Tutoriel	**Lotus Esprit Turbo Challenge**
Protection	**MFM (multifile)**
Auteur Original	**Musashi9**
Soumit par (sur flashtro.com)	**Musashi9 [2005-19]**
Version	**24/06/2018 Gi@nts**
Vérification/Correction	**V1.2, Testé et fonctionnel de A à Z**

* LOTUS ESPRIT TURBO CHALLENGE *
CRACK TUTORIEL

Table des matières

Matériels nécessaires

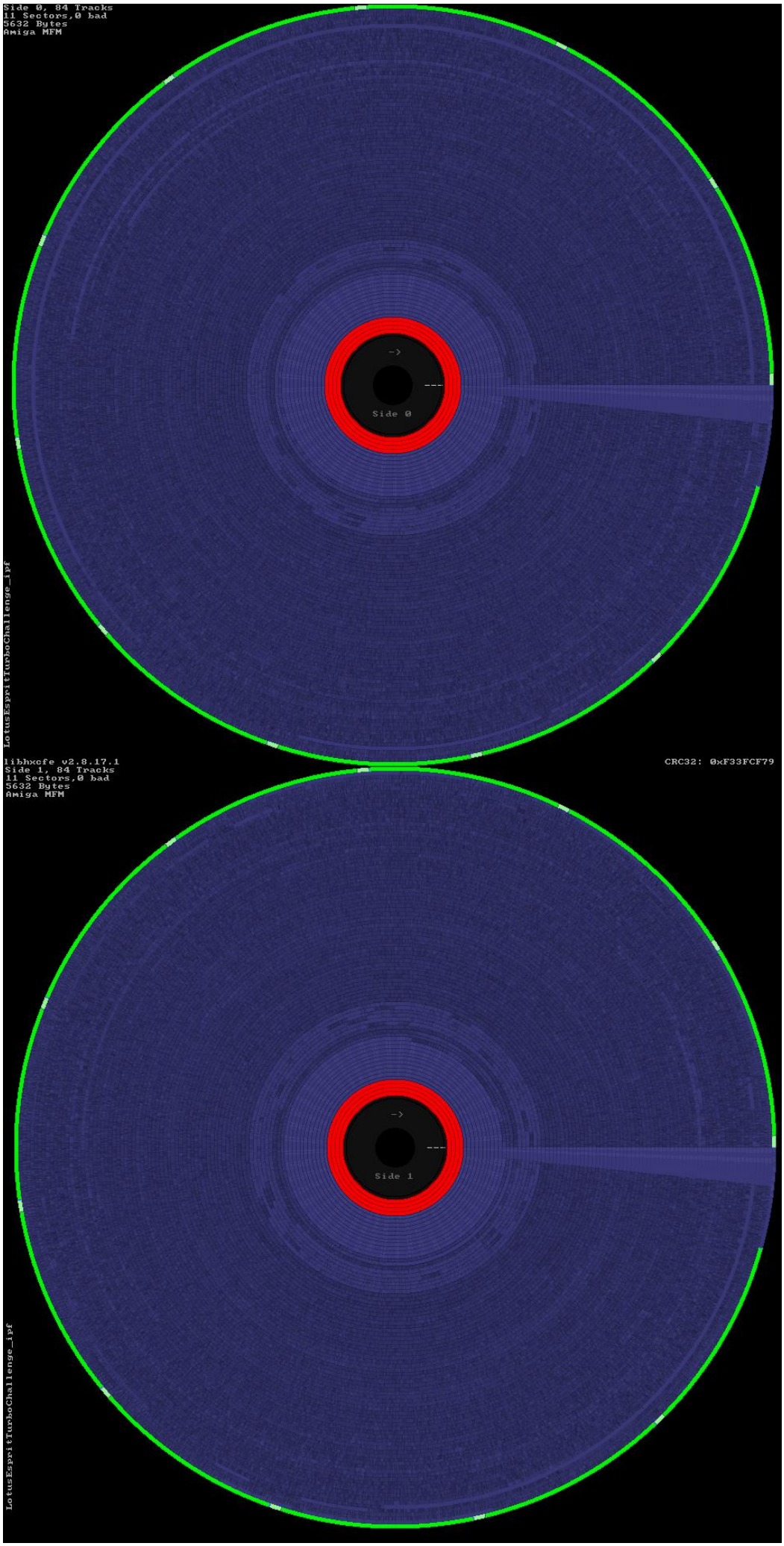
1) Un AMIGA avec une extension mémoire de 512K ou l'émulateur WINUAE
2) Un lecteur externe en plus est fortement recommandé.
3) Une Carte ACTION REPLAY (ou ça ROM Image) selon configuration utilisé.
4) Le jeu Original Lotus Esprit Turbo Challenge ou son image CAPS française (SPS 0774)
5) Le logiciel Xcopy-pro en disquette ou son image disk.
6) L'assembleur ASM-One
7) Les connaissances pour utiliser un l'assembleur ci-dessus !

Général Info

Ce tutoriel Français est basé sur le tutoriel original mis en ligne sur le site flashtro.com.
Ce document n'est pas une traduction mot par mot de celui-ci mais plus une nouvelle version.
Suivit pas à pas avec des nouvelles informations et parfois une nouvelle approche mais toujours basé sur la trame de l'original.

Bon Tuto.

Gi@nts


Side 0, 84 Tracks
11 Sectors,0 bad
5632 Bytes
Amiga MFM
->

Side 0
LotusEspritTurboChallenge_ipf
libhxcfe v2.8.17.1
Side 1, 84 Tracks
11 Sectors,0 bad
5632 Bytes
Amiga MFM
CRC32: 0xF33FCF79
->

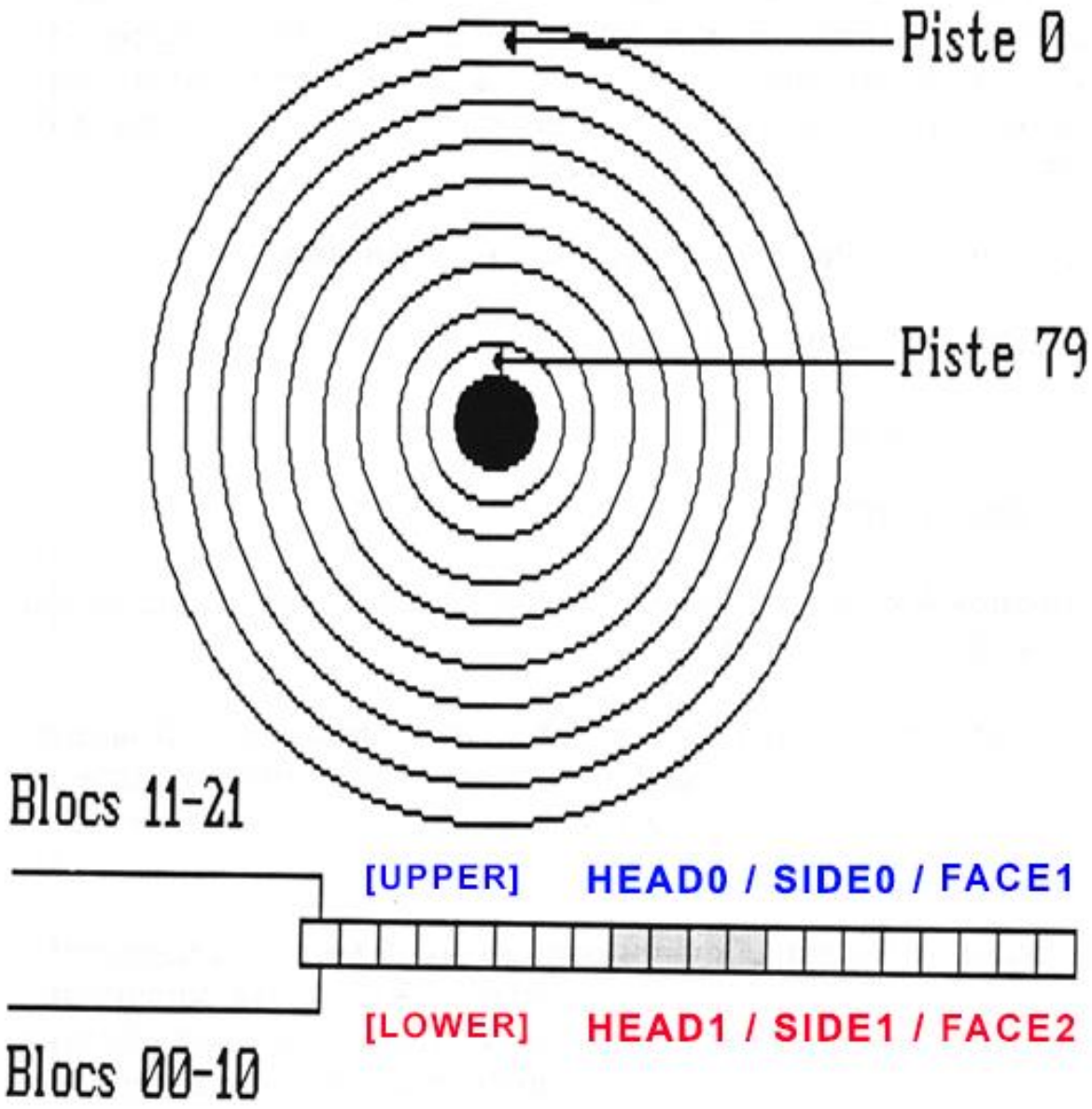
Side 1
LotusEspritTurboChallenge_ipf

Agencement des disquettes Amiga v1.2

En France :

On utilise des termes comme : *piste, bloc, secteur, face*…
Piste : 0 à 79 Certaine disquette pousse jusqu'à 81 voire 82 pistes mais le standard reste quand même 80 pistes (de 0 à 79)
Face : 0/1 ou **1/2** ou **A/B**, Dessus ou dessous tout simplement. Sur Amiga nous avons deux faces utilisées sur 99% des jeux.

Chaque piste, pour un format standard 'AmigaDOS' est composée de plusieurs *bloc ou secteur*, en général 11 **par face**.
Le terme piste peut désigner l'ensemble d'une piste (les deux 'side' du disque), ou uniquement une 'side' d'une piste.
Une piste standard *amigados* est découpée en plusieurs partie appelé **bloc**, **secteur**, **sector**.



Dans d'autre pays :

On utilisera d'autre terme, comme **sector, keys, tracks, cylindre**, **head**…

Le terme **track** par exemple que l'on aurait vite fait de traduire 'piste' ne colle pas forcément à notre description française.
En général, le terme **tracks** désigne toujours une position sur la disquette mais **elle va de 0 à 159** (soit 160 **tracks**)
Le maximum étant 160 et non 80 car on a deux faces bien sûres, en fait, elle correspond à une piste sur une face.

Il peut néanmoins arriver que l'on utilise dans des tuto anglo-saxon le terme *tracks* dans le sens 'piste' en français
(donc de 0 à 79 et non de 0 à 159). Mais en règle générale, il a plutôt une plage de 0 à 160.

C'est le terme **cylindre** qui 'colle' plus à notre définition française de **piste.**
En effet, il est courant d'utiliser le terme **cylindre** pour désigner une position sur la disquette de 0 à 79.

Le terme **sector** ou **key** quant à lui corresponds au terme français **bloc** ou **secteur.**

Sur une disquette au format ***Amigados***, nous avons 880ko et nous avons 11 secteurs par face, par piste.

La taille d'une piste ayant une valeur physiquement maximum.
Le nombre maximum de **sector** sur une piste dépends assez logiquement de la taille de ses **sectors.**

Pref…beaucoup de terme qui ne sont pas forcément utilisé dans leur sens propre, le mieux est de lire un tuto et de comprendre quel sens l'auteur a voulu leurs donner.

Il existe aussi un autre type d'appellation utilisé par exemple par le logiciel **MFM-Warp** de Ferox*
**C'est un programme qui scan le disque en bas niveau et essaye d'en réaliser une copie.*

Track	Calcul	Résultat	Format utilisé sous MFMWarp
0	0/2	0 et pair	0.0
1	1/2	0 et impair	0.1
2	2/2	1 et pair	1.0
3	3/2	1 et impair	1.1
156	156/2	78 et pair	78.0
157	157/2	78 et impair	78.1
158	158/2	79 et pair	79.0
159	159/2	79 et impair	79.1

On notera que :

Le premier secteur (secteur 0) appelé aussi *bootbloc* commence sur la *lowerSide* en piste 00
et se fini en piste 79 sur le *upperside*

En *tracks* c'est le même système sauf que l'on terminera en Track 179 et non 79.

La piste Zero est celle situé le plus à l'extérieure du disque.

Le 1er secteur logique, donc le premier bloc sur la disquette, se trouve **piste 0 secteur 0**
Les *bloc* se suivent physiquement mais ne sont pas forcément ordonnée, on parle aussi d'entrelacement.

Le bloc 11 (si on part de 0 bien sur) n'est pas le 1er secteur de la seconde piste mais le 1er secteur *de la face suivante.*
(voir image ci-dessus)

En format **Amigados**, **la taille d'un secteur est de 512 octets**
Ce qui nous donnes comme taille disponible : 512*11 secteurs*80 pistes*2 faces = 901 120 octets soit 880Ko
Une 'track' AmigaDos a une taille de 512 * 11 = **5632** en décimal soit ***$1600*** *octets*

Mise en application sous l'AR :

Il existe deux commandes sous l'AR qui permettent de charger sauver des pistes, à savoir : **RT** et **WT**
Elles fonctionnent pareil.
L'une permet la lecture, l'autre l'écriture.

#**RT** alias Read Track. Permet le chargement de donnée située sur la disquette vers la mémoire.
#la première valeur sera la track de **départ** [0 à 159] à indiquer en hexa. **/!\ ne pas confondre avec piste**

#La seconde valeur sera le nbr de demi track à copié à partir de là.

#**WT** alias Write Track. Permet la sauvegarde de donnée située en mémoire vers la disquette.

Exemples :

```
RT 20 1 50000
```
Start Track = $20 et taille à lire = 1
On copiera donc la piste !16 (en décimal) side 0 en mémoire $50000

Oui car **20** est donné en hexa, ce qui nous donnes !32 en décimal **mais** il indique une track (de 0 à 159) **PAS en piste.**
Pour avoir l'équivalent en piste on divisera donc par 2 (car deux faces).

$20/2=$10 = !16 (en décimal donc) et comme il n'y a pas de retenu on est sur la face0.

```
RT 21 2
```
Start Track = $21 et taille à lire = 2
On copiera la piste !16 side 1 et la piste !17 side 0 en mémoire 50000

21 est donné en hexa **donc $21 = !33** en décimal.
33/2 = 16.5 ,donc piste 16 side 1 et comme on continue à lire/copier les donnes (**taille à lire =2**), on continue la copie.
On change donc de track car on est déjà sur la face 1 (il existe que 2 faces sur une disquette)
On arrive donc sur la prochaine track à savoir, piste 17 en side 0 puisque que c'est la première face au niveau structure la side 0.

Les secteurs

On peut aussi adresser un disque avec la notion de secteur.
Comme on l'a vue au-dessus, un disque AmigaDos fait 80 Piste (0-79), 2 faces et 11 secteurs par Piste
Chaque secteur fait 512 octets.
Si on fait le calcul cella nous donnes : 80*2*11=1760 Secteurs
Ainsi on peut avoir une position exacte en secteur sur une disque amiga avec une valeur entre 0 et 1759

Exemple DiskBlock Position 520 correspond au Secteur 03 de la piste 23 sur la face 1

On peut aussi fonctionner en mode **RAW**, directement en adressant en octet, cela dépends du 'trackloader' en question.

Le format MFM

Original source : *Codetapper/Action!*
Original Date : *updated: 11/2/2001*

Les DATA sur les *tracks* d'une disquette AmigaDos sont codées/décodées au format **MFM.**
Chaque *tracks* contient 11 secteurs de 512 octets chacun.
Chaque *secteur* à un *header* qui nous donnes le numéro de track, le numéro de secteur et d'autres données.

Le contenu d'un disque normal AmigaDos est le suivant :

+ Gap A normalement une valeur de **00** octet soit $AAAA au format MFM.
+ Track Contient normalement 11 secteurs qui sont :

- S E C T O R -

00

2 octets
00 Octet soit **$AAAA** au format **MFM**

Sync

2 octets
(**A1**) converti au format **MFM** et 'Clock pulse' n'est pas pris en compte
Donc le résultat nous donnes : **$4489** qui est le 'SyncWord'.
Aucune DATA ne sera jamais convertie dans ce pattern. (en **MFM**)

SectorHeader

1 longWord
Header du secteur

Format	Track	Sector	Length
Amiga 1.0 format : **$FF** **1 octet**	Numéro de track **1 octet**	Numéro de secteur **1 octet**	Nombre de secteur avant Gap de fin **1 octet**

Fill

16 octets
Zone destinée pour l'AmigaDos OS 'Recovery' mais jamais utilisée.
donc zone remplie de Zero.

HeaderChecksum

1 longWord
Checksum de la zone Header.
Il est calculé en utilisant un XOR et ne contient que des 'databits'

DataChecksum

1 longWord
Checksum de la zone Data.
Il est calculé en utilisant un XOR et ne contient que des 'databits'

Data

512 octets
Block de DATA

Noter qu'il n'y a pas de **GAP** entre chaque secteur.

La conversion en **MFM** est faite selon le principe suivant :
Prenez 2 bits de Data et ajouter 1 de Clock entre les deux.
Le bit de Clock est à **1** si les 2 bits de Data sont à 0 sinon, le bit de Clock est à **0**
Ce système permet de ne pas avoir de longue série de 0 ou de 1 qui se suivent.

Un Exemple :

```
Bit de Data   :   0 0 0 1 1 0 1 1 ...
Bit de Clock  :  ? 1 1 0 0 0 0 0 0...
ENCODAGE MFM  :  ?0101001010001010...
```

Chaque octet est converti en word.
Comme l'extension d'octets est assez compliqué à réaliser sur Amiga, les données sont d'abords divisées en deux moitiés.
Les impaires et les paires.
Les premiers bits à être convertis sont les bits pairs et ensuite les bits impaires.

```
Binaire        : 01001110
Bits pairs     : 0 0 1 1
bits impaires  :  1 0 1 0
```

Cela permet un traitement des données plus rapides car l'extraction des moitiés paires et impaires peuvent être réaliser facilement via des opération logiques '**AND**'

```
Moitié pairs   : DATA 'AND' 0xAAAA
Moitié impairs : DATA 'AND' 0x5555
```

Pour ripper les Data d'un disque vous devez en premier trouver le `'SyncWord'` qui est l'endroit ou démarre-les **Data** sur le **Track.**
Le `'SyncWord'` est en fait marqueur.

Il existe deux registres, CIA-A et CIA-B que vous devez comprendre et apprendre par cœur.
Une fois que vous savez comment fonctionne $BFD100 et $BFD001, vous serez capable de décoder la majorité des loaders.

La plupart des *TrackLoader* n'ont pas de compteur de track pour savoir qu'elle track la tête de lecture à terminé de lire ou écrire.

$BFE001 PRA *Peripheral Data Register for Data Port A :: Status: Read/Write. Chip:* ***CIA-A***

Bit 0: OVL: Bit de 'Memory Overlay' , toujours à 0. Ne change pas.

Bit 1: LED: Bit de 'Power Led/cutoff filter'

- **Valeur1** Led Lecteur diminué et 'cutoff filter' inactivé
- **Valeur0** Led lecteur pleine puissance et 'cutoff filter' activé

Bit 2: CHNG: Changement de disque (1 = aucun changement effectuée, 0 = changement effectué)
Bit 3: WPRO: Disque protégé en écriture (1 = pas protégé ; 0 = protégé)
Bit 4: TK0: Disque track Zero (1 = pas sur track ;0 = positionné sur track 0)
Bit 5: **RDY**: Disque prêt (1 = pas prêt; 0 = prêt)
Bit 6: FIR0: Bouton Fire port1 (1 = pas appuyé; 0 = appuyé)
Bit 7: DIR1: Bouton Fire port2 (1 = pas appuyé; 0 = appuyé)

$BFD100 PRB *Peripheral Data Register for Data Port B :: Status: Read/Write. Chip:* ***CIA-B***

Bit 0: STEP: Déplace la tête du lecteur d'une track dans une direction.
DIR bit (mis à 1, puis 0, et encore à 1 pour déplacer la tête)
Bit 1: DIR: Direction to move drive head (1 =vers l'extérieur, 0 =vers l'intérieur)
Bit 2: SIDE: Sélection de la tête du lecteur (1 = bas ; 0 = haut)
Bit 3: SEL0: Sélection DF0: (1 = pas sélectionné; 0 = sélectionné)
Bit 4: SEL1: Sélection DF1: (1 = pas sélectionné; 0 = sélectionné)
Bit 5: **SEL2**: Sélection DF2: (1 = pas sélectionné; 0 = sélectionné)
Bit 6: SEL3: Sélection DF3: (1 = pas sélectionné; 0 = sélectionné)
Bit 7: MTR: Motor on-off status (1 = motor off; 0 = motor on)

Bit 0: Ce bit contrôle le déplacement de la tête de tous les lecteur sélectionnés.
Pour déplacer la tête, vous devez basculer la valeur de ce bit de 1 à 0 puis, de revenir à 1.
Cette opération déplace la tête d'une distance d'une **Track**
Avant de déplacer la tête du lecteur, vous devez sélectionner une direction '**Bit1**'

Après le déplacement de la tête de lecture il est important d'attendre 3ms avant de faire une nouvelle action avec le lecteur de disquette. Comme les boucles de synchro logiciel ne sont pas précise (dépends de la vitesse d'horloge de l'ordinateur qui varie d'un ordinateur à l'autre), il est recommandé d'utiliser un des timers des chipset **CIA** pour attendre les 3ms nécessaire.

Bit 1: La valeur de ce bit détermine la direction tête sur les lecteurs de disquette sélectionné.

- **Valeur1** Vers l'extérieur en direction de la piste 0
- **Valeur0** Vers l'intérieur en direction de la piste 79

Pour être sûr de la direction de déplacement effectué, positionné d'abord ce bit à 0 et n'essayer jamais de déplacer une tête de lecteur plus loin que la piste 79 ou avant la piste 0.
Vous pouvez vérifier si la tête du lecteur sélectionné est sur la **Track** 0 en lisant le bit 4 du **CIA-A** située à l'adresse : **$BFE001**

Bit 2: Les lecteurs de disquettes amiga sont double faces. Cela veut dire que les lecteurs doivent avoir 2 têtes.
Une tête de lecture/écriture pour la face du haut, et une autre tête pour la face du bas.
Ce bit détermine quel tête du lecteur doit être utilisé quand une opération de lecture ou d'écriture est demandée.

- **Valeur1** Sélection de la tête du bas
- **Valeur0** Sélection de la tête du haut

La valeur de ce bit affecte seulement le(s) lecteur(s) sélectionné(s).

Bit 3-6: Ces 4 bits permettent de sélectionner quel lecteur(s) de disquette est utilisé(s).
Seulement les lecteurs sélectionnés sont affectés par les valeurs stocké dans les registres précédent.
Le Hardware de l'amiga permet de supporter quatre lecteur de disquettes 3p1/2.
Sur l'Amiga500 et 1000, le lecteur de disquette interne est connu sous le nom de **driver 0** (DF0)
Les lecteurs de disquettes externes sont connectés en séries.
Le lecteur connectée directement a l'ordinateur est le **drive 1** (DF1)
Le lecteur connectée au drive 1 est le **drive 2** (DF2) et le lecteur connecté au drive 2 est le **drive 3** (DF3)
Sur les ordinateurs Amiga 2000, 2500 et 3000, les deux lecteurs de disquette interne sont les **drive 0** et **1**
Le premier lecteur de disquette externe est le **drive 2** (DF2), même si un seul des lecteur de disquette interne est connecté. N'importe quel lecteur de disquette connecté à ce lecteur de disquette externe sera le drive 3.

Pour sélectionner un lecteur de disquette l'on doit positionner son bit correspondant à 0
Pour désélectionner un lecteur de disquette l'on doit positionner son bit correspondant à 1
Bit 3 drive 0 (**DF0**)
Bit 4 drive 1 (**DF1**)
Bit 5 drive 2 (**DF2)**
Bit 6 drive 3 (**DF3**)

Toutes les combinaisons des lecteurs de disquette peuvent être sélectionné à tout moment.
Les autres Bit de ce registre affectent TOUS les lecteur sélectionnés. Il est donc possible de réaliser des taches simultanément comme le déplacement de tête sur plus d'un lecteur de disquette.

Bit 7: Ce Bit active ou pas le moteur du lecteur sélectionné.

- **Valeur1** Moteur OFF
- **Valeur0** Moteur ON

L'état ON/OFF peut être visualisé par la petit lumière présente sur le devant du lecteur de disquette.
Ce Bit doit être défini avant de choisir un lecteur. Si un lecteur est déjà sélectionné et vous désirer changer l'état de son moteur, vous devriez désélectionner le lecteur, définissez le **bit 7**, puis resélectionnez le lecteur souhaité.

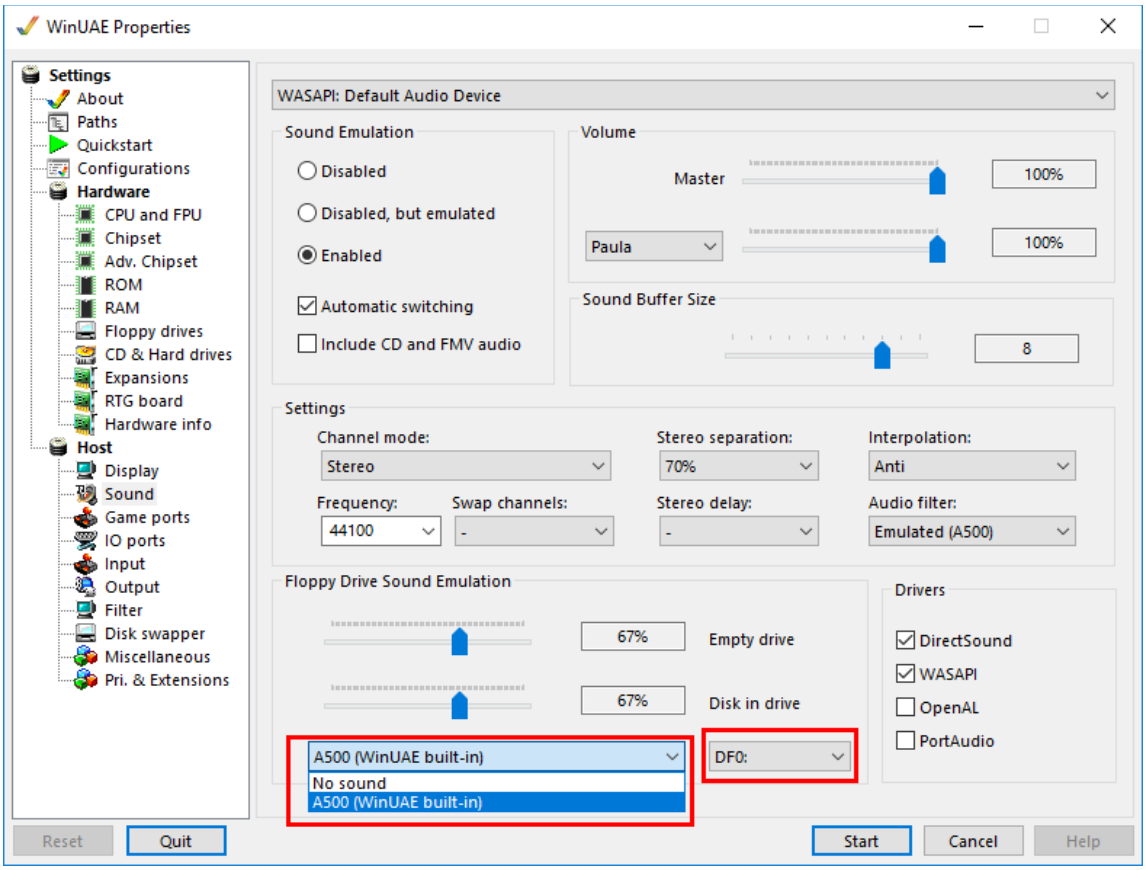
Quand le moteur d'un lecteur est passé à **ON**, vous devez attendre que celui-ci atteigne sa pleine vitesse de rotation avant d'effectuer d'autre opération. Vous pouvez vérifier ceci en lisant le bit 5 du **CIA-A** située à l'adresse : **$BFE001.** Il passe à 0 quand le lecteur a atteint sa pleine vitesse de rotation et que le lecteur est prêt à recevoir de nouvelles commandes.

Le code suivant utilise ce Bit pour passer le moteur a ON sur le drive 0 puis le passer à OFF
Ce programme part bien sur principe que tout le système multitache est désactivé et que vous avez le contrôle total de l'ordinateur.

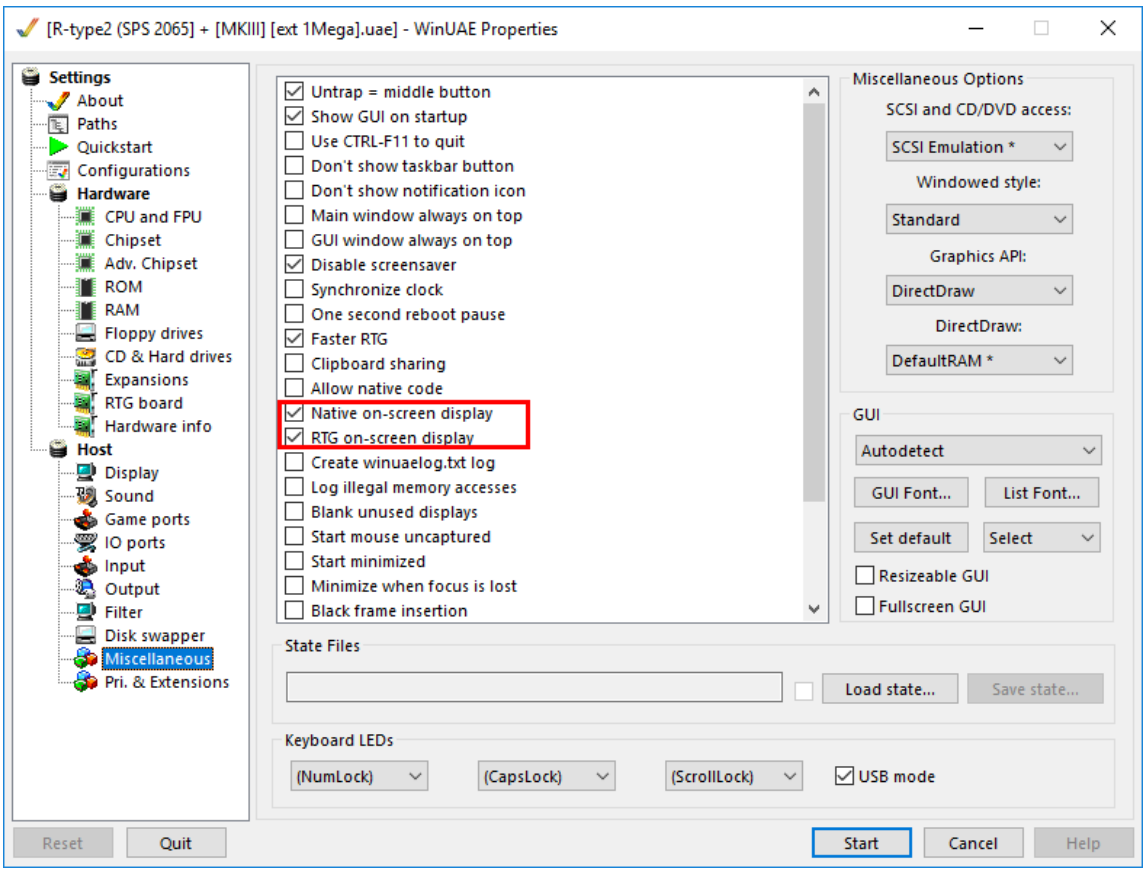
```
CIAAPRA         equ     $BFE001
CIABPRB         equ     $BFD100

                or.b    #$08,CIABPRB    ;S'assure que le lecteur DFO: est désélectionné
                and.b   #S7F,CIABPRB    ;Motor ON en effaçant le bit 7
                and.b   #$F7,CIABPRB    ;Sélectionne DFO: pour passer le moteur à ON

Wait:           btst.b  #5,CIAAPRA      ;Vérifie le Bit Check RDY
                bne.s   Wait            ;On attend que la pleine vitesse de rotation soit atteinte
                or.b    #S88,CIABPRB    ;Motor Off et désélection de DF0:
                and.b   #$F7,CIABPRB    ;Sélectionne DFO: pour passer le moteur à ON
                or.b    #$08,CIABPRB    ;Désélectionne DFO: pour plus de sécurité.
```

WinUAE

Pour ceux qui utilisent ***winUAE*** pour ces tutoriels (j'imagine, la plupart des personnes), Je vous conseille fortement d'activer le son des lecteurs de disquette histoire d'entendre ce que le lecteur effectue comme accès.
HOST -> SOUND -> FLOPPY DRIVE SOUND EMULATION - > DF0 Built-In

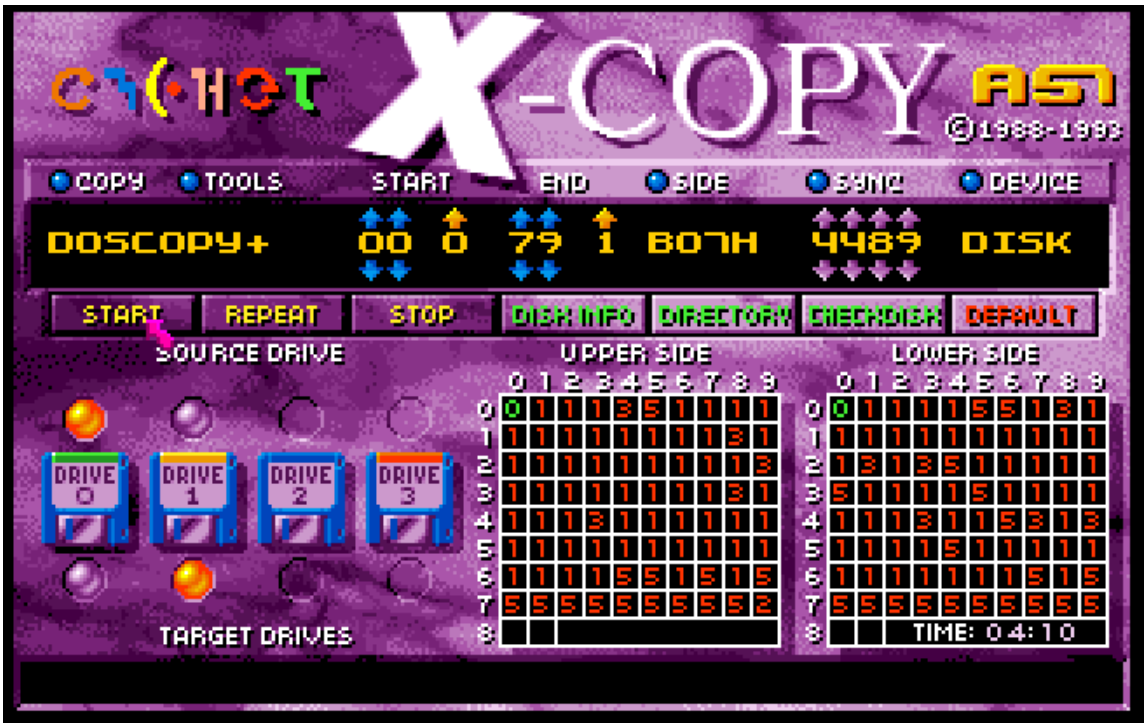


Voir même, pour plus d'information. Par exemple afficher sur qu'elle face l'on se trouve, d'activer :
Host -> Miscellaneous -> Native on-screen display AND RTG on-screen display



Part 1 X-copy

Time to check the protection
On va dans un premier temps essayer de copier la disquettes originale à l'aide de Xcopy Pro.



Bon…Nous avons ici visiblement un format propriétaire sur l'ensemble de la disquette.
Aucune chance bien sur que notre copie fonctionne **néanmoins, gardons sous le coude ce backup.**

Rappel des codes d'erreur de Xcopy :

1. ***Less or more than 11 sectors***
2. ***No sync found***
3. ***No sync after gap found***
4. *Header checksum error*
5. ***Error in header/format long***
6. *Data block checksum error*
7. *Long track*
8. *Verify error*

Part 2 Analyse de l'image IPF

FILENAME	0774_LotusEspritTurboChallenge.ipf
TYPE	Floppy_Disk
ENCODER	CAPS(V1)
FILE	774(V1)
DISK	2
TRACK	00-83
SIDE	0-1
PLATFORM	Amiga
REVOLUTION	5

Comme prévue, un format Standard *AmigaDOS* sur la 1er piste, 11 blocks standard. (sur les deux faces)
Mais ensuite, toutes les pistes sont dans un format non standard avec un '*Data Length*' de 6193 bytes d'un seul bloc

TrackNumber	Size Record (bytes)	Crc	Status	Track Size	Detail Tr. Size	Start Byte	Bit	DataKey	Block	Density	Signal	Encoder	Flag
T00.0	80	72E91FF0	Good	12507	100056 bits = Data=95744 + Gap=4312	304	2435	001	11	Auto	cell_2us	0	None
T00.1	80	8A92E4A4	Good	12503	100024 bits = Data=95744 + Gap=4280	306	2455	002	11	Auto	cell_2us	0	None
T01.0	80	CA8944CF	Good	12509	100072 bits = Data=98432 + Gap=1640	165	1322	003	1	Auto	cell_2us	0	None
T01.1	80	3FC2F60C	Good	12509	100072 bits = Data=98432 + Gap=1640	165	1325	004	1	Auto	cell_2us	0	None

TRACK		Data Length (bytes)		Data (bits)				CRC32 of the complete Extra Data Block			Adress
Data block Description	Sector ID	Data MFM bits	Data bytes	bytes/sector	GAP MFM bits	GAP bytes	Codage	GapDef	DataOff MFM bits	DataOff bytes	Adresse
[T00.0]		6446		51568				D0B835A7			13576-20021
#0	8	8704	545	512	0	1	MFM	0352	0352	15	13576-13607
#1	9	8704	545	512	0	1	MFM	0906	0906	57	13608-13639
#2	10	8704	545	512	0	1	MFM	1460	1460	92	13640-13671
#3	0	8704	545	512	0	1	MFM	2014	2014	126	13672-13703
#4	1	8704	545	512	0	1	MFM	2568	2568	161	13704-13735
#5	2	8704	545	512	0	1	MFM	3122	3122	196	13736-13767
#6	3	8704	545	512	0	1	MFM	3676	3676	230	13768-13799
#7	4	8704	545	512	0	1	MFM	4230	4230	265	13800-13831
#8	5	8704	545	512	0	1	MFM	4784	4784	300	13832-13863
#9	6	8704	545	512	0	1	MFM	5338	5338	334	13864-13895
#10	7	8704	545	512	4312	270	MFM	5892	5892	369	13896-13927
[T00.1]		6446		51568				DD65DF88			20050-26495
#0	0	8704	545	512	0	1	MFM	0352	0352	15	20050-20081
#1	1	8704	545	512	0	1	MFM	0906	0906	57	20082-20113
#2	2	8704	545	512	0	1	MFM	1460	1460	92	20114-20145
#3	3	8704	545	512	0	1	MFM	2014	2014	126	20146-20177
#4	4	8704	545	512	0	1	MFM	2568	2568	161	20178-20209
#5	5	8704	545	512	0	1	MFM	3122	3122	196	20210-20241
#6	6	8704	545	512	0	1	MFM	3676	3676	230	20242-20273
#7	7	8704	545	512	0	1	MFM	4230	4230	265	20274-20305
#8	8	8704	545	512	0	1	MFM	4784	4784	300	20306-20337
#9	9	8704	545	512	0	1	MFM	5338	5338	334	20338-20369
#10	10	8704	545	512	4280	268	MFM	5892	5892	369	20370-20401
[T01.0]		6193		49544				D595110F			26524-32716
#0	N/A	98432	6153	N/A	1640	103	MFM	0032	0032	2	26524-26555
[T01.1]		6193		49544				6E950A9E			32745-38937
#0	N/A	98432	6153	N/A	1640	103	MFM	0032	0032	2	32745-32776
[T02.0]		6193		49544				A53EFA29			38966-45158

Hormis les deux dernières pistes qui ont-elles une '**Data Length**' plus petite, à savoir 59 (piste vide au vue de la taille ?)

[T77.0]		6193		49544				1C3CCC24			972116-978308
#0	N/A	98432	6153	N/A	1632	103	MFM	0032	0032	2	972116-972147
[T77.1]		6193		49544				8C01C403			978337-984529
#0	N/A	98432	6153	N/A	1640	103	MFM	0032	0032	2	978337-978368
[T78.0]		6193		49544				975C4B1B			984558-990750
#0	N/A	98432	6153	N/A	1640	103	MFM	0032	0032	2	984558-984589
[T78.1]		6193		49544				FFEE815D			990779-996971
#0	N/A	98432	6153	N/A	1648	104	MFM	0032	0032	2	990779-990810
[T79.0]		59		472				C05E5AB3			997000-997058
#0	N/A	288	19	N/A	103416	6464	MFM	0032	0032	2	997000-997031
[T79.1]		59		472				B25DB7F4			997087-997145
#0	N/A	288	19	N/A	103776	6487	MFM	0032	0032	2	997087-997118

Part 3 Analyse et modification du bootblock

La seule piste lisible est la piste de *bootblock* au format *AmigaDos*, nous allons donc la charger et regarder de plus près tout ça.
Insérer notre **disquette de backup** préalablement crée avec *Xcopy* dans le lecteur

#RT alias Read Track, permet le chargement de la track 0 à 1 (1ère piste de la face 0)
#D, alias Désassemble
Taper : `RT 0 1 10000` puis `D 10000+c`

```
rt 0 1 50000
Disk ok

d 50000+c
~05000C LEA     00000E00.S,A7
~050010 LEA     5001C(PC),A0
~050014 MOVE.L  A0,00000080.S
~050018 TRAP    #0
~05001A BRA     0005001A
;=====================================
~05001C LEA     00001000.S,A7
~050020 MOVE.W  #0,SR
~050024 LEA     5004A(PC),A0
~050028 LEA     00040000,A1
~05002E LEA     00001000.S,A3
~050032 MOVEA.L A1,A2
~050034 MOVE.W  #FF,D7
~050038 MOVE.W  D7,D6
~05003A MOVE.L  (A0)+,(A1)+
~05003C DBF     D7,0005003A
~050040 MOVE.L  (A2)+,(A3)+
~050042 DBF     D6,00050040
~050046 JMP     00001010.S
;=====================================
~05004A BRA     000500EE
```

+c car le code du bootblock commence à cette adresse, avant c'est la signature AmigaDos du disque.

Regardons ça en détail :

```
05001C LEA      00001000.S,A7     ; On met 1000 dans A7
050020 MOVE.W   #0,SR             ; Mr Propre
050024 LEA      5004A(PC),A0      ; On met 5004A(PC) dans A0, donc le 'après' notre routine.
050028 LEA      00040000,A1       ; On met 40000 dans A1
05002E LEA      00001000.S,A3     ; Et 1000 dans A3
050032 MOVEA.L  A1,A2             ; On copie A1 dans A2
050034 MOVE.W   #FF,D7            ; On met #FF dans D7
050038 MOVE.W   D7,D6             ; On le recopie dans D6
05003A MOVE.L   (A0)+,(A1)+       ; + 1er Boucle de copie de (A0) vers (A1), le 'après notre routine' vers $4000
05003C DBF      D7,0005003A       ;    Décrémenter le compteur D7 et branche en 5003A tant que compteur il a.
050040 MOVE.L   (A2)+,(A3)+       ; + 2em Boucle de copie de (A2) vers (A3) donc de $4000 vers $1000
050042 DBF      D6,00050040       ;    Décrémenter le compteur D6 et branche en 50040 tant que compteur il a.
050046 JMP      00001010.S        ; Et finalement on saute en 1010
```

Comme il est préférable de travailler avec les adresses réelles utilisées dans le jeu et pour un meilleur facilité d'analyse, nous allons booter et voir le code directement en **$10000** puisque c'est là qu'il est copié.

Rebooter votre Amiga et au 1er accès disque, entrer dans l'**AR**
*Environ 1 à 2seconde après le boot.

Taper : `D 1010`

```
d 1010
~001010 BSR     000013A4
~001014 MOVE.W  #7FFF,9A(A6)
~00101A MOVE.W  #7FFF,9C(A6)
~001020 MOVE.W  #4000,24(A6)
~001026 MOVE.W  #20,96(A6)
~00102C MOVE.W  #8210,96(A6)
~001032 MOVE.W  #7F00,9E(A6)
~001038 MOVE.B  #FF,300(A5)
~00103E MOVE.B  #3,1201(A5)
~001044 LEA     1078(PC),A0
~001048 MOVE.L  A0,80(A6)
~00104C BSR     000010A4
~00104E BSR     0000107C
~001050 MOVE.L  #3000,D0
~001056 MOVE.L  #800,D1
~00105C LEA     00000400.S,A0
~001060 BSR     00001110
~001064 MOVE.L  #3800,D0
~00106A MOVE.L  00000BF8.S,D1
~00106E MOVEA.L 00000BFC.S,A0
~001072 MOVE.L  A0,-(A7)
~001074 BRA     00001110
;======================================
~001078 LINEF
```

1er ligne, un **BSR** vers **$13A4**, on regarde ce qu'il y a à cette adresse, **taper :** `D 13A4`

```
d 13A4
~0013A4 LEA     00DFF000,A6
~0013AA LEA     00BFD000,A5
~0013B0 LEA     13B6(PC),A4
~0013B4 RTS
;======================================
```

On regarde ça de plus prêt.
Rapidement un branchement est effectué vers $13A4 qui à son tour positionne **BLTDDAT** en **A6** (DFF000 = BLTDDAT, utilisation blitter).
ainsi que le **CIA-B** en **A5** (BFD000 = CIAB).

CIAB qui rappelons le, avec ces registres **pra** et **prb** permet d'effectuer toute sorte d'opération avec le floppy, il est très souvent utilisé dans une routine *trackloader*.
Sans oublier le **$13B6** (en relatif bien sûr) dans **A4**

$13b6 qui se trouve donc juste après notre sous routine et qui semble ne pas contenir de code.

```
d 13B6
~0013B6 ORI.B   #2,D4
~0013BA BTST    #B,D0
~0013BE BCHG    D7,D6
~0013C0 ORI.B   #0,D0
~0013C4 ORI.B   #0,D0
m 13B6
:0013B6 00 04 00 02 08 00 00 0B 0F 46 00 00 0C 00 00 00 .........F.......
:0013C6 00 00 00 07 48 00 00 07 48 00 00 00 00 00 00 00 ....H...H.......
```

Revenons à nos moutons. Qu'avons-nous donc dans le code en **$1010.**
On peut voir que nous avons 2 branchements vers **$1110** et qu'à chaque fois, les **registres A0**, **D0** et **D1** sont renseignés.

```
~001050 MOVE.L  #3000,D0
~001056 MOVE.L  #800,D1
~00105C LEA     00000400.S,A0
~001060 BSR     00001110
~001064 MOVE.L  #3800,D0
~00106A MOVE.L  00000BF8.S,D1
~00106E MOVEA.L 00000BFC.S,A0
~001072 MOVE.L  A0,-(A7)
~001074 BRA     00001110
```

	1er appel	2nd appel
D0	3000	3800
D1	800	(BF8)
A0	400	(BFC)

Sur le 1er appel est utilisé des valeurs direct, à savoir 3000, 800 et 400
mais dans le second appel, **D1** et **A0** pointe vers une adresse mémoire

Un *trakloader* fonctionne toujours de la même façon, il à besoin d'une position sur la disquette à lire.
Une destination mémoire pour la copie des données et une fin de donnée soit en longueur, soit directement en adresse.
Bien sûr, toutes ces valeurs peuvent être adressées directement, soit via un tableau en mémoire.
Elles peuvent être brut (position Raw sur le disque) ou en secteur, en piste, etc.

On peut voir dans le 1er appel, **D0** à 3000 et **D1** à 800 en adressage direct et dans le second, 3800 en adressage direct.

3800… vous voyez la relation avec les données du 1er appel. (3000, 800 et 400 !)
On peut assez facilement conclure que **D1** va nous indiquer la quantité de donnée à copier.
Que **D0** lui va nous donner une position sur le disque.
Et que **A0** va pointer vers une adresse mémoire.

On va vérifier tout ça.
Il est bien sur impensable d'aller modifier/écrire sur notre disquette original mais comme la 1er piste est au standard *AmigaDos*
Rien ne nous empêche d'utiliser notre copie réalisé sous X-Copy 😊

#A, alias Assemble
Insérez votre disquette de Backup donc dans le lecteur et **taper :** `RT 0 1 50000`
Puis : `A 50046`
`^050046 bra 50046` <RETURN>
`^050048`

```
rt 0 1 50000
Disk ok
d 50046
^050046 JMP     00001010.S
;=====================================

a 50046
^050046 bra 50046
^050048
```

On re-calcul le checksum de boot et on écrit le tout sur la disquette :
#BOOTCHK Alias Boot Check. Permet de calculer un nouveau checksum pour un bootblock.
#WT, alias Write Track. Permet d'écrire une zone mémoire sur la disquette à l'adresse indiqué en cylindre.
Taper : `bootchk 50000` puis `WT 0 1, 50000`

```
bootchk 50000
Old checksum was 22F537F9, now is set to 22F525F3
wt 0 1 50000
Disk ok
```

Et on reboot sur notre disquette de backup !
Le *bootsecteur* est chargé et notre boucle sans fin est atteinte, entrer dans l'**AR**
Taper : `D` (on est bien dans notre **BRA**)

```
d
^00159E BRA     0000159E
;=====================================
^0015A0 MOVE.B  (A0),D0
```

Nous allons bien sur supprimer cette boucle sans fin et continuer le code mais avant, nous allons poser plusieurs *BreakPoints.*
#BS, alias BreakPoint. Permet, dès que l'adresse mémoire indiquée est atteinte, d'effectuer un arrêt du code.
Taper : `BS 1060` et `BS 1064`

Puis : `A 159E`
`^00159E jmp 1010` `<RETURN>`
`^0015A2`

```
d
^00159E BRA     0000159E
:=======================================
^0015A0 MOVE.B  (A0),D0

bs 1060
Breakpoint inserted
Ready.
bs 1064
Breakpoint inserted
Ready.

a 159E
^00159E bra 1010
^0015A2
```

Enlever votre disquette de backup du lecteur de disquette et **insérez la disquette original** en lieu et place.
On retourne au code, **taper :** `X`

Très vite notre *BreakPoint* **$1060** est atteint, on entre automatiquement dans l'**AR**

#O, alias Fill mémory. Permet de remplir une zone mémoire avec une valeur hexa.
On remplit notre zone potentiel de destination avec une petite marge**, taper :** `O "AA", 400-10 400+800+10`
On retourne au code, **taper :** `X`

Après un très rapide accès disque, notre *BreakPoint* **$1064** est atteint, on entre dans l'**AR**
On vérifie le contenue de la mémoire en **400**
#M alias memory read. Permet de voir les données en mémoire au format HEXA/ASCII.
Taper : `M 400-10` puis `M 400+800-10`

```
Breakpoint raised at address: 00001060

o "AA", 400-10 400+800+10
Ready.

x
No known virus in memory!
Ready.
Breakpoint raised at address: 00001064
m 400-10
:0003F0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
:000400 00 00 06 5C 00 0B 37 74 00 00 06 14 00 0B 3D D0 ...\..7t......=.
:000410 00 00 09 A4 00 0B 43 E4 00 00 0A 64 00 0B 4D 88 ......Cä...d..M.

m 400+800-10
:000BF0 00 00 11 74 00 05 C6 76 00 00 DF 48 00 07 20 00 ...t...v..ßH.. .
:000C00 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
```

BINGO, notre déduction du fonctionnement du *trackloader* était la bonne.

Maintenant, pourquoi dans le second appel du trackloader, on n'utilise pas un adressage direct mais une zone mémoire ?
On va regarde ce qui se trouve à ces adresses, **taper :** `M BF8` puis `M BFC`

```
d
^001064 MOVE.L  #3800,D0
^00106A MOVE.L  00000BF8.S,D1
^00106E MOVEA.L 00000BFC.S,A0
^001072 MOVE.L  A0,-(A7)
^001074 BRA     00001110

m BF8
:000BF8 00 00 DF 48 00 07 20 00 41 41 41 41 41 41 41 41 ..ßH.. .AAAAAAAA

m BFC
:000BFC 00 07 20 00 41 41 41 41 41 41 41 41 41 41 41 41 .. .AAAAAAAAAAAA
```

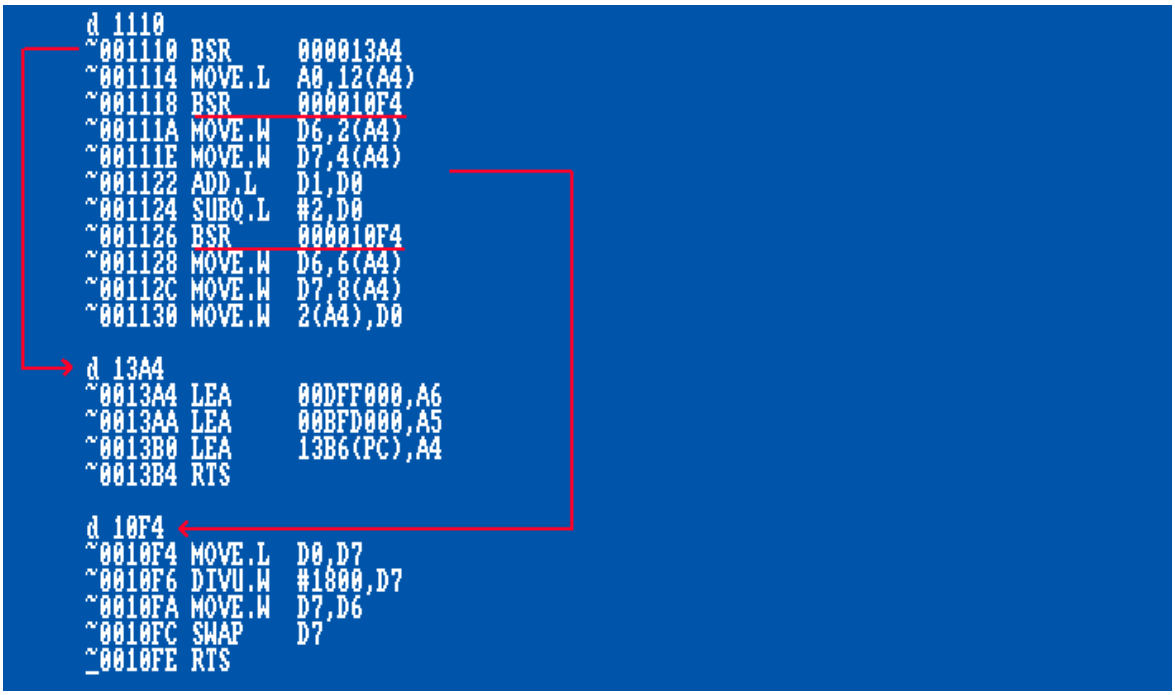
Si ça ce n'est pas un mini tableau, je veux bien être pendu. 😉

Ce qui nous donnerais donc comme appel :

		1er appel	2nd appel
Position sur le disque	D0	3000	3800
Quantité à copier	D1	800	DF48
Zone MEM de destination	A0	400	72000

Noter que ce second appel est exécuté après sont chargement car nous n'avons rien d'autre dans le code après ce dernier appel.
Revenons sur notre code en **$1010**, **taper :** `d 1110`

Nous savons maintenant que **D0** contient la position de départ sur le disque à lire.
On peut voir, que la sous routine en **$10F4** ne fait que mettre **D0** dans **D7**
Diviser **D7** par **$1800**, copier le résultat obtenus dans **D6**, faire un **swap** de **D7**



Pourquoi diviser **D0** (qui nous indique la position sur le disque comme point de départ) par $1800 ?!
Comme vous le savez sans doute (revoir la partie Agencement des disquettes si ce n'est pas le cas), une **Track** *AmigaDos* Fait **$1600**
Ici nous sommes dans un format propriétaire il est fort probable que la taille d'une track ici fasse **$1800** au lieu des **$1600**
Cela permettrait au *trackloader* de connaitre la *track* de départ.
S'il vous en prends l'envie, je vous invite à décortiquer le code à partir de **$1110** pour comprendre exactement le code du *trackloader*.
Ici, nous n'aborderons pas plus (pour une fois) ce chapitre.

Comme les deux premiers Track (la 1er piste) sont au format *AmigaDos*, cela nous donnerait :
1800 * 2 (sides) * 79 (80 – 1 track *AmigaDos*) = de donnée format spécifique à lire : **$ED000** (970 752 octet, soit 948Ko)

Cela va nous poser un problème car une disquette ***AmigaDos*** n'a pas cette capacité.
On va croiser les doigts pour qu'il n'y ait finalement pas autant de donnée à ripper, sinon… il faudra passer par une compression.

Part 4 Rip des données vers une disquette AmigaDos

Ce jeu est donné fonctionnel sur amiga500 de base (donc sans extension de mémoire) et que nous avons demandé une configuration minium au début de ce tuto de : un Amiga500 + Extension mémoire 512K

Ce qui nous donnes 512K de libre à partir de la zone **$C0000** jusqu'à **$C80000 de libre**

1.5.1 ORGANISATION DE LA MEMOIRE

la Bible AMIGA

Configuration normale

$000000	512 KB Chip-RAM	Réflexion de la zone mémoire allant de $FC0000 à SFFFFFF
$080000	Réflexion de la Chip-RAM	
$100000	Réflexion de la Chip-RAM	
$180000	Réflexion de la Chip-RAM	
$200000	8 MB Zone Fast RAM	
$A00000	CIAs	Adresse de base du CIA-B Adresse de base du CIA-A
$C00000	A500 & A2000 512 Ko RAM d'expansion	
$C80000	Vide	
$DC0000	A500 & A2000 Horloge	Adresse de base de l'horloge
$DF0000	Custonchips	Adresse de base des CustonChips
$E00000	Vide	
$E80000	Zone des slots d'Expansion	
$F00000	Module ROM	
$F80000	256 Ko Réflexion de la ROM KickStart	
$FC0000	256 Ko ROM KickStart	

Réinsérer notre disquette de backup dans le lecteur et rebooter votre AMIGA

Après un court chargement du *bootsecteur*, nous sommes dans notre boucle sans fin, **entrer** dans l'**AR** et **taper :**

```
A 159E
^00159E jmp 1010 <RETURN>
^0015A2  <RETURN>
```

Puis

```
A 1050
^001050 move.w  #$f00,$dff180 <RETURN>      ; Ecran rouge, indiquant, PRET à entrer nos valeurs A0, D0 et D1
^001058 btst    #6,$bfe001 <RETURN>         ; ICI, il faut bien sur entrer dans l'AR et définir A0, D0 et D1
^001060 bne     1050 <RETURN>               ; Tant que l'on n'appuie pas sur la souris, on boucle sur $1050
^001062 move.w  #$00F,$dff180 <RETURN>      ; Ecran bleu ,pour indique le chargement des est données en cours.
^00106A bsr     1110 <RETURN>               ; On part sur la routine du trackloader
^00106E move.w  #$0F0,$dff180 <RETURN>      ; Ecran vert, chargement des données fini, ICI on peut aller sauver
^001076 bra     106E <RETURN>               ; On boucle indéfiniment ici. (pour sauver via l'AE)
^001078 <RETURN>                            ;
```

Comme précédemment, enlever votre disquette de backup du lecteur de disquette et **insérez la disquette original** en lieu et place.
On retourne au code, **taper : X**

Assez rapidement, notre écran rouge apparait, il est maintenant temps d'entrer nos propre valeurs dans **A0** et **D0 et D1**

Nous avons donc **512Ko** de disponible, à savoir **: $80000** en hexa
Néanmoins, on va se donner une marge car nous avons toujours le code de l'**AR** en mémoire et de plus ses fichiers devront être chargées par la suite sous *AsmOne* qui fonctionne sous Worbench, bref… Tout ça nous limite la taille disponible.
Pour plus de sécurité, on va fonctionner en palier de 256Ko

Mais comme l'on veut finalement créer une disquette au format ***AmigaDos***, il est préférable de ripper des fichiers de taille du multiple d'une *TRACK AmigaDos* (Track *AmigaDos* = $1600 ou 6152 en décimal), revoir paragraphe **Agencement des disquettes Amiga**)

1 Tracks AmigaDos = $1600 = !5632 bytes
256Ko = 262144 bytes
262144 / 5632 = 46,5 'TRACKS'
Il nous faut bien sur un chiffre pair, donc **46 tracks, soit 46* 5632=259072 bytes = $3F400**

Nous allons donc ripper par pallier de $3F400 bytes par fichier.
Et l'on **commence notre rip** au début des données Non standard *AmigaDos* du disque donc position **$3000** comme vue précédemment.

Taper : `O "AA", C00000 C00000+3F400+100` puis `R A0 C00000` et `R D0 3000` sans oublier `R D1 3F400`
Sans oublier de retourner au code avec la commande **X**

```
o "AA", C00000 C00000+3F400+100
Ready.

R A0 C00000
D0=00003000 0003F400 00000001 00000000  00000000 00000000 FFFFFFFF 0000FFFF
A0=00C00000 00040400 00040400 00001400  000013B6 00BFD000 00DFF000 00001000
PC = 00001060 USP = 00000E00 SR = 0000 T=0 S=0 I=000 X=0 N=0 Z=0 V=0 C=0

r D0 3000
D0=00003000 0003F400 00000001 00000000  00000000 00000000 FFFFFFFF 0000FFFF
A0=00C00000 00040400 00040400 00001400  000013B6 00BFD000 00DFF000 00001000
PC = 00001060 USP = 00000E00 SR = 0000 T=0 S=0 I=000 X=0 N=0 Z=0 V=0 C=0

r D1 3F400
D0=00003000 0003F400 00000001 00000000  00000000 00000000 FFFFFFFF 0000FFFF
A0=00C00000 00040400 00040400 00001400  000013B6 00BFD000 00DFF000 00001000
PC = 00001060 USP = 00000E00 SR = 0000 T=0 S=0 I=000 X=0 N=0 Z=0 V=0 C=0

x
```

De retour sur notre écran rouge, appuyer sur le bouton de la souris
Le fond d'écran passe en **BLEU** et le chargement des données commence.
Une fois le chargement terminé, notre fond d'écran passe en **VERT**, il est temps pour nous d'entrer dans l'AR et de sauver tout ça.

Entrer dans l'AR, nous allons vérifier le contenue de la mémoire, **taper :** `M C00000` puis `M C00000+3F400-20`

```
m C00000
:C00000 00 00 06 5C 00 0B 37 74 00 00 06 14 00 0B 3D D0 ...\..7t......=.
:C00010 00 00 09 A4 00 0B 43 E4 00 00 0A 64 00 0B 4D 88 ......Cä...d..M.

m C00000+3F400-20
:C3F3E0 1C 81 08 14 E2 BC E1 80 DD 52 80 05 18 8A E1 64 ........R.....d
:C3F3F0 B7 54 28 09 47 BD 0E 1E 32 05 8C 38 50 3A 59 68 .T(.G...2..8P:Yh
:C3F400 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
:C3F410 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
```

On sauve tout ça sur disque, **Insérez une nouvelle disquette vierge dans le lecteur.**

#FORMAT, Permet de formater une disquette au format AmigaDOS.
#SM, alias SaveMemory. Permet donc de sauver une zone mémoire vers un fichier.
Taper `FORMAT LOTUS`

```
format LOTUS
Ready to format disk in drive DF0: (y/n)?
y
Formatting track !65, head 0
```

Puis, **taper** `SM 01, C00000 C00000+3F400`

Enlever notre disquette LOTUS fraichement crée du lecteur de disquette et **réinsérez la disquette original** en lieu et place.

Taper : `G 1050` , notre écran rouge apparait de nouveau, **entrer de nouveau dans l'AR**

Maintenant, on rippe toujours vers votre destination mémoire **$C00000** dans **A0**
Par contre, sur le disque, on avance la position de 'début de lecture' de la taille des données que l'on vient de rippé donc **+$3F400** dans **D0**
Bien sûr, la taille à ripper ne bouge toujours pas, on rip toujours **$3F400** de donnée dans **D1.**
Ce qui nous donnes :

Taper : `R A0 C00000` et `R D0 3000+3F400` sans oublier `R D1 3F400`
Et on retourne au code avec la commande **X**.

De retour sur notre écran rouge, **appuyer sur le bouton de la souris**
Le fond d'écran passe en **BLEU** et le chargement des données commence.
Une fois le chargement terminé, notre fond d'écran passe en **VERT**, il est temps pour nous **d'entrer dans l'AR** et de sauver tout ça.

On vérifie le contenue de la mémoire, **taper :** `M C00000` puis `M C00000+3F400-20`

```
M C00000
:C00000 38 02 94 70 E7 45 59 40 39 02 80 A7 1B C0 9B A6 8..p.EY@9.. .....
:C00010 46 08 19 C8 14 67 1D 71 40 CD 90 29 52 3A 7A 82 F....g.q@..)R:z.

M C00000+3F400-20
:C3F3E0 00 84 00 40 60 32 26 08 40 04 7C E2 72 8E 80 08 ...@`2&.@.|.r...
:C3F3F0 42 14 26 14 E8 40 7C 20 80 42 28 80 08 04 0D 00 B.&..@| .B(.....
:C3F400 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
:C3F410 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
```

Enlever la disquette original du lecteur de disquette et réinsérez la disquette LOTUS préalablement crée et utilisée
Puis, **taper** `SM 02, C00000 C00000+3F400`

Et on recommence 😊 ➔
Enlever notre disquette LOTUS du lecteur de disquette et **insérez la disquette original** à sa place.

Taper : `G 1050` , notre écran rouge apparait de nouveau, **entrer dans l'AR**

Taper : `R A0 C00000` et `R D0 3000+3F400+3F400` sans oublier `R D1 3F400`
Et on retourne au code avec la commande **X**.

De retour sur notre écran rouge, **appuyer sur le bouton de la souris**
Le fond d'écran passe en **BLEU** et le chargement des données commence.
Une fois le chargement terminé, notre fond d'écran passe en **VERT**, il est temps pour nous **d'entrer dans l'AR** et de sauver tout ça.

On vérifie le contenue de la mémoire, **taper :** `M C00000` puis `M C00000+3F400-20`

```
M C00000
:C00000 83 CA 18 04 08 2A F8 00 06 40 22 90 C8 04 20 02 .....*...@".... .
:C00010 10 1A 01 14 AA 00 23 B0 40 04 30 78 00 22 18 35 ......#.@.0x."5

M C00000+3F400-20
:C3F3E0 89 D0 00 00 00 00 80 12 00 00 80 12 00 00 00 00 ................
:C3F3F0 04 00 80 12 00 00 8B CC 00 00 00 00 04 00 80 12 ................
:C3F400 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
:C3F410 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
```

Enlever la disquette original du lecteur de disquette et réinsérez la disquette LOTUS préalablement crée et utilisée
Puis, **taper** `SM 03, C00000 C00000+3F400`

C'est reparti pour un petit tour, 😊 ➔

Enlever notre disquette LOTUS du lecteur de disquette et **insérez la disquette original** à sa place.

Taper : `G 1050` , notre écran rouge apparait de nouveau, **entrer dans l'AR**

Si on met encore une taille de **$3F400**, on va dépasser la taille maximum du disque original.
Même si la 1er piste est au format *AmigaDos*, le *trackloader* fonctionne lui taille de track **$1800**
Ce qui nous donne une taille max du disk original : $1800*80*2=$F0000

Et nous sommes déjà en **(3000+3F400+3F400+3F400) = $C0C00**
$F0000-$C0C00 = $2F400

On va donc mettre D1 à $2F400

Taper : `O "AA", C00000 C00000+2F400+100` puis `R A0 C00000` et `R D0 3000+3F400+3F400+3F400` sans oublier `R D1 2F400`
Et on retourne au code avec la commande `X`.

De retour sur notre écran rouge, appuyer sur le bouton de la souris
Le fond d'écran passe en **BLEU** et le chargement des données commence.
Une fois le chargement terminé notre lecteur s'arrête **mais nous n'avons pas** le fond d'écran **VERT**.
Entrer dans l'AR, on va vérifier et de sauver tout ça.

On vérifie le contenue de la mémoire, **taper :** `M C00000` puis `M C00000+2F400-20`

```
m C00000
:C00000 00 00 87 D8 00 00 00 00 04 00 80 12 00 00 87 EC ................
:C00010 89 30 00 00 04 00 80 12 00 00 8F E4 00 00 00 00 .0..........ä....

m C00000+2F400-20
:C2F3E0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
:C2F3F0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
:C2F400 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
:C2F410 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
```

Comme on pouvait s'y attendre au vue du paragraphe **Part 03 : Analyser de l'image IPF** et des dernières pistes.
Le disque ne contient pas des données jusqu'à la fin.

On va chercher jusqu'où à été remplie la zone mémoire.
Taper : `F "AA", C00000` et appuyer rapidement sur la touche **ESC** des apparitions des résultats

```
f "AA", C00000
Search from: C00000 to: C80000
C2C400  C2C401  C2C402  C2C403  C2C404  C2C405  C2C406  C2C407  C2C408  C2C409
C2C40A  C2C40B  C2C40C  C2C40D  C2C40E  C2C40F
```

Taper : `M C2C400` et remonter jusqu'à tomber sur ce qui ressemble à des données 😊

```
:C0E8F0 68 65 20 64 61 79 73 20 73 65 65 6D 65 64 20 74 he days seemed t
:C0E900 6F 20 6C 61 73 74 20 66 6F 72 65 76 65 72 20 61 o last forever a
:C0E910 6E 64 20 61 20 77 68 69 6C 65 0A 42 75 74 20 74 nd a while.But t
:C0E920 68 65 6E 20 74 68 65 20 77 65 61 74 68 65 72 20 hen the weather
:C0E930 74 75 72 6E 65 64 20 74 6F 20 67 72 65 79 0A 41 turned to grey.A
:C0E940 6E 64 20 74 68 65 20 73 74 6F 72 6D 20 77 61 73 nd the storm was
:C0E950 68 65 64 20 69 74 20 61 6C 6C 20 61 77 61 79 0A hed it all away.
:C0E960 0A 49 20 73 74 69 6C 6C 20 6C 6F 76 65 20 79 6F .I still love yo
:C0E970 75 20 52 61 63 68 65 6C 0A 0A 0A 00 00 00 00 00 u Rachel........
:C0E980 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:C0E990 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:C0E9A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:C0E9B0 00 00 00 00 00 00 00 00 FF FF FF FF FF FF FF FF ................
:C0E9C0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
:C0E9D0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
:C0E9E0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
:C0E9F0 FF FF FF FF FF FF FF FF 00 00 00 00 00 00 00 00 ................
```

On peut estimer la fin des données vers **$C0E97A**
Ce qui nous donnes comme taille théorique à sauver : **$C0E97A-$C00000=$E97A**
Mais comme on fonctionne toujours en palier de $1600, donc :
$E97A= !59770
!59770/5632=10,6
On doit sauver au minimum une longueur de 11 tracks
Soit !11*$1600 = $F200

Enlever la disquette original du lecteur de disquette et réinsérez la disquette LOTUS préalablement crée et utilisée
Puis, **taper** `SM 04, C00000 C00000+F200`

Bon, bonne nouvelle finalement, tout tient sur une disquette au format ***AmigaDos*** 😉

```
dir
Directory of (LOTUS)
      259072  01
      259072  02
      259072  03
      061952  04
0010 blocks free, 99.4 % of disk used
Disk ok
```

Notre rip est presque complet.
Presque car il manque la partie *bootsecteur*

Enlever notre disquette LOTUS du lecteur de disquette et **insérez la disquette original** à sa place.
Taper : `RT 0 2 50000`

Enlever la disquette original du lecteur de disquette et insérez une nouvelle disquette vierge.
Taper `FORMAT LOTUS2`

```
format LOTUS2
Ready to format disk in drive DF0: (y/n)?
y
Formatting track !69, head 0
```

Puis, **taper** `SM boot, 50000 50000+2C00`

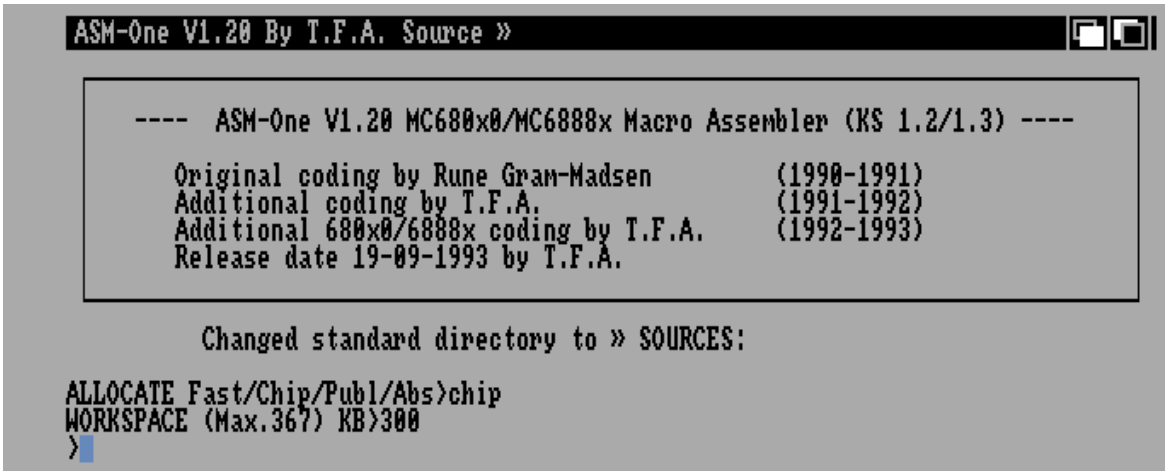
$2C00 car on copie 2 tracks qui sont au format *AmigaDos* donc $1600*2=**$2C00**

Part 5 Construction de notre disquette

Nous avons besoin maintenant de remplacer le *trackloader* original par un trackloader 'standard' AmigaDos
Nous allons utiliser pour cela le *trackloader* d'**AlphaOne** version 2004, celui qui fait 404 bytes une fois assemblé.
Vous le trouverez assez facilement sur internet ainsi qu'au format binaire et source à la fin de ce tuto ci-besoin.
Pour ce tuto, il est placé sur la disquette **LOTUS2** sous le nom de **TRACKLOADER_AlphaOne[2004].bin**

Maintenant, au niveau du code tout va se passer sous **AsmOne**

Charger donc l'assembleur, réserver un peu de mémoire chip, **300ko** suffira.



Passer en mode édition **<ESC>**, **taper** ces quelques lignes de code :

```
;--------------------------------
                LEA     LOTUSLOADER(PC),A0                      ;Adr du début du code dans A0
                LEA     LOADER+$15E,A1                          ;Adr du bootsecteur chargée+$15E dans A1
                MOVE.L  #(LOTUSLOADERENDE-LOTUSLOADER),D0       ;Compteur des données à copier dans D0

REPLACELOADER:  MOVE.B  (A0)+,(A1)+                             ;Boucle de copie de notre code/patch
                DBF     D0,REPLACELOADER                        ;Tant que le compteur est pas fini, on recopie
                RTS
;*****************************************************************
LOTUSLOADER:                                                    ;Code pour adapter A0 D0 D1 pour notre trackloader ➔
                MOVE.L  #$1F68,A2                               ;Adr. utilisée par notre trackloader pour charger nos données
                MOVE.L  D0,D4                                   ;Copie l'adr. Physique de départ à lire dans D4
                MOVE.L  D1,D3                                   ;Copie la longueur à lire dans D3
                DIVS    #$1800,D0                               ;Divise l'adresse physique de départ à lire par $1800
                MOVE.L  D0,D1                                   ;Met celle-ci dans D1, D1=N° Track disk original
                EXT.L   D1                                      ;'nettoie' D1
                MULS    #$1600,D0                               ;Multiplie D0 par $1600, D0=N°track AmigaDos (notre disk hacké)
                SUB.L   D0,D4                                   ;Soustraie D0 de l'adresse physique de départ à lire
                MOVE.L  D3,D0                                   ;Copie la longueur à lire dans D0
                SUB.L   #$400,D4                                ;On soustraie $400 de l'adresse physique à lire de départ
                MOVE.L  D4,D2                                   ;Copie celle-ci dans D2 qui nous donnes l'offset
                incbin  LOTUS2:TRACKLOADER_AlphaOne[2004].bin   ;On ajoute le code du trackloader, ça aide…
LOTUSLOADERENDE:
;*****************************************************************
LOADER:         INCBIN LOTUS2:boot                              ;Sans oublier aussi celui du boot, c'est mieux.
LOTUSDUMP:      INCBIN Lotus:01                                 ;Et bien sûr, les données que l'on a rippé
;--------------------------------
```

On sauve la source sur notre disquette **LOTUS2** sous le nom que vous voulez (par exemple Crack_Lotus).
Utiliser le menu Project/Write/Source ou la commande **W** en ligne de commande.
Toujours En ligne de commande, on **assemble** et on **exécute**.
Taper A puis **J**

Le tout est maintenant disponible en mémoire sous l'étiquette **LOADER**.
Il nous reste plus qu'à écrire tout ça sur une disquette.
Insérer un nouveau disque dans le lecteur de disquette de l'amiga et taper, en ligne de commande **WT**
comme Write Track

Comme adresse mémoire de départ on utilise le pointer/l'etiquette **LOADER**
Pour l'adresse physique (en *Tracks*) de départ sur le disque, on part de **0** bien sur
Et pour la longueur, (en *Tracks* toujours), **48**

Oui 48 car le fichier 01 lui-même fait déjà 259072 bytes soit **46** *Tracks* **+** la piste de boot (1 piste = 2 *Tracks*), *nous donnes* ***48***

Sans oublier la commande **CC** pour calculer le checksum de Boot et l'écrire sur le secteur de boot (ça aide pour booter ☺)

```
>wt
RAM PTR>LOADER
DISK PTR>0
LENGTH>48
>cc
>
```

Plus qu'à écrire les autres fichiers sur cette disquette =)
Effacer tout le code et taper celui qui suit :

```
LOTUSDUMP:      INCBIN Lotus:02
```

On assemble et on sauve, en ligne de commande **taper A**

```
RAM PTR         : LOTUSDUMP          ← Le marqueur dans le code tout simplement
DISK PTR        : 48                 ← On continue là où on s'est arrêté sur la précédemment étape.
LENGTH          : 46                 ← fichier 02=259072 bytes = 46 tracks (voir au-dessus)
```

On retourne à notre code, on change le fichier 02 par 03 et c'est repartie.

```
LOTUSDUMP:      INCBIN Lotus:03
```

On assemble et on sauve, en ligne de commande **taper A**

```
RAM PTR         : LOTUSDUMP          ← Le marqueur dans le code tout simplement
DISK PTR        : 94                 ← On continue là où on s'est arrêté sur la précédemment étape, 48+46=94
LENGTH          : 46                 ← fichier 03=259072 bytes = 46 tracks (voir au-dessus)
```

On retourne à notre code, on change le fichier 03 par 04 et c'est repartie.

```
LOTUSDUMP:      INCBIN Lotus:04
```

On assemble et on sauve, en ligne de commande **taper A**

```
RAM PTR         : LOTUSDUMP          ← Le marqueur dans le code tout simplement
DISK PTR        : 140                ← On continue là où on s'est arrêté sur la précédemment étape, 94+46=140
LENGTH          : 11                 ← fichier 04=61952 bytes = 11 tracks (voir au-dessus)
```

Et voilà, notre disque est créé, on peut essayer de booter dessus et voir ce qui se passe.

Part 6 Test de notre disquette et modification

La disquette boot et notre code du *BootSecteur* est exécuté, un chargement s'effectue ensuite jusqu'à la piste 06 et l'Amiga semble figé.
Entrer dans l'AR et taper : D

```
d
~0012C6 BNE     000012C2
~0012C8 RTS
d 12C2
~0012C2 BTST    #5,(A5)
~0012C6 BNE     000012C2
```

On semble bloqué dans cette boucle sans fin…

Ok, nous savons que le chargement se fait jusqu'à la piste 06 ce qui s'signifie qu'il s'agit du second appel *
**Voir Part 3 Analyse et modification du bootblock*

		1er appel	2nd appel
Position sur le disque	D0	3000	3800
Quantité à copier	D1	800	DF48
Zone MEM de destination	A0	400	72000

Que celui-ci ne fait pas un **bsr** mais un **bra** sur le *trackloader*, donc il charge des données du disque et les exécute tout de suite après.
On va regarder ça de plus prêt.

Rebooter votre Amiga et **avant** qu'il est fini d'atteindre la **6eme piste, entrer dans l'AR**
Taper : D 72000 et faite défiler le code jusqu'à trouver un branchement, ceux-ci (il y en a deux), se trouve à la fin de la routine en question.

```
~07219A ANDI.W  #FFF,D0
~07219E ADDI.W  #800,D0
~0721A2 DBF     D0,000721A2
~0721A6 DBF     D7,00072190
~0721AA MOVE.W  #C028,0007D7F6
~0721B2 MOVE.W  #3FFF,9A(A6)
~0721B8 MOVE.W  #7FFF,9C(A6)
~0721BE MOVE.B  #1,00BFEE01
~0721C6 MOVE.L  #72C30,00000068.S
~0721CE BSR     00072B92
~0721D2 BRA     00072D1C
;=====================================
```

On va changer le code en *$721D2* pour mettre une boucle sans fin, **taper : A 721D2**

```
^0721D2 bra 721D2 <RETURN>
^0721D4 <RETURN>
```

Et on retourne au code, taper **X**
Le chargement s'effectue et de nouveau l'amiga semble ne plus rien faire, nous sommes encore bloqués dans une boucle sans fin.
Entrer dans l'AR et **taper D**

```
d
~0012C6 BNE     000012C2
~0012C8 RTS
d 12C2
~0012C2 BTST    #5,(A5)
~0012C6 BNE     000012C2
```

Nous sommes toujours bloqués dans la boucle sans fin du code original, nous n'avons donc pas atteins le **BRA 72D1C** en **$721D2**
Donc quoi qu'il se passe, notre problème est quelque part en **$72B92** (*Le BSR 72B92 en $721CE, ben alors, faut suivre* 😉)

taper `D 72B92` et suivons le code.

```
d 72B92
~072B92 MOVE.W  #9E,D0
~072B96 JSR     0000100C.S

d 100C
~00100C BRA     0000117C

d 117C
~00117C BRA     00001156

d 1156
~001156 BTST    #4,(A5)
~00115A BEQ     0000117E
~00115C BSET    #1,(A4)
~001160 BSET    #0,(A4)
~001164 NOP
```

Finalement, on arrive en **$1156**. Nous sommes en plein dans la routine du *trackloader original**
**voir Part 3 Analyse et modification du bootblock*

On va désactiver ce bout de code en plaçant un **RTS** à la place du code en **$72B96**
Rebooter votre Amiga et **avant** qu'il est fini d'atteindre la **6eme piste, entrer dans l'AR**

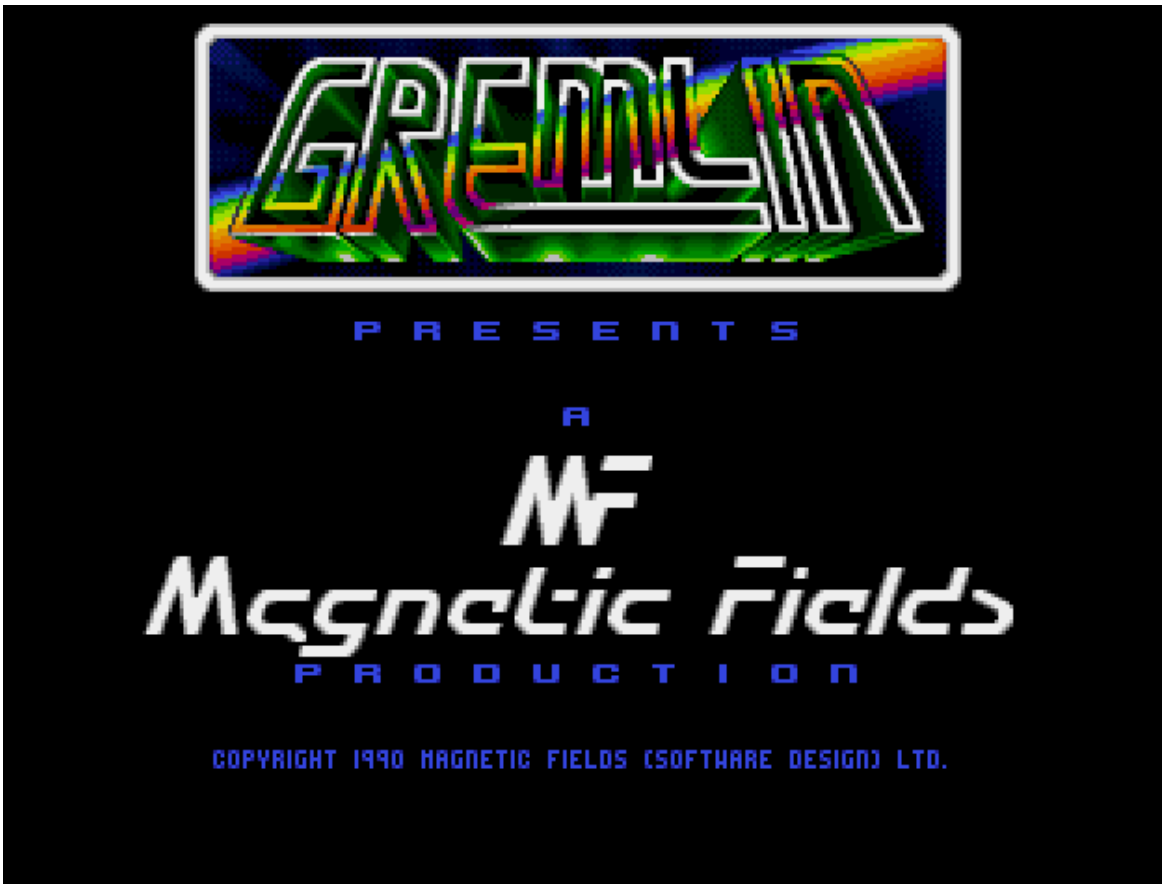
Taper `D 72B92` afin d'être sûr que le code est bien chargé et on va en profiter pour noter l'upcode **taper** `M 72B92`

```
~072B92 MOVE.W  #9E,D0
~072B96 JSR     0000100C.S
~072B9A MOVE.W  #0,D7
M 72B92
:072B92 30 3C 00 9E 4E B8 10 0C 3E 3C 00 00 2C 3C 00 06 0<..N...>...,<..
```

Puis : `A 72B96` et changer le **JSR 100C.S** par un **RTS**

```
d 72B92
~072B92 MOVE.W  #9E,D0
~072B96 JSR     0000100C.S

a 72B96
^072B96 rts
^072B98
```

Et on retourne au code, taper `X`
Le chargement s'effectue et de nouveau et …le code continue et le jeu continue son chargement 😉



Il faut maintenant trouver où se trouve ***l'UpCode*** sur nos données rippé.
Vue que c'est au début du chargement, **l'Upcode** se trouve à coup sur dans le fichier 01 que l'on a rippé.

Entrer dans l'AR
Enlever notre disquette du jeu crackéd du lecteur et **réinsérez** la disquette **LOTUS préalablement crée et utilisée lors des RIP.**
Taper : `LM 01, 30000`

On cherche notre *Upcode du* ***jsr 100C.s***, **Taper** : `F 4E B8 10 0C, 30000`

```
lm 01, 30000
Loading from 030000 to 06F400
Disk ok

F 4E B8 10 0C, 30000
Search from: 030000 to: C80000
031396
Ready.
```

Bingo, Trouver en **$31396**

Petit calcul : **$31396-$30000=$1396**

Mais il ne faut pas oublier que sur le disque, ce n'est pas cette position.
Sur le disque nous avons en premier notre secteur de boot qui fait 1 PISTE donc une taille de **$2C00** ($1600*2)
Il faut donc ajouter celui-ci à notre valeur calculée, ce qui nous donnes : **$2C00+$1396=$3F96**

Il nous reste maintenant à modifier notre code de crack sous *AsmOne.*
Ajouter ce PATCH et réécrire un bout de la disquette (le début, ne vous inquiétez pas, on ne doit pas tout recommencer)

Recharger donc *AsmOne*, réserver un peu de mémoire chip comme précédemment, **300ko**

```
ASM-One V1.20 By T.F.A. Source »

  ----  ASM-One V1.20 MC680x0/MC6888x Macro Assembler (KS 1.2/1.3) ----

     Original coding by Rune Gram-Madsen          (1990-1991)
     Additional coding by T.F.A.                  (1991-1992)
     Additional 680x0/6888x coding by T.F.A.      (1992-1993)
     Release date 19-09-1993 by T.F.A.

        Changed standard directory to » SOURCES:

ALLOCATE Fast/Chip/Publ/Abs>chip
WORKSPACE (Max.367) KB>300
>
```

Recharger votre code source de hack, ajouter la ligne en **rouge** ci-dessous :

```
;--------------------------------
                LEA     LOTUSLOADER(PC),A0
                LEA     LOADER+$15E,A1
                MOVE.L  #(LOTUSLOADERENDE-LOTUSLOADER),D0

REPLACELOADER:  MOVE.B  (A0)+,(A1)+
                DBF     D0,REPLACELOADER
                MOVE.W  #$4E75,LOADER+$3F96                    ; 4E75 étant l'upcode d'un RTS
                RTS
;************************************************************
LOTUSLOADER:
                MOVE.L  #$1F68,A2
                MOVE.L  D0,D4
                MOVE.L  D1,D3
                DIVS    #$1800,D0
                MOVE.L  D0,D1
                EXT.L   D1
                MULS    #$1600,D0
                SUB.L   D0,D4
                MOVE.L  D3,D0
                SUB.L   #$400,D4
                MOVE.L  D4,D2
                incbin  LOTUS2:TRACKLOADER_AlphaOne[2004].bin
LOTUSLOADERENDE:
;************************************************************
LOADER:         INCBIN LOTUS2:boot
LOTUSDUMP:      INCBIN Lotus:01
;--------------------------------
```

```
;-------------------------------
                LEA     LOTUSLOADER(PC),A0
                LEA     LOADER+$15E,A1
                MOVE.L  #(LOTUSLOADERENDE-LOTUSLOADER),D0

REPLACELOADER:  MOVE.B (A0)+,(A1)+
                DBF     D0,REPLACELOADER
                MOVE.W  #$4E75,LOADER+$3F96
                RTS
;*****************************************************************
LOTUSLOADER:
```

Exécuter le via les commandes **A** puis **J**
Insérez bien sur le disque **LOTUS2** dans le second lecteur ou swaper quand demandé.

```
>a
Pass 1..
Pass 2..
Incbin      : "LOTUS2:TRACKLOADER_ALPHAONE[2004].BI =       404 (=$00000194 )
Incbin      : "LOTUS2:BOOT"                         =     11264 (=$00002C00 )
Incbin      : "LOTUS:01"                            =    259072 (=$0003F400 )
No Errors
>j
```

Il nous reste plus qu'à écrire tout ça sur notre disquette préalablement crée de crack.
Insérer de nouveau notre disque de crack crée dans le lecteur de disquette de l'amiga et taper, en ligne de commande **WT**
Renseigner les valeurs comme précédemment, à savoir :

```
RAM PTR         : LOADER
DISK PTR        : 0
LENGTH          : 48
```

Sans oublier la commande **CC** pour calculer le checksum de Boot et l'écrire sur le secteur de boot.
Rebooter votre Amiga et apprécier le jeu ☺

Part 7 Binaire du TrackLoader d'AlphaOne v2004 – 404bytes

Pour ceux qui n'ont pas réussi à trouver le binaire du **trackloader d'AlphaOne**, voilà la version hexa

```
00000 |49 F9 00 BF D1 00 4B F9 00 BF E0 01 7E 00 D0 82| I.....K.....~... |
00010 |18 BC 00 7D 4E 71 4E 71 18 BC 00 75 61 00 01 6E| ...}NqNq...ua..n |
00020 |08 15 00 04 67 22 08 D4 00 01 08 D4 00 00 4E 71| ....g".........Nq |
00030 |4E 71 08 94 00 00 4E 71 4E 71 08 D4 00 00 61 00| Nq....NqNq....a. |
00040 |01 40 61 00 01 48 60 D8 83 FC 00 02 48 41 4A 41| .@a..H`.....HAJA |
00050 |67 06 08 94 00 02 60 04 08 D4 00 02 48 41 4A 01| g.....`.....HAJ. |
00060 |67 24 08 94 00 01 08 D4 00 00 4E 71 4E 71 08 94| g$........NqNq.. |
00070 |00 00 4E 71 4E 71 08 D4 00 00 61 00 01 04 61 00| ..NqNq....a...a. |
00080 |01 0C 51 C9 FF DA 61 00 01 04 3D 7C 82 10 00 96| ..Q...a...=|.... |
00090 |3D 7C 7F 00 00 9E 3D 7C 85 00 00 9E 3D 7C 44 89| =|....=|....=|D. |
000A0 |00 7E 3D 7C 40 00 00 24 2D 4A 00 20 3D 7C 99 00| .~=|@..$-J. =|.. |
000B0 |00 24 3D 7C 99 00 00 24 3D 7C 00 02 00 9C 08 39| .$=|...$=|.....9 |
000C0 |00 01 00 DF F0 1F 67 F6 3D 7C 40 00 00 24 7A 00| ......g.=|@..$z. |
000D0 |22 4A 28 3C 55 55 55 55 0C 59 44 89 66 FA 0C 51| "J(<UUUU.YD.f..Q |
000E0 |44 89 67 F4 26 11 22 29 00 04 C6 84 C2 84 E3 83| D.g.&.")........ |
000F0 |86 81 E0 9B B6 05 67 08 D3 FC 00 00 04 3E 60 D8| ......g......>`. |
00100 |D3 FC 00 00 00 38 2C 3C 00 00 00 7F 22 29 02 00| .....8,<....").. |
00110 |26 19 C6 84 C2 84 E3 83 86 81 B4 87 6E 06 48 43| &...........n.HC |
00120 |30 C3 48 43 54 87 B4 87 6E 02 30 C3 54 87 B0 87| 0.HCT...n.0.T... |
00130 |6F 00 00 40 51 CE FF D6 52 05 0C 05 00 0B 66 90| o..@Q...R.....f. |
00140 |08 14 00 02 67 08 08 94 00 02 60 00 FF 3A 08 D4| ....g.....`..:.. |
00150 |00 02 08 94 00 01 08 D4 00 00 4E 71 4E 71 08 94| ..........NqNq.. |
00160 |00 00 4E 71 4E 71 08 D4 00 00 61 00 00 14 60 00| ..NqNq....a...`. |
00170 |FF 16 18 BC 00 FD 4E 71 4E 71 18 BC 00 E7 4E 75| ......NqNq....Nu |
00180 |28 3C 00 00 25 00 51 CC FF FE 4E 75 08 15 00 05| (<..%.Q...Nu.... |
00190 |66 FA 4E 75                                      | f.Nu            |
```

```
49 F9 00 BF D1 00 4B F9 00 BF E0 01 7E 00 D0 82 18 BC 00 7D 4E 71 4E 71 18
BC 00 75 61 00 01 6E 08 15 00 04 67 22 08 D4 00 01 08 D4 00 00 4E 71 4E 71
08 94 00 00 4E 71 4E 71 08 D4 00 00 61 00 01 40 61 00 01 48 60 D8 83 FC 00
02 48 41 4A 41 67 06 08 94 00 02 60 04 08 D4 00 02 48 41 4A 01 67 24 08 94
00 01 08 D4 00 00 4E 71 4E 71 08 94 00 00 4E 71 4E 71 08 D4 00 00 61 00 01
04 61 00 01 0C 51 C9 FF DA 61 00 01 04 3D 7C 82 10 00 96 3D 7C 7F 00 00 9E
3D 7C 85 00 00 9E 3D 7C 44 89 00 7E 3D 7C 40 00 00 24 2D 4A 00 20 3D 7C 99
00 00 24 3D 7C 99 00 00 24 3D 7C 00 02 00 9C 08 39 00 01 00 DF F0 1F 67 F6
3D 7C 40 00 00 24 7A 00 22 4A 28 3C 55 55 55 55 0C 59 44 89 66 FA 0C 51 44
89 67 F4 26 11 22 29 00 04 C6 84 C2 84 E3 83 86 81 E0 9B B6 05 67 08 D3 FC
00 00 04 3E 60 D8 D3 FC 00 00 00 38 2C 3C 00 00 00 7F 22 29 02 00 26 19 C6
84 C2 84 E3 83 86 81 B4 87 6E 06 48 43 30 C3 48 43 54 87 B4 87 6E 02 30 C3
54 87 B0 87 6F 00 00 40 51 CE FF D6 52 05 0C 05 00 0B 66 90 08 14 00 02 67
08 08 94 00 02 60 00 FF 3A 08 D4 00 02 08 94 00 01 08 D4 00 00 4E 71 4E 71
08 94 00 00 4E 71 4E 71 08 D4 00 00 61 00 00 14 60 00 FF 16 18 BC 00 FD 4E
71 4E 71 18 BC 00 E7 4E 75 28 3C 00 00 25 00 51 CC FF FE 4E 75 08 15 00 05
66 FA 4E 75
```

Part 8 Code source du TrackLoader d'AlphaOne v2004 – 404bytes

Ou directement les sources pour mieux le comprendre 😊

```
init:
        move.w   $dff01c,oldintena
        move.w   $dff01e,oldintreq
        bset     #7,oldintena
        bset     #7,oldintreq
        move.w   #$7fff,$dff09a            ; interrupts aus.
        move.w   #$7fff,$dff09c            ;

;*************************************************************
        lea      $dff000,a6
        lea      mfmbuffer,a2
        lea      buffer(pc),a0
        move.l   #4*$1600,d0
        moveq    #0,d1
        move.l   #0,d2
        jsr      TRACKLOADER

;*************************************************************

        move.w   oldintena(pc),$dff09a
        move.w   oldintreq(pc),$dff09c
        moveq    #0,d0
        rts

oldintena:      dc.w     0
oldintreq:      dc.w     0
buffer:         blk.b    130000,0
mfmbuffer:      blk.w    6400,0

; HARDWARE-DISKLOADER (c) ALPHA ONE 2004.
; EASY VERSION -> WITHOUT TRACKCOUNTER.
; *************************************
; IN: A6=$DFF000
;     A2=MFMBUFFER.L
;     A0=BUFFER.L
;     D0=LENGTH.L
;     D1=TRACKNR.L
;     D2=BYTEOFFSET.L

TRACKLOADER:

                LEA      $BFD100,A4                ; DRIVESELECT REGISTER
                LEA      $BFE001,A5                ; DRIVESTATUS REGISTER
                MOVEQ    #0,D7                     ; D7 = BYTECOUNTER
                ADD.L    D2,D0                     ; BYTES TO READ + BYTEOFFSET

                MOVE.B   #$7D,(A4)                 ; SWITCH MOTOR DRIVE 0 ON
                NOP                                ; ======================
                NOP
                MOVE.B   #$75,(A4)
                BSR.W    DISKREADY

STEPHEADZERO:   BTST     #4,(A5)                   ; MOVE HEADS TO CYLINDER 0
                BEQ.B    HEADONZERO                ; ========================
                BSET     #1,(A4)
                BSET     #0,(A4)
                NOP
                NOP
                BCLR     #0,(A4)
                NOP
                NOP
                BSET     #0,(A4)
                BSR.W    DELAY
                BSR.W    DISKREADY
                BRA.B    STEPHEADZERO

HEADONZERO:     DIVS.W   #2,D1                     ; GET CURRENT CYLINDER NUMBER
                SWAP     D1                        ; ============================
                TST.W    D1
                BEQ.B    CHOOSEHEADDOWN
                BCLR     #2,(A4)
                BRA.B    MOVEHEADS
CHOOSEHEADDOWN: BSET     #2,(A4)

MOVEHEADS:      SWAP     D1                        ; MOVE HEADS TO CORR. CYLINDER
MOVELOOP:       TST.B    D1                        ; ============================
                BEQ.B    READTRACK
                BCLR     #1,(A4)
                BSET     #0,(A4)
                NOP
                NOP
                BCLR     #0,(A4)
                NOP
                NOP
                BSET     #0,(A4)
                BSR.W    DELAY
                BSR.W    DISKREADY
                DBF      D1,MOVELOOP
```

```
READTRACK:        BSR.W    DISKREADY                ; READ TRACK
                  MOVE.W   #$8210,$96(A6)           ; ================
                  MOVE.W   #$7F00,$9E(A6)
                  MOVE.W   #$8500,$9E(A6)
                  MOVE.W   #$4489,$7E(A6)
                  MOVE.W   #$4000,$24(A6)
                  MOVE.L   A2,$20(A6)
                  MOVE.W   #$9900,$24(A6)
                  MOVE.W   #$9900,$24(A6)
                  MOVE.W   #$2,$9C(A6)
TRACKREADY:       BTST     #1,$DFF01F
                  BEQ.B    TRACKREADY
                  MOVE.W   #$4000,$24(A6)

                  MOVEQ    #0,D5                    ; DECODE TRACK
DECODE:           MOVE.L   A2,A1                    ; ============
                  MOVE.L   #$55555555,D4
FINDSYNC:         CMP.W    #$4489,(A1)+
                  BNE.B    FINDSYNC
                  CMP.W    #$4489,(A1)
                  BEQ.B    FINDSYNC
                  MOVE.L   (A1),D3
                  MOVE.L   4(A1),D1
                  AND.L    D4,D3
                  AND.L    D4,D1
                  ASL.L    #1,D3
                  OR.L     D1,D3
                  ROR.L    #8,D3
                  CMP.B    D5,D3
                  BEQ.B    SECTORFOUND
                  ADD.L    #1086,A1
                  BRA.B    FINDSYNC
SECTORFOUND:      ADD.L    #56,A1
                  MOVE.L   #(512/4)-1,D6
DECODESECTOR:     MOVE.L   512(A1),D1
                  MOVE.L   (A1)+,D3
                  AND.L    D4,D3
                  AND.L    D4,D1
                  ASL.L    #1,D3
                  OR.L     D1,D3
                  CMP.L    D7,D2
                  BGT.B    BELOWOFFSET1
                  SWAP     D3
                  MOVE.W   D3,(A0)+
                  SWAP     D3
BELOWOFFSET1:     ADDQ.L   #2,D7
                  CMP.L    D7,D2
                  BGT.B    BELOWOFFSET2
                  MOVE.W   D3,(A0)+
BELOWOFFSET2:     ADDQ.L   #2,D7
                  CMP.L    D7,D0
                  BLE.W    READREADY
                  DBF      D6,DECODESECTOR
                  ADDQ.B   #1,D5
                  CMP.B    #11,D5
                  BNE.B    DECODE

TRACKDONE:        BTST     #2,(A4)                  ; TRACK DONE, GET ONTO NEXT.
                  BEQ.B    MOVECYLINDER             ; ============================
                  BCLR     #2,(A4)
                  BRA.W    READTRACK
MOVECYLINDER:     BSET     #2,(A4)
                  BCLR     #1,(A4)
                  BSET     #0,(A4)
                  NOP
                  NOP
                  BCLR     #0,(A4)
                  NOP
                  NOP
                  BSET     #0,(A4)
                  BSR.W    DELAY
                  BRA.W    READTRACK

READREADY:        MOVE.B   #$FD,(A4)                ; SWITCH MOTOR DRIVE 0 OFF
                  NOP                               ; ========================
                  NOP
                  MOVE.B   #$E7,(A4)
                  RTS

DELAY:            MOVE.L   #$2500,D4                ; DELAY ROUTINE
WAIT:             DBF      D4,WAIT                  ; =============
                  RTS

DISKREADY:        BTST     #5,(A5)                  ; WAIT FOR DISK-READY
                  BNE.B    DISKREADY                ; ====================
                  RTS

TRACKLOADERENDE:

; ===========================================================================
```