,dflag    * Différent de 0 alors positionner flag
dezpr3: jsr      deznum
        swap     d3
        ext.l    d3          * diviser le reste par 100
        divs     #100,d3
        beq      dezpr4
        move.w   #-1,dflag
dezpr4: jsr      deznum        * et imprimer
dezpr5: swap     d3
        ext.l    d3
        divs     #10,d3      * Diviser le reste par 10
        beq      dezpr7
        move.w   #-1,dflag
dezpr7: jsr      deznum        * Et imprimer à la place des 10ièmes
        swap     d3          * Imprimer le reste de la dernière
        move.w   #-1,dflag    * division car même si on a un 0,
        jsr      deznum        * il n'est pas imprimé en tant que tel
        move.l   (a7)+,a0      *
        addq.l   #2,a7         * Récupérer adresse de retour dans A0
        jmp      (a0)         * Restaurer pile et retour à l'appel

deznum: tst.w    dflag        * Imprime un chiffre '0'-'9'
        bne      deznum1      * mais seulement si dflag différent de 0
        move.b   #' ',d0      *
        move.w   d0,-(a7)
        bra      deznum2
deznum1: add.b   #'0',d3       * Addition DE LA VALEUR ASCII de '0'
        move.w   d3,-(a7)
deznum2: jsr     conout        * Imprimer
        rts

```

```

dezlpr:  move.w    #0,dflag      * Imprimer un INTEGER sur 4-Byte
        move.l    4(a7),d3      * qui est envoyé sur la pile en tant que
        move.l    d3,d4        * mot, sous forme décimale.
        divs      #10000,d3     * Les 0 au début ne sont
        ext.l     d3           * pas pris en compte on imprime
        divs      #10,d3       * des spaces à la place.
        move.w    d3,d5        * diviser d'abord par 100000
        tst.w     d3           * Si 0 alors par de 100000 ièmes
        beq       dezlpr1
        move.w    #-1,dflag
dezlpr1:  jsr      deznum        * Sinon imprimer les 100000ièmes
        move.w    d5,d3        * Multiplier le resultat de la division
        muls      #10,d3       * par 100000. Traiter le reste
        muls      #10000,d3    * comme avec dezpr
        sub.l     d3,d4        *
        move.l    d4,d3
        divs      #10000,d3
        beq       dezlpr3
dezlpr2:  move.w    #-1,dflag
dezlpr3:  jsr      deznum
        swap      d3
        ext.l     d3
        divs      #1000,d3
        beq       dezlpr4
        move.w    #-1,dflag
dezlpr4:  jsr      deznum
        swap      d3
        ext.l     d3
        divs      #100,d3
        beq       dezlpr5
        move.w    #-1,dflag
dezlpr5:  jsr      deznum
        swap      d3
        ext.l     d3
        divs      #10,d3
        beq       dezlpr6
        move.w    #-1,dflag
dezlpr6:  jsr      deznum
        swap      d3
        move.w    #-1,dflag

```

```

jsr      deznum
move.l   (a7)+,a0
addq.l   #4,a7
jmp      (a0)

```

```

#####
*   Affichage du menu                                   *
#####

```

```

dispmen: move.w #0,column      * Curseur sur ligne supérieure
         move.w #0,ligne
         jsr     loccurs       *
         move.l  menuadr,a6    * Il y a ici un pointeur sur
         move.l  revnum,d6     * les adresses des chaînes de
         subq.l   #1,d6        * menus. Le numéro de l'option en
         beq      dispweil     * inversé se trouve dans
         subq.l   #1,d6        * revnum
dispmen1: move.l  (a6)+,a0      * Récupérer les adresses et
         jsr      printf       * imprimer avec printf jusqu'à ce
         dbra     d6,dispmen1  * ce que toutes les chaînes jusqu'à la chaî
n
dispweil: jsr     revon        * inversée soient affichée. puis
         move.l  (a6)+,a0      * afficher chaîne inversée
         jsr      printf       * redésactiver inversé
         jsr      revout
         move.l  ganz,d7       * Nombre total de toutes les options
         sub.l   revnum,d7     * moins nombre inversé
         beq      dispmen3     * Si 0 c'était le dernier
         subq.l   #1,d7        * Sinon afficher les autres
dispmen2: move.l  (a6)+,a0      * options en normal
         jsr      printf
         dbra     d7,dispmen2  * Toutes les options
dispmen3: jsr      hline       * Puis dessiner une ligne
         jsr      delrest      * horizontale et retour
         rts

```

```

#####
*

```

```

* Ecrit le contenu d'un secteur mémoire dont l'adresse de début *
* est dans topptr, en tant que 16 hexa sur 2 chiffres *
* et les 16 ASCII correspondant sur l'écran *
*****

```

```

dispbuf: movem.l a3-a5/d3-d7,savereg * Mémoriser registre
        move.l topptr,a4 * mémoriser Adresse de début du
        move.l a4,a5 * secteur mémoire
        move.w head2,head1 * compteur pour offset de Block
        move.w #15,d3 * compteur de colonnes = 16 colo.
        move.w d3,d4 * mémoriser
        move.w ligcount,d5 * Envoyer nombre de lignes
        move.w #4,ligne * dans ligcount
        move.w #4,curlin1 * Cursor sur ligne 4 colonne 0
        move.w #0,column
        jsr loccurs *

```

```

dispb1: move.w #0,coluan * Cursor ligne courante
        move.w curlin1,ligne *
        move.w d4,d3 * compteur de colonnes
        jsr loccurs
        jsr header * Imprimer compteur dans bloc
        jsr hex16 * Imprimer 16 hexas
        move.w d4,d3 * compteur de colonnes
        move.l a5,a4 * topptr
        move.w #59,column * caractères ASCII en colonne 59
        move.w curlin1,ligne
        jsr loccurs
        jsr char16 * Imprimer 16 caractères Ascii
        add.l #16,a5 * Additionner 16 au pointeur en mém
        add.w #1,curlin1 * Continuer en ligne suivante
        add.w #16,head1 * additionner 16 au compteur
        dbra d5,dispb1 * jusqu'à ce que toutes les lignes
        jsr videbuff * imprimées. vider tampon clavier
        movem.l savereg,a3-a5/d3-d7 * et Récupérer registre
        rts

```

```

*****
* Ecrit un Header avant chaque ligne *

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

header:  move.w    head1,d6          #
         lsr.w     #8,d6             # diviser compteur par 256 (High-Byte)
         move.w    d6,-(a7)         # et imprimer sous forme d'hexa
         jsr       hexpr
         move.w    head1,-(a7)      # Imprimer Low-Byte
         jsr       hexpr
         move.b     #' ',d6         # Imprimer ":"
         move.w    d6,-(a7)
         jsr       conout
         rts

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#   Ecrit 16 hexadécimaux
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

hex16:   move.w    #$20,-(a7)       # Imprimer 2 espaces
         jsr       conout
         move.w    #$20,-(a7)
         jsr       conout
hex161:  move.b     (a4)+,d7         # Imprimer le contenu du secteur mém
         move.w    d7,-(a7)         # en hexa
         jsr       hexpr           # avec un espace tous les 16
         move.w    #$20,-(a7)
         jsr       conout          # le compteur est mis dans d3
         dbra      d3,hex161
         rts

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#   Ecrit 16 caractères ASCII à l'écran
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

char16:  move.b     #' ',d7         # d'abord un ":"
         move.w    d7,-(a7)
         jsr       conout
         move.w    #$20,-(a7)       # puis deux espaces
         jsr       conout
char161: move.b     (a4)+,d7         # et 16 caractères ASCII

```

```

cmp.b    $$20,d7      * Imprimer un point pour tout ce qui
bgt      char162      * est inférieur à $20
move.b   #' ',d7
char162: ext.w    d7      * sinon masquer High-Byte
and.w    #$00ff,d7
move.w   d7,-(a7)
jsr      cout      * et imprimer
dbra     d3,char161  * mettre 11 fois dans d3
rts

```

```

#####
*   Ecrit une ligne complète à l'écran   *
#####

```

```

displine: move.w  column,oldspal  * Une ligne en format 16/16
move.w   #0,column      *
jsr      loccurs      * positionner curseur
move.w   oldspal,column
move.w   #15,d3      * 16 colonnes
move.w   d3,d4
move.l   topptr,a4      * Pointeur sur début du secteur mém
clr.l    d0      * Calculer la position relative
move.w   ligne,d0      * au début du secteur avec la ligne
subq.w   #4,d0      * courante
lsl.w    #4,d0      * fois 16
move.w   d0,d1      * mémoriser
add.w    head2,d0      * Additionner compteur
move.w   d0,head1      * Mettre dans compteur courant
ext.l    d1
add.l    d1,a4      * Additionner au pointeur sur secteur
move.l   a4,a5      * mémoire et mettre pointeur
jsr      header      * sur la ligne en cours d'édition
jsr      hex16      * Imprimer 16 chiffres
move.w   column,oldspal
move.w   #59,column
jsr      loccurs
move.w   d4,d3
move.l   a5,a4      * et imprimer 16 ASCII's
jsr      char16

```

```

move.w    oldspal,column    ‡ Récupérer ancienne colonne
jsr       loccurs           ‡ positionner curseur
rts

```

```

#####
‡  Routine d'émulation de Terminal
#####

```

```

revon:    move.l    #revers1,a0    ‡ Invers ON
jsr       printf
rts

```

```

revout:   move.l    #revers2,a0    ‡ Invers OFF
jsr       printf
rts

```

```

delrest:  move.l    #clrest2,a0     ‡ Efface le reste de la ligne
jsr       printf
rts

```

```

delline:  move.l    #delline1,a0    ‡ Efface la ligne complète
jsr       printf
rts

```

```

clear:    move.l    #clear1,a0      ‡ Efface l'écran
jsr       printf                  ‡ et positionne le curseur
rts                                           ‡ dans le coin gauche en haut

```

```

home:     move.l    #home1,a0       ‡ positionne le curseur dans
jsr       printf                  ‡ le coin gauche en haut
rts

```

```

crlinef:  move.w    #$1c,-(a7)      ‡ envoie Carriage-Return et
jsr       conout                    ‡ Linefeed sur périph. de sortie
move.w    #$0a,-(a7)
jsr       conout
rts

```

```

clrest: move.l    #clrest1,a0      * Efface le reste de l'écran
        jsr      printf
        rts

curson:  move.l    #curon1,a0      * Cursor on
        jsr      printf
        rts

cursoff: move.l    #curout1,a0     * Cursor off
        jsr      printf
        rts

cursmess: move.w   #30,column      * positionne le Cursor
        move.w    #2,ligne        * Pour affichage
        jsr      loccurs
        rts

cursbuf: move.w    #0,column       * Positionne le curseur pour
        move.w    #4,ligne        * pour affichage du tampon de secteur
        jsr      loccurs
        rts

```

```

*****
*   positionnement du curseur                               *
*****

```

```

loccurs: move.l    #loccurs1,a0    * positionne le curseur sur
        addq.l    #2,a0            * les coordonnées de ligne et de
        move.w    ligne,d0        * colonne (0-79), (0-24)
        add.w     #32,d0          * additionner Offset interne
        move.b    d0,(a0)+        * mémoriser
        move.w    column,d0
        add.w     #32,d0          * Additionner Offset interne
        move.b    d0,(a0)+        * et mémoriser
        move.l    #loccurs1,a0    * Enfin imprimer
        jsr      printf          * instruction de positionnement
        rts

```

```

curstab: move.w    tab1,column    * Positionne le curseur
          jsr      loccurs        * en colonne tab1 de la ligne courante
          rts

```

```

#####
*   test du clavier. Pas d'attente, le code de la touche et le   *
*   code ASCII sont renvoyés dans D0. Si aucune touche n'est pressée *
*   D0 égale 0                                                    *
#####

```

```

touch:   move.w    #2,-(a7)        * Test du clavier
          move.w    #1,-(a7)        * Envoie code ASCII de la touche
          trap      #13             * dans Low-Byte du
          addq.l    #4,a7           * mot inférieur de D0 et le
          tst.w     d0              * Scan-Code dans Low-Byte du mot
          bpl       endtast2        * supérieur de D0
          move.w    #2,-(a7)        * Si touche pressée alors
          move.w    #2,-(a7)        * Récupérer touche dans tampon
          trap      #13             * et retour
          addq.l    #4,a7
          rts

```

```

endtast2: move.l    #0,d0          * Sinon renvoyer 0
          rts

```

```

videbuff: move.w    #0b,-(a7)      * vider tampon de clavier
          trap      #1
          addq.l    #2,a7
          tst.w     d0
          beq       vider1
          move.w    #7,-(a7)
          trap      #1
          addq.l    #2,a7           * Répéter jusqu'à plus de caractère
          bra       videbuff        * dans le tampon
vider1:   rts

```

```

conout:  move.w    4(a7),d0        * Envoie un caractère dans le
          move.w    d0,-(a7)        * périphérique indiqué par device
          move.w    device,-(a7)    * Voir LA BIBLE DE L'ATARI ST
          move.w    #3,-(a7)

```



```

trap      #13
addq.l    #6,a7
move.l    (a7)+,a0      ‡ Récupérer adresse de retour
addq.l    #2,a7
jmp       (a0)

wtast:     move.w    #1,-(a7)      ‡ saisie de clavier, attend la
trap      #1                ‡ saisie et affiche un curseur
addq.l    #2,a7              ‡ clignotant
rts

#####
‡ Saisie d'un hexa sur un octet à l'adresse spécifiée par la ‡
‡ pile ‡
#####

hexon:     jsr       wtast        ‡ Envoie un hexa saisi au clavier
move.l     d0,varl1             ‡ à la variable indiquée par la
cmp.b      #'f',d0              ‡ pile (2 digits)
bgt        hexeierr             ‡ Si touche interdite alors
cmp.b      #'a',d0              ‡ envoi de -1
blt        hexon1
sub.b      #'a',d0              ‡ Teste si entre 'a' et 'f'
add.b      #10,d0               ‡ si oui alors soustraire 'a' et
bra        hexon2               ‡ additionner 10
hexon1:     cmp.b     #'0',d0     ‡ Sinon, tester si
blt        hexeierr             ‡ entre 0 et 9
cmp.b      #'9',d0
bgt        bstest1              ‡ Sinon tester si ('A'-'F')
sub.b      #'0',d0              ‡ sinon soustraire '0'
hexon2:     lsl.w      #4,d0      ‡ fois 16 = High-nibble
move.w     d0,varw1             ‡ mémoriser
jsr        wtast                ‡ Récupérer nibble suivant
move.l     d0,varl1             ‡ mémoriser
cmp.b      #'f',d0              ‡ même test que premier Nibble
bgt        hexeierr
cmp.b      #'a',d0
blt        hexon3
sub.b      #'a',d0
add.b      #10,d0

```

```

        bra        hexon4
hexon3: cmp.b      #'0',d0
        blt        hexeierr
        cmp.b      #'9',d0
        bgt        bstest2          * test si majuscule
        sub.b      #'0',d0
hexon4: move.w     varw1,d1
        or.w       d1,d0
        ext.w      d0
        and.w      #$00ff,d0
        move.w     d0,varw2          * Retour sans erreur
hexon5: move.l     4(a7),a0
        move.w     d0,(a0)
        move.l     (a7)+,a0
        addq.l     #4,a7
        jmp        (a0)              * back to caller

hexeierr: move.w   #-1,d0
        bra        hexon5            * Retour avec code d'erreur

bstest1: cmp.b     #'F',d0            * Test si entre 'A' et 'F'
        bgt        hexeierr          * sinon retour avec code erreur
        cmp.b     #'A',d0            *
        blt        hexeierr          * sinon soustraire ascii 'A' et
        sub.b     #'A',d0            * additionner 10
        add.b     #10,d0
        bra        hexon2            * Récupérer nibble suivant

bstest2: cmp.b     #'F',d0            * Comme bstest1 pour
        bgt        hexeierr          * le deuxième Nibble
        cmp.b     #'A',d0
        blt        hexeierr
        sub.b     #'A',d0
        add.b     #10,d0
        bra        hexon4

```

```

*****
*****
* Variables du programme de base
*
```

```

#####
#####

```

```

#####
* données de menu pour le menu principal, adresses des chaînes de
* menu, adresses des Subroutines (haincjmp)
#####

```

data

```

haincjmp: dc.l    gotrack
           dc.l    gosync
           dc.l    gosector
           dc.l    goclust
           dc.l    goformat
           dc.l    goinit
           dc.l    endmain2

```

```

menmain: dc.l    mihaula
           dc.l    mihaula1
           dc.l    mihaulb
           dc.l    mihaulb1
           dc.l    mihaulc
           dc.l    mihauld1
           dc.l    mihaule

```

```

mihaula: dc.b    ' TRACK ',0
mihaula1: dc.b    ' TRACK/SYNCS ',0
mihaulb: dc.b    ' SECTOR ',0
mihaulb1: dc.b    ' CLUSTER ',0
mihaulc: dc.b    ' FORMAT ',0
mihauld: dc.b    ' FATS ',0
mihauld1: dc.b    ' OPTIONS ',0
mihaule: dc.b    ' FIN ',0

```

```

haques1: dc.b    27,'p A LITTLE DISK UTILITY (C) U. Braun 1986 '
           dc.b    27,'q',0

```

```
haques2: dc.b    27,'p    DATA BECKER & MICRO APPLICATION    '  
         dc.b    27,'q',0  
haques3: dc.b    27,'p    Sélectionner les menus avec les flèches    '  
         dc.b    27,'q',0
```

```
*****  
* Adresses du texte du menu sector (mensect) et des routines    *  
* du menu sector                                              *  
*****
```

```
seincjmp: dc.l    incdrive  
         dc.l    incside  
         dc.l    inctrack  
         dc.l    incsect  
         dc.l    readsec  
         dc.l    writsec  
         dc.l    editsec  
         dc.l    gomain
```

```
sedecjmp: dc.l    decdrive  
         dc.l    decside  
         dc.l    dectrack  
         dc.l    decsect  
         dc.l    readsec  
         dc.l    writsec  
         dc.l    editsec  
         dc.l    gomain
```

```
mensect: dc.l    misecta  
         dc.l    misectb  
         dc.l    misectc  
         dc.l    misectd  
         dc.l    misecte  
         dc.l    misectf  
         dc.l    misectg  
         dc.l    misecth
```

```

misecta: dc.b      ' drive: '
mdrive:  dc.b      '0',' ',0
misectb: dc.b      ' side: '
mside:   dc.b      '0',' ',0
misectc: dc.b      ' track: '
mtrack:  dc.b      '0','0',' ',0
misectd: dc.b      ' sector: '
msector: dc.b      '0','1',' ',0
misecte: dc.b      ' READ ',0
misectf: dc.b      ' WRITE ',0
misectg: dc.b      ' EDIT ',0
misecth: dc.b      ' BACK ',0

```

```

wrques1: dc.b      27,'p',' Ecrire ce secteur > : ',27,'q',0
wrques2: dc.b      27,'p' <yes,no> ? ',27,'q',0
wrques3: dc.b      27,'p' Annulé      <Return>      ',27,'q',0
seques1: dc.b      27,'p' MODE SECTEUR ',27,'q',0
edques1: dc.b      27,'p' MODE EDIT:  <Return> := FIN ',27,'q',0

```

```

#####
# Adresses pour menu track                                     #
#####

```

```

trincjmp: dc.l      incdrive
          dc.l      incside
          dc.l      inctrack
          dc.l      incstra
          dc.l      readitr
          dc.l      writitr
          dc.l      edittr
          dc.l      gomain

```

```

trdecjmp: dc.l      decdrive
          dc.l      decside
          dc.l      dectrack
          dc.l      decstra
          dc.l      readitr
          dc.l      writitr
          dc.l      edittr

```

```

                dc.l    gomain

mentrack: dc.l    m1secta
                dc.l    m1sectb
                dc.l    m1sectc
                dc.l    mitracal
                dc.l    mitracka
                dc.l    mitrackb
                dc.l    mitrackc
                dc.l    mitrackd

mitracal: dc.b     ' Sec/Trac: '
setrack:  dc.b     '0','9',' ',0
mitracka: dc.b     ' READ ',0
mitrackb: dc.b     ' WRITE ',0
mitrackc: dc.b     ' EDIT Tr. ',0
mitrackd: dc.b     ' BACK ',0

trques1:  dc.b     27,'p MODE TRACK ',27,'q',0
trques2:  dc.b     27,'p MODE TRACK + SYNCs ',27,'q',0
trques3:  dc.b     27,'p Sector: ',0
trques4:  dc.b     ' ',27,'q',0
trques5:  dc.b     27,'p Ecrire cette piste > ',27,'q',0
trques6:  dc.b     27,'p < yes/no > ',27,'q',0

```

```

#####
* Adresses pour le menu Track mit Syncs
#####

```

```

syincjmp: dc.l    incdrive
                dc.l    incside
                dc.l    inctrack
                dc.l    rdtracks
                dc.l    readadr
                dc.l    gomain

sydecjmp: dc.l    decdrive

```

```

dc.l    dectside
dc.l    dectrack
dc.l    rdtracks
dc.l    readadr
dc.l    gomain

```

```

mensync: dc.l    misecta
          dc.l    misectb
          dc.l    misectc
          dc.l    misynca
          dc.l    misyncb
          dc.l    mitrackd

```

```

misynca: dc.b    ' READ WITH SYNCs ',0
misyncb: dc.b    ' ADDR. FIELD ',0

```

```

#####
* Cluster
#####

```

```

clincjmp: dc.l    incdrive
           dc.l    incclust
           dc.l    rdclust
           dc.l    nextclst
           dc.l    wrclust
           dc.l    edclust
           dc.l    stclust
           dc.l    gomain

```

```

cldecjmp: dc.l    decdrive
           dc.l    decclust
           dc.l    rdclust
           dc.l    nextclst
           dc.l    wrclust
           dc.l    edclust
           dc.l    stclust
           dc.l    gomain

```

```

aenclust: dc.l  m1secta
               dc.l  m1clusa
               dc.l  m1secte
               dc.l  m1clusb
               dc.l  m1sectf
               dc.l  m1clusd
               dc.l  m1clusc
               dc.l  m1secth

```

```

m1clusa: dc.b  ' CLUST: '
m1clusa1: dc.b '0','0','0','0',' ' ',0
m1clusb: dc.b  ' NEXT ',0
m1clusc: dc.b  ' STARTofFILE ',0
m1clusd: dc.b  ' EDIT ',0

```

```

clques1: dc.b  27,'p CLUSTER MODE ',27,'q',0
clques2: dc.b  27,'p En quittant le mode cluster, le secteur'
               dc.b 'est mis à jour dans le menu ',27,'q',0
clques4: dc.b  27,'p Dernier cluster ',27,'q',0
sclques1: dc.b  27,'p Nom: Attribut: '
               dc.b ' Startcluster: Nombre de Bytes: ',27,'q',0
sclques2: dc.b  27,'p mettre Start-Cluster dans menu avec <RETURN>'
               dc.b ' lecture par <up>, <down>. ',27,'q',0
clques5: dc.b  27,'p Ecrire ce cluster > : ',27,'q',0

```

```

tretsiz: dc.b  ' Dctets/Secteur: ',0
tclsiz: dc.b  ' Secteur/Cluster: ',0
tclsizb: dc.b  ' Dctets/Cluster: ',0
trdlen: dc.b  ' Secteur/Directory: ',0
tfsiz: dc.b  ' Secteur/FAT: ',0
tfatrec: dc.b  ' 2e numéro de secteur FAT:',0
tdatrec: dc.b  ' Secteur du 1er Clusterdonnée:',0
tnumcl: dc.b  ' Nombre de clusters: ',0
tnbrface: dc.b  ' Nombre de faces: ',0
tdir1: dc.b  27,'p First Directory-sector > Face: 0 Track: 1 '
        dc.b ' Sector: 3 ',27,'q',0
tdir2: dc.b  27,'p First Directory-sector > Face: 1 Track: 0 '
        dc.b ' Sector: 3 ',27,'q',0

```



```

tfolder: dc.b      ' Sousdirectory ',0
treadwr: dc.b      ' Read/Write    ',0
treadon: dc.b      ' Read only     ',0
thidden: dc.b      ' HIDDEN File   ',0
tdelet:  dc.b      ' Effacé        ',0
tdisname: dc.b     ' Nom du disque ',0

```

```

#####
* menu Format
#####

```

```

foincjmp: dc.l      incdrive
           dc.l      incside
           dc.l      inctrack
           dc.l      incstra
           dc.l      formatl
           dc.l      xformat
           dc.l      gogaps
           dc.l      gomain

```

```

fodecjmp: dc.l      decdrive
           dc.l      decside
           dc.l      dectrack
           dc.l      decstra
           dc.l      formatl
           dc.l      xformat
           dc.l      gogaps
           dc.l      gomain

```

```

formmen:  dc.l      misecta
           dc.l      misectb
           dc.l      misectc
           dc.l      mitracal
           dc.l      miformd
           dc.l      miforme
           dc.l      miformf
           dc.l      miformg

```

```

mlformd: dc.b      ' FORMAT ',0
mlforme: dc.b      ' XFORMAT ',0
mlformf: dc.b      ' GAPS ',0
mlformg: dc.b      ' BACK ',0

```

```

foques1: dc.b      27,'p Format Track Mode ',27,'q',0
foques2: dc.b      27,'p Track:',0
foques3: dc.b      ' Formatage ? <yes/no> ',27,'q',0
foques4: dc.b      27,'p Not formatted <Taste> ',27,'q',0
foques5: dc.b      ' en Face:',0
foques6: dc.b      ' du Drive:',0
xfques1: dc.b      27,'p Formatage avec nouveaux GAP entre pis'
                dc.b      'tes et secteurs ? <yes/no> ',27,'q',0
xfques2: dc.b      27,'p Un instant puis <Return> ',27,'q',0
mlform1: dc.l      ' Format Track ',0

```

```

#####
*   Init Menu                                           *
#####

```

```

inincjmp: dc.l      incdrive
                dc.l      incmaxtr
                dc.l      incmaxse
                dc.l      dodrivin
                dc.l      showbpb
                dc.l      gomain

```

```

indecjmp: dc.l      decdrive
                dc.l      decmaxtr
                dc.l      decmaxse
                dc.l      dodrivin
                dc.l      showbpb
                dc.l      gomain

```

```

meninit:  dc.l      alsecta
                dc.l      mladrina
                dc.l      aldrinb
                dc.l      mldrinc

```

```

dc.l    mldrinc1
dc.l    mldrind

mldrina: dc.b    ' MAXTRACK: '
maxitr:  dc.b    '7','9',' ',0
mldrindb: dc.b    ' MAXSECTOR: '
maxise:  dc.b    '0','9',' ',0
mldrinc: dc.b    ' INIT DRIVE ',0
mldrinc1: dc.b    ' SHOW BPB ',0
mldrind: dc.b    ' BACK ',0

driques1: dc.b    27,'p INIT DRIVE MENUE ',27,'q',0
driques2: dc.b    27,'p Bios Parameter Block du drive actif '
          dc.b    ' < Return > ',27,'q',0
catfra1:  dc.b    27,'p Directory débute en Face: 0 Track: 1 Sector: 3 '
          dc.b    27,'q',0
catfra2:  dc.b    27,'p Directory débute en Face: 1 Track: 0 Sector: 3 '
          dc.b    27,'q',0

device:   dc.w    2
drive:    dc.w    0
side:     dc.w    0
track:    dc.w    0
sector:   dc.w    0

seek:     dc.w    3
savesr:   dc.w    0
flstatus: dc.w    0

#####
* Gap-Menu
#####

gpincjmp: dc.l    incgap1
          dc.l    incgap2
          dc.l    incgap3
          dc.l    incgap4
          dc.l    incgap5

```

	dc.l	incbyte
	dc.l	goformat
gpdecjmp:	dc.l	decgap1
	dc.l	decgap2
	dc.l	decgap3
	dc.l	decgap4
	dc.l	decgap5
	dc.l	decbyte
	dc.l	goformat
mengap:	dc.l	mlgapa
	dc.l	mlgapb
	dc.l	mlgapc
	dc.l	mlgapd
	dc.l	mlgape
	dc.l	mlgapf
	dc.l	mlgapg
mlgapa:	dc.b	' GAP1: '
mgap1:	dc.b	'60 ',0
mlgapb:	dc.b	' GAP2: '
mgap2:	dc.b	'12 ',0
mlgapc:	dc.b	' GAP3: '
mgap3:	dc.b	'22 ',0
mlgapd:	dc.b	' GAP4: '
mgap4:	dc.b	'40 ',0
mlgape:	dc.b	' GAP5: '
mgap5:	dc.b	'664 ',0
mlgapf:	dc.b	' Byte/Sec: '
mdrisekt:	dc.b	'0512 ',0
mlgapg:	dc.b	' BACK ',0
drques1:	dc.b	27,'p Drive Format Mode ',27,'q',0
gpques1:	dc.b	27,'p Change Gaps entre les secteurs ',27,'q',0
slques1:	dc.b	27,'p Un instant puis <Return> ',27,'q',0
slques3:	dc.b	27,'p SECTOR MODE ',27,'q',0

```
drbyte: dc.w    512
gap1:   dc.w    60
gap2:   dc.w    12
gap3:   dc.w    22
gap4:   dc.w    40
gap5:   dc.w    664
```

```
*****
```

```
sadques1: dc.b    27,'p Track:  Face:  Sector:  Bytes:  Checsum(hex) '
          dc.b    27,'q',0
```

```
*****
```

```
* Séquences Escape pour l'émulation de Terminal *
* comme invers on et off, positionner curseur etc. *
*****
```

```
clrest1: dc.b    27,'J',0
clrest2: dc.b    27,'K',0
revers1: dc.b    27,'p',0
revers2: dc.b    27,'q',0
loccurs1: dc.b    27,'Y',33,33,0
home1:   dc.b    27,'H',0
clear1:  dc.b    27,'E',0
curup1:  dc.b    27,'A',0
curdown1: dc.b    27,'B',0
insline1: dc.b    27,'L',0
delline1: dc.b    27,'I',0
overout1: dc.b    27,'W',0
curout1: dc.b    27,'F',0
curon1:  dc.b    27,'e',0
spaces:  dc.b    '      ',0
hilcurs: dc.b    27,'J',0
```

```
*****
```

```
* Adresses des textes d'erreurs *
*****
```

```
errtab: dc.l    error1
        dc.l    error2
        dc.l    error3
        dc.l    error4
        dc.l    error5
        dc.l    error6
        dc.l    error7
        dc.l    error8
        dc.l    error9
        dc.l    error10
        dc.l    error11
        dc.l    error12
        dc.l    error13
        dc.l    error14
        dc.l    error15
        dc.l    error16
        dc.l    error17
        dc.l    error18
        dc.l    error19
        dc.l    error20
        dc.l    error21
        dc.l    error22
        dc.l    error23
        dc.l    error24
        dc.l    error25
        dc.l    error26
        dc.l    error27
        dc.l    error28
        dc.l    error29
```

```
#####
*  textes d'erreur                                     *
#####
```

```
error1: dc.b    27,'p',' PAS de BOOTSECTOR ',27,'q',0
error2: dc.b    27,'p Défaut dans le secteur de DIR <Return> ',27,'q',0
error3: dc.b    ' erreur3',0
error4: dc.b    ' erreur4',0
error5: dc.b    ' erreur5 ',0
error6: dc.b    ' erreur6 ',0
```

```

error7:  dc.b      27,'p',' Disque absent/pas de piste ',27,'q',0
error8:  dc.b      ' erreur8 ',0
error9:  dc.b      27,'p',' Pas de secteur !',27,'q',0
error10: dc.b      ' erreur10',0
error11: dc.b      ' erreur11',0
error12: dc.b      ' erreur12',0
error13: dc.b      ' erreur13 ',0
error14: dc.b      27,'p Le disque est protégé en écriture ',27,'q',0
error15: dc.b      ' erreur15 ',0
error16: dc.b      ' erreur16 ',0
error17: dc.b      ' erreur17',0
error18: dc.b      ' erreur18 ',0
error19: dc.b      ' erreur19 ',0
error20: dc.b      27,'p Plus de Cluster ',27,'q',0
error21: dc.b      ' erreur21 ',0
error22: dc.b      ' erreur22 ',0
error23: dc.b      ' erreur23 ',0
error24: dc.b      ' erreur24 ',0
error25: dc.b      ' erreur25 ',0
error26: dc.b      ' erreur26 ',0
error27: dc.b      ' erreur27 ',0
error28: dc.b      ' erreur28 ',0
error29: dc.b      ' erreur29 ',0

```

```

#####
pattern: dc.w      $ffff

```

```

      bss

```

```

menuadr: ds.l      1
ganz:    ds.l      1
revnum:  ds.l      1
jumptable: ds.l    1

```

```

wtrack:  ds.w      1
wsector: ds.w      1
wside:   ds.w      1
wdrive:  ds.w      1
wclust:  ds.w      1

```

maxtrack: ds.w 1
maxsect: ds.w 1
maxdriv: ds.w 1
maxside: ds.w 1
maxclust: ds.w 1

topptr: ds.l 1
oldtop: ds.l 1
botptr: ds.l 1

column: ds.w 1
ligne: ds.w 1
head1: ds.w 1
head2: ds.w 1

curlini: ds.w 1
curspal: ds.w 1
oldlini: ds.w 1
oldspal: ds.w 1

ligcount: ds.w 1
prcount: ds.w 1

retwl: ds.w 1
incvar: ds.l 1
decvar: ds.l 1
usstack: ds.l 1
sustack: ds.l 1

dmastat: ds.w 1

currdma: ds.b 1
highdma: ds.b 1
middma: ds.b 1
lowdma: ds.b 1

maxhead: ds.w 1

savebpb: ds.l 1
recsiz: ds.w 1

clsiz:	ds.w	1
clsizb:	ds.w	1
rdlen:	ds.w	1
fsiz:	ds.w	1
fatrec:	ds.w	1
datrec:	ds.w	1
numcl:	ds.w	1
bflags:	ds.w	1
oldsec:	ds.w	1
dflag:	ds.w	1
eflag:	ds.w	1
edflag:	ds.w	1
nbrface:	ds.w	1
tabl:	ds.w	1
oldclst:	ds.w	1
newclst:	ds.w	1
clstnum:	ds.w	1
logsect:	ds.w	1
asector:	ds.w	1
topdma:	ds.l	1
editptr:	ds.l	1
savereg:	ds.l	16
maxdown:	ds.w	1
maxup:	ds.w	1
lineavar:	ds.l	1
varl1:	ds.l	1
varw1:	ds.w	1
varw2:	ds.w	1
varw3:	ds.w	1

dirptr: ds.l 1

dirbuf: ds.w 4000

fatbuf: ds.w 4000

formbuf: ds.w 6000

platztr: ds.w 6000

end

Les sous-programmes accessoires sont inclus dans la partie "options" et doivent être insérés dans le programme "édits" où toutes les routines sont remplacées par des RTS. Au mieux, gardez le même ordre que moi car de nombreuses options font accès à d'autres options. Si vous saisissez le programme tel qu'il est, vous n'aurez aucun problème.

Nous commençons par les menus d'options afin que vous puissiez analyser les secteurs et les pistes en modifiant leur nombre et la BPB.

```

#####
#####
* Subroutines de l'option INIT, doivent d'abord être implémentées *
* car elles permettent de lire le 10ième Secteur sur la piste *
* B2 Track etc. De plus, certaines routines sont appelés par *
* d'autres parties de programme. *
#####
#####

```

```

#####
* Initialiser Drive courant (à partir du menu) et mémoriser les *
* variables du Biosparameterbloc *
#####

```

```

initdrv: move.w wdrive,d0      * drive courant
        move.w wdrive,-(a7)    * sur la pile
        move.w #7,-(a7)        * Getbpb Fonction
        trap    #13            * BIOS-Trap
        addq.l  #4,a7          * Restaurer la pile
        tst.l   d0             * Erreur ?
        bne     doinit1
        move.w  d0,-(a7)        * Si oui, l'envoyer
        jsr     errhand         * et retour
        bra     doiniten
doinit1: move.l  d0,a0          * sinon d0 = Basisadress du BPB

```

```

    move.w    (a0)+,recsiz    * Bytes pro Sector
    move.w    (a0)+,clsiz    * Sectors/ Cluster
    move.w    (a0)+,clsizb   * Bytes/Cluster
    move.w    (a0)+,rdlen    * Sectors/Directory
    move.w    (a0)+,fsiz     * Sectors/Fat
    move.w    (a0)+,fatrec    * Sec. Num. de la deuxième FAT
    move.w    (a0)+,datrec    * Sec.Num. du premier Dat.Clust.
    move.w    (a0)+,numcl    * Nombre de clusters de données
    move.w    (a0)+,bflags    * Flags
    move.w    (a0)+,nbrface   * encore dummy
    move.w    (a0)+,nbrface   * Nombre de faces
doiniten: rts                * Et retour

```

```

#####
* Lit les secteurs de FAT de la disquette dans le Fatbuffer
#####

```

```

rdfat:  move.w    wdrive,-(a7)    * drive courant
        move.w    fatrec,-(a7)    * Sectnumber de la deuxième Fat
        move.w    fsiz,-(a7)      * Nombre de secteurs par FAT
        move.l    #fatbuf,-(a7)   * bufferadress on Stack
        move.w    #2,-(a7)        * Lire sans condition
        move.w    #4,-(a7)        * Rwabs Fonction
        trap      #13             * BIOS-Trap
        add.l     #14,a7          * restaurer pile
        tst.w     d0              * Erreur ?
        bmi       rdfater         * Si oui traiter
rdfatend: rts                    * sinon retour

```

```

rdfater: move.w    d0,-(a7)        * Fehlernumber sur Stack
        jsr       errhand         * traiter
        bra       rdfatend        * et retour

```

```

#####
* Lit les secteurs de catalogue de la disquette dans un tampon
#####

```

```

rddir:  move.w    wdrive,-(a7)    * Drive courant
        move.w    fsiz,d0        * Nombre de secteurs de FAT
        lsl.w     #1,d0          * fois deux (FAT's) plus un
        addq.w    #1,d0          * égale numéro de secteur logique du
        move.w    d0,-(a7)       * premier secteur de catalogue
        move.w    rdlen,-(a7)    * Nombre de secteurs Directory
        move.l    #dirbuf,-(a7)  * Adresse du tampon
        move.w    #2,-(a7)       * Lire sans condition
        move.w    #4,-(a7)       * Rwabs Fonction
        trap      #13            * BIOS
        add.l     #14,a7
        tst.w     d0             * Erreur ?
        bni       rddirer        * oui
rddiread: rts                    * Sinon, retour

rddirer: move.w    d0,-(a7)       * Numéro d'erreur
        jsr       errhand
        bra       rddirend

```

```

#####
* Incrémente le nombre max. de piste dans le menu Init-Drive-      *
* on peut incrémenter le n° de piste courant dans tous les autres  *
* menus jusqu'à cette valeur                                         *
#####

```

```

incmaxtr: move.w    maxtrack,d0
        cmp.w     #99,d0        * 99 est le Maximum
        blt       incma1        * Sinon même processus que
        move.w    #0,d0         * avec ancien changement de menu
        bra       incma2

incma1:  addq.w     #1,d0
incma2:  move.w     d0,maxtrack
        ext.l     d0
        divu      #10,d0
        add.b     #'0',d0
        move.b     d0,maxltr     * Modifier également texte du menu

```

```

swap      d0
add.b     #'0',d0
move.b    d0,max1tr+1
jsr       dispmen      * Afficher menu
rts       * Et retour

decmaxtr: move.w    maxtrack,d0      * Décrémente nombre max de
        cmp.w      #0,d0            * Tracknumber
        ble        decma1
        subq.w     #1,d0
        bra        decma2
decma1:   move.w    #99,d0
decma2:   move.w    d0,maxtrack
        ext.l      d0
        divu       #10,d0
        add.b      #'0',d0
        move.b     d0,max1tr        * Modifier dans texte du menu
        swap       d0
        add.b      #'0',d0
        move.b     d0,max1tr+1
        jsr        dispmen          * Afficher menu
        rts         * et retour

incmaxse: move.w    maxsect,d0      * idem avec nombre max de secteurs
        cmp.w      #99,d0          *
        blt        incmas1
        move.w     #0,d0
        bra        incmas2

incmas1:  addq.w    #1,d0
incmas2:  move.w    d0,maxsect
        ext.l      d0
        divu       #10,d0
        add.b      #'0',d0
        move.b     d0,max1se        * Mettre dans texte de menu
        swap       d0
        add.b      #'0',d0
        move.b     d0,max1se+1

```

```

    jsr    dispmen      * afficher menu
    rts                * und zurück

decmaxse: move.w    maxsect,d0      * erniedrig den maximal einstell-
    cmp.w    #0,d0      * baren Sektor
    ble      decmas1
    subq.w    #1,d0
    bra      decmas2
decmas1: move.w    #99,d0
decmas2: move.w    d0,maxsect
    ext.l    d0
    divu     #10,d0
    add.b     #'0',d0
    move.b    d0,maxlse      * im Menutext ändern
    swap     d0
    add.b     #'0',d0
    move.b    d0,maxlse+1
    jsr      dispmen      * afficher menu
    rts                * et retour

```

```

#####
* Voici la routine Drive-Init proprement dite qui initialise le      *
* Drive courant (dans wdrive) et qui lit le Bios-Parameter-        *
* Bloc ainsi que les secteurs de FAT- et Directory dans le tampon   *
* désiré                                                              *
#####

```

```

dodrvin: jsr    initdriv      * Initialiser Drive
    jsr      rdfat          * Lire secteurs FAT
    jsr      rddir         * Lire secteurs Directory
    jsr      showbpb       * Afficher BIOS-Parameter-Bloc
    jsr      curleft
dodriven: rts                * et retour

```

```

#####

```

* Afficher BIOS-Parameter-Bloc

```

showbpb: move.w    #4,ligne      * Curseur en ligne 4, colonne 10
         move.w    #10,column
         jsr       loccurs      * positionner
         move.l    #driques2,a0 * Afficher Message
         jsr       printf
         move.w    #42,tab1     * Tabpoint Sur l'écran pour
         move.w    #6,ligne     * Impression des chiffres
         move.w    #12,column
         jsr       loccurs
         move.l    #trecsiz,a0  * Bytes pro Cluster
         jsr       printf
         jsr       curstab      * Ecrire Texte
         move.w    recsiz,-(a7) * ecrire Bytes pro Cluster en chiffres
         jsr       dezpr        * décimaux
         addq.w    #1,ligne     * Ligne suivante
         move.w    #12,column
         jsr       loccurs
         move.l    #tclsiz,a0   * Sectors pro Cluster
         jsr       printf
         jsr       curstab
         move.w    clsiz,-(a7)
         jsr       dezpr
         addq.w    #1,ligne
         move.w    #12,column
         jsr       loccurs
         move.l    #tclsizb,a0  * Bytes pro Cluster
         jsr       printf
         jsr       curstab
         move.w    clsizb,-(a7)
         jsr       dezpr
         addq.w    #1,ligne
         move.w    #12,column
         jsr       loccurs
         move.l    #trdien,a0   * Sectors pro Directory
         jsr       printf
         jsr       curstab
         move.w    rdlen,-(a7)

```



```

jsr      dezpr
addq.w   #1,ligne
move.w   #12,column
jsr      loccurs
move.l   #tfsiz,a0      * Sectors pro FAT
jsr      printf
jsr      curstab
move.w   fsiz,-(a7)
jsr      dezpr
addq.w   #1,ligne
move.w   #12,column
jsr      loccurs
move.l   #tfatrec,a0    * Sectorsnumber de la deuxième FAT
jsr      printf
jsr      curstab
move.w   fatrec,-(a7)   *
jsr      dezpr
addq.w   #1,ligne
move.w   #12,column
jsr      loccurs
move.l   #tdatrec,a0    * Sectorsnumber du premier cluster de
jsr      printf        * données
jsr      curstab
move.w   datrec,-(a7)
jsr      dezpr
addq.w   #1,ligne
move.w   #12,column
jsr      loccurs
move.l   #tnumcl,a0     * Nombre de cluster de données
jsr      printf
jsr      curstab
move.w   numcl,-(a7)
jsr      dezpr
addq.w   #1,ligne
move.w   #12,column
jsr      loccurs
move.l   #tnbrface,a0   * Nombre de faces de la disquette
jsr      printf
jsr      curstab
move.w   nbrface,-(a7)

```

```

        jsr      dezpr
        addq.w   #2,ligne
        move.w   #10,column
        jsr      loccurs
        move.l   #tdir1,a0      * Indique où se trouve le premier secteur
        move.w   nbrface,d0     * de directory, diffère entre
        cmp.w    #2,d0          * les disquettes simple et double face
        bne      showbpb1
        move.l   #tdir2,a0
showbpb1: jsr      printf
        jsr      videbuff      * vider tampon clavier
        jsr      wtast         * attendre touche
        jsr      cursmess
        jsr      delline
        jsr      cursmess
        move.l   #driques1,a0   * afficher message
        jsr      printf
        rts                  * et retour

```

```

#####
#####
* Subroutines de l'option TRACK du menu principal plus *
* Une routine d'écriture de secteurs personnelle *
#####
#####

```

```

#####
* Routine d'écriture de secteurs personnelle, accède directement au *
* contrôleur et au DMA. La routine XBIOS d'écriture de secteur *
* ne permet pas contrairement à la routine de lecture de secteurs *
* d'écrire des secteurs de 1024 octets. *
* Afin de pouvoir insérer cette routine, il faut auparavant *
* implémenter le menu rdstrack car certaines routines de ce menu *
* sont appelées (super, seldrive, ect.). Dans la configuration *
* de base du programme (Seulement avec menu sector), il n'est *
* malheureusement pas possible d'écrire 1024 octets par secteur *
* *
#####

```

```

selfsect: jsr    super      * Supervisor-Mode
          st      flock     * Désactiver Floppy-Interrupt
          jsr     seldrive   * Sélectionner drive et face
          jsr     flreset    * reset du Controller
          jsr     searcht    * rechercher Track dans wtrack
          jsr     selwrite   * Ecrire secteur
          sf      flock     * Autoriser Floppy-Interrupt
          jsr     videbuff   * vider tampon clavier
          jsr     cursmess   * positionner curseur
          jsr     delline    * Effacer ligne
          jsr     flreset    * Controller reset
          jsr     user       * Mettre en User-Mode
          move.l  #slques1,a0 * Afficher message
          jsr     printf
          jsr     wtast      * Attendre touche
          jsr     super      * Super-Visor-Mode
          jsr     deselect   * Deselectionner Floppy
          jsr     user       * Activer User-Mode
          jsr     cursmess   * positionner curseur
          jsr     delline    * Effacer ligne
          jsr     cursmess   * repositionner
          move.l  #slques3,a0 * Afficher message
          jsr     printf
          move.l  (a7)+,a3-a6/d3-d7 * Récupérer registre
          rts              * et retour

```

```

#####
*
* Le secteur est inscrit sur la disquette
*
#####

```

```

selwrite: jsr    setplat2   * Mettre adresse du tampon
          move.w  #$190,dma0 * Mettre en écriture
          move.w  #$90,dma0  * en "Toggle"isant La broche de
          move.w  #$190,dma0 * Lecture/écriture
          move.w  #4,d6      * Mettre 4 dans Sectorcount-Register

```

dma000 : \$FF8606

```

        jsr      wrcontr      *
        move.w   #$184,dmamode * Sélectionner Sectorregister du FDC
        move.w   wsector,d6   * Envoyer secteur courant au FDC
        jsr      wrcontr
        move.w   #$190,dmamode * Sélectionner Controller
        move.w   #$a0,d6      * envoyer instruction Sector-Write
        jsr      wrcontr      * au contrôleur
        move.l   #$50000,d7    * Compteur Time-out
selwrit!: bts    #5,mfp        * Entrée Interrupt du FDC au MFP
        beq      selwrend     * si 1 alors fini
        subq.l   #1,d7        * Diminuer Timeout
        bne      selwrit!     * Si pas terminé continuer attente
        move.w   #-9,-(a7)     * Sinon erreur 9 sur Stack
        jsr      errhand      * Donner au "traiteur" d'erreur
        jsr      cursmess     * Enfin effacer ligne d'impression
        jsr      delline      *
        rts

selwrend: jsr      rdstatus     * Teste si fonctionnement sans erreur
        move.w   f1status,d0   *
        bts      #6,d0         * writeprotect
        bne      selwerri     * oui
        rts                  * sinon retour

selwerri: move.w   #-8,-(a7)    * Message d'erreur Nr. 8(writeprotect)
        jsr      errhand      * Afficher en effacer ligne
        jsr      cursmess     * D'affichage
        jsr      delline
        rts

```

```

#####
#####
* Subroutine de l'option TRACK du menu principal *
#####
#####

#####

```

```

*
* Cette ligne lit une piste entière ou le nombre de secteurs de la
* variable asector. Il s'agit de secteurs standards de
* 512 octets. Des différences éventuelles doivent être assurées
* Par la modification de la variable asector
*
*****

readtr: move.w    #512,d0          % Taille de secteur standard
        mulu      asector,d0      % Nombre de secteurs par piste
        move.w    d0,maxhead      % Nombre max d'octets comme compteur
        move.w    asector,-(a7)    % Nombre de secteurs/piste dans menu
        move.w    wside,-(a7)      % face courante
        move.w    wtrack,-(a7)     % Piste courante
        move.w    #1,-(a7)         % A partir du secteur 1
        move.w    wdrive,-(a7)     % Drive courant
        clr.l      -(a7)           % mot long dummy
        move.l     #platr,-(a7)     % bufferaddress
        move.w     #8,-(a7)        % X-tended Bios Fonction 8
        trap       #14             % Appeler et éclaircir la piste
        add.l      #20,a7          % Et tenter une sortie sans
        tst.w      d0              % erreur
        bmi        readtr          % Si erreur alors afficher
        jsr        showtr          % Sinon afficher piste et
readti2: rts                      % retour

readtr: move.w     d0,-(a7)         % Envoyer numéro d'erreur sur pile
        jsr        crchand         % au "traiteur" d'erreur
        jsr        videbuff        % et vider tampon clavier
        jsr        wlast           % et Attendre touche
        jsr        cursmes         % Afficher message
        move.l     #trquesl,a0
        jsr        printf
        jsr        delrest
        bra        readti2         % et retour

*****

```

```

* Incrémente le nombre de secteurs par piste lors de la pression de *
* Curseur-haut *
#####

```

```

incstra: move.w    asector,d0      * comparer Nombre de secteurs par piste
        cmp.w      maxsect,d0     * avec nombre max de secteurs
        blt        incst1         * Si supérieur ou égal
        move.w     #0,d0          * alors mettre à zéro Sectors/Track
        bra        incst2         * et retour
incst1:  addq.w     #1,d0          * sinon additionner 1 au nombre Sect/pis
incst2:  move.w     d0,asector     *
        ext.l      d0             * Mettre la modification dans le
        divu       #10,d0         * texte de menu. Mettre en octets
        add.b      #'0',d0        * ASCII en divisant par 10
        move.b     d0,setrack     * et mettre dans menu
        swap       d0            * et transformer Low-Byte
        add.b      #'0',d0        * dans format ASCII
        move.b     d0,setrack+1   * Mettre dans menu
        jsr        dispmen        * enfin afficher menu
        rts                * et retour

```

```

#####
* Décrémnte secteurs/piste dans menu *
#####

```

```

decstra: move.w    asector,d0      * sectors/track
        cmp.w      #0,d0
        ble        decst1         * Si supérieur à 0, alors
        subq.w     #1,d0          * soustraire 1
        bra        decst2
decst1:  move.w     maxsect,d0     * Mettre valeur max
decst2:  move.w     d0,asector
        ext.l      d0
        divu       #10,d0
        add.b      #'0',d0
        move.b     d0,setrack     * Dans données du menu
        swap       d0

```

```

add.b    #0',d0
move.b   d0,setrack+1
jsr      dispmen      * afficher menu et retour
rts

```

```

*****
* Met des valeurs dans les variables de la routine d'edition      *
* (naxdown, maxup ect.) et appelle la routine EDIT                *
*****

```

```

editlr:  move.w    #0,naxdown      * Il faut éditer seulement 512 Byte
         move.w    #208,maxup      *
         move.w    #16,ligcount    * Et afficher 19 lignes
         move.l    topptr,d0       * Pointeur dans tampon de piste
         sub.l     #platztr,d0     * moins adresse de début du tampon
         divu      #512,d0         * Diviser par le nombre d'octets par
         swap      d0              * secteur. S'il y a un reste, ce
         tst.w     d0              * n'était pas le début du tampon.
         beq       edittr:        *
         sub.l     #256,topptr     * Il faut donc soustraire 256
edittr:  move.l    topptr,editptr  * Envoyer ce pointeur sur début de
         move.w    #0,head2        * secteur dans le tampon à edittr
         move.w    #20,column      * Afficher Message en colonne 20
         move.w    #2,ligne        * de la ligne 2
         jsr       loccurs
         move.l    #edques1,a0
         jsr       printf
         jsr       edittr         * Et appeler EDIT
         jsr       cursmess       * Enfin effacer ligne de message
         jsr       delline        * et
         jsr       cursmess
         move.l    #trques1,a0    *
         jsr       printf
         jsr       curleft        * mettre le menu sur Read
         jsr       curleft
         jsr       cursbuf        *
         jsr       clrest         * Effacer l'écran restant
         rts                    * et retour

```

```

#####
# Autorise l'affichage de toute la piste lue dans le tampon #
#
#####

```

```

showtr: move.w    #0,head2          # Compteur d'octets
        move.l    #piatr,lopper    # Début du tampon.
        move.w    #0,edflag        # Flag
        move.w    #15,ligcount     # Il faut afficher 16 lignes
        move.w    #2,ligne         # En colonne 39 de la 2 ième
        move.w    #59,column       # ligne du secteur courant
        jsr       loccurs
        move.l    #trques3,a0      # afficher
        jsr       printf
        clr.w     d0
        move.w    #1,d0
        move.w    d0,-(a7)
        jsr       dezpr            # Imprimer secteur
        move.l    #trques4,a0
        jsr       printf

showt1: move.w    #4,ligne          # positionner curseur
        move.w    #0,column
        jsr       loccurs
        jsr       clrest           # Effacer reste de l'écran
        jsr       videbuff

showt2: jsr       dispbuf           # Afficher la première page et
showt3: jsr       touch            # Tester clavier
        swap      d0
        cmp.b     #$48,d0          # Cursor up ?
        beq       showtup
        cmp.b     #$50,d0          # Cursor down ?
        beq       showtdo
        cmp.b     #$1c,d0          # Return ?
        beq       showten1
        cmp.b     #$4b,d0          # Cursor left ?
        beq       showtli

```



```

        cmp.b    #$40,d0          ‡ Cursor right ?
        beq      showtre
        bra      showt3          ‡ Aucun, continuer le test
showtre: jsr     curright         ‡ curseur sur option de droite
        bra      showteni        ‡ et retour

showt11: jsr     curleft         ‡ inverser option gauche
        bra      showteni        ‡ et retour

showtup: move.w  head2,d0         ‡ Comparer compteur d'octets
        cmp.w    #0,d0           ‡ à 0
        beq      showtuen        ‡ si différent alors
        sub.w    #256,head2      ‡ soustraire 256, ce qui correspond à
        sub.l    #256,tpptr      ‡ un demi secteur
showtuen: move.w  head2,d0        ‡ Compteur d'octets
        lsr.w    #8,d0           ‡ Divisé par 512
        lsr.w    #1,d0
        add.w    #1,d0           ‡ plus un égale numéro de secteur
        move.w    d0,varw3
        move.w    #59,column
        move.w    #2,ligne
        jsr      ioccur
        move.l    #trques3,a0     ‡ Afficher numéro de secteur courant
        jsr      printf          ‡ en ligne 2
        move.w    varw3,-(a7)
        jsr      dezpr
        move.l    #trques4,a0
        jsr      printf
        jsr      delrest         ‡ Effacer reste de la ligne

showtuel: bra     showt2         ‡ Et retour à la boucle

```

#####

```

showtda: move.w  head2,d0         ‡ Cursordown-Handling
        move.w    maxhead,d1
        sub.w     #256,d1
        cmp.w     d1,d0
        beq      shwtrden
        add.w     #256,head2      ‡ Additionner 256 au pointeur tampon

```

```

        add.l    #256,topptr    * et au compteur d'octet
shwtrden: move.w  head2,d0
        lsr.w    #8,d0
        lsr.w    #1,d0        * Diviser la compteur d'octet par 512
        add.w    #1,d0        * et additionner 1
        move.w   d0,varw3      * donne numéro de secteur courant
        move.w   #59,column
        move.w   #2,ligne
        jsr      loccurs
        move.l   #trques3,a0
        jsr      printf
        move.w   varw3,-(a7)    * Afficher numéro de secteur
        jsr      dezpr
        move.l   #trques4,a0
        jsr      printf
        jsr      delrest      * Effacer reste de la ligne
shwtrdl: bra     showt2
showten1: jsr    videbuff      * vider tampon clavier
        rts                  * et retour

```

```

writltr: move.l  a4,-(a7)      * Récrit la piste lue sur
        move.w   #2,ligne      * la disquette
        jsr      loccurs
        jsr      delline
        move.l   #trques5,a0
        jsr      printf
        move.w   #33,d2        * Afficher 34octets à partir de mlsecta

        move.l   #mlsecta,a4    * à l'écran
writl1: move.b   (a4)+,d0
        move.w   d0,-(a7)
        jsr      conout
        dbra     d2,writl1
        move.l   #trques6,a0   * Tester si vraiment écrire
        jsr      printf        * sur disquette
        jsr      videbuff      * vider tampon clavier

```

```

        jsr      wlast          ‡ tester clavier
        cmp.b    #'Y',d0        ‡ tester Y supérieur ou inférieur
        beq      writit2        ‡
        cmp.b    #'y',d0
        bne      writlten       ‡ Sauf si autre touche
writit2: move.w    asector,-(a7) ‡ Nombre de secteurs sur pile
        move.w    wside,-(a7)   ‡ face courante
        move.w    wtrack,-(a7)  ‡ piste courante
        move.w    #1,-(a7)      ‡ Startsector égale un
        move.w    wdrive,-(a7)  ‡ Drive courant
        clr.l     -(a7)         ‡ mot long dummy
        move.l    #platztr,-(a7) ‡ bufferadress
        move.w    #9,-(a7)      ‡ Instruction Flopwr sur pile
        trap      #14           ‡ XBIOS-Trap
        add.l     #20,a7        ‡ Restaurer pile
        tst.w     d0            ‡ Erreur ?
        bmi      writler1       ‡ oui

writlten: jsr      cursmess      ‡ pas d'erreur alors effacer
        jsr      delline        ‡ ligne d'état et Afficher message
writitel: jsr      cursmess
        move.l    #trques1,a0
        jsr      printf
        move.l    (a7)+,a4       ‡ récupère a4
        rts

writler1: move.w    d0,-(a7)     ‡ Numéro d'erreur sur la pile
        jsr      errhand        ‡ traiter erreur
        bra      writitel       ‡ et Fin

```

```

#####
#####
‡ Subroutine de l'option TRACK with SYNCs, les routines ‡
‡ n'accèdent à aucune autre routine. Peut donc être implémenté ‡
‡ à volonté ‡
#####
#####

```



```

#####
* Remet en User-Mode
#####

```

```

user:   move.l   #1,-(a7)
        move.w   #$20,-(a7)      * GEMDOS-Fonction Super
        trap     #1
        add.l    #6,a7
        tst.w    d0
        beq      user1          * Déjà en User-Mode
        move.l   usstack,-(a7)
        move.w   #$20,-(a7)
        trap     #1
        add.l    #6,a7
user1:   rts

```

```

#####

```

```

fattend: dbra    d7,fattend
        rts

```

```

#####
* Reset du Floppydiskcontroller (FDC)
#####

```

```

flreset: jsr      super          * Mettre en Supervisor-Mode
        move.w    #$80,dnamode   * Accès au registre FDC
        move.w    #$d0,d6       * Reset par instruction Interrupt
        jsr       wrcontr        * Instruction vers Controller
        move.w    #40,d7        * Attendre un peu
        jsr       fattend
        rts                    * et retour

```

```

#####

```

```
* Lit le registre d'état du contrôleur et le mémorise      *
*****
```

```
rdcontr: jsr      super      * Mettre en Supervisor-Mode
         move.w   dmadat,d3   * Registre d'état d'après D3
         jsr      readcol    * Attendre un peu
readcol: move.w   sr,-(a7)
         move.w   d7,-(a7)   * Mémoriser compteur Timeout
         move.w   #40,d7
readco2: dbra     d7,readco2
         move.w   (a7)+,d7    * enfin retour
         move.w   (a7)+,sr
         rts
```

```
*****
* envoie le nombre se trouvant dans D6 Floppy-Disk-Controller *
*****
```

```
wrcontr: jsr      super      * Supervisor on
         jsr      readcol
         move.w   d6,dmadat
         jsr      readcol    * Attendre un peu
         rts
```

```
*****
* lit le Statusregister du FDC et le mémorise dans flstatus *
*****
```

```
rdstatus: jsr      super      * Supervisor on
         jsr      readcol
         move.w   dmadat,flstatus * Etat d'après flstatus
         jsr      readcol    * Attendre un peu
         rts                * et retour
```

```
*****
* selectionne le Drive courant (lampe rouge allumée)      *
*****
```

```

selddrive: jsr      super          ‡ Supervisor on
            move.w   wdrrive,d0     ‡ Drive courant
            cmp.w    #1,d0          ‡ supérieur à 1
            bgt      seiddrend      ‡ Si oui, alors retour
            addq.b   #1,d0          ‡ Sinon changer avec face courante
            lsl.b    #1,d0
            or.w     wside,d0       ‡
            eor.b    #7,d0
            and.b    #7,d0
select:     move.w   sr,-(a7)
            or.w     #$700,sr       ‡ Désactiver Interrupt car la Floppy
            move.b   #e,flselec     ‡ Interrupt désélectionne les Drives
            move.b   flselec,d1     ‡ de nouveau
            and.b    #$f8,d1
            or.b     d0,d1
            move.b   d1,flwrite     ‡ envoyer à l'ACIA
            move.w   (a7)+,sr       ‡ Récupérer registre d'état
seiddrend: rts                    ‡ et retour

```

```

*****
‡ Désélectionner Drive courant (lampe rouge allumée) ‡
‡ le Timing entre le Floppy-Reset et la désélection doit ‡
‡ correspondre sinon le moteur du lecteur continue à tourner ‡
*****

```

```

deselect:  jsr      super          ‡ Supervisor on
            move.w   #$80,dramode   ‡ Sélectionner registre FDC
            move.b   #7,d0
            jsr      select         ‡ désélectionner
            rts                    ‡ et retour

```

```

*****
‡ Lit une piste complète avec tous les Syncs dans le tampon qui ‡
‡ commence à l'adresse platztr ‡
*****

```

```

rdstrack: jsr      super          * Supervisor on
          clr.l    currdma
          move.w   sr, varw5      * Sauvegarder ancien registre d'état
          move.w   #$2700, sr     * Il n'est pas utile de mémoriser
          move.w   #$90, dmanode  * les interrupts. activer Registre
          move.w   #$190, dmanode * Sector-count Toggle DMAMODE pour lecture
          move.w   #$90, dmanode  * et effacer registre DMA
          move.w   #$16, d5       * Il faut lire 22*512 octets
          move.w   $512, d2       * (Il n'y en a pas autant
          mulu     d5, d2         * sur la disquette)
          move.w   d2, maxhead    * mais
          add.l    #platztr, d2   * calcul de la DMA-Endaddress
          move.l    d2, topdma    * la mémoriser
          jsr      wrcontr       * envoyer ds (nombre de secteurs) au FDC
          move.l    #platztr, d0  * Envoyer Adresse du tampon DMA au
          move.b    d0, dmalow    * DMA-Chip
          lsr.l    #8, d0
          move.b    d0, dnamid
          lsr.l    #8, d0
          move.b    d0, dmahigh
          move.w   #$90, dmanode  * Sélectionner FDC-Register
          move.w   #$eB, d6       * envoyer Readtrack-Command au FDC
          jsr      wrcontr       *
          move.l    #$50000, d7   * Compteur Timeout
          move.l    topdma, a5    * DMA-Endaddress
          move.w   #$200, d0      * Attendre un peu

rdi:      dbra     d0, rdi

rdstrll:  btst     #5, mfp        * Instruction déjà traitée ?
          beq      rdtrendl      * Si oui alors fin
          subq.l   #1, d7        * sinon décrémenter compteur Time-out
          beq      rdirerr1      * Si compteur terminé alors erreur
          move.b   dmahigh, highdma * Tester si DMA-Adresse déjà
          move.b   dnamid, middma * atteinte est inutile car
          move.b   dmalow, lowdma  * le Controller s'est déjà arrêté (moins
          cmp.l    currdma, a5    * d'octets sur disquette)
          bgt      rdstrll

```



```

rdtrend1: move.w    #$90,dmamode    ‡ Mettre sur Sectorcountregister
         move.w    dmamode,d5      ‡ Lire état du DMA-Chips
         move.w    d5,daastat      ‡ et mémoriser
         btst      #0,d5
         beq       rdtrerr2
         move.w    #$80,dmamode    ‡ Mettre sur FDC-Register
         jsr       rdstatus        ‡ Lire état FDC
rdtend:  move.w    varn3,sr        ‡ Récupérer registre d'état
         rts                ‡ et retour

rdtrerr2: bra      rdtend

rdtrerr1: bra      rdtend

#####
‡      met la tête de lecture sur la piste indiquée dans wtrack      ‡
#####

searcht: jsr       super          ‡ Supervisor on
         jsr       track0         ‡ Rechercher piste 0
         move.w    #$86,dmamode    ‡ Sélectionner Track Register
         move.w    wtrack,d6      ‡ envoyer piste courante au Trackregister
         jsr       wrcontr        ‡
         move.w    #$B0,dmamode    ‡ Sélectionner FDC-Register
         move.w    #$1b,d6        ‡ Envoyer Instruction Search-Track
         jsr       wrcontr        ‡ auu registr de piste
         move.l    #$60000,d7      ‡ Compteur Timeout
searchi: subq.l    #1,d7
         beq       searend1
         btst      #5,mfp         ‡ Instruction déjà traitée ?
         bne       searchi        ‡ Non, alors continuer à attendre
         rts

searend1: move.w    #-7,-(a7)      ‡ Erreur = pas de disquette
         jsr       errhand
         rts

#####
‡ Recherche de la piste 0 ‡

```

```
#####
```

```
track0:  move.w    seek,d6      * Seek-Rate
         and.w     #3,d6       * faire ET avec instruction piste 0
         move.l    #$50000,d7  * Compteur Time-out
         move.w    #$80,dmanode * Accès au registre FDC
         jsr       wrcontr     * Envoyer instruction
```

```
track0l1: subq.l   #1,d7       * Décrémenter compteur
         beq       track0er    * Timeout
         btst      #5,nfp      * FDC fini ?
         bne       track0l1    * non alors continuer attente
         rts         * et retour
```

```
track0er: move.w   #-7,-(a7)    * envoyer numéro d'erreur
         jsr       errhand     * au "traiteur" d'erreur
         rts         * et retour
```

```
#####
```

```
* donne l'Adresse du tampon platztr aun DMA-Controller *
```

```
#####
```

```
setplatz: move.l   #platztr,d0
         move.b    d0,dmalow
         lsr.l     #8,d0
         move.b    d0,damid
         lsr.l     #8,d0
         move.b    d0,dmahigh
         rts
```

```
#####
```

```
* Routine de contrôle Readtrack avec tous les Syncs. Elle appelle *
```

```
* tous les sous-programmes possibles *
```

```
#####
```

```

rdtracks: movem.l a3-a6/d3-d7,-(a7)    ‡ Sauvegarder Registre
        jsr      cursmess
        jsr      delline
        jsr      cursmess
        move.l   #trques2,a0          ‡ Afficher message
        jsr      printf
        move.w   #18,ligcount         ‡ pour Subroutine Disbuf = 19 lignes
        jsr      super                ‡ Supervisor on
        st       flock                ‡ Floppy-interrupt off
        jsr      seldrive             ‡ Sélectionner Drive
        jsr      flreset              ‡ Reset du Controller
        jsr      searcht              ‡ Rechercher piste courante
        jsr      rdstrack             ‡ lire deux fois cette piste car
        jsr      rdstrack             ‡ la disquette ne serait pas en route
        jsr      flreset              ‡ reset du FDC
        jsr      user                 ‡ User-Mode on
        jsr      shtracks             ‡ afficher cette piste
        jsr      super                ‡ Supervisor on
        jsr      deselect             ‡ Désélectionner Floppy
        sf       flock                ‡ Autoriser Floppy-Interrupt
        jsr      user                 ‡ Mettre en User-Mode
        movem.l  (a7)+,a3-a6/d3-d7    ‡ Récupérer registre
        rts                          ‡ et retour

```

```

#####
‡ Envoi des paramètres pour l'affichage de la piste à la routine ‡
‡ showit                                                         ‡
#####

```

```

shtracks: move.w #0,head2
        move.l   #platztr,topptr
        move.w   #18,ligcount      ‡ 19 lignes à l'écran
        move.w   #100,prcount      ‡ 101 lignes pour l'imprimante
        move.w   #7680,maxdown
        move.w   #7888,maxup
        jsr      cursbuf
        jsr      clrest            ‡ Effacer le reste de l'écran

```

```

    jsr    showit          % afficher tampon avec handle
    jsr    videbuff        % des touches de curseur, etc.
    rts

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Lecture du champ de données sur disquette                                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

readadr: jsr    cursmess    % positionner curseur
          jsr    delline    % Effacer ligne
          move.l  #hilcurs,a0 % Afficher message
          jsr    printf
          move.w  #drive,d0
          cmp.w   #2,d0
          bgt     rdaderr
          jsr     super      % Supervisor on
          jsr     seldrive   % Sélectionner Drive courant
          jsr     fireset    % FDC reset
          jsr     searcht    % Rechercher deux fois la piste courante
          jsr     searcht    % fin que le drive soit en route
          jsr     setplatz   % Mettre DMA-Transfer-Adress
readadr: jsr     rdadr       % Lire champ d'adresse
          jsr     fireset    % FDC Reset
          jsr     user       % User-Mode on
          jsr     showadr    % Afficher champ d'adresse
          jsr     super      % Supervisor on
          jsr     deselect   % Désélectionner Drive courant
          jsr     user       % User-Mode on
          rts               % et retour

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                               %
% Lit 25 champs d'adresse sur disquette                                %
%                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

rdadr:   jsr     super      % Activer Supervisor-Mode
          move.w  #90,dramode % Toggle de la broche read- write

```

```

move.w    #$190,dmamode    * Effacer DMA-Status, Reset DMA
move.w    #$90,dmamode     * Chips, mettre en lecture Sectorcount-
move.w    #1,d6            * Register. Lire 1 Secteur
jsr       wrcontr          * pour FDC-Controller
move.w    #$80,dmamode     * Mettre en FDC-Register
move.w    #24,d4           * Lire 24+1 champs d'adresse
rdadr1:   move.w    #$c8,d6    * Instruction Read Address
move.l    #$40000,d7        * Compteur Time-out
jsr       wrcontr          * Instruction vers le FDC
rdadr2:   btst      #5,a7p     * Instruction déjà traitée ?
beq       rdadren1          * oui
subq.l    #1,d7             * sinon décrémenter Timeout
beq       rdaderr           * Fini ? alors erreur
bra       rdadr2            * sinon continuer
rdadren1: dbra      d4,rdadr1   * Répéter 25 fois
rts                          * et retour

rdaderr:  move.w    #-6,-(a7)   * Message d'erreur
jsr       errhand           * et quitter
rts

```

```

#####
* Affichage des champs d'adresse. On a lu plus de champs d'adresse *
* qu'afficher car le DMA-Controller transfère les octets par *
* groupe de 6 octets alors qu'un champ d'adresse contient 6 octets *
* *
#####

```

```

showadr:  jsr       cursmess    * Effacer position du curseur et
jsr       delline            * afficher message
move.l    #sacques1,a0
jsr       printf
jsr       cursbuf            * positionner curseur
move.w    #17,d5             * Afficher 18 Adressfields
move.l    #platztr,a3        * bufferadress du champ d'adresse
showadr1: move.w    #2,d4      * Envoyer 3 données (Track, face
move.w    #$20,-(a7)         * secteur) envoyer seulement un
jsr       conout              * espace
showadr2: move.b    (a3)+,d0   * Récupérer l'octet dans le tampon

```

```

move.w    d0,-(a7)      * mettre le mot sur la pile
jsr       dezpr         * Et afficher en décimal
move.w    #$20,-(a7)    * Envoyer deux espaces ensuite
jsr       conout
move.w    #$20,-(a7)
jsr       conout
dbr       d4,showadr2   * répéter deux fois
move.w    #$20,-(a7)    * Puis écrire 2 Spaces
jsr       conout
move.w    #$20,-(a7)
jsr       conout
move.b    (a3)+,d0      * Octet suivant du tampon
ext.w     d0            * (contient la taille du secteur)
move.w    #128,d1       * un 0 signifie 128 Byte/Sector
cap.w     #0,d0
beq       showadr7
move.w    #256,d1
cmp.w     #1,d0         * 1 égale 256 Byte/Sector
beq       showadr7
move.w    #512,d1
cmp.w     #2,d0         * 2 égale 512 Byte/Sector
beq       showadr7
move.w    #1024,d1      * sinon 1024 Byte/Sector par défaut
showadr7: move.w    d1,-(a7) * Afficher le nombre de Byte/Sector
jsr       dezpr         * en décimal
move.w    #$20,-(a7)    * Afficher un Space
jsr       conout
move.l    #spaces,a0    * plusieurs Spaces
jsr       printf
move.b    (a3)+,d0      * L'octet suivant du tampon est
move.w    d0,-(a7)      * la Checksum du champ d'adresse
jsr       hexpr         * qui est envoyé en hexa
move.b    (a3)+,d0      * Envoyer l'octet suivant du tampon
move.w    d0,-(a7)
jsr       hexpr         * en hexa

move.w    #13,-(a7)     * Carriage-Return plus Linefeed
jsr       conout        *
move.w    #10,-(a7)
jsr       conout

```

```

dbra    d5,showadr1    * Répéter 16 fois
jsr     wtast           * et Attendre touche
rts                                           * puis retour

```

```

#####
#####
* Subroutine de l'option CLUSTER du menu principal. les routines *
* accèdent à des routines de l'option INIT. Implémentez donc *
* le menu Init auparavant *
#####
#####

```

```

edclust: jsr    cursmess    * Cursor pos. ect.
          jsr    delline
          move.w #20,column
          move.w #2,ligne
          jsr    loccurs
          move.l #edques1,a0  * Message
          jsr    printf
          move.w #512,maxdown * Scroll up and down Variables
          move.w #720,maxup
          move.l #platztr,editptr * bufferadress
          jsr    editit      * Editer Cluster
          jsr    cursmess    * Effacer ligne Message
          jsr    delline
          jsr    cursmess
          move.l #clques1,a0  * Afficher message
          jsr    printf
          jsr    curleft     * trois fois gauche
          jsr    curleft     * Branchement à read
          jsr    curleft     * Sous-menus
          jsr    shclust     * affichage du Clusters
          rts               * et retour

```

```

#####
* décrémentation du du numéro de cluster dans le menu du cluster *

```

```

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

```

```

decclust: move.w    #1,-(a7)
              move.w    #11,-(a7)      * Tester Keyboard-Shift
              trap      #13            * Si une touche SHIFT a été pressée
              addq.l    #4,a7          * alors décrementation de
              btst      #0,d0          * 10, sinon décrementation de 1
              bne       deccshi        *
              btst      #1,d0
              bne       deccshi        * Shift pressée
              move.w    #1,d2          * sinon Shift non pressée
              bra       declst0        * et donc décrementer de 1
declshi: move.w    #10,d2              * décrementer de 10
declst0: move.w    wclust,d0          * numéro de cluster courant
              sub.w     d2,d0          * Soustraire décrement
              cmp.w     #0,d0          * 0 déjà dépassé
              blt       declst1        * oui
              bra       declst2        * non
declst1: move.w    maxclust,d0        * mémoriser numéro de cluster max comme
declst2: move.w    d0,wclust          * numéro de cluster
              ext.l     d0              * Il faut mettre le numéro de Cluster
              divu      #1000,d0       * dans le menu
              add.b     #'0',d0        * puis mettre en puissances de dix
              move.b    d0,miclusal    * par division et
              swap      d0              * ajouter au menu Cluster
              ext.l     d0
              divu      #100,d0
              add.b     #'0',d0
              move.b    d0,miclusal+1  * Mettre en 100'èmes
              swap      d0
              ext.l     d0
              divu      #10,d0
              add.b     #'0',d0
              move.b    d0,miclusal+2  * mettre en 10ièmes
              swap      d0
              add.b     #'0',d0
              move.b    d0,miclusal+3  * Et enfin l'unité
              jsr       dispmen        * afficher menu et
              rts          * retour

```



```

#####
* Incrément du numéro de cluster courant
#####

```

```

incclust: move.w    #-1, -(a7)
           move.w    #11, -(a7)      * tester Keyboard-Shift
           trap       #13            * comme avec decclust
           addq.l     #4, a7
           btst       #0, d0
           bne        incclshi
           btst       #1, d0
           bne        incclshi      *
           move.w     #1, d2          * pas une touche SHIFT alors 1
           bra        inclst0        * sert d'incrément
incclshi: move.w     #10, d2         * sinon Increment égale 10
inclst0:  move.w     wclust, d0      * Additionner Increment au numéro de
           add.w      d2, d0         * cluster et comparer avec numéro
           cmp.w      maxclust, d0   * maximum
           blt        inclst1        * Inférieur à numéro maximum
           move.w     #0, d0         * Prendre numéro 0
inclst1:  move.w     d0, wclust      * Mémoriser numéro courant
           ext.l      d0             * et le mettre dans le menu
           divu       #1000, d0      * en 1000ièmes
           add.b      #'0', d0
           move.b     d0, m1clusa1   * Insérer les 1000ièmes
           swap       d0
           ext.l      d0
           divu       #100, d0
           add.b      #'0', d0
           move.b     d0, m1clusa1+1 * Mettre les 100ièmes
           swap       d0
           ext.l      d0
           divu       #10, d0
           add.b      #'0', d0
           move.b     d0, m1clusa1+2 * Mettre les 10ièmes
           swap       d0
           add.b      #'0', d0
           move.b     d0, m1clusa1+3 * Mettre les unités

```

```

jsr    dispmen    * afficher menu et
rts                    * retour

```

```

*****
* Recherche le Cluster suivant le Cluster courant,          *
* S'il n'y en a pas, c'est indiqué                          *
*****

```

```

nextclst: move.w  wclust,d0    * Numéro de cluster courant
         move.w  d0,oldclst    * Mémoire temporaire
         jsr     findclst     * recherche du Cluster suivant
         move.w  newclst,d0    * Le Cluster suivant est ici
         tst.w   d0           * Ou un 0, qui signale
         beq     neclerr1     * une erreur
         cmp.w   #$ffB,d0     * Ou un caractère de fin,
         bge     neclerr1     * qui montre le dernier Cluster
         subq.w  #1,d0        * soustraire un. Pour une meilleure
         move.w  d0,wclust    * gestion de l'affichage des menus
         move.l  #3,revnum    * on peut appeler la routine incclust
         jsr     incclust     * qui affiche le cluster
         jsr     rdclust      * incrémenté puis lit
neclend: rts                    * ce Cluster

```

```

neclerr1: jsr     cursmess
         move.w  #-19,-(a7)    * Afficher le dernier Cluster
         jsr     errhand
         jsr     cursmess
         move.l  #clques1,a0
         jsr     printf
         bra     neclend      * et retour

```

```

*****

```

```

findclst: move.l  #fatbuf,a0    * Adresse du FAT-Buffer
         move.w  oldclst,d0    * Ancien numéro de cluster

```

```

    move.w    #3,d1          * fois 3, et
    mulu      d0,d1
    lsr.w     #1,d1          * divisé par 2, égale fois 1.5
    btst      #0,d0          * était l'ancien Clusternr. pair ou
    bne       impair        * impair (divisible ou non par 2)
pair: move.b  1(a0,d1.w),d0  * si pair alors récupérer mostsignificant
    lsl.w     #8,d0          * Nibble, décaler de 8 Bits vers
    or.b      0(a0,d1.w),d0  * la gauche et ordonner les
    and.w     #$0fff,d0      * Nibble restants
    move.w    d0,newclst     * Mémoriser comme nouveau numéro de cluste
r
    bra       ficlend        * et retour
impair: move.b 1(a0,d1.w),d0 * sinon récupérer most significant Nibble
    lsl.w     #8,d0          * et Nibble suivant, décaler de 8 Bits
    move.b    0(a0,d1.w),d0  * vers la gauche least significant Nibble
    lsr.w     #4,d0          * les 12-Bits supérieurs contiennent le
    and.w     #$0fff,d0      * numéro de cluster. donc décaler 4 Bit
    move.w    d0,newclst     * vers la droite. masquer, mémoriser
ficlend: rts                * et retour

```

```

#####
*   Inscrit le Cluster courant sur disquette après question   *
#####

```

```

wrclust: movem.l  a3-a5/d3-d5, -(a7) * Sauvegarder registre
    move.w    #0,column
    move.w    #2,ligne          * Afficher message
    jsr       loccurs
    move.l     #clques5,a0
    jsr       printf
    move.l     #misecta,a3      * Track courant
    move.w     #9,d3
wrcl1:  move.b  (a3)+,d0
    move.w     d0,-(a7)
    jsr       conout
    dbra      d3,wrcl1
    move.l     #m1clusa,a3      * et afficher Cluster pour question
    move.w     #12,d3

```

```

wrc12:  move.b    (a3)+,d0
        move.w    d0,-(a7)
        jsr      concut
        dbra     d3,wrc12
        move.l    #wrcques2,a0
        jsr      printf
        jsr      videbuff
        jsr      wtask          ‡ Vraiment écrire
        cmp.b     #'y',d0       ‡ oui
        beq      writc1st
        cmp.b     #'Y',d0       ‡ oui
        bne      wrclendi       ‡ sinon ne pas écrire
writc1st: move.w   wdrive,-(a7)   ‡ Envoyer Drive courant
        move.w    wclust,d0       ‡ Numéro de cluster courant
        sub.w     #2,d0           ‡ le numéro 2 est le premier cluster
        ruls      clsiz,d0        ‡ donnée. calculer numéro de secteur
        add.w     datrec,d0       ‡ logique
        move.w    d0,-(a7)        ‡ numéro de secteur logique du ST
        move.w    clsiz,-(a7)     ‡ Nombre de secteurs par Cluster
        move.l    #platztr,-(a7)  ‡ Adresse de départ du Cluster
        move.w    #3,-(a7)        ‡ Ecrire, ignorer changement de disquette
        move.w    #4,-(a7)        ‡ Rwabs
        trap      #13             ‡ BIOS-Trap
        add.l     #14,a7          ‡ restaurer la pile
        tst.w     d0              ‡ Erreur
        bmi      wrclster         ‡ Si oui alors traiter
wrclendi: jsr      cursmess
        jsr      delline          ‡ Sinon afficher message
        jsr      cursmess
        move.l    #clques1,a0
        jsr      printf
        move.l    (a7)+,a3-a5/d3-d5 ‡ Récupérer registre
        rts                          ‡ et retour

wrclster: move.w   d0,-(a7)        ‡ envoyer numéro d'erreur au
        jsr      errhand          ‡ ErrorHandler et afficher
        bra      wrclendi         ‡ et retour

wrclendi: jsr      cursmess

```

```

jsr    delline
jsr    cursmess
move.l #wrques3,a0
jsr    printf
jsr    videbuff
jsr    wtast
jsr    delline
bra    wrclend
rts

```

```

#####
# Lit le Cluster courant en mémoire. fonctionne également avec #
# le RAM-Disk #
#####

```

```

rdclust: move.l a3-a6/d3-d7,-(a7)    # Sauvegarder registre
rdcl0:  move.w wdrive,-(a7)    # Drive courant
        move.w wclust,d0        # Cluster courant
        subq.w #2,d0            # Calculer numéro de secteur logique
        muls    clsiz,d0
        add.w   datrec,d0
        tst.w   d0
        bpl     rdcl2            # supérieur à 0
        move.w  #0,d0            # Sinon prendre 0
rdcl2:  move.w  d0,logsect        # Memoriser numéro de secteur logique
        move.w  d0,-(a7)        # et mettre sur la pile
        move.w  #2,-(a7)        # Lire 2 secteurs
        move.l  #plat2tr,-(a7)  # bufferadress
        move.w  #0,-(a7)        #
        move.w  #4,-(a7)        # Instruction Rnabs
        trap    #13            # BIOS-Trap
        add.l   #14,a7          # Restaurer la pile
        tst.w   d0              # Erreur ?
        bmi     rdclster        # Si oui, afficher
        move.w  logsect,d0      # Numéro de secteur logique, en physique
        divs    #9,d0            # doit encore être modifié
        swap    d0              # recalculer
        addq.w   #1,d0          # Ajouter 1 au reste de la division

```

```

        move.w    d0,wsector    * Egale secteur physique
        swap     d0
        move.w    d0,d2        * Mémoriser résultat de la division
        move.w    #0,wside     * Face 0 par défaut
        move.w    nbrface,d1   * Nombre de faces
        cmp.w     #2,d1        * Si 2 faces
        bne      rdcl3
        lsr.w     #1,d0        * Alors diviser par 2
        btst     #0,d2        * Teste si résultat impair
        beq      rdcl4        *
        move.w    #1,wside

rdcl3:
rdcl4:  move.w    d0,wtrack     * Egale secteur physique
        jsr      secinmem      * Sector into Memory
        jsr      shclust       * Afficher Cluster
rdclend: jsr      cursmess
        move.l    #clques1,a0   * Afficher message
        jsr      printf
        movem.l   (a7)+,a3-a6/d3-d7 * Récupérer registre
        rts              * et retour

rdclster: jsr      initdriv     * Si erreur
        tst.l     d0           * Initialiser d'abord lecteur
        bne      rdcl0        * Si pas d'erreur, encore un
        move.w    d0,-(a7)
        jsr      errhand       * Doit encore être modifié
        bra      rdclend

secinmem: move.w    wside,d0    * Met le secteu courant dans le menu
        add.b     #'0',d0      * secteur pour affichages ultérieurs
        move.b     d0,mside
        move.w     wsector,d0
        ext.l     d0
        divs      #10,d0
        add.b     #'0',d0
        move.b     d0,msector
        swap      d0
        add.b     #'0',d0
        move.b     d0,msector+1 * low Byte du secteur

```

```

move.w    wtrack,d0
ext.l     d0
divs      #10,d0
add.b     #'0',d0
move.b    d0,mtrack
swap      d0
add.b     #'0',d0
move.b    d0,mtrack+1    * low Byte de la piste
rts

```

```

*****
* affiche le Cluster
*****

```

```

shclust: move.w    #0,head2      * Compteur d'octets
         move.w    #18,ligcount  * Il faut afficher 19 lignes
         move.w    #63,prcount   * si impression, 64 lignes
         move.l    #platztr,topptr * bufferadress
         move.w    #512,maxdown   * Limite de scrolling
         move.w    #720,maxup
         jsr       cursbuf        * positionner curseur et
         jsr       clrest         * Effacer le reste de l'écran
         move.w    #0,column      * Curseur sur dernière ligne de l'écran
         move.w    #24,ligne
         jsr       loccurs        *
         move.l    #clques2,a0
         jsr       printf
         jsr       showit         * Et afficher le Cluster
         rts                     * Enfin retour

```

```

*****
* affichage du Startcluster des fichiers se trouvant sur la disquette
* Les Startcluster sont indiqués dans le menu courant par <return>
* Si le nom de fichier en inversé est un sous-directory, on y
* pénètre

```

```

stclust: movem.l  a3-a5/d3-d7, -(a7)
        jsr      initdriv      * Initialiser Drive
        jsr      rdfat         * Lire les secteurs FAT- et Directory
        jsr      rddir         * dans le tampon désiré

stclst0: move.w   #0, column
        move.w   #2, ligne
        jsr      loccurs       * Positionner curseur
        move.l   #sciquesl, a0
        jsr      printf
        move.w   #17, ligcount * Il faut afficher 18 lignes
        jsr      delrest       * Effacer le reste de la ligne
        move.l   #dirbuf, a3   * Adresse du Directory-Buffers
        move.l   a3, a4        * Mémoire temporaire
        move.l   a3, topptr     * Utiliser comme pointeur
        move.l   a3, oldtop     * Rémémoriser
        jsr      cursbuf       * Cursor
        jsr      showdir       * Afficher 18 lignes
        jsr      cursbuf       * Cursor sur début
        move.l   #dirbuf, topptr * Début du Directory-buffers
        jsr      revon         * Mettre en Inversé
        jsr      dirlinl       * Premier nom de fichier (nom de disquette)

        jsr      revout        * Réécrire en inversé puis remettre
stclsti: jsr      touch         * en inversé et tester clavier
        swap     d0
        cmp.b    #$1c, d0       * Touche Return Pressée ?
        beq      dirclsel       * Si oui
        cmp.b    #$4B, d0       * cursor up?
        beq      stclup         * oui
        cmp.b    #$50, d0       * Cursor down?
        beq      stcldo         * oui
        cmp.b    #$4b, d0       * Cursor left
        beq      stclli         * oui
        cmp.b    #$4d, d0       * Cursor right
        bne      stclsti
        jsr      curright       * Oui alors appeler
        bra      stclendi

```



```

stcli:  jsr      curleft
        bra      stclend1

stclup:  move.w  ligne,d0      ‡ Ligne de curseur courante
        cmp.w   #4,d0         ‡ Ligne 4 égale limite supérieure
        ble     stclup3       ‡ égale 4, alors scrolling
        move.w  #0,column
        jsr     loccurs
        jsr     dirlin1       ‡ Sinon soustraire un de la ligne
        subq.w  #1,ligne      ‡ courante, mettre Curseur
        move.w  #0,column     ‡ sur nouvelles ligne et colonne
        jsr     loccurs       ‡ Mettre à 0
        jsr     revon
        jsr     dirlin1       ‡ Et afficher cette ligne en inversé
        jsr     revout        ‡ Mettre en inversé
        bra     stclupen      ‡ et retour

stclup3: cmp.l   #dirbuf,topptr ‡ Ligne supérieure du tampon ?
        beq     stclupen      ‡ Si oui, pas de scrolling mais retour

        move.l  topptr,d0     ‡ Sinon décrémenter pointeur du nombre de
        move.w  ligcount,d0   ‡ lignes fois nombre de caractères par
        addq.w  #1,d0         ‡ ligne. modifié le 18.8.86
        muls    #32,d0
        sub.l   d0,topptr     ‡ Décrémenter pointeur dans tampon
        jsr     showdir       ‡ Afficher 18 lignes
        move.w  #21,ligne
        move.w  #0,column     ‡ Dernière ligne en inversé
        jsr     loccurs       ‡ Cursor pos.
        jsr     revon         ‡ invers on
        jsr     dirlin1       ‡ Afficher ligne
        jsr     revout        ‡ Inversé OFF
        jsr     videbuff      ‡ modifié le 15.10.86
stclupen: bra     stclst1     ‡ Et retour à Loop

stcldo:  move.w  ligne,d0      ‡ Ligne courante supérieure à 20
        cmp.w   #20,d0
        bgt     stcldo3       ‡ oui

```

```

move.w  ligne,d0      ‡ Sinon additionner 1
addq.w  #1,d0         ‡
sub.w   #4,d0         ‡ Multiplier Offset de bord supérieur de
ext.l   d0
lsl.l   #5,d0         ‡ 1'écran par 32
move.l  topptr,a6     ‡ Pointeur sur Directorybuffer
move.b  0(a6,d0.l),d0 ‡ Récupérer premier octet de cette donnée
beq     stcldoen       ‡ Si Byte=0, Alors donnée vide
move.w  #0,column     ‡ sinon positionner curseur et
jsr     loccurs
jsr     dirlinl       ‡ afficher l'ancienne ligne normale
addq.w  #1,ligne      ‡ Icrémenter compteur de lignes
move.w  #0,column
jsr     loccurs
jsr     revon         ‡ Mettre la nouvelle ligne
jsr     dirlinl       ‡ en inversé
jsr     revout        ‡ Inversé OFF
jsr     loccurs
stcldol: bra    stcldoen ‡ et retour à Loop

stcldo3: move.w  ligne,d0 ‡ Récupérer premier octet de la
addq.w  #1,d0         ‡ donnée suivante du catalogue, additionne
r
sub.w   #4,d0         ‡ un et soustraire Offset de la bordure
ext.l   d0
lsl.l   #5,d0         ‡ supérieure fois 32 (Nombre d'octets par
move.l  topptr,a6     ‡ donnée). pointeur sur début de Dirbuffer
move.b  0(a6,d0.l),d0 ‡ Si donnée suivante égale 0
beq     stcldoen       ‡ alors retour à Loop
move.w  ligcount,d0   ‡ Nombre de lignes à afficher
addq.w  #1,d0         ‡ plus un
muls    #32,d0        ‡ fois 32 égale Offset du début de buffer
add.l   topptr,d0     ‡ additionner Offset au topptr
move.l  d0,topptr
jsr     cursbuf       ‡ Cursor pos.
jsr     showdir       ‡ Afficher Directory
jsr     cursbuf       ‡ Et mettre la première donnée en inversé
jsr     revon
jsr     dirlinl       ‡
jsr     revout

```

```

        jsr      videbuff      * modifié 15.10.86
stcldoen: bra      stclstl      * retour à Loop

stclendl: jsr      cursmess      * effacer ligne de message
        jsr      delline
        jsr      cursmess
        move.l    #clquesl,a0    * Afficher Mode
        jsr      printf
        movem.l   (a7)+,a3-a5/d3-d7  * Récupérer registre
        rts              * et retour

```

```

#####
* Réagit à une pression de la touche RETURN. Saisit ou      *
* affiche un Subdirectory                                     *
#####

```

```

dirclsel: move.l   topptr,a0      * Pointeur sur début courant du tampon
        move.w     ligne,d0
        sub.w      #4,d0          *
        ext.l      d0
        lsl.l      #5,d0          * fois 32
        move.b     11(a0,d0.l),d1 * récupérer File-Type Byte
        cmp.b      #$10,d1        * Est-ce un Subdirectory ?
        beq        subdir         * oui

```

```

dirsel1: move.w     clstnum,d0     * Sinon soustraire 1 du numéro de
        subq.w      #1,d0          * cluster courant à cause de incclust
        move.w      d0,wclust
        move.l      #3,revnum      * 3. Option en inversé
        jsr         incclust       * Incréments numéro de cluster
        bra         stclendl       * et afficher puis retour à Loop

```

```

#####

```

```

dirsel2: jsr      rddir          * Relire les secteurs de directory
        bra      subdiren

```

```

subdir:  tst.w    clstnum      * Si le numéro de cluster est 0 alors
        beq      dirsel2     * relire seulement les secteurs de
        move.w   clstnum,d0   * directory sinon lire le numéro de cluste
r
        move.l   #dirbuf,dirptr * de début du secteur logique du 2. subd
        clr.w    d3
subdir1: move.w   clstnum,d0
        move.w   wdrive,-(a7)  * Drive courant
        subq.w   #2,d0         * calculer Numéro de cluster dans numéro
        muls     clsiz,d0      * de secteur logique
        add.w    datrec,d0
        move.w   d0,-(a7)
        move.w   #2,-(a7)      * Lire 2 secteurs logiques
        move.l   dirptr,-(a7)  * bufferadress
        move.w   #2,-(a7)      * Lire quoi qu'il arrive
        move.w   #4,-(a7)      * BIOS fwords
        trap     #13           * BIOS Trap
        add.l    #14,a7
        tst.w    d0            * Erreur ?
        bni      subdierr      * Oui, alors traiter
        add.l    #1024,dirptr   * Sinon additionner 1024 Byte pro Cluster
        move.w   clstnum,oldclst * Mémoriser numéro de cluster
        jsr      findclst      * et evt. rechercher cluster suivant
        move.w   newclst,d0
        move.w   d0,clstnum
        tst.w    d0            * Cluster suivant trouvé ?
        beq      subdiren      * Sinon alors fin.
        cmp.w    #$ff8,d0      * N'était pas un caractère de fin
        bge      subdiren      * Si oui alors fin
        bra      subdir1       * Sinon lire le second

subdiren: bra     stclst0       * A Loop

subdierr: move.w  d0,-(a7)      * Traiter erreur
        jsr      errhand
        bra      stclend1

```

```

#####
* Affiche données de directory pour une face *

```

```

*****

```

```

showdir: move.w    #0,eflag
         jsr      cursbuf      ‡ Cursor pos.
         jsr      clrest      ‡ Effacer reste de l'écran
         move.l    topptr,a5    ‡ Pointeur dans Dirbuffer
         move.w    ligcount,d7 ‡ Nombre de lignes par face
showd1:  move.b    #' ',d0      ‡ Puis afficher spaces
         move.w    d0,-(a7)
         jsr      conout
         move.b    #' ',d0
         move.w    d0,-(a7)
         jsr      conout
         clr.l     d4
         move.w    #9,d6        ‡ Longueur des noms de fichiers avec EXT.
         move.b    0(a5,d4.l),d0 ‡ Teste si donnée DIR vide
         beq       showdien     ‡ Si oui alors fin
         addq.l    #1,d4        ‡ Sinon alors
         move.w    d0,-(a7)
         jsr      conout
showd2:  move.b    0(a5,d4.l),d0 ‡ Afficher le nom de fichier avec EXT.
         addq.l    #1,d4
         move.w    d0,-(a7)
         jsr      conout
         dbra     d6,showd2
         move.w    #20,tab1
         jsr      curstab      ‡ Alors afficher le Fileattribut
         jsr      disattr      ‡
         move.w    #40,tab1
         jsr      curstab
         jsr      disclus      ‡ enfin afficher Startcluster
         move.w    #55,tab1
         jsr      curstab
         jsr      dissize      ‡ et taille de fichier en octets
         move.w    #0,column
         addq.w    #1,ligne
         jsr      loccurs
         add.l     #32,a5        ‡ 32 Byte par donnée de Dir
         dbra     d7,showd1
showd8:  move.w    #0,column      ‡ Message de saisie en dernière ligne

```

```

    move.w    #24,ligne      ‡
    jsr      loccurs
    move.l    #sclques2,a0
    jsr      printf
    rts                          ‡ et retour

```

```

showdien: move.w    #1,eflag
          bra      showdB

```

```

#####
‡ Affiche une ligne de directory à l'écran ‡
#####

```

```

dirlin1: move.l    topptr,a3    ‡ Pointeur dans Dir-buffer
          move.w    #0,eflag
          move.w    ligne,d3
          sub.w     #4,d3        ‡ Offset de bordure supérieur de l'écran
          ext.l     d3
          lsl.l     #5,d3        ‡ fois 32, Correspond à une donnée de dir
          move.b     #' ',d4
          move.w     d4,-(a7)     ‡ deux Spaces
          jsr      conout
          move.w     d4,-(a7)
          jsr      conout
          move.b     0(a3,d3.l),d0 ‡ Premier octet de la donnée
          beq      dirzend1      ‡ Si 0 alors dernière donnée
          move.w     d0,-(a7)     ‡ Sinon afficher octet
          jsr      conout
          addq.l     #1,d3
          move.w     #9,d6        ‡ Taille restante de la donnée
dirlin1: move.b     0(a3,d3.l),d0 ‡ Récupérer octet restant de la donnée
          addq.l     #1,d3
          move.w     d0,-(a7)
          jsr      conout        ‡ Et afficher
          dbra      d6,dirlin1
          move.w     #20,tab1
          jsr      curstab
          jsr      disattr       ‡ Afficher le Fileattribut
          move.w     #40,tab1

```

```

jsr    curstab
jsr    disclus      * et le Startcluster
move.w #55,tbl1
jsr    curstab
jsr    dissize      * enfin la taille de fichier en Byte
dirzend: rts        * et retour

```

```

dirzend: move.w #1,eflag
        bra    dirzend

```

```

#####
* Affiche le numéro de cluster de début de la donnée DIR courante *
#####

```

```

disclus: move.b #' ',d0
        move.w d0,-(a7)
        jsr    conout      * Afficher Space
        move.w ligne,d0
        sub.w  #4,d0
        ext.l  d0
        lsl.l  #5,d0      * fois 32
        move.b 27(a3,d0.l),d1 * Accès à la donnée du Startcluster
        lsl.w  #8,d1      * fois 256 car High-Byte
        move.b 26(a3,d0.l),d1 * charger Low-Byte
        move.w d1,clstnum  * Mémoriser comme numéro de cluster
        move.w d1,-(a7)
        jsr    dezpr      * et afficher en décimal
        move.b #' ',d0    * Encore un Space et
        move.w d0,-(a7)
        jsr    conout
        rts              * retour

```

```

#####
* donne en octets la taille de fichier de la donnée DIR courante *
#####

```

```

dissize: move.w ligne,d3
        sub.w  #4,d3      * soustraire Offset à partir de la bordure

```

```

ext.l    d3          * d'écran supérieure
lsl.l    #5,d3        * fois 32
clr.l    d1
move.l    topptr,a3
move.b    31(a3,d3.l),d1 * Most significant Byte d'abord
lsl.l    #8,d1        * (Intel), déplacer octet dans Byte-Pos.

move.b    30(a3,d3.l),d1 * suivante. Charger next signif. Byte
lsl.l    #8,d1        * et déplacer d'une position d'octet
move.b    29(a3,d3.l),d1 * vers la gauche
lsl.l    #8,d1        * jusqu'à avoir les 4 octets de la
move.b    28(a3,d3.l),d1 * donnée et mettre en format
move.l    d1,-(a7)    * Motorola
jsr       dezlpr      * Puis afficher la taille en
move.w    #$20,-(a7)  * décimal. Puis encore un Space derrière
jsr       conout
rts                          * et retour

```

```

*****
* Donne le File-Attribut de la donnée de directory courante *
*****

```

```

disattr: move.w    ligne,d3      * Ligne courante
sub.w    #4,d3      * Offset de la bordure supérieure d'écran
ext.l    d3
lsl.l    #5,d3      * fois 32
move.l    topptr,a3  * pointeur dans Dir.-buffer
move.b    0(a3,d3.l),d1 * Prendre premier octet de la donnée
cmp.b    #$e5,d1    * Est-ce le caractère du fichier effacé ?
beq       deleted   * Si oui, afficher
move.b    11(a3,d3.l),d1 * Sinon récupérer attribut de fichier
cmp.b    #$10,d1    * Est-ce un Subdirectory? Si oui
beq       folder    * Alors afficher 'Subdirectory'
cmp.b    #$01,d1    * Ouvrir pour lecture
beq       readonly
cmp.b    #$02,d1    * S'agit-il d'un fichier caché ?
beq       hidden
cmp.b    #$08,d1    * Est-ce le nom de la disquette ?
beq       disname

```



```

        move.l    #treadr,a0    $ Default : le fichier peut être lu et
        jsr      printf         $ écrit
disatten: rts                  $ et retour

```

```

disname: move.l    #tdisname,a0  $ Afficher et
        jsr      printf
        bra      disatten        $ retour

```

```

folder:  move.l    #tfolder,a0   $ Afficher et
        jsr      printf
        bra      disatten        $ retour

```

```

readonly: move.l    #treadon,a0   $ Afficher
        jsr      printf
        bra      disatten        $ et retour

```

```

hidden:  move.l    #thidden,a0    $ afficher et
        jsr      printf
        bra      disatten        $ retour

```

```

deleted: move.l    #tdelet,a0     $ afficher
        jsr      printf
        bra      disatten        $ et retour

```

```

#####
#####
$ Subroutine format du sous-menu $
$ Ces routines accèdent au sous programmes de l'option $
$ TRACK with SYNCs. Il faut donc implémenter l'option TRACK with $
$ SYNCs auparavant puis seulement le Formatter $
#####
#####

```

```

format1: jsr      cursness        $ positionner curseur dans Messageline
        jsr      delline         $ Effacer ligne
        jsr      videbuff        $ Et effacer tampon clavier
        move.l    #foques2,a0    $ Tester si vraiment formater

```

```

jsr      printf
move.w   wtrack,-(a7)
jsr      dezpr
move.l   #foques5,a0
jsr      printf
move.w   wside,-(a7)
jsr      dezpr
move.l   #foques6,a0
jsr      printf
move.w   wdrive,-(a7)
jsr      dezpr
move.l   #foques3,a0
jsr      printf
jsr      wtast           * tester clavier
cmp.b    #'y',d0
beq       doform1
cmp.b    #'Y',d0
beq       doform1
jsr      delline        * Si pas Y
jsr      videbuff       * Alors pas de formatage
move.l   #foques4,a0
jsr      printf
jsr      wtast           * Attendre touche et
bra       formlend      * retour

```

```

doform1: move.w   #$e5e5,-(a7)    * sinon formater. valeur Virgin
move.l   #$87654321,-(a7)    * Magic number
move.w   #1,-(a7)           * sector-interleave
move.w   wside,-(a7)        * face courante
move.w   wtrack,-(a7)       * piste courante
move.w   asector,-(a7)      * Nombre courant de secteurs/piste
move.w   wdrive,-(a7)       * Drive courant
clr.l    -(a7)              * mot long Dummy
move.l   #formbuf,-(a7)     * Place pour créer la piste
move.w   #10,-(a7)          * XBIOS Flopfmt
trap     #14                * XBIOS Trap
add.l    #26,a7
tst.w    d0
bmi      form1err           * Erreur ?
formlend: jsr      cursmess

```

```

jsr    delline
jsr    cursmess
move.l #foques1,a0    * Afficher message
jsr    printf
jsr    curright    * Corriger curseur de menu
rts     * afficher menu et retour

formierr: move.w d0,-(a7)    * Numéro d'erreur sur pile, traiter
jsr    errhand
bra    formlend    * et retour

```

```

#####
*   Crée une piste avec tous les syncs à partir de l'adresse formbuf *
*   puis l'écrit avec writetr du Controlier pour le formatage      *
*   xformat dans menu                                              *
#####

```

```

maketr: move.w #1,sector    * Le premier secteur a le numéro 1
move.l #formbuf,a2    * Adresse du tampon, dans lequel la piste
move.w gap1,d0    * est crée. Mettre
move.w #$4e,d7    * octet de pre-piste est $4E
jsr    wpuff    * gap1 fois dasn le tampon
makt1: move.w gap2,d0    * Nombre d'octets
move.w #0,d7    * valeur de l'octet 0
jsr    wpuff    * Mettre dasn tampon
move.w #3,d0    * trois fois $F5 dans tampon comme Syncbyt

e
move.b #$f5,d7    * Et effacer le CRC-Register
jsr    wpuff    * (Checksum), Mettre dans tampon
move.b #$fe,(a2)+    * Adressmark directegent dans buffer
move.w wtrack,d0
move.b d0,(a2)+    * Ainsi que la piste courante
move.w wside,d0
move.b d0,(a2)+    * la face courante
move.w sector,d0
move.b d0,(a2)+    * Le secteur courant
move.w drbyte,d0    * Nombre d'octets par secteur
cmp.w #1024,d0

```

```

      beq      makt2
      cmp.w   #512,d0      * comparer avec les 4 valeurs possibles
      beq      makt3      * et mettre la
      cmp.w   #256,d0
      beq      makt4
      move.w   #0,d1
      bra      makt5
makt4: move.w   #1,d1
      bra      makt5
makt3: move.w   #2,d1
      bra      makt5
makt2: move.w   #3,d1
makt5: move.b   d1,(a2)+    * valeur obtenue
      move.b   #$f7,(a2)+  * Checksum of Adressfield in buff.
      move.w   gap3,d0     * nombre d'octets dans fenetre 3
      move.w   #$4e,d7     * Füllbyte est $4E
      jsr      wpuff       * Ecrire dans tampon
      move.w   gap2,d0     * Et mettre encore gap2 fois 0
      move.w   #0,d7       * Dans le tampon
      jsr      wpuff
      move.w   #3,d0       * 3 fois Syncbbyt, sont écrit comme A1
      move.w   #$f5,d7     * sur la disquette
      jsr      wpuff       * Ecrire $F5 dans tampon
      move.b   #$fb,(a2)+  * Dataaddressmark
      move.w   drbyte,d0   * Nombre de Byte/Sector sert de compteur
      move.b   #$e5,d7     * Pour les octets de données de ce
      jsr      wpuff       * secteur. Ecrire $E5 comme octet de
      move.b   #$f7,(a2)+  * donnée dans tampon. Ecrire Checksum
      move.w   gap4,d0     * gap5 fois $4E: octet de remplissage pour
      move.w   #$4e,d7
      jsr      wpuff       * bloc 4. Ecrire dans tampon
      move.w   sector,d0   * Incrémenter secteur de 1
      addq.w   #1,d0
      move.w   d0,sector
      cmp.w   asector,d0   * Comparer avec nombre de secteurs par pist
e
      ble      makt1      * Si supérieur alors
      move.w   gap5,d0     * Le dernier secteur est écrit dans le
      move.w   #$4e,d7     * tampon et on peut inscrire
      jsr      wpuff       * $4E dans 5ième vide

```

rts ‡ puis retour

```

#####
* Ecrit la valeur en octet dans le registre D7, et D0 fois dans le
* tampon adressé par le registre d'adresse A2
#####

```

```

wpuff:   subq.w    H1,d0      ‡ Adapter compteur
wpuff1:  move.b    d7,(a2)+   ‡ Ecrire dans tampon
        dbra      d0,wpuff1   ‡ D0 fois
        rts

```

```

#####
* Envoie l'adresse du tampon de piste au DMA-Controller, la
* Routine doit être appelée en Supervisor-Mode
#####

```

```

setbuf:  move.l    #formbuf,d0 ‡ Adresse du Trackbuffer
        move.b     d0,dmalow   ‡ Mettre Low-Byte
        lsr.l      #8,d0       ‡ Décaler de 8-Bit à droite
        move.b     d0,dmanid   ‡ Et mettre octet suivant
        lsr.l      #8,d0       ‡ Décaler 8 bit à droite
        move.b     d0,dmahigh  ‡ et mettre Highbyte
        rts

```

```

#####
* Formate une piste en inscrivant le contenu du registre de piste
* ( dans formbuf) directement sur disquette avec l'instruction
* write track du contrôleur
#####

```

```

xfortrac: move.w    #$190,dmamode ‡ Effacer DMA et inscrire tampon
        move.w     #$90,dmanode  ‡ désactiver
        move.w     #$190,dmamode
        move.w     #$1f,d6       ‡ Mettre 31 dans le registre
        jsr        wrcontr      ‡ de compteur de secteurs

```

```

        move.w    #$1B0,dmamode    ‡ Sélectionner FDC-Register
        move.w    #$f8,d6         ‡ Instruction write-Track
        jsr       mrcontr
        move.l    #$60000,d7      ‡ décrémenter Compteur Time-out
xforti:  subq.l    #1,d7            ‡
        beq       xforterr        ‡ Si arrivé à la fin alors erreur
        btst      #5,mfp          ‡ FDC Déjà prêt ?
        bne       xfort1          ‡ Sinon continuer attente
        rts                     ‡ Déjà retour

```

```

xforterr: move.w   #-24,-(a7)      ‡ Numéro d'erreur sur pile
        jsr       errhand         ‡ traiter
        rts

```

```

*****
‡ Appele la Routine pour le formatage direct d'une piste
*****

```

```

xformat: move.l    a3-a6/d3-d7,-(a7)
        jsr       cursmess
        jsr       delline        ‡ Sauvegarder registre et afficher Message
        move.l    #xfques1,a0    ‡
        jsr       printf
        jsr       videbuff       ‡ Vider tampon clavier et
        jsr       wtast          ‡ Attendre touche
        cmp.b     #'y',d0
        beq       xformit
        cmp.b     #'Y',d0
        bne       xformend      ‡ Si pas y ni Y

```

```

xformit: jsr       super          ‡ sinon Supervisor on
        st        flac1          ‡ Remarquer Floppy-Interrupt
        jsr       setplatz       ‡ Il faut lire une fois la piste
        jsr       seldrive       ‡ Pour améliorer la lecture
        jsr       flreset        ‡ car la vitesse de la disquette
        jsr       searcht        ‡ ne serait pas suffisante pour
        jsr       rdstrack       ‡ les pistes intérieures.
        jsr       setbuf         ‡ Envoyer Trackbuffer à DMA-Co.
        jsr       maketr         ‡ Créer piste courante dans
        jsr       searcht        ‡ tampon et rechercher piste

```

```

jsr    xfortrac      ‡ Puis écrire piste sur disquette
jsr    videbuff
jsr    cursmess
jsr    delline
jsr    flreset       ‡ Reset du Controller
jsr    user          ‡ Remettre en User-Mode
move.l #x4ques2,a0
jsr    printf        ‡ Afficher message
jsr    wtast         ‡ tester clavier
jsr    super         ‡ Mettre en Supervisor
sf     flock         ‡ Autoriser Floppy-Interrupt
jsr    deselect     ‡ Désélectionner lecteur
jsr    user          ‡ Activer User-Modus

xformend: jsr    cursmess
jsr    delline       ‡ Afficher message, récupérer
jsr    cursmess      ‡ registre et
move.l #drques1,a0
jsr    printf
movem.l (a7)+,a3-a6/d3-d7
rts                ‡ retour

*****
‡ Les options suivantes autorisent l'incrémentation et la ‡
‡ décrémentation des gaps dans le menu. Voir menu sector pour ‡
‡ les particularités ‡
*****

incgaps: cmp.w    #99,d0      ‡ Nombre max d'octets de remplissage
        blt      incgaps1    ‡ de tous les gap = 99, cette Routine
        move.w   #0,d0       ‡ est appelée par tous
        bra      incgaps2    ‡ les menus incgap car les limites
incgaps1: addq.w  #1,d0       ‡ sont les mêmes
incgaps2: rts

incgap1: move.w   gap1,d0
        jsr      incgaps
        move.w   d0,gap1

```

```
divu    #10,d0
add.b   #'0',d0
move.b  d0,mgap1
swap    d0
add.b   #'0',d0
move.b  d0,mgap1+1
jsr     dispmen
rts

incgap2: move.w  gap2,d0
        jsr     incgaps
        move.w  d0,gap2
        divu    #10,d0
        add.b   #'0',d0
        move.b  d0,mgap2
        swap    d0
        add.b   #'0',d0
        move.b  d0,mgap2+1
        jsr     dispmen
        rts

incgap3: move.w  gap3,d0
        jsr     incgaps
        move.w  d0,gap3
        divu    #10,d0
        add.b   #'0',d0
        move.b  d0,mgap3
        swap    d0
        add.b   #'0',d0
        move.b  d0,mgap3+1
        jsr     dispmen
        rts

incgap4: move.w  gap4,d0
        jsr     incgaps
        move.w  d0,gap4
        divu    #10,d0
        add.b   #'0',d0
        move.b  d0,mgap4
        swap    d0
```



```

add.b    #'0',d0
move.b   d0,mgap4+1
jsr      dispmen
rts

incgap5: move.w    #-1,-(a7)
         move.w    #11,-(a7)      * Keyboard-Shift pressée ?
         trap      #13
         addq.l    #4,a7
         move.w    #10,d1
         btst      #0,d0
         bne       incgap5x
         btst      #1,d0
         bne       incgap5x
         move.w    #1,d1
incgap5x: move.w    gap5,d0
         add.w     d1,d0
         cmp.w     #999,d0
         ble       incgap5a
         move.w    #0,d0
incgap5a: move.w    d0,gap5
         ext.l     d0
         divs      #100,d0
         add.b     #'0',d0
         move.b    d0,mgap5
         swap      d0
         ext.l     d0
         divs      #10,d0
         add.b     #'0',d0
         move.b    d0,mgap5+1
         swap      d0
         add.b     #'0',d0
         move.b    d0,mgap5+2
         jsr      dispmen
         rts

#####

decgaps: cmp.w     #0,d0          * est appelé par toute les options decgap
         ble       decgaps1      * car le nombre max et min

```

```
        subq.w  #1,d0          * est le même pour tous les gaps
        bra     decgaps2
decgaps1: move.w  #99,d0
decgaps2: rts

decgap1: move.w  gap1,d0
        jsr     decgaps
        move.w  d0,gap1
        divu    #10,d0
        add.b   #'0',d0
        move.b  d0,mgap1
        swap    d0
        add.b   #'0',d0
        move.b  d0,mgap1+1
        jsr     dispmen
        rts

decgap2: move.w  gap2,d0
        jsr     decgaps
        move.w  d0,gap2
        divu    #10,d0
        add.b   #'0',d0
        move.b  d0,mgap2
        swap    d0
        add.b   #'0',d0
        move.b  d0,mgap2+1
        jsr     dispmen
        rts

decgap3: move.w  gap3,d0
        jsr     decgaps
        move.w  d0,gap3
        divu    #10,d0
        add.b   #'0',d0
        move.b  d0,mgap3
        swap    d0
        add.b   #'0',d0
        move.b  d0,mgap3+1
        jsr     dispmen
        rts
```

```

decgap4: move.w    gap4,d0
        jsr       decgaps
        move.w    d0,gap4
        divu      #10,d0
        add.b     #'0',d0
        move.b    d0,mgap4
        swap      d0
        add.b     #'0',d0
        move.b    d0,mgap4+1
        jsr       dispmen
        rts

```

```

decgap5: move.w    #-1,-(a7)
        move.w    #11,-(a7)
        trap      #13
        addq.l    #4,a7
        move.w    #10,d1
        btst      #0,d0
        bne       decgap5x
        btst      #1,d0
        bne       decgap5x
        move.w    #1,d1

```

! touche SHIFT de droite

```

decgap5x: move.w    gap5,d0
        sub.w     d1,d0
        bpl       decgap5a
        move.w    #999,d0

```

```

decgap5a: move.w    d0,gap5
        ext.l     d0
        divs      #100,d0
        add.b     #'0',d0
        move.b    d0,mgap5
        swap      d0
        ext.l     d0
        divs      #10,d0
        add.b     #'0',d0
        move.b    d0,mgap5+1
        swap      d0

```

```

add.b    #'0',d0
move.b   d0,mgap5+2
jsr      dispmen
rts

```

```

#####
* modifie le nombre d'octets par secteur qui est mémorisé dans drbyte*
* et qui influe sur le nombre d'octets affichés et écrits dans le   *
* menu sector. Naturellement seulement si le module Format est     *
* implémenté                                                         *
#####

```

```

incbyte: move.w   drbyte,d0      * Nombre possible de Byte/Sector
        cmp.w     #128,d0        * est 128, 256, 512 ou 1024 Byte
        beq       incby1
        cmp.w     #256,d0
        beq       incby2
        cmp.w     #512,d0
        beq       incby3
        move.w    #128,d0
        move.b    #'0',mdrisekt  * Mettre aussi dans texte du menu
        move.b    #'1',mdrisekt+1
        move.b    #'2',mdrisekt+2
        move.b    #'8',mdrisekt+3
        bra       incbywei

```

```

incby1: move.w    #256,d0
        move.b    #'0',mdrisekt
        move.b    #'2',mdrisekt+1
        move.b    #'5',mdrisekt+2
        move.b    #'6',mdrisekt+3
        bra       incbywei

```

```

incby2: move.w    #512,d0
        move.b    #'0',mdrisekt
        move.b    #'5',mdrisekt+1
        move.b    #'1',mdrisekt+2
        move.b    #'2',mdrisekt+3

```

```

bra      incbywei
incby3:  move.w  #1024,d0
         move.b  #'1',mdrisekt
         move.b  #'0',mdrisekt+1
         move.b  #'2',mdrisekt+2
         move.b  #'4',mdrisekt+3

```

```

incbywei: move.w  d0,drbyte
         jsr      dispmen
         rts

```

```

#####
*  décrémente le nombre octet/secteur et ne laisse que les 4 valeurs *
*  possibles du FDC (128, 256, 512, 1024) pour incbyte          *
#####

```

```

decbyte: move.w  drbyte,d0
         cmp.w   #128,d0
         beq     decby1
         cmp.w   #256,d0
         beq     decby2
         cmp.w   #512,d0
         beq     decby3
         move.w  #512,d0
         move.b  #'0',mdrisekt
         move.b  #'5',mdrisekt+1
         move.b  #'1',mdrisekt+2
         move.b  #'2',mdrisekt+3
         bra     decbywei

```

```

decby1:  move.w  #1024,d0
         move.b  #'1',mdrisekt
         move.b  #'0',mdrisekt+1
         move.b  #'2',mdrisekt+2
         move.b  #'4',mdrisekt+3
         bra     decbywei

```

```

decby2:  move.w  #128,d0
         move.b  #'0',mdrisekt

```

```
move.b    #'1',mdrisekt+1
move.b    #'2',mdrisekt+2
move.b    #'8',mdrisekt+3
bra       decbywei
```

```
decby3:   move.w    #256,d0
move.b    #'0',mdrisekt
move.b    #'2',mdrisekt+1
move.b    #'5',mdrisekt+2
move.b    #'6',mdrisekt+3
```

```
decbywei: move.w    d0,drbyte
jsr       dispmen
rts
```

L'utilisation du moniteur de disquettes

Le contrôle du moniteur de disquettes se fait presque entièrement avec les touches du curseur. Curseur-gauche et curseur-droite sélectionnent les différentes options, curseur-haut et curseur-bas exécutent ces options ou modifient les paramètres (drive, side, track, etc). Dans certaines options, la sélection conduit à un nouveau menu qui offre différentes options.

Voici l'explication des différentes options :

7.2.1 Le menu principal

Toutes les options du menu principal à l'exception de l'option "FIN" conduisent à un nouveau menu.

TRACK :

Sélectionne le menu track qui permet le traitement de pistes complètes.

TRACK/SYNCS :

Sélectionne le menu track-with-sync. Ce mode permet d'avoir accès à toutes les informations de la disquette, c'est-à-dire aux GAP et aux octets de synchronisation par exemple.

SECTOR :

Sélectionne le mode sector qui permet la lecture et la modification de secteurs.

CLUSTER :

Sélectionne le mode cluster qui permet d'avoir accès aux clusters sur disquette.

FORMAT :

Sélectionne le menu FORMAT qui permet de formater des pistes dans différents formats, même ceux qui ne sont pas du standard ATARI.

OPTIONS :

Sélectionne le menu options qui permet de définir le lecteur utilisé. De plus, on peut définir le nombre maximum de pistes et de secteurs.

FIN :

Interrompt le programme et revient au desktop.

7.2.2 Le menu TRACK

Le menu TRACK contient plusieurs sous-menus :

- drive : 0* Si on sélectionne cette option, on peut changer de numéro de lecteur avec les touches curseur-haut et curseur-bas. Toutes les options qui sont suivies d'un ":" et d'un chiffre offrent cette possibilité de modification avec les touches du curseur.
- side : 0* Sélection de la face 0 ou 1.
- track : 00* Sélection de la piste à laquelle on veut avoir accès. Le numéro maximum est défini dans le menu options.
- Sect/Trac : 00* Détermine le nombre de secteurs à lire ou à écrire par piste.
- READ :* Cette option charge la piste définie et l'affiche. Curseur-haut et curseur-bas permettent d'effectuer un scrolling sur les différents secteurs de cette piste. Il est également possible d'appeler la fonction edit avec les touches curseur-droite et curseur-gauche et de modifier ainsi la piste.
- WRITE :* La piste est reproduite sur la disquette après une question de confirmation.
- EDIT :* Offre la possibilité d'éditer un secteur de la piste. Elle peut être appelée à partir de la fonction READ. On ne peut éditer qu'un secteur à la fois.
- BACK :* Revient au menu principal.

7.2.3 Le menu TRACK-WITH-SYNC

Voici les sous-menus :

<i>drive</i> : 0	Sélection du lecteur
<i>side</i> : 0	Sélection de la face
<i>track</i> : 00	Sélection de la piste

READ WITH SYNC : Lit toute la piste avec les gaps et permet le scrolling sur cette piste

ADDR.FIELD : Affiche les champs d'adresse de la piste avec la taille en octets et la checksum. L'affichage se fait partiellement en double car 16 champs d'adresse sont affichés à la fois.

BACK : Retour au menu principal.

7.2.4 Le menu sector

Voici les différents sous-menus :

<i>drive</i> : 0	Sélection du lecteur
<i>side</i> : 0	Sélection de la face
<i>track</i> : 00	Sélection de la piste
<i>sector</i> : 00	Sélection du secteur

READ : Lecture du secteur et affichage.

WRITE : Ecriture du secteur se trouvant en mémoire dans le secteur affiché momentanément après une question de confirmation.

EDIT : Mode édition qui permet la saisie de valeurs hexadécimales et la modification de tous les octets du secteur avec les touches du curseur. Le mode édition est quitté en pressant la touche RETURN.

BACK : Retour au menu principal.

7.2.5 Le menu Cluster

Les différents sous-menus :

drive : 0 Sélection du lecteur

clust : 0000 Sélection du cluster

READ : Lit le cluster courant et l'inscrit en mémoire. Calcule le secteur physique, la piste et la face auxquels ce cluster commence. Les informations concernant le secteur physique sont indiquées dans le menu sector. On peut lire immédiatement le startsector de ce cluster en sélectionnant READ.

NEXT : Calcule le cluster suivant du fichier à l'aide de la File Allocation Table et le charge en mémoire.

WRITE : Ecrit le cluster se trouvant en mémoire dans le cluster courant affiché par le menu. Une question demande confirmation.

EDIT : Permet l'édition d'un cluster, RETURN quitte l'option EDIT.

STARTofFILE : Affiche tous les fichiers du lecteur courant avec les attributs correspondants. Le scrolling entre les fichiers est possible grâce aux touches curseur-haut et curseur-bas. Si on presse la touche RETURN, le startcluster du fichier sélectionné sur le moment est mis dans le menu cluster. Si le fichier sélectionné représente un sous-catalogue, on entre dans ce catalogue et on peut sélectionner un fichier. Pour revenir à la racine, il suffit de sélectionner un point.

BACK : Retour au menu principal.

7.2.6 Le menu FORMAT

Les options du menu format :

<i>drive</i> : 0	Sélection du lecteur
<i>side</i> : 0	Sélection de la face
<i>track</i> : 00	Sélection de la piste
<i>Sec/Track</i> : 00	Sélection du nombre de secteurs par piste. Le nombre maximal dépend de l'option MAXSECT du menu option. Si MAXSECT est de 10, sec/track peut être également de 10.
<i>FORMAT</i> :	Formatage de la piste courante avec sec/track secteurs après une question de confirmation. Le formatage se fait avec la fonction XBIOS et formate les secteurs avec 512 Octets (format ATARI).
<i>XFORMAT</i> :	Formatage de la piste avec les paramètres modifiés dans le menu GAP, ce qui permet de déterminer non seulement le nombre d'octets/secteur mais également le nombre d'octets de synchronisation.
<i>GAPS</i> :	Offre un sous-menu qui permet de modifier des paramètres pour XFORMAT.
<i>BACK</i> :	Retour au menu principal.

7.2.7 Le menu GAP

Les options du menu GAP :

<i>GAP1</i> : 00	Détermine le nombre d'octets de remplissage au début de la piste. Au plus : 99.
<i>GAP2</i> : 00	Détermine le nombre d'octets nuls.
<i>GAP3</i> : 00	Détermine le nombre d'octets pour le GAP3.

- GAP4 : 00* Détermine le nombre d'octets pour le GAP4.
- GAP5 : 00* Détermine le nombre d'octets de remplissage qui sont insérés en fin de piste.
- Byt/SECT :* Permet de déterminer le nombre d'octets par secteurs possibles avec le contrôleur (128, 256, 512, 1024). Le choix influe sur la lecture de secteurs dans le menu sector. Cela permet d'éditer des secteurs de taille différente.
- BACK :* Retour au menu principal.

7.2.8 Le menu option

Les options du menu option :

- drive : 0* Sélection du lecteur courant.
- MAXTRACK : 00* Sélection du nombre maximum de pistes possible pour le menu track.
- MAXSECTOR : 00* Sélection du nombre maximum de secteurs pour le menu sector. Définit également le nombre maximum de secteurs par piste.
- INIT DRIVE :* Appel de la fonction BIOS, affiche le disk parameter.
- SHOW BPB :* Affiche le BIOS parameter bloc du lecteur courant.
- BACK :* Retour au menu principal.

7.3 EXEMPLE D'UTILISATION DE L'ÉDITEUR DE DISQUETTES

En premier lieu, nous allons tester l'éditeur de disquettes sur le catalogue et la FAT. Pour cela, formatez une disquette simple face avec le nom WORK.TST et copiez l'accessoire DESK2.ACC de la disquette système. Il a une taille de 6258 octets (TOS dernière version).

Comme nous l'avons déjà indiqué GEMDOS divise les disquettes en blocs (clusters) de deux secteurs de chacun 512 octets, soit en tout 1024 octets. Six fois 1024 donne 6144, le DESK2.ACC ne tient donc pas sur 6 clusters mais utilise des octets du 7^{ième} cluster. Chargez l'éditeur de disquettes et déterminez le numéro de lecteur, track 1, side 0 et sector 3. Sur ce secteur se trouve le catalogue. Sélectionnez l'option READ et regardez le résultat. Vous pouvez également imprimer le secteur en pressant sur la touche "P".

Analysons le résultat de cette action. Vous avez en premier le nom choisi lors du formatage. Chaque donnée du catalogue, dont le nom de la disquette, tient exactement sur 32 octets. Le nom de la disquette utilise les octets 0 à 31 et la première donnée du catalogue commence à l'octet 32 (hexa \$20). Les 11 premiers octets (0 à 10) de chaque donnée du catalogue sont réservés au nom de fichier et les lettres non utilisées des 8 premiers octets sont remplies par des espaces (\$20). Les trois octets suivants (8-10) représentent l'extension.

Les données suivantes sont au format INTEL. Ainsi, si les données tiennent sur plus d'un octet, l'octet de poids faible est inscrit avant l'octet de poids fort. Le mot \$1234 est donc représenté sous la forme \$34 \$12.

Le douzième octet (numéro 11) est l'attribut de fichier et détermine les modes d'accès au fichier. \$08 près le nom de disquette indique qu'il s'agit ici du nom de la disquette.

On a ensuite 11 octets (12 à 21, \$0C-\$15) qui n'ont pas de signification particulière. Les octets 22 et 23 (\$16 et \$17) contiennent l'heure à laquelle le fichier a été inscrit sur disquette pour la dernière fois. L'heure est codée sous formes de bits comme nous vous l'avons indiqué dans le chapitre 3.3. Remarquons que les secondes sont incrémentées par pas de 2. Ainsi, il faut multiplier la valeur des secondes par 2.

A la suite des octets indiquant l'heure se trouvent les octets indiquant la date (24-25, \$18 \$19) qui sont codés de la même manière.

Nous en arrivons aux deux octets les plus importants d'une donnée du catalogue, les octets 26 et 27 (\$1A, \$1B). Ils indiquent le cluster de départ du fichier. Comme vous le voyez vous-même, notre fichier a pour start cluster \$02 \$00 ce qui donne le cluster \$0002 qui est le premier cluster libre d'une disquette pour le système d'exploitation. Le fichier DESK2.ACC commence donc au cluster numéro 2. Pour transformer le numéro de cluster en numéro de secteur logique et physique, il faut retrouver certains paramètres de la BPB et du boot secteur. Le calcul s'effectue comme suit :

1. Soustraction de 2 du numéro de cluster car le cluster numéro 2 est le premier cluster libre.
2. Multiplication par le nombre de secteurs par cluster (clsize = 2 sur l'ATARI ST).
3. Addition du numéro du premier secteur logique (datrec = 18 sur l'ATARI ST).

Le nombre obtenu représente le numéro logique de secteur du premier secteur de ce cluster. Le deuxième secteur du cluster est celui qui suit. (Sur les disquettes simple face). Sur les disquettes double face, le premier secteur du premier cluster de données se trouve sur la face 0 piste 1 secteur 1, le deuxième secteur du cluster est sur le secteur 2 de la piste 1 face 0. Les clusters sont donc constitués dans ce cas également de secteurs consécutifs. Mais quand on atteint le dernier secteur d'une piste sur la face 0, le système d'exploitation inscrit le secteur suivant sur la même piste de la face 1. Si on a des pistes de 9 secteurs comme sur l'ATARI ST (format normal), le secteur physique du 5 ième cluster se trouve donc sur la face 0 piste 1 secteur 9 et le deuxième secteur est sur la face 1 piste 1 secteur 1.

Revenons au secteur logique du start cluster. Il manque encore le calcul de la piste et du secteur physique sur lequel se trouve se secteur logique. Reprenons l'exemple d'une disquette simple face et divisons les numéros de secteurs logiques par le nombre de secteurs par piste (spt = 9 format ATARI). Le résultat de cette division est la piste, le reste de la division est le début du secteur 1 de cette piste.

Exemple du premier fichier : cluster numéro 2 moins 2 donne 0, fois 2 secteurs par cluster donne 0, plus 18 égale 18, divisé par 9 secteurs par piste égale 2, reste 0 : le premier secteur du premier cluster du fichier "DESK2.ACC" est sur la piste 2 secteur 1.

Pour une disquette double face, le calcul reste le même jusqu'au secteur logique mais l'obtention de la piste et du secteur physiques dépend de la face : le numéro de secteur logique est divisé de nouveau par le nombre de secteurs par piste (spt) et le reste de cette division donne le début du secteur par rapport au secteur 1. Seul le calcul de la piste diffère un peu. Le résultat de la division est divisé par le nombre de face. Si la division ne donne pas de reste, le résultat est la piste de ce secteur sur la face 0. S'il y a un reste, la piste calculée se trouve sur la face 1.

Pour calculer le secteur physique du cluster 2 sur une disquette formatée en double face, voici comment faire. Le calcul du secteur logique n'est pas modifié : on obtient également 18, puis on divise par le nombre de secteurs par piste (9) et on obtient 2, reste 0. Le numéro de secteur est donc 1 (car il n'y a pas de reste). Pour obtenir la piste et la face, on divise le 2 qu'on vient d'obtenir par le nombre de faces (2), ce qui donne 1, reste 0. Ainsi, le secteur se trouve sur la piste 1 face 0.

Pour continuer, nous allons retrouver le start sector du catalogue pour des disquettes simple et double faces. Le secteur de départ logique est obtenu à partir du BPB et du boot secteur. Pour cela, on additionne le nombre de secteurs réservés (res = 1 sur l'ATARI) au produit du nombre de FAT (FAT = 2) et du nombre de secteurs par FAT (spf = 5). Le secteur logique est donc $1+2*5=11$. Le début du directory sur une disquette ATARI est donc sur le secteur 11. Sur une disquette simple face, on a pour le secteur physique :

11 divisé par 9 donne 1 (secteurs/piste), reste 2. Le secteur est 1 plus reste = 3. Le secteur logique 11 se trouve sur la piste 1 pour une disquette simple face. Secteur 3 pour une disquette en double face :

11 divisé par 9 égale 1, reste 2. Le secteur est 1 plus reste = 3, 1 divisé par 2 donne 0, reste 1. Il existe un reste => face 1, piste égale le reste de la première division (1).

Le secteur logique 11 se trouve sur les disquettes double face sur la face 1 piste 0 secteur 3.

Dans le calcul de cluster, nous avons perdu de vue les données du catalogue pour le fichier "DESK2.ACC". Il manque en effet quatre octets pour faire 32 octets. (28-31, \$1C-\$1F). Ils indiquent la taille du fichier en octets. si on a \$72 \$18 \$00 \$00, cela correspond à \$00001872 (format INTEL) ce qui donne en décimal 6258 (cela est ce qu'on trouve dans le desktop).

Avant de résumer ce que nous venons de dire dans un tableau, il faut expliquer deux octets qui ont une fonction particulière : c'est le premier octet de la donnée (octet 0) et le douzième octet (octet 11), l'octet d'attribut.

Le premier octet du nom, dans notre exemple \$44 correspond au code ASCII de la première lettre du nom de fichier sauf s'il est égal à \$E5, \$00 et \$2E.

Ces valeurs ont les significations suivantes :

\$E5 : Ce fichier a déjà été utilisé mais il a été effacé.

\$2E : Cet octet indique le chemin du sous-directory. Si l'octet suivant est également \$2E, le champ du numéro de cluster contient le numéro de cluster du directory supérieur. Si le deuxième octet est \$00, le directory supérieur est la racine. Nous expliquerons plus précisément cela par la suite.

L'octet d'attribut (octet 11) peut prendre les valeurs suivantes :

\$00 : Ce fichier peut être lu et écrit.

\$01 : Ce fichier peut seulement être lu (read only)

\$02 : Ce fichier n'est pas présent dans le catalogue (hidden)

\$08 : Cela représente le nom de la disquette, tous les octets après le 10 n'ont aucune signification.

\$10 : Il s'agit d'un sous-directory dans le cas où on est sur un nom de fichier.

Signification des 32 octets de chaque donnée du directory :

Octet	Signification
0-10	Nom de fichier avec l'extension, le premier octet est éventuellement l'état (\$E5, \$2E).
11	Attribut du fichier (Read/write, readonly, subdirectory).
12-21	Inutilisé
22-23	Heure
24-25	Date
26-27	Cluster de début du fichier
28-31	Taille du fichier en octets

7.3.1 File Allocation Table

Vous savez maintenant comment trouver le début d'un fichier sur disquette : il suffit de transformer les octets numéros 26 et 27 de la donnée du directory en décimal. Le numéro de cluster ainsi obtenu est transformé en numéro de secteur logique selon la méthode décrite précédemment. Mais où se trouve le cluster suivant du fichier, et le dernier ?

C'est la File Allocation Table qui répond à toutes ces questions. Le début de cette FAT se trouve toujours en face 0, piste 0, secteur 2 (simple et double face). Pour mieux comprendre, lisez donc le secteur 2 de la face 0 piste 0 de votre disquette formatée. Vous connaissez déjà le nom de fichier. Utilisez un moniteur de disquette. Vous reconnaissez une structure déjà vue à partir des 3 premiers octets (\$F7, \$FF, \$FF).

Cette FAT contient les informations concernant chaque cluster de la disquette : s'il est utilisé et si oui, quel est le cluster suivant du fichier. Utilisons pour l'instant la structure simplifiée (03 40 00 05) et voyons les chiffres 3, 4 et 5. Supposons que le tableau a l'apparence suivante :

Début : 1 2 3 4 5

Le début symbolise l'adresse à laquelle se trouve le numéro 1. L'ensemble forme une liste linéaire dont l'élément numéro 0 a pour valeur 1, l'élément numéro 1 a pour valeur 2, etc...

Si nous lisons l'adresse Début, nous trouvons la valeur 1. Ce chiffre 1 représente l'information sur l'élément suivant de la liste. Pour obtenir l'adresse, nous additionnons le chiffre 1 à l'adresse de Début et nous lisons ce qu'il y a à l'adresse obtenue (Début + 1). Cela nous donne le chiffre 2.

Ce 2 indique : le numéro de l'élément suivant de la liste se trouve à l'adresse Début+2. C'est donc le chiffre 3. Ainsi, on peut passer d'un élément à l'autre en lisant simplement leur valeur et en l'additionnant à la valeur relativement au début de la liste. Cela paraît incompréhensible à première vue mais c'est exactement la méthode selon laquelle le système d'exploitation gère les cluster de la disquette.

Voyons la donnée simplifiée de notre FAT (3,4,5) et construisons deux éléments quelconques avant cette liste (x,x,3,4,5). Le numéro de cluster du fichier "DESK2.ACC" est 2. Lorsqu'on lit l'adresse Début+0, on obtient x, de même que l'adresse Début+1. Mais si nous lisons l'adresse Début+2, nous obtenons 3, Début+3 donne 4. Pour DESK2.ACC, cela signifie que le cluster suivant est le numéro 3 puis le numéro 4 et ainsi de suite.

Nous avons déjà expliqué le calcul de numéro de secteurs à partir des numéros de cluster. Il faut maintenant une méthode pour trouver la fin de fichier. Voici la solution du système d'exploitation avec la FAT : Si une adresse obtenue après lecture a une valeur déterminée, c'est la fin de fichier.

Afin que cela soit un peu plus compliqué, on a affaire au format INTEL avec une représentation sur 12 bits. Une donnée de la FAT tient sur 12 bits, soient 3 nibbles de 4 bits. Les 12 bits suffisent complètement car une disquette n'a jamais plus de $2^{12} = 4096$ clusters.

Pour expliquer le format et retrouver les numéros de cluster, jetez un oeil sur la FAT. Représentons nous pour l'instant 16 bits par donnée de la FAT, en format Intel. Dans ces 16 bits, le most significant nibble (les quatre premiers bits) du High Byte est inutile et nous le représenterons par X. On met dans ce nibble libre le least significant nibble du low byte (4 derniers bits).

Le most significant nibble du low Byte et le least significant nibble du High Byte sont mis dans le low byte mais interchangent leur place. On a donc maintenant une valeur sur 12 bits à partir d'une valeur sur 16 bits en économisant un nibble. Etant donné qu'on utilise à chaque fois un octet de moins, le système est périodique et après 12 bits, le décalage de nibble est fait et on recommence au début.

Pour transformer la donnée sur 12 bits en format INTEL en format MOTOROLA, le numéro de la donnée est donc utile. On commence avec le chiffre 0 et on compte une donnée tous les 3 nibbles. Comme nous l'avons expliqué dans l'exemple simplifié (x,x), les deux premières données (numéro 0 et 1) de la FAT (F7 FF FF) n'ont pas de signification et la première donnée valable est la suivante, soit le numéro 2. Si le numéro d'accès est pair comme dans ce cas (2), les deux nibbles de poids faible (Format MOTOROLA) sont dans le premier octet (03) et le nibble de poids fort (Format MOTOROLA) du cluster suivant est dans le deuxième nibble de l'octet suivant (0). Ainsi, le cluster suivant est le cluster 003 (Format MOTOROLA).

Si on veut trouver le cluster suivant, on recompte par groupe de 3 à partir du début et on trouve pour la quatrième donnée (numéro 3) les octets 40 00. Le début du groupe est le 0 du premier octet. Si le numéro est impair, les deux nibbles du deuxième octet représentent les nibbles de poids fort et le nibble de poids faible est le premier nibble du premier octet : dans l'exemple concret, le numéro de cluster est 004.

7.3.2 Dossiers et sous-catalogues sur la disquette

Le système de fichier du TOS ATARI est organisé de manière hiérarchique et récursive. Cela signifie que l'on peut revenir de n'importe quel sous-directory au directory principal (racine). Pour mieux comprendre cela, créez deux dossiers avec l'option "nouveau dossier" : "DOSSIER1.SUB" et "DOSSIER2.SUB". Mettez en route l'éditeur de disquettes et lisez le secteur de directory (face 0 piste 1 secteur 3).

L'octet numéro 11 de la donnée "DOSSIER1.SUB" (\$10) dont nous avons déjà parlé vous indique que cette donnée est un sous-catalogue.

Le cluster de départ de ce sous-catalogue est le cluster numéro 9 (octets 26 et 27) qui est sur la piste 3 secteur 6 de la disquette simple face. Lisez ce secteur avec le menu secteur.

Chaque subdirectory a droit à des secteurs de directory propres, offerts gracieusement par le système d'exploitation. Dans ces secteurs de sous-catalogue, les deux premières données sont toujours occupées, même s'il n'y a pas de fichier dans le dossier. La première donnée commence par un point (\$2E) suivi d'espaces. Dans l'octet d'attribut, on reconnaît le sous-directory à \$10 et le cluster de départ est le propre cluster du subdirectory, soit dans ce cas le numéro 9. La deuxième donnée d'un subdirectory qui est précédée de deux points (\$2E) contient la donnée du cluster de début (octets 26 et 27) du sous-catalogue inférieur. Dans notre cas, vous trouvez deux octets nuls (00 00) ce qui signifie que le subdirectory inférieur est la racine. En effet, "DOSSIER1.SUB" a été créé à partir du directory principal.

Quittez maintenant le moniteur de disquettes et affichez le contenu du subdirectory à partir du desktop. Maintenant que vous voyez seulement un sous-directory vide, créez un nouveau dossier ayant pour nom "DOSSUP1.SUB" et remettez le moniteur en route. Rien n'a changé dans la racine (piste 1 secteur 3). Allez donc directement au secteur de départ du subdirectory qui se trouve en piste 3 secteur 6, avec le menu secteur.

Le dossier que vous venez de créer dans ce sous-catalogue est inscrit comme vous le voyez dans le secteur de catalogue du dossier. Et le startcluster porte pour numéro 11. Le cluster numéro 11 sur une disquette simple face commence à la piste 4 secteur 1. Lisez ce secteur.

La première donnée du sous-directory "DOSSUP1.SUB" indique récursivement le secteur lui-même alors que la deuxième donnée qui indique le cluster du dossier inférieur, soit "DOSSIER1.SUB" qui a pour valeur 9.

7.3.3 Formatage dans un format non-standard (non ATARI).

Il existe deux types de formats non standards : d'abord l'utilisation de plus de pistes et de secteurs qu'il n'en existe dans le format classique avec le desktop (80 pistes 0-79 et 9 secteurs par piste 1-9), ensuite un nombre différent d'octets par secteur et plus ou moins d'octets de synchronisation entre les champs d'adresses et de données. Si vous voulez par exemple formater la piste 81 avec 10 secteurs, sélectionnez le menu option et augmentez la variable MAXTRACK à l'aide des touches du curseur. De la même manière, augmentez la variable MAXSECTOR. Quittez le menu options avec BACK et revenez au menu Format. Mettez le numéro de piste à 81 et la variable SECT/TRAC à 10. Enfin, il vous suffit de sélectionner FORMAT et de répondre à la question de confirmation par Y : la piste 81 est formatée avec 10 secteurs.

Si vous voulez quitter complètement le format ATARI et déterminer vous-même les gaps entre les secteurs ainsi que la taille des secteurs, il faut s'écarter de la programmation avec le TOS et programmer directement le contrôleur de disquettes. Naturellement, nous serons limités par le contrôleur et nous ne pourrons pas formater 630 octets/secteur par exemple car le contrôleur n'autorise que quatre possibilités (128, 256, 512, 1024). Si vous voulez formater la piste 79 avec 4 secteurs de 1024 octets et avoir un gap de pré-piste de 32 fois \$4E au lieu de 60 fois \$4E, sélectionnez l'option GAPS du menu FORMAT. Vous avez un nouveau menu qui vous permet de diminuer la valeur de GAP1 et d'augmenter la valeur de BYT/SEC à 1024. Revenez au menu Format avec BACK et décrémente l'option SEC/TRAC à 4. Sélectionnez enfin l'option XFORMAT.

Répondez à la question de confirmation par Y et la piste 79 est formatée. Attendez une seconde car le moteur du lecteur risque de continuer à tourner. Naturellement, j'aurais pu introduire une boucle de temporisation mais l'option du moteur en rotation était très utile dans la phase de test car les différents lecteurs ont des Spin-Up différents. Cela signifie que le lecteur du 1040 STF atteint sa vitesse de rotation optimale plus rapidement que l'ancien SF 354. Ainsi, les parties de programmes qui font accès directement au contrôleur fonctionnent différemment avec les différents lecteurs (une fois ça marche, une fois ça ne marche pas...).

Cette erreur nous est apparue dans la phase de test mais elle est maintenant corrigée. Mais si vous avez branché un lecteur complètement extraterrestre (ancien 5"1/4) et que les options READ with SYNC ou XFORMAT ne fonctionnent pas comme prévu, attendez un petit moment que le moteur ait atteint sa vitesse de croisière en quittant l'option READ with SYNC puis rappelez le menu.

Track with Syncs

Pour tester pratiquement l'option TRACK/SYNCS prenez une disquette formatée normalement puis sélectionnez cette option. Chargez ensuite la piste 79 en sélectionnant READ with SYNCS et affichez les premiers octets.

Pour le premier gap, vous devez avoir au début de la piste environ 60 fois \$4E, mais il peut y avoir d'autres valeurs (par exemple : \$E4, \$9C, \$27). Ce phénomène est dû au contrôleur car le processus de lecture n'est pas synchronisé au début de la piste, si bien que le premier bit de \$4E est ignoré et que le contrôleur considère les 8 bits suivants comme étant le premier octet.

Par exemple :

4 E 4 E 4 ...-> 9 C 9 C 9

0100 1110 0100 1110 0100 1...-> 1001 1100 1001 1100 1001

Après le premier gap qui fait environ 60 octets se trouvent 12 octets nuls (\$00) dont le premier et le dernier octet peuvent se trouver coupés et indiquer d'autres valeurs. Ensuite, vous trouvez les premiers octets de synchronisation, soit 3 fois \$A1, sachant que le premier \$A1 peut ne pas être lu correctement. Les 3 octets \$A1 activent le 'créateur de checksum hardware'. Cela signifie que le contrôleur de disquettes détermine la checksum à partir des octets suivants. L'octet suivant est \$FE qui définit les 6 octets suivants comme étant le champ d'adresse.

Le contenu de ces octets est (\$4F, \$00, \$01, \$02, \$70, \$1D), \$4F : piste 79, \$00 : sur la face 0, \$01 : secteur 1, \$02 : le secteur suivant est constitué de 512 octets, \$70 \$1D : checksum du champ d'adresse. Ensuite, vous avez de nouveau 22 fois \$4E et 12 fois \$00 suivis de 3 fois \$A1. L'octet suivant (\$FB) informe sur les 512 octets suivants (qui sont tous \$E5 sur une disquette qui vient d'être formatée) suivi de deux octets de checksum (\$C4 \$02). A la fin du secteur de données, il y a 40 fois \$4E, puis de nouveau le secteur de synchronisation (\$00, \$4E, \$A1, etc...) pour les 8 champs d'adresse et de données de cette piste.

Quittez le menu TRACK/SYNC et sélectionnez l'option GAP du menu FORMAT, puis incrémentez la valeur de GAP2 avec les touches du curseur pour la faire passer de 12 à 15, sélectionnez ensuite XFORMAT et répondez à la question par Y. Si vous regardez la piste numéro 79 formatée de cette manière, avec l'option TRACK/SYNC, vous verrez cette fois 15 fois \$00 entre chaque champ d'adresse de données.

7.4 L'ASSEMBLAGE AVEC DIFFERENTS ASSEMBLEURS

Digiteal Research :

1. Assembler avec : `as68.ttp -l -u edit.s`
2. Linker avec : `link68.ttp [u] edit 68k=edit.o`
3. Rendre exécutable par : `relmod.ttp edit.68k edit.tos`

Le fichier `edit.tos` sera exécuté en faisant un double click.

Assembleur Gst :

1. Il faut mettre "opt abs" au début du fichier "edit.s" et toutes les directives de mémoire comme "text,bss,bata" doivent être précédées par "section". "text" devient alors "section text". Sauvegardez ensuite le programme source sous le nom "edit.gst".
2. Créer un fichier de linkage ayant pour nom "asl.lnk" qui contiendra seulement une ligne "input*".
3. Assembler "edit.gst" avec `asm.prg edit.gst - errors.`

4. Linker avec link.prg edit - with asl.lnk - prog edit.tos.

Le fichier edit.tos est exécutable.

Assembleur Metacomco :

Le programme source "edit.s" peut être utilisé sans modification.

1. Création d'un fichier linkable "asl.lnk" qui contient une seule ligne "input*".
2. Assemblage avec `assem.ttp edit.s to edit.bin`.
3. Linkage avec `link.ttp edit.bin - with asl`.

Le programme edit.prg est exécutable.

8. Programmes utilitaires en BASIC

Le BASIC de l'ATARI ST dispose de pas mal de fonctions et il est assez rapide. Mais il existe un certain nombre de problèmes qui ne peuvent pas être résolus en BASIC (ou très difficilement). De plus, il y a des problèmes de temps lorsqu'il faut traiter de grandes masses de données.

Il faut souvent se servir d'un petit programme en langage machine qui est relié au programme en BASIC. On peut très facilement mettre ce programme dans un tableau d'entiers (p. ex. A%(n)) et l'y appeler. Mais il faut faire attention à certaines choses dont nous allons parler.

8.1 MISE EN ROUTE ET PARAMETRAGE

Dans presque tous les BASIC, il existe deux instructions qui autorisent la liaison entre assembleur et BASIC. Il s'agit des commandes USR et CALL.

Malheureusement, la commande USR n'existe pas dans le BASIC ATARI. Il existait même une ancienne version qui affichait "Function not yet implemented". Intéressons-nous donc à l'autre commande.

CALL appelle, comme l'indique le nom, un programme en langage machine. Il faut donner pour cela les paramètres suivants :

CALL A (P1,P2,P3)

A indique l'adresse du programme en langage machine, P1, P2 et P3 sont les paramètres qui sont donnés au programme. Le nombre de paramètres est libre, soit entre 0 et un très grand nombre.

Si on indique une valeur directement (par exemple 1), elle est saisie telle quelle. Si on indique une variable, c'est son contenu qui est utilisé. Il faut faire attention à ce que les valeurs soient mises dans un mot long. Ainsi, on n'a droit qu'à des valeurs entre + et - 2 milliards. On ne peut pas utiliser des chiffres à virgule.

Si on indique une chaîne de caractères (p. ex. AS), l'adresse de la chaîne est mise dans la mémoire. On évite ainsi d'utiliser la fonction VARPTR. Par contre, on a besoin de cette fonction pour calculer l'adresse de début du programme.

Le programme en langage machine retrouve les paramètres suivants sur la pile :

D'abord l'adresse de retour à laquelle revient le processeur lorsqu'il rencontre un RTS. Cette valeur est la plupart du temps sans intérêt et ne doit être modifiée sous aucun prétexte !

Ensuite, il y a un mot qui contient le nombre de paramètres indiqués. On obtient ce mot avec l'instruction MOVE.W 4(SP),D0 et le nombre est mis dans le registre D0.

Le mot suivant contient un pointeur sur la liste de paramètres. Ce pointeur peut être chargé dans le registre d'adresse A0 par l'instruction MOVE.L 6(SP),A0.

La liste des paramètres contient les mots longs des paramètres dans l'ordre indiqué sous BASIC. Ces valeurs peuvent être traitées en langage machine.

Lors des expérimentations avec cette fonction, nous avons obtenu un effet surprenant. Certains programmes fonctionnent sans difficulté. D'autres, parfois plus simples, plantaient l'ordinateur. Après de nombreuses tasses de café, des vidages fréquents de cendriers et une fois que je n'avais plus de cheveux à arracher, je suis arrivé à la solution suivante : le registre d'adresse A6 ne doit pas être utilisé ni modifié dans un programme en langage machine ! Après avoir transformé tous les registres A6 en registres A4, cela fonctionna à merveille !...

8.2 QUELQUES EXEMPLES DE PROGRAMMES

Dans le chapitre qui suit, nous allons vous présenter quelques routines pour des programmes BASIC. Nous y présenterons quelques solutions à des problèmes que vous pourrez utiliser pour vos propres programmes. De plus, les programmes vous permettront de comprendre comment s'utilisent et s'échangent différents paramètres.

Ainsi, vous pourrez écrire d'autres programmes en langage machine et améliorer sensiblement vos programmes BASIC.

Les exemples sont toujours donnés sous la forme de listings en assembleur ou en BASIC. Le programme en BASIC comprend également un chargeur qui génère le programme en langage machine. Vous pouvez également mettre les données du programme en langage machine dans un fichier sur disquette et le charger avec l'instruction BLOAD "nom_de_fichier",A. Mais cela nécessite un accès à la disquette ce qui est évité avec le chargeur BASIC.

8.2.1 Interface BASIC/TOS

Le système d'exploitation de l'ATARI ST offre de nombreuses fonctions auxquelles on ne peut pas accéder avec le BASIC.

On peut facilement résoudre ce problème avec un programme en langage machine. Un tel programme doit offrir la possibilité de saisir un nombre quelconque de paramètres à partir du programme BASIC et de le mettre sur la pile. Mais il faut faire attention à ce que les données soient sous forme de mots (16 bits). Il faudra donc diviser les mots longs en deux parties.

Le dernier paramètre qui est envoyé avec l'instruction CALL a une signification particulière. Etant donné que certaines fonctions de GEMDOS renvoient une valeur, cette valeur est rendue sur la dernière adresse. Dans l'exemple, nous montrons la fonction CONIN qui attend la pression d'une touche et renvoie sa valeur.

Mais voyons d'abord le programme en langage machine. On envoie quelques paramètres sur la pile au moyen d'une boucle et on les spécifie au système d'exploitation avec une instruction TRAP. La valeur de retour qui se trouve dans le registre D0 est écrite à l'adresse du dernier paramètre.

```
;; Interface BASIC-TOS 6/86 S.D. ;;
;; Appel avec CALL ADR (Liste_de_Parametres,X) ;;
;; X étant l'adresse de la valeur de retour D0 ;;

run:
    move    4(sp),d0    ;Nombre de paramètres
    move.l  6(sp),a5    ;Pointeur sur Parameterbloc
    subq    #2,d0       ;Corriger nombre de parametres
    move.l  sp,a4       ;Sauvegarder ancien Stackpointer

loop:
    move.l  (a5)+,d1    ;Récupérer paramètres
    move    d1,-(sp)    ;et continuer avec sur la
    dbra    d0,loop     ;pile

    trap    #1         ;Appel TOS
    move.l  (a5),a5     ;Créer adresse de retour
    move.l  d0,(a5)     ;Renvoyer D0
    move.l  a4,sp       ;Restaurer la pile
    rts              ;Fin !
```

Comme vous le voyez, le programme est très simple. Nous allons donc en venir au programme en BASIC. Ce programme crée le programme en langage machine et fait un test. La fonction créée ainsi est la fonction CONIN qui attend la pression d'une touche et renvoie la valeur de la touche sur D0. Le mot de poids faible de D0 est la valeur de la touche et le mot de poids fort contient le code interne du clavier. On obtient les deux valeurs dans le mot long D0 qui est décrit dans la variable BS.

```

10  '## Interface BASIC-TOS S.D. ##
20  defdbl s
30  dim a%(200)
40  a=varptr(a%(0))
50  s=0
60  for i=0 to 14 :read a%(i)
70  s=s+a%(i) :next i
80  if s<> 174830 then ?"Erreur !":stop
100 b$=space$(10)
110 b=varptr(b$)      : 'Adresse de retour
120 call a (1,b)      : 'Appel de la routine
130 ?peek(b),b$      : 'Affichage du résultat
1000 '## Données pour BASTOS ##
1010 data &H302F,4,&H2A6F,6,&H5540,&H284F,&H221D,&H3F01
1020 data &H51C8,&HFFFA,&H4E41,&H2A55,&H2A80,&H2E4C,&H4E75

```

8.2.2 Lecture du catalogue

L'ATARI ST a un manque grave : la possibilité de lire un directory. On peut afficher le directory avec la commande DIR, mais à quoi cela sert-il ?

Mais si on veut utiliser dans un programme les informations contenues dans le catalogue, il faut encore une fois réaliser un programme en langage machine. Vous trouverez un tel programme dans ce chapitre. Outre l'accès normal aux noms de fichiers, il vous donne les informations qu'on trouve dans un catalogue (voir chapitre 6.3). De plus, il vous indique la capacité de la disquette et le nombre d'octets disponibles.

```
;; Directory pour BASIC S.D. ;;
```

```
run:
```

```
bra sfirst ;Branchement 1
```

```
snext:
```

```
;Branchement 2
```

```
move #$4f,-(sp)
```

```
trap #1 ;Fonction snext
```

```
addq.l #2,sp
```

```
tst d0
```

```
bne yapas ;Pas d'autres données
```

```
rts
```

```
sfirst:
```

```
cmp #3,4(sp)
```

```
bne quit ;pas de 3 paramètre !
```

```
move.l 6(sp),a5 ;Pointeur sur Parameterbloc
```

```
lea buffer(pc),a4
```

```
move.l 8(a5),(a4) ;Pufferadresse retten
```

```
move.l 8(a5),-(sp) ;bufferadress
```

```
move #$1a,-(sp)
```

```
trap #1 ;SETDTA-Function
```

```
addq.l #6,sp
```

```
move 6(a5),-(sp) ;Attribut
```

```
move.l (a5),-(sp) ;Filename
```

```
move #$4e,-(sp)
```

```
trap #1 ;SFIRST-Function
```

```
addq.l #8,sp
```

```
tst d0
```

```
bne yapas
```

```
quit:
```

```
rts ;=> BASIC
```

```
yapas:
```

```
move.l buffer(pc),a4
```

```
clr    -(sp)      ;Lecteur courant
move.l a4,-(sp)   ;buffer-Adress
move   #$36,-(sp)
trap   #1         ;GET-FREE-SPACE-Function
addq.l #8,sp

move.l #'Frei',30(a4) ;Pas de Filename !
rts    ;=> BASIC
```

```
buffer: dc.l 0
```

La première chose qui saute aux yeux avec ce programme sont les deux points d'entrée. Cela tient au fait que le programme est constitué de deux programmes indépendants.

La première partie est la fonction `SEARCH_FIRST`. Cette fonction du GEMDOS reçoit quelques paramètres comme le nom de recherche, l'attribut de fichier et l'adresse du tampon. La fonction de l'autre partie de programme `SEARCH_NEXT` n'utilise pas de paramètres car elle emploie l'installation de la fonction `SEARCH_FIRST`.

Pour le programme en BASIC qui appelle, cela signifie qu'il faut d'abord appeler le programme du début puis continuer avec l'adresse+4.

D'autre part, on n'envoie les paramètres qu'une seule fois.

Si la fonction `SEARCH_FIRST` ou `SEARCH_NEXT` ne trouve aucun autre fichier qui correspond au critère de recherche, une autre fonction est appelée. Cette fonction renvoie un bloc de paramètres qui contient les informations sur la taille de la disquette et le nombre d'octets restants. Le programme BASIC récupère ces informations ainsi que les données du directory dans le bloc de paramètres. Il peut donc reconnaître par exemple le nom de fichier.

Voici le premier appel du programme :

CALL A (F\$,A,PS)

Signification des paramètres :

- A Adresse de départ du programme en langage machine.
- F\$ Chaîne de caractères qui contient le nom de chemin des fichiers à rechercher (p. ex. B:*.BAS). La chaîne doit être terminée par un octet nul.
- A Attribut des fichiers à rechercher. Un nul recherche tous les programmes normaux.
- PS Chaîne de caractères tampon pour les données renvoyées par le programme en langage machine. Pour connaître la structure du tampon, voyez le chapitre 6.3.

Voici un programme en BASIC qui génère le programme en langage machine et qui donne un exemple d'application. Il affiche tous les fichiers de la disquette ainsi que leur longueur. A la fin on obtient la capacité de la disquette en octets.


```

10  '## Lecture du catalogue S.D. ##
20  defdbl s
30  dim a%(200)
40  d=varptr(a%(0))      : '1. branchement pour SEARCH_FIRST
50  s=0
60  for i=0 to 52 :read a%(i)
70  s=s+a%(i) :next i
80  if s<> 610895 then ?"Erreur !":stop
130 d1=d+4                : '2. branchement pour SEARCH_NEXT
150 f$="\x.\t"+chr$(0)   : 'chaîne de recherche
160 p$=space$(50)        : 'Effacer buffer
170 call d (f$,0,p$)      : 'SEARCH_FIRST
180 goto loop
190 loop:
200 call d1                : 'SEARCH_NEXT
210 loop:
220 if mid$(p$,31,4) = "Frei" then 260 : 'Fin
230 i=27: gosub calc       : 'Calculer longueur
240 ?mid$(p$,31,14),1     : 'Afficher nom et longueur
250 goto loop
260 i=1: gosub calc        : 'calculer nombre de Ko libres
270 ?"## Octets libres : ";i*1024 : 'Et afficher en octets
280 end                    : 'Fin du programme
290 calc:
300 l=asc(mid$(p$,i+3,1))+&H100*asc(mid$(p$,i+2,1))
310 l=l+&H10000*asc(mid$(p$,i+1,1))
320 return
1000 '## Données pour BASDIR ##
1010 data &H6000,&H12,&H3F3C,&H4F,&H4E41,&H548F,&H4A40
1020 data &H6600,&H3C,&H4E75,&HC6F,3,4,&H6600,&H2E,&H2A6F
1030 data 6,&H49FA,&H42,&H28AD,8,&H2F2D,8,&H3F3C
1040 data &H1A,&H4E41,&H5C8F,&H3F2D,6,&H2F15,&H3F3C,&H4E
1050 data &H4E41,&H508F,&H4A40,&H6600,4,&H4E75,&H287A,&H18
1060 data &H4267,&H2F0C,&H3F3C,&H36,&H4E41,&H50BF,&H297C
1070 data &H4672,&H6569,&H1E,&H4E75,0,0
    
```

8.2.3 Lecture/écriture de secteurs

Comme nous l'avons déjà dit, les données sur une disquette sont divisées en secteurs. On n'a normalement pas accès directement à ces secteurs. Le système d'exploitation charge seulement les secteurs sur lesquels se trouve le fichier choisi.

Si on veut accéder à un secteur en particulier, il faut faire un programme en langage machine qui lit ou écrit un secteur. Nous allons vous présenter un programme de la sorte.

On donne 3 paramètres au programme : le numéro de secteur logique, une commande de lecture/écriture et l'adresse du tampon utilisé.

Le numéro de secteur logique peut varier entre 0 et la valeur maximum. Ce maximum dépend du format de la disquette.

La commande de lecture/écriture peut prendre les valeurs suivantes :

- 0 - lire secteur
- 1 - écrire secteur
- 2 - lire secteur en ignorant un changement de disquette
- 3 - écrire secteur en ignorant un changement de disquette

Si on utilise 0 ou 1 avec la commande, le programme accède à la disquette se trouvant dans le lecteur. Si on change de disquette, l'accès n'est plus possible.

Voici le programme en langage machine :

```

;##      Lecture de secteur   S.D.      ##
;## CALL A (Sector,rw (2=read,1=write),buffer) ##

run:
    cmp     #3,4(sp)      ;3 Parametre ?
    bne     quit          ;non => Quitter

    move.l  6(sp),a5      ;Pointeur sur Parametre

    clr     -(sp)         ;Lecteur A
    move     2(a5),-(sp)
    move     #1,-(sp)      ;1 Secteur
    move.l   8(a5),-(sp)   ;buffer
    move     6(a5),-(sp)   ;Read/Write
    move     #4,-(sp)      ;RWABS-Fonction
    trap     #13           ;Appel BIOS
    add.l    #14,sp

quit:
    rts                ;=> BASIC

```

Le programme n'accède qu'au lecteur A. Si vous voulez pouvoir le modifier, vous pouvez changer le programme pour passer 4 paramètres.

Voici le programme BASIC correspondant qui crée un programme en langage machine et fait une petite démonstration :

```

10  '## Recherche dans un champ d'INTEGER S.D. ##
30  dim a%(100),f%(300)
40  a=varptr(a%(0))
50  defdbl s
60  s=0
70  for i=0 to 22: read a%(i) : 'Lire programme
80  s=s+a%(i) : next i
90  if s(<) 165974 then ?"Erreur !" : stop
100 f=varptr(f%(0))
200 input "Sector, rw : ";s%,r%
210 call a (s%,r%,f)          : 'Appel du programme
220 for i=0 to 255
230 if (i mod 16)=0 then ?
240 ?aki$(f%(i));             : 'Affichage ASCII du secteur
250 next i : ?

1000 '## Données pour programme en langage machine ##
1010 data &HC6F,3,4,&H6600,&H24,&H2A6F,6,&H4267
1020 data &H3F2D,2,&H3F3C,1,&H2F2D,8,&H3F2D,6
1030 data &H3F3C,4,&H4E4D,&H0FFC,0,&H4E,&H4E75

```

8.2.4 Différents types de formatage de disquettes

Comme nous l'avons déjà vu dans le chapitre 6, les disquettes 3 pouces 1/2 peuvent être formatées de différentes manières. Le nombre de faces, de pistes et de secteurs est variable.

Pour formater une disquette avec un programme BASIC, il faut utiliser une petite routine en langage machine car il n'existe pas de commande convenante. De plus, le DESKTOP n'autorise que deux types de formatage. On aura donc intérêt à utiliser un programme appelé à partir du BASIC et utilisant quelques paramètres. Ces paramètres représentent le format dans lequel la disquette est initialisée.

Le programme rappelle en gros celui que nous avons étudié dans le chapitre 6. Mais si nous l'observons de plus près, nous constatons quelques différences.

D'une part, le menu a disparu et avec lui, les paramètres correspondants. Tous les paramètres utiles sont envoyés à partir du BASIC.

D'autre part, les adressages des variables sont effectués d'une manière différente. Cela est compliqué car le programme est inscrit dans un secteur mémoire par le chargeur BASIC. Et ce secteur est inconnu du programme lui-même. Tous les adressages doivent donc être en relatif. Il n'y a aucune adresse absolue.

Le programme en langage machine est appelé avec une commande CALL ayant la structure suivante :

CALL A (F,T,SPT,LC)

Signification des variables :

- A C'est l'adresse mémoire à laquelle le programme en langage machine est inscrit. Dans l'exemple donné, il s'agit de l'adresse du champ d'entiers A% qui est obtenue par la fonction VARPTR.
- F Nombre de faces qui doivent être formatées. Il faut indiquer le nombre - 1 (simple face = 0, double face = 1).

- T Nombre de pistes (Tracks). Normalement, une disquette contient 80 pistes mais physiquement, on peut en mettre 82 (même 83 parfois).
- SPT Sectors Per Tracks. On inscrit normalement 9 à cet endroit. Mais on peut mettre une valeur entre 1 et 10 inclus.
- LC C'est le lecteur. On définit avec cette variable le lecteur dans lequel se trouve la disquette à formater. 0 indique le lecteur A et 1, le lecteur B. N'essayez pas de mettre un 3 pour formater éventuellement votre RAM DISK. Cela signifie que vous adressez à la fois le lecteur A et le lecteur B...

Voici le programme en langage machine qui reçoit les paramètres à partir du BASIC, qui les adapte et qui formate la disquette.

;; Sous-programme BASIC : Routines de formattage S.D. ;;

run:

```

move    4(sp),d0
cmp     #4,d0           ;4 Parametre ?
bne     quit            ;non => Fin
move.l  6(sp),a5        ;Pointeur sur Parameterbloc

```

```

lea     faces(pc),a4
move.l  (a5)+,d1
move    d1,(a4)         ;Faces
move.l  (a5)+,d1
move    d1,2(a4)        ;Tracks
move.l  (a5)+,d1
move    d1,4(a4)        ;Sectors pro Track
move.l  (a5)+,d1
move    d1,6(a4)        ;Numero de lecteur

```

```

move    tracks(pc),B(a4)
subq    #1,B(a4)

```

floop:

```

move    faces(pc),10(a4) ;Déterminer les faces

```

floop1:

```

bsr     fmttr           ;format Track
bne     quit
sub     #1,10(a4)       ;Face -1
bpl     floop1
sub     #1,8(a4)
bpl     floop           ;Piste suivante

```

setboot:

```

clr     -(sp)           ;Execute-Flag
moveq   #2,d0
or      faces(pc),d0
move    d0,-(sp)        ;Disktyp, face
move.l  #$1000000,-(sp) ;Créer numéro de série
pea     12(a4)          ;buffer-Adress
move    #$12,-(sp)
trap    #14             ;Créer le Boot-Sector

```

```

    add.l    #14,sp

    lea      12(a4),a0
    clr.l    d0
    cmp      #9,4(a4)      ;9 Secteurs par piste ?
    beq      sok           ;oui
    move.b   #10,24(a0,d0) ;Mettre 10 SPT
    move     tracks(pc),d1
    tst      (a4)           ;1 face ?
    beq      sd11           ;oui
    lsl      #1,d1          ;Sinon doubler
sd11:
    bsr      addsec         ;SEC + nombre de pistes

sok:
    cmp      #80,2(a4)      ;80 Tracks ?
    beq      trok
    move     #18,d1
    tst      (a4)           ;1 face ?
    beq      sd12           ;oui
    lsl      #1,d1          ;Sinon doubler
sd12:
    bsr      addsec         ;SEC + 2*9 ou 4*9

trok:
    move     #1,-(sp)       ;1 Secteur
    clr.l    -(sp)          ;Face 0, Track 0
    move     #1,-(sp)       ;Secteur 1
    move     drive(pc),-(sp) ;Lecteur
    clr.l    -(sp)
    pea      12(a4)         ;Tampon
    move     #9,-(sp)
    trap     #14             ;flopwr
    add.l    #20,sp

quit: rts                  ;Retour au BASIC

addsec:
    move.b   20(a0,d0),d2   ;SEC = SEC + D1
    lsl      #8,d2          ;HI
    
```

```
move.b 19(a0,d0),d2    ;LO
add     d1,d2
move.b  d2,19(a0,d0)    ;set LO
lsr     #8,d2
move.b  d2,20(a0,d0)    ;set HI
rts

fattr:                                ;Formattage d'une piste
clr     -(sp)              ;Données vierges
move.l  #$87654321,-(sp) ;Nombre magique
move    #1,-(sp)          ;interleave
move    face(pc),-(sp)    ;Seite
move    tracksl(pc),-(sp) ;Track
move    secptr(pc),-(sp)  ;Sectors/Track
move    drive(pc),-(sp)   ;Lecteur
clr.l   -(sp)
pea     12(a4)
move    #10,-(sp)
trap    #14                ;flopfat
add.l   #26,sp
tst     d0                  ;Test si erreur
rts

faces:  dc.w 1
tracks: dc.w 80
secptr: dc.w 9
drive:  dc.w 0
tracksl: dc.w 80
face:   dc.w 0
tanpon: dc.b $200
```


Ce programme est totalement relogeable. Cela signifie qu'il fonctionne à n'importe quelle adresse. Nous avons déjà expliqué les différentes composantes du programme. Il est donc auto-explicatif.

Voici enfin un programme BASIC qui appelle la routine de formatage. Il contient un chargeur qui génère le programme à partir des lignes de DATA. Naturellement, on peut également le charger sur disquette.

```

10   '## Formattage d'une disquette ##
15   'fullw 2
17   defdbl s
20   dim a%(400)
30   a=varptr(a%(0))
35   s = 0
40   for i=0 to 144: read a%(i)
45   s =s +a%(i) : next i
46   if s <> 1033402 then ?"Erreur !":stop
50   print "Faces, Pistes, Sectors/Track, Lecteur "
60   input s,t,spt,lw
80   call a (s,t,spt,lw)
90   '## Données pour BFORMAT.obj ##
100  data &H302F,&H0004,&H0C40,&H0004,&H6600,&H00CC,&H2A6F,&H0006
110  data &H49FA,&H0110,&H221D,&H3881,&H221D,&H3941,&H0002,&H221D
120  data &H3941,&H0004,&H221D,&H3941,&H0006,&H397A,&H00FB,&H000B
130  data &H536C,&H000B,&H397A,&H00EC,&H000A,&H6100,&H00B4,&H6600
140  data &H0096,&H046C,&H0001,&H000A,&H6A00,&HFFF0,&H046C,&H0001
150  data &H000B,&H6A00,&HFFED,&H4267,&H7002,&H807A,&H00C6,&H3F00
160  data &H2F3C,&H0100,&H0000,&H486C,&H000C,&H3F3C,&H0012,&H4E4E
170  data &H0FFC,&H0000,&H000E,&H41EC,&H000C,&H4280,&H0C6C,&H0009
180  data &H0004,&H6700,&H001B,&H11BC,&H000A,&H081B,&H323A,&H0096
190  data &H4A54,&H6700,&H0004,&HE349,&H6100,&H003E,&H0C6C,&H0050
200  data &H0002,&H6700,&H0012,&H323C,&H0012,&H4A54,&H6700,&H0004
210  data &HE349,&H6100,&H0024,&H3F3C,&H0001,&H42A7,&H3F3C,&H0001
220  data &H3F3A,&H0066,&H42A7,&H486C,&H000C,&H3F3C,&H0009,&H4E4E
230  data &H0FFC,&H0000,&H0014,&H4E75,&H1430,&H0B14,&HE14A,&H1430
240  data &H0B13,&HD441,&H11B2,&H0B13,&HE04A,&H11B2,&H0B14,&H4E75
250  data &H4267,&H2F3C,&H8765,&H4321,&H3F3C,&H0001,&H3F3A,&H002E
260  data &H3F3A,&H002B,&H3F3A,&H0020,&H3F3A,&H001E,&H42A7,&H486C
270  data &H000C,&H3F3C,&H000A,&H4E4E,&H0FFC,&H0000,&H001A,&H4A40
280  data &H4E75

```

8.2.5 Recherche de données

Un programme en langage machine sert souvent de recherche dans une liste. Une telle recherche peut durer très longtemps avec une liste longue et un programme en BASIC ne serait d'aucune utilité. Imaginez à une gestion de base de données qui met une minute pour retrouver un numéro de téléphone...

Il est très facile d'écrire un programme en langage machine qui fasse ce travail. Il n'est constitué que de deux parties :

1. Saisie des paramètres à partir du BASIC
2. Boucle de recherche
3. Retour de résultat au BASIC

Voici un programme de la sorte :

```

;## Recherche dans un champ d'Integers S.D. ##
;## CALL A (Début_du_champ,Nombre,Mot_de_recherche) ##

run:
    cmp    #3,4(sp)      ;3 Parametre ?
    bne    quit          ;non => Exit

    move.l 6(sp),a5      ;Pointeur sur paramètre

loop:
    move.l (a5),a4       ;Pointeur sur champ de paramètres
    tst    (a4)+         ;Mettre sur fX(1)
    move.l 4(a5),d1       ;Nombre de données
    move.l 8(a5),d2       ;Mot de recherche
    moveq  #1,d3         ;Index=1

    cmp    (a4)+,d2      ;Comparaison
    beq    ok1           ;Trouvé
    addq   #1,d3         ;Index+1
    cmp    d3,d1         ;Fin ?
    bne    loop         ;non

```

```

        move    #-1,d3          ; Introuvable !
ok1:
        move.l  (a5),a5         ; Adresse de retour
        move    d3,(a5)         ; Retourner Index

quit:
        rts                    ; => BASIC

```

Ce petit programme effectue une recherche en une fraction de secondes et renvoie le numéro de la donnée recherchée dans le premier élément de la liste. Il faut faire attention à n'utiliser que des éléments allant de 1 à n. Si l'élément est introuvable, le programme renvoie -1.

Il y a de nombreuses applications pour le programme et le chargeur est très simple. Le principe de la routine et son utilisation sont enfantin.

```

10  '## Recherche dans champ d'Integers S.D. ##
30  dim a%(60),f%(1000)
40  a=varptr(a%(0))
50  defdbl s
60  s=0
70  for i=0 to 25 :read a%(i)
80  s=s+a%(i) :next i
90  if s<> 211865 then ?"Erreur !" :stop
130 f=varptr(f%(0))
140 for i%=1 to 8
150 read f%(i%)          : 'Lire valeurs exemples
170 next i%
200 input "mot de recherche : ";s%
210 call a (f,i%,s%)
220 if f%(0)=-1 then ?"## Introuvable ! ##" :goto 200
230 ?s%;" c'est la ";f%(0);". Donnée." :goto 200
1000 '## Données pour programme en langage machine ##
1010 data &HC6F,3,4,&H6600,&H2A,&H2A6F,6,&H2855
1020 data &H4A5C,&H222D,4,&H242D,8,&H7601,&HB45C,&H6700
1030 data &HE,&H5243,&HB243,&H6600,&HFFF4,&H363C,&HFFF
1040 data &H2a55,&H3A83,&H4E75

```

1100 data 6,2,99,345,7,3,0,4

8.2.6 Tri de données

Le tri d'une grande masse de données est quelque chose de très long. Un programme en langage machine qui doit trier 1000 données prend un temps tel qu'il peut être rédhibitoire. Par contre, un programme en langage machine fait cela très rapidement.

Nous vous proposons un programme pour cela. Il peut trier un champ d'INTEGER du BASIC de n'importe quelle taille. Les paramètres envoyés au programme sont l'adresse du début de vecteur et le nombre de données à trier. On peut donc trier une partie du tableau.

L'algorithme utilisé par ce programme est très simple. Ce n'est pas le plus rapide mais cela ne joue pas un grand rôle avec la grande rapidité du processeur 68000.

```
;; Tri du champ d'Integers S.D. ;;
;; CALL A (Début du champ, nombre) ;;

run:
    cmp     #2,4(sp)    ;2 Parametre ?
    bne     quit        ;non => Exit

    move.l  6(sp),a5     ;Pointeur sur le paramètre

lop1:
    move.l  (a5),a4      ;Pointeur sur le champ de paramètre
    move.l  4(a5),d1      ;Nombre de données
    clr     d3            ;Effacer Flag de changement

lop2:
    move     (a4),d0
    cmp     2(a4),d0      ;Comparaison
```

```

ble    ok1      ;OK
move   2(a4), (a4) ;Echanger
move   d0, 2(a4)
st     d3       ;Positionner Flag d'échange
ok1:
addq.l #2, a4    ;Valeur suivante
subq.l #1, d1
bne    lop2

tst    d3       ;fini ?
bne    lop1     ;non => suite

quit:
rts     ;=> BASIC

```

Voici un programme BASIC qui contient le programme en langage machine sous forme de DATAs et qui crée un champ à partir de ces data. On introduit des valeurs dans un autre tableau et on termine la liste par -1. Puis on appelle le programme en langage machine qui trie toutes les données du tableau. Puis les valeurs triées sont affichées.

Ce type d'application ne sert que d'exemple. Cela est intéressant pour une grande masse de données, pour laquelle la vitesse du BASIC ne convient pas.

```
10  '** Tri d'un champ d'INTEGERS S.D. **'
20  defdbl s
30  dim a%(200)
40  a=varptr(a%(0))
50  s=0
60  for i=0 to 28 :read a%(i)
70  s=s+a%(i) :next i
80  if s(> 280743 then ?"Erreur !":stop
100 dim f%(1000)           :'Préparer ce champ de données
110 defint i
120 a=varptr(a%(0))        :'Adresse du programme en langage machine
130 f=varptr(f%(1))        :'Adresse des données
140 for i=1 to 1000
150 input "Donnée : ";f%(i) :'Saisir les données
160 if f%(i)=-1 then 180    :'Fin ?
170 next i                  :'non, continuer
180 call a (f,i-2)         :'Trier
190 for j=1 to i
200 ?j;".: ";f%(j)        :'Et afficher de nouveau
210 next j
1000 '** Données pour BASSORT **'
1010 data &HC6F,2,4,&H6600,&H30,&H2A6F,6,&H2B55
1020 data &H222D,4,&H4243,&H3014,&HB06C,2,&H6F00,&HC
1030 data &H38AC,2,&H3940,2,&H50C3,&H548C,&H5381,&H6600
1040 data &HFFE6,&H4A43,&H6600,&HFFDB,&H4E75
```

8.2.7 Lecture de la date de manière formatée

Chaque base de donnée qui doit être modifiée quotidiennement doit contenir la date et éventuellement l'heure de chaque modification. Malheureusement, le BASIC ATARI ne possède aucune fonction pour cela. Il faut donc faire un programme en langage machine.

Le programme présenté lit la date et l'heure de l'ordinateur et les renvoie au programme BASIC. Ces informations sont formatées afin de pouvoir être directement traitées. L'appel se fait simplement avec CALL A (A\$) où la variable A\$ récupère le résultat. Voici le format employé :

HH.MM.SS JJ.MM.AAAA

.....	Année (p. ex. 1986)
.....	Mois (p.ex. 07 pour juillet)
.....	Jour
.....	Secondes (par pas de 2)
.....	Minutes
.....	Heures (0 à 23)

Le 3 juillet 1986 à 22 heures 13 et 34 secondes sera représenté par 22.13.34 03.07.1986.

Voici le programme en langage machine qui lit la date et la renvoie formatée, suivi du programme BASIC qui crée le programme en langage machine et qui fait une démonstration.

```

;## Lecture de l'heure, formatée S.D. ##
;# Appel avec CALL A (A$) renvoie dans A$ #
;# SS.MM.SS. TT.MM.JJJJ., Heure et date #

```

run:

```

cmp     #1,4(sp)    ;un paramètre ?
bne     quit        ;non => Quitter

move.l  6(sp),a5     ;Pointeur sur liste des paramètres
move.l  (a5),a5      ;Pointeur sur chaîne

```

go:

```
move    #$2c,-(sp)
trap    #1          ;get_time-Fonction de la BIOS
addq.l  #2,sp

and.l   #$ffff,d0   ;inverser mot de poids faible

move     d0,d1
lsr      #8,d1
lsr      #3,d1       ;Heures
bsr      set2b       ;Mettre l'heure

move.l   d0,d1
lsr      #5,d1
and      #%111111,d1
bsr      set2b       ;Mettre les minutes

move.l   d0,d1
lsl      #1,d1       ;Secondes *2
and      #$3f,d1     ;et inverser
bsr      set2b       ;Mettre les secondes

move.b   #'',(a5)+   ;Mettre blanc de séparation

move     #$2a,-(sp)
trap     #1          ;get_date-Fonction de la BIOS
addq.l   #2,sp

and.l    #$ffff,d0   ;inverser mot de poids faible

move.l   d0,d1
and      #%11111,d1   ;effacer jour
bsr      set2b       ;Mettre jour

move.l   d0,d1
lsr      #5,d1
and      #%11111,d1   ;effacer mois
bsr      set2b       ;Mettre mois
```



```

move.l d0,d1
lsr     #8,d1
lsr     #1,d1
and     #%1111111,d1 ;effacer année
add     #80,d1        ;corriger
move.b  #'1',(a5)+
move.b  #'9',(a5)+    ;Préparer 1900
bsr     set2b         ;Mettre l'année

quit:
rts                     ;fini !

set2b:                    ;Afficher D1 avec deux caractères
divu     #10,d1
add.l    #$300030,d1 ;Corriger valeur ASCII
move.b   d1,(a5)+    ;HI-Nibble
swap     d1
move.b   d1,(a5)+    ;LO-Nibble
move.b   #'',(a5)+   ;mettre point de séparation
rts

10      '## Réalisation GET_TIME S.D. ##
20      defdbl s
30      dim a%(200)
40      a=varptr(a%(0))
50      s=0
60      for i=0 to 76 :read a%(i)
70      s=s+a%(i) :next i
80      if s<> 399794 then ?"Erreur !":stop

90      t$=space$(20)
100     call a (t$)
110     ? "Aujourd'hui, nous sommes le "; right$(t$,10)
120     ? "L'heure : "; left$(t$,8)

990     '## Datas pour GETTIME ##
1000    data &H0C6F,1,4,&H6600,&H7A,&H2A6F,6,&H2A55
1010    data &H3F3C,&H2C,&H4E41,&H54BF,&H280,0,&HFFFF,&H3200
    
```

```
1020 data &HE049,&HE649,&H6100,&H5E,&H2200,&HEA49,&H241,&H3F
1030 data &H6100,&H52,&H2200,&HE349,&H241,&H3F,&H6100,&H46
1040 data &H1AFC,&H20,&H3F3C,&H2A,&H4E41,&H54BF,&H2B0,0
1050 data &HFFFF,&H2200,&H241,&H1F,&H6100,&H2A,&H2200,&HEA49
1060 data &H0241,&H1F,&H6100,&H1E,&H2200,&HE049,&HE249,&H241
1070 data &H007F,&H641,&H50,&H1AFC,&H31,&H1AFC,&H39,&H6100
1080 data 4,&H4E75,&HB2FC,&HA,&H681,&H30,&H30,&H1AC1
1090 data &H4841,&H1AC1,&H1AFC,&H2E,&H4E75
```

8.3 LA PROGRAMMATION DU FDC EN BASIC

Il existe un certain nombre de moniteurs de disquettes pour l'ATARI ST. Mais malheureusement, ceux que nous connaissons n'utilisent que les routines offertes par le système d'exploitation. Cela suffit pour la plupart des applications mais si vous voulez savoir précisément se qui se cache sur une disquette, il faudra utiliser certaines fonctions qui ne peuvent être atteintes que par la programmation directe du contrôleur de disquettes.

Nous comprenons dans ces fonctions par exemple la lecture du champ d'ID d'une piste, la lecture d'une piste complète ou le formatage quelconque d'une piste.

Notre conseil : écrivez simplement un moniteur de disquettes qui offre ces possibilités. En BASIC ? Oui, pourquoi pas ? Si les fonctions du contrôleur sont accessibles, alors cela est possible aussi en BASIC. Mais on ne peut pas y avoir accès au travers du système d'exploitation. Le GEMDOS, le BIOS et le XBIOS ne sont d'aucune utilité dans ce cas.

Pour vous aider, nous avons écrit une routine qui réunit toutes les fonctions du FDC et qui les met à disposition du BASIC.

L'époque à laquelle on ne pouvait pas lire des secteurs parce qu'ils étaient recouverts de champs d'ID "raffinés" appartient au passé. Si vous obtenez par exemple un message du type "Le lecteur A ne répond pas...", vous pourrez ouvrir les entrailles de la disquette et peut-être voir ce qui ne va pas.

Bien entendu, il reste un travail à effectuer : il faut interpréter les résultats que renvoient les commandes du FDC. Vous trouverez toutes les informations utiles dans le chapitre sur le contrôleur de disquettes WD 1772. L'analyse de disquette ne sera donc plus un problème pour vous.

Mais il y a d'autres avantages qui assurent la flexibilité de notre interface BASIC/FDC. Il est toujours possible de modifier les mots de commande. Ainsi, il y a un bit d'option qui influe sur l'exécution des instructions. Ainsi, on peut lire ou écrire tous les secteurs d'une piste avec une seule commande du FDC. Il est possible également de ne formater qu'une partie de la piste. Cela peut être très pratique pour réaliser une copie de protection.

Mais chaque chose en son temps : voyons d'abord l'interface FDC. Nous développerons les exemples d'application ultérieurement.

8.3.1 Le programme d'interfaçage BASIC/FDC

Commençons avec le programme en langage machine qui représente notre interface avec le FDC.

Pour l'obtenir, il y a trois possibilités :

1. Vous saisissez le listing en assembleur et vous l'assemblez (la version donnée ici a été réalisée avec l'assembleur SEKA).
2. Vous saisissez le listing BASIC et vous mettez en route par RUN.
3. Vous achetez la disquette du livre.

Si vous vous êtes décidé pour une des deux premières possibilités : *bonne chance !...*

D'abord le listing en assembleur qui vous montre comment fonctionne notre interface FDC/BASIC. Si vous avez une expérience en assembleur, vous trouverez toutes les routines qui permettent de gérer le FDC. La documentation devrait vous permettre d'utiliser sans problème toutes les routines dans vos propres programmes.

Les programmeurs BASIC, auxquels ce chapitre est dédié, nous excuseront cette incursion dans le monde de l'assembleur. Nous nous permettons de mettre le listing en assembleur car la programmation du FDC nous semble d'un intérêt considérable.

Si c'est seulement le programme exécutable qui vous intéresse ou si vous ne possédez que le BASIC ST, sautez cette partie et saisissez le chargeur BASIC "FDCCREAT.BAS" qui génère le programme en assembleur. Le programme en assembleur est généré sur disquette. Il fonctionne non seulement avec un BASIC ST mais également (et à plus forte raison, pourrait-on dire !) avec d'autres BASIC (GfA BASIC par exemple).

```
;*****
;*****          INTERFACE FDC/BASIC          *****
;*****
```

```
;Hardware-Register
```

```
dmamode   = $ffB606
dmascnt   = $ffB604
dmalow    = $ffB60d
dmamid    = $ffB60b
dmahigh   = $ffB609
giselct   = $ffBB00
giwrite   = $ffBB02
mfp       = $fffa01
```

```
;Mots de contrôle pour le contrôleur DMA (sens des données DMA => READ)
```

```
srcmd = $B0 ; Sélection du Command-Register
srtrk = $B2 ; Sélection du Track-Register
srsec = $B4 ; Sélection du Sector-Register
srdat = $B6 ; Sélection du Data-Register
srcnt = $B0 ; Sélection du DMA-Sectorcount-Register
```

```
;Mots de contrôle pour le contrôleur DMA (sens des données DMA => WRITE)
```

```
swcmd = $1B0 ; signification comme pour => READ
swtrk = $1B2
swsec = $1B4
swdat = $1B6
swcnt = $190
```

```
;*****
```

```
even
```

```
st:
```

```
bra.s run ; vers début du programme
```

```
;***** Mots de commande *****
```

```
rest: dc.w $01 ; Restore MD, 3ms Step-Rate
see: dc.w $11 ; Seek MD, 3ms Step-Rate
stp: dc.w $31 ; Step MD, 3ms Step-Rate, Update Trackreg.
stpi: dc.w $51 ; Step-in MD, 3ms Step-Rate, Update Trackreg.
stpo: dc.w $71 ; Step-out MD, 3ms Step-Rate, Update Trackreg.
```

```
rsec: dc.w $90 ; Read-Sector MD, multiple
wsec: dc.w $b0 ; Write-Sector MD, multiple, Write-Precompensation
```

```
radr: dc.w $c0 ; Read-Address MD,
rtrk: dc.w $e0 ; Read-Track MD,
wtrk: dc.w $f0 ; Write-Track MD, Write-Precompensation
```

```
forc: dc.w $d0 ; Force-Interrupt
```

```
;***** Paramètres de saisie *****
```

```
prm: dc.w 00 ; Numéro de fonction
dc.w 00 ; Numéro de lecteur
dc.w 00 ; Numéro de piste
dc.w 00 ; Numéro de secteur
dc.w 00 ; Nombre d'octets à transférer
dc.w 00 ; Nombre de champ d'ID à lire
dc.w 00 ; FDC-Status
dc.w 00 ; DMA-Status
dc.w 00 ; Timeout? (1=timeout)
dc.w 00 ; Nombre d'octets transférés
```

```

dc.l 00 ; DMA-Start-Adress
dc.l 00 ; DMA-End-Adress
dc.l 00 ; Adresse du tampon de piste
dc.l 00 ; Adresse du tampon de secteur
dc.l 00 ; Adresse du tampon d'ID
dc.l 00 ; Adresse du tampon d'état de l'ID

;***** C'est parti ! *****

run:

    tst.w 4(sp)          ; Des paramètres ont-ils été donnés ?
    bne exit            ; Oui, retour au BASIC

;Comme on n'a le droit d'adresser la source que relativement au PC, nous
;prenons A3 comme compteur de programme.

    lea st(pc),a3        ; Program-Start dans Adress-Reg.3
    movem.l d0-d7/a0-a6,savreg-st(a3) ; sauvegarde des Registres

;***** Set Supervisor-Mode *****

    clr.l -(sp)          ; Userstack => Superv.Stack
    move.w #$20,-(sp)    ; Command => Super
    trap #1
    addq.l #6,sp         ; Correction de la pile
    move.l d0,savstack-st(a3) ; sauvegarde de l'ancien Stackpointer

;*** Suppression de certains Flags et calcul de l'adresse absolue de ***
;*** la fonction désirée ***

    lea prm-st(a3),a5    ; pointeur sur Parameter-Block

    move.w #1,$43e       ; mémorisation du Floppy-VBL
    move.w #0,16(a5)     ; Effacer Timeout-Flag
    move.w #0,dma-st(a3) ; Effacer DMA-Flag
    move.w #0,vblflag-st(a3) ; Effacer VBL-flag de retour

    move.w 0(a5),d0       ; Récupération du numéro de fonction
    and.l #$0f,d0        ; Il n'y a que 16 fonctions (0-15)

```

```

lsl.l #2,d0          ; fois 4 = functab-Offset

lea functab-st(a3),a4 ; func-Table-Adress
move.l 0(a4,d0),d0    ; Adresse de départ relative de la routine

jsr 0(a3,d0)          ; +Programstart=Adr. abs. de la routine

tst.w vblflag-st(a3)  ; Activation du VBL (après désélection)?
beq letoff            ; non
move.w #0,$43e        ; activation
    
```

letoff:

;*** Retour en User-Mode *******

```

move.l savstack-st(a3),d0 ; Récupération de l'ancien Stackpointer
move.l d0,-(a7)           ; Saisie de l'ancien Stackpointer
move.w #$20,-(sp)         ; Command => Super
trap #1
addq.l #6,sp              ; Correction de la pile

movem.l savreg-st(a3),d0-d7/a0-a6 ; Recupération du registre
    
```

```

exit:
rts          ; retour au BASIC
    
```

; Voila ! Maintenant, vos avez (seulement) les routines

;*** Restore FDC *******

restore:

```

move.w #srcmd,dmanode ; Sélection du Command-Reg.
move.w rest-st(a3),d7 ; Command => Restore
bsr wrt1772           ; Envoi de la commande
bsr fdcwait           ; Attente jusqu'à ce que FDC soit prêt
rts
    
```

```
;***** SEEK TRACK *****
```

```
seek:
```

```
move.w #srdat,dmanode ; Sélection du registre de données
move.w 4(a5),d7        ; Tracknr. dans d7
bsr wrt1772            ; Ecrire Tracknr.
move.w #srcmd,dmanode  ; Sélection du Command-Reg.
move.w see-st(a3),d7   ; Command => Seek
bsr wrt1772            ; Ecrire commande
bsr fdcwait            ; Attente jusqu'à ce que FDC soit prêt
rts
```

```
;***** Step *****
```

```
step:
```

```
move.w #srcmd,dmanode ; Sélection du FDC-Commandreg.
move.w step-st(a3),d7 ; Command => Step
bsr wrt1772           ; Ecrire commande
bsr fdcwait           ; Attente jusqu'à ce que FDC soit prêt
rts
```

```
;***** Step in *****
```

```
stepin:
```

```
move.w #srcmd,dmanode ; Sélection du FDC-Commandreg.
move.w stepi-st(a3),d7 ; Command => Step in
bsr wrt1772           ; Ecrire commande
bsr fdcwait           ; Attente jusqu'à ce que FDC soit prêt
rts
```

```
;***** Step out *****
```

```
stepout:
```

```
move.w #srcmd,dmanode ; Sélection du FDC-Commandreg.
move.w stepo-st(a3),d7 ; Command => Step out
```



```

bsr wrt1772      ; Ecrire commande
bsr fdcwait      ; Attente jusqu'à ce que FDC soit prêt
rts
    
```

;*** Force Interrupt *******

Force:

```

move.w forc-st(a3),d7      ; Command => Force Interrupt
bsr wrt1772                ; Ecrire commande
move.w #$100,d7            ; Boucle de temporisation
wtfrc:
dbra d7,wtfrc
rts
    
```

;*** READ SECTOR(S) *******

readsector:

```

move.l 32(a5),d7      ; DMA-Adress sur tampon de secteur
bsr setdma
move.w #1,dma-st(a3)  ; Positionner DMA-Flag
move.w #srcnt,dmamode  ; DMA-R/W toggel (Flip/Flop)
move.w #swcnt,dmamode
move.w #srcnt,dmamode  ; Sélection du DMA-Sectorcount
move.w #$0c,d7        ; Charger avec 12 (Correspond à 6ko)
bsr wrt1772            ; Charger DMA-Scnt

move.w #srsec,dmamode  ; Sélectionner Sector-Reg.
move.w 6(a5),d7        ; Sektor-Nr. dans d7
bsr wrt1772            ; Ecrire Sektor-Nr.

move.w #srcmd,dmamode  ; Sélection du Command-Reg.
move.w rsec-st(a3),d7  ; Command => Read multiple Sectors
bsr wrt1772            ; Ecrire commande

bsr fdcwait            ; attente jusqu'à ce que FDC soit prêt
bsr readstat           ; Lire état et nbr d'octets
rts
    
```

```
;***** Read Address *****
```

```
readaddress:
```

```

move.l 40(a5),a4      ; Charger adresse du tampon d'état
move.l 36(a5),d7      ; DMA-Adress sur tampon de champ d'ID
bsr setdma
move.w #srcnt,dmamode ; DMA-R/W toggle
move.w #swcnt,dmamode
move.w #srcnt,dmamode ; Sélection du DMA-Sectorcount
move.w #$01,d7        ; Charger avec 1 (Correspond à 512 octets)
bsr wrt1772
move.w #srcmd,dmamode ; Sélection du FDC-Commandreg.
move.w 10(a5),d4      ; #ID-Felder dans D4
and.w  #$7f,d4        ; Mais au max. 128

```

```
idloop:
```

```

move.w radr-st(a3),d7 ; Command => Read Address
bsr wrt1772           ; Ecrire commande
bsr fdcwait           ; attente jusqu'à ce FDC soit prêt
move.b d0,(a4)+       ; Mettre l'état dans le tampon
tst.w 16(a5)          ; Timeout ?
dbne d4,idloop        ; non, lire chemp suivant
bsr readstat          ; Lire état et nombre d'octets
rts

```

```
;***** READ TRACK *****
```

```
readtrack:
```

```

move.l 28(a5),d7      ; DMA-Adress sur Track-Buffer
bsr setdma
move.w #1,dma-st(a3)  ; Positionner DMA-Flag
move.w #srcnt,dmamode ; DMA-R/W toggle
move.w #swcnt,dmamode
move.w #srcnt,dmamode ; Sélection DMA-Sectorcount
move.w #$0e,d7        ; charger avec 14 (correspond à 7ko)
bsr wrt1772
move.w #srcmd,dmamode ; Sélection du Command-Reg.
move.w rtrk-st(a3),d7 ; Command => Read Track

```

```

bsr wrt1772      ; Ecrire commande
bsr fdcwait      ; Attente jusqu'à ce que FDC soit prêt
bsr readstat     ; Lire l'état et le nombre d'octets
rts
    
```

;***** WRITE SECTOR(S) *****

writesector:

```

move.l 32(a5),d7      ; DMA-Adress sur Sector-Buffer
bsr setdma
move.w #1,dma-st(a3)  ; Positionner DMA-Flag
move.w #swcnt,dma-mode ; DMA-R/W toggle
move.w #srcnt,dma-mode
move.w #swcnt,dma-mode ; Sélection du DMA-Sectorcount
move.w #$0c,d7        ; charger avec 12 (correspond à 6ko)
bsr wrt1772          ; Ecrire DMA-Scnt
move.w #swsec,dma-mode ; Sélectionner Sector-Reg.
move.w 6(a5),d7       ; Sektornr. dans d7
bsr wrt1772          ; Ecrire Sektor-Reg.

move.w #swcmd,dma-mode ; Sélectionner Command-Reg.
move.w wsec-st(a3),d7 ; Command => Write multiple Sectors
bsr wrt1772          ; Ecrire commande
bsr fdcwait          ; Attente jusqu'à ce que FDC soit prêt
bsr readstat         ; Lire l'état et le nombre d'octets
rts
    
```

;***** WRITE TRACK *****

writetrack:

```

move.l 2B(a5),d7      ; DMA-Adress sur Track-Buffer
bsr setdma
move.w #1,dma-st(a3)  ; Positionner DMA-Flag
move.w #swcnt,dma-mode ; DMA-R/W toggle
move.w #srcnt,dma-mode
move.w #swcnt,dma-mode ; Sélectionner DMA-Sectorcount
move.w #$0e,d7        ; Charger avec 14 (correspond à 7ko)
bsr wrt1772          ; Ecrire DMA-Scnt
    
```

```

move.w #swcmd,dmamode ; Sélectionner Command-Reg.
move.w wtrk-st(a3),d7 ; Command => Write Track
bsr wrt1772 ; Ecrire commande
bsr fdcwait ; attente jusqu'à ce que FDC soit prêt
bsr readstat ; Lire l'état et le nombre d'octets
rts

```

```

;*****
;*****

```

```

;C'étaient les routines appelant les commandes du WD 1772
;
;Voici d'autre sous-routines qui sont appelées partiellement par
;la routine principale et partiellement par le BASIC (p.ex. setdrive)
;

```

```

;***** Lecture du Sector-Register *****

```

```

rsecreg:

```

```

move.w #srsec,dmamode ; Sélection du Sector-Reg.
bsr read1772 ; et lecture
and.w #$ff,d0 ; Seulement octets faibles
move.w d0,6(a5) ; dans le FDC-Array
move.w #srcmd,dmamode ; Sélection du Command-Reg.
rts

```

```

;***** Lecture du Track-Register *****

```

```

rtrkreg:

```

```

move.w #srtrk,dmamode ; Sélection du track-Reg.
bsr read1772 ; et lecture
and.w #$ff,d0 ; Seulement octets faibles
move.w d0,4(a5) ; dans FDC-Array
move.w #srcmd,dmamode ; Sélection du Command-Reg.
rts

```

```

;***** Lecture du Status-Reg. *****

```

rstareg:

```

move.w #srcmd,dmamode ; Sélection du Status-Reg.
bsr read1772           ; et lecture
and.w #$ff,d0          ; Etat dans les octets faibles
move.w d0,12(a5)       ; dans le FDC-Array
rts

```

;***** Ecriture du Track-Reg. *****

wtrkreg:

```

move.w #srtrk,dmamode ; Sélection du track-Reg.
move.w 4(a5),d7        ; Récupérer n° de piste
and.w #$ff,d7
bsr wrt1772            ; et écrire
move.w #srcmd,dmamode ; Sélection du Command-Reg.
rts

```

;***** Set DMA-Transfer Adress *****

setdma:

```

move.l d7,20(a5)       ; Mettre Start-Adress dans FDC-Array
move.b d7,dmalow       ; Ecrire d'abord Low-Byte
lsl.l #8,d7
move.b d7,dmamid       ; puis Mid-Byte
lsl.l #8,d7
move.b d7,dmahigh      ; et enfin High-Byte

```

```

move.l 20(a5),d7       ; Récupération de la Start-Adress
clr.l d6
move.w 8(a5),d6        ; Nombre d'octets à transférer
add.l d6,d7            ; Addition des deux
move.l d7,24(a5)       ; =Adresse de fin attendue
rts

```

;*** Lecture du DMA-Status; calcul des octets à transférer ***

readstat:

```

move.w dmanode,d0      ; Lecture du DMA-Status
and.w #$7,d0           ; Prendre seulement les 3 bits inférieurs
move.w d0,14(a5)       ; ert après fdcout

clr.l d1               ; Lecture de la DMA-Endadress
move.b dmahigh,d1
lsl.l #8,d1
move.b dmaaid,d1
lsl.l #8,d1
move.b dmaow,d1

move.l d1,24(a5)       ; End-Adress dans Array
sub.l 20(a5),d1        ; End-Adr. moins Start-Adr.
move.w d1,18(a5)       ; =Nombre d'octets
rts

;***** Ecriture du FDC-Register *****

wrt1772:

bsr wait
move.w d7,dmascnt      ; Ecrire FDC-Reg. ou DMA-Sectorcount
bsr wait
rts

;***** Lecture du FDC-Register *****

read1772:

bsr wait
move.w dmascnt,d0      ; Lire FDC-Reg. ou DMA-Sectorcount
bsr wait
rts

;***** Attente jusqu'à ce que FDC soit prêt *****

fdcwait:

move.l #$180,d5        ; Attendre un peu jusqu'à ce Busy soit positionn

```

```

é
litlwt:
    dbra d5,litlwt

    move.l #$40000,d5          ; d5 sert de compteur Timeout
    cmp.w #$9,0(a5)           ; Commande READ-ADDRESS?
    bne readmfp
    move.l #$2B000,d5          ; Oui, timeout plus court

readmfp:

    btst #5,mfp                ; Commande finie ?
    beq fdcready               ; oui

    subq.l #1,d5               ; non, Décrémenter compteur Timeout
    beq timeout                ; Si arrivé à la fin

    tst.w dma-st(a3)           ; Commande de transfert de données ?
    beq readmfp                ; non, Continuer le test

    move.b dmahigh,temp+1-st(a3) ; C'est la DMA-Endadress attendue
    move.b dmanid,temp+2-st(a3) ; déjà atteinte ?
    move.b dmaalow,temp+3-st(a3)
    move.l temp-st(a3),d7
    cap.l 24(a5),d7
    blt readmfp                ; non, continuer à tester

    bsr force                   ; Si oui, interrompre la commande
    move.w #0,dma-st(a3)        ; Effacer dma-Flag
    bra fdcready                ; et quitter normalement la routine

timeout:

    move.w dmascnt,d0           ; Lire l'état avant l'interruption
    and.w #$ff,d0              ; Masquer l'octet de poids fort
    move.w d0,12(a5)            ; Et mettre dans l'Array
    bsr force                   ; Arrêter la commande
    move.w #1,16(a5)            ; Positionner Timeoutflag
    rts
    
```

fdcready:

```

move.w dmascnt,d0      ; Lire l'état
and.w #$ff,d0          ; Masquer l'octet de poids fort
move.w d0,12(a5)       ; et mettre dans le FDC-Array
rts

```

;***** Attendre jusqu'à ce que le moteur soit arrêté *****

motoroff:

```

move.w #srcmd,d#amode  ; Sélection du Statusreg.
test:
bsr read1772           ; et lecture
btst #7,d0             ; Motor-on positionné
bne test               ; oui, continuer l'attente
rts

```

;***** Wait *****

wait:

```

move.w sr,-(a7)        ; Sauvegarder Status
move.w #$20,d5         ; d5 sert de compteur
wt2:
dbf d5,wt2
move.w (a7)+,sr        ; Récupérer Status
rts

```

;***** Sélection du lecteur et de la piste *****

setdrive:

```

clr.l d7
move.w 2(a5),d7        ; Récupérer Drive-Nr.
bne set
bsr motoroff           ; Si 0, d'abord désélectionner quand motor OFF
move.w #1,vblflag-st(a3) ; Positionner flag de retour de VBL
set:
eor.b #7,d7            ; Inverser les Bits pour le Hardware

```



```

and.b #7,d7          ; Seuls les 3 Low-Bits sont modifiés
move.w sr,-(a7)       ; Sauvegarder Status
or.w #$700,sr         ; Désactiver Interruptions
move.b #$e,giselect  ; Sélectionner Port A du Sound-Chips
move.b giselect,d0    ; Lire Port A
and.b #$f8,d0         ; Effacer Bits 0-2
or.b d0,d7            ; Positionner nouveaux bits
move.b d7,girwrite    ; Et écrire sur Port A
move.w (a7)+,sr       ; restore Status
rts

```

```

;*****
;***** Variables und Tableaux *****
;*****

```

even

```

savreg:  blk.l 16,0
savprm:  dc.l 0
savstack: dc.l 0

```

```

vblflag: dc.w 0
dma:     dc.w 0
temp:    dc.l 0

```

```

functab: dc.l restore-st,seek-st
         dc.l step-st,stepin-st
         dc.l stepout-st,readsector-st
         dc.l writesector-st,readtrack-st
         dc.l writetrack-st,readaddress-st
         dc.l force-st,setdrive-st
         dc.l rsecreg-st,rtrkreg-st
         dc.l rstareg-st,wtrkreg-st

```

even

```

;***** FIN *****

```

```

10  '***** File-Maker      A.S.  *****
15  '
20  ?:fullw 2:clearw 2:gotoxy 0,0
25  ? "Création du fichier >> fdcinter.img <<":?::?::?
30  dia c%( 688):cs#=0
35  for i=0 to 688
40  read a$:c%(i)=val("%H"+a$)
45  check#=check#+(c%(i))
50  next i
55  if check#= 2458472.96 then 70
60  ?"Cela ne va pas. Il y a un problème avec les DATA's."
65  goto 80
70  bsave "fdcinter.img",varptr(c%(0)), 1378
75  ? "Ecriture du programme >> fdcinter.img <<."
80  ?::?::?"Pressez une touche.":a=inp(2):end
85  '
90  '***** DATA's pour fdcinter.img *****
95  '
100 DATA 6042,0001,0011,0031,0051,0071,0090,00B0
101 DATA 00C0,00E0,00F0,00D0,0000,0000,0000,0000
102 DATA 0000,0000,0000,0000,0000,0000,0000,0000
103 DATA 0000,0000,0000,0000,0000,0000,0000,0000
104 DATA 0000,0000,4A6F,0004,6600,0074,47FA,FFB2
105 DATA 48EB,7FFF,04D2,42A7,3F3C,0020,4E41,5CBF
106 DATA 2740,0516,4BEB,001B,33FC,0001,0000,043E
107 DATA 3B7C,0000,0010,377C,0000,051C,377C,0000
108 DATA 051A,302D,0000,02B0,0000,000F,E588,49EB
109 DATA 0522,2034,0800,4EB3,0800,4A6B,051A,6700
110 DATA 000A,33FC,0000,0000,043E,202B,0516,2F00
111 DATA 3F3C,0020,4E41,5CBF,4CEB,7FFF,04D2,4E75
112 DATA 33FC,0080,00FF,B606,3E2B,0002,6100,02EA
113 DATA 6100,0306,4E75,33FC,0086,00FF,B606,3E2D
114 DATA 0004,6100,02D4,33FC,0080,00FF,B606,3E2B
115 DATA 0004,6100,02C4,6100,02E0,4E75,33FC,0080
116 DATA 00FF,B606,3E2B,0006,6100,02AE,6100,02CA
117 DATA 4E75,33FC,0080,00FF,B606,3E2B,0008,6100
118 DATA 029B,6100,02B4,4E75,33FC,0080,00FF,B606

```

```
119 DATA 3E2B,000A,6100,02B2,6100,029E,4E75,3E2B
120 DATA 0016,6100,0274,3E3C,0100,51CF,FFFE,4E75
121 DATA 2E2D,0020,6100,0202,377C,0001,051C,33FC
122 DATA 0090,00FF,8606,33FC,0190,00FF,B606,33FC
123 DATA 0090,00FF,8606,3E3C,000C,6100,023C,33FC
124 DATA 0084,00FF,B606,3E2D,0006,6100,022C,33FC
125 DATA 0080,00FF,B606,3E2B,000C,6100,021C,6100
126 DATA 0238,6100,01E0,4E75,2B6D,0028,2E2D,0024
127 DATA 6100,01A6,33FC,0090,00FF,B606,33FC,0190
128 DATA 00FF,B606,33FC,0090,00FF,B606,3E3C,0001
129 DATA 6100,01E6,33FC,00B0,00FF,B606,382D,000A
130 DATA 0244,007F,3E2B,0010,6100,01CE,6100,01EA
131 DATA 18C0,4A6D,0010,56CC,FFEC,6100,01B8,4E75
132 DATA 2E2D,001C,6100,0152,377C,0001,051C,33FC
133 DATA 0090,00FF,8606,33FC,0190,00FF,8606,33FC
134 DATA 0090,00FF,8606,3E3C,000E,6100,01EC,33FC
135 DATA 0080,00FF,8606,3E2B,0012,6100,017C,6100
136 DATA 0198,6100,0140,4E75,2E2D,0020,6100,010A
137 DATA 377C,0001,051C,33FC,0190,00FF,8606,33FC
138 DATA 0090,00FF,8606,33FC,0190,00FF,8606,3E3C
139 DATA 000C,6100,0144,33FC,0184,00FF,8606,3E2D
140 DATA 0006,6100,0134,33FC,0180,00FF,8606,3E2B
141 DATA 000E,6100,0124,6100,0140,6100,00EB,4E75
142 DATA 2E2D,001C,6100,00B2,377C,0001,051C,33FC
143 DATA 0190,00FF,8606,33FC,0090,00FF,8606,33FC
144 DATA 0190,00FF,8606,3E3C,000E,6100,00EC,33FC
145 DATA 0180,00FF,8606,3E2B,0014,6100,00DC,6100
146 DATA 00FB,6100,00A0,4E75,33FC,00B4,00FF,8606
147 DATA 6100,00D6,0240,00FF,3B40,0006,33FC,0080
148 DATA 00FF,8606,4E75,33FC,00B2,00FF,8606,6100
149 DATA 00B8,0240,00FF,3B40,0004,33FC,0080,00FF
150 DATA 8606,4E75,33FC,0080,00FF,8606,6100,009A
151 DATA 0240,00FF,3B40,000C,4E75,33FC,0082,00FF
152 DATA 8606,3E2D,0004,0247,00FF,6100,006C,33FC
153 DATA 00B0,00FF,8606,4E75,2B47,0014,13C7,FFFF
154 DATA 860D,E08F,13C7,FFFF,860B,E08F,13C7,FFFF
155 DATA 8609,2E2D,0014,4286,3C2D,0008,DE86,2B47
156 DATA 001B,4E75,3039,00FF,8606,0240,0007,3B40
157 DATA 000E,4281,1239,FFFF,8609,E189,1239,FFFF
158 DATA 860B,E189,1239,FFFF,860D,2B41,001B,92AD
```

159 DATA 0014,3B41,0012,4E75,6100,00CA,33C7,00FF
160 DATA 8604,6100,00C0,4E75,6100,00BA,3039,00FF
161 DATA 8604,6100,00E0,4E75,2A3C,0000,0180,51CD
162 DATA FFFE,2A3C,0004,0000,0C6D,0009,0000,6600
163 DATA 000B,2A3C,0002,B000,0839,0005,00FF,FA01
164 DATA 6700,005C,5385,6700,003C,4A6B,051C,6700
165 DATA FFE8,1779,FFFF,8609,051F,1779,FFFF,860B
166 DATA 0520,1779,FFFF,860D,0521,2E2B,051E,READ
167 DATA 0018,6D00,FFC4,6100,FD06,377C,0000,051C
168 DATA 6000,001C,3039,00FF,8604,0240,00FF,3B40
169 DATA 000C,6100,FCEA,3B7C,0001,0010,4E75,3039
170 DATA 00FF,8604,0240,00FF,3B40,000C,4E75,33FC
171 DATA 00B0,00FF,8606,6100,FF50,0800,0007,6600
172 DATA FFF6,4E75,40E7,3A3C,0020,51CD,FFFE,46DF
173 DATA 4E75,42B7,3E2D,0002,6600,900C,6100,FFD0
174 DATA 377C,0001,051A,0A07,0007,0207,0007,40E7
175 DATA 007C,0700,13FC,000E,00FF,BB00,1039,00FF
176 DATA BB00,0200,00FB,8E00,13C7,00FF,8B02,46DF
177 DATA 4E75,0000,0000,0000,0000,0000,0000,0000
178 DATA 0000,0000,0000,0000,0000,0000,0000,0000
179 DATA 0000,0000,0000,0000,0000,0000,0000,0000
180 DATA 0000,0000,0000,0000,0000,0000,0000,0000
181 DATA 0000,0000,0000,0000,0000,0000,0000,0000
182 DATA 0000,0000,00C0,0000,0036,0000,00FC,0000
183 DATA 0112,0000,0128,0000,0150,0000,0248,0000
184 DATA 0200,0000,02A0,0000,01AB,0000,013E,0000
185 DATA 0492,0000,02E8,0000,0306,0000,0324,0000
186 DATA 033A

Si vous n'avez commis aucune erreur lors de la saisie du programme, vous disposerez du programme FDCINTER.IMG qui vous permettra d'appeler toutes les routines du FDC (et même plus encore).

Nous vous expliquons comment réaliser l'interfaçage avec le BASIC dans la première partie du listing du chapitre suivant. C'est tellement simple qu'il est inutile de donner d'autres explications. On peut même réduire la taille de cette partie. Si vous ne voulez pas avoir toutes les informations sur une piste à la fois, il vous suffit de mettre l'adresse de départ du même tampon à la place des adresses de départ de chacun des tampons.

L'envoi de paramètres à FDCINTER.IMG

Avant de voir les programmes de démonstration, parlons un peu du programme en langage machine. Supposons que vous voulez intégrer des fonctions dans vos programmes BASIC. Il serait malaisé de devoir rechercher les informations utiles dans le listing du programme. Une bonne description du programme rend ce travail superflu.

Commençons par une vue d'ensemble des paramètres qu'il faut envoyer avec quelques informations.

* **Tableau : paramètres de saisie** *

PARAMS. D'ENTREE	N° de Fonction	N° de Lecteur	N° de Plate	Start - Sector - Nr.	Nbre d'octets à transférer	Nbre de cylindres à lire	startad. tampon de piste	startad. tampon de secteur	startad. tampon champ id	startad. tampon stat id
FONCTION	FDC%(12)	FDC%(13)	FDC%(14)	FDC%(15)	FDC%(16)	FDC%(17)	FDC%(26) FDC%(27)	FDC%(28) FDC%(28)	FDC%(30) FDC%(31)	FDC%(32) FDC%(33)
Restore	00									
Seek	01		xx							
Stop	02									
Stop - In	03									
Stop - Out	04									
Read - Sector	05			xx (1)	xx (2)			xxxx (4)		
Write - Sector	06			xx (1)	xx (2)			xxxx (4)		
Read - Track	07				xx (2)		xxxx (4)			
Write - Track	08				xx (2)		xxxx (4)			
Read - Address	09								xxxx (4)	xxxx (4)
Force - Interrupt	10									
Select. lecteur	11	xx				xx (2)				
lire reg. secteurs	12									
lire reg. pistes	13									
lire reg. Ebx	14									
Ecrire reg. pistes	15		xx							

- (1) On inscrit le numéro du premier secteur à lire ou à écrire dans FDC%(15). Faites attention à ce que cette donnée concerne toujours la piste sur laquelle se trouve la tête de lecture/écriture. Si vous voulez travailler avec des numéros de secteurs logiques, il faudra d'abord les transformer en adresse absolue de piste/secteur.
- (2) On inscrit indirectement le nombre de secteurs à lire ou à écrire en mettant le nombre d'octets transférés dans FDC%(16). Cela paraît surprenant mais présente l'avantage de pouvoir lire et écrire sans difficultés les formats avec différentes tailles de secteurs.

Si vous voulez par exemple transférer 5 secteurs au format ATARI, il faut mettre la valeur 5×512 dans FDC%(16). Si vous utiliser en fait un format de secteurs de 256 octets, il faudrait mettre 5×256 octets dans FDC%(16). On peut également utiliser ce mode d'accès pour les pistes dans lesquelles se cotoient différentes tailles de secteurs. Si vous devez par exemple lire 4 secteurs consécutifs dont les tailles respectives sont 1024, 512, 256 et 128, il faudra inscrire "1024+512+256+128" dans FDC%(16). Il est possible également de ne lire que la moitié d'un secteur par exemple. Il suffit d'indiquer le nombre correspondant d'octets.

Cela est valable également pour les commandes READ TRACK et WRITE TRACK. S'il faut traiter une piste complète, il suffit d'inscrire une valeur supérieure à 6300. En inscrivant une valeur inférieure, on ne formatera qu'une partie de la piste. En formatant une piste de plusieurs manières, on crée des particularités étonnantes sur la piste, ce qui ne manquera pas de provoquer la réflexion suivante dans les milieux pirates : "Encore un nouveau truc diabolique !".

>>>>> Très important <<<<<<<<< : si on inscrit des données (WRITE SECTOR, WRITE TRACK), il faut additionner obligatoirement le nombre 32 (\$20) au nombre d'octets à transférer. En effet, le contrôleur DMA utilise toujours 32 octets pour ses registres internes avant d'effectuer le transfert de données.

Ainsi, les données pour deux secteurs (2×512 octets = \$400) ne seront transférées que lorsque l'adresse de fin de DMA sera de \$420 supérieure à l'adresse de début de DMA.

- (3) Le nombre de champ d'ID lisibles est limité à 128. Cela suffit pour lire tous les champ d'ID des formats les plus courants. Il faut inscrire le nombre de champs - 1 dans FDC%(17). On lira au moins 3 champs avant de transférer des données dans le tampon avec le contrôleur DMA. Il faut que la valeur ($n^{\circ} \text{d'ID} - \text{Champs} \times 16$) soit divisible par 16 si on veut avoir toutes les informations sur les ID en mémoire.

(Voir description de la commande READ ADDRESS).

- (4) Les adresses de début de tampons ne sont pas indiquées à chaque appel. En général, cela ne se produit qu'une seule fois, comme dans notre programme de démonstration. Si vous gardez les informations concernant plusieurs pistes en mémoire, le programme se contente d'étendre des vecteurs dont l'adresse de début a été indiquée avant l'appel. Etant donné qu'on envoie toujours des mots longs pour les adresses, il suffit d'effectuer un POKE. Si nous utilisons par exemple un deuxième tampon de secteur, cela donne :

```
DIM SEC2%(3200) : DEF SEG = 0 : POKE FDC#+56,VARPTR(SEC2%(0))
```

Les paramètres de FDCINTER.IMG

Bien entendu, le programme en langage machine renvoie de nombreux paramètres. Nous vous les indiquons dans un tableau :

* Tableau des paramètres de retour *

Output Para.	PISTE Nr.	Secteur Nr.	ETAT du FDC	ETAT du DMA	Timeout	adr. de deb. DMA	adr. de fin DMA	nombre d'octets transférés
FONCTION	FDC%(14)	FDC%(15)	FDC%(18)	FDC%(19)	FDC%(20)	FDC%(22) FDC%(23)	FDC%(24) FDC%(25)	FDC%(21)
Restore			XX		XX			
Seek			XX		XX			
Step			XX		XX			
Step - In			XX		XX			
Step - Out			XX		XX			
Read - Sector			XX	XX	XX	XXXX	XXXX	XX
Write - Sector			XX	XX	XX	XXXX	XXXX	XX
Read - Track			XX	XX	XX	XXXX	XXXX	XX
Write - Track			XX	XX	XX	XXXX	XXXX	XX
Read - Address			XX	XX	XX	XXXX	XXXX	XX
Force - Interrupt								
Sélection lecteur								
Lire Reg. secteurs		XX						
Lire Reg. pistes	XX							
Lire Reg. état			XX					
Ecrire Reg. pistes								

Ce tableau vous indique les éléments FDC%(14) et FDC%(15) qui sont déjà utilisés lors de la spécification des paramètres. Ils constituent la seule exception. Sinon, les paramètres d'entrée et les paramètres de sortie sont très différents. Ainsi, les paramètres d'entrée ne sont jamais modifiés par le programme, sauf pour "lire registre de secteur" et "lire registre de piste".

En ce qui concerne FDC%(18), nous vous conseillons de lire les informations contenues dans le chapitre sur l'état du FDC après les commandes. Bien entendu, la description des commandes du FDC est très utile par la suite également.

L'état du DMA FDC%(19) est très simple à expliquer. Seuls 3 bits sont intéressants. Le bit 0 est positionné quand il n'y a eu aucune erreur lors du transfert DMA. Le bit 1 est positionné si le contenu de sector-count-register n'est pas arrivé à 0. Le contrôleur DMA découvre grâce à ce registre quel est le nombre maximum de données à transférer à partir de l'adresse de départ. Il est inutile de vous préoccuper de cela car le programme le prend en charge. Le bit 2 est une copie de la sortie DRQ du FDC. Lorsqu'un transfert s'est effectué sans erreur, les bit 0 et bit 1 sont positionnés. Vous ne trouverez donc 3 dans FDC%(19). Si cela n'était pas le cas, le LOST DATA bit serait automatiquement positionné dans le mot d'état du FDC.

L'adresse de début de DMA contient l'adresse du tampon courant. Après une commande READ SECTOR, on trouvera ici l'adresse de départ du tampon de secteur.

L'adresse de fin de DMA renvoie le pointeur de tampon du contrôleur DMA. Cette adresse moins l'adresse de départ est rendue dans FDC%(21) comme étant le nombre des octets transférés. Il faut faire attention à l'interprétation de cette donnée. En lecture, le pointeur atteint cette valeur après la réception de 16 (\$10) octets de données et ces octets sont transférés dans le tampon (ils étaient jusqu'alors dans la mémoire interne du DMA). En écriture, le DMA lit 32 octets (\$20 octets) dans cette mémoire temporaire et incrémente le pointeur de tampon d'autant.

Vous rencontrerez rarement un TIMEOUT (FDC%(20)=1) car le temps d'attente du programme a été correctement calculé et le FDC aura déjà interrompu la commande en cas d'erreur.

Etant donné que le FDC fait cela après 1.5 sec. seulement, nous avons réduit le délai pour la commande READ ADDRESS. En voici la raison : supposons que vous vouliez lire 100 champs d'ID (nous vous laissons définir vous-même si cela a un sens ou non) et que vous inscrivez pour cela la valeur 99 dans FDC%(17) avant d'appeler la fonction READ ADDRESS. Le programme en langage machine exécutera 100 fois la commande READ ADDRESS avant de revenir au BASIC. S'il n'y a aucun champ d'ID à trouver, vous devrez attendre plus de 2 minutes. Cela risque de vous conduire à éteindre l'ordinateur et nous ne voulons pas provoquer ce genre de réaction. Ainsi, si le FDC ne peut pas lire un champ d'ID dans le temps défini, il est interrompu par la commande FORCE INTERRUPT et revient au BASIC.

Les mots de commande du FDC

Il y a une autre chose qui rend notre interface vraiment universelle et que nous avons laissé de côté pour l'instant. Il s'agit des mots de commande qui sont envoyés au FDC. Il serait dommage de ne pouvoir les adapter à ses propres besoins. Nous avons pensé à cela. Le tableau qui suit vous indique où se trouvent les mots de commandes et quelles sont les valeurs avec lesquelles ils sont initialisés.

Mot de commande pour	se trouve dans	est initialisé avec
RESTORE	FDC%(1)	\$01
SEEK	FDC%(2)	\$11
STEP	FDC%(3)	\$31
STEP-IN	FDC%(4)	\$51
STEP-OUT	FDC%(5)	\$71
READ SECTOR	FDC%(6)	\$90
WRITE SECTOR	FDC%(7)	\$B0
READ ADDRESS	FDC%(8)	\$C0
READ TRACK	FDC%(9)	\$E0
WRITE TRACK	FDC%(10)	\$F0
FORCE INTERRUPT	FDC%(11)	\$D0

Vous pourrez trouver la signification précise du bit d'option du mot de commande dans la description des commandes du FDC.

Remarquez qu'avec READ SECTOR et WRITE SECTOR, le bit m est positionné (m pour Multi-sector READ/WRITE). Si vous effacez ce bit, un seul secteur sera traité à chaque fois.

8.3.2 Demo 1 - Accès à toutes les commandes du FDC

Après autant d'informations sur un programme en langage machine aussi petit, il serait peut-être temps de voir s'il répond à nos exigences.

Comme nous l'avons déjà indiqué dans le chapitre précédent, ce listing est constitué de 2 parties. La première partie vous montre simplement comment relier le programme au BASIC. Intéressons nous donc à la seconde partie.

Cette partie du listing est un programme qui a pour rôle de vous montrer pratiquement comment envoyer les paramètres au programme en langage machine avant l'appel de la fonction. Cela a vraiment un caractère de démonstration car il autorise l'accès direct à toutes les commandes du FDC et il affiche toutes les informations, de l'état à la donnée. Vous pourrez donc faire toutes les expériences que vous voulez avec le contrôleur de disquettes. Si vous vous posez des questions à ce sujet, le chapitre concernant le FDC y répondra sans doute.

Le programme est constitué d'un écran d'information divisé en deux parties :

1. La partie supérieure vous présente 20 fonctions. Les 16 (0-15) premières sont celles de notre interface FDC.

Avant d'appeler une commande du FDC (Fonctions 0-10), il faut sélectionner un lecteur. Les valeurs sont :

- 2 => Lecteur A face 0
- 3 => Lecteur A face 1
- 4 => Lecteur B face 0
- 5 => Lecteur B face 1
- 0 => Désélection

Si vous quittez le programme avec la fonction 19 (Fin), le lecteur sera automatiquement désélectionné.

Etant donné que certaines fonctions effectuent un transfert de données, il serait vraiment dommage de ne pas pouvoir les visualiser. Nous avons donc ajouté les options 16-18 qui permettent une visualisation des données.

Si vous étendez ce programme de manière à pouvoir modifier les données du tampon, vous aurez créé un véritable moniteur de disquette qui vous offrira des possibilités inexistantes autre part.

2. La partie inférieure de l'écran contient des paramètres qui sont envoyés aux routines ou qui proviennent des routines. En haut se trouve l'adresse de départ de tous les tampons.

A première vue, cette masse d'informations est déconcertante et présente l'appel du programme en langage machine sous un jour plus compliqué qu'il n'est en réalité. Si vous regardez dans le tableau des entrées-sorties les paramètres servant à un appel, vous verrez à côté du numéro de fonction qu'il ne faut envoyer que deux paramètres au plus. (Si l'on excepte l'adresse de départ du tampon). Pour la moitié des fonctions, il suffit d'indiquer le numéro de fonction. On pourrait difficilement faire plus simple.

En dernier lieu, il faut dire que les paramètres utilisés par les différentes fonctions sont demandés par le programme. Les paramètres utilisés précédemment sont directement affichés et si vous voulez les employer de nouveau, il suffit de presser la touche RETURN.

```

1000 '#####
1010 '#### Interface FDC en BASIC (partie 1) A.S. (7/86) ####
1020 '#####
1030 '
1040 ' L'installation des routines FDC nécessite peu de préparation. Mais
1050 ' nous avons besoin de mémoire dans laquelle nous mettrons d'une
1060 ' part les routines elles-mêmes et d'autre part les données qui sont
1070 ' nécessaires pour les opérations sur disquette. Pour cela, nous
1080 ' dimensionnons quelques vecteurs INTEGER et nous mémorisons leurs
1090 ' Adresses de départ.
1100 '
1110 dim fdc%(700) :fdc# =varptr(fdc%(0))
1120 dim trk%(3200):trk# =varptr(trk%(0))
1130 dim sec%(2600):sec# =varptr(sec%(0))
1140 dim adr%(768) :adr# =varptr(adr%(0))
1150 dim stat%(64) :stat#=varptr(stat%(0))
1160 '
1170 ' On charge les routines du lecteur dans le vecteur fdc% .....
1180 '
1190 bload "fdcinter.img",fdc#
1200 '
1210 ' et on "POKE" l'adresse de départ des autres vecteurs.
1220 '
1230 def seg = 0 : nous POKONS des mots longs
1240 '
1250 poke fdc#+52,trk# : ' Tampon de piste
1260 poke fdc#+56,sec# : ' Tampon de secteur
1270 poke fdc#+60,adr# : ' Tampon de champ d'ID
1280 poke fdc#+64,stat# : ' Tampon d'état de l'ID
1290 '
1300 ' C'est tout ! Nous vous expliquons comment appeler les fonctions
1310 ' dans la partie suivante.
1320 '
1330 '#####
1340 '##### PARTIE 2 #####
1350 '#####
1360 '
1370 ' Ceci n'est qu'une démo que nous ne voulons pas rendre trop
1380 ' désagréable. Commençons par un petit menu qui affiche les numéros
1390 ' de fonctions et tous les paramètres.

```

```

1400 'Les fonctions 0-15 sont celles qui sont traitées par la routine
1410 'en langage machine alors que les 16-18 servent seulement à étudier
1420 ' le tampon. Vous pouvez bien sûr employer ces fonctions de manière
1430 'plus confortable. La fonction 19 interrompt la démo et "désélectionne
"
1440 'le lecteur
1450 '
1460 '
1470 fullw 2 : width 255
1480 Menu:
1490 ? : clearw 2 : gotoxy 0,0
1500 '
1510 ?" ----- Fonctions disponibles -----";
1520 ?"-----"
1530 ?" 0 => Restore          1 => Seek          2 => Step  "
1540 ?" 3 => Step-in         4 => Step-out        5 => Read-S";
1550 ?"ector"
1560 ?" 6 => Write-Sector     7 => Read-Track      8 => Write-";
1570 ?"Track"
1580 ?" 9 => Read-Address     10=> Force-Interrupt  11=> Select";
1590 ?" Drive"
1600 ?" 12=> Lire reg. secteur 13=> lire reg. piste  14=> Status";
1610 ?"Lire reg."
1620 ?" 15=> Ecrire reg. piste 16=> affich. tampon piste17=> Sector";
1630 ?"-Afficher tampon"
1640 ?" 18=> afficher champ d'ID 19=> Fin du programme":?
1650 '
1660 ?" ----- Affichage de tous les param. -----"
;
1670 ?"-----"
1680 '
1690 ?" Fonction :          FDC-Status :$          tampon-piste :$"
1700 ?" Lecteur :          DMA-Status :$          tampon-secteur :$"
1710 ?" Piste :            Timeout :$            Tampon-champ-ID:$"
1720 ?" Secteur :          DMA-Start :$           Tampon-état-ID :$"
1730 ?" #octets :$         DMA-Fin :$"
1740 ?" #Id's -1 :         #DMA-bytes :$"
1750 ?" -----";
1760 ?"-----"
1770 '
    
```

```

1780 main:
1790 '=====
1800 gotoxy 06,10 :?right$(" "+str$(fdc%(12)),4)
1810 gotoxy 06,11 :?right$(" "+str$(fdc%(13)),4)
1820 gotoxy 06,12 :?right$(" "+str$(fdc%(14)),4)
1830 gotoxy 06,13 :?right$(" "+str$(fdc%(15)),4)
1840 gotoxy 06,14 :?right$(" "+hex$(fdc%(16)),4)
1850 gotoxy 06,15 :?right$(" "+str$(fdc%(17)),4)
1860 gotoxy 17,10 :?right$(" "+hex$(fdc%(18)),6)
1870 gotoxy 17,11 :?right$(" "+hex$(fdc%(19)),6)
1880 gotoxy 17,12 :?right$(" "+hex$(fdc%(20)),6)
1890 gotoxy 17,13 :?right$(" "+hex$(fdc%(22))+hex$(fdc%(23)),6)
1900 gotoxy 17,14 :?right$(" "+hex$(fdc%(24))+hex$(fdc%(25)),6)
1910 gotoxy 17,15 :?right$(" "+hex$(fdc%(21)),6)
1920 gotoxy 30,10 :?right$(" "+hex$(fdc%(26))+hex$(fdc%(27)),6)
1930 gotoxy 30,11 :?right$(" "+hex$(fdc%(28))+hex$(fdc%(29)),6)
1940 gotoxy 30,12 :?right$(" "+hex$(fdc%(30))+hex$(fdc%(31)),6)
1950 gotoxy 30,13 :?right$(" "+hex$(fdc%(32))+hex$(fdc%(33)),6)
1960 '
1970 gotoxy 0,17:?spc(220);
1980 key:
1990 gotoxy 1,17:?spc(50)
2000 gotoxy 1,17:input " Ouelle fonction";fnc$:fnc=val(fnc$)
2010 if fnc<0 or fnc>19 then menu
2020 fnc=fnc+1
2030 if fnc=20 then fdc%(12)=11:fdc%(13)=0:call fdc#:end
2040 '
2050 if fnc<17 then 2110
2060 reset
2070 fnc=fnc-16:clearw 2:
2080 on fnc gosub dumptrk,dumpsec,dumpid
2090 openw 2:goto key
2100 '
2110 on fnc gosub a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p
2120 gotoxy 1,19:? "Exécution de la fonction (o/n) ?";
2130 if chr$(inp(2))<>"o" then main
2140 call fdc#
2150 goto main
2160 '
2170 '*****

```



```

2180 '#####
2190 '
2200 'Voici les 16 fonctions créées par nos routines en langage
2210 'machine. Vous voyez quels sont les paramètres à définir avant
2220 'l'appel "call fdc#". Dans de nombreux cas, il suffit de mettre
2230 'le numéro de fonction dans fdc%(12).
2240 '
2250 '
2260 '===== RESTORE =====
2270 a:
2280 fdc%(12)=0
2290 gotoxy 1,17:"RESTORE - Pas de paramètre";:return:
2300 '
2310 '===== SEEK =====
2320 b:
2330 fdc%(12)=1
2340 gotoxy 1,17:"SEEK - Quel numéro de piste.(ancien=>";
2350 ?fdc%(14);":)";:input v$:if len(v$)=0 then return
2360 fdc%(14)=val(v$):return
2370 '
2380 '===== STEP =====
2390 c:
2400 fdc%(12)=2
2410 gotoxy 1,17:"STEP - Pas de paramètre";:return
2420 '
2430 '===== STEP_IN =====
2440 d:
2450 fdc%(12)=3
2460 gotoxy 1,17:"STEP IN - Pas de paramètre";:return
2470 '
2480 '===== STEP_OUT =====
2490 e:
2500 fdc%(12)=4
2510 gotoxy 1,17:"STEP OUT - Pas de paramètre";:return
2520 '
2530 '===== READ_SECTOR(s) =====
2540 f:
2550 fdc%(12)=5
2560 gotoxy 1,17:"READ SECTOR - Quel secteur de départ (ancien=>";
2570 ?fdc%(15);":)";:input v$:if len(v$)=0 then 2590

```

```
2580 fdc%(15)=val(v$)
2590 gotoxy 1,18:"Nombre d'octets (ancien=>$";hex$(fdc%(16));)";
2600 input v$:if len(v$)=0 then return
2610 fdc%(16)=val(v$):return
2620 '
2630 '===== WRITE_SECTOR(s) =====
2640 g:
2650 fdc%(12)=6
2660 gotoxy 1,17:"WRITE SECTOR - Quel secteur de départ (ancien=";
2670 ?">";fdc%(15);)";:input v$:if len(v$)=0 then 2690
2680 fdc%(15)=val(v$)
2690 gotoxy 1,18:"Nombre d'octets (ancien=>$";hex$(fdc%(16));)";
2700 input v$:if len(v$)=0 then return
2710 fdc%(16)=val(v$):return
2720 '
2730 '===== READ_TRACK =====
2740 h:
2750 fdc%(12)=7
2760 gotoxy 1,17:"READ TRACK - Nombre d'octets (ancien=>$";
2770 ?hex$(fdc%(16));)";:input v$:if len(v$)=0 then return
2780 fdc%(16)=val(v$):return
2790 '
2800 '===== WRITE_TRACK =====
2810 i:
2820 fdc%(12)=8
2830 gotoxy 1,17:"WRITE TRACK - Nombre d'octets (ancien=>$";
2840 ?hex$(fdc%(16));)";:input v$:if len(v$)=0 then return
2850 fdc%(16)=val(v$):return
2860 '
2870 '===== READ_ADDRESS =====
2880 j:
2890 fdc%(12)=9
2900 gotoxy 1,17:"READ ADDRESS - Nombre de champs d'ID - 1 (ancien=";
2910 ?fdc%(17);)";:input v$:if len(v$)=0 then return
2920 fdc%(17)=val(v$):return
2930 '
2940 '===== FORCE INTERRUPT =====
2950 k:
2960 fdc%(12)=10
2970 gotoxy 1,17:"FORCE INTERRUPT - Pas de paramètres";:return
```

```

2980 '
2990 '===== Sélection du lecteur =====
3000 l:
3010 fdc%(12)=11:gotoxy 1,17
3020 ?"(X=Lfw/Seite) : 2=A/0; 3=A/1; 4=B/0; 5=B/1; 0=deselection"
3030 gotoxy 1,18:?"Quel lecteur (ancien=>";fdc%(13);)";
3040 input v$:if len(v$)=0 then return
3050 fdc%(13)=val(v$):return
3060 '
3070 '===== Lecture du registre de secteurs =====
=
3080 m:
3090 fdc%(12)=12
3100 gotoxy 1,17:?"LECTURE DU SECTOR-REGISTER - Pas de paramètres";
3110 return
3120 '
3130 '===== Lecture du registre de pistes =====
3140 n:
3150 fdc%(12)=13
3160 gotoxy 1,17:?"LECTURE DU TRACK-REGISTER - Pas de paramètres";
3170 return
3180 '
3190 '===== Lectuire du registre d'état =====
=
3200 o:
3210 fdc%(12)=14
3220 gotoxy 1,17:?"LECTURE DU STATUS-REGISTER - Pas de paramètres";
3230 return
3240 '
3250 '===== Ecriture du registre de pistes =====
=
3260 p:
3270 fdc%(12)=15
3280 gotoxy 1,17:?"ECRIRE TRACK-REGISTER - Quel n° de piste (ancien=>";
3290 ?fdc%(14);)";:input v$:if len(v$)=0 then return
3300 fdc%(14)=val(v$):return
3310 '
3320 '#####
3330 '#####
3340 '

```

```

3350 'Les fonctions qui suivent(16-18) n'ont rien à voir avec les routines
3360 'en langage machine. Elle affichent le contenu des tampons
3370 '
3380 '===== Affichage du tampon de piste =====
3390 dumptrk:
3400 gotoxy 0,0:"AFFICHER TRACK-BUFFER (toutes valeurs=>hex, (s)uite, ";
3410 ?" (f)in)":?
3420 ?" BUFFER 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 01234";
3430 ?"56789ABCDEF"
3440 ?" -----";
3450 ?"-----"
3460 ch$=" "
3470 '
3480 lcnt=0
3490 for id = 1 to (fd%16)+1-16 step 16
3500 lcnt=lcnt+1
3510 id%(1)=id-1:" "+right$("0000"+hex$(id%(1)),4);" ";
3520 for by=0 to 15:if seg=id+by:id%(1)=peek(trk#-1)
3530 ?right$("00"+hex$(id%(1)),2);" ";
3540 if id%(1)=7 or id%(1)=10 or id%(1)=13 then id%(1)=20
3550 mid$(ch$,by+1,1)=chr$(id%(1)):next by:" ";ch$
3560 '
3570 if lcnt<10 then 3610
3580 lcnt=0:dum=inp(2)
3590 if chr$(dum)="f" then id=70000:goto 3610
3600 if chr$(dum)<>"s" then 3580
3610 next id
3620 ?"Fini ! Pressez une touche...";
3630 dum=inp(2):return
3640 '
3650 '===== Affichage du tampon de secteurs =====
3660 =
3660 dumpsec:
3670 lcnt=0
3680 gotoxy 0,0:"AFFICHER SECTOR-BUFFER (Toutes valeurs=>hex, (s)uite, ";
3690 ?" (f)in)":?
3700 ?" BUFFER 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 01234";
3710 ?"56789ABCDEF"
3720 ?" -----";

```

```

3730 ?"-----"
3740 '
3750 ch$="          "
3760 for id = 1 to (fdc%(16)+1)-16 step 16
3770 lcnt=lcnt+1
3780 id%(1)=id-1:" "+right$("0000"+hex$(id%(1)),4);" ";
3790 for by=0 to 15: def seg=id+by: id%(1)=peek(sec#-1)
3800 ?right$("00"+hex$(id%(1)),2);" ";
3810 if id%(1)=7 or id%(1)=10 or id%(1)=13 then id%(1)=20
3820 mid$(ch$,by+1,1)=chr$(id%(1)):next by:" ";ch$
3830 '
3840 if lcnt<10 then 3880
3850 lcnt=0:dum=inp(2)
3860 if chr$(dum)="e" then id=70000:goto 3880
3870 if chr$(dum)<>"w" then 3850
3880 next id
3890 ?"Fin! Pressez une touche...";
3900 dum=inp(2):return
3910 '
3920 '===== Affichage du champ d'ID =====
=
3930 dupid:
3940 gotoxy 0,0:"AFFICHER CHAMPS-ID (toutes valeurs=>hex, (s)uite, (f)in)
"
3950 ?:" BUFFER TRACK FACE SECTOR LONGUEUR CRC1 CRC2 FDC-Status"
3960 ?" -----"
3970 '
3980 lcnt=0
3990 for id = 1 to (fdc%(17)+1)*6 step 6
4000 lcnt=lcnt+1
4010 id%(1)=id-1:" "+right$("00"+hex$(id%(1)),2);" ";
4020 for by=0 to 5: def seg=id+by: id%(1)=peek(adr#-1)
4030 ?right$("00"+hex$(id%(1)),2);" ";:next by
4040 def seg=id/6+1: id%(1)=peek(stat#-1)
4050 ?" ";right$("00"+hex$(id%(1)),2)
4060 '
4070 if lcnt<9 then 4110
4080 lcnt=0:dum=inp(2)
4090 if chr$(dum)="e" then id=1000:goto 4110
4100 if chr$(dum)<>"w" then 4080

```

```
4110 next id
4120 ?"Fin! ! Pressez une touche...";
4130 dum=inp(2):return
4140 '
4150 '===== FIN =====
```

8.3.3 Demo 2 - Copie de disquette

Le petit programme BASIC qui suit vous montre une autre application des routines FDC et de la simplicité de l'accès direct au contrôleur avec le BASIC. Le résultat est un programme de copie qui duplique une disquette en environ 85 secondes.

Il n'est vraiment pas difficile d'écrire un tel programme. La seule limitation est que ce programme fonctionne avec deux lecteurs. Cela tient au fait que nous n'avons pas voulu utiliser trop de tampons. En effet, le ST BASIC autorise seulement des vecteurs de 32 Koctets.

Cela nous empêche donc de faire un "DIM sec%(79,2303)" ce qui nous aurait permis de copier d'une seule fois la moitié d'une disquette double face. On pourrait obtenir le même résultat avec "DIM sec1%(2303),sec2%(2303),..." mais nous n'avons pas envie de le faire. C'est une simple démonstration et nous ne voulons pas vous ôter le plaisir de faire les développements vous-même.

Mais revenons à nos moutons. Nous mémorisons à chaque fois les informations concernant 2 pistes (pour la face recto et la face verso) et nous les inscrivons sur la même piste de la disquette objet.

Ce processus aurait pour conséquence de devoir changer de disquette 80 fois avec un seul lecteur ! C'est bien sûr inutilisable.

Le DESKTOP prend environ 195 Sec pour la copie d'une disquette double face. Ce temps est de 50 % plus élevé que notre programme (130 sec.) pour le même travail.

Cette valeur est considérablement réduite si l'on considère que la piste en cours de traitement est affichée en même temps. Le routine FDC dispose d'un OVERHEAD qui est exécuté à chaque fois (p. ex. activation/désactivation du mode superviseur). De plus, la désélection des lecteurs survient plusieurs fois. Malgré ces limitations, 130 secondes est un temps acceptable.

On peut obtenir un véritable TUNE UP en supprimant les instructions PRINT (lignes 60, 66, 76 et 82) qui ralentissent considérablement le temps. Cela correspond à un gain de temps de 45 secondes, avec une perte de confort d'utilisation malheureusement. Ainsi, le temps de copie ne prend que 85 secondes !

Dans ce programme, il y a un cas où une modification d'un mot de commande est la condition sine qua non pour l'écriture du registre de piste.

Commençons avec le fonctionnement du programme sur la piste 0. A la fin de la lecture de cette piste (recto verso), un STEP IN est exécuté sur le lecteur A. La tête de lecture/écriture se trouve alors sur la piste 1 de ce lecteur et le registre de piste contient la valeur 1.

Il faut alors écrire la piste 0 sur le lecteur B. Si l'accès est incorrect, voici ce qui se passe :

Le FDC interrompt la commande WRITE SECTOR avec une erreur Record Not Found car le registre de piste est à 1 alors que les champs d'ID ont pour valeur 0 en ce qui concerne le numéro de piste.

On peut avoir deux solutions :

- 1) Dans une commande STEP IN pour le lecteur A, le bit u est effacé dans le mot de commande ce qui signifie que le registre de piste n'est pas modifié.
- 2) Après un STEP IN sur le lecteur A, on inscrit dans le registre de piste la même valeur que précédemment. Dans notre programme, cela peut prendre la forme suivante :

FDC%(12) = 15 : FDC%(14) = piste : CALL FDC#

Nous nous sommes décidés pour la première solution. Mais il faut modifier le mot de commande pour chaque STEP IN car il faudra ensuite effectuer un STEP IN sur le lecteur B. Si on ne modifiait que le registre de piste, il ne serait nécessaire de le faire qu'une seule fois. Mais un CALL FDC# supplémentaire serait indispensable. La solution consistant à modifier le mot de commande est plus rapide, même si elle est exécutée deux fois.

Une dernière remarque concernant le nombre d'octets transférés (on ne le répètera jamais assez). En lecture, on inscrit toujours le nombre désiré. Pour 9 secteurs de 512 octets (une piste), on aura donc : $9 * \$20 = \1200 . C'est très logique mais en écriture, c'est différent. Il faudra inscrire la valeur suivante : $9 * \$200 + \20 .


```

1  '## Programme de copie pour 2 lecteurs et disquettes DF/DD A.5##
2  '
3  'Nous avons besoin de 3 vecteurs, pour le programme en langage
4  'machine et pour le tampon de secteur des pistes 0 et 1
5  '
6  dim fdc%(700),sec0%(2400),sec1%(2400):def seg=0
7  '
8  'Chargement du programme en langage machine
9  '
10 fdc#=varptr(fdc%(0)):bload "fdcinter.img",fdc#
11 '
12 'Mémorisation des adresses des deux tampons
13 '
14 sec0#=varptr(sec0%(0)):sec1#=varptr(sec1%(0))
15 ' (lire=$1200,écrire=$1220) le nbre d'octets à transférer
16 '
17 'et les mots de commande pour STEP-IN (avec et sans sauvegarde)
18 '
19 lirenbr=&H1200:ecritrenbr=&H1220:stpi=&H49:stpiu=&H59
20 '
21 'Nous commençons sur chaque piste avec le secteur 1 et nous posons
22 'des mots longs
23 fdc%(15)=1:def seg=0
24 '
25 copier:
26 ?:"fullw 2:clearw 2:gotoxy 0,1
27 ?:" Programme de copie pour disquettes DF/DD et 2 lecteurs
28 ?:" Disquette source dans A
29 ?:" et disquette objet dans lecteur B.
30 ?:" c => copier : autre touche => Fin du programme"
31 if chr$(inp(2))<>"c" then end
32 '
33 init:
34 clearw 2:gotoxy 0,2
35 '
36 '----- Restore lecteur B et teste Write-protect-----
37 '
38 fdc%(12)=11:fdc%(13)=4:call fdc#
39 fdc%(12)=0:call fdc#
40 if fdc%(18) < &H47 then goto copi

```

```

41 '
42 ?" La disquette du lecteur B est protégée! Enlevez la      ";
43 ?" protection.      ":?
44 ?" s => suite ; autre touche => recommencer"
45 fdc%(12)=11:fdc%(13)=0:call fdc#
46 if chr$(inp(2))="s" then init
47 goto copier
48 '
49 copi:
50 '----- Restore lecteur A-----
51 fdc%(12)=11:fdc%(13)=2:call fdc#
52 fdc%(12)=0:call fdc#
53 '
54 '----- COPIE DES PISTES 0 A 79 -----
55 '
56 for piste = 0 to 79 : fdc%(16)=lirenbre
57 '
58 '----- Lecture face A/0 et affichage de l'état---
59 fdc%(12)=5:poke fdc#+56,sec0#:call fdc#
60 gotoxy 10,2:"piste";piste;"Lecture face 0      "
61 gosub checkstat
62 '
63 '----- Lecture face A/1 et affichage de l'état---
64 fdc%(12)=11:fdc%(13)=3:call fdc#
65 fdc%(12)=5:poke fdc#+56,sec1#:call fdc#
66 gotoxy 10,2:"piste";piste;"lecture face 1      "
67 gosub checkstat
68 '
69 '----- Step In lecteur A sans sauvegarde-
70 fdc%(12)=3:fdc%(4)=stpi:call fdc#
71 '
72 '----- Ecriture face B/0 et affichage de l'état-----
73 fdc%(16)=ecrirenbr
74 fdc%(12)=11:fdc%(13)=4:call fdc#
75 fdc%(12)=6:poke fdc#+56,sec0#:call fdc#
76 gotoxy 10,4:"piste";piste;"Ecriture face 0      "
77 gosub checkstat
78 '
79 '----- Ecriture face B/1 et affichage de l'état-----
80 fdc%(12)=11:fdc%(13)=5:call fdc#

```

```

81  fdc%(12)=6:poke fdc#+56,sec1#:call fdc#
82  gotoxy 10,4:"piste";piste;"Ecriture face 1      "
83  gosub checkstat
84  '
85  '----- Step In lecteur B avec sauvegarde---
86  fdc%(12)=3:fdc%(4)=stpiu:call fdc#
87  '
88  '----- et de nouveau selection A/0-----
89  fdc%(12)=11:fdc%(13)=2:call fdc#
90  '
91  next piste
92  '
93  fdc%(12)=11:fdc%(13)=0:call fdc#
94  ?:"Termine ! .....(r)ecommence ou (f)in ?"
95  if chr$(inp(2))<>"r" then end
96  goto copier
97  '-----
98  checkstat:
99  if fdc%(18)=8#80 and fdc%(19)=3 and fdc%(20)=0 then return
100 gotoxy 0,7:" FDC-STATUS :$";hex$(fdc%(18))
101 ?" DMA-STATUS :$";hex$(fdc%(19))
102 ?" #DMA-BYTES :$";hex$(fdc%(21))
103 ?" TIMEDOUT   :$";hex$(fdc%(20)):?
104 '
105 ?" La copie a été interrompue en raison d'une erreur.      "
106 ?:" Pressez une touche ... "
107 fdc%(12)=11:fdc%(13)=0:call fdc#
108 key=inp(2):goto copier

```

8.3.4 Demo 3 - Création de formats standards et spéciaux

Le programme qui suit vous montre une autre application des routines du FDC, pour le formatage de disquettes.

Le programme est très simple d'emploi. D'une part, vous voyez clairement comment un tampon de piste est préparé (le tampon de piste est écrit sur la disquette avec WRITE TRACK et crée ainsi le format). D'autre part, vous pourrez formater les disquettes de manière à pouvoir les utiliser sur d'autres systèmes (dont les lecteurs de disquette sont gérés par un WD 1772 ou compatible). L'inverse est bien sûr possible également. Il peut survenir qu'un format ne convienne pas pour les possibilités du FDC et qu'il ne puisse pas créer les secteurs dans ce format. N'oubliez donc pas la remarque qui va suivre.

Important : La création de format est un travail qui exige une connaissance parfaite de la commande WRITE TRACK. La modification des paramètres doit être faite avec précaution. Cela fonctionne souvent mais pas toujours. En clair : de mauvaises valeurs donnent de mauvais résultats. Reportez-vous à la description des commandes du FDC. Vous y verrez quels seront les effets des modifications sur les pistes. Pour vos premiers essais, voyez les formats non standards qui sont décrits à la fin de la partie sur la commande WRITE TRACK.

Voyons quelles sont les possibilités du programme :

- 1) Un tampon de piste peut être préparé avec de nombreux paramètres (dont les limites sont larges). Pour savoir quelles sont les valeurs utilisables, les paramètres sont initialisés avec le format ATARI. Les paramètres peuvent à tout moment être remis à ces valeurs.

Le tampon est suffisamment grand pour contenir tous les paramètres en plus du format de la piste. Bien sûr le tampon prédéfini ne peut pas être complet. Il y a certaines valeurs qui diffèrent suivant les pistes ou les secteurs. Ainsi, on met le numéro de la piste qui doit être formatée dans le champ d'ID.

Les adresses des champs d'ID sont inscrites dans le tampon de piste.

- 2) Vous n'avez pas besoin de noter les valeurs que vous spécifiez pour les saisir de nouveau à chaque mise en route du programme. Vous pouvez inscrire votre tampon sur disquette sous la forme d'un fichier. Ainsi, vous pouvez vous créer une bibliothèque de format standards ou protégés.
- 3) A quoi bon sauvegarder le contenu des tampons s'il ne peut pas être chargé par la suite ? Vous pouvez bien sûr charger un fichier de formatage que vous avez déjà sauvegardé.
- 4) Bien entendu, vous pouvez formater une disquette. Cette routine de formatage récupère les adaptations des numéros de piste dans les champs d'ID. Vous pouvez également spécifier une valeur qui sera additionnée à chaque numéro de piste. Cela sert seulement pour les protections contre les copies.

```

10  '#####
12  '#### Formats multiples de disquettes      A.S. (7/86) ####
14  '#####
16  dim fdc%(700) : fdc# = varptr(fdc%(0))
18  dim trk%(3200) : trk# = varptr(trk%(0))
20  bload "fdcinter.img", fdc#
22  def seg=0: poke fdc#+52, trk#
24  '#####
26  gosub default: 'Lecture des valeurs standards pour le format ATARI
28  menu:
30  ?:fullw 2: clearw 2: gotoxy 0,0:width 80
32  '
34  ?" a) Modification du GAP1                !"
36  ?" b) Modification du GAP2                !"
38  ?" c) Modification du GAP3 (partie 1)      !"
40  ?" d) Modification du GAP3 (partie 2)      !"
42  ?" e) Modification du GAP4                !"
44  ?" f) Modification du champ de données    !"

```

```

46  ?" g) Modif. octets-SYNC (avant champ d'ID)  !"
48  ?" h) Modif. oct.-SYNC (avant champ données) !"
50  ?" i) Modification du DATA-ADDRESS-MARK    !"
52  ?" j) Modification du START-SECTOR          !"
54  ?" k) Modif. longueur secteur (ds champ d'ID) !"
56  ?" l) Modification du nbr de RECORD          !"
58  ?" m) Modification du GAP5                   !"
60  ?" =====";
62  ?"=====
64  ?" n) Préparation du tampon de piste          ";
66  ?" q) Mettre valeurs au format ATARI"
68  ?" o) Charger tampon de piste sur disque  ";
70  ?" r) Formater disquette"
72  ?" p) Sauver tampon de piste en fichier  ";
74  ?" s) Fin du programme"
76  '
78  for prm=0 to 15 Step 2
80  gotoxy 21,prm/2:"Nombre:";trk%(3150+prm);"  "
82  gotoxy 30,prm/2:"Val:$ ";hex$(trk%(3151+prm));"  ":next prm
84  for prm=16 to 20
86  gotoxy 30,prm-B:"Val:$ ";hex$(trk%(3150+prm));"  ":next prm
88  '
90  touche:
92  gotoxy 0,18:spc(60):gotoxy 0,18:" Quelle fonction?";
94  key=inp(2):if chr$(key)<"a" or chr$(key)>"s" then 94
96  choix=key+1-asc("a")
98  if choix=19 then end
100 if choix<9 then goto deuxvaleurs
102 if choix<14 then goto unevaleur
104 choix=choix-13:
106 on choix gosub preparer,charger,sauver,default,formater
108 goto menu
110 '
112 '===== Saisie du nombre et des valeurs =====
=
114 deuxvaleurs:
116 gotoxy 0,18:spc(60):gotoxy 0,18
118 ?" >> ";chr$(key);" << Saisissez un nouveau nombre: ";:input nbr$
120 if len(nbr$)=0 then goto deux
122 trk%(3148+choix*2)=val(nbr$):gotoxy 21,choix-1:"Nombre:";

```

```

124 ?trk%(3148+choix*2);"  "
126 deux:
128 gotoxy 0,18:?spc(60):gotoxy 0,18
130 ?" >> ";chr$(key);" << Saisissez une nouvelle valeur: ";:input w$
132 if len(w$)=0 then goto touche
134 trk%(3149+choix*2)=val(w$):gotoxy 30,choix-1:"valeur:$ ";
136 ?hex$(trk%(3149+choix*2));"  ":goto touche
138 '
140 '===== Saisie d'une valeur =====
142 unevaleur:
144 gotoxy 0,18:?spc(60):gotoxy 0,18
146 ?" >> ";chr$(key);" << Saisissez une nouvelle valeur: ";:input w$
148 if len(w$)=0 then goto touche
150 trk%(3157+choix)=val(w$):gotoxy 30,choix-1:"Wert:$ ";
152 ?hex$(trk%(3157+choix));"  ":goto touche
154 '
156 '===== PREPARATION DU TAMPON DE PISTE =====
=
158 preparer:
160 clearw 2:gotoxy 12,0:"préparer tampon de piste":?
162 '----- TEST DE LA LONGUEUR TOTALE -----
-
164 totale=0
166 for i=3152 to 3164 step 2:totale=totale+trk%(i):next i
168 totale=(totale+9)*trk%(3169)+trk%(3150)
170 if totale <= 6234 then goto suite
172 if totale >6250 then ?:"?:" Information-piste trop longue":goto fail
174 ?:"?:" Il ne reste pour GAP5 (Post-piste) que ";6250-totale;
176 ?" octets."?:?" C'est trop peu!"
178 fail:
180 ?:"?:" Pressez une touche ...":key=inp(2):return
182 suite:
184 '----- ECRITURE DU TAMPON -----
186 offset=1:" Pre-piste (";trk%(3150);"octets )
188 nombre=trk%(3150):valeur=trk%(3151):gosub bufpoke:' GAP1
190 for record = 1 to trk%(3169):?" Record:";record
192 nombre=trk%(3152):valeur=trk%(3153):gosub bufpoke:' GAP2
194 nombre=trk%(3162):valeur=trk%(3163):gosub bufpoke:' SYNC
196 def seg=offset:trk%(3170+record)=offset:'Marquage de l'ID-Adr.
198 poke trk#-1,&Hfe:' ID-AM

```

```

200 poke trk#+2,record-1+trk%(3167):' START-SECTOR
202 poke trk#+3,trk%(3168):' LONGUEUR SECTEUR
204 poke trk#+4,&Hf7:offset=offset+6:' ID-CRC
206 nombre=trk%(3154):valeur=trk%(3155):gosub bufpoke:' GAP3
208 nombre=trk%(3156):valeur=trk%(3157):gosub bufpoke:' GAP3
210 nombre=trk%(3164):valeur=trk%(3165):gosub bufpoke:' SYNC
212 def seg=offset:poke trk#-1,trk%(3166):' DAM
214 offset=offset+1
216 nombre=trk%(3160):valeur=trk%(3161):gosub bufpoke:' DONNEES DU SECTEUR
218 def seg=offset:poke trk#-1,&Hf7:offset=offset+1:' DATA-CRC
220 nombre=trk%(3158):valeur=trk%(3159):gosub bufpoke:' GAP4
222 next record
224 ?" Post-piste (";6250-offset;"octets )"
226 nombre=6300-offset:valeur=trk%(3170):gosub bufpoke:' GAP5
228 '
230 ?:" Le tampon est prêt!":pready=1:return
232 '-----
234 bufpoke:
236 For i = 0 to nombre-1:def seg=offset+i:poke trk#-1,valeur:next i
238 offset=offset+nombre:return
240 '
242 '===== Valeurs standards pour le format ATARI =====
244 default:
246 restore 406:for i=3150 to 3170:read standard:trk%(i)=standard
248 next i: pready=0:return
250 '
252 '===== ECRITURE TAMPON DE PISTE SUR DISQUETTE =====
254 sauver:
256 clearw 2:gotoxy 12,2:"Ecriture du tampon de piste sur disque":?:"?:"
258 if pready=1 then goto mem2
260 ?" Le tapon de piste est encore vide. Préparez le avant la ";
262 ?"sauvegarde!":?:" Pressez une touche ...":key=inp(2):return
264 mem2:
266 ?:" Sauver les données du tampon sous forme de fichier (o/n) ?":?:"
268 if chr$(inp(2))<>"o" then return
270 input " Entrez les noms de fichiers pour les données du format:",file$
272 bsave file$,trk#,6402:return

```



```

274 '
276 '===== CHARGER TAMPON DE PISTE SUR DISQUETTE =====
278 charger:
280 clearw 2:gotoxy 12,2:"Chargement du tampon de piste":??:?
282 ? " Quelle est le nom du fichier contenant le format ?";
284 ? :?:input file$
286 open"R",1,file$,1:test=lof(1):close 1
288 if test=6402 then bload file$,trk#:pready=1:return
290 ??:? " Ce n'est pas un fichier de données !!!  Pressez une touche...
."
292 a=inp(2):return
294 '===== FORMATTAGE DE LA DISQUETTE =====
296 formater:
298 clearw 2:gotoxy 12,2:"Formattage de la disquette":??:?
300 if pready=1 then goto frmt2
302 ?" Le tampon de piste est encore vide. Préparez le ";
304 ?"d'abord!":??:? Pressez une touche ...":key=inp(2):return
306 frmt2:
308 ?" Voulez-vous formater une disquette (o/n) ?"
310 if chr$(inp(2))<>"o" then return
312 '
314 ??:?" Mettez la disquette à formater dans le lecteur A.":?
316 input "A partir de quelle piste?",a$:a=val(a$):if a<0 or a>82 then 31
6
318 input " Jusqu'à quelle piste ?",a$:b=val(a$)
320 if b<0 or b>82 then 318:if b<a then 316
322 ??:? " A partir de quelle valeur les numéros de piste des champ d'ID";
324 ?"commencent-ils?":? " Ils peuvent prendre une valeur de 0 à";244-b;
326 ?" Mais elle est normalement de >> 0 <<":?
328 input " Valeur? ",a$:if val(a$)<0 or val(a$)>244-b then 328
330 off=val(a$)
332 ??:? " Quelle face doit être formatée ?"
334 ??:? " (0)=recto ou (1)=verso"
336 key=inp(2)
338 if chr$(key)="0" then drive=2:goto format
340 if chr$(key)="1" then drive=3:goto format
342 goto 336
344 format:
346 ??:? " Dernière possibilité d'interruption! (f)ormatter (q)uitter"
348 key=inp(2):if chr$(key)="q" then goto formater

```

```

350 if chr$(key)<>"f" then 348
352 '----- formater -----
354 fdc%(12)=11:fdc%(13)=drive:call fdc#:' Sélection du lecteur
356 encore:
358 fdc%(12)=0:call fdc#:'RESTORE
360 if fdc%(18)<&Ha7 then goto sek
362 "??" *** ERREUR! La disquette est protégée!"
364 "??" (r)ecom mencer ou (q)uitter ?"
366 key=inp(2):if chr$(key)="q" then goto frmt
368 if chr$(key)="r" then fdc%(12)=10:call fdc#:goto encore
370 goto 366
372 sek:
374 fdc%(12)=1:fdc%(14)=a:call fdc#:' Tête sur piste de début
376 '
378 fdc%(16)=6400
380 for piste=a to b
382 for record=1 to trk%(3169)
384 def seg=trk%(3170+record):poke trk#,piste+off
386 poke trk#+1,drive-2:next record
388 '
390 fdc%(12)=8:call fdc#:' WRITE-TRACK
392 if fdc%(18)<>&HB0 or fdc%(19)<>3 then "?" *** ERREUR! piste:";piste
394 fdc%(12)=3:call fdc#:' STEP-IN
396 next piste
398 frmt:
400 fdc%(12)=11:fdc%(13)=0:call fdc#:goto formater:' désélectionner
402 end
404 '----- Valeurs pour le format ATARI -----
-
406 data 60,78,12,0,22,78,12,0,40,78,512,229,3,245,3,245,251,1,2,9,78

```

8.3.5 Demo 4 - Conversion de disquettes simple face

en disquettes double face

Le dernier exemple d'application des routines pour le FDC en langage machine servira aux utilisateurs de lecteurs double face.

Si vous voulez copier une disquette formatée en simple face, cela ne pose pas de problèmes, bien entendu. Mais si vous essayez de la copier sur une disquette double face, le DESKTOP vous renverra un message d'erreur. Le message vous indique que les formats de la disquette source et de la disquette objet ne correspondent pas et qu'il est donc impossible d'exécuter la copie. Mais comme possesseur de lecteur double face, vous préférez certainement utiliser la pleine capacité de vos disquettes. Dans ce cas, la seule possibilité qu'il vous reste est de copier un à un tous les programmes de votre disquette simple face.

S'il n'y a que des fichiers sur la disquette source, ce type de copie est parfois plus rapide que la copie de disquette. Mais cela prend beaucoup plus de temps si le nombre de fichiers est important. Il n'est pas rare de trouver sur une telle disquette cinquante fichiers ou plus. La durée de ce type de copie est très grande. Mais vous l'avez déjà compris : la fameuse "tasse de café" si souvent recommandée dans de tels cas vous sera épargnée.

Comme nous l'avons déjà dit, le DESKTOP n'est pas en mesure de copier une disquette complète si la disquette source et la disquette objet n'ont pas le même format. On comprend aisément qu'il ne soit pas possible de copier une disquette double face sur une disquette simple face. Les 720 Ko d'une disquette double face tiendront difficilement sur les 360 Ko d'une disquette simple face ! Par contre, les informations d'une disquette simple face tiennent sans problème sur une disquette double face. On n'a donc pas de problèmes de place mémoire.

Dans ces considérations, il ne faut pas oublier que chaque disquette dispose d'un secteur réservé qui contient des informations importantes (directory, FAT, boot secteur). Vous trouverez plus de détail au chapitre 3. Il faut être assuré qu'après la copie, les informations en question se trouveront bien là où le système d'exploitation les attend.

Si on divise une disquette en secteurs logiques comme le fait le système d'exploitation, on se rend compte que le nombre de ces secteurs diffère sur une disquette simple face et une disquette double face. Une disquette simple face est divisée en 720 secteurs logiques alors qu'une disquette double face en contient 1440. Par chance, les adresses de secteurs logiques de la FAT, du catalogue et du boot secteur sont les mêmes pour chaque format. La place réservée aux informations reste la même.

Le travail du programme se résumera donc à transformer les adresses de secteurs logiques en adresses de secteurs physiques. Voici un petit exemple :

La numérotation des secteurs logiques commence à 0. Le catalogue commence au secteur 11, c'est à dire au 12ième secteur de la disquette. Mais où est le 12ième secteur d'une disquette ? C'est l'adresse de secteur physique qui contient justement cette information. Sur une disquette simple face, le secteur logique numéro 11 est sur la face 0, piste 1, secteur 3. Sur une disquette double face, ce secteur est sur le face 1, piste 0, secteur 3.

Après ce long discours, venons-en au programme. Mais commençons par éclaircir les aspects les plus importants de son déroulement :

1. La disquette simple face est mise dans le lecteur A et la disquette double face dans le lecteur B.
2. Le boot secteur de la disquette objet est mémorisé.
3. Les pistes de la disquette source sont lues les unes à la suite des autres et inscrites sur les pistes de la disquette objet (recto et verso) jusqu'à ce que 80 pistes aient été copiées.
4. Le boot secteur de la disquette objet est recopié.

Après cette conversion, tous les programmes de la disquette simple face sont copiés sur la disquette double face.

```

10  '## Conversion de disquettes simple face en disquettes double face A.
S.##
20  '
30  'Chargement du programme en langage machine
40  dim fdc%(700) : fdc#=varptr(fdc%(0)) : bload "fdcinter.img",fdc#
50  '
60  'Nous avons besoin de 3 tampons et de leurs adresses de départ
70  dim sec0%(2400) : sec0#=varptr(sec0%(0))
80  dim sec1%(2400) : sec1#=varptr(sec1%(0))
90  dim bootsec%(512) : bootsec#=varptr(bootsec%(0))
100 '
110 def seg=0 : ' Nous pokons des mots longs
120 '
130 lirenbre=&H1200 : ' READ_SECTOR => 9 * 512 Byte
140 ecrirebre=&H1220 : ' WRITE_SECTOR => 9 * 512 + 32 Byte
150 '
160 fdc%(15)=1 : ' Nous commençons dans chaque piste avec le secteur 1
170 '
180 '## START #####
190 '
200 start:
210 ? : fullw 2 : clearw 2 : gotoxy 0,1
220 ? " Conversion de disquettes simple face en double face"
230 ?:"?:"? " Mettre disquette source simple face dans lecteur A"
240 ?:"? " Et disquette objet double face dans lecteur B."
250 ?:"?:"? " c => conversion : autre touche => Fin du programme"
260 if chr$(inp(2))<>"c" then end
270 '
280 init:
290 clearw 2 : gotoxy 0,2
300 '
310 '## Lecteur B => Test de Restore et Write-Protect #####
320 '
330 fdc%(12)=11 : fdc%(13)=4 : call fdc#
340 fdc%(12)=0 : call fdc#
350 if fdc%(18) < &HA7 then goto convert
360 '
370 ? " La disquette du lecteur B est protégée. Enlevez";
380 ? " la protection":?
    
```

```

390 ? " s => suite ; autre touche => redémarrage"
400 fdc%(12)=11 : fdc%(13)=0 : call fdc#
410 if chr$(inp(2))="s" then init
420 goto start
430 '
440 convert:
450 ?" Conversion en cours. Patientez ....."
460 '
470 '*** Lecteur B => Lecture du Bootsector *****
***
480 '
490 fdc%(16)=%H200 : fdc%(12)=5 : poke fdc#+56,bootsec# : call fdc#
500 gosub checkstat
510 '
520 '*** Lecteur A => Restore *****
*
530 '
540 fdc%(12)=11 : fdc%(13)=2 : call fdc#
550 fdc%(12)=0 : call fdc#
560 '
570 '*** Copie des pistes 0-79 (Lecteur A) après pistes 0-39 (Lecteur B)
*****
580 '
590 trackcnta=0 : trackcntb=0 : ' Compteur de pistes pour lecteur A et lec
teur B
600 loop:
610 '
620 '*** Lecteur A => lire piste X (X=trackcnta) *****
*
630 '
640 fdc%(12)=11 : fdc%(13)=2 : call fdc# : ' Sélection lecteur A/B
650 fdc%(12)=15 : fdc%(14)=trackcnta : call fdc# : ' Ecrire reg. piste
660 fdc%(16)=lirenbre
670 fdc%(12)=5 : poke fdc#+56,sec0# : call fdc# : ' READ_SECTOR
680 gosub checkstat
690 fdc%(12)=3 : call fdc# : ' STEP_IN
700 '
710 '*** Lecteur A => lire piste X+1 *****
**
720 '

```

```

730 fdc%(12)=5 : poke fdc#+56,sec1# : call fdc# : ' READ_SECTOR
740 gosub checkstat
750 fdc%(12)=3 : call fdc# : ' STEP_IN
760 trackcnta=trackcnta+2
770 '
780 '*** Lecteur B => Ecrire piste X sur face 0 *****
790 '
800 fdc%(12)=11 : fdc%(13)=4 : call fdc# : ' Sélection lecteur B/0
810 fdc%(12)=15 : fdc%(14)=trackcntb : call fdc# : ' Ecrire reg. piste
820 fdc%(16)=ecrirenbre
830 fdc%(12)=6 : poke fdc#+56,sec0# : call fdc# : ' WRITE_SECTOR
840 gosub checkstat
850 '
860 '*** Lecteur B => Ecrire piste x+1 sur face 1 *****
870 '
880 fdc%(12)=11 : fdc%(13)=5 : call fdc# : ' Sélection lecteur B/1
890 fdc%(12)=6 : poke fdc#+56,sec1# : call fdc# : ' WRITE_SECTOR
900 gosub checkstat
910 fdc%(12)=3 : call fdc# : ' STEP_IN
920 trackcntb=trackcntb+1
930 '
940 '*** Teste si on a déjà écrit 80 pistes *****
950 '
960 if trackcntb<40 then goto loop
970 '
980 '*** Lecteur B => Recopier Restore et Boot-Sector *****
990 '
1000 fdc%(12)=11 : fdc%(13)=4 : call fdc# : ' Sélection lecteur B/0
1010 fdc%(12)=0 : call fdc# : ' RESTORE
1020 fdc%(16)=&H220
1030 fdc%(12)=6 : poke fdc#+56,bootsec# : call fdc# : ' WRITE_SECTOR
1040 gosub checkstat
1050 '
1060 '*** Lecteur A => Restore et "désélection" *****
1070 '
1080 fdc%(12)=11 : fdc%(13)=2 : call fdc# : ' Sélection lecteur A/0
1090 fdc%(12)=0 : call fdc# : ' RESTORE
1100 fdc%(12)=11 : fdc%(13)=0 : call fdc# : ' désélection
1110 '
1120 '*** FERTIG *****
    
```

```
1130 '
1140 ??:? " Fini ! .....(r)estart ou (f)in ?"
1150 if chr$(inp(2))<>"r" then end
1160 goto start
1170 '
1180 '## Test état #####
1190 '
1200 checkstat:
1210 if fdc%(18)=&HB0 and fdc%(19)=3 and fdc%(20)=0 then return
1220 gotoxy 0,7 : ? " FDC-STATUS :$";hex$(fdc%(18))
1230 ? " DMA-STATUS :$";hex$(fdc%(19))
1240 ? " #DMA-BYTES :$";hex$(fdc%(21))
1250 ? " TIMEOUT :$";hex$(fdc%(20)):?
1260 '
1270 ? " Le processus de copie a été interrompu en raison d'une erreur."
1280 ??:? " Pressez une touche..."
1290 fdc%(12)=11 : fdc%(13)=0 : call fdc#
1300 key=inp(2) : goto start
1310 '
1320 '##### FIN #####
```


8.4 CREATION DE CHARGEURS BASIC

Ce chapitre n'a pas grand chose à voir avec le lecteur. Il s'agit plutôt d'un outil simple mais très pratique qui peut avoir de nombreuses applications. Les fanas d'assembleur qui ont une parade en langage machine à tous les problèmes se sentiront plus particulièrement visés. Il n'est pas toujours indispensable d'offrir un programme du domaine public pour faire plaisir. Parfois de petites choses peuvent être très appréciées. Mais lisez plutôt.

De nombreux problèmes de programmation peuvent être résolus par un programme en BASIC (confortablement et rapidement). Mais on atteint facilement les limites de ce langage lorsqu'on a besoin de vitesse élevée. Ce sont souvent de petites choses qui empêchent certains d'appliquer leurs idées en BASIC. S'il faut réaliser un graphisme animé pour un jeu par exemple, le programmeur BASIC doit passer la main. Bien sûr, le BASIC permet de gérer facilement des fichiers. Mais s'il faut trier de grandes masses de données, c'est autre chose.

Venons-en donc à notre propos. Certains programmeurs en assembleurs créent de petites routines en langage machine à la chaîne. Ces personnes créent même parfois des programmes en BASIC. Ils ne semblent pas rencontrer les problèmes de vitesse du programmeur BASIC. En fait, ils remplacent purement et simplement les cas où l'interpréteur BASIC serait trop lent par des programmes en langage machine. La relation entre les deux langages est très simple : un CALL suffit.

Ce serait formidable si ces routines pouvaient être utilisées par ceux qui n'ont pas d'assembleur. Cela ne tient pas tellement au fait qu'on n'a jamais les routines en langage machine sans contrepartie. Mais ce qui gêne le plus est qu'on doit écrire un programme en BASIC qui contient les données de la routine en langage machine et qui soit les "poke" en mémoire soit crée un fichier sur disquette (pour un BLOAD). Mais pourquoi devrait-on effectuer ce travail pour les autres ? Cela ne nous sert à rien puisque nous avons déjà le fichier en langage machine créé avec l'assembleur !

Par contre, il serait très intéressant d'avoir un programme qui crée les DATAs à partir d'une routine assemblée.

C'est ce que nous proposons aux programmeurs BASIC. Nous sommes certains qu'ils seront satisfaits par une solution **READY TO HACK IN**.

Le programme en BASIC effectue une conversion **ASSEMBLEUR-BASIC-ASSEMBLEUR**. A la mise en route, le programme demande le nom du fichier à convertir. Ensuite, il faut entrer deux autres noms de fichiers.

1. Le nom du chargeur BASIC à créer. Ce programme se trouve le plus souvent sous la forme d'un listing. Etant donné que la saisie de DATA se fait rarement sans erreur, il intègre un checksum.
2. Le nom du fichier que doit créer le chargeur BASIC. Le chargeur teste si la checksum des DATA est valable. Si c'est le cas, il écrit un fichier sur disquette qui est le même que celui qu'on a converti.
3. Il faut noter que les datas se trouvent au début du programme créé. Cela n'a pas d'incidence sur le fonctionnement du programme.

```

10 '#####
20 '##### Data-Maker A.S. 10/B6 #####
30 '#####
40 '
50 'Crée un programme en BASIC à partir de n'importe quel fichier
60 'Si on le met en route par 'RUN' par la suite, il écrit un fichier
70 'identique au précédent.
80 '
90 '#####
100 '
110 ? :fullw 2:clearw 2:gotoxy 0,0
120 input "A partir de quel fichier faut-il créer les DATAs "; prg$
130 '
140 open "R", #1, prg$, 1: bytes = lof(1): close: champlen = cint(bytes/2-1)
150 ? : ? "INFO --> La longueur de >> "; prg$; " << est de "; bytes; " Byte"
160 ? : ? : ?
170 '
180 input "Quel doit être le nom du fichier-objet "; bas$ : ? : ?
190 '
200 ? "Un chargeur sera intégré dans le fichier >> "; bas$; " << "
210 ? : ? "Un fichier correspondant au fichier de saisie"
220 ? : ? ">> "; prg$; " << sera créé à partir des DATAs.": ? : ?
230 input "Entrez un nom ici aussi "; make$
240 ? : ? "#####"
250 ? : ? "C'est parti ! Afin de ne pas vous ennuyer, vous avez des infos"
260 ? "sur les processus"
270 ? : ? : ? "Un vecteur d'Integer (c%) est dimensionné avec "; champlen; " "
280 dim c%(champlen)
290 ?
300 ? "Le fichier d'entrée >> "; prg$; " << est chargé après varptr(c%(0)) "
310 bload prg$, varptr(c%(0))
320 '
330 ? : ? "Le fichier >> "; bas$; " << est ouvert"
340 open "O", 1, bas$
350 '
360 ? : ? : ? "Le contenu du vecteur C% est écrit dans ce fichier sous"
370 ? "forme de lignes de DATAs."
380 ? : ? : ? "Soyez patient....."
390 '
400 check# = 0: z = 0: z1 = 100

```

```

410 '
420 if z mod 8 = 0 then print #1:print #1,str$(z1);" DATA ";z1=z1+1
430 '
440 print #1,right$("0000"+hex$(c%(z)),4);
450 '
460 check#=check#+c%(z):z=z+1
470 if z=champlen+1 then 510
480 if z mod 8 <> 0 then print #1," ";
490 goto 420
500 '
510 "?:?:?"Voici enfin le programme de chargement à ajouter ...."
520 '
530 print #1
540 print #1,str$(10);" ***** File-Maker A.S. *****"
550 print #1,str$(15);" "
560 print #1,str$(20);" ?:fullw 2:clearw 2:gotow 0,0"
570 print #1,str$(25);" ? ";chr$(34);"Le fichier >> ";make$;
580 print #1," << est crée";chr$(34);":?:?:?"
590 '
600 print #1,str$(30);" dim c%(";str$(champlen);"):cs#=0"
610 '
620 print #1,str$(35);" for i=0 to ";str$(champlen)
630 '
640 print #1,str$(40);" read a$:c%(i)=val(";chr$(34);"%H";chr$(34);"+a$)"
650 '
660 print #1,str$(45);" check#=check#+(c%(i))"
670 '
680 print #1,str$(50);" next i"
690 '
700 print #1,str$(55);" if check#=";str$(check#);" then ";str$(70)
710 '
720 print #1,str$(60);" ?";chr$(34);"Ca ne va pas encore, ";
730 print #1,"car quelque chose ne convient pas dans les DATAs.";chr$(34)
740 print #1,str$(65);" goto 80"
750 '
760 print #1,str$(70);" bsave ";chr$(34);make$;chr$(34);",varptr(c%(0)),
;
770 print #1,str$(bytes)
780 '
790 print #1,str$(75);" ? ";chr$(34);"Le programme >> ";make$;

```

```

800 print #1," << est écrit.";chr$(34)
810 '
820 print #1,str$(80);" ??:?:?";chr$(34);"Pressez une touche";chr$(34);
830 print #1,"a=inp(2):end"
840 print #1,str$(85);" '"
850 print #1,str$(90);" '##### DATA's pour ";make$;" #####"
860 print #1,str$(95);" '"
870 '
880 ??:?"...Fermeture du fichier objet et ...
890 close #1
900 '
910 ??:?"Le programme >> ";bas$;" >> est fini."
920 ??:?"Pressez une touche":a=inp(2):end

```

ANNEXE 1.**Chargeur BASIC de editor.tos**

```

10 '***** File-Maker A.S. *****
15 '
20 ? :fullw 2:clearw 2:gotox y 0,0
25 ? "Le fichier >> editor.tos << est crée":? :? :?
30 dim c%( B437):cs#=0
35 for i=0 to B437
40 read a$:c%(i)=val("&H"+a$)
45 check#=check#+(c%(i))
50 next i
55 if check#= 81149235.2 then 70
60 ? "Ca ne va pas encore, car quelque chose ne convient pas dans les DATAs
."
65 goto 80
70 bsave "editor.tos",varptr(c%(0)), 16876
75 ? "Le programme >> editor.tos << est écrit."
80 ? :? :? :?"Pressez une touche":a=inp(2):end
85 '
90 '***** DATA's pour editor.tos *****
95 '
100 DATA 601A,0000,2FF6,0000,0DCE,0000,9D28,0000
101 DATA 0000,0000,0000,0000,0000,0000,2A4F,2A6D
102 DATA 0004,202D,000C,D0AD,0014,D0AD,001C,D0BC
103 DATA 0000,1100,220D,D280,C2BC,FFFF,FFFE,2E41
104 DATA 2F00,2F0D,3F00,3F3C,004A,4E41,DFFC,0000
105 DATA 000C,3F3C,0003,3F3C,000B,3F3C,0023,4E4E
106 DATA 5C8F,6108,2F3C,0000,0000,4E41,6100,2A1A
107 DATA 6100,2EAC,6100,2DD6,610A,6100,01F4,6100
108 DATA 00EA,4E75,6100,2E0B,6100,2DC2,33FC,0000
109 DATA 0000,3DD4,33FC,0000,0000,3DD8,33FC,0000
110 DATA 0000,3DDA,33FC,0001,0000,3DD6,303C,0000
111 DATA 33FC,0006,0000,3DE2,33FC,0001,0000,3DE4
112 DATA 33FC,004F,0000,3DDE,33FC,0009,0000,3DE0
113 DATA 33FC,0009,0000,3E4C,13FC,0030,0000,32C1

```

114 DATA 13FC,0039,0000,32C2,33FC,05DC,0000,3DE6
115 DATA 23FC,0000,AC0C,0000,3E52,6102,4E75,6100
116 DATA 2E1E,33FC,0014,0000,3DF4,33FC,000A,0000
117 DATA 3DF6,6100,2DAE,207C,0000,3083,6100,293C
118 DATA 33FC,0014,0000,3DF4,33FC,000C,0000,3DF6
119 DATA 6100,2D90,207C,0000,3085,6100,291E,33FC
120 DATA 0014,0000,3DF4,33FC,000E,0000,3DF6,6100
121 DATA 2D72,207C,0000,30E7,6100,2900,6100,2DF2
122 DATA 6100,2CEA,6100,2DB8,4E75,6100,2D90,4AB0
123 DATA 67F8,4840,B03C,0044,672A,B03C,004B,6604
124 DATA 6140,60E6,B03C,004D,6604,6158,60DC,B03C
125 DATA 0050,6604,611E,60D2,B03C,004B,6602,6106
126 DATA 60CB,508F,4E75,23F9,0000,3E0A,0000,3DD0
127 DATA 615A,4E75,23F9,0000,3E0E,0000,3DD0,614C
128 DATA 4E75,2039,0000,3DCC,53B0,670B,23C0,0000
129 DATA 3DCC,600A,23F9,0000,3DCB,0000,3DCC,6100
130 DATA 2A3C,4E75,2039,0000,3DCC,5280,B0B9,0000
131 DATA 3DCB,6E08,23C0,0000,3DCC,600A,23FC,0000
132 DATA 0001,0000,3DCC,6100,2A14,4E75,6100,2D10
133 DATA 2079,0000,3DD0,2039,0000,3DCC,53B0,E58B
134 DATA 2270,0800,4ED1,33FC,000A,0000,3DF4,33FC
135 DATA 0002,0000,3DF6,6100,2C8A,6100,2C04,302F
136 DATA 0004,4440,B07C,001D,6D04,303C,001D,E54B
137 DATA 227C,0000,3BA0,2071,0000,6100,27FE,6100
138 DATA 2CF0,6100,2BDC,6100,2C46,205F,54BF,4ED0
139 DATA 6100,2BDA,23FC,0000,0007,0000,3DCB,23FC
140 DATA 0000,0001,0000,3DCC,23FC,0000,3012,0000
141 DATA 3DC4,23FC,0000,2FF6,0000,3E0A,23FC,0000
142 DATA 2FF6,0000,3E0E,6100,2974,4E75,6100,2B9E
143 DATA 23FC,0000,3296,0000,3DC4,23FC,0000,000B
144 DATA 0000,3DCB,23FC,0000,0005,0000,3DCC,23FC
145 DATA 0000,3256,0000,3E0A,23FC,0000,3276,0000
146 DATA 3E0E,6100,2938,6100,2BB2,207C,0000,32E7
147 DATA 6100,2768,4E75,23FC,0000,0006,0000,3DCB
148 DATA 23FC,0000,0004,0000,3DCC,23FC,0000,3352
149 DATA 0000,3E0A,23FC,0000,336A,0000,3E0E,23FC
150 DATA 0000,3382,0000,3DC4,6100,2BF2,6100,2B6C
151 DATA 207C,0000,32FA,6100,2722,4E75,6100,2B0E
152 DATA 23FC,0000,315A,0000,3DC4,23FC,0000,311A
153 DATA 0000,3E0A,23FC,0000,313A,0000,3E0E,23FC

154 DATA 0000,AC0C,0000,3E52,23FC,0000,0008,0000
155 DATA 3DCB,23FC,0000,0005,0000,3DCC,6100,2B9E
156 DATA 6100,2B1B,207C,0000,321B,6100,26CE,4E75
157 DATA 6100,092A,6100,098C,23FC,0000,0008,0000
158 DATA 3DCB,23FC,0000,0003,0000,3DCC,23FC,0000
159 DATA 33FA,0000,3DC4,23FC,0000,33BA,0000,3E0A
160 DATA 23FC,0000,33DA,0000,3E0E,6100,2ACE,207C
161 DATA 0000,3447,6100,26B4,6100,2842,4E75,6100
162 DATA 2A6C,23FC,0000,3772,0000,3DC4,23FC,0000
163 DATA 000B,0000,3DCB,23FC,0000,0003,0000,3DCC
164 DATA 23FC,0000,3732,0000,3E0A,23FC,0000,3752
165 DATA 0000,3E0E,6100,2806,6100,2AB0,207C,0000
166 DATA 3AB0,6100,2636,4E75,6100,2A22,23FC,0000
167 DATA 3A44,0000,3DC4,23FC,0000,0007,0000,3DCB
168 DATA 23FC,0000,0001,0000,3DCC,23FC,0000,3A0C
169 DATA 0000,3E0A,23FC,0000,3A2B,0000,3E0E,6100
170 DATA 27BC,6100,2A36,207C,0000,3ACA,6100,25EC
171 DATA 4E75,23FC,0000,0006,0000,3DCB,23FC,0000
172 DATA 0004,0000,3DCC,23FC,0000,3BAC,0000,3E0A
173 DATA 23FC,0000,3BC4,0000,3E0E,23FC,0000,3BDC
174 DATA 0000,3DC4,6100,2776,6100,29F0,207C,0000
175 DATA 393A,6100,25A6,4E75,3039,0000,3DDA,B079
176 DATA 0000,3DE2,6D06,303C,0000,6002,5240,33C0
177 DATA 0000,3DDA,D03C,0030,13C0,0000,31B2,6100
178 DATA 273C,4E75,3039,0000,3DDA,B07C,0000,6F04
179 DATA 5340,6006,3039,0000,3DE2,33C0,0000,3DDA
180 DATA D03C,0030,13C0,0000,31B2,6100,2710,4E75
181 DATA 3039,0000,3DDB,B07C,0001,6D06,303C,0000
182 DATA 6004,303C,0001,33C0,0000,3DD8,D03C,0030
183 DATA 13C0,0000,31BC,6100,26E4,4E75,3039,0000
184 DATA 3DD8,B07C,0000,6F06,303C,0000,6004,303C
185 DATA 0001,33C0,0000,3DD8,D03C,0030,13C0,0000
186 DATA 31BC,6100,26B8,4E75,3039,0000,3DD4,B079
187 DATA 0000,3DDE,6D06,303C,0000,6002,5240,33C0
188 DATA 0000,3DD4,4BC0,80FC,000A,D03C,0030,13C0
189 DATA 0000,3197,4B40,D03C,0030,13C0,0000,319B
190 DATA 6100,267A,4E75,3039,0000,3DD4,B07C,0000
191 DATA 6F04,5340,6006,3039,0000,3DDE,33C0,0000
192 DATA 3DD4,4BC0,80FC,000A,D03C,0030,13C0,0000
193 DATA 3197,4B40,D03C,0030,13C0,0000,319B,6100

194 DATA 263C,4E75,3039,0000,3DD6,B079,0000,3DE0
 195 DATA 6D06,303C,0000,6002,5240,33C0,0000,3DD6
 196 DATA 48C0,80FC,000A,D03C,0030,13C0,0000,31A4
 197 DATA 4840,D03C,0030,13C0,0000,31A5,6100,25FE
 198 DATA 4E75,3039,0000,3DD6,B07C,0000,6F04,5340
 199 DATA 6006,3039,0000,3DE0,33C0,0000,3DD6,48C0
 200 DATA 80FC,000A,D03C,0030,13C0,0000,31A4,4840
 201 DATA D03C,0030,13C0,0000,31A5,6100,25C0,4E75
 202 DATA 3039,0000,3B24,323C,0001,B07C,0400,6604
 203 DATA 323C,0002,3F01,3F39,0000,3DD8,3F39,0000
 204 DATA 3DD4,3F39,0000,3DD6,3F39,0000,3DDA,42A7
 205 DATA 2F3C,0000,AC0C,3F3C,000B,4E4E,DFFC,0000
 206 DATA 0014,4A40,6B04,6118,4E75,3F00,6100,FB7B
 207 DATA 6100,27E8,207C,0000,321B,6100,239E,4E75
 208 DATA 33FC,0000,0000,3DFA,23F9,0000,3E52,0000
 209 DATA 3DEB,33FC,001F,0000,3E06,33FC,0012,0000
 210 DATA 3E04,33FC,0000,0000,3E96,33FC,00D0,0000
 211 DATA 3E98,3039,0000,3B24,B07C,0400,6618,33FC
 212 DATA 0200,0000,3E96,33FC,02D0,0000,3E9B,33FC
 213 DATA 003F,0000,3E06,6102,4E75,6100,2792,6100
 214 DATA 27FE,33FC,0000,0000,3DFA,33F9,0000,3DFA
 215 DATA 0000,3DF8,6100,254B,6100,27E4,6100,2770
 216 DATA 6100,27BA,4B40,B03C,0019,6700,00AA,B03C
 217 DATA 004B,6724,B03C,0050,675C,B03C,001C,6712
 218 DATA B03C,004B,670C,B03C,004D,66D4,6100,FA76
 219 DATA 6004,6100,FA4E,4E75,3039,0000,3DFA,B07C
 220 DATA 0000,672E,B079,0000,3E9B,6714,0479,0100
 221 DATA 0000,3DFA,04B9,0000,0100,0000,3DEB,6012
 222 DATA 0479,00D0,0000,3DFA,04B9,0000,00D0,0000
 223 DATA 3DEB,6000,FF76,3039,0000,3DFA,B079,0000
 224 DATA 3E9B,672E,B079,0000,3E96,6614,0679,00D0
 225 DATA 0000,3DFA,06B9,0000,00D0,0000,3DEB,6012
 226 DATA 0679,0100,0000,3DFA,06B9,0000,0100,0000
 227 DATA 3DEB,6000,FF36,33FC,0000,0000,39FC,48F9
 228 DATA 38FB,0000,3E56,2A7C,0000,317A,3E3C,002D
 229 DATA 101D,3F00,6100,2720,51CF,FFF6,2A7C,0000
 230 DATA 341A,3E3C,000D,101D,3F00,6100,270A,51CF
 231 DATA FFF6,6100,2630,6100,262C,2B79,0000,3DEB
 232 DATA 2A4C,33F9,0000,3DFA,0000,3DF8,363C,000F
 233 DATA 3B03,3A39,0000,3E06,3604,6100,24C4,6100

234 DATA 24E4,3604,2B4D,3E3C,0005,3F3C,0020,6100
235 DATA 26C6,51CF,FFF6,6100,24F2,DBFC,0000,0010
236 DATA 0679,0010,0000,3DFB,6100,25DA,51CD,FFCA
237 DATA 6100,26BC,4CF9,38FB,0000,3E56,33FC,0002
238 DATA 0000,39FC,6000,FE9A,4E75,6100,25EE,207C
239 DATA 0000,3230,6100,21A4,33FC,0000,0000,3DF4
240 DATA 33FC,0004,0000,3DF6,6100,25F8,6100,25A8
241 DATA 3039,0000,3B24,B07C,0400,6612,33FC,0200
242 DATA 0000,3E96,33FC,02D0,0000,3E9B,6010,33FC
243 DATA 0000,0000,3E96,33FC,00D0,0000,3E98,33FC
244 DATA 0012,0000,3E04,23FC,0000,AC0C,0000,3E52
245 DATA 6134,6100,F8AE,6100,F8AA,33FC,0002,0000
246 DATA 3DF6,6100,259E,6100,251B,6100,256E,207C
247 DATA 0000,321B,6100,2124,6100,2554,6100,FD82
248 DATA 6100,25DC,4E75,4BE7,1F1E,23F9,0000,3E52
249 DATA 0000,3DE8,33FC,0000,0000,3DFA,33FC,0000
250 DATA 0000,3DF8,6100,231B,6100,25B4,33FC,0007
251 DATA 0000,3DF4,33FC,0004,0000,3DF6,6100,2544
252 DATA 6100,2500,2F3C,0000,3E08,6100,25CE,6100
253 DATA 24FE,4A79,0000,3E08,6B4C,3039,0000,3DF6
254 DATA 5940,E948,3439,0000,3DF4,5F42,4BC2,84FC
255 DATA 0003,D042,3239,0000,3E0B,2679,0000,3DEB
256 DATA 1781,0000,6100,23D4,0C79,0034,0000,3DF4
257 DATA 6D08,33FC,0004,0000,3DF4,5679,0000,3DF4
258 DATA 6100,24E0,609A,2039,0000,3E9E,4B40,B03C
259 DATA 004B,673A,B03C,004D,675A,B03C,0050,6700
260 DATA 007A,B03C,004B,6700,011C,B03C,0052,6700
261 DATA 01BC,B03C,0072,6700,01B4,B03C,001C,6700
262 DATA 01AC,6100,2376,6100,249A,6000,FF54,3039
263 DATA 0000,3DF4,B07C,0007,6E0B,33FC,0037,0000
264 DATA 3DF4,5779,0000,3DF4,6100,247B,6100,24D0
265 DATA 6000,FF2E,3039,0000,3DF4,B07C,0034,6D08
266 DATA 33FC,0004,0000,3DF4,5679,0000,3DF4,6100
267 DATA 2452,6100,24AA,6000,FF0B,6100,2412,3039
268 DATA 0000,3DF6,B07C,0016,6D00,00BB,3039,0000
269 DATA 3DFA,B079,0000,3E96,6638,0679,00D0,0000
270 DATA 3DFA,06B9,0000,00D0,0000,3DE8,33F9,0000
271 DATA 3DF4,0000,3E02,6100,21C6,33F9,0000,3E02
272 DATA 0000,3DF4,33FC,0005,0000,3DF6,6100,23F4
273 DATA 604A,B079,0000,3E98,6742,0679,0100,0000

274 DATA 3DFA,06B9,0000,0100,0000,3DEB,33F9,0000
 275 DATA 3DF4,0000,3E02,6100,21B6,33F9,0000,3E02
 276 DATA 0000,3DF4,33FC,0006,0000,3DF6,6100,23B4
 277 DATA 600A,5279,0000,3DF6,6100,23AB,6100,2400
 278 DATA 6000,FE5E,6100,2368,3039,0000,3DF6,B07C
 279 DATA 0004,6600,00E6,3039,0000,3DFA,B07C,0000
 280 DATA 6700,00E2,B079,0000,3E98,673B,0479,0100
 281 DATA 0000,3DFA,04B9,0000,0100,0000,3DEB,33F9
 282 DATA 0000,3DF4,0000,3E02,6100,2114,33F9,0000
 283 DATA 3E02,0000,3DF4,33FC,0013,0000,3DF6,6100
 284 DATA 2342,6040,0479,0000,0000,3DFA,04B9,0000
 285 DATA 00D0,0000,3DEB,33F9,0000,3DF4,0000,3E02
 286 DATA 6100,20DC,33F9,0000,3E02,0000,3DF4,33FC
 287 DATA 0013,0000,3DF6,6100,230A,5379,0000,3DF6
 288 DATA 6100,2300,6100,235B,6000,FDB6,33FC,0000
 289 DATA 0000,3DF4,33FC,0004,0000,3DF6,6100,22E4
 290 DATA 4CDF,78FB,4E75,4E7,1F1E,33FC,0000,0000
 291 DATA 3DF4,33FC,0002,0000,3DF6,6100,22C6,207C
 292 DATA 0000,31CD,6100,1E54,267C,0000,317A,363C
 293 DATA 002D,101B,3F00,6100,231E,51CB,FFF6,207C
 294 DATA 0000,31E9,6100,1E34,6100,22F4,6100,2322
 295 DATA B03C,0079,6706,B03C,0059,6660,3039,0000
 296 DATA 3B24,B07C,0400,6700,0440,323C,0001,3F01
 297 DATA 3F39,0000,3DD8,3F39,0000,3DD4,3F39,0000
 298 DATA 3DD6,3F39,0000,3DDA,42A7,2F3C,0000,AC0C
 299 DATA 3F3C,0009,4E4E,DFFC,0000,0014,4A40,6B34
 300 DATA 6100,21BE,6100,2298,6100,2210,207C,0000
 301 DATA 321B,6100,1DC6,4CDF,78FB,4E75,6100,21A2
 302 DATA 207C,0000,31FB,6100,1DB2,6100,2272,6100
 303 DATA 22A0,60CC,3F00,6100,F56E,60C4,3039,0000
 304 DATA 3DDA,3F39,0000,3DDA,3F3C,0007,4E4D,588F
 305 DATA 4AB0,6608,3F00,6100,F54E,6044,2040,33DB
 306 DATA 0000,3E26,33DB,0000,3E28,33DB,0000,3E2A
 307 DATA 33DB,0000,3E2C,33DB,0000,3E2E,33DB,0000
 308 DATA 3E30,33DB,0000,3E32,33DB,0000,3E34,33DB
 309 DATA 0000,3E36,33DB,0000,3E40,33DB,0000,3E40
 310 DATA 4E73,3F39,0000,3DDA,3F39,0000,3E30,3F39
 311 DATA 0000,3E2E,2F3C,0000,5DEC,3F3C,0002,3F3C
 312 DATA 0004,4E4D,DFFC,0000,000E,4A40,6B02,4E75
 313 DATA 3F00,6100,F4D2,60F6,3F39,0000,3DDA,3039

314 DATA 9000,3E2E,E348,5240,3F00,3F39,0000,3E2C
315 DATA 2F3C,0000,3EAC,3F3C,0002,3F3C,0004,4E4D
316 DATA 0FFC,0000,000E,4A40,5B02,4E75,3F00,6100
317 DATA F496,60F6,3039,0000,3DDE,B07C,0063,6D06
318 DATA 303C,0000,6002,5240,33C0,0000,3DDE,48C0
319 DATA 80FC,000A,D03C,0030,13C0,0000,3900,4B40
320 DATA D03C,0030,13C0,0000,3901,6100,1E50,4E75
321 DATA 3039,0000,3DDE,B07C,0000,6F04,5340,6004
322 DATA 303C,0063,33C0,0000,3DDE,43E0,80FC,000A
323 DATA D03C,0030,13C0,0000,3900,4B40,D03C,0030
324 DATA 13C0,0000,3901,6100,1E14,4E75,3039,0000
325 DATA 3DE0,B07C,0063,6D06,303C,0000,6002,5240
326 DATA 33C0,0000,3DE0,48C0,80FC,000A,D03C,0030
327 DATA 13C0,0000,3911,4B40,803C,0030,13C0,0000
328 DATA 3912,6100,1DD8,4E75,3039,0000,3DE0,B07C
329 DATA 0000,6F04,5340,6004,303C,0063,33C0,0000
330 DATA 3DE0,48C0,80FC,000A,D03C,0030,13C0,0000
331 DATA 3911,4B40,D03C,0030,13C0,0000,3912,6100
332 DATA 1B9C,4E75,6100,FE36,6100,FE93,6100,FECA
333 DATA 6106,6100,F32E,4E75,33FC,0004,0000,3DF6
334 DATA 33FC,000A,0000,3DF4,6100,2018,207C,0000
335 DATA 3953,6100,1BA6,33FC,002A,0000,3E42,33FC
336 DATA 0006,0000,3DF6,33FC,000E,0000,3DF4,6100
337 DATA 1FF2,207C,0000,3597,6100,1B80,6100,2010
338 DATA 3F39,0000,3E26,6100,1C12,5279,0000,3DF6
339 DATA 33FC,000C,0000,3DF4,6100,1FCE,207C,0000
340 DATA 359A,6100,1B56,6100,1FE6,3F39,0000,3E28
341 DATA 6100,1BE8,5279,0000,3DF6,33FC,000C,0000
342 DATA 3DF4,6100,1F9E,207C,0000,35AE,6100,1B2C
343 DATA 6100,1FEC,3F39,0000,3E2A,6100,1B8E,5279
344 DATA 0000,3DF6,33FC,000C,0000,3DF4,6100,1F74
345 DATA 207C,0000,35C1,6100,1B02,6100,1F92,3F39
346 DATA 0000,3E2C,6100,1B74,5279,0000,3DF6,33FC
347 DATA 000C,0000,3DF4,6100,1F4A,207C,0000,35D7
348 DATA 6100,1AD8,6100,1F68,3F39,0000,3E2E,6100
349 DATA 1B6A,5279,0000,3DF6,33FC,000C,0000,3DF4
350 DATA 6100,1F20,207C,0000,35E7,6100,1AAE,6100
351 DATA 1F3E,3F39,000C,3E30,6100,1B40,5279,0000
352 DATA 3DF6,33FC,000C,0000,3DF4,6100,1EF6,207C
353 DATA 0000,3603,6100,1AE4,6100,1F14,3F39,0000

354 DATA 3E32,6100,1B16,5279,0000,3DF6,33FC,000C
 355 DATA 0000,3DF4,6100,1EEC,207C,0000,3623,6100
 356 DATA 1A5A,6100,1EEA,3F39,0000,3E34,6100,1AEC
 357 DATA 5279,0000,3DF6,33FC,000C,0000,3DF4,6100
 358 DATA 1E42,207C,0000,363A,6100,1A30,6100,1EC0
 359 DATA 3F39,0000,3E40,6100,1AC2,5479,0000,3DF6
 360 DATA 33FC,000A,0000,3DF4,6100,1E78,207C,0000
 361 DATA 364E,3039,0000,3E40,B07C,0002,6606,207C
 362 DATA 0000,3680,6100,19F4,6100,1EB4,6100,1EE2
 363 DATA 6100,1E28,6100,1DCA,6100,1E20,207C,0000
 364 DATA 393A,6100,19D6,4E75,6100,04AA,50F9,0000
 365 DATA 043E,6100,0544,6100,04E2,6100,0688,6150
 366 DATA 51F9,0000,043E,6100,1E76,6100,1DEE,6100
 367 DATA 1D90,6100,04CC,6100,04A0,207C,0000,3AF0
 368 DATA 6100,1998,6100,1E8A,6100,046A,6100,0550
 369 DATA 6100,04B6,6100,1DC4,6100,1D66,6100,1DBC
 370 DATA 207C,0000,3910,6100,1972,4C0F,78F9,4E75
 371 DATA 6100,0630,33FC,0190,00FF,8606,33FC,0090
 372 DATA 00FF,8606,33FC,0190,00FF,8606,3C3C,0004
 373 DATA 6100,04A2,33FC,0184,00FF,8606,3E39,0000
 374 DATA 3DD6,6100,0490,33FC,0180,00FF,8606,3C3C
 375 DATA 00A0,6100,0480,2E3C,0005,0000,0B39,0005
 376 DATA 00FF,FA01,6716,5387,66F2,3F3C,FFF7,6100
 377 DATA F0C6,6100,1D46,6100,1CEB,4E75,6100,0466
 378 DATA 3039,0000,3A0A,0200,0006,6602,4E75,3F3C
 379 DATA FFF8,6100,F0B2,6100,1D22,6100,1CC4,4E75
 380 DATA 303C,0200,C0F9,0000,3E4C,33C0,0000,3E20
 381 DATA 3F39,0000,3E4C,3F39,0000,3DD8,3F39,0000
 382 DATA 3DD4,3F3C,0001,3F39,0000,3DDA,42A7,2F3C
 383 DATA 0000,AC0C,3F3C,0008,4E4E,0FFC,0000,0014
 384 DATA 4A4C,6B06,6100,0136,4E75,3F00,6100,F058
 385 DATA 6100,1D4C,6100,1D7A,6100,1EC0,207C,0000
 386 DATA 32E7,6100,1876,6100,1C4C,600C,3039,0000
 387 DATA 3E4C,B079,0000,3DE0,6E06,303C,0000,6002
 388 DATA 5240,33C0,0000,3E4C,4B00,80FC,000A,D03C
 389 DATA 0030,13C0,0000,32C1,4B40,D03C,0030,13C0
 390 DATA 0000,32C2,6100,19F6,4E75,3039,0000,3E4C
 391 DATA B07C,0000,6F04,5340,6006,3039,0000,3DE0
 392 DATA 33C0,0000,3E4C,4B00,80FC,000A,D03C,0030
 393 DATA 13C0,0000,32C1,4B40,D03C,0030,13C0,0000

394 DATA 32C2,6100,198B,4E75,33FC,0000,0000,3E96
395 DATA 33FC,00D0,0000,3E98,33FC,0012,0000,3E04
396 DATA 2039,0000,3DEB,90BC,0000,AC0C,80FC,0200
397 DATA 4B40,4A40,670A,04B9,0000,0100,0000,3DEB
398 DATA 23F7,0000,3DEB,0000,3E52,33FC,0000,0000
399 DATA 3DFA,33FC,0014,0000,3DF4,33FC,0002,0000
400 DATA 3DF6,6100,1BFE,207C,0000,3230,6100,178C
401 DATA 6100,F674,6100,1BC4,6100,1B66,6100,1BBC
402 DATA 207C,0000,32E7,6100,1772,6100,EED6,6100
403 DATA EED2,6100,1BBA,6100,1B7E,4E75,33FC,0000
404 DATA 0000,3DFA,23FC,0000,AC0C,0000,3DEB,33FC
405 DATA 0000,0000,3E3E,33FC,000F,0000,3E04,33FC
406 DATA 0002,0000,3DF6,33FC,003E,0000,3DF4,6100
407 DATA 1B92,207C,0000,3315,6100,1720,4240,303C
408 DATA 0001,3F00,6100,17B4,207C,0000,3322,6100
409 DATA 170A,33FC,0004,0000,3DF6,33FC,0000,0000
410 DATA 3DF4,6100,1B5E,6100,1B0E,6100,1BB2,6100
411 DATA 1F0E,6100,1B8B,4B40,B03C,004B,672E,B03C
412 DATA 0050,6700,0092,B03C,001C,6700,00FC,B03C
413 DATA 004B,6710,B03C,004D,6702,60D6,6100,EE46
414 DATA 6000,00E6,6100,EE1C,6000,00DE,3039,0000
415 DATA 3DFA,B07C,0000,6712,0479,0100,0000,3DFA
416 DATA 04B9,0000,0100,0000,3DEB,3039,0000,3DFA
417 DATA E04B,E248,5240,33C0,0000,3EA6,33FC,003B
418 DATA 0000,3DF4,33FC,0002,0000,3DF6,6100,1AD4
419 DATA 207C,0000,3315,6100,1662,3F39,0000,3EA6
420 DATA 6100,16F8,207C,0000,3322,6100,164E,6100
421 DATA 1A24,6000,FF5A,3039,0000,3DFA,3239,0000
422 DATA 3E20,927C,0100,B041,6712,0679,0100,0000
423 DATA 3DFA,06B9,0000,0100,0000,3DEB,3039,0000
424 DATA 3DFA,E04B,E248,5240,33C0,0000,3EA6,33FC
425 DATA 003E,0000,3DF4,33FC,0002,0000,3DF6,6100
426 DATA 1A62,207C,0000,3315,6100,15F0,3F39,0000
427 DATA 3EA6,6100,1686,207C,0000,3322,6100,15DC
428 DATA 6100,19B2,6000,FEED,6100,1A94,4E75,2F0C
429 DATA 33FC,0002,0000,3DF6,6100,1A2B,6100,19A2
430 DATA 207C,0000,3326,6100,15B2,343C,0021,2B7C
431 DATA 0000,317A,101C,3F00,6100,1A7C,51CA,FFF6
432 DATA 207C,0000,3341,6100,1592,6100,1A52,6100
433 DATA 1AB0,B03C,0059,6706,B03C,0079,6634,3F39

434 DATA 0000,3E4C,3F39,0000,3DD8,3F39,0000,3DD4
 435 DATA 3F3C,0001,3F39,0000,3DDA,42A7,2F3C,0000
 436 DATA AC0C,3F3C,0009,4E4E,DFFC,0000,0014,4A40
 437 DATA 6B1A,6100,1986,6100,192B,6100,197E,207C
 438 DATA 0000,32E7,6100,1534,2B5F,4E75,3F00,6100
 439 DATA ECF6,60E6,2F3C,0000,0001,3F3C,0020,4E41
 440 DATA 5C8F,4A40,6610,42A7,3F3C,0020,4E41,5C8F
 441 DATA 23C0,0000,3E12,4E75,2F3C,0000,0001,3F3C
 442 DATA 0020,4E41,5C8F,4A40,670E,2F39,0000,3E12
 443 DATA 3F3C,0020,4E41,5C8F,4E75,51CF,FFFE,4E75
 444 DATA 61B2,33FC,0080,00FF,8606,3C3C,0000,6124
 445 DATA 3E3C,0028,61E4,4E75,619A,3639,00FF,8604
 446 DATA 4E71,40E7,3F07,3E3C,002B,51CF,FFFE,3E1F
 447 DATA 46DF,4E75,6100,FF7E,61E8,33C6,00FF,8604
 448 DATA 61E0,4E75,6100,FF6E,61DB,33F9,00FF,8604
 449 DATA 0000,3A0A,61CC,4E75,6100,FF5A,3039,0000
 450 DATA 3DDA,B07C,0001,6E34,5200,E308,8079,0000
 451 DATA 3DD8,0A00,0007,C03C,0007,40E7,007C,0700
 452 DATA 13FC,000E,00FF,8B00,1239,00FF,8B00,C23C
 453 DATA 00FB,8200,13C1,00FF,8B02,46DF,4E75,6100
 454 DATA FF14,33FC,0080,00FF,8606,103C,0007,61CA
 455 DATA 4E75,6100,FF00,42B9,0000,3E1C,40F9,0000
 456 DATA 3EA6,46FC,2700,33FC,0090,00FF,8606,33FC
 457 DATA 0190,00FF,8606,33FC,0090,00FF,8606,3C3C
 458 DATA 0016,343C,0200,C4C6,33C2,0000,3E20,D4BC
 459 DATA 0000,AC0C,23C2,0000,3E4E,6100,FF38,203C
 460 DATA 0000,AC0C,13C0,00FF,860D,E08B,13C0,00FF
 461 DATA 860B,E08B,13C0,00FF,8609,33FC,0080,00FF
 462 DATA 8606,3C3C,00EB,6100,FF0C,2E3C,0005,0000
 463 DATA 2A79,0000,3E4E,303C,0200,51C8,FFFE,0B39
 464 DATA 0005,00FF,FA01,672A,53B7,6756,13F9,00FF
 465 DATA 8609,0000,3E1D,13F9,00FF,860B,0000,3E1E
 466 DATA 13F9,00FF,860D,0000,3E1F,BBF9,0000,3E1C
 467 DATA 6ECC,33FC,0090,00FF,8606,3A39,00FF,8606
 468 DATA 33C5,0000,3E1A,0B05,0000,6714,33FC,00B0
 469 DATA 00FF,8606,6100,FEAE,46F9,0000,3EA6,4E75
 470 DATA 60F6,60F4,6100,FE0E,6142,33FC,0086,00FF
 471 DATA 8606,3C39,0000,3DD4,6100,FE7A,33FC,00B0
 472 DATA 00FF,8606,3C3C,001B,6100,FE6A,2E3C,0006
 473 DATA 0000,53B7,670C,0B39,0005,00FF,FA01,66F2

474 DATA 4E75,3F3C,FFF9,6100,EABE,4E75,3C39,0000
475 DATA 3A06,CC7C,0003,2E3C,0005,0000,33FC,0080
476 DATA 00FF,8606,6100,FE2E,5387,670C,0839,0005
477 DATA 00FF,FA01,66F2,4E75,3F3C,FFF9,6100,EAB9
478 DATA 4E75,203C,0000,AC0C,13C0,00FF,B60D,E088
479 DATA 13C0,00FF,B60B,E0BB,13C0,00FF,8609,4E75
480 DATA 48E7,1F1E,6100,16D4,6100,1676,6100,16CC
481 DATA 207C,0000,32FA,6100,12B2,33FC,0012,0000
482 DATA 3E04,6100,FD50,50F9,0000,043E,6100,FD5A
483 DATA 6100,FD8E,6100,FF2E,6100,FE38,6100,FE3A
484 DATA 6100,FD7E,6100,FD52,611B,6100,FD28,6100
485 DATA FE0E,51F9,0000,043E,6100,FD3E,4CDF,78FB
486 DATA 4E75,33FC,0000,0000,3DFA,23FC,0000,AC0C
487 DATA 0000,3DE8,33FC,0012,0000,3E04,33FC,0064
488 DATA 0000,3E06,33FC,1E00,0000,3E96,33FC,1ED0
489 DATA 0000,3E98,6100,165B,6100,161C,6100,EEBC
490 DATA 6100,16BC,4E75,6100,1632,6100,15D4,207C
491 DATA 0000,3B9C,6100,11E4,3039,0000,3DDA,B07C
492 DATA 0002,6E00,008A,6100,FCAC,6100,FD4C,6100
493 DATA FCF0,6100,FE90,6100,FEBC,6100,FF06,611A
494 DATA 6100,FCDE,6100,FCB2,6100,006E,6100,FC86
495 DATA 6100,FD6C,6100,FCA2,4E75,6100,FC78,33FC
496 DATA 0090,00FF,8606,33FC,0190,00FF,8606,33FC
497 DATA 0090,00FF,B606,3C3C,0001,6100,FCDB,33FC
498 DATA 0080,00FF,8606,3B3C,0018,3C3C,00C8,2E3C
499 DATA 0004,0000,6100,FCBE,0839,0005,00FF,FA01
500 DATA 6706,5387,6708,60F0,51CC,FFE0,4E75,3F3C
501 DATA FFFA,6100,E912,4E75,6100,1580,6100,1522
502 DATA 207C,0000,3B30,6100,1132,6100,15B2,3A3C
503 DATA 0011,267C,0000,AC0C,3B3C,0002,3F3C,0020
504 DATA 6100,15F4,101B,3F00,6100,1180,3F3C,0020
505 DATA 6100,15E4,3F3C,0020,6100,15DC,51CC,FFE6
506 DATA 3F3C,0020,6100,15D0,3F3C,0020,6100,15CB
507 DATA 101B,4BB0,323C,0080,B07C,0000,6718,323C
508 DATA 0100,B07C,0001,670E,323C,0200,B07C,0002
509 DATA 6704,323C,0400,3F01,6100,1160,3F3C,0020
510 DATA 6100,1594,207C,0000,3B90,6100,10AE,101B
511 DATA 3F00,6100,10D8,101B,3F00,6100,10D0,3F3C
512 DATA 000D,6100,1572,3F3C,000A,6100,156A,51CD
513 DATA FF68,6100,157C,4E75,6100,14C0,6100,1462

514 DATA 33FC,0014,0000,3DF4,33FC,0002,0000,3DF6
 515 DATA 6100,14D0,207C,0000,3230,6100,105E,33FC
 516 DATA 0200,0000,3E96,33FC,02D0,0000,3E98,23FC
 517 DATA 0000,AC0C,0000,3E52,6100,EF2C,6100,147C
 518 DATA 6100,141E,6100,1474,207C,0000,3447,6100
 519 DATA 102A,6100,E78E,6100,E7BA,6100,E786,6100
 520 DATA 03AA,4E75,3F3C,FFFF,3F3C,000B,4E4D,588F
 521 DATA 0B00,0000,660C,0B00,0001,6606,343C,0001
 522 DATA 6904,343C,000A,3039,0000,3DDC,9042,B07C
 523 DATA 0000,6D02,6006,3039,0000,3DE6,33C0,0000
 524 DATA 3DDC,4BC0,80FC,03E8,D03C,0030,13C0,0000
 525 DATA 3422,4B40,4BC0,80FC,0064,D03C,0030,13C0
 526 DATA 0000,3423,4B40,4BC0,80FC,000A,D03C,0030
 527 DATA 13C0,0000,3424,4B40,D03C,0030,13C0,0000
 528 DATA 3425,6100,1158,4E75,3F3C,FFFF,3F3C,000B
 529 DATA 4E4D,588F,0B00,0000,660C,0B00,0001,6606
 530 DATA 343C,0001,6094,343C,000A,3039,0000,3DDC
 531 DATA D042,B079,0000,3DE6,6D04,303C,0000,33C0
 532 DATA 0000,3DDC,4BC0,80FC,03E8,D03C,0030,13C0
 533 DATA 0000,3422,4B40,4BC0,80FC,0064,D03C,0030
 534 DATA 13C0,0000,3423,4B40,4BC0,80FC,000A,D03C
 535 DATA 0030,13C0,0000,3424,4B40,D03C,0030,13C0
 536 DATA 0000,3425,6100,10D6,4E75,3039,0000,3DDC
 537 DATA 33C0,0000,3E44,6148,3039,0000,3E46,4A40
 538 DATA 6722,B07C,0FFB,6C1C,5340,33C0,0000,3DDC
 539 DATA 23FC,0000,0003,0000,3DDC,6100,FF4C,6100
 540 DATA 015A,4E75,6100,1314,3F3C,FFED,6100,E698
 541 DATA 6100,130B,207C,0000,3447,6100,0EBE,60E2
 542 DATA 207C,0000,5DEC,3039,0000,3E44,323C,0003
 543 DATA C2C0,E249,0B00,0000,6616,1030,1001,E148
 544 DATA 8030,1000,C07C,0FFF,33C0,0000,3E46,6016
 545 DATA 1030,1001,E148,1030,1000,EB4B,C07C,0FFF
 546 DATA 33C0,0000,3E46,4E75,4BE7,1C1C,33FC,0000
 547 DATA 0000,3DF4,33FC,0002,0000,3DF6,6100,12D4
 548 DATA 207C,0000,356A,6100,0E52,267C,0000,317A
 549 DATA 363C,0009,101B,3F00,6100,131C,51CB,FFF6
 550 DATA 267C,0000,341A,363C,000C,101B,3F00,6100
 551 DATA 1306,51CB,FFF6,207C,0000,31E9,6100,0E1C
 552 DATA 6100,12DC,6100,130A,B03C,0079,6706,B03C
 553 DATA 0059,6660,3F39,0000,3DDA,3039,0000,3DDC

554 DATA 5540,C1F9,0000,3E2B,D079,0000,3E32,3F00
555 DATA 3F39,0000,3E2B,2F3C,0000,AC0C,3F3C,0003
556 DATA 3F3C,0004,4E4D,DFFC,0000,000E,4A40,6B1C
557 DATA 6100,1208,6100,11AA,6100,1200,207C,0000
558 DATA 3447,6100,0CB6,4CDF,3938,4E75,3F00,6100
559 DATA E576,60DC,6100,11E4,6100,11B6,6100,11DC
560 DATA 207C,0000,31FB,6100,0D92,6100,1252,6100
561 DATA 1280,6100,116C,60B8,4E75,48E7,1F1E,3F39
562 DATA 0000,3DDA,3039,0000,3DDC,5540,C1F9,0000
563 DATA 3E2B,D079,0000,3E32,4A40,6A04,303C,0000
564 DATA 33C0,0000,3E4A,3F00,3F3C,0002,2F3C,0000
565 DATA AC0C,3F3C,0000,3F3C,0004,4E4D,DFFC,0000
566 DATA 000E,4A40,6B5C,3039,0000,3E4A,B1FC,0009
567 DATA 4B40,5240,33C0,0000,3DD6,4B40,3400,33FC
568 DATA 0000,0000,3DD8,3239,0000,3E40,B27C,0002
569 DATA 6610,E24B,0802,0000,6708,33FC,0001,0000
570 DATA 3DD8,33C0,0000,3DD4,612A,6100,007E,6100
571 DATA 112A,207C,0000,3447,6100,0CE0,4CDF,7BFB
572 DATA 4E75,6100,EF3B,4A80,6600,FF54,3F00,6100
573 DATA E496,60DA,3039,0000,3DD8,D03C,0030,13C0
574 DATA 0000,318C,3039,0000,3DD6,4BC0,B1FC,000A
575 DATA D03C,0030,13C0,0000,31A4,4B40,D03C,0030
576 DATA 13C0,0000,31A5,3039,0000,3DD4,4BC0,B1FC
577 DATA 000A,D03C,0030,13C0,0000,3197,4B40,D03C
578 DATA 0030,13C0,0000,3198,4E75,33FC,0000,0000
579 DATA 3DFA,33FC,0012,0000,3E04,33FC,003F,0000
580 DATA 3E06,23FC,0000,AC0C,0000,3DEB,33FC,0200
581 DATA 0000,3E96,33FC,02D0,0000,3E9B,6100,1090
582 DATA 6100,1054,33FC,0000,0000,3DF4,33FC,001B
583 DATA 0000,3DF6,6100,108C,207C,0000,345C,6100
584 DATA 0C1A,6100,E8D6,4E75,4BE7,1F1C,6100,EE6E
585 DATA 6100,EED0,6100,EF02,33FC,0000,0000,3DF4
586 DATA 33FC,0002,0000,3DF6,6100,1058,207C,0000
587 DATA 34CB,6100,0BE6,33FC,0011,0000,3E04,6100
588 DATA 0FB4,267C,0000,3EAC,284B,23CB,0000,3DEB
589 DATA 23CB,0000,3DEC,6100,1016,6100,027E,6100
590 DATA 100E,23FC,0000,3EAC,0000,3DEB,6100,0F6E
591 DATA 6100,0334,6100,0F72,6100,1042,4B40,B03C
592 DATA 001C,6700,0182,B03C,004B,6724,B03C,0050
593 DATA 6700,00AA,B03C,004B,670E,B03C,004D,66DB

594 DATA 6100,E302,6000,0144,6100,E20B,6000,013C
 595 DATA 3039,0000,3DF6,B07C,0004,6F30,33FC,0000
 596 DATA 0000,3DF4,6100,0FBC,6100,02DC,5379,0000
 597 DATA 3DF6,33FC,0000,0000,3DF4,6100,0FA6,6100
 598 DATA 0EFC,6100,02C2,6100,0F00,604C,0CB9,0000
 599 DATA 3EAC,0000,3DE8,6740,2039,0000,3DE8,3039
 600 DATA 0000,3E04,5240,C1FC,0020,91B9,0000,3DE8
 601 DATA 6100,01CB,33FC,0015,0000,3DF6,33FC,0000
 602 DATA 0000,3DF4,6100,0F5C,6100,0EB2,6100,027B
 603 DATA 6100,0EB6,6100,0FAB,6000,FF3E,3039,0000
 604 DATA 3DF6,B07C,0014,6E50,3039,0000,3DF6,5240
 605 DATA 5940,48C0,EB88,2C79,0000,3DE8,1036,0B00
 606 DATA 6700,0084,33FC,0000,0000,3DF4,6100,0F14
 607 DATA 6100,0234,5279,0000,3DF6,33FC,0000,0000
 608 DATA 3DF4,6100,0EFE,6100,0E54,6100,0214,6100
 609 DATA 0E5B,6100,0EEE,604E,3039,0000,3DF6,5240
 610 DATA 5940,48C0,EB88,2C79,0000,3DE8,1036,0B00
 611 DATA 6734,3039,0000,3E04,5240,C1FC,0020,0CB9
 612 DATA 0000,3DE8,23C0,0000,3DE8,6100,0EA2,6100
 613 DATA 010A,6100,0E9A,6100,0E04,6100,01CA,6100
 614 DATA 0E0B,6100,0EFA,6000,FE90,6100,0E6E,6100
 615 DATA 0E10,6100,0E66,207C,0000,3447,6100,0A1C
 616 DATA 4C0F,38FB,4E75,2079,0000,3DE8,3039,0000
 617 DATA 3DF6,5940,48C0,EB88,1230,0B0B,B23C,0010
 618 DATA 6726,3039,0000,3E4B,5340,33C0,0000,3DDC
 619 DATA 23FC,0000,0303,0000,3DCC,6100,FA4C,60AA
 620 DATA 6100,ECD6,6000,0086,4A79,0000,3E4B,67F0
 621 DATA 3039,0000,3E4B,23FC,0000,3EAC,0000,3EAB
 622 DATA 4243,3039,0000,3E4B,3F39,0000,3DDA,5540
 623 DATA C1F9,0000,3E2B,0C79,0000,3E32,3F00,3F3C
 624 DATA 0002,2F39,0000,3EAB,3F3C,0002,3F3C,0004
 625 DATA 4E40,0FFC,0000,000E,4A40,6B34,06B9,0000
 626 DATA 0400,0000,3EAB,33F9,0000,3E4B,0000,3E44
 627 DATA 6100,FAE,3039,0000,3E4B,33C0,0000,3E4B
 628 DATA 4A40,670B,B07C,0FFB,6C02,6096,6300,FDAA
 629 DATA 3F00,6100,E112,6000,FF12,33FC,0000,0000
 630 DATA 3E3C,6100,0DBA,6100,0D4E,2A79,0000,3DE8
 631 DATA 3E39,0000,3E04,103C,0020,3F00,6100,0DFB
 632 DATA 133C,0020,3F00,6100,0DEE,4284,3E3C,0009
 633 DATA 1035,4B00,6700,00B6,5284,3F00,6100,0DEB

634 DATA 1035,4800,5284,3F00,6100,0000,510E,FFF2
635 DATA 33FC,0014,0000,3E42,6100,0074,6100,015E
636 DATA 33FC,002B,0000,3E42,6100,0064,6100,003C
637 DATA 33FC,0037,0000,3E42,6100,0054,6100,0104
638 DATA 33FC,0000,0000,3DF4,5279,0000,3DF6,6100
639 DATA 0012,0EFC,0000,0020,510F,FF7C,33FC,0000
640 DATA 0000,3DF4,33FC,001B,0000,3DF6,6100,00F4
641 DATA 207C,0000,3E1C,6100,0882,4E75,33FC,0001
642 DATA 0000,3E3C,60D6,2679,0000,3DEB,33FC,0000
643 DATA 0090,3E3C,3639,0000,3DF6,5943,48C3,EB8B
644 DATA 1B3C,0020,3F04,6100,002E,3F04,6100,0D2B
645 DATA 1033,3B00,674A,3F00,6100,0D1C,52B3,3C3E
646 DATA 0009,1033,3B00,52B3,3F00,6100,0D0A,510E
647 DATA FFF2,33FC,0014,0000,3E42,6100,00B2,6100
648 DATA 009C,33FC,002B,0000,3E42,6100,00A2,611A
649 DATA 33FC,0037,0000,3E42,6100,0094,6144,4E75
650 DATA 33FC,0001,0000,3E3C,60F4,103C,0020,3F00
651 DATA 6100,0CC4,3039,0000,3DF6,5940,48C0,EB8B
652 DATA 1233,0B1B,5149,1233,0B1A,33C1,0000,3E4B
653 DATA 3F01,6100,0B66,103C,0020,3F00,6100,0C9B
654 DATA 4E75,3639,0000,3DF6,5943,48C3,EB8B,42B1
655 DATA 2679,0000,3DEB,1233,3B1F,5189,1233,3B1E
656 DATA 5189,1233,3B1D,5189,1233,3B1C,2F01,6100
657 DATA 0832,3F3C,0020,6100,005E,4E75,3639,0000
658 DATA 3DF6,5943,48C3,EB8B,2679,0000,3DEB,1233
659 DATA 3800,B23C,0055,675B,1233,3E0B,B23C,0010
660 DATA 672A,B23C,0001,6730,B23C,0002,6736,B23C
661 DATA 000B,670C,207C,0000,36ED,6100,073E,4E75
662 DATA 207C,0000,3721,6100,0732,60F2,207C,0000
663 DATA 36DC,6100,0726,60EE,207C,0000,36EE,6100
664 DATA 071A,60DA,207C,0000,36FF,6100,070E,60CE
665 DATA 207C,0000,3710,6100,0702,60C2,6100,0B3C
666 DATA 6100,06DE,6100,0B8B,207C,0000,37D0,6100
667 DATA 06EA,3F39,0000,3D34,6100,07B0,207C,0000
668 DATA 3B19,6100,06D6,3F39,0000,3DEB,6100,076C
669 DATA 207C,0000,3B24,6100,06C2,3F39,0000,3D3A
670 DATA 6100,075B,207C,0000,37D8,6100,06AE,6100
671 DATA 0B40,B03C,0079,671E,B03C,0059,671B,6100
672 DATA 0AB0,6100,0B5A,207C,0000,37F7,6100,068C
673 DATA 6100,0B7E,603E,3F3C,E5EE,2F3C,8765,4321

674 DATA 3F3C,0001,3F39,0000,3DD8,3F39,0000,3DD4
 675 DATA 3F39,0000,3E4C,3F39,0000,3DDA,42A7,2F3C
 676 DATA 0000,7D2C,3F3C,000A,4E4E,DFFC,0000,001A
 677 DATA 4A4C,6B1C,6100,0A8A,6100,0A2E,6100,0A7C
 678 DATA 207C,0000,37B6,6100,0832,6100,3CE9,4E75
 679 DATA 3F00,6100,DDF2,60DC,33FC,0001,0000,3A04
 680 DATA 247C,0000,7D2C,3039,0000,3B26,3E3C,004E
 681 DATA 6100,00D4,3039,0000,3B2B,3E3C,0000,6100
 682 DATA 00C6,303C,0003,1E3C,00F5,6100,00BA,14FC
 683 DATA 00FE,3039,0000,3DD4,14C0,3039,0000,3DD8
 684 DATA 14C0,3039,0000,3A04,14C0,3039,0000,3B24
 685 DATA B07C,0400,671E,B07C,029C,6712,B07C,0100
 686 DATA 6706,323C,0000,601C,323C,0001,600A,323C
 687 DATA 0002,6004,323C,0003,14C1,14FC,00F7,3039
 688 DATA 0000,3B2A,3E3C,004E,615C,3039,0000,3B2B
 689 DATA 3E3C,0000,6150,303C,0003,3E3C,00F5,6146
 690 DATA 14FC,00FE,3039,0000,3B24,1E3C,00E5,6136
 691 DATA 14FC,00F7,3039,0000,3B2C,3E3C,004E,6126
 692 DATA 3039,0000,3A04,5240,33C0,0000,3A04,B079
 693 DATA 0000,3E4C,6F00,FFFE,3039,0000,3B2E,3E3C
 694 DATA 004E,6192,4E75,5340,14D7,51C8,FFFC,4E75
 695 DATA 203C,0000,7D2C,13C0,00FF,860D,E08B,13C0
 696 DATA 00FF,860B,E08B,13C0,00FF,8609,4E75,33FC
 697 DATA 0190,00FF,8606,33FC,0090,00FF,8606,33FC
 698 DATA 0190,00FF,8606,3C3C,001F,6100,F04B,33FC
 699 DATA 01B0,00FF,8606,3C3C,00FB,6100,F03B,2E3C
 700 DATA 0006,0000,53B7,670C,0839,0005,00FF,FA01
 701 DATA 66F2,4E75,3F3C,FFEE,6100,DC2C,4E75,4BE7
 702 DATA 1F1E,6100,0B56,6100,0B9B,207C,0000,3B30
 703 DATA 6100,04A8,6100,0968,6100,0996,E03C,0079
 704 DATA 670B,303C,0059,6600,0064,6100,EF6B,50F9
 705 DATA 0000,043E,6100,F1CC,6100,FFFE,6100,EFA2
 706 DATA 6100,F142,6100,F04C,6100,FF46,6100,FE4A
 707 DATA 6100,F132,6100,FF5B,6100,0924,6100,0B9C
 708 DATA 6100,0E3E,6100,EF7A,6100,EF4E,207C,0000
 709 DATA 3B7B,6100,0446,6100,093B,6100,EF12,51F9
 710 DATA 0000,043E,6100,FFFE,6100,EF2E,6100,0B6C
 711 DATA 6100,0B0E,6100,0B64,207C,0000,3AB0,6100
 712 DATA 041A,4CDF,7BFB,4E75,B07C,0063,6D06,303C
 713 DATA 0000,6002,5240,4E75,3039,0000,3B26,61E8

714 DATA 33C0,0000,3B26,80FC,000A,D03C,0030,13C0
715 DATA 0000,3A67,4B40,D03C,0030,13C0,0000,3A68
716 DATA 6100,059A,4E75,3039,0000,3B2B,61BA,33C0
717 DATA 0000,3B2B,80FC,000A,D03C,0030,13C0,0000
718 DATA 3A72,4B40,D03C,0030,13C0,0000,3A73,6100
719 DATA 056C,4E75,3039,0000,3B2A,618C,33C0,0090
720 DATA 3B2A,80FC,000A,D03C,0030,13C0,0000,3A7D
721 DATA 4B40,D03C,0030,13C0,0000,3A7E,6100,053E
722 DATA 4E75,3039,0000,3B2C,6100,FF5E,33C0,0000
723 DATA 3B2C,80FC,000A,D03C,0030,13C0,0000,3A8B
724 DATA 4B40,D03C,0030,13C0,0000,3A89,6100,050E
725 DATA 4E75,3F3C,FFFF,3F3C,000B,4E4D,5E8F,323C
726 DATA 000A,0B00,0000,660A,0B00,0001,6604,323C
727 DATA 0001,3039,0000,3B2E,D041,B07C,03E7,6F04
728 DATA 303C,0000,33C0,0000,3B2E,4B00,81FC,0064
729 DATA D03C,0030,13C0,0000,3A93,4B40,4ECC,81FC
730 DATA 000A,D03C,0030,13C0,0000,3A94,4B40,D03C
731 DATA 0030,13C0,0000,3A95,6100,04A2,4E75,B07C
732 DATA 0000,6F04,5340,6004,303C,0063,4E75,3039
733 DATA 0000,3B26,61EB,33C0,0000,3B26,80FC,000A
734 DATA D03C,0030,13C0,0000,3A67,4B40,D03C,0030
735 DATA 13C0,0000,3A62,6100,0464,4E75,3039,0000
736 DATA 3B2B,61BA,33C0,0000,3B2B,80FC,000A,D03C
737 DATA 0030,13C0,0000,3A72,4B40,D03C,0030,13C0
738 DATA 0000,3A73,6100,0436,4E75,3039,0000,3B2A
739 DATA 618C,33C0,0000,3B2A,80FC,000A,D03C,0030
740 DATA 13C0,0000,3A7D,4B40,D03C,0030,13C0,0000
741 DATA 3A7E,6100,040B,4E75,3039,0000,3B2C,6100
742 DATA FF5E,33C0,0000,3B2C,80FC,000A,D03C,0030
743 DATA 13C0,0000,3A8B,4B40,D03C,0030,13C0,0000
744 DATA 3A89,6100,03D8,4E75,3F3C,FFFF,3F3C,000B
745 DATA 4E4D,5E8F,323C,000A,0B00,0000,660A,0B00
746 DATA 0001,6604,323C,0001,3039,0000,3B2E,9041
747 DATA 6A04,303C,03E7,33C0,0000,3B2E,4B00,81FC
748 DATA 0064,D03C,0030,13C0,0000,3A93,4B40,4B00
749 DATA 81FC,000A,D03C,0030,13C0,0000,3A94,4B40
750 DATA D03C,0030,13C0,0000,3A95,6100,0370,4E75
751 DATA 3039,0000,3B2A,B07C,00B0,6732,B07C,0100
752 DATA 6752,B07C,0200,6772,303C,00B0,13FC,0030
753 DATA 0000,3AA3,13FC,0031,0000,3AAA,13FC,0032

754 DATA 0000,3AA5,13FC,0038,0000,3AA5,6070,303C
 755 DATA 0100,13FC,0030,0000,3AA3,13FC,0032,0000
 756 DATA 3AA4,13FC,0035,0000,3AA5,13FC,0036,0000
 757 DATA 3AA6,604A,303C,0200,13FC,0030,0000,3AA3
 758 DATA 13FC,0035,0000,3AA4,13FC,0031,0000,3AA5
 759 DATA 13FC,0032,0000,3AA6,6024,303C,0400,13FC
 760 DATA 0031,0000,3AA3,13FC,0030,0000,3AA4,13FC
 761 DATA 0032,0000,3AA5,13FC,0034,0000,3AA6,33C0
 762 DATA 0000,3B24,6100,02B6,4E75,3039,0000,3B24
 763 DATA B07C,00B0,6732,B07C,0100,6752,B07C,0200
 764 DATA 6772,303C,0200,13FC,0030,0000,3AA3,13FC
 765 DATA 0035,0000,3AA4,13FC,0031,0000,3AA5,13FC
 766 DATA 0032,0000,3AA6,6070,303C,0400,13FC,0031
 767 DATA 0000,3AA3,13FC,0030,0000,3AA4,13FC,0032
 768 DATA 0000,3AA5,13FC,0034,0000,3AA6,604A,303C
 769 DATA 00B0,13FC,0030,0000,3AA3,13FC,0031,0000
 770 DATA 3AA4,13FC,0032,0000,3AA5,13FC,003B,0000
 771 DATA 3AA6,6024,303C,0100,13FC,0030,0000,3AA3
 772 DATA 13FC,0032,0000,3AA4,13FC,0035,0000,3AA5
 773 DATA 13FC,0036,0000,3AA6,33C0,0000,3B24,6100
 774 DATA 01FC,4E75,2079,0000,3E9A,317C,0000,0026
 775 DATA 317C,0014,002B,317C,027F,002A,317C,0001
 776 DATA 001B,317C,0000,0024,217C,0000,32C2,002E
 777 DATA 317C,0000,0032,A004,4E75,2F0B,3F3C,0009
 778 DATA 4E41,5CBF,4E75,A000,23C8,0000,3E9A,317C
 779 DATA 0000,0020,317C,FFFF,0022,317C,0000,0024
 780 DATA 317C,0001,001B,4E75,61DC,4E75,322F,0004
 781 DATA C27C,03FF,33C1,0000,3EA2,EB49,4881,C27C
 782 DATA 00FF,B27C,0009,6E04,6120,6002,6132,3239
 783 DATA 0000,3EA2,C27C,000F,B27C,0009,6E04,610A
 784 DATA 6002,611C,205F,54BF,4ED0,B27C,0030,3F01
 785 DATA 3F39,0000,39FC,3F3C,0003,4E4D,5CBF,4E75
 786 DATA 927C,000A,B27C,0041,3F01,3F39,0000,39FC
 787 DATA 3F3C,0003,4E4D,5CBF,4E75,33FC,0000,0000
 788 DATA 3E3A,362F,0004,4BC3,87FC,2719,670B,33FC
 789 DATA FFFF,0000,3E3A,614E,4B43,4BC3,87FC,03EB
 790 DATA 670B,33FC,FFFF,0000,3E3A,613A,4B43,4BC3
 791 DATA 87FC,0064,670B,33FC,FFFF,0000,3E3A,6126
 792 DATA 4B43,4BC3,87FC,000A,670B,33FC,FFFF,0000
 793 DATA 3E3A,6112,4B43,33FC,FFFF,0000,3E3A,6106

794 DATA 205F,548F,4ED0,4A77,0000,3E3A,6608,103C
795 DATA 0320,3F00,6006,D63C,0030,3F03,6100,03BE
796 DATA 4E75,33FC,0000,0000,3E3A,262F,0004,2803
797 DATA 87FC,2710,4ED3,87FC,000A,3A03,4A43,6708
798 DATA 33FC,FFFF,0000,3E3A,61BC,3605,07FC,000A
799 DATA C7FC,2710,9883,2604,87FC,2710,6708,33FC
800 DATA FFFF,0000,3E3A,619E,4843,48C3,57FC,03EB
801 DATA 670B,33FC,FFFF,0000,3E3A,619A,4843,48C3
802 DATA 87FC,0064,6708,33FC,FFFF,0000,3E3A,6100
803 DATA FF76,4843,48C3,87FC,000A,670B,33FC,FFFF
804 DATA 0000,3E3A,6100,FF60,4843,33FC,FFFF,0000
805 DATA 3E3A,6100,FF52,205F,585F,4ED0,33FC,0000
806 DATA 0000,3DF4,33FC,0000,0000,3DF6,6100,0294
807 DATA 2C79,0000,3DC4,2C39,0000,3DCC,5386,670C
808 DATA 5386,205E,6100,FE14,51CE,FFFF,6100,01CE
809 DATA 205E,6100,FE06,6100,01D0,2E39,0000,3DCB
810 DATA 7EB9,0000,3DCC,670C,5387,205E,6100,FBED
811 DATA 51CE,FFFF,6100,FD4E,6100,01BA,4E75,4BF9
812 DATA 3BF8,0000,3E56,2879,0000,3DEB,2A4C,33F9
813 DATA 0000,3DFA,0000,3DF8,363C,000F,3B03,3A39
814 DATA 0000,3E04,33FC,0004,0000,3DF6,33FC,0004
815 DATA 0000,3DFC,33FC,0000,0000,3DF4,6100,0204
816 DATA 33FC,0000,0000,3DF4,33F9,0000,3DFC,0000
817 DATA 3DF6,3604,6100,01EC,6146,616B,3604,2E4D
818 DATA 33FC,0032,0000,3DF4,33F9,0000,3DFC,0000
819 DATA 3DF6,6100,01CE,6100,0072,DFFC,0000,0019
820 DATA 5279,0000,3DFC,0679,0019,0000,3DF8,51CD
821 DATA FFB0,6100,020A,6CF9,3BF8,0000,3E56,4E75
822 DATA 3C39,0000,3DF6,504E,3F06,6100,FB60,3F39
823 DATA 0000,3BF8,6100,FB5A,1C3C,003A,3F06,6100
824 DATA 01FA,4E75,3F3C,0020,6100,01ED,3F3C,0020
825 DATA 6100,0154,1E1C,3F07,6100,FB32,3F3C,0020
826 DATA 6100,01B4,51CB,FFEE,4E75,1E3C,003A,3F07
827 DATA 6100,01C4,3F3C,0020,6100,01ED,1E1C,8E3C
828 DATA 0020,6E04,1E3C,002E,4B37,CE7D,00FF,3F07
829 DATA 6100,0104,51CB,FFE6,4E75,33F9,0000,3DF6
830 DATA 0000,3E02,33FC,0000,0000,3DF4,6100,0114
831 DATA 33F9,0000,3E02,0000,3DF4,363C,000F,3B03
832 DATA 2879,0000,3DEB,4280,3C39,0000,3DF6,5940
833 DATA E94B,3200,0079,0000,3DFA,33C0,0000,3DF8

834 DATA 48D1,39C1,2A4C,6100,FF3E,6100,FF5B,33F9
 835 DATA 0000,3DF4,0000,3E02,33FC,003B,0000,3DF4
 836 DATA 6100,0000,3604,2B4D,6100,FF60,33F9,0000
 837 DATA 3E02,0000,3DF4,6100,00AA,4E75,207C,0000
 838 DATA 3B6A,6100,FC36,4E75,207C,0000,3B6D,6100
 839 DATA FC2A,4E75,207C,0000,3B67,6100,FC1E,4E75
 840 DATA 207C,0000,3B84,6100,FC12,4E75,207C,0000
 841 DATA 3B7B,6100,FC06,4E75,207C,0000,3B75,6100
 842 DATA FBFA,4E75,3F3C,001C,6100,000C,3F3C,000A
 843 DATA 6100,00C4,4E75,207C,0000,3B64,6100,FB5C
 844 DATA 4E75,207C,0000,3B8D,6100,FB30,4E75,207C
 845 DATA 0000,3B8A,6100,FB04,4E75,33FC,001E,0000
 846 DATA 3DF4,33FC,0002,0000,3DF6,6116,4E75,33FC
 847 DATA 0000,0000,3DF4,33FC,0004,0000,3DF6,6102
 848 DATA 4E75,207C,0000,3B70,548B,3037,0000,3DF6
 849 DATA D07C,0020,10C0,3039,0000,3DF4,D07C,0020
 850 DATA 10C0,207C,0000,3E7C,6100,FB70,4E75,33F9
 851 DATA 0000,3E42,0000,3DF4,6102,4E75,3F3C,0002
 852 DATA 3F3C,0001,4E4D,58BF,4A40,6A0E,3F3C,0002
 853 DATA 3F3C,0002,4E4D,58BF,4E75,7000,4E75,3F3C
 854 DATA 000B,4E41,548F,4A40,670A,3F3C,0007,4E41
 855 DATA 54BF,60EA,4E75,302F,0004,3F00,3F39,0000
 856 DATA 39FC,3F3C,0003,4E4D,5C9F,205F,548F,4E00
 857 DATA 3F3C,0001,4E41,548F,4E75,61FA,23C0,0000
 858 DATA 3E9E,B03C,0066,6E00,007B,B03C,0061,6D0A
 859 DATA 903C,0061,D03C,000A,6010,B03C,0030,6D60
 860 DATA B03C,0039,6E60,903C,0030,594B,33C0,0000
 861 DATA 3EA2,61BC,23C0,0000,3E9E,B03C,0066,6E40
 862 DATA B03C,0061,6D0A,903C,0061,D03C,000A,6010
 863 DATA B03C,0030,6D2A,B03C,0039,6E40,903C,0030
 864 DATA 3239,0000,3EA2,8041,4B90,D07C,00FF,33C0
 865 DATA 0000,3EA4,206F,0004,3080,205F,58BF,4E00
 866 DATA 303C,FFFF,60EE,B03C,0046,6EF4,B03C,0041
 867 DATA 6DEE,903C,0041,D03C,000A,60EE,B03C,0046
 868 DATA 6EDE,B03C,0041,6DCB,903C,0041,D03C,000A
 869 DATA 60AE,0000,0290,0000,02DA,0000,0320,0000
 870 DATA 0374,0000,03C2,0000,0456,0000,0186,0000
 871 DATA 302E,0000,303B,0000,3046,0000,3050,0000
 872 DATA 305B,0000,306E,0000,307A,2020,5452,4143
 873 DATA 4B20,2000,2054,5241,474B,2F53,594E,4753

974 DATA 2000,2053,4543,544F,5220,2000,2043,4C55
975 DATA 5354,4552,2020,0020,464F,524D,4154,2020
976 DATA 0020,2046,4154,5320,2000,2020,4F50,5449
977 DATA 4F4E,5320,2000,2020,4649,4E20,2020,0019
978 DATA 7020,2041,204C,4954,544C,4520,4449,5342
979 DATA 2055,5449,4C49,5459,2020,2029,4329,2055
980 DATA 2E20,4272,6173,6E20,3139,3836,2020,1B71
981 DATA 0019,7020,2020,2020,4441,5441,2042,4543
982 DATA 4B45,5220,2620,4D49,4352,4F20,4150,504C
983 DATA 4943,4154,494F,4E20,2020,2020,2020,2020
984 DATA 1B71,001B,7020,2020,5382,6C65,6374,696F
985 DATA 6E6E,6572,206C,6573,206D,656E,7573,2061
986 DATA 7665,6320,6C65,7320,666C,3A63,6865,7320
987 DATA 2020,1B71,0000,0000,049C,0000,04F4,0000
988 DATA 054C,0000,05CB,0000,0644,0000,08CA,0000
989 DATA 082E,0000,0254,0000,04CB,0000,0520,0000
990 DATA 058A,0000,0606,0000,0644,0000,08CA,0000
991 DATA 0E3E,0000,0254,0000,317A,0000,3185,0000
992 DATA 313F,0000,319B,0000,31A8,0000,31B1,0000
993 DATA 31EB,0000,31C4,2054,7269,766E,3A20,3020
994 DATA 0020,7369,6465,3A20,3020,0020,7472,6163
995 DATA 6B3A,2030,3020,0020,7365,6374,6F72,3A20
996 DATA 3031,2000,2020,5245,4144,2020,0020,2057
997 DATA 5249,5445,2020,0020,2045,4449,5420,2000
998 DATA 2020,4241,434B,2020,001B,7020,4563,7269
999 DATA 7265,2063,6520,7365,6374,6575,7220,3E20
900 DATA 3A20,1B71,001B,7020,203C,796E,732C,6E6F
901 DATA 3E20,3F20,1B71,001B,7020,416E,6E75,6C82
902 DATA 2020,2020,2020,3C52,6574,7572,6E3E,2020
903 DATA 2020,2020,1B71,001B,7020,204D,4F44,4520
904 DATA 534E,4354,4555,5220,2019,7100,1B70,2020
905 DATA 4D4F,4445,2045,4449,543A,2020,203C,5265
906 DATA 7475,726E,3E20,203A,3320,4649,4E20,2019
907 DATA 7100,0000,049C,0000,04F4,0000,054C,0000
908 DATA 11D0,0000,1164,0000,1472,0000,124C,0000
909 DATA 0254,0000,04CB,0000,0520,0000,058A,0000
910 DATA 120E,0000,1164,0000,1472,0000,124C,0000
911 DATA 0254,0000,317A,0000,3185,0000,318F,0000
912 DATA 32B6,0000,32C5,0000,32CD,0000,32D5,0000
913 DATA 32E0,2053,6563,2F54,7261,633A,2030,3920

914 DATA 0020,5245,4144,2020,0020,5752,4954,4520
 915 DATA 0020,4544,4954,2054,722E,2000,2042,4143
 916 DATA 4B20,001B,7020,204D,4F44,4520,5452,4143
 917 DATA 4B20,201B,7100,1B70,2020,4D4F,4445,2054
 918 DATA 5241,434B,202B,2053,594E,4353,2020,1E71
 919 DATA 001B,7020,2053,6563,746F,723A,2000,201B
 920 DATA 7100,1B70,2045,6372,6972,6520,6365,7474
 921 DATA 6520,7069,7374,6520,3E20,1B71,001B,7020
 922 DATA 3C20,7965,732F,6E6F,203E,201B,7100,0030
 923 DATA 049C,0000,04F4,0000,054C,0000,17A4,0000
 924 DATA 184A,0000,0254,0000,04CB,0000,0520,0000
 925 DATA 058A,0000,17A4,0000,184A,0000,0254,0000
 926 DATA 317A,0000,3185,0000,31EF,0000,339A,0000
 927 DATA 33AC,0000,32E0,2052,4541,4420,5749,5448
 928 DATA 2053,594E,4353,2000,2041,4444,522E,2046
 929 DATA 4945,4C44,2000,0000,049C,0000,1AAC,0000
 930 DATA 1CBE,0000,1B2E,0000,1BCC,0000,19BC,0000
 931 DATA 1E2C,0000,0254,0000,04CB,0000,1A2B,0000
 932 DATA 1CBE,0000,1B2E,0000,1BCC,0000,19BC,0000
 933 DATA 1E2C,0000,0254,0000,317A,0000,341A,0000
 934 DATA 31A8,0000,3429,0000,31B1,0000,3440,0000
 935 DATA 3432,0000,31C4,2043,4C55,5354,3A20,3030
 936 DATA 3030,2020,0020,204E,4558,5420,2000,2053
 937 DATA 5441,5254,6F66,4649,4C45,2000,2045,4449
 938 DATA 5420,001B,7020,2043,4C55,5354,4552,204D
 939 DATA 4F44,4520,201B,7100,1B70,2020,456E,2071
 940 DATA 7569,7474,616E,7420,6C65,206D,6F64,6520
 941 DATA 636C,7573,7465,722C,206C,6520,736E,6374
 942 DATA 6575,7265,7374,206D,6973,2095,206A,6F75
 943 DATA 7220,6461,6E73,206C,6520,6D65,6E75,2020
 944 DATA 2020,201B,7100,1B70,2020,4465,726E,6965
 945 DATA 7220,636C,7573,7465,7220,2020,2020,2020
 946 DATA 2020,2020,1B71,001B,7020,204E,6F6D,3A20
 947 DATA 2020,2020,2020,2020,2020,2020,2020,2041
 948 DATA 7474,7269,6275,743A,2020,2020,2020,2020
 949 DATA 2020,5374,6172,7463,6C75,7374,6572,3A20
 950 DATA 2020,204E,6F6D,6272,6520,6465,2042,7974
 951 DATA 6573,3A20,201B,7100,1B70,2020,6D65,7474
 952 DATA 7265,2053,7461,7274,2D43,6C75,7374,6572
 953 DATA 2064,616E,7320,6D65,6E75,2061,766E,6320

954 DATA 3C52,4554,5552,4E3E,206C,6563,7A75,7265
955 DATA 2070,6172,203C,7570,3E2C,203C,646F,776E
956 DATA 3E2E,201B,7100,1B70,2045,6372,6972,6520
957 DATA 6365,2063,6C75,7374,6572,203E,2020,3A20
958 DATA 1B71,0020,204F,6374,6574,732F,5365,6374
959 DATA 6575,723A,2000,2020,5365,6374,6575,722F
960 DATA 436C,7573,7465,723A,2000,2020,4F63,7465
961 DATA 7473,2F43,6C75,7374,6572,3A20,0020,2053
962 DATA 6563,7465,7572,2F44,6972,6563,746F,7279
963 DATA 3A20,0020,2053,6563,7465,7572,2F46,4154
964 DATA 3A20,0020,2032,6520,6E75,6DE2,726F,2064
965 DATA 6520,7365,6374,6575,7220,4641,543A,0020
966 DATA 2053,6563,7465,7572,2064,7520,3165,7220
967 DATA 436C,7573,7465,7264,6F6E,6E82,653A,0020
968 DATA 204E,6F6D,6272,6520,6465,2063,6C75,7374
969 DATA 6572,733A,2000,2020,4E6F,6D62,7265,2064
970 DATA 6520,6661,6365,733A,2000,1B70,2020,4669
971 DATA 7273,7420,4469,7265,6374,6F72,792D,7365
972 DATA 6374,6F72,203E,2020,4661,6365,3A20,3020
973 DATA 2054,7261,6369,3A20,3120,2053,6563,746F
974 DATA 723A,2033,2020,1B71,001B,7020,2045,6972
975 DATA 7374,2044,6972,6563,746F,7279,2D73,6E63
976 DATA 746F,7220,3E20,2046,6163,653A,2031,2020
977 DATA 5472,6163,653A,2030,2020,5365,6374,6F72
978 DATA 3A20,3320,201B,7100,2020,536F,7573,6469
979 DATA 7265,6374,6F72,7920,0020,2052,6561,642F
980 DATA 5772,6974,6520,2020,2000,2020,5265,6164
981 DATA 206F,6E6C,7920,2020,2020,0020,204B,4944
982 DATA 4445,4E20,4669,6C65,2020,2000,2020,6566
983 DATA 6661,6362,2020,2020,2020,2020,0020,204E
984 DATA 6F6D,2064,7520,6469,7371,7565,2000,0000
985 DATA 049C,0000,04F4,0000,056C,0000,1100,0000
986 DATA 2340,0000,2582,0000,040C,0000,0254,0000
987 DATA 04CB,0000,0520,0000,058A,0000,120E,0000
988 DATA 2340,0000,2582,0000,040C,0000,0254,0000
989 DATA 317A,0000,3185,0000,318F,0000,3286,0000
990 DATA 3792,0000,379B,0000,37A5,0000,37AD,2046
991 DATA 4F52,4D41,5420,0020,5846,4F52,4D41,5420
992 DATA 0020,2047,4150,5320,0020,2042,4143,4320
993 DATA 2000,1B70,2020,466F,726D,6174,2054,7261

994 DATA 636B,204D,6F64,6520,201B,7100,1B70,2020
 995 DATA 5472,6163,6B3A,0020,2046,6F72,6D61,7461
 996 DATA 6765,203F,2020,3C79,6573,2F6E,6F3E,2020
 997 DATA 1B71,001B,7020,204E,6F74,2066,6F72,6D61
 998 DATA 7474,6564,2020,2020,2020,3C54,6173,7465
 999 DATA 3E20,1B71,0020,2065,6E20,4661,6365,3A00
 1000 DATA 2020,6475,2044,7269,7665,3A00,1B70,2046
 1001 DATA 6F72,6D61,7461,6765,2061,7665,6320,6E6F
 1002 DATA 7576,6561,7578,2047,4150,2065,6E74,7265
 1003 DATA 2070,6973,7465,7320,6574,2073,6563,7465
 1004 DATA 7572,7320,3F20,2020,3C79,6573,2F6E,6F3E
 1005 DATA 201B,7100,1B70,2055,6E20,696E,7374,616E
 1006 DATA 7420,7075,6973,203C,5265,7475,726E,3E20
 1007 DATA 1B71,0000,2020,466F,726D,6174,2054,7261
 1008 DATA 636B,2020,0000,0000,0000,049C,0000,0D78
 1009 DATA 0000,0DF0,0000,0E6B,0000,0E7C,0000,02E4
 1010 DATA 0000,04C9,0000,0DB4,0000,0E2C,0000,0E6B
 1011 DATA 0000,0E7C,0000,02E4,0000,317A,0000,38F4
 1012 DATA 0000,3904,0000,3915,0000,3924,0000,3931
 1013 DATA 2020,4D41,5B54,5241,434B,3A20,3739,2000
 1014 DATA 2020,4D41,5B53,4543,544F,523A,2030,3920
 1015 DATA 0020,2049,4E49,5420,4452,4956,4520,2000
 1016 DATA 2020,534B,4F57,2042,5042,2020,0020,2042
 1017 DATA 4143,4B20,2000,1B70,2020,494E,4954,2044
 1018 DATA 5249,5645,204D,454E,554E,2020,1B71,001B
 1019 DATA 7020,2042,696F,7320,5061,7261,6D65,7465
 1020 DATA 7220,426C,6F63,6B20,6475,2064,7269,7665
 1021 DATA 2061,6374,6966,2020,203C,2052,6574,7572
 1022 DATA 6E20,3E20,201B,7100,1B70,2020,2044,6972
 1023 DATA 6563,746F,7279,2064,8262,7574,6520,656E
 1024 DATA 2046,6163,653A,2030,2054,7261,636B,3A20
 1025 DATA 3120,5365,6374,6F72,3A20,3320,201B,7100
 1026 DATA 1B70,2020,2044,6972,6563,746F,7279,2064
 1027 DATA 8262,7574,6520,656E,2046,6163,653A,2031
 1028 DATA 2054,7261,636B,3A20,3020,5365,6374,6F72
 1029 DATA 3A20,3320,201B,7100,0002,0000,0000,0000
 1030 DATA 0000,0003,0000,0000,0000,263C,0000,266A
 1031 DATA 0000,269E,0000,26C6,0000,26F6,0000,2694
 1032 DATA 0000,03C2,0000,2772,0000,2700,0000,27CE
 1033 DATA 0000,27FC,0000,282C,0000,294E,0000,03C2

1034 DATA 0000,3A60,0000,3A6E,0000,3A7E,0000,3A81
1035 DATA 0000,3AEC,0000,3A9B,0000,3AA9,2047,4150
1036 DATA 313A,2036,3020,0020,4741,5032,3A20,3132
1037 DATA 2000,2047,4150,333A,2032,3220,0020,4741
1038 DATA 5034,3A20,3430,2000,2047,4150,353A,2036
1039 DATA 3634,2000,2042,7974,652F,5365,633A,2030
1040 DATA 3531,3220,0020,4241,434B,2000,1B70,2020
1041 DATA 4472,6976,6520,466F,726D,6174,204D,6F64
1042 DATA 6520,201B,7100,1B70,2020,436B,616E,6765
1043 DATA 2047,6170,7320,656E,7472,6520,6C65,7320
1044 DATA 7365,6374,6575,7273,201B,7100,1B70,2020
1045 DATA 556E,2069,6E73,7461,6E74,2070,7569,7320
1046 DATA 3C52,6574,7572,6E3E,201B,7100,1B70,2020
1047 DATA 5345,4354,4F52,204D,4F44,4520,201B,7100
1048 DATA 0290,003C,000C,0016,002B,0298,1B70,2054
1049 DATA 7261,636B,3A20,2020,4661,6365,3A20,2053
1050 DATA 6563,746F,723A,2020,4279,7465,733A,2020
1051 DATA 436B,6563,7375,6D2B,6B65,7B29,201B,7100
1052 DATA 1B4A,001B,4B00,1B70,001B,7100,1B59,2121
1053 DATA 001B,4B00,1B45,001B,4100,1B42,001B,4C00
1054 DATA 1B6C,001B,7700,1B66,001B,6500,2020,2020
1055 DATA 2020,2020,2020,2000,1B4A,0000,0000,3C14
1056 DATA 0000,3C2C,0000,3C59,0000,3C62,0000,3C6B
1057 DATA 0000,3C75,0000,3C7F,0000,3CA0,0000,3CAA
1058 DATA 0000,3CD0,0000,3CDA,0000,3CD4,0000,3CDE
1059 DATA 0000,3CE9,0000,3D12,0000,3D1D,0000,3D2B
1060 DATA 0000,3D32,0000,3D3D,0000,3D4B,0000,3D5E
1061 DATA 0000,3D69,0000,3D74,0000,3D7F,0000,3D8A
1062 DATA 0000,3D95,0000,3DAC,0000,3DAB,0000,3DB6
1063 DATA 1B70,2050,4153,2064,6520,424F,4F54,5345
1064 DATA 4354,4F52,201B,7100,1B70,2044,6266,6175
1065 DATA 7420,6461,6E73,206C,6520,7365,6374,6575
1066 DATA 7220,6465,2044,4952,203C,5265,7475,726E
1067 DATA 3E20,1B71,0020,6572,7265,7572,3300,2065
1068 DATA 7272,6575,7234,0020,6572,7265,7572,3520
1069 DATA 0020,6572,7265,7572,3620,001B,7020,4469
1070 DATA 7371,7565,2061,6273,656E,742F,7061,7320
1071 DATA 6465,2070,6973,7465,201B,7100,2065,7272
1072 DATA 6575,723B,2000,1B70,2050,6173,2064,6520
1073 DATA 7365,6374,6575,7220,211B,7100,2065,7272

1074 DATA 6575,7231,3000,2065,7272,6575,7231,3100
 1075 DATA 2065,7272,6575,7231,3200,2065,7272,6575
 1076 DATA 7231,3320,001B,7020,204C,6520,6469,7371
 1077 DATA 7565,2065,7374,2070,726F,7482,6782,2065
 1078 DATA 6520,8263,7269,7475,7265,201B,7100,2065
 1079 DATA 7272,6575,7231,3520,0020,6572,7265,7572
 1080 DATA 3135,2000,2065,7272,6575,7231,3700,2065
 1081 DATA 7272,6575,7231,3820,0020,6572,7265,7572
 1082 DATA 3139,2000,1970,2050,6C75,7320,6465,2043
 1083 DATA 6C75,7374,6572,201B,7100,2065,7272,6575
 1084 DATA 7232,3120,0020,6572,7265,7572,3232,2000
 1085 DATA 2065,7272,6575,7232,3320,0020,6572,7265
 1086 DATA 7572,3234,2000,2065,7272,6575,7232,3520
 1087 DATA 0020,6572,7265,7572,3236,2000,2065,7272
 1088 DATA 6575,7232,3720,0020,6572,7265,7572,3238
 1089 DATA 2000,2065,7272,6575,7232,3920,0000,FFFF
 1090 DATA 0000,0074,0B08,0B0C,0B08,0B08,0B08,0B06
 1091 DATA 0410,0B0A,0C0E,0A0C,0B0A,5404,0A04,0A0A
 1092 DATA 0B04,0C08,0B0C,1006,1208,2028,0A06,0406
 1093 DATA 0406,0410,040A,0A06,0406,040E,100A,0604
 1094 DATA 0604,0604,0E10,0406,0406,0406,040A,0A0E
 1095 DATA 180A,0604,0604,0604,0A14,040A,0A06,0406
 1096 DATA 040E,1004,0A0A,0604,0604,0E10,0A06,0406
 1097 DATA 0406,040E,0C06,100A,0C10,060A,0C16,0A0C
 1098 DATA 160A,0C06,1010,0C0C,1006,100C,0C06,1010
 1099 DATA 0C0C,1006,100C,0C16,0606,0608,240E,0604
 1100 DATA 0B08,0B08,060E,0B08,1406,044A,0C0A,0A0A
 1101 DATA 0A0A,060E,0A0A,0A0A,0C0B,0616,1E08,040C
 1102 DATA 3014,0B10,0C0B,0E0E,0B0A,0B0B,0604,1212
 1103 DATA 1C04,0B0B,1008,0E0E,0B0A,1006,100A,060C
 1104 DATA 480E,0612,0E06,160E,060A,0A06,040A,040B
 1105 DATA 0C0A,0A06,040A,040B,0C16,0E0E,0A0A,0604
 1106 DATA 0A04,0B0E,0A06,040A,040B,0A14,0B16,0B0A
 1107 DATA 0A16,1E14,0606,060B,2214,1C06,1C06,0606
 1108 DATA 0606,0606,0606,060B,0606,0624,060C,0624
 1109 DATA 1410,0C0C,1410,0C0C,1410,0C0C,1410,0C22
 1110 DATA 0B0A,0C0B,0B0A,0E0A,0B0A,0E0A,0B0A,0E0A
 1111 DATA 0B0A,0E0A,0B0A,0E0A,0B0A,0E0A,0B0A,0E0A
 1112 DATA 0B0A,0E0A,0B0A,0E0A,0B0A,060C,1E3E,263C
 1113 DATA 4424,0606,0606,0A0B,2E10,0610,100C,0C10

1114 DATA 0510,0C0E,0E0B,0605,1406,040B,0B0B,0A1A
1115 DATA 1E06,040B,0B0B,0B0A,160C,0E5C,0E0A,060C
1116 DATA 0B0B,0A0A,0A12,0610,0A06,0C0B,0B0A,0A0A
1117 DATA 1C0E,0E12,1E06,060A,0B22,321A,640E,104A
1118 DATA 062E,0606,0A32,200A,0A06,161B,1A3A,362E
1119 DATA 0C4B,0604,0B0B,0E0B,200A,0E12,723E,0B0A
1120 DATA 0C0B,0604,163E,1006,1012,120C,2E0B,0C10
1121 DATA 1212,0C0C,050B,120A,200C,0622,1B0E,0B0A
1122 DATA 0A16,161E,060B,060B,0626,241E,060B,060E
1123 DATA 0C1A,0E0C,0616,0610,220A,0610,0E06,100C
1124 DATA 0A0B,0B06,040B,0610,0B0A,220B,0A0C,0A0B
1125 DATA 0612,044A,0E0E,0B1B,040B,060C,0E0B,1E0C
1126 DATA 0E10,0E0B,1C0E,0C0C,0632,1006,160B,0A14
1127 DATA 0B06,040B,060B,060C,1E06,040A,0622,0E06
1128 DATA 4210,1010,0616,0B0A,0E0B,0B06,4010,0E10
1129 DATA 1216,1E0E,2C0C,2C0C,0C0C,0C0C,1B0A,0A0A
1130 DATA 0A0A,0A22,1E06,0606,0B22,1A06,060E,1E0B
1131 DATA 0B0B,340C,1A10,0C0B,060A,1B7A,622C,200B
1132 DATA 0E0C,0C0B,0E0C,0C0B,0E0C,0C0A,0E0C,2C12
1133 DATA 1012,0C1C,0B0E,0C0C,0B0E,0C0C,0B0E,0E0C
1134 DATA 0A0E,0C2C,0E10,120C,0C1E,0B0B,0B0E,0B0B
1135 DATA 0B0E,0B0B,0E0E,0B0B,0B06,0C1E,0B0B,0B0E
1136 DATA 0B0B,0B0E,0B0B,0E0E,0B0B,0B06,0C24,202C
1137 DATA 1A22,1A12,1414,1414,0C0E,1E1E,1E14,1416
1138 DATA 0E12,0B0A,0624,0620,060B,040C,0B0B,0B0C
1139 DATA 0604,1606,0414,0B10,0B0E,5C04,0B0A,040C
1140 DATA 0B0C,0614,040B,1204,0C0C,0C0C,0C0C,1E0C
1141 DATA 0C0E,0B0C,0B0A,020C,0C0C,044A,2030,0B2C
1142 DATA 0E42,0404,0404,0404,0404,0404,0404,04F0
1143 DATA 0404,0404,0404,0404,0404,0404,0404,0404
1144 DATA 0404,0404,0404,04E0,0404,0404,0404,0404
1145 DATA 0404,0404,0404,0404,0404,0404,0404,04A0
1146 DATA 0404,0404,0404,0404,0404,0404,0404,0404
1147 DATA 0424,0404,0404,0404,0404,0404,0404,0404
1148 DATA 0404,0404,0404,0404,0401,0101,2204,0404
1149 DATA 0404,0404,0404,0404,0404,0404,0404,0404
1150 DATA 0404,0404,0120,0404,0404,0404,0404,0404
1151 DATA 0404,0404,0404,0401,1E04,0404,0404,0404
1152 DATA 0404,0404,0404,0404,0404,0404,0401,4604
1153 DATA 0404,0404,0404,0404,0404,0404,0404,0404
1154 DATA 0404,0404,0404,0404,0404,0400

ANNEXE 2.

Tableau des codes ASCII

Le tableau qui suit vous montre les codes ASCII de l'ATARI ST. Vous obtiendrez la valeur correspondant à un caractère en concaténant le chiffre hexadécimal de la bordure supérieure au chiffre hexadécimal de la bordure gauche (valeur obtenue par la fonction BASIC ASC("")). Ainsi, la valeur du caractère "A" est \$41.

Ce tableau a été réalisé avec le programme suivant en GFA BASIC. Le programme vous permettra d'imprimer le tableau vous-même (Hardcopy) :

```

Cls
For I=0 To 15
  Print At(I*3+7,3);Hex$(I)
  Print At(4,I+4);Hex$(I)
  For J=0 To 15
    Print At(I*3+7,J+4);Chr$(I*16+J);
  Next J
Next I
DefText 1,0,0,4
Text 42,170,"Bell"
Text 46,218,"LF"
Text 46,266,"CR"
Text 70,234,"ESC"
Text 92,56,"SPC"
For I=1 To 18
  Draw 18,I*16+15 To 424,I*16+15
  Draw I*24-7,31 To I*24-7,303
Next I
Draw 17,31 To 40,47
Repeat
Until Mousek

```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0			SPC	0	@	P	`	p	ç	é	á	ã	ij	o	α	≡
1			!	1	A	Q	a	q	ü	æ	í	õ	j	u	β	±
2			"	2	B	R	b	r	é	ë	ó	ø	κ	g	Γ	≥
3			#	3	C	S	c	s	â	ô	ú	ø	ι	χ	π	≤
4			\$	4	D	T	d	t	ä	ö	ñ	œ	λ	η	Σ	∫
5			%	5	E	U	e	u	ä	ö	ñ	œ	τ	γ	σ	∫
6			&	6	F	V	f	v	ä	ö	ñ	œ	π	μ	μ	÷
7	bell		'	7	G	W	g	w	ç	ù	o	ñ	ι	π	τ	≈
8			(8	H	X	h	x	ê	ü	¿	ö	τ	ι	ö	°
9)	9	I	Y	i	y	ë	ö	¿	ö	π	γ	θ	•
A	LF		*	:	J	Z	j	z	è	ü	¿	ö	π	γ	Ω	•
B		ESC	+	;	K	I	k	{	ï	ç	½	†	°	γ	δ	√
C			,	<	L	\	l		î	£	¼	¶	γ	φ	φ	π
D	CR		-	=	M]	m	}	ì	¥	¡	©	¿	§	φ	²
E			.	>	N	^	n	~	ÿ	β	«	©	ñ	Λ	E	³
F			/	?	O	_	o	Δ	ÿ	f	»	©	J	∞	Π	⁻

ANNEXE 3.**Index****A**

Accessoire	6.1
ASCII	2, Annexe 1
Attribut de fichier	3.3

B

Base de données	2.6
Bloc de commandes HDC	5.1
Boot secteur	3.2
Bootage	3.2
BPB, BIOS Parameter Bloc	3.2
Bss	3.5.2

C

Catalogue	3.3, 8.2.2
Champ de données	2
Chargeur BASIC	8.4
Cluster	3.1
Contrôleur	4.2.2, 5.1.1, 8.3

D

Data	3.5.2
Directory	3.3, 8.2.2
Disque dur	5
DMA	4.2.1, 5.1
Driver de Harddisk	5.2
DTA	3.3

E

Enregistrement	2
Extension	2.1

F

FAT, File Allocation Table	3.4
Fichier	2
Fichier séquentiel	2
Fichier séquentiel indexé	2
Fichier à accès direct	2
File-Header	3.5.1
FLOCK	5.1.1
Fonctions C	2.4
Fonctions FORTRAN	2.5
Fonctions PASCAL	2.3
Fonctions TOS/GEMDOS	2.1, 8.2.1
Formatage	3.1, 3.2, 7.3, 8.2, 8.3

G

GEMDOS	2.1, 8.2.1
--------	------------

H

Handle	2.1
Harddisk	5
HDC	5.1
Heure/Date	3.3, 8.2.7

I

Instruction BASIC pour disquettes	2.2
Interface SHUGART	4.2.3
Interleave	5.1.1

L

Lecteurs non ATARI	4.3
--------------------	-----

N

NM68	3.5.2
Nom de fichier	3.3

P

Partition	3.6, 5.1
Pistes	3.1, 4.1

R

RAM Disk	6
RANDOM ACCESS	2
Recherche de données	2.6, 8.2.5
Relocation	3.5.2

S

Secteurs	3.1, 4.1
Shipping position	5.1.1
SHUGART	4.2.3

T

Table symbolique	3.5.2
Text, data, bss	3.5.2
Tracks	3.1, 4.1
Tri de fichiers	2.6, 8.2.5
Tête de lecture/écriture	4.1, 5.1

Achevé d'imprimer
sur les presses de l'imprimerie IBP
à Rungis (Val-de-Marne 94) (1) 46.86.73.54
Dépôt légal - Février 1989
N° d'impression: 5093

ATARI ST

BRAUN / DITTRICH / SCHRAMM

DISQUETTE ET DISQUE DUR

Débutants ou utilisateurs confirmés, tirez le maximum de votre lecteur de disquette ou disque dur. Grâce à cet ouvrage vous saurez tout sur la structure d'une disquette, l'emplacement des programmes, l'accès direct aux données par le biais du langage machine ou du Basic. Enfin, profitez des nombreux programmes utilitaires fournis : copie, formatages spéciaux...

Principaux sujets traités.

- Les boots secteurs et le BIOS parameter bloc.
- Formatage par programme de la disquette et du disque dur.
- Structure des fichiers.
- RAM disk et programme de copie de la disquette vers le RAM disk.
- Interface Basic TOS.
- Formatage non standard des disquettes.
- Accès au contrôleur des disquettes en Basic.
- Moniteur de disque.



9 782868 990877

REF. ML 172 / PRIX 179 F.

213 / ISBN 2-86899-087-8 / ISSN 0980.1928

EDITIONS MICRO APPLICATION

58, RUE DU FAUBOURG-POISSONNIÈRE
75010 PARIS. TÉL. : (1) 47 70 32 44