Tutoriel	**AmigaCracking : Xenon2 Megablast**
Protection	**COPYLOCK(s)**
Auteur Original	**Gi@nts**
Source original	**http://flashtro.com**
Version	**18/09/2019 Gi@nts**
Vérification/Correction	**V1.0, Testé et fonctionnel de A à Z**

* XENON 2 - 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 en disquette ou son image CAPS (SPS 0297)

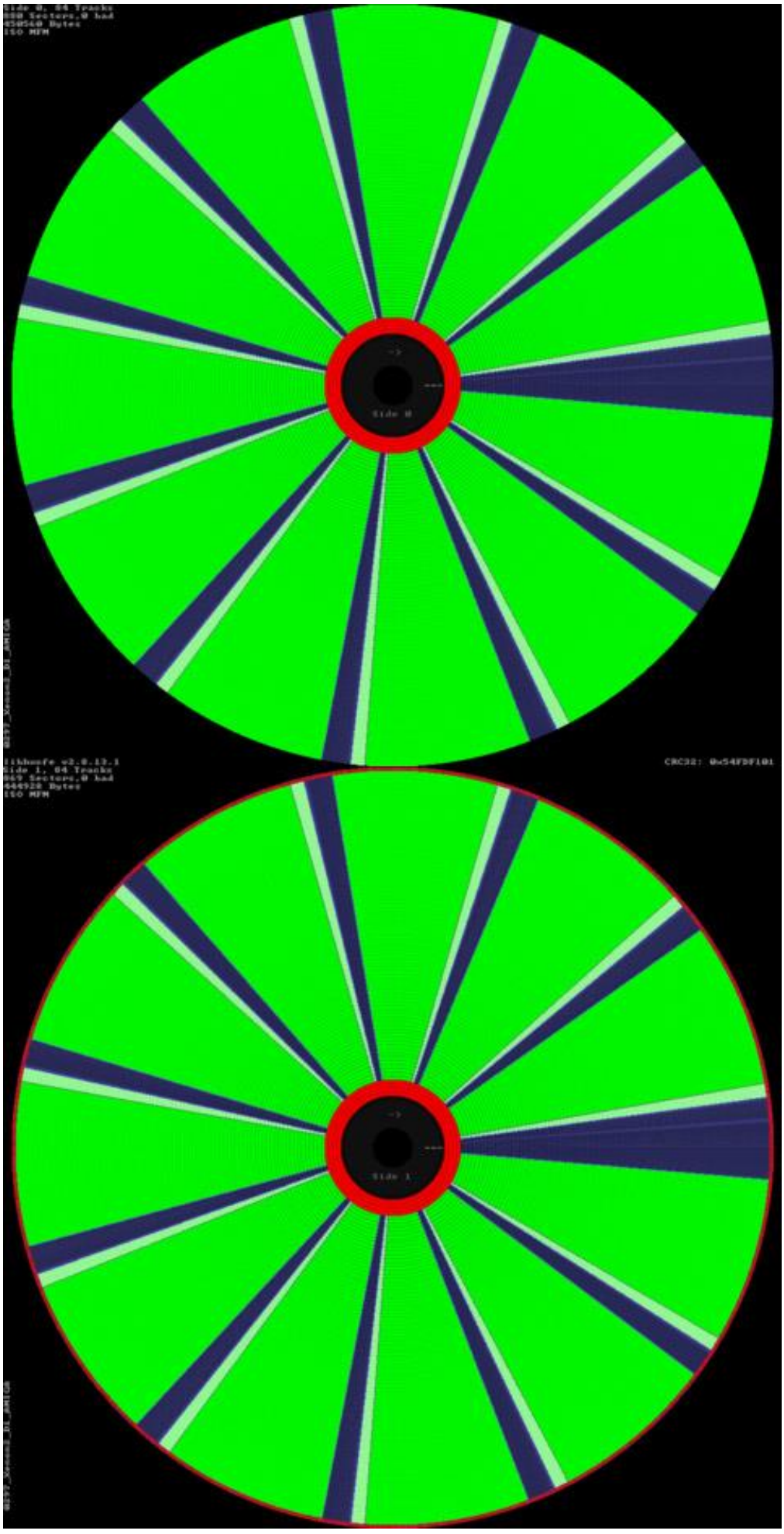
Général Info

Très gros Tutorial en approche, sortez vos pelles, ça va être long et détaillé.
A noter que ce tuto est basé uniquement sur mon travail d'analyse.
Qu'il est à mon sens assez complet. Il est fortement conseillé de suivre chronologiquement le tuto sinon vous risquer d'être perdu dans la compréhension.

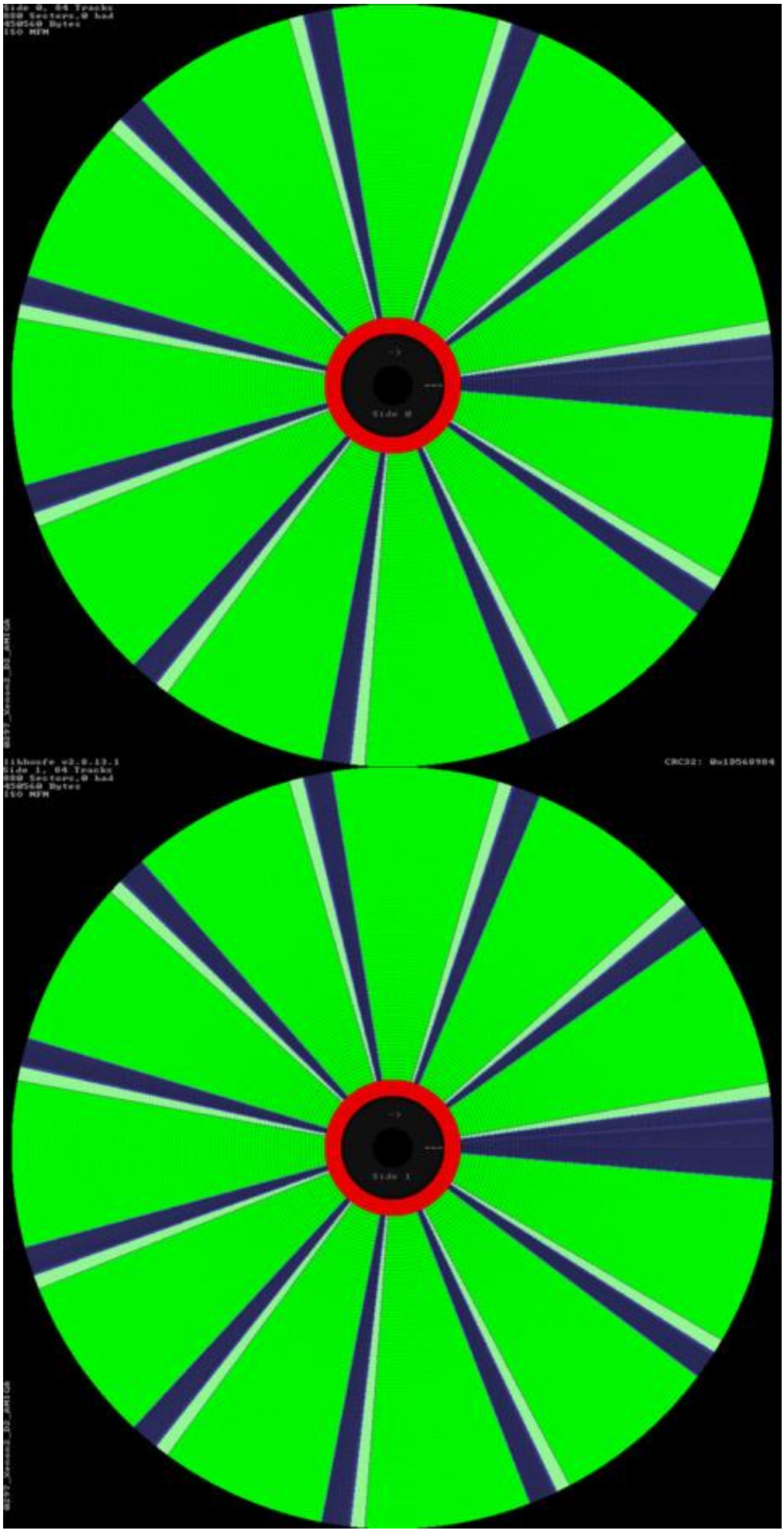
Bon Tuto.

Gi@nts

Disk 1



Disk2

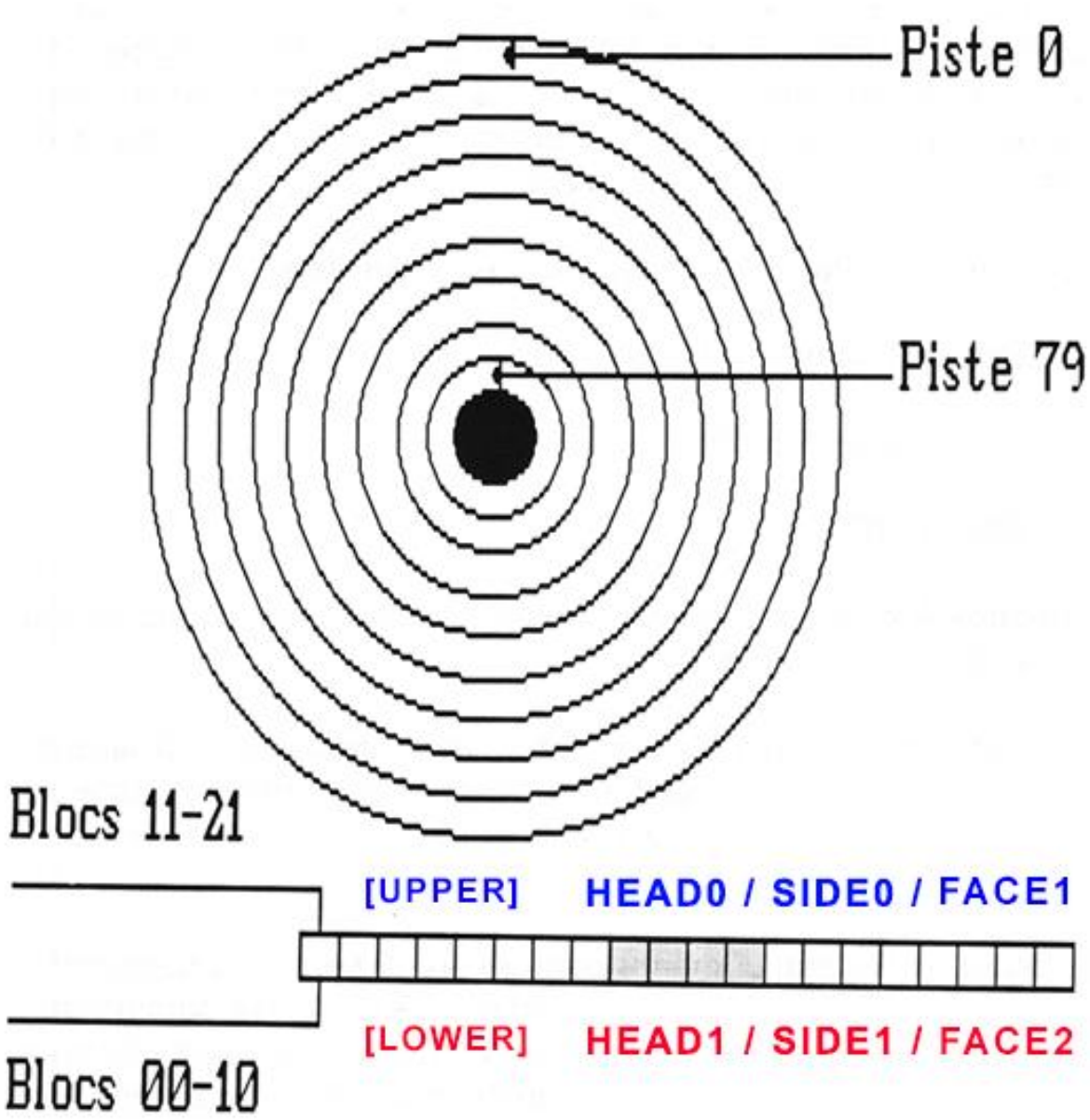


Agencement des disquettes Amiga

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 sûr) 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 piste (Track donc) à copier 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

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	...	TRACK	...	GAP	...

+ 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	00	Sync	Sync	SectorHeader	Fill	Headerchecksum	Datachecksum	Data

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 **[8 octets MFM]**
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 **[32 octets MFM]**
Zone destinée pour l'AmigaDos OS 'Recovery' mais jamais utilisée.
donc zone remplie de Zero.

HeaderChecksum

1 longWord **[8 octets MFM]**
Checksum de la zone Header.
Il est calculé en utilisant un XOR et ne contient que des 'databits'

DataChecksum

1 longWord **[8 octets MFM]**
Checksum de la zone Data.
Il est calculé en utilisant un XOR et ne contient que des 'databits'

Data

512 octets **[1024 octets MFM]**
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 opérations 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.

Les registres CIA-A et CIA-B

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 terminer 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 sur 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 et il est de 18 ms entre un changement de direction.

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

Ce **Signal** doit être mis avant l'impulsion **STEP**, de plus pour être sûr de la direction de déplacement effectué, positionner 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 quelle 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).
SIDE doit être maintenant pendant 100 microSecondes avant une opération d'écriture et pendant 1,3 MilliSeconde entre un changement de face.

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 lecteurs 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 lecteurs 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 devez 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.
```

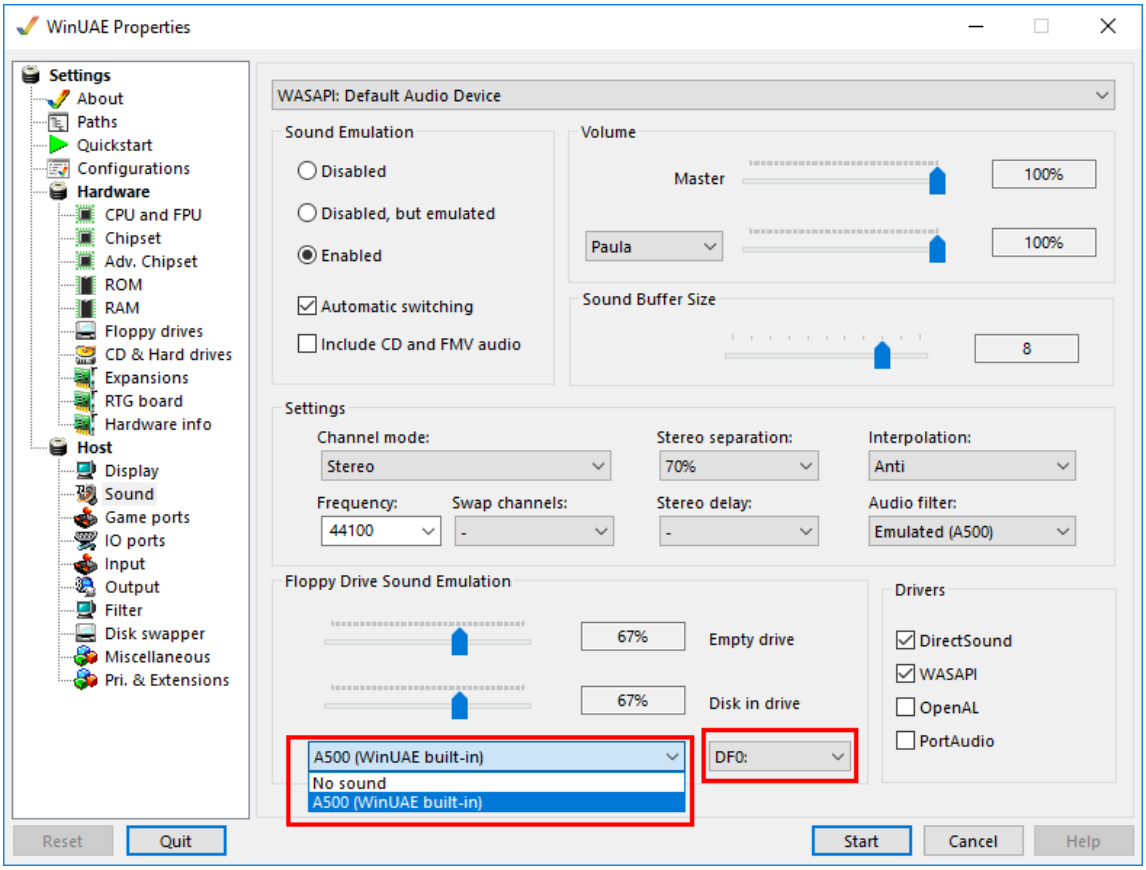
Extra Info :

Le registre **$DFF016 POTGOR**

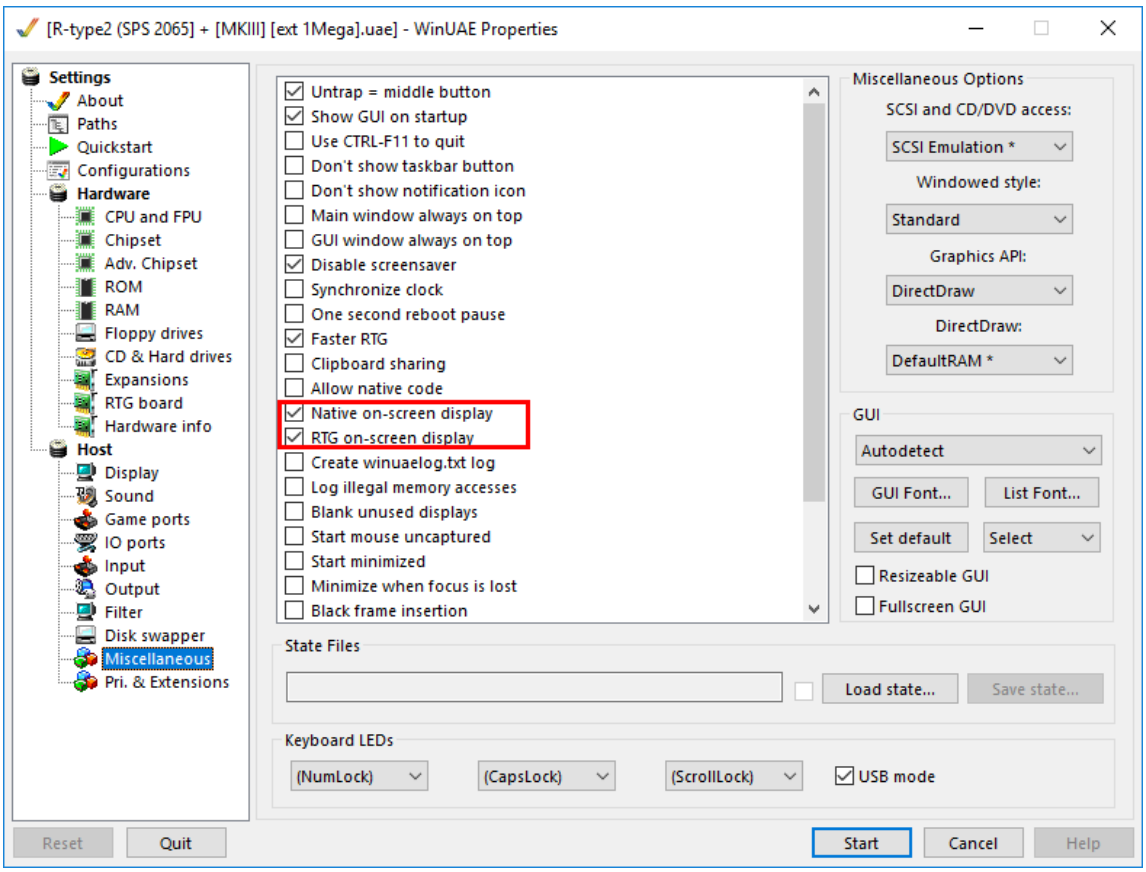
On va juste noter que sont **bit 10** sert pour **tester l'appuie sur le Bouton Droit de la souris**
Exemple : `btst #10, $dff016`

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

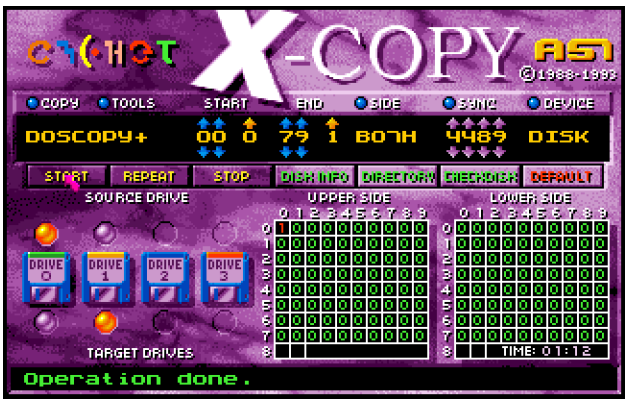


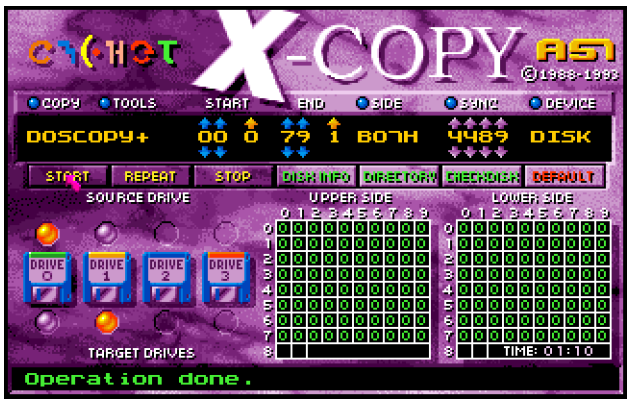
Voir même, pour plus d'information. Par exemple afficher sur qu'elle face l'on se trouve, d'activer :
Host -> Miscellaneous -> Native on-screen display AND RTG on-screen display



Part 1 X-copy

La première chose à faire et d'essayer de faire un backup de nos disquettes.
Pour cela on va utiliser X-Copy.





Hormis la piste 00 de la 1er disquette, l'ensemble semble copiable.
Il est fort probable qu'à cause de la Piste 00 notre backup ne fonctionne pas **mais gardons quand même cette copie sous le coude**.
Ce genre de 'format' ressemble à une simple protection *copylock*.

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

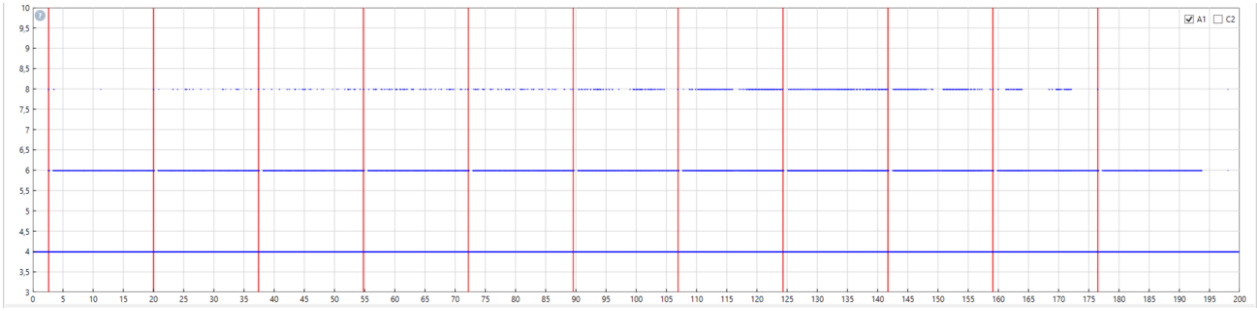
FILENAME	0297_Xenon2_Megablast.ipf
TYPE	Floppy_Disk
ENCODER	CAPS(V1)
FILE	297(V1)
DISK	1
TRACK	00-83
SIDE	0-1
PLATFORM	Amiga
REVOLUTION	5
PROTECTION	COPYLOCK [T00.1]

Comme prévue, L'ensemble du disque est au format Standard *AmigaDOS* (11 blocks standard)
Ensuite on retrouve notre 'Track' incopiable vu sous *X-Copy*. Elle est identifiée dans le format IPF comme track : **Amiga_Copylock** ainsi que par mon Analyseur d'image IPF.

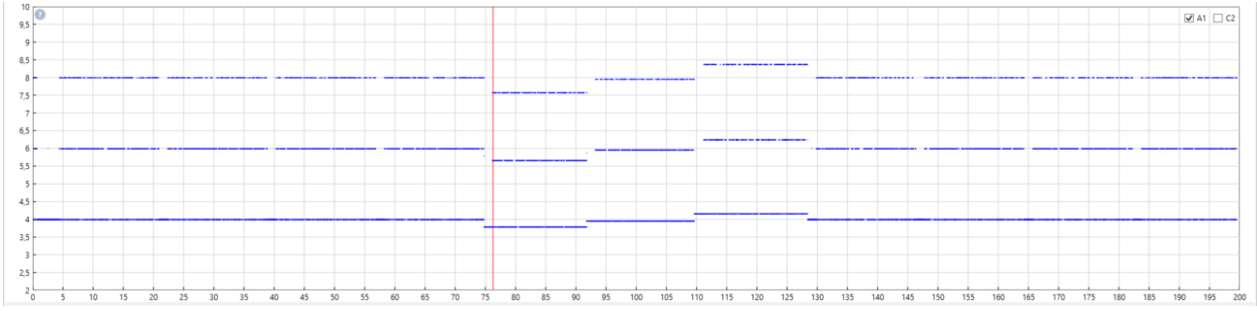
TrackNumber	Size Record (bytes)	Crc	Status	Track Size	Detail Tr. Size	Start Byte	Bit	DataKey	Block	Density	Signal	Encoder	Flag	Aufit Track View
T00.0	80	C393488A	Good	12506	100048 bits = Data=95744 + Gap=4304	274	2194	001	11	Auto	cell_2us	0	None	
T00.1	80	708F24EC	Good	12508	100048 bits = Data=90992 + Gap=9056	116	931	002	11	Copylock_Amiga	cell_2us	0	None	
T01.0	80	F3358FE0	Good	12506	100048 bits = Data=95744 + Gap=4304	272	2182	003	11	Auto	cell_2us	0	None	
T01.1	80	DD303AE7	Good	12506	100048 bits = Data=95744 + Gap=4304	274	2199	004	11	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		bytes/sector	GAP		Codage	GapDef	DataOff		Adresse
		MFM bits	bytes		MFM bits	bytes			MFM bits	bytes	
[T00.0]		6446		51568				0EB91C21			13576-20021
#0	0	5704	545	512	0	1	MFM	0352	0352	15	13576-13607
#1	1	5704	545	512	0	1	MFM	0906	0906	57	13608-13639
#2	2	5704	545	512	0	1	MFM	1460	1460	92	13640-13671
#3	3	5704	545	512	0	1	MFM	2014	2014	126	13672-13703
#4	4	5704	545	512	0	1	MFM	2568	2568	161	13704-13735
#5	5	5704	545	512	0	1	MFM	3122	3122	196	13736-13767
#6	6	5704	545	512	0	1	MFM	3676	3676	230	13768-13799
#7	7	5704	545	512	0	1	MFM	4230	4230	265	13800-13831
#8	8	5704	545	512	0	1	MFM	4784	4784	300	13832-13863
#9	9	5704	545	512	0	1	MFM	5338	5338	334	13864-13895
#10	10	5704	545	512	4304	270	MFM	5892	5892	369	13896-13927
[T00.1]		6138		49104				77C59DEF			20050-26187
#0	N/A	5272	515	N/A	720	46	MFM	0352	0352	15	20050-20081
#1		5272	515		720	46	MFM	0578	0578	55	20082-20113
#2		5272	515		720	46	MFM	1404	1404	55	20114-20145
#3		5272	515		724	46	MFM	1930	1930	121	20146-20177
#4		5272	515		715	45	MFM	2456	2456	154	20178-20209
#5		5272	515		744	47	MFM	2982	2982	187	20210-20241
#6		5272	515		713	45	MFM	3508	3508	220	20242-20273
#7		5272	515		720	46	MFM	4034	4034	253	20274-20305
#8		5272	515		720	46	MFM	4560	4560	286	20306-20337
#9		5272	515		720	46	MFM	5086	5086	318	20338-20369
#10		5272	515		1837	115	MFM	5612	5612	351	20370-20401

Ci-dessous, la vue d'une taille normale *AmigaDos* (que l'on retrouve sur l'ensemble du disque donc)



Et ci-dessous, la vue de la track 00.1, détectée en tant que *Copylock*



Cheats

Divers Cheat trouvés avec l'AR et ces fonctions de recherche + recherche dans le code (les marqueurs)
+ recherche sur le web ☺

```
$CCA                    : Nombre de vie          (trouver avec l'option TS)
$C93                    : Nombre de point        Max 9999999 soit $98 96 7F en hexa

$7B00                   : SUBQ.W  #1,CCA.S       Code qui enlève une vie (trouver avec la commande MS $CCB)
$5D80                   : SUB.W   D0,CC6.S       Sup 1 bout d'énergie quand contact avec ennemi.
$5AEC                   : ADD.L  D0.C92.S        Code qui ajoute x point quand on abat un ennemi

Vie infinie             $7B00    SUB.W    #1,CCA.S        à remplacer par   TST.W CCA.S
Energie infinie         $5D80    SUB.W    D0,CC6.S        à remplacer par   TST.W CC6.S
Bloque l'ajout de point $5AEC    ADD.L    D0,C92.S        à remplacer par   TST.W C92.S
AutoFire                $C65             Max 03          défaut 00
Vitesse de tir          $C61             Max 06          défaut 08
Vitesse du vaisseau     $C9F             Max 03          défaut 00
```

Part 3 Comportement des chargements du jeu et test de notre Backup

Avant d'entrer dans le vif du sujet on va s'attarder 5 minutes pour voir comment le jeu se comporte au niveau des chargements.
Insérez la 1er disquette originale dans le lecteur et booter dessus.

Voilà ce que l'on peut déduire par rapport au bruit de chargement des pistes du lecteur de disquette.

Nbr Piste Lu	Affichage Après chargement	Piste Lue WinUAE
00➔	BOOT	00
40	TRACKLOAD #01	01-40
00🡐	Retour T00	00
➔ 40 ?	Retour T40 ?	40
00🡐	Retour T00	00
➔ 40 ?	Retour T40 ?	40
Intro IMAGEWORKS PRESENT (avec zoom)		
00🡐	Retour T00	00
23	TRACKLOAD #01	41-64
➔ 64 ?	Retour T64	64
Greetings Xenon2 and co		
PRESS FIRE		
00🡐	Retour T00	00
INSERT GAME DISK		
DISK 2 DANS LE LECTEUR		
11		00-11
GET READY PLAYER 1		
N/A	START LEVEL 1	N/A
ESC appuyé		
N/A	Retour animation Greetings Xenon2 and co	N/A

Noter que ce n'est pas forcément 'exact' à une unité prête car il n'est pas évident à l'oreille d'être formel sur le chargement ou pas d'une piste. Mais ça permet d'avoir une idée des accès disque.

Cela nous donne plusieurs informations :
- Cela valide que le deuxième disque n'a pas de protection. (et donc qu'il y a une protection QUE sur le disque 1)

Il est temps maintenant de tester notre copie réalisée avec X-copy.
Insérez notre backup de la 1er disquette dans le lecteur et booter dessus.

Nbr Piste Lu	Affichage Après chargement	Piste Lue WinUAE
00➔	BOOT	00
40	TRACKLOAD #01	01-40
🡐	Retour T00	00
PLANTAGE de l'AMIGA		

On peut facilement en déduire que le '**Retour T00**' préalablement vue est une routine de protection, notre fameux code *CopyLock*

Part 4 Analyse et modification du bootblock

Allons donc voir et analyser notre *bootblock.*
Toujours avec la disquette de backup dans le lecteur.

#RT alias Read Track, permet le chargement de track <Track Start> <Count> <Destination_memoire>
#D, alias Désassemble
Taper : `RT 0 1 20000` puis `D 20000+c`

```
rt 0 1 20000
Disk ok

d 20000+c
~02000C LEA     20016(PC),A0
~020010 MOVE.L  A0,00000080.S
~020014 TRAP    #0
~020016 MOVE.W  #2700,SR
~02001A LEA     0007D864,A7
~020020 LEA     2000C(PC),A0
~020024 LEA     0007D868,A1
~02002A MOVE.W  #FF,D0
~02002E MOVE.L  (A0)+,(A1)+
~020030 DBF     D0,0002002E
~020034 JMP     0007D896
;=================================
~02003A MOVE.L  #7DC40,00000064.S
~020042 LEA     00BFD000,A4
~020048 MOVE.B  #FF,100(A4)
```

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

Regardons ça en détail :

```
00000C  LEA     20016(PC),A0    ; A0=$20016
020010  MOVE.L  A0,80.S         ; Copie de A0 en mémoire $80 (adr. standard pour le Trap #0)
020014  TRAP    #0              ; Exécution en mode Superviseur du code pointé à l'adresse $80 (donc $20016)
020016  MOVE.W  #2700,SR        ; SR = $2700 (supervisor mode, no interrupts)
02001A  LEA     7D864,A7        ; A7=$7D864
020020  LEA     2000C(PC),A0    ; A0=$2000C(PC)
020024  LEA     7D868,A1        ; A1=$7D868
02002A MOVE.W   #FF,D0          ; D0=$FF
02002E  MOVE.L  (A0)+,(A1)+     ; → Boucle de copie du code actuel (A0 donc,) vers la zone mémoire A1
020030  DBF     D0,2002E        ; ← On boucle sur la copie tant que D0 est différent de -1 (donc 256 fois)
020034  JMP     7D896           ; Le code est ainsi recopié vers la zone mémoire désirée et exécuté
;==============================
02003A  MOVE.L  #7DC40,64.S     ;
020042  LEA     BFD000,A4       ; A4=BFD000    CIA-B
020048  MOVE.B  #FF,100(A4)     ; RAS de BFD100, à savoir PRB
...
```

Comme il est préférable de travailler avec les adresses réelles utilisées dans le jeu et pour un meilleur facilité d'analyse, nous allons faire quelques modifications sur ce code.

Taper :
#A, alias Assemble, Instruction qui va permettre de taper du code assembleur.
#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.

On va désactiver le **JUMP** en *$20034*

```
A 20034
$020034  BRA    20034          ; Notre Dead-Loop
$020036  <RETURN>
```

On recalcul le checksum de notre code final
`BOOTCHK 20000`

Et on écrit le nouveau *bootblock*
`WT 0 1 20000`

Rebooter ensuite votre Amiga normalement.

Part 5 Analyse du TrackLoader #1

Le *bootblock* se charge et nous sommes maintenant dans notre *DeadLoop*
Entrer dans l'**AR, taper :** `D 7D896`

Comme on va le voir un peu plus loin, on a ici un *trackloader*.
Le but n'est pas de comprendre et maitriser toutes les subtilités du code en question mais de comprendre son fonctionnement global.
Voici ce que l'on peut déduire après analyse du Trackloader.

Init_Trackload

```
7D896    MOVE.L   #7DC40,64.S       ;
7D89E    LEA      BFD000,A4         ; A4=CIA-B
7D8A4    MOVE.B   #FF,100(A4)       ; RAS de PRB
7D8AA    MOVE.W   #87,100(A4)       ; Conf de PRB
                                    ; MOTOR OFF, DF3 SELECT, DF2 SELECT, DF1 SELECT, DF0 SELECT, SIDE DOWN, DIR INT., STEP TRACK=1

7D8B0    MOVE.B   #FF,100(A4)       ; On repositionne correctement PRB
7D8B6    MOVE.W   #3FFD,DFF09A      ; Pré-Conf INTENA
7D8BE    MOVE.W   #C002,DFF09A      ; Conf INTENA
7D8C6    MOVE.W   #5EF,DFF096       ; Pré-conf DMACON
7D8CE    MOVE.W   #8210,DFF096      ; Conf DMACON
7D8D6    MOVE.W   #1F,1D01(A4)      ; Conf ICR CIA-A
7D8DC    MOVE.W   #1F,D00(A4)       ; Conf ICR CIA-B
7D8E2    LEA      DFF180,A0         ; Couleur de fond dans A0
7D8E8    LEA      -3C(A0),A1        ;
7D8EC    MOVEQ    #0,D0             ; D0=0
7D8EE    MOVEQ    #7,D1             ; D1=7, c'est notre compteur, donc 8 fois
7D8F0    MOVE.L   D0,(A0)+          ; ➔
7D8F2    MOVE.L   D0,(A1)+          ; Boucle de nettoyage vidéo
7D8F4    DBF      D1,7D8F0          ; ←
7D8F8    MOVE.W   #3DDD,DFF09C      ; Configuration INTREQ
7D900    MOVE     #2000,SR          ; Normal Superviseur Mode
7D904    LEA      7DC4E(PC),A6      ; A6=$7DC4E
7D908    BSR      7D988             ; GoSub ➔ #Conf_PRB
7D90C    MOVE.W   #2,1C(A6)         ; Met $2     en (A6)+1C (7DC4E+1C=7DC6A)    : Compteur #1, Track de départ
7D912    MOVE.L   #400,6(A6)        ; Met $400   en (A6)+6, (7DC4E+06=7DC54)    : Compteur #2, Adr destination
7D91A    MOVE.L   #6B6C0,A(A6)      ; Met $6B6C0 en (A6)+A, (7DC4E+0A=7DC58)    : Compteur #3, Taille à lire
                                    ;
                                    ; $6B6C0=!440000, Taille d'une track AmigaDOS=$1600=!5632
                                    ; !440000 / !5632 = 78,125 Tracks, 1 face=2 Tracks
                                    ; donc !78 / !2 = !39 Pistes
                                    ; En partant de la seconde (car 1er = copylock) on arrive bien à la piste 40

                                    ; -TEST-START-
                                    ; On pose un BS en 7D984, en modifie les valeurs en 7D90C, 7D912 et 7D91A
                                    ; bref en faisant mumuse avec le trackloader, on valide bien
                                    ; ce que l'on a écrit au-dessus sur les compteurs.
                                    ; -TEST-END-
                                    ;
7D922    BSR      7D9BC             ; GoSub ➔ #Return_T00
7D926    MOVE.W   1C(A6),D7         ; Met le contenu de (A6)1C dans D7 (donc pour l'instant 2)
7D92A    BRA      7D930             ; GoTo ➔ #Go_Trackload
============================
7D92C    BSR.W    7D9DC             ; GoSub ➔ #Conf_PRB_DIR_BASE
```

Go_Trackload

```
7D930   SUBQ.W  #02, D7          ; D7=D7-2
7D932   BPL.B   7D92C           ; Si résultat Positif, on branche en $7D92C
7D934   BSET    #2, 100(A4)     ; Conf du bit 2 de PRB de CIAB
7D93A   BTST    #0, 1D(A6)      ; Test du bit 0 de 7DC4E+1D
7D940   BEQ.B   7D948           ; Si égale alors GoTo➔ #Base_TrackLoad
7D942   BCLR    #2, 100(A4)     ; Unset du bit 2 de PRB de CIAB

#Base_TrackLoad
7D948   BSR.W   7DA04           ; GoSub ➔ #Clean_ME
7D94C   BMI.B   7D922           ; Si flag N est vide, alors GoTo ➔ #Return_T00 via $7D922
7D94E   BCHG    #2, 100(A4)     ; Change le bit 2 de PRB de CIAB
7D954   BNE.B   7D95A           ; Si flag Z est défini, GoTo ➔ #Avance_d'une_Track
7D956   BSR.W   7D9DC           ; GoSub ➔ #Conf_PRB_DIR_BASE
```

Avance_d'une_Track

```
7D95A   ADDQ.W  #1, 1C(A6)      ; Ajoute 1 au compteur de track en (A6)+1C (7DC4E+1C=7DC6A)
                                ; Compteur #1, Track en cours
                                ;
7D95E   LEA     78000, A0       ; A0=$78000
7D964   MOVE.L  6(A6), A1       ; Met le contenu de (A6)+6, (7DC4E+06=7DC54)
                                ; Compteur #2 en A1
                                ;
7D968   MOVE.W  #57F, D0        ; D0=57F
7D96C   MOVE.L  (A0)+, (A1)+    ; ➔ Copie (A0) dans (A1) puis A0=A0+4 et A1=A1+4
7D96E   DBF     D0, 7D96C       ; ← Boucle en 7D96C tant que D0 est différent de -1 (donc $580 fois)
                                ;
7D972   MOVE.L  A1, 6(A6)       ; Met à jour le Compteur #2
7D976   SUBI.L  #1600, A(A6)    ; On enlève la taille d'une track au 'compteur' du Compteur #3
7D97E   BGT.B   7D948           ; Si résultat est plus grand que, alors c'est reparti pour un tour
                                ; GoTo ➔ #Base_TrackLoad
                                ;
7D980   BSR.W   7D9A8           ; Sinon, le chargement est fini et GoSub ➔ #FLOPPY_OFF

#End_TrackLoad
7D984   JMP     400.S           ; On exécute le code chargé en $400 qui lance la 1er phase du CopyLock
```

Conf_PRB

```
7D988   MOVE.B  #FF,100(A4)     ; RAS PRB
                                ; MOTOR OFF, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 UNSELECT, SIDE DOWN, DIR INT., STEP TRACK=1
                                ;
7D98E   MOVE.B  #7F,100(A4)     ; Conf PRB
                                ; MOTOR ON, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 UNSELECT, SIDE DOWN, DIR INT., STEP TRACK=1
                                ;
7D994   MOVE.B  #77,100(A4)     ; Positionnement PRB
                                ; MOTOR ON, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 SELECT, SIDE DOWN, DIR INT., STEP TRACK=1
                                ;
7D99A   MOVEQ   #3,D0           ; D0=3
7D99C   MOVEQ   #FFFFFF,D1      ; ➔ Compteur D1=FFFFFF
7D99E   DBF     D1,7D99E        ;       1er pause
7D9A2   DBF     D0,7D99E        ; ←     2nd pause
7D9A6   RTS                     ; E.T Retour maison
```

Floppy_OFF

```
7D9A8   MOVE.W  #FF,100(A4)     ; RAS de PRB
                                ; MOTOR OFF, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 UNSELECT, SIDE DOWN, DIR INT., STEP TRACK=1
                                ;
7D9AE   MOVE.W  #87,100(A4)     ; Conf de PRB
                                ; MOTOR OFF, DF3 SELECT, DF2 SELECT, DF1 SELECT, DF0 SELECT, SIDE DOWN, DIR INT., STEP TRACK=1
                                ;
7D9B4   MOVE.W  #FF,100(A4)     ; re-Position standard PRB
7D9BA   RTS                     ; E.T  Retour maison
```

Return_T00

```
7D9BC   MOVE.W  #FF,D7          ; D7=FF
7D9C0   BTST    #4,1001(A4)     ; ➔ Test Bit4 du PRA (TK0)
7D9C6   BEQ     7D9D0           ;     Si égale à Zero alors on branche #SIMPLE_RETOUR
7D9C8   BSR     7D9D2           ;     Sinon GoSub ➔ #Conf_PRB_2
7D9CC   DBF     D7,7D9C0        ; ← On effectue ce check tant que D7 est différent de -1
```

Simple_Retour

```
7D9D0   RTS                     ; E.T Retour maison
```

Conf_PRB_2

```
7D9D2   MOVE.B  100(A4),D1      ; D1=PRB
7D9D6   ORI.B   #2,D1           ; Fonction sur D1
7D9DA   BRA     7D9E4           ; GoTo ➔ #Deplacement_Tete
```

Conf_PRB_DIR_BASE

```
7D9DC   MOVE.W  100(A4),D1      ; D1=PRB, (on récupère la conf de PRB)
7D9E0   ANDI.B  #FD,D1          ; Fonction sur D1 pour la direction désiré.
```

Deplacement_Tete

```
7D9E4   MOVE.W  D1,D2           ; D2=D1
7D9E6   BCLR    #0,D1           ; On efface le bit en D1
7D9EA   BSET    #0,D2           ; et on le positionne en D2
7D9EE   MOVE.B  D1,100(A4)      ; Configuration de PRB avec la nouvelle valeur
7D9F2   NOP                     ; Attente conseillée entre chaque déplacement de tête.
7D9F4   NOP                     ;
7D9F6   MOVE.B  D2,100(A4)      ; D2=PRB
7D9FA   MOVE.W  #BB8,D3         ; D3=BB8
7D9FE   DBF     D3,7D9FE        ; ➔➔ Boucle d'attente
7DA02   RTS                     ; E.T Retour Maison
```

Clean_Me

```
7DA04   MOVE.W  #3,1A(A6)       ;
7DA0A   SF      0(A6)           ;
7DA0E   LEA     E(A6),A1        ;
7DA12   MOVEQ   #A,D0           ; Début du compteur D0
7DA14   CLR.B   (A1)+           ; ➔ Boucle de nettoyage de A1
7DA16   DBF     D0,7DA14        ; ➔
```

Base_TrackLoad_START

```
7DA1A   MOVE.W  #4000,DFF024    ; Configuration de DiskLength       DSKLEN, Disk DMA=OFF
7DA22   MOVE.L  #79600,DFF020   ; Configuration de Dskpth           adr. MFM buffer=$79600
7DA2C   MOVE.W  #1002,DFF09C    ; Configuration de INTREQ
7DA34   MOVE.W  #6600,DFF09E    ; Configuration de ADKCON
7DA3C   MOVE.W  #9100,DFF09E    ; Configuration de ADKCON
7DA44   MOVE.W  D00(A4),D0      ; Conf ICR CIAB                     BFD000+D00=BFDD00
                                ;
Conf_ICR_CIAB
7DA48   MOVE.W  D00(A4),D0      ; Copie de la configuration ICR CIAB dans D0
7DA4C   BTST    #4,D0           ; Test sur le Bit4, en l'occurrence DSKINDEX
7DA50   BEQ     7DA48           ; si le flag z de CCR est défini, alors GoTo ➔ #Conf_ICR_CIAB
                                ;
7DA52   MOVE.W  #A000,DFF024    ; Sinon, configuration DSKLEN
                                ; Length (# of words) of DMA data=$2000=!8192, DMA_Enabled=YES
                                ; Ce qui nous fait une Zone tampon pour le MFM de
                                ; $79600--->$79600+$(20000*2)=$7D600
                                ; Et comme on l'a vue en 7D912, Zone_de_Destination_Mémoire=$400
                                ;
7DA5A   MOVE.W  #A000,DFF024    ; Toujours deux fois pour déclencher la lecture
```

Ready?

```
7DA62   TST.B   0(A6)           ; On test le marqueur en 0(A6)
7DA66   BEQ     7DA62           ; si le flag z de CCR est défini, alors GoTo ➔ #READY?
7DA68   MOVE.W  #4000,DFF024    ; Sinon, Configuration de DSKLEN, (déclenche la lecture)
7DA70   LEA     79604,A0        ; A0=79604
7DA76   BSR     7DB42           ; GoSub ➔ #Decrypt_MFM_Base
7DA7A   BMI     7DB36           ; Si flag N est vide, alors GoTo ➔ #Fond_Rouge
7DA7E   LEA     79602,A1        ; A1=79602
```

Check_SyncWord_1

```
7DA84   CMPI.W  #4489,(A1)+     ; Check si (A1) contient le SyncWord et A1=A1+4
7DA88   BEQ     7DA84           ; Non alors GoTo ➔ #Check_SyncWord_1
7DA8A   SUBQ.L  #2,A1           ; Sinon on enlève 2 au longword A1
7DA8C   MOVEQ   #A,D7           ; D7=A
7DA8E   MOVEQ   #28,D1          ; D1=28
7DA90   BSR     7DBE6           ; GoSub ➔ #Check_SyncWord_3d
7DA94   MOVE.W  #0,D1           ;
7DA98   MOVE.W  #28,D2          ;
7DA9C   BSR     7DBA4           ; GoSub ➔ #Check_SyncWord_3b
7DAA0   BNE     7DB36           ; Si flag Z est défini, alors GoTo ➔ #Fond_Rouge
7DAA4   MOVEQ   #30,D1          ;
7DAA6   BSR     7DBE6           ; GoSub ➔ #Check_SyncWord_3d
7DAAA   MOVE.W  #0,D1           ;
7DAAE   MOVE.W  #28,D2          ;
7DAB2   BSR     7DBA4           ; GoSub ➔ #Check_SyncWord_3b
7DAB6   BNE     7DB36           ; Si flag Z est défini, alors GoTo ➔ #Fond_Rouge
7DABA   MOVEQ   #4,D1           ;
7DABC   LEA     2(A6),A0        ;
7DAC0   BSR     7DBC0           ; GoSub ➔ #Decrypt_MFM_003c
7DAC4   MOVE.B  1D(A6),D0       ;
7DAC8   CMP.B   3(A6),D0        ;
7DACC   BNE     7DB36           ; Si pas égale, alors GoTo ➔ #Fond_Rouge
7DAD0   LEA     30(A1),A1       ;
7DAD4   MOVEQ   #0,D2           ;
7DAD6   MOVE.B  4(A6),D2        ;
7DADA   CMPI.W  #A,D2           ; Comparer D2 avec $A
7DADE   BHI     7DB36           ; Si plus grand que, alors GoTo ➔ #Fond_Rouge
7DAE2   LEA     E(A6),A0        ;
7DAE6   ADDA.W  D2,A0           ;
7DAE8   BSET    #7,(A0)         ;
7DAEC   BNE     7DB36           ; Si flag Z est défini, alors GoTo ➔ #Fond_Rouge
7DAF0   MOVEQ   #9,3            ;
7DAF2   LSL.W   D3,D2           ;
7DAF4   LEA     78000,A0        ;
7DAFA   ADDA.L  D2,A0           ;
7DAFC   MOVE.W  #200,D1         ;
7DB00   BSR     7DBC0           ; GoSub ➔ #Decrypt_MFM_003c
7DB04   CMPI.B  #1,5(A6)        ;
7DB0A   BNE     7DB24           ; Si flag Z est défini, alors GoTo ➔ #Check_SyncWord_2.2
7DB0C   MOVEA.L A1,A0           ;
7DB0E   BSR     7DB42           ; GoSub ➔ #Decrypt_MFM_Base
7DB12   BMI     7DB36           ; Si résultat négatif c'est que quelque chose à foiré dans le trackload
                                ; alors GoTo ➔ #Fond_Rouge
                                ;
7DB14   LEA     79602,A1        ; A1=79602
```

Check_SyncWord_2

```
7DB1A   CMPI.W  #4489,(A1)+     ; Check si (A1) contient le SyncWord et A1=A1+4
7DB1E   BEQ     7DB1A           ; Non alors GoTo ➔ #Check_SyncWord_2
7DB20   SUBQ.L  #2,A1           ; Sinon on enlève 2 au longword A1
7DB22   BRA     7DB26           ; On branche en #Check_SyncWord_2.2b
================================

Check_SyncWord_2.2
7DB24   ADDQ.L  #8,A1           ; A1=A1+4

Check_SyncWord_2.2b
7DB26   DBF     D7,7DA8E        ; ← Décrémente D7, Tant que D7 est différent de -1, on boucle en $7DA8E
7DB2A   MOVE.W  #0,DFF180       ; Couleur du fond d'écran en noir
7DB32   MOVEQ   #0,D0           ; D0=0
7DB34   RTS                     ; E.T Retour maison
```

Fond_Rouge

```
7DB36   MOVE.W  #700,DFF180     ; Changement de couleur du fond
7DB3E   MOVEQ   #FFFFFFFF,D0    ; D0=FFFFFFFF
7DB40   RTS                     ; E.T Retour maison
```

Decrypt_MFM_Base

```
7DB42    LEA      ABC(A0),A1                    ;

Decrypt_MFM_Base.b
7DB46    MOVE.W   #5555,D0                      ; Conf Mask MFM bit impair
7DB4A    MOVE.W   #AAAA,D1                      ; Conf Mask MFM bit pair

Decrypt_MFM_001
7DB4E    MOVE.W   (A0)+,D2                      ;
7DB50    CMP.W    D0, D2                        ;
7DB52    BEQ.B    7DB60                         ;
7DB54    CMP.W    D1, D2                        ;
7DB56    BEQ.B    7DB68                         ; GoTo → #Decrypt_MFM_002
7DB58    CMPA.L   A1, A0                        ;
7DB5A    BLT.B    7DB4E                         ; GoTo → #Decrypt_MFM_001
7DB5C    MOVEQ    #FFFFFFFF, D0                 ;
7DB5E    RTS                                    ;
==============================
7DB60    LEA      7DC00(PC), A2                 ;
7DB64    MOVEQ    #F, D0                        ;
7DB66    BRA.B    7DB6E                         ; GoTo → #Decrypt_MFM_002b
==============================

Decrypt_MFM_002
7DB68    LEA      7DC20(PC), A2                 ;
7DB6C    MOVEQ    #E, D0                        ;

Decrypt_MFM_002b
7DB6E    CMP.W    (A0)+, D2                     ;
7DB70    BEQ.B    7DB6E                         ; GoTo → #Decrypt_MFM_002b
7DB72    SUBQ.L   #2, A0                        ;
7DB74    MOVE.L   (A0), D2 ;

Decrypt_MFM_002c
7DB76    CMP.L    (A2)+, D2                     ;
7DB78    BEQ.B    7DB80                         ; GoTo → #Decrypt_MFM_003
7DB7A    SUBQ.W   #2, D0                        ;
7DB7C    BPL.B    7DB76                         ; GoTo → #Decrypt_MFM_002c
7DB7E    BRA.B    7DB46                         ; GoTo → #Decrypt_MFM_Base
==============================

Decrypt_MFM_003
7DB80    LEA      79600, A1                     ; Logique, On retrouve l'adresse du DSKPTH
7DB86    MOVE.W   #$1FFF, D2                    ;
7DB8A    MOVEQ    #$10, D1                      ;
7DB8C    SUB.W    D0, D1                        ;
7DB8E    MOVE.W   -2(A0), D3                    ; →
7DB92    LSL.W    D1, D3                        ;
7DB94    MOVE.W   (A0)+, D4                     ;
7DB96    LSR.W    D0, D4                        ;
7DB98    OR.W     D3, D4                        ;
7DB9A    MOVE.W   D4, (A1)+                     ;
7DB9C    DBF      D2, 7DB8E                     ; ←
7DBA0    MOVEQ    #0, D0                        ;
7DBA2    RTS                                    ; E.T Retour Maison
==============================
Decrypt_MFM_003b
7DBA4    LEA      0(A1, D1), A2                 ;
7DBA8    MOVE.L   (A2)+, D3                     ;
7DBAA    LSR.W    #2, D2                        ;
7DBAC    SUBQ.W   #2, D2                        ;
7DBAE    MOVE.L   (A2)+, D1                     ; →
7DBB0    EOR.L    D1, D3                        ;
7DBB2    DBF D2,  7DBAE                         ; ←
7DBB6    ANDI.L   #$55555555, D3                ;
7DBBC    CMP.L    D0, D3                        ;
7DBBE    RTS                                    ; E.T Retour Maison
==============================
Decrypt_MFM_003c
7DBC0    LEA      0(A1, D1), A2                 ;
7DBC4    LSR.W    #2, D1                        ;
7DBC6    SUBQ.W   #1, D1                        ;
7DBC8    MOVE.L   (A1)+, D2                     ; →
7DBCA    ANDI.L   #$55555555, D2                ;
7DBD0    ADD.L    D2, D2                        ;
7DBD2    MOVE.L   (A2)+, D3                     ;
7DBD4    ANDI.L   #$55555555, D3                ;
7DBDA    OR.L     D3, D2                        ;
7DBDC    MOVE.L   D2, (A0)+                     ;
7DBDE    DBF      D1, 7DBC8                     ; ←
7DBE2    MOVE.L   A2, A1                        ;
7DBE4    RTS                                    ; E.T Retour Maison
==============================
Decrypt_MFM_003d
7DBE6    MOVE.L   0(A1, D1), D0                 ;
7DBEA    ANDI.L   #$55555555, D0                ;
7DBF0    ADD.L    D0, D0                        ;
7DBF2    MOVE.L   4(A1, D1), D3                 ;
7DBF6    ANDI.L   #$55555555, D3                ;
7DBFC    OR.L     D3, D0                        ;
7DBFE    RTS                                    ; E.T Retour Maison
==============================
```

```
7DC00   MOVE.L D4, A1                          ;
7DC02   LINEA                                  ;
7DC04   MOVEM.W D2/D4/A0/A3/A7, (A1)           ;
7DC08   ADDQ.B  #01, -(A4)                     ;
7DC0A   TST.B   -(A4)                          ;
7DC0C   ADDQ.L  #$02, A1                       ;
7DC0E   MOVE.B  A1, (A1)                       ;
7DC10   SUBQ.B  #$02, -(A2)                    ;
7DC12   NEG.L   -(A2)                          ;
7DC14   SUBQ.W  #2, A0                         ;
7DC16   SUBX.B  D0, $5552(A0)                  ;
7DC1A   MOVE.L  A2, A2                         ;
7DC1C   SUBQ.W  #02, (A4)                      ;
7DC1E   OR.B    D4, (A2)                       ;
7DC20   SUBX.B  D0, -(A2)                      ;
7DC22   SUBQ.B  #8, -(A2)                      ;
7DC24   LINEA                                  ;
7DC26   SUB.W   A0, D2                         ;
7DC28   LINEA                                  ;
7DC2A   MOVE.L  (A2), -(A2)                    ;
7DC2C   LINEA                                  ;
7DC2E   OR.W    D4, D4                         ;
7DC30   LINEA                                  ;
7DC32   MOVE.L  (A1), A1                       ;
7DC34   LINEA                                  ;
7DC36   MOVEM.W D0/D2/D4/D6/A0/A2/A4/A7, (A4)  ;
7DC3A   MOVE.B  -(A5),D1                       ;
7DC3C   NEG.L   A1                             ;
7DC3E   NEG.L   A1                             ;
7DC40   MOVE.W  #2,DFF09C                      ; Conf INTREQ
7DC48   ST      0(A6)                          ;
7DC4C   RTE                                    ; E.T Retour Maison
```

Après cette analyse, je vous invite à remettre le code d'origine dans le bootsecteur.
Taper :

```
RT 0 1 20000
```

```
A 20034
$020034  JMP 7D896
$02003A  <RETURN>
```

On recalcul le checksum de notre code final.

```
BOOTCHK 20000
```

Et on écrit le nouveau *bootblock*

```
WT 0 1 20000
```

Analyse du Code : CopyLock #01

Revenons à la suite de notre analyse.

Dans un premier temps, on s'aperçoit qu'une fois le *Trackload #01* fini, il effectue un **JUMP** en *$400*

```
#End_TrackLoad
7D984   JMP     400.W           ; On exécute le code en 400 qui lance la 1ère phase du CopyLock
```

Laisser votre disquette de backup de côté pour l'instant **et insérez le disk1 original du jeu dans le lecteur.**
Redémarrer votre Amiga puis **entrez dans l'AR pendant le trackload**

Toujours sous l'AR, on va poser un *BreakPoint* à l'adresse du **JUMP**.
#BS, alias BreakPoint. Permet, dès que l'adresse mémoire indiquée est atteinte, d'effectuer un arrêt du code.
Taper : `BS 7D984`

Une fois le *BreakPoint* atteint, on regarde ce qu'il y a de présent en *$400* *(à savoir* `#PRE_CopyLock_01`*)*
#BDA, alias AllBreakPoint Delete. Permet de supprimer tous les breakpoints
Taper : `BDA` puis `D 400`

```
d 400
~000400 MOVE.W  #2700,SR
~000404 LEA     400(PC),A0
~000408 LEA     00000008.S,A1
~00040C MOVE.L  A0,(A1)+
~00040E BRA     0000040C
;=====================================
~000410 LEA     00064E52,A7
~000416 BRA     000004D6
;=====================================
~00041A ORI.B   #0,D0
~00041E ORI.B   #0,D0
```

Bon...rien de bien transcendant, on continue notre analyse vers l'adresse mémoire *$4D6* (à cause du **BRA en 40E** bien sûr)
Taper : `D 4D6-8`

```
d 4D6-8
~0004CE ORI.B   #0,D0
~0004D2 LINEF
~0004D4 LINEF
~0004D6 MOVE.L  A6,-(A7)
~0004D8 LEA     460(PC),A6
~0004DC MOVEM.L D0-D7/A0-A7,(A6)
~0004E0 LEA     40(A6),A6
~0004E4 MOVE.L  (A7)+,-8(A6)
~0004E8 MOVE.L  00000010,D1
~0004EE PEA     4FA(PC)
~0004F2 MOVE.L  (A7)+,00000010
~0004F8 ILLEGAL
~0004FA PEA     518(PC)
```

Ici c'est typhique le début d'une routine de *CopyLock (expérience)*
La pour le coup on peut chercher la signature d'un *CopyLock* histoire de valider tout ça.
#F, alias FIND. Permet de rechercher une suite ascii ou hexa en mémoire.
Taper : `F 48 7A`

```
f 48 7A
Search from: 000000 to: 080000
0004EE  0004FA  00BA04  00D3CE  00D9CB  01A8C3
Ready.
```

Bon, clairement on est ici en présence d'un code *CopyLock*
Vous pouvez vérifier les autres adresses mais vous ne trouverez rien de probant (à mon sens).
Donc…

```
4D4     LINEF
4D6     MOVE.L  A6,-(A7)                ; Début de la routine de #Copylock_#01
4D8     LEA     460(PC),A6              ; en général elle fait un peu plus de $520 Octets
4DC     MOVEM.L D0-D7/A0-A7,(A6)        ; Donc on regarde la fin de cette routine
4E0     LEA     40(A6),A6               ; 4D6+520 = 9D6
...
```

On va donc regarder la fin du code de cette routine qui devrait se trouver vers *$9D6*
Taper : `D 9D6` et on descend dans le code jusqu'à trouver une fin, un retour probable
En général c'est une **mise à jour des registres** suivit d'un **RTE**, en tout cas, c'est parlant par rapport au reste du code.

Bingo !

```
~000A06 LINEF
~000A08 ORI.B   #F9,D2
~000A0C ORI.?   #8,SR
~000A12 MOVEM.L 45E(PC),D0-D7/A0-A6
~000A18 RTE
;======================================
~000A1A ROL.L   #8,D0
~000A1C MOVE.L  -8(A1),D1
~000A20 EOR.L   D1,D0
~000A22 ROR.L   #8,D0
~000A24 JMP     00064BB6
;======================================
```

On va faire quelques tests, toujours avec notre disquette **originale** dans le lecteur**.**

Taper :
D 64BB6

```
d 64BB6
~064BB6 LEA     00000A2A.S,A0
~064BBA LEA     00000400.S,A1
~064BBE MOVEA.L A1,A2
~064BC0 ADDA.L  #647CC,A2
~064BC6 MOVEQ   #2,D1
~064BC8 MOVE.W  #4AFC,(A2)
~064BCC EORI.L  #7670CF45,D0
~064BD2 EOR.B   D0,(A0)
~064BD4 MOVE.B  (A0)+,D0
```

On va maintenant poser une *Dead-Loop* à la fin de notre code *CopyLock*

Taper :
A A18
$000A18 BRA A18 ; Notre Dead-Loop
$000A1A <RETURN>

Puis retourner au code, taper : X

La routine de *CopyLock* va s'exécuter, aller en **T00** puis retourner en **Piste40**
Une fois que ceci sera fait, le code devrait être bloqué dans notre *Dead-Loop* (il ne se passe plus rien à l'écran)

Entrer dans l'AR, on en profite pour noter l'état des registres, Taper : R

$A18 **Fin #Copylock_#01**

D0	7670CF6B	0000000D	5552FFFF	55552000	00002AAF	00000400	FFFFFFFF	0000FFFF
A0	00000400	00000410	0007DC1C	00FE86EE	00BFD000	000018B6	0007DC4E	00064E4C

Ou, selon la réponse plus ou moins tardive du lecteur

$A18 **Fin #Copylock_#01**

D0	7670CF6B	0000000E	AAA6FFFF	5555A400	0000555E	00000000	FFFFFFFF	0000FFFF
A0	00000400	00000410	0007DC3C	00FE86EE	00BFD000	000018B6	0007DC4E	00064E4C

Noter que l'on retrouve fréquemment en fin de routine *CopyLock* **la clé** en **D0** et/ou à l'adresse mémoire **$24**
Et c'est le cas ici regardé plutôt : **Taper : M 24**

```
m 24
:000024 76 70 CF 6B 00 00 04 00 00 00 04 00 00 00 04 00 vp.k............
```

Vous pouvez aussi faire ce test avec **notre backup**, vous verrez que **D0** sera à 00000000 (pas de clé *CopyLock* trouvée)

On en profite pour regarder si notre code en *$64BB6* à changer, on ne sait jamais.
Taper :
D 64BB6

Le code est exactement le même que précédemment, ce qui signifie que le code *TrackLoadé* en *$64BB6* n'a pas été traité.
On devrait donc retrouver les données BRUT au format *AmigaDOS* sur notre Disquette.
Ce qui est une bonne nouvelle si on doit le modifier ☺

On n'oublie pas de remettre le code d'origine en **A18**
Taper :
A A18
$000A18 RTE
$000A1A <RETURN>

Analyse du Code : CopyLock #02

Hormis le petit bout de code en *$A1A*, le prochain code à analyser sera celui en *$64BB6*, à cause du : **$A24 JUMP 64BB6**

Taper : `D 64BB6`

`PRE_Copylock_#02_1`

```
64BB6   LEA     A2A.S,A0                ; A0=A2A
64BBA   LEA     400.S,A1                ; A1=400
64BBE   MOVE.L  A1, A2                  ; Copie le LongWord A1 vers A2
64BC0   ADDA.L  #647CC, A2              ; A2=A2+647CC     donc A2=64BCC
64BC6   MOVEQ   #02, D1                 ; D1=02
64BC8   MOVE.W  #4AFC, (A2)             ; Copie le word $4AFC à l'adresse de A2,
                                        ; Ce qui se traduit par un effacement du EOR ci-dessous remplacer par un ILLEGAL
============================
EOR_fonction_Base
64BCC   EORI.L  #$7670CF45, D0          ; ➔ Commande bidon qui va être écrasée pour ne garder
                                        ;    qu'au final l'Upcode : 4A FC 76 70 CF 45
============================            ;
                                        ;    Qui se traduira par un changement de code du EORI par
                                        ;    64BCC                 ILLEGAL
        ------------------------------------------------------------------------------------
        EOR_fonction
        64BD2   EOR.B   D0, (A0)                ; Modification de (A0) avec EOR de D0
        64BD4   MOVE.B  (A0)+, D0               ; Modification de D0, puis A0=A0+1
        64BD6   MOVE.B  D0, (A1)+               ; Modification de (A1), puis A1=A1+1
                                                ;
        64BD8   EOR.B   D0, (A0)                ; Modification de (A0) avec EOR de D0
        64BDA   MOVE.B  (A0)+, D0               ; Modification de D0, puis A0=A0+1
        64BDC   MOVE.B  D0, (A1)+               ; Modification de (A1), puis A1=A1+1
                                                ;
        64BDE   EOR.B   D0, (A0)                ; Modification de (A0) avec EOR de D0
        64BE0   MOVE.B  (A0)+, D0               ; Modification de D0, puis A0=A0+1
        64BE2   MOVE.B  D0, (A1)+               ; Modification de (A1), puis A1=A1+1
                                                ;
        64BE4   EOR.B   D0, (A0)                ; Modification de (A0 A0avec EOR de D0
        64BE6   MOVE.B  (A0)+, D0               ; Modification de D0, puis A0=A0+1
        64BE8   MOVE.B  D0, (A1)+               ; Modification de (A1), puis A1=A1+1
                                                ;
        64BEA   CMPA.L  #64BB6, A0              ; Compare A0 // 64BB6, sinon c'est reparti pour un tour
                                                ; (1er passage A0=A2E pour info)
                                                ;
        64BF0   BLT.B   $64BD2                  ; si plus petit alors GoTo ➔ #EOR_Fonction
                                                ; Ce qui donnes un décodage de la zone mémoire : $400 ➔ $64BB6
        ------------------------------------------------------------------------------------
                                        ;
64BF2   DBF     D1, 64BCC               ; ➔ D1=D1-1, et tant que D1 est différent de -1 alors GoTo ➔ #EOR_Fonction_Base
                                        ; 3 passages théoriques/boucles donc pour générer le code final en mémoire.
                                        ;
                                        ; Mais comme ça sera un ILLEGAL en 64BCC et on va sauter sur l'adresse
                                        ; pointé en $10, à savoir, $400, voir quelques lignes plus bas pour explication
                                        ;
64BF6   JMP     $1000.S                 ; JMP ne sera jamais effectué.
============================
```

TEST : Si vous voulez tester le **JMP 1000**, mettre une *Dead-Loop* en *$64BF6* et vous verrez qu'elle ne sera jamais atteinte.

On va poser un *BrekPoint* en *$64BF2*, **Taper :** `BS 64BF2` puis `X` pour continuer le code
Une fois notre *BrekPoint* atteint, on efface notre *BrekPoint* et on regarde le code en *$64BCC*, **Taper** `BDA` puis `D 64BCC`

Effectivement, notre analyse du code semble être bonne, nous avons maintenant un joli **ILLEGAL** en *$64BCC*
On regarde l'adresse stocké en *$10*, cela permet de connaitre l'endroit du prochain Saut, en l'occurrence *$400*

Alors…aucune chance que les fonctions de TRACE sous la MKIII ne puisse fonctionner avec la commande **ILLEGAL**

Donc, logiquement on va continuer notre analyse du code à l'adresse mémoire *$400*
D 400 puis de toute ses sous-routines.

PRE_Copylock_#02_2

```
================================
00400   MOVE.W  #0, C32         ;
00408   MOVE.L  #70000, C38     ;
00412   MOVE.L  #78000, C34     ;
0041C   MOVE.W  #4, C2C         ;
00424   ST      B603            ;
0042A   BTST    #7, 00BFEE01    ; Test du bit 7 du cra du CIAA
00432   BNE.B   442             ;
00434   MOVE.W  #5, C2C         ;
0043C   SF      B636            ;
================================
00442   MOVE.W  C2C, C2E        ; Copie de $C2C en mémoire C2E
0044C   MOVE.L  #B98, $0068.S   ; Copie de $B98 en mémoire  68
00454   MOVE.L  #C4C, 6C.S      ; Copie de $C4C en mémoire C6C
                                ;
0045C   MOVE.B  #$FF, BFD100    ; MOTOR OFF, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 UNSELECT, SIDE DOWN, DIR INT., STEP TRACK=1
                                ;
00464   MOVE.B  #$87, BFD100    ; MOTOR OFF, DF3 SELECT , DF2 SELECT  , DF1 SELECT  , DF0 SELECT  , SIDE DOWN, DIR INT., STEP TRACK=1
                                ; Cherche sur tous les lecteurs
                                ;
0046C   MOVE.B  #$FF, BFD100    ; MOTOR OFF, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 UNSELECT, SIDE DOWN, DIR INT., STEP TRACK=1
                                ;
00474   MOVE.W  #$4000, DFF100  ; Conf BPLCON0 (Bit Plane Control Register)
0047C   MOVE.W  #0, DFF102      ; Conf BPLCON1 (Bit Plane Control Register)
00484   MOVE.W  #38, DFF092      ; Conf DDFSTRT (Display Data Fetch Start)
0048C   MOVE.W  #D0, DFF094     ; Conf DDFSTOP (Display Data Fetch Stop)
00494   MOVE.W  #$3A81,DFF08E   ; Conf DIWSTRT (Display Window start)
0049C   MOVE.W  #2C1, DFF090    ; Conf DIWSTOP (Display Window stop)
004A4   MOVE.W  #78, DFF108     ; Conf BPL1MOD (Bit plane modulo Odd planes)
004AC   MOVE.W  #78, DFF10A     ; Conf BPL2MOD (Bit plane modulo Even planes)
004B4   MOVE.W  #C00, DFF034    ; Conf POTGO (Pot port)
004BC   MOVE.W  #$3FFF, DFF09C  ; Conf INTREQ
004C4   MOVE.W  #$3FD5, DFF09A  ; Conf POTINP
004CC   MOVE.W  #$C02A, DFF09A  ; Conf POTINP
004D4   MOVE.W  #$002F, DFF096  ; Conf DMACON
004DC   MOVE.W  #$87D0, DFF096  ; Conf DMACON
004E4   MOVE.B  #$1F, BFED01    ; Conf ICR - CIAA
004EC   MOVE.B  #$1F, BFDD00    ; Conf ICR - CIAB
004F4   MOVE.B  #$88, BFED01    ; Conf ICR - CIAA
004FC   MOVE.B  BFED01, D0      ; D0=Configuration actuel de ICR - CIAA
00502   MOVE.B  BFDD00, D0      ; D0=Configuration actuel de ICR - CIAB
00508   MOVE.W  #FF, DFF09E     ; Conf ADKCON (Audio, Disk, UART)
00510   MOVEQ   #0, D0          ; D0=0
00512   MOVE.L  D0, DFF144      ; D0=Configuration actuel de SPR0DATA Registre A
00518   MOVE.L  D0, DFF14C      ; D0=Configuration actuel de SPR1DATA Registre A
0051E   MOVE.L  D0, DFF154      ; D0=Configuration actuel de SPR2DATA Registre A
00524   MOVE.L  D0, DFF15C      ; D0=Configuration actuel de SPR3DATA Registre A
0052A   MOVE.L  D0, DFF164      ; D0=Configuration actuel de SPR4DATA Registre A
00530   MOVE.L  D0, DFF16C      ; D0=Configuration actuel de SPR5DATA Registre A
00536   MOVE.L  D0, DFF174      ; D0=Configuration actuel de SPR6DATA Registre A
0053C   MOVE.L  D0, DFF17C      ; D0=Configuration actuel de SPR7DATA Registre A
00542   MOVE.L  #C02, DFF080    ; Conf de COP1LCH
0054C   MOVE.L  #C02, C26       ; Copie de $C02 en mémoire $C26, ???
00556   MOVE.W  D0, DFF088      ; Conf COPJMP1
0055C   MOVE.W  #$2000, SR      ; Configuration de la pile Superviseur
00560   BSR.W   E6C             ; GoSub ➔ #RAS_Registry_Dx
00564   BSR.W   C80             ; GoSub ➔ #Sync_Vidéo
00568   MOVEM.L C60,D0-D7       ; Restauration de D0-D7 ?
00570   MOVEM.L D0-D7,DFF180    ; Modification de la couleur du fond.
00578   LEA     B88, A0         ; A0=$B88
0057E   BRA.W   63E             ; GoTo ➔ #Copylock_#02
================================
```

RAS_Registry_Dx

```
================================
E6C     MOVEA.L C38.S,S0            ;
E70     LEA     7D00(A0),A0         ; A0=7D00
E74     MOVEQ   #0,D0               ; D0=0
E76     MOVE.L  D0,D1               ;
E78     MOVE.L  D0,D2               ;
E7A     MOVE.L  D0,D3               ;
E7C     MOVE.L  D0,D4               ;
E7E     MOVE.L  D0,D5               ;
E80     MOVE.L  D0,D6               ;
E82     MOVEA.L D0,A1               ;
E84     MOVE.W  #7C,D7              ; D7=7C
E88     TST.B   C2B.S               ; Test du byte en $C2B
E8C     BEQ     E92                 ; Saute la configuration de D7 si besoin
E8E     MOVE.W  #41,D7              ; Sinon D7=41
                                    ;
E92     MOVE.L  D0-D6/A1,-(A0)      ; →
E96     MOVE.L  D0-D6/A1,-(A0)      ;
E9A     MOVE.L  D0-D6/A1,-(A0)      ;
E9E     MOVE.L  D0-D6/A1,-(A0)      ;
EA2     MOVE.L  D0-D6/A1,-(A0)      ;
EA6     MOVE.L  D0-D6/A1,-(A0)      ;
EAA     MOVE.L  D0-D6/A1,-(A0)      ;
EAE     MOVE.L  D0-D6/A1,-(A0)      ;
EB2     DBF     D7,E92              ; ←
EB6     RTS                         ; E.T Retour maison
================================
```

Sync_Video

```
================================
C80     MOVE.W  C30.S,D0            ;
C84     SUB.W   C32.S,D0            ;
C88     CMP.W   C2C.S,D0            ; Test de D0 avec $C2C
C8C     BCS     C80                 ; Ci le flag C est défini, C'est reparti pour un tour
C8E     MOVE.W  C30.S,C32.S         ;
C94     LEA     DFF006,A0           ; A0=VHPOSR             (Vertical position)

Attente de Synchro 1
C9A     BTST    #0,DFF005           ;
CA2     BEQ     C9A                 ; ← Tant que ce n'est pas synchro, GoTo → #Attente_de_Synchro_1

Attente de Synchro 2
CA4     CMPI.B  #2,(A0)             ; Compare 2 avec (A0), Vertical Position
CA8     BCS     CA4                 ; ← Tant que ce n'est pas synchro, GoTo → #Attente_de_Synchro_2
                                    ;
CAA     MOVE.L  C38.S,D0            ;
CAE     MOVE.L  C34.S,C38.S         ;
CB4     MOVE.L  D0,C34.S            ;
CB8     MOVE.L  #17E0,D0            ;
CBE     SUB.L   C26.S,D0            ;
CC2     MOVE.L  D0,C26.S            ;
CC6     MOVE.L  D0,DFF080           ;
CCC     MOVE.L  D0,DFF080           ;
CD2     RTS                         ; E.T Retour maison
================================
```

Copylock_#02

```
63E     MOVE.L  A6,-(A7)            ; Bon, encore une routine typique de copylock
640     LEA     5C8(PC),A6          ;
644     MOVEM.L D0-D7/A0-A7,(A6)    ; Sauvegarde des registres à l'adresse pointée par A6
648     LEA     40(A6),A6           ;
64C     MOVE.l  (A7)+,-8(A6)        ;
650     MOVE.L  10,D1               ;
656     PEA     662(PC)             ;
65A     MOVE.L  (A7)+,10            ;
660     ILLEGAL                     ; (OPCode 4A FC), on TRAP à l'adresse pointé par $10, à savoir $662
662     PEA     680(PC)             ;
666     MOVE.L  (A7)+,10            ;
66C     MOVEQ.L A7,A1               ;
...                                 ;
69E     ILLEGAL                     ; (OPCode 4A FC), on TRAP à l'adresse pointé par $10, à savoir $714
...                                 ;
B6E     LINEF                       ;
B70     ORI.B   #F9,D2              ;
B74     ORI.?   #8,SR               ;
B7A     MOVEM.L 5C6(PC),D0-D7/A0-A6 ;
B80     RTE                         ; Fin de la routine #Copylock_#02
================================
```

Suite du code Copylock_#02

```
Post_Traitement_Copylock_#02
B82     MOVE.L  #7670CB6B,D1            ; Copy de la nouvelle clé copylock en dur dans D1
                                        ; Ancienne = 7670CF6B, nouvelle=7670CB6B
                                        ;
B88     MOVE.L  D1,(A0)                 ; Copy de la clé à l'adresse pointé par A0 (à savoir $B88)
B8A     MOVE.L  (A0),24.S               ; Copy de la clé à l'adresse mémoire $24
B8E     LEA     898F42E9,A1             ; A1=898F42E9
B94     JMP     0(A1,D0.L)              ; GoTo ➔ #INTRO (à savoir JMP $1254)* voir analyse plus bas.
================================

================================
B88     MOVEQ   #70,D3                  ; Code modifié par le code juste au-dessus
B8A     AND.W   D5,24(A3)               ;
B8E     LEA     898F42E9,A1             ; identique que précédemment
B94     JMP     0(A1,D0.L)              ; GoTo ➔ #INTRO (à savoir JMP $1254)* voir analyse plus bas.
================================
```

Les routine de *CopyLock* étant assez bas niveau, on préfèrera une *DeadLoop* qu'un *BreakPoint.*
On poser donc une *Dead-Loop* à la fin de notre code *CopyLock* en ***$B80***

Taper :

```
A B80
$000B80  BRA     B80            ; Notre Dead-Loop
$000B82  <RETURN>
```

Puis retourner au code, taper : X

Notre nouvelle routine de *CopyLock* va s'exécuter, aller en **T00** puis retourner en **Piste40**
Une fois que ceci sera fait, le code devrait être bloqué dans notre *Dead-Loop*

Entrer dans l'AR, on en profite pour noter l'état des registres, Taper : R

$B80 **Fin #copylock_#02**

D0	7670CF6B	05550777	06660444	02220111	00000333	05550777	06660444	02220111
A0	00000B00	00000000	00064BCC	00FE86EE	00BFD000	00C014B6	0007DC4E	00064E46

Sans oublier de regarder à l'adresse mémoire : *$24*

```
M 24
:000024 76 70 CF 6B 00 00 04 00 00 00 04 00 00 00 04 00 vp.k............
```

On n'oublie pas de remettre le code d'origine en **B80**
Taper :

```
A B80
$000B80  RTE
$000B82  <RETURN>
```

Et on va regarder aussi l'état des registres en fin de **Post_Traitement_CopyLock_02**, à savoir en *$B94*

Taper :

```
A B94
$000B94  BRA     B94            ; Notre Dead-Loop
$000B96  <RETURN>
```

Puis retourner au code, taper : X
Immédiatement on se retrouve dans notre *Dead-Loop*
Entrer dans l'AR et allons jeter un coup d'œil au registres, Taper : R

$B94 **Fin #Post_Traitement_copylock_#02**

D0	7670CF6B	7670CB6B	06660444	02220111	00000333	05550777	06660444	02220111
A0	00000B88	898F42E9	00064BCC	00FE86EE	00BFD000	00C014B6	0007DC4E	00064E4C

Petit calcul simple pour connaitre l'adresse de **JMP** utilisé en *$B94*
D0=7670 CF6B Adressage spécifiée dans l'instruction : (.L). On prend donc le *LongWord* **CF6B** de **D0** pour le calcul.
A1=898F 42E9 L'adressage sur un **JMP** est en LongWord, On prend donc le *LongWord* **42E9** de **A1** pour le calcul.

$CF6F**+$42E9**=$**00011254**, et comme le jump fonctionne en *LongWord*, ça nous donne que *$1254*
Cela nous donne donc finalement un : `B94 JMP 1254`

On peut remettre le code original en **B94**
Taper :

```
A B94
$000B94  JMP     0(A1,D0.L)
$000B98  <RETURN>
```

Analyse du Code : Intro

Et c'est partie pour l'analyse du code en mémoire *$1254* (Merci aux BreakPoints 😉)
Taper : D 1254

Intro

```
================================
01254   EORI.L #$7670CF6B, $24.W        ; Fonction Ou Exclusif immédiat sur la clé copylock stocké en $24
0125C   MOVEQ #$0, D0                   ;
0125E   JSR $B010                       ; GoSub → #Start_Music_Intro
01264   LEA $149E, A0                   ; A0=$149E        A0=adr.Mem.Texte          // texte=IMAGEWORKS
0126A   MOVEQ #$0, D0                   ;
0126C   MOVE.L #$1082, $C3E.W           ; Marqueur C3E=adr.memoire_1082
01274   BSR.W $F9E                      ; GoSub → #Apparition_TEXTE               // 'IMAGEWORKS'
01278   MOVE.L #$10B4, $C42.W           ; Marqueur C42=adr.memoire_10B4
01280   LEA $14B3, A0                   ; A0=$14B3        A0=adr.Mem.Texte          // texte=PRESENT
01286   MOVE.W #$B8, D0                 ;
0128A   BSR.W $F9E                      ; GoSub → #Apparition_TEXTE               // 'PRESENT'
0128E   MOVE.L #$10E2, $C3E.W           ; Marqueur C3E=adr.memoire_10E2
01296   LEA $3190, A0                   ;
0129C   MOVE.W #$FFE0, D0               ;
012A0   MOVEQ #-$42, D1                 ;
012A2   MOVEQ #$3F, D2                  ;
012A4   MOVE.W #$84, D3                 ;
012A8   MOVE.W #$1, $EB8.W              ;
012AE   BSR.W $EE8                      ; GoSub → #Zoom_Avant                     // 'Bloc ImageWork's'
012B2   MOVEQ #$1E, D7                  ;
012B4   BSR.W $ED6                      ; GoSub → #Tempo
012B8   BSR.W $F12                      ; GoSub → #Zoom_Arriere                   // 'Bloc ImageWorks'
012BC   MOVE.L #$1082, $C3E.W           ; Marqueur C3E=adr.memoire_1082
012C4   LEA $14B3, A0                   ; A0=14B3         A0=adr.Mem.Texte          // texte=PRESENT
012CA   MOVE.W #$B8, D0                 ;
012CE   BSR.W $1012                     ; GoSub → #Disparition_TEXTE              // 'PRESENT'
012D2   MOVE.L #$C4A, $C42.W            ; Marqueur C42=adr.memoire_C4A
012DA   LEA $149E, A0                   ; A0=149E         A0=adr.Mem.Texte          // texte=IMAGEWORKS
012E0   MOVEQ #$0, D0                   ;
012E2   BSR.W $1012                     ; GoSub → #Disparition_TEXTE              // 'IMAGEWORKS'
012E6   MOVE.L #$C4A, $C3E.W            ; Marqueur C3E=adr.memoire_C4A
012EE   LEA $14C8, A0                   ; A0=14C8         A0=adr.Mem.Texte          // texte='BITMAP_BROTHERS'
012F4   MOVEQ #$0, D0                   ;
012F6   MOVE.L #$1082, $C3E.W           ; Marqueur C3E=adr.memoire_1082
012FE   BSR.W $F9E                      ; GoSub → #Apparition_TEXTE               // 'BITMAP_BROTHERS'
01302   MOVE.L #$10C2, $C42.W           ; Marqueur C42=adr.memoire_10C2
0130A   LEA $14DD, A0                   ; A0=14DD         A0=adr.Mem.Texte          // texte=GAME
01310   MOVE.W #$B8, D0                 ;
01314   BSR.W $F9E                      ; GoSub → #Apparition_TEXTE               // 'GAME'
01318   MOVE.L #$10F4, $C3E.W           ; Marqueur C3E=adr.memoire_10F4
01320   LEA $2984, A0                   ;
01326   MOVE.W #$FFE0, D0               ;
0132A   MOVEQ #-$33, D1                 ;
0132C   MOVEQ #$4F, D2                  ;
0132E   MOVE.W #$66, D3                 ;
01332   MOVE.W #$1, $EB8.W              ;
01338   BSR.W $EE8                      ; GoSub → #Zoom_Avant                     // 'Bloc TheBitmapBrothers'
0133C   MOVE.L #$10D0, $C46.W           ; Marqueur C46=adr.memoire_10D0
01344   MOVEQ #$1E, D7                  ;
01346   BSR.W $ED6                      ; GoSub → #Tempo
0134A   MOVE.L #$C4A, $C46.W            ; Marqueur C46=adr.memoire_C4A
01352   BSR.W $F12                      ; GoSub → #Zoom_Arriere                   // 'Bloc TheBitmapBrothers'
01356   MOVE.L #$1082, $C3E.W           ; Marqueur C3E=adr.memoire_1082
0135E   LEA $14DD, A0                   ; A0=14DD         A0=adr.Mem.Texte          // texte=GAME
01364   MOVE.W #$B8, D0                 ;
01368   BSR.W $1012                     ; GoSub → #Disparition_TEXTE              // 'GAME'
0136C   MOVE.L #$C4A, $C42.W            ; Marqueur C42=adr.memoire_C4A
01374   LEA $14C8, A0                   ; A0=14C8         A0=adr.Mem.Texte          // texte=A BITMAP BROTHERS
0137A   MOVEQ #$0, D0                   ;
0137C   BSR.W $1012                     ; GoSub → #Disparition_TEXTE              // 'A BITMAP BROTHERS'
01380   MOVE.L #$C4A, $C3E.W            ; Marqueur C3E=adr.memoire_C4A
01388   BSR.W $1242                     ; GoSub → #RAZ_Color_Table
0138C   BSR.W $E6C                      ; GoSub → #RAZ_Registry_Dx
01390   BSR.W $C80                      ; GoSub → #Conf_Coprocessor
01394   BSR.W $E6C                      ; GoSub → #RAZ_Registry_Dx
01398   ST C2B.S                        ;
0139A   CMPI.B #$, -$0  71E(A3)         ;
013A0   LEA $C60.W, A2                  ;
013A4   BSR.W $115A                     ; GoSub → $115A                           // Aucune importance dans notre tuto
013A8   LEA $C60.W, A2                  ;
013AC   BSR.W $11B2                     ; GoSub → #Affiche_Logo_Xenon2_MEGABLAST
```

Suite Intro

```
013B0   LEA $2330, A0                   ;
013B6   MOVE.W #$50, D0                 ;
013BA   MOVEQ #$13, D1                  ;
013BC   MOVE.W #$4F, D2                 ;
013C0   MOVE.W #$50, D3                 ;
013C4   MOVE.W #$1, $EB8.W              ;
013CA   BSR.W $EE8                      ; GoSub ➔ #Zoom_Avant                       // 'Bloc Droit - RHYTHM KING'
013CE   MOVE.L #$111E, $C42.W           ; Marqueur C42=adr.memoire_111E
013D6   LEA $1C28, A0                   ;
013DC   MOVE.W #$FF60, D0               ;
013E0   MOVEQ #$19, D1                  ;
013E2   MOVEQ #$5F, D2                  ;
013E4   MOVE.W #$4A, D3                 ;
013E8   MOVE.W #$1, $EB8.W              ;
013EE   BSR.W $EE8                      ; GoSub ➔ #Zoom_Avant                       // 'Bloc gauche - BOMB THE BASS'
013F2   MOVE.W #$20, $C3C.W             ;
013F8   BSR.W $EBA                      ; GoSub ➔ #RAS_Registry_Dx // #JSR_Marqueur_C3E_C42_C46 // #Conf_Coprocessor
013FC   BSR.W $EBA                      ; GoSub ➔ #RAS_Registry_Dx // #JSR_Marqueur_C3E_C42_C46 // #Conf_Coprocessor
01400   CMPI.L #$78000, $C34.W          ;
01408   BEQ.B $140E                     ; GoSub ➔ $140E                             // Aucune importance dans notre tuto
0140A   BSR.W $EBA                      ; GoSub ➔ #RAS_Registry_Dx//#JSR_Marqueur_C3E_C42_C46 //#Conf_Coprocessor
0140E   SF BDC.S                        ;
01410   BSET D5, (A4)+                  ;
01412   TST.B $BDC.W                    ; ➔ Test du marqueur $BDC
01416   BNE.B $1422                     ;       Si différent de Zero, on saute directement sur le #Zoom_Arriere en $1422
01418   BTST #$7, $BFE001               ;       Bouton Fire appuyé ?
01420   BNE.B $1412                     ; ←     Non ? On boucle
01422   BSR.W $F12                      ; GoSub ➔ #Zoom_Arriere                     // 'Bloc gauche - BOMB THE BASS'
01426   MOVE.L #$C4A, $C42.W            ; Marqueur C42=adr.memoire_C4A
0142E   LEA $2330, A0                   ;
01434   MOVE.W #$50, D0                 ;
01438   MOVEQ #$13, D1                  ;
0143A   MOVE.W #$4F, D2                 ;
0143E   MOVE.W #$50, D3                 ;
01442   MOVE.W #$1, $EB8.W              ;
01448   BSR.W $F12                      ; GoSub ➔ #Zoom_Arriere                     // 'Bloc droit - RHYTHM KING'
0144C   BSR.W $E6C                      ; GoSub ➔ #RAZ_Registry_Dx
01450   BSR.W $C80                      ; GoSub ➔ #Conf_Coprocessor
01454   LEA $14F6, A0                   ; A0=14F6         A0=adr.Mem.Texte          // texte=LOADING GAME
0145A   MOVE.W #$86, D0                 ;
0145E   MOVE.L #$1082, $C46.W           ; Marqueur C46=adr.memoire_1082
01466   BSR.W $F9E                      ; GoSub ➔ #Apparition_TEXTE                 // 'LOADING GAME'
0146A   BSR.W $E6C                      ; GoSub ➔ #RAS_Registry_Dx
0146E   MOVE.W #$14, $F9C.W             ;
01474   BSR.W $1082                     ; GoSub ➔ $1082                             // Aucune importance dans notre tuto
01478   BSR.W $C80                      ; GoSub ➔ #Conf_Coprocessor
0147C   CMPI.L #$78000, $C34.W          ; Si en Adr. Mémoire C34 on trouve 78000
01484   BEQ.B $1492                     ; On saute en $1492 si c'est le cas.
                                        ;
                                        ; Sinon ...
01486   BSR.W $E6C                      ; GoSub ➔ #RAS_Registry_Dx
0148A   BSR.W $1082                     ; GoSub ➔ $1082                             // Aucune importance dans notre tuto
0148E   BSR.W $C80                      ; GoSub ➔ #Conf_Coprocessor
                                        ;
01492   JSR $B02C                       ; GoSub ➔ #Clear_DMA/ADKCON/AUDxVOL
01498   JMP $64194                      ; GoSub ➔ #PRE_Trackload2
================================
```

RAZ_Color_table

```
1242    MOVEQ   #0,D0                   ; D0=0
1244    MOVEQ   #7,D1                   ; D1=7
1246    LEA     DFF180,A0               ; A0=DFF180 (couleur de fond)
124C    MOVE.L  D0,(A0)+                ; ➔
124E    DBF     D1,124C                 ; ← D1=D1-1, boucle en $124C tant que D1 est différent de -1
1252    RTS                             ; E.T Retour Maison
```

J'ai bypassé volontairement certaine analyse et ai mis ici l'essentiel. (enfin, je pense)
Vous pourrez de vous-mêmes arriver aux mêmes résultats en posant divers *BreakPoints* et en conclure rapidement les diverses Sous-Routines listées ci-dessus.

Compte rendu rapide :
En *$1254* nous avons le début du code pour l'Animation principale.
En *$1498* nous avons la dernière instruction du code (fin de l'anim donc) vers le second TrackLoader en *$64194*

Analyse du Code : TrackLoader_#2

Et c'est partie pour l'analyse du code en mémoire *$64194*
Taper : D 64194

Pre_Trackload2

```
================================
64194   LEA     74864,A7            ; A7=74864
6419A   MOVE.L  #64560,64.S         ;
641A2   LEA     BFD000,A4           ; A4=BFD000       ak CIAB
641A8   MOVE.W  #3FFD,DFF09E        ; Configuration de ADKCON
641B0   MOVE.W  #C002,DFF09A        ; Conf INTENA
641B8   MOVE.B  #1F,1D01(A4)        ; Conf de icr sur CIAA (1D01+BFD000=BFED01=CIAA_icr)
641BE   MOVE.B  #1F,D00(A4)         ; Conf de icr sur CIAB (D00+BFD000=BFDD00=CIAB_icr)
641CE   MOVE.L  #64284,DFF080       ; Conf de COP1LCH
641D4   MOVE.W  D0,DFF088           ; Conf COPJMP1
641D8   BSR     642A8               ; GoSub → #Motor_ON/DF0_SELECT
641DC   MOVE.W  #52,1C(A6)          ; Compteur #1=$52
641E2   MOVE.L  24.S,6(A6)          ;
641E8   MOVE.L  #40000,A(A6)        ; Compteur #3 (nbr de Byte à copier) pour le 2nd_Trackloader
641F0   BSR     642DC               ; GoSub → #Retour_T00_2
641F4   MOVE.W  1C(A6),D7           ; D7=Compteur #1
641F8   BRA     641FE               ; GoTo → #Pre_TrackLoad2_2
================================
```

Pre_TrackLoad2_2

```
================================
641FA   BSR     642FC               ; →     GoSub → #Conf_PRB_DIR_BASE2
641FE   SUBQ.W  #2,D7               ;       D7=D7-2
64200   BPL     641FA               ; ←   Si résultat négatif, on boucle
                                    ;
64202   BSET    #2,100(A4)          ; Conf du CIAB - SIDE -> bas
64208   BTST    #0,1D(A6)           ; Test d'un marqueur
                                    ; A coup sûr, on test le marqueur de la 'side' à lire (Up or Down)
6420E   BEQ     64216               ; ! On saute si besoin le BCLR en dessous !
64210   BCLR    #2,100(A4)          ; Conf du CIAB - SIDE = haut
64216   BSR     64324               ; GoSub → #TrackLoad2
================================
...
```

TrackLoad2

```
================================
64324   MOVE.W  #3,1A(A6)       ;
6432A   SF      0(A6)           ;
6432E   LEA     E(A6),A1        ;
6433E   MOVEQ   #A,D0           ; Début du compteur D0
64334   CLR.B   (A1)+           ; → Boucle de nettoyage de A1
64336   DBF     D0,64324        ; ←
================================
```

Base_TrackLoad2_START

```
================================ ;
6433A   MOVE.W  #4000,DFF024    ; Configuration de DiskLength      DSKLEN, Disk DMA=OFF
64342   MOVE.L  #70600,DFF020   ; Configuration de Dskpth          adr. MFM buffer=$70600
                                ; La zone du tampon_MFM à changer de place
                                ;
6434C   MOVE.W  #1002,DFF09C    ; Configuration de INTREQ
64354   MOVE.W  #6600,DFF09E    ; Configuration de ADKCON
6435C   MOVE.W  #9100,DFF09E    ; Configuration de ADKCON
65364   MOVE.W  D00(A4),D0      ; Conf ICR CIAB, BFD000+D00=BFDD00 dans D0
                                ;
Conf_ICR_CIAB                   ;
64368   BTST    #4,D0           ; Test sur le Bit4, en l'occurrence DSKINDEX
6436C   BEQ     64368           ; si le flag z de CCR est défini, alors GoTo → #Conf ICR CIAB
64370   MOVE.W  #A000,DFF024    ; Sinon, configuration DSKLEN
64372   MOVE.W  #A000,DFF024    ; Toujours deux fois pour déclencher la lecture
.....                           ; Tjrs zone de destination mémoire $400 pour info
                                ; ------------------------------------------------------------------------------
                                ; Zone remplie : $400 → $40E00
                                ; $40E00=!265728  265728/5632=47.18 48 Tracks         48/2=24 Piste
                                ; 41+24=65
                                ; 65-1 car piste 41 comprise dans le calcul
                                ; On retombe bien sur une fin de piste à 64 vues au début du tuto
                                ; ------------------------------------------------------------------------------
```

Vous pouvez, avant le démarrage du second *TrackLoad* remplir la mémoire d'un pattern quelconque et poser un *BreakPoint* en fin de *TrackLoad*
Vous obtiendrez/validerez les valeurs calculées données au-dessus.

Suite Base_TrackLoad_START

```
...
6421A   BMI     641F0           ; Si flag N est vide, alors GoTo ➔ #Retour_T00_2 via autre sous-routine
6421C   BCHG    #2,100(A4)      ; Change le bit 2 de PRB de CIAB
64222   BNE     64228           ; Si flag Z est défini, GoTo ➔ #Avance_D'une_Track
64224   BSR     642FC           ; GoSub ➔ #Conf_PRB_DIR_BASE2
==============================
```

Avance_D'une_Track

```
============================== ;
64228   ADDQ.W  #1,1C(A6)       ; Ajoute 1 au compteur de track en (A6)+1C, Track en cours ?
                                ; compteur #1                       (track actuel)
                                ;
6422C   LEA     6F000,A0        ; A0=6F000                          (A0 semble servir de Zone tampon)
64232   MOVEA.L 6(A6),A1        ; Met (A6)+6, (Compteur #2) en A1
64236   MOVE.W  #57F,D0         ; D0=57F
                                ;
6423A   MOVE.L  (A0)+,(A1)+     ; ➔ Copie (A0) dans (A1) puis A0=A0+4 et A1=A1+4
                                ;    (Recopie de la zone mémoire depuis $6F000 vers $400)
6423C   DBF     D0,6423A        ; ➔ Boucle en 6423A tant que D0 est différent de -1          (donc $580 fois)
64240   MOVE.L  A1,6(A6)        ; Met à jour le Compteur #2
64244   SUBI.L  #1600, A(A6)    ; On enlève la taille d'une track AmigaDos $1600, au 'compteur' du Compteur #3
                                ; (en $6456E+A, au début =$40000) mis en : 641E8 MOVE.L #40000,A(A6)
                                ;
                                ; -$1600 à chaque passage, ce qui fait !46 Passages
                                ; $40000=!262144    $1600=6532    262144/5632=46,6, donc 47 Tracks à lire
                                ; 1 piste = 2 Tracks de $1600 donc !46/2=23 Piste à lire
                                ;
                                ; 41+23=64, on retombe sur nos pieds
                                ;
6424C   BGT.B   64216           ; Si résultat est plus grand que, alors c'est reparti pour un tour,
                                ; GoTo ➔ #TrackLoad2 via $64216
                                ;
6424E   BSR.W   642C8           ; Sinon, le chargement est fini et GoSub ➔ #FLOPPY_OFF2
============================== ;
6424E                           ; #Fin_Second_TrackLoad
```

Décrypt_Creation_du_code_Copylock_#03

```
==============================
64252                           ;
64252   MOVEQ   #1,D1           ; D1=1
                                ;
Boucle #2                       ;
64254   MOVEQ.L 24.S,A0         ; ➔ A0=$400 pour info (adresse écrite en $24)
64258   MOVEQ   #2A,D0          ;       D0=2A
                                ;
Boucle #1                       ;  ➔ Début routine de modification de D0 et de création de (A0)
6425A   EOR.B   D0,(A0)         ;          ... ... 1er phase de decrypt_EOR
6425C   MOVE.B  (A0)+,D0        ;          ... A0=A0+1
                                ;
6425E   EOR.B   D0,(A0)         ;          ... ... 2er phase de decrypt_EOR
64260   MOVE.B  (A0)+,D0        ;          ... A0=A0+1
                                ;
64262   EOR.B   D0,(A0)         ;          ... ... 3em phase de decrypt_EOR
64264   MOVE.B  (A0)+,D0        ;          ... A0=A0+1
                                ;
64266   EOR.B   D0,(A0)         ;          ... ... 4em phase de decrypt_EOR
64268   MOVE.B  (A0)+,D0        ;          ... A0=A0+1
                                ;
6426A   CPMA.L  #40000,A0       ;          Compare A0 avec 40000
64270   BLT     6425A           ;       ➔ Si A0 est plus petit que 40000 alors, GoTo ➔ #Boucle_#01
64272   DBF     D1,64254        ; ➔ Sinon, tant que D1 est différent de -1 (donc 3 tours), GoTo ➔ #Boucle_#02
                                ; Ce qui donnes un décodage de la zone mémoire : $400 ➔ $40000
                                ;
==============================
64276   PEA     64252           ; A7=A7-4
6427C   MOVE.W  (A7),-(A7)      ;
6427E   EORI.W  #E552,(A7)      ;
64282   RTE                     ; GoTo $400 (les BRA vers les routines de copylock sont tjs crée ici sur ce jeu)
==============================
```

Bon...la prochaine étape à coup sûr est celle du Second *TrackLoad*
Suivi visiblement d'une bonne grosse routine de décryptage pour générer le code en *$400*
Et au final, lancer le code de *CopyLock* en *$400*

On va donc tranquillement analyser ceci point par point.

Analyse du Code : Post_TrackLoader_#2_&_CopyLock #03

Poser un *BreakPoint* en *$6244E* afin de regarder ce qu'il a TrackLoadé en *$400*
Taper : `BS 6424E` puis `X`
Passer l'intro principale, on arrive sur le début du *TrackLoad*

Une fois le trackload terminé (en piste 64 pour info), notre *BreakPoint* est atteint, on entre automatiquement dans l'AR
Allons jeter un coup d'œil en *$400*
Taper : `D 400` puis `M 400`

```
d 400
~000400 BRA     0000042C
;=====================================
~000402 CMPI.B  #30,0(A0,D0.W)
~000408 ORI.B   #0,D0
~00040C ORI.B   #0,D0
~000410 ORI.B   #0,D0
~000414 ORI.B   #0,D0
~000418 ORI.B   #0,D0
~00041C ORI.B   #0,D0
~000420 ORI.B   #0,D0
~000424 ORI.B   #0,D0
~000428 ORI.B   #0,D0
~00042C ORI.B   #0,D0
~000430 ORI.B   #0,D0
~000434 ORI.B   #0,D0
~000438 ORI.B   #0,D0
~00043C ORI.B   #0,D0
~000440 ORI.B   #0,D0
~000444 ORI.B   #0,D0
m 400
:000400 60 2A 0C 30 6C 30 00 00 00 00 00 00 00 00 00 00 `*.0l0..........
:000410 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:000420 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:000430 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Bon, clairement ça valide que ce n'est pas le code qui va être exécuté* MAIS c'est celui qui a été trackloadé.
*** 400 BRA 42C et en $42C il y a…. rien**

On enlève notre *BreakPoint* et en poser un nouveau en fin de routine `#Décrypt_Creation_du_code_Copylock_#03`
C'est à dire en *$64282*

Taper : `BDA` puis `BS 64282` sans oublier de retourner au code avec la commande `X`

Assez rapidement notre *BreakPoint* est atteint et on entre automatiquement dans l'AR
Taper : `D 400`

```
Ready.
Breakpoint raised at address: 00064282

d 400
~000400 BRA     00007032
;=====================================
~000404 ORI.B   #0,D0
~000408 ORI.B   #0,D0
~00040C ORI.B   #0,D0
```

Assez logiquement on continu notre analyse en *$7032* et ainsi de suite
Taper : `D 400` et ainsi de suite

```
==============================
400     BRA     7032            ; GoTo ➔#Pre_CopyLock_#03
==============================
```

`Pre_CopyLock_#03`

```
==============================
7032    MOVE.W  #2700,SR        ; SR = $2700 (supervisor mode, no interrupts
7036    BRA     70F6            ; GoTo ➔ #Copylock_#03
==============================
```

`Copylock_#03`

```
==============================
70F4    LINEF                           ;
70F6    MOVE.L  A6,-(A7)                ;
70F8    LEA     7080(PC),A6             ; En général elle fait un peu plus de $520 Octets
70FC    MOVEM.L D0-D7/A0-A7,(A6)        ; Donc on peut regarder la fin de cette routine en 70F6+520=7616
7100    LEA     40(A6),A6               ;
7104    MOVE.L  (A7)+,8-(A6)            ;
7108    MOVE.L  10,D1                   ;
710E    PEA     411A(PC)                ; Bingo, Signature copylock
7112    MOVE.L  (A7)+,10                ;
7118    ILLEGAL                         ;
711A    PEA     7138(PC)                ; Bingo, Signature copylock
... ... ...
==============================
```

Bingo !
Et voilà notre fameuse 3eme routine de ***CopyLock*** identifiée.

Comme on l'a précédemment vu, les routines de CopyLock font en général un peu plus de $520 Octets
On regarde donc vers : **$70F4+$520=$7614**

Taper : `D 7616` et descendez dans le code, on cherche un jolie **RTE** avec juste avant un retour des valeurs du registre.
Bingo en $7638

`Fin Copylock_#03`

```
...
7616    BLS     7616
7618    LINEF
761A    MOVE.L  -63FD(A6),2900(A7)
7620    MOVE.L  70F2(PC),D0
7624    BMI     762A
7626    LINEF
7628    ORI.B   #F9,D2
762C    ORI.?   #8,SR
7632    MOVEM.L 707E(PC),D0-D7/A0-A6    ; Fin typique des routines
7638    RTE                             ; de copylock                          (Ici on est déjà retourné en T00.)
==============================
```

Les routines de *CopyLock* n'aimant pas vraiment les *BreakPoint*, on va passer par une *DeadLoop*
Taper :

```
A 7638
$007638  BRA    7638            ; Notre Dead-Loop
$00763A  <RETURN>
```

Puis on retourne code, taper : `X`

La routine de *CopyLock* va s'exécuter, aller en **T00** puis retourner en **Piste64**
Une fois que ceci sera fait, le code devrait être bloqué dans notre *Dead-Loop*

Entrer dans l'AR, on en profite pour noter l'état des registres, Taper : `R`

$7638			**Fin #copylock_#03**				
D0	7670CF6B	00000001	AAA9FFFF	50440000	00000000	00000000	0000FFFF
A0	00040000	00040E00	0006455C	0007F59E	00BFD000	00001082	0006456E	00074858

On n'oublie pas de remettre le code d'origine.
Taper :

```
A 7638
$007638  RTE
$00763A  <RETURN>
```

Il serait bien de savoir si le code en *$7032* en 'fin de code du second Trackload' est déjà présent ou s'il est lui aussi modifié/crée par une routine.
Pour cela je vous invite (**plus tard**) à rebooter votre amiga, passer l'Intro et pendant l'exécution du second TrackLoader, **entrer dans l'AR**, poser un *BreakPoint* en *$6424E.*
Retourner à l'exécution du code et attendre que notre *BreakPoint* soit atteint, puis, regarder le code en *$7632*
Vous verrez qu'effectivement, ce code n'existe pas encore cela veut dire que la phase de décryptage que l'on a vue traite aussi cette zone.

Mais revenons à notre analyse :

Analyse du Code : Menu Principal

Suivant notre analyse des chargements effectuées au début de ce tuto, à partir d'ici il n'y a plus de vérification de la **T00**
Donc, normalement, que du code 'standard' du jeu lui-même, on regarde quand même (dans le doute…)

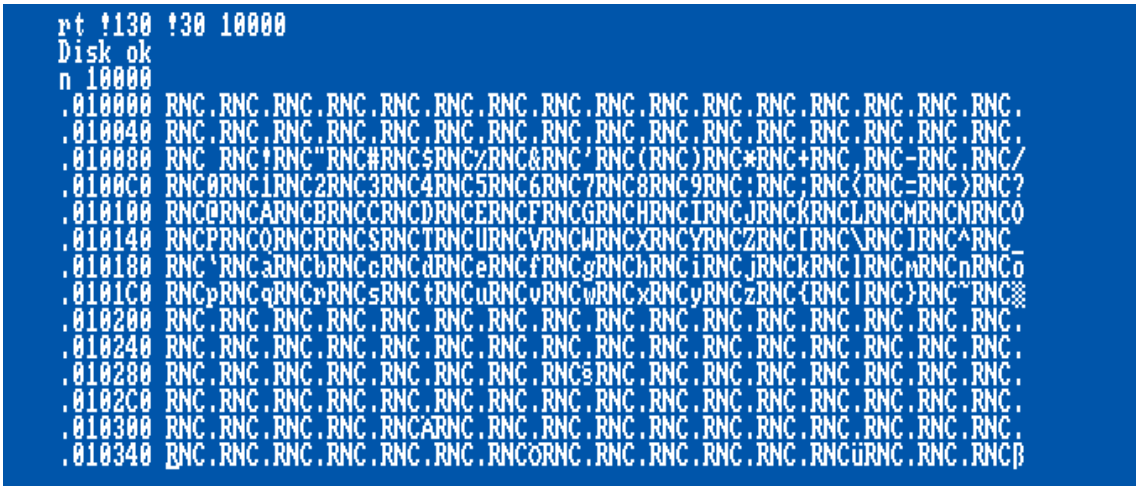
Base_Pre_menu_Principal

```
==============================
763A    ADD.W   D0,D1                   ;
763C    LEA     435FC,A7                ; A7=435FC
7642    LEA     8.S,A0                  ; A0=8
7646    LEA     7032.S,A1               ; A1=7032
764A    MOVE.L  A1,(A0)+                ; Copie A1 dans (A0) puis A0=A0+4
764C    CMPA.L  #400,A0                 ; Compare A0 avec 400
7652    BLT     764A                    ;
7654    BSR     6FE8                    ;
7658    MOVE.W  #0,404.S                ;
765E    MOVE.L  #4C700,406.S            ;
7666    MOVE.L  #44000,40A.S            ;
766E    MOVE.L  #FFFFFFFF,414.S         ;
7676    MOVE.L  #FFFFFFFF,418.S         ;
767E    MOVE.W  #4,410.S                ;
7684    ST      1A591                   ;
768A    BTST    #7,BFEE01               ; Timer du CIAA
7692    BNE     76A0                    ;
7694    MOVE.W  #5,410.S                ;
769A    ...                             ;
...
782A    BSR     8220                    ; GoSub ➔ #Lancement_Animation_Greeting
782E    BSR     8CB6                    ; GoSub ➔ #Menu_Principal                (Select, 1 player, 2 player, Music)
...
7852    BRA     917E                    ; GoSub ➔ #Loading_Level1_&_Insert_Disk_2
...

Et on sait que le second disque est au standard AmigaDOS
Donc à partir d'ici, on est sûr qu'il n'y aura pas d'autre protection de type CopyLock.
```

Suivant les diverses informations récoltées le long du tuto, on sait maintenant, à ce stade, que les chargements sur le disk1 de Xenon2
Sont linéaires et que la dernière Piste Lue est là 64eme.

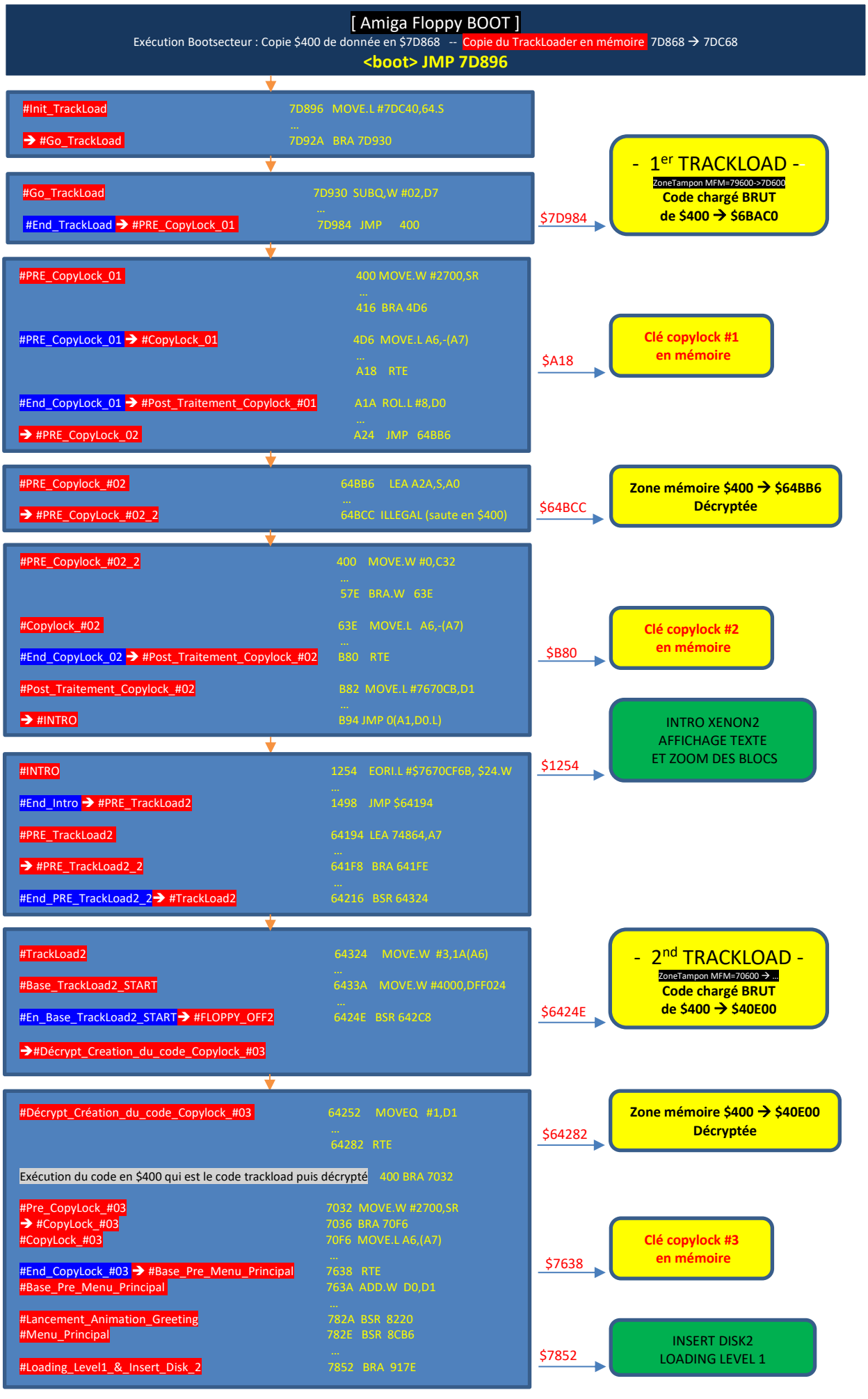
D'ailleurs il est tout à fait possible de regarder ce que contient les pistes suivantes, à savoir piste !65 à !79
Taper : `RT !130 !30 10000` et on regarde les données chargées `N 10000`



Vous pouvez jeter un coup d'œil sur toute la zone chargée, vous pourrez remarquer qu'il n'y a aucune donnée, aucun code (crypté ou pas)
Juste une 'signature' bidon.

Ceci est intéressant car cela nous donne un espace disponible pour une Grosse ***CrackTro*** ☺

Analyse du Code : Organigramme des diverses étapes



CRACK : Mode opératoire

Tout en se basant sur l'organigramme du précédent chapitre on va mettre en place notre mode opératoire.

Le Copylock #1 et **#2** étant intimement imbriqué l'un dans l'autre, il est logique d'intervenir à la suite du dernier executé.

L'idée va être de laisser la routine de décryptage $64BCC faire son job puis de sauver les données trackloadées sur une disquette vierge.

De modifier ce code pour qu'il ne lance pas la routine de CopyLock #2 mais juste notre nouvelle routine où l'on va redéfinir l'état des registres que l'on a noté en sortie de **CopyLock #2**.

De désactiver si nécessaire la partie décryptage qui suit le 1er TrackLoad (ça ne sera pas nécessaire pour info 😊)
De réécrire le tout sur notre disquette de backup.

Normalement, à ce stade on aura Bypassé le **CopyLock #1 et le CopyLock #2**

Même punition pour le **CopyLock #3** j'aurai tendance à dire.

On va laisser le Second Trackloader chargé et décrypter les données, puis les sauver sur une disquette,
les modifier et réécrire le tout sur notre disquette de backup.

La modification de celui-ci étant de désactiver la routine de décryptage
(vue que l'on va sauver et réécrire des données décryptées)
De patcher la routine **CopyLock #3** avec l'état des registres que l'on a noté en sortie de l'original **CopyLock #3**

Simple non ?!

Let's Rock'n Roll

Création du Crack étape par étape : Etape #1 – Bypass Copylock #01 et CopyLock #02

- **Booter sur le disque Original**.

- En plein trackload, **entrer dans l'AR** et **poser un BreakPoint en fin de trackload : BS 7D984**
  Revenir à l'exécution du code, à savoir le TrackLoad avec la commande **X**
  Notre BreakPoint est atteint et on entre automatiquement dans l'AR, le code en *$400* a été chargé/tracklodé à savoir : **#PRE_Copylock_#01**

- On le note : **M 400**
  46 FC 27 00 41 FA FF FA 43 F8 00 08
  Cela correspond au début du trackload effectué vers la mémoire en *$400*, ce sont les données BRUT lue sur le disque au format AmigaDOS (du moins ça devrait si on a bien analysé et compris le code)
  Théoriquement, ça devrait être en Track 2.
  En effet :

  Track0=Piste 00.0 Boot
  Track1=Piste 00.1 CopyLock
  Track2=Piste 01.0 <Début du trackload_#01>

- On va vérifier tout ça.
  Si on fait un : **RT 2 2 10000**, on devrait pouvoir trouver en mémoire notre chaine hexadécimal notée.
  F 46 FC 27 00 41 FA FF FA 43 F8 00 08, 10000
  Effectivement, on la trouve bien et au début de l'adr. $10000,
  donc au début de notre TrackLoad et en début de Track2.

  Cela valide deux choses :
  Que les données sont bien au Format lue normalement et son dispo tel qu'elle sur le disque.
  Qu'elles sont bien à l'endroit attendu (Track 2)

- On va donc laisser le code original faire son job et décrypter complétement ces données.
  Ce qui nous fait arriver en **#PRE_Copylock_#02_2**, juste avant le 'saut' en *$400*.

  Pour cela, **on reboot l'Amiga et pendant le TrackLoad on entre dans l'AR, on pose un BreakPoint en fin de TrackLoad : BS 7D984**

  On revient à l'exécution du code, à savoir le TrackLoad avec la commande **X**
  Notre BreakPoint est atteint et on entre automatiquement dans l'AR, on efface notre BreakPoint BDA
  A cette étape le trackload est fini et on s'apprête à effectuer un **JMP en $400**. (Voir organigramme)
  On modifie le code en *$64BF2* par une **DeadLoop**

  A 64BF2
  64BF2 BRA 64BF2
  64BF4 <ENTRER>

  Et on revient à l'exécution du code avec la commande **X**
  Assez rapidement nous sommes bloquées dans notre **DeadLoop**

- On entre dans l'AR et on remet le code original en *$64BF2*
  A 64BF2
  64BF2 DBF D1, 64BCC
  64BF6 <ENTRER>

  On peut vérifier que le code en *$400* est bien le même code décrypté vue **#PRE_Copylock_#02_2**
  M 400, effectivement c'est identique à ce que l'on a déjà vue et analysé.

- Le code **#PRE_Copylock_#02_2** finit normalement en *$CD4* (dernière sous routine de ce code)
  et fini par un **BRA 63E**, qui effectue un branchement vers la routine **#CopyLock_#02**

  On va donc modifier le code **#CopyLock_#02** en *$63E*
  La patcher avec ce que l'on relevé au niveau des registres précédemment.

$B80 **Fin #copylock_#02**

D0	7670CF6B	05550777	06660444	02220111	00000333	05550777	06660444	02220111
A0	00000B00	00000000	00064BCC	00FE86EE	00BFD000	00C014B6	0007DC4E	00064E46

- On écrase le code du **#CopyLock_#02**, **taper : A 63E**

```
63E  NOP                    ; Histoire de retomber sur nos pas
640  LEA      5C8(PC),A6    ; Identique au code d'origine
                            ;
644  MOVE.L   #7670CF6B,D0  ; ➔ Et on recopie les registres notés en fin de routine Fin #Copylock_#02
64A  MOVE.L   D0,24.S       ; Re-copie de la clé CopyLock en $24
64E  MOVE.L   D0,(A6)       ;
650  MOVE.L   #5550777,D1   ;
656  MOVE.L   #6660444,D2   ;
65C  MOVE.L   #2220111,D3   ;
662  MOVE.L   #333,D4       ;
668  MOVE.L   #5550777,D5   ;
66E  MOVE.L   #6660444,D6   ;
674  MOVE.L   #2220111,D7   ;
67A  LEA      B00,A0        ;
680  LEA      0,A1          ;
686  LEA      64BCC,A2      ;
68C  LEA      FE86EE,A3     ;
692  LEA      BFD000,A4     ;
698  LEA      C014B6,A5     ;
69E  LEA      7DC4E,A6      ;
6A4  LEA      64E4C,A7      ;
6AA  JMP      B82           ; On saute juste après en la fin de la routine d'origine
6AE           <RETURN>      ; 'Fin de la routine #Copylock_#02' en #Post_Traitement_Copylock_#02
                            ; Afin de retomber sur nos pattes
```

- Il nous reste plus qu'à sauver celui-ci **sur une disquette vierge**.
  Ne pas oublier que cela concerne toute la zone mémoire décryptée, donc de *$400* → *64BB6*
  Il faut donc sauver toute celle-ci.

  Il n'est pas utile de désactiver le code de décryptage en *$64BD2* car au moment où on le sauve, il s'est déjà désactiver tout seul.
  En effet, le **ILLEGAL** en *$64BCC* est déjà présent, donc il effectuera directement un saut en *$400* sans passer par la phase **EOR** (revoir plus haut si nécessaire)

- Disquette vierge insérée dans le second lecteur et on sauve.
  SM 1:TrackLoad_#1_hacked, 400 64BB6+2

- Maintenant on peut écrire ce code **sur notre disquette de backup réalisé sous Xcopy**
  On calcul combien de track il est nécessaire de sauver.

  $64BB6-$400= $647B6= !411574 — 1 track = $1600 = !5632
  Donc, 411574 / 5632 = 73.07 — (!74 Tracks)

- Comme il est impossible d'écrire avec la fonction **WT** sous l'AR une zone mémoire en *$400*
  On relie notre fichier en le déplaçant à une adresse plus haute : **LM 1:TrackLoad_#1_hacked, 10000**
  Et on réécrit le tout sur notre backup (swaper donc l'original en DF0 par votre Backup), puis
  WT 2 !74 10000

  ! Normalement, à ce stade, on devait avoir passer les deux 1ères routines de **CopyLock**.

- On va vérifier ça simplement, **rebooter votre Amiga toujours avec la disquette de backup dans DF0**
  Le trackload fonctionne correctement, ensuite est bypassé les routines de copylock (plus de check T00)
  L'intro se lance normalement et l'on arrive sur le second TrackLoad (que l'on n'a pas encore pacthé)
  Second trackload qui fonctionne normalement aussi (logique) mais s'en suit un crach de l'Amiga car la routine de **#CopyLock_#03** est exécuté et ne trouve pas ses petits. (Normal)

- Vous l'avez compris, la prochaine étape sera de patcher la routine de **#CopyLock_#03**

Création du Crack étape par étape : Etape #2 – Bypass Copylock #03

- **Booter de nouveau sur la disquette originale** et à la fin de l'intro principale, juste avant le début du second **trackload, entrer dans l'AR** et **poser un *BreakPoint* en fin de trackload**
  BS 6424C

  C'est pile poil dans la routine **#Avance_D'une_Track** juste pendant le test de <fin de donnée à Trackloader?>, comme ça, on va arriver sur notre ***BreakPoint*** dès la fin du 1er passage du second TrackLoad.

  On revient à l'exécution du code, à savoir le TrackLoad avec la commande **X**
  Notre BreakPoint est atteint très rapidement (logique) et on entre automatiquement dans l'AR
  Du code en *$400* a été chargé/tracklodé.

  Normalement, si vous avez bien suivi toute l'explication du tuto, vous savez que l'on vient de trackloader les données de la piste 41, on va vérifier ça.

- **On regarde en $400, taper : M 400** ,on trouve : **60 2A 0C 30 6C 00**
  Donc normalement c'est le début de la piste 41 = Track 82 (2 Tracks par Piste rappelé vous)
  On va vérifier ça, **taper : RT !82 1 10000**

  On regarde maintenant à l'adresse mémoire $1000, M 10000 et effectivement on retrouve notre chaine hexa (cool)

  Théoriquement il devrait lire *$40000* de donnée.
  dixit **641E8 MOVE.L #40000,A(A6)** *; 6456E+A=$64578 (compteur de donnée à copier)*

  $40000 = !262144 — 1 Track = $1600 = $5632 — 262144 / 5632=46,6
  !46 * !6532 = !259072 — !262144 - !259072 = !3072 = $C00
  Donc il va 'lire' $4000 + $C00 de donnée, soit *$40C00*

  Adr. de la zone mémoire final = $40000 + $400 = $40400 + $C00 = *$41000*
  En partant de *$400* ça nous donnes une zone mémoire : $400 → $41000

  En réel, (testé) = $40E00 (piste 41 à 64)
  Il a lu !512 octets en moins soit 1 secteur de moins que prévu, étrange !
  Mais bon, en fin de donnée ce n'est plus du code mais du texte de type 'RNC'
  Peut-être est-ce une question de longueur du buffer du trackloadeur, aller comprendre....
  Peu importe.

  Cela valide encore deux choses :
  Que les données sont bien au Format lue normalement et son dispo tel qu'elle sur le disque.
  Qu'elles sont bien à l'endroit attendu (Track 41)

- On va donc laisser le code original faire son job et décrypter complétement ces données.
  Ce qui nous devrait nous faires arriver en fin de routine **#Décrypt_Creation_du_code_Copylock_#03**,
  en *$64282* juste avant le 'saut' en *$400*.

- Pour cela, **on reboot l'Amiga et pendants le second TrackLoad** on **entre dans l'AR, on pose un BreakPoint en** *$64282* **: BS 64282**

  On revient à l'exécution du code, à savoir le second TrackLoad avec la commande **X**
  Notre BreakPoint est atteint et on entre automatiquement dans l'AR, on efface notre BreakPoint BDA

- Nous voici donc avec les données du second trackload décryptées en mémoire.
  Nous allons donc devoir faire quelque modification du code original, à savoir :
  Désactiver la routine de décrypt : **#Décrypt_Création_du_code_Copylock_#03**
  car on va réécrire ces données décryptées sur notre backup, il ne sera plus nécessaire de les décrypter.

- Cette routine se trouve en *$64252* donc normalement **cette adr. Mémoire se situe sur notre 1er rip** préalablement effectué : **TrackLoad_#1_hacked**

  Petit rappel :
  TrackLoad_#1_hacked = $400 $64BB6+2 **Piste 01->40**
  TrackLoad_#2_hacked = $400 $41000 **Piste 41->64**

  Mais pour l'instant, on va sauvegarder nos données décryptées
  SM 1:TrackLoad_#2_hacked, 400 41000+2

- Maintenant, allons donc patcher notre routine de décryptage, **taper : LM 1:TrackLoad_#1_hacked, 400**
  Il faut désactiver complètement les **EOR**, c'est à dire de de ***$6425A*** à ***$64284*** par des **NOP**

  Taper : A 64252

```
64252       MOVEQ    #1,D1            ; Code d'origine
64254       MOVEQ.L  24.S,A0          ; Code d'origine
64258       MOVEQ    #2A,D0           ; Code d'origine
6425A       NOP
6425C       NOP
6425E       NOP
64260       NOP
64262       NOP
64264       NOP
64266       NOP
64268       NOP
6426C       NOP
6426C       NOP
6426E       NOP
64270       NOP
64272       NOP
64274       NOP
64276       PEA      64252           ; Code d'origine
6427C       MOVE.W   (A7),-(A7)      ; Code d'origine
6427E       EORI.W   #E552,(A7)      ; Code d'origine
64282       RTE                      ; Code d'origine
64284       <RETURN>
```

- Et bien sûr re-sauver le tout avec la commande : **SM 1:TrackLoad_#1_hacked, 400 64BB6+2**
  Le recharger en $10000 : **LM 1:TrackLoad_#1_hacked, 10000**
  Et on réécrit le tout sur notre backup (swaper donc l'original en DF0 par votre Backup), puis
  WT 2 !74 10000

- On revient à nos moutons…
  Donc après avoir désactiver la routine **#Décrypt_Création_du_code_Copylock_#03**
  Le code devrait s'exécuté 'normalement' jusqu'à se retrouver dans la routine **#Copylock_#03** en *$70F6*

  On va donc patcher cette routine **#Copylock_#03** comme on l'a déjà fait sur le **1er CopyLock**
  Elle est située en *$70F6* en d'&autre terme, sur notre Second fichier de rip : **trackLoad_#2_hacked**
  On le recharge donc en mémoire, taper : **LM TrackLoad_#2_hacked, 4000**

- Nous allons la patcher avec ce que l'on relevé au niveau des registres précédemment,
  puis finir notre bout de code avec un **JMP en 763A.S.**

  JMP qui remplace le **RTE** du **#Copylock_#03** original et qui continuait en **#Base_Pre_Menu_Principal**
  (à savoir justement en *$763A.S*)

$7638 **Fin #copylock_#03**

D0	7670CF6B	00000001	AAA9FFFF	50440000	00000000	00000000	00000000	0000FFFF
A0	00040000	00040E00	0006455C	0007F59E	00BFD000	00001082	0006456E	00074858

- On écrase le code du **#CopyLock_#03**

  Taper : A 70F6

```
70F6   NOP
70F8   LEA      7080(PC),A6             ; Code d'Origine
70FC   MOVE.L   #7670CF6B,D0            ;
7102   MOVE.L   D0,24.S                 ; On copie la clé CopyLock en mémoire $24, au cas ou
7106   MOVE.L   D0,(A6)                 ;
7108   MOVE.L   #1,D1                   ;
710E   MOVE.L   #AAA9FFFF,D2            ;
7114   MOVE.L   #50440000,D3            ;
711A   CLR.L    D4                      ;
711C   CLR.L    D5                      ;
711E   CLR.L    D6                      ;
7120   MOVE.L   #FFFF,D7                ;
7126   LEA      40000,A0                ;
712C   LEA      40E00,A1                ;
7132   LEA      6455C,A2                ;
7138   LEA      7F59E,A3                ;
713E   LEA      BFD000,A4               ;
7144   LEA      1082,A5                 ;
714A   LEA      6456E,A6                ;
7150   JMP      763A.S                  ;
7154   <RETURN>
```

- On re-sauve le tout sur notre disquette de sauvegarde toujours en **DF1** : **SM 1:TrackLoad_#2_hacked, 400 41000+2**

  Et on re-écrie le tout sur notre disquette au bon endroit
  O 00, 10000 10000+41000+1600 (on nettoie un peu la mémoire)

  LM TrackLoad_#2_hacked, 10000 (on recharge le code en $10000**)**

  WT !82 !47 10000 (et on écrit le tout sur notre disquette de crack tjs présent dans DF0)

- Concernant la disquette 2, comme vu au début de ce tuto elle ne possède aucune protection, vous pouvez donc la copier directement avec X-Copy.

  Il ne vous reste plus qu'à rebooter votre Amiga avec notre disquette cracké et apprécier le jeu 😊
  Crack + jeu testé jusqu'à la fin du jeu pour info.