Tutoriel	**AmigaCracking : Shadow Of The Beast II**
Protection	**MFM + CRC + Autres**
Auteur Original	**Gi@nts**
Source original	**Giants**
Version	**Commencé en 28/10/2019 Gi@nts** **Done le 10/08/2020 (si si, ah faut ce qu'il faut)**
Vérification/Correction	**v1.0e** **Relecture : Amigars [AmigaImpact]**

* SHADOW OF THE BEAST II - CRACK TUTORIEL *

Table des matières

Matériels nécessaires

1) Un AMIGA 500 ou l'émulateur WINUAE.
2) Une petite floppée de disquettes vierges.
3) Un lecteur de disquette externe est fortement conseillé.
4) Une Carte ACTION REPLAY (ou sa ROM Image) selon configuration utilisée.
5) Le jeu Original en disquette ou son image CAPS (SPS 1359)
6) Code source du trackloader d'AlphaOne 2004 (v. 404 Bytes)
7) Code source du trackloader d'AlphaOne 2005 pro v2 (v. 460 Bytes)
8) Archive avec code source du cruncher/decruncher RNC propack (Aminet : util/pack/RNC_ProPack.lha)
9) Beaucoup de temps.

Infos Générales

On va essayer à travers ce tuto d'être le plus complet possible. (trop ?)
Ce tuto est basé sur mon travail d'analyse directement effectué sur l'original.
Il est à mon sens plus que complet.
Il est fortement conseillé de suivre dans l'ordre chronologique les étapes sinon vous risquez d'être perdu dans son appréhension.
Ne pas hésitez aussi à revenir sur des chapitres pour une meilleure compréhension de l'ensemble.

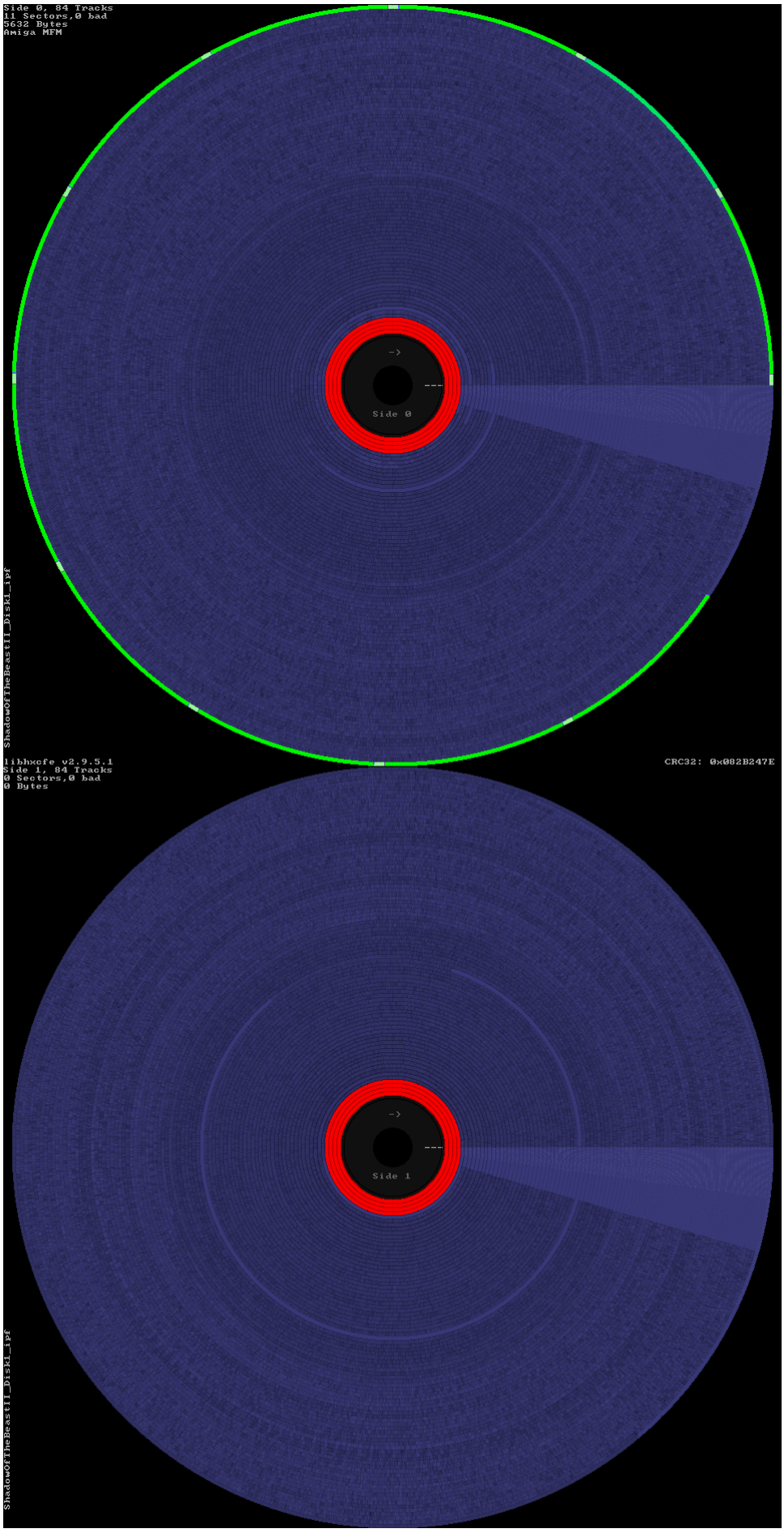
A noter que c'est un tutorial de crack et non pas un document sur 'comment apprendre l'assembleur sur Amiga'.
Le but est donc de comprendre les instructions sans pour autant toutes les détailler, de pouvoir comprendre le fonctionnement global du code afin de pouvoir isoler les parties en rapport avec le Hack (Trackloader, routine de CheckSum, CopyLock…).
Au vu de la taille du document, il existe sûrement quelques coquilles, ne pas hésiter à me faire un retour si nécessaire.
Attention, ce tuto c'est du lourd 😊

Au passage, un grand merci à Amigars du Forum AmigaImpact.org qui a effectuer une relecture et correction du Tuto.

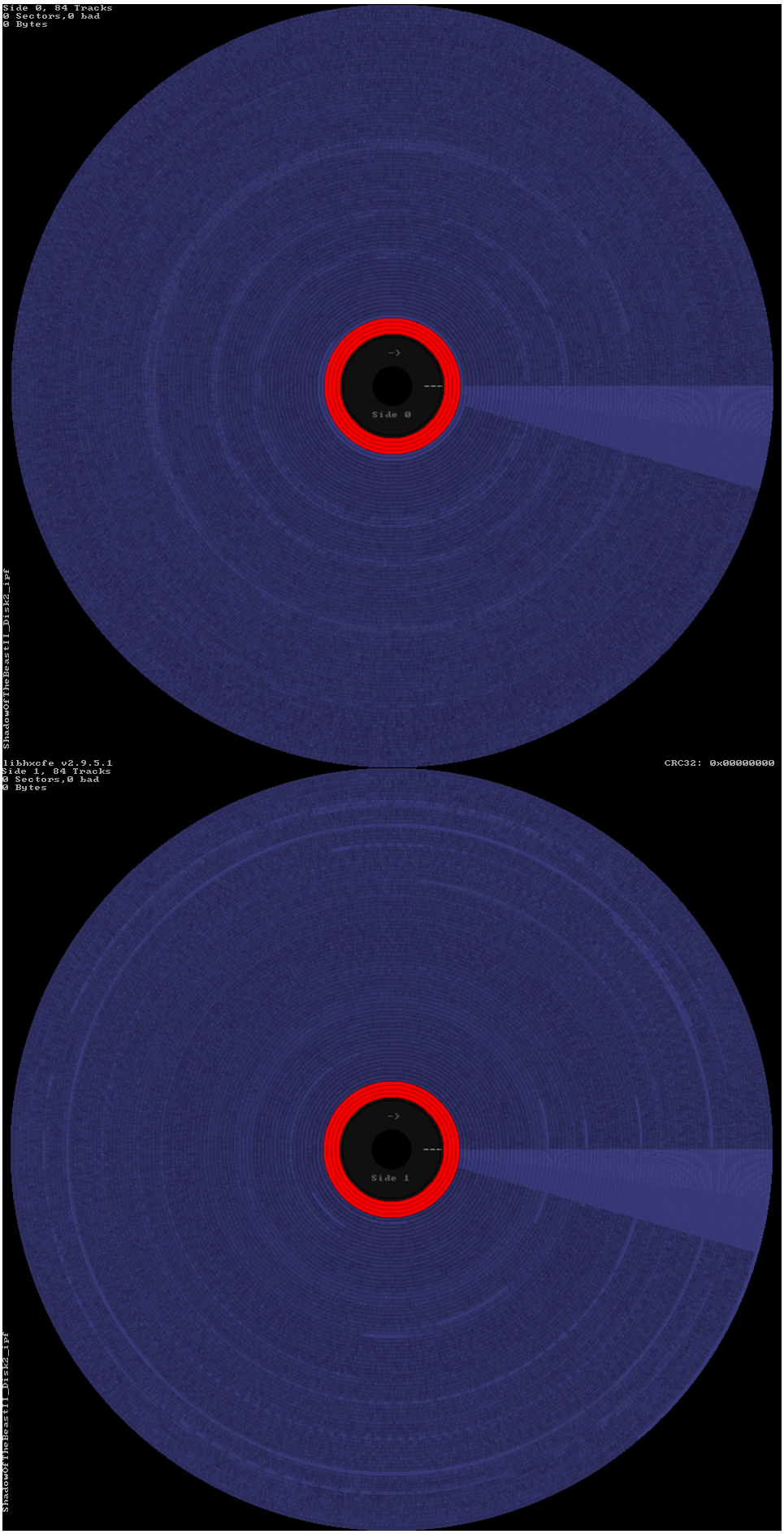
Bon Tuto.

Gi@nts

Disk 1


Side 0, 84 Tracks
11 Sectors,0 bad
5632 Bytes
Amiga MFM
Side 0
ShadowOfTheBeast_Disk1.ipf
libhxcfe v2.9.5.1
Side 1, 84 Tracks
0 Sectors,0 bad
0 Bytes
CRC32: 0x082B247E
Side 1
ShadowOfTheBeast_Disk1.ipf

Disk 2

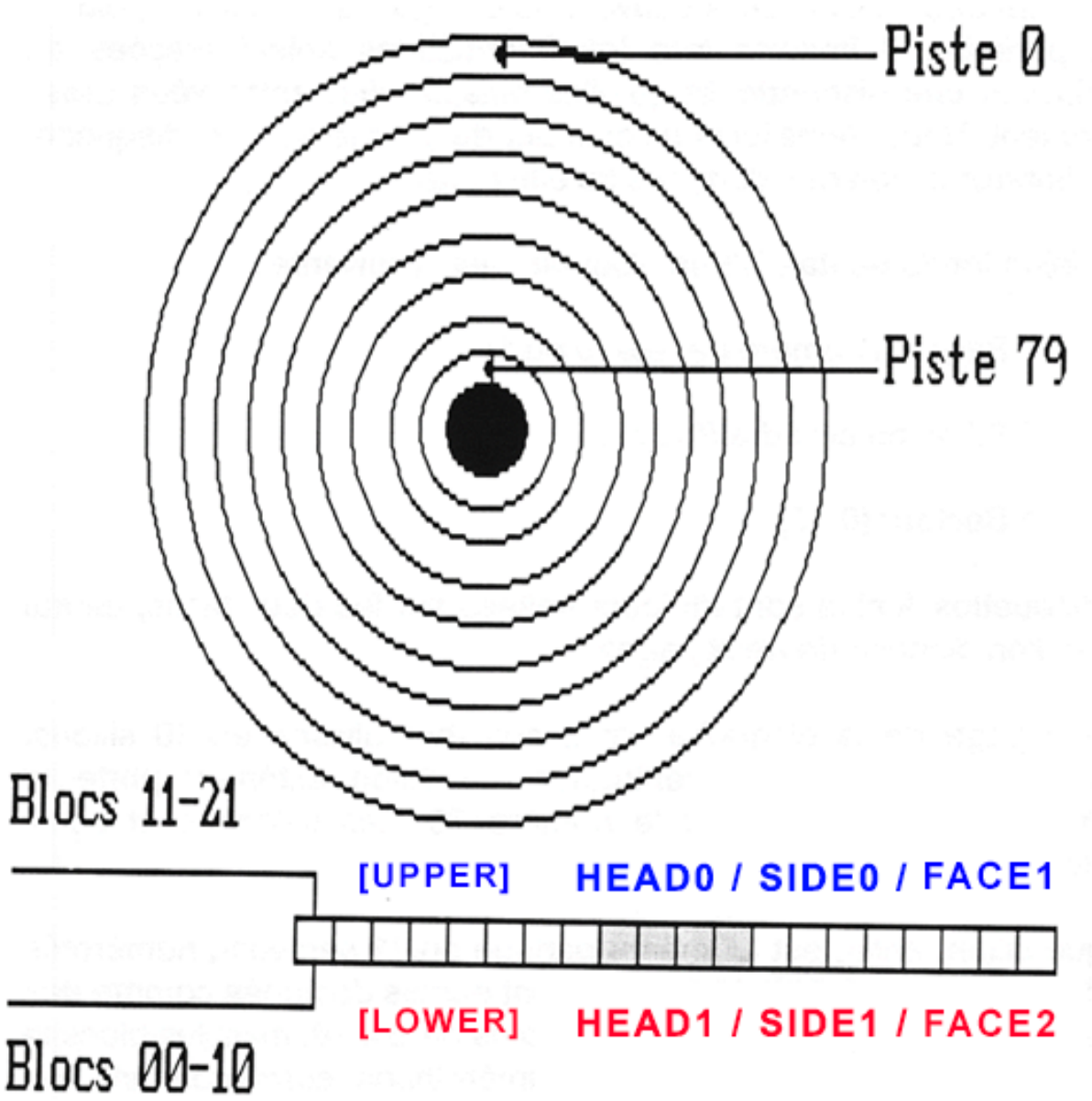

Side 0, 84 Tracks
0 Sectors,0 bad
0 Bytes
Side 0
ShadowOfTheBeast_Disk2_ipf
libhxcfe v2.9.5.1
Side 1, 84 Tracks
0 Sectors,0 bad
0 Bytes
CRC32: 0x00000000
Side 1
ShadowOfTheBeast_Disk2_ipf

Agencement des disquettes Amiga

En France :

On utilise des termes comme : *piste, bloc, secteur, face*…
Piste : 0 à 79 Certaines disquettes poussent 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 parties appelées **bloc**, **secteur**, **sector**.



Dans d'autre pays :

On utilisera d'autres termes, comme **sector, keys, track, 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 correspond 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 physique maximum.

Le nombre maximum de **sector** sur une piste dépend assez logiquement de la taille de ses **sectors.**

Pref…beaucoup de termes qui ne sont pas forcément utilisés 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 scanne 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 finit 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 située 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 ordonnés, 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 donne 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 et sauver des pistes, à savoir : **RT** et **WT**
Elles fonctionnent de la même manière.
L'une permet la lecture, l'autre l'écriture.

#**RT** alias Read Track. Permet le chargement de données situées 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 données situées 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 donne !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 retenue on est sur la face0.

RT 21 2 50000
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 données (**taille à lire =2**), on continue la copie.
On change donc de track car on est déjà sur la face 1 (il n'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 cela nous donne : 80*2*11=1760 Secteurs
Ainsi on peut avoir une position exacte en secteur sur un 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 dépend du 'trackloader' en question.

Le Format MFM

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

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

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

- S E C T O R -

00

2 octets
00 Octet soit $AAAA au format MFM

Sync

2 octets
(**A1**) converti au format MFM et 'Clock pulse' n'est pas pris en compte
Donc le résultat nous donne : **$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

À 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 compliquée 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 impairs  :  1 0 1 0
```

Cela permet un traitement des données plus rapide car l'extraction des moitiés paires et impaires peuvent être réalisées 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 la **Track.**
Le 'SyncWord' est en fait un 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 connaitre l'endroit où 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 lecteurs 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 logicielles ne sont pas précises (dépend 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 chipsets **CIA** pour attendre les 3ms nécessaires et il est de 18 ms entre un changement de direction.

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

- **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 effectuée, positionner d'abord ce bit à 0 et n'essayez 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 face. 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 utilisée 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 maintenu 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(s) lecteur(s) de disquette est(sont) utilisé(s).
Seulement les lecteurs sélectionnés sont affectés par les valeurs stockées 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 série.
Le lecteur connecté directement à l'ordinateur est le **drive 1** (DF1)
Le lecteur connecté 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 on doit positionner son bit correspondant à 0
Pour désélectionner un lecteur de disquette 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 sélectionnées 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 plusieurs lecteurs 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 désirez changer L'état de son moteur, vous devez désélectionner le lecteur, définir 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'autres opérations. 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.
```

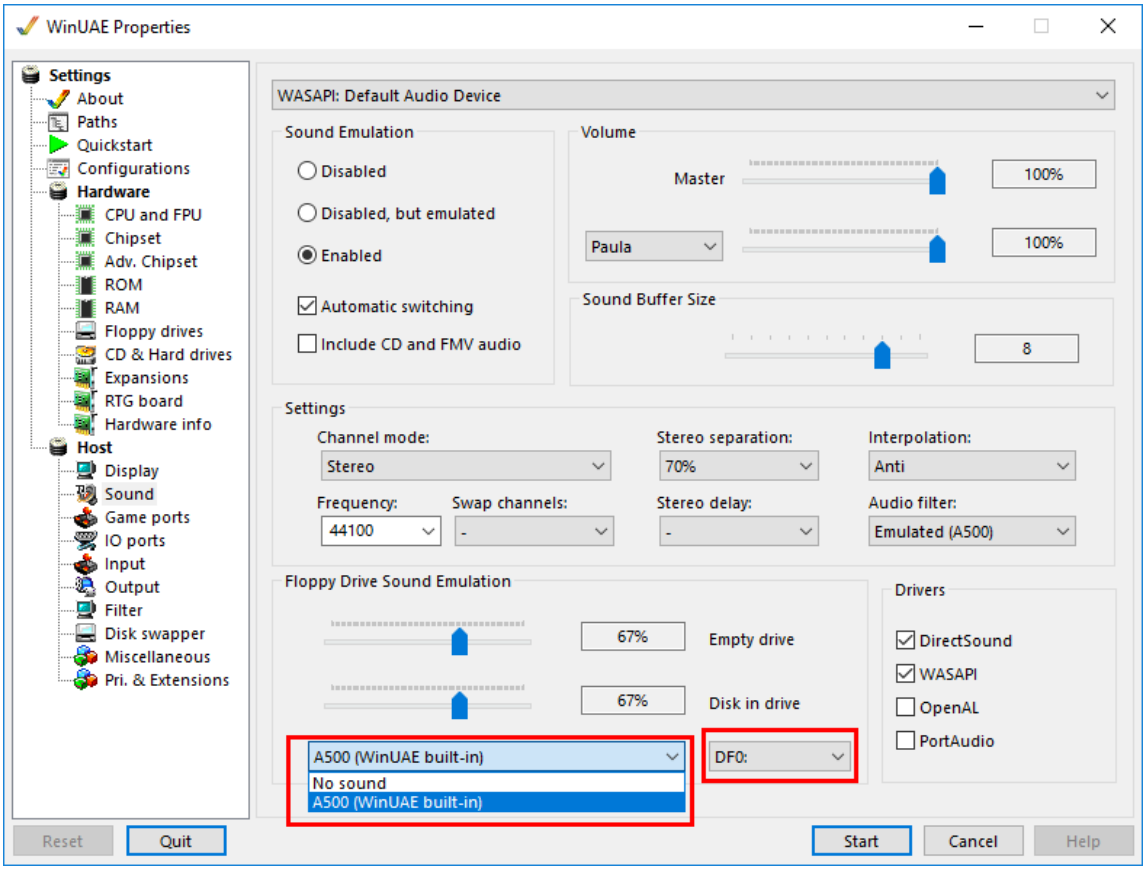
Extra Info :

Le registre **$DFF016 POTGOR**

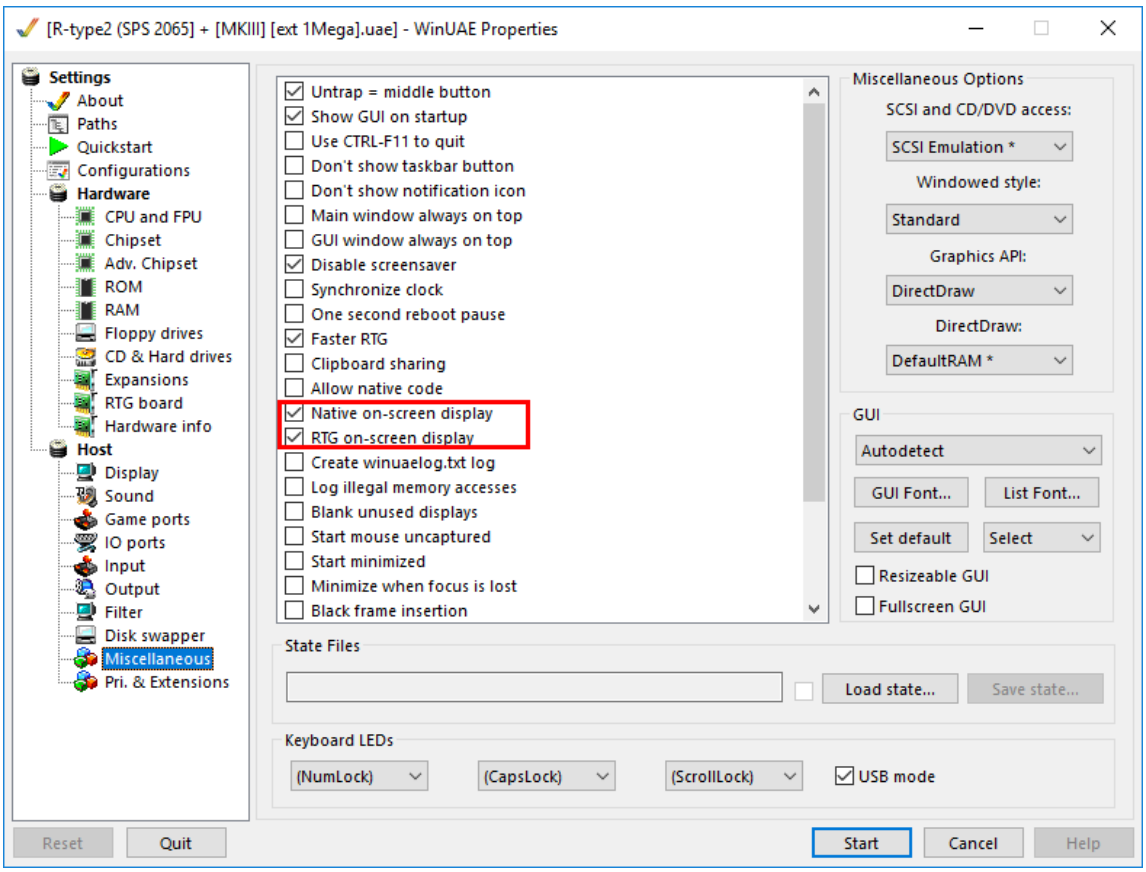
On va juste noter que sont **bit 10** sert pour **tester l'appuie sur le Bouton Droit de la souris**
Exemple : `btst #10, $dff016`

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



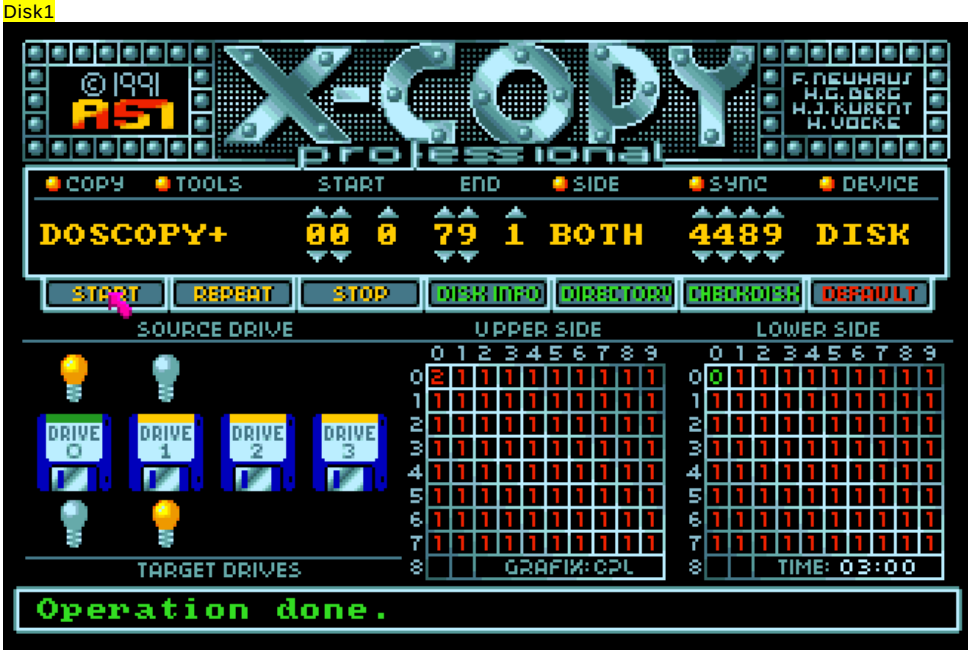
Voir même, pour plus d'informations comme afficher sur quelle face l'on se trouve :
Host -> Miscellaneous -> Native on-screen display AND RTG on-screen display



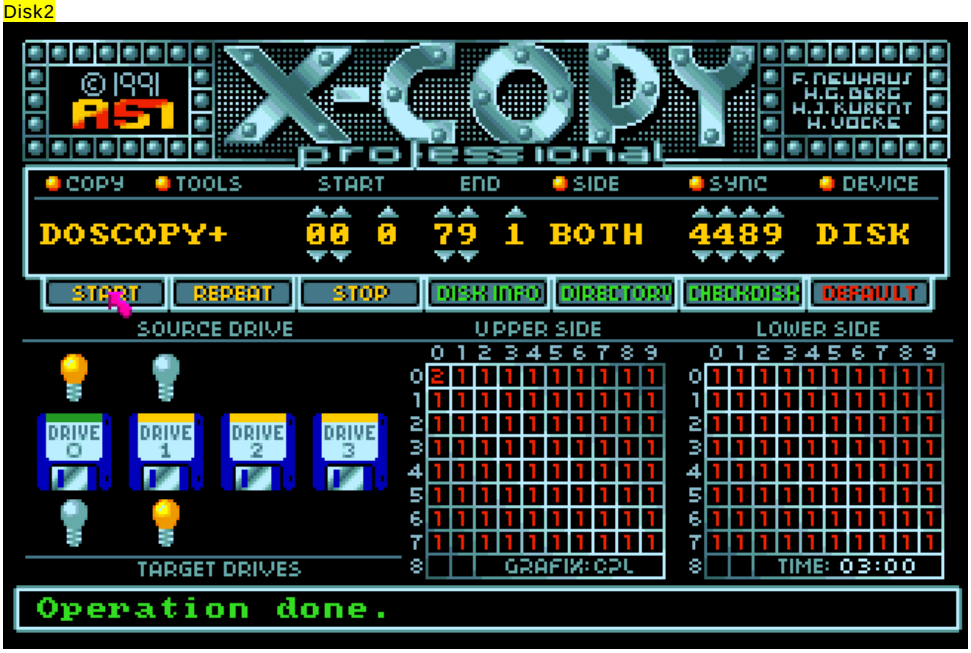
Part 1 X-copy

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

Disk1



Disk2



Hormis la track 00 de la 1er disquette, l'ensemble semble incopiable.
A noter une Track qui semble différente sur les deux disquettes en position **T00 Upper**, surement une piste dédiée à la signature des disques.
Il semble évident que nos copies ne fonctionneront pas. Néanmoins on va garder la copie de la 1er disquette afin de travailler sur son *bootSecteur*

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	1359_ShadowOfTheBeastII_D1_AMIGA.ipf
TYPE	Floppy_Disk
ENCODER	CAPS(V1)
FILE	1359(V1)
DISK	1
TRACK	00-83
SIDE	0-1
PLATFORM	Amiga
REVOLUTION	5
PROTECTION	Unknown, but they are many non standard Tracks

Concernant la Track 00.0 du 1er disque, rien de spécial : 11 secteurs de 512 bytes, standard AmigaDOS

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				4C164D7B			13576-20021
#0	0	8704	545	512	0	1	MFM	0352	0352	15	13576-13607
#1	1	8704	545	512	0	1	MFM	0906	0906	57	13608-13639
#2	2	8704	545	512	0	1	MFM	1460	1460	82	13640-13671
#3	3	8704	545	512	0	1	MFM	2014	2014	126	13672-13703
#4	4	8704	545	512	0	1	MFM	2568	2568	161	13704-13735
#5	5	8704	545	512	0	1	MFM	3122	3122	196	13736-13767
#6	6	8704	545	512	0	1	MFM	3676	3676	230	13768-13799
#7	7	8704	545	512	0	1	MFM	4230	4230	265	13800-13831
#8	8	8704	545	512	0	1	MFM	4784	4784	300	13832-13863
#9	9	8704	545	512	0	1	MFM	5338	5338	334	13864-13895
#10	10	8704	545	512	9872	618	MFM	5892	5892	369	13896-13927

Ensuite et sur l'ensemble des deux disquettes on retrouve un format unique et propriétaire jusqu'à la fin du disque.
(**Souligné en vert ci-dessous**)
Taille des **Data : 6352**

Disque 1

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	
[T00.1]		6048		48384				39DE4C18			20050-26097
#0	N/A	96112	6008	N/A	15376	962	MFM	0032	0032	2	20050-20081
[T01.0]		6352		50816				A1D37604			26126-32477
#0	N/A	100976	6312	N/A	4640	291	MFM	0032	0032	2	26126-26157
[T01.1]		6352		50816				477D023B			32506-38857
#0	N/A	100976	6312	N/A	4640	291	MFM	0032	0032	2	32506-32537
[T02.0]		6352		50816				1292FFE8			38886-45237
#0	N/A	100976	6312	N/A	4640	291	MFM	0032	0032	2	38886-38917
[T02.1]		6352		50816				2EC87572			45266-51617

Seul les Track **T00.1** du disque 1 et 2 sortent du lot, on peut le voir sous X-copy avec une **Error Type 2**
Comme sous mon analyseur d'image IPF au niveau de la taille des Data : **6048**
(souligné en rouge dans l'images ci-dessus du Disque1 et ci-dessous du Disque2)

Disque 2

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]		6352		50816				616D61B5			13576-19927
#0	N/A	100976	6312	N/A	4616	289	MFM	0032	0032	2	13576-13607
[T00.1]		6048		48384				6ACDE18D			19956-26003
#0	N/A	96112	6008	N/A	15344	960	MFM	0032	0032	2	19956-19987
[T01.0]		6352		50816				89FBE992			26032-32383
#0	N/A	100976	6312	N/A	4608	289	MFM	0032	0032	2	26032-26063
[T01.1]		6352		50816				F5E7E137			32412-38763
#0	N/A	100976	6312	N/A	4616	289	MFM	0032	0032	2	32412-32443
[T02.0]		6352		50816				F8D0C550			38792-45143

Ce qui confirmerait l'idée que l'on trouve la signature des disquettes en **T00.1** et non des Data

Quoi qu'il en soit, les deux disquettes semblent bien remplies.

Part 3 Cheats

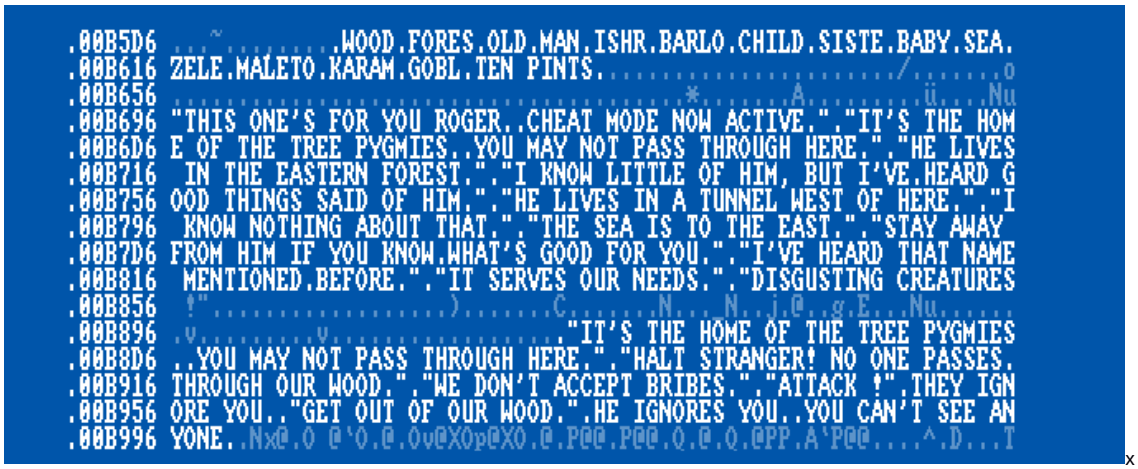
Touche [S] pendant une partie permet d'afficher le **SCORE** et **MONEY.**

```
$2B1 → $2B3                    : MONEY, codé en décimal, maximum = 99 99 99
$2B5 → $2B7                    : SCORE, codé en décimal, maximum = 99 99 99

$2AF                           : Energie. Assez vicieux à trouver, le compteur est inversé.
                                 $00=Energie Full
```

Un **MS 2AF** permet de tomber sur `$006FCE  DBF D5,6FC6` on remonte dans le code (1 ligne au-dessus) et on tombe sur
`$006FCA  ADDQ.W #4,2AE.S` que , pour avoir l'Energie infinie, l'on remplace par
`$006FCA  TST.W 2AE.S`

Touche [Q] ou [A] (selon le clavier de l'Amiga) pendant une partie permet de créer une discussion.
Uniquement possible lorsque l'on rencontre une vie intelligente.

F "CHEAT" nous donne un seul résultat ➔ *$B6B1*
On remonte un peu plus haut et on voit une zone de texte. Des **mots** et des **phrases**.


x

Directement sous le jeu il est donc possible d'ouvrir un dialogue lors d'une rencontre avec une vie intelligente.
Exemple : Au début du jeu, tout à droite on rencontre un indigène avec une lance.
En essayant les mots ci-dessus (WOOD, FORES, OLD…) il est possible de créer le 'tableau' suivant

```
???          "I DON'T UNDERSTAND THAT."
WOOD         "IT'S THE HOME OF THE TREE PYGMIES. YOU MAY NOT PASS THROUGH HERE."
FORES        "IT'S THE HOME OF THE TREE PYGMIES. YOU MAY NOT PASS THROUGH HERE."
OLD          "HE LIVES IN THE EASTERN FOREST."
MAN          "HE LIVES IN THE EASTERN FOREST."
ISHR         "HE LIVES IN A TUNNEL WEST OF HERE."
BARLO        "I KNOW LITTLE OF HIM, BUT I'VE HEARD GOOD THINGS SAID OF HIM."
CHILD        "I KNOW NOTHING ABOUT THAT."
SISTE        "I KNOW NOTHING ABOUT THAT."
BABY         "I KNOW NOTHING ABOUT THAT."
SEA          "THE SEA IS TO THE EAST."
ZELE         "STAY AWAY FROM HIM IF YOU KNOW WHAT'S GOOD FOR YOU."
MALETO       "I'VE HEARD THAT NAME MENTIONED BEFORE."
KARAM        "IT SERVES OUR NEEDS."
GOBL         "DISGUSTING CREATURES !"
TEN PINTS    "THIS ONE'S FOR YOU ROGER. CHEAT MODE NOW ACTIVE."
```

Exemple, lors d'un rencontre nécessitant un mot de passe et en utilisant la même méthode on tombe aussi sur une chaine **ASCII**.
F "WHISPER" nous donne un seul résultat ➔ *$E03E* *$E46C* *$E49C* *$E4CE* *$E500*

```
E03E : Fin d'un ensemble de donnée ascii, rien de suite après.

$E46C        WHISPER THE WORD "OBERON"."THANK YOU STRANGER..
$E49C        WHISPER THE WORD "ETHERNITY"."THANK YOU STRANGER..
$E4CE        WHISPER THE WORD "SUNSTONE"."THANK YOU STRANGER..
$E500        WHISPER THE WORD "NECROPOLIS".AT THE GATE AND I'M SURE MY.MASTER BARLOOM WILL REWARD YOU.
```

Level_1	Base Level	(Land of Karamoon)
Level_2	Cave	(FROM HOUSE OF CAVOON)
Level_3	Lift_From_Cage	(After Door must be open with key)
Level_4	Barloom Level	
Level_5	Crystal_Caverns	
Level_6	Water_Vortex	
Level_7	Castle	

Tips : Entrer dans l'AR pendant l'animation et faire un **G 638** pour sauter toutes les anims. (voir fin Part8 pour plus d'info)

Part 4 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 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.

Piste Lue	Affichage Après chargement ou info	Piste Lue WinUAE
00➔	**BOOT**	00
➔47	**N/A**	N/A
24	***N/A***	47-70 **Lower**
➔ 00	***ANIMATION PSYGNOSIS***	N/A
47	***ANIMATION PART #1***	01-47 **Lower**
➔ 00	***N/A***	N/A
48	***ANIMATION PART #2***	01-48 **Upper**
➔ 70	***N/A***	N/A
7	***N/A***	70-76 **Lower**
➔ 00	**INSERT DISK2**	N/A
INSERT DISK 2		
1	**N/A**	1
➔ 67	**N/A**	N/A
3	**N/A**	67-69
2	**N/A**	70-71
➔ 64	**N/A**	N/A
➔ 00	**N/A**	N/A
1	**N/A**	01-03
1	**N/A**	04-04
10	**N/A**	05-14
1	**N/A**	15-15
2	**N/A**	15-16
MENU PRINCIPAL		
PRESS FIRE		
9	**N/A**	16-24
2	**N/A**	24-25
9	**N/A**	26-34
2	**N/A**	35-36
2	**N/A**	36-37
➔ 00	**N/A**	N/A
LEVEL1		

Noter que ce n'est pas forcément 'exact' à une unité près 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.

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

Nbr Piste Lu	Affichage Après chargement ou info	Piste Lue WinUAE
00➔	**BOOT**	00
➔47	**N/A**	N/A
- ECRAN ROUGE -		

Part 5 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**
+c car le code du bootblock commence à cette adresse, avant c'est la signature disque AmigaDOS

Regardons ça en détail :

```
2000C    MOVEA.L  A1,A5                  ; Sauvegarde de l'adr. du trackdisk.device présent dans A1 au boot dans A5
2000E    MOVE.L   #1200,D0               ; D0     = 1200
20014    MOVE.L   D0,24(A1)              ; $24(A1)= 1200 Nombre de Bytes à réserver
20018    MOVE.L   #2,D1                  ; D1     = 2 (req)
2001E    MOVEA.L  4.S, A6                ; ExecBase dans A6
20022    JSR      -C6(A6)                ; Appel AllocMem (réservation mémoire)

20026    MOVE.L   D0,28(A5)              ; 28(A5) = Adr de l'AllocMem dans D0   Memory Destination Adr. for trackdisk.device
2002A    BEQ      200D8                  ; Problème ? GoTo ➔ #Crash_red_screen
2002E    MOVEA.L  D0,A4                  ; A4=D0  = $59E8  On récupère l'adresse de la zone réservée
                                         ; pour info : Amiga500 + Extension mémoire 512K = Zone mémoire $59E8
                                         ; et $A498 sans ext. Mémoire.
20030    MOVE.L   #400,2C(A5)            ; 2C(A1) = $400   Adr. Disk de départ.
20038    MOVE.W   #2,1C(A5)              ; 1C(A5)= Mode lecture
2003E    MOVE.L   A5,A1                  ; On restaure A1 avec l'adr. Sauvegardée du trackdisk.device
20040    JSR      -1C8(A6)               ; Appel du Trackdisk.device pour le chargement de $1200 bytes
20044    TST.L    D0                     ; Test D0
20046    BNE      200D8                  ; Problème de chargement ? GoTo ➔ #Crash_red_screen
                                         ;
                                         ; #Base_Conf
2004A    MOVE.W   #7FFF,D6               ; D6=$7FFF
2004E    LEA      DFF000,A6              ; A6=DFF000
20054    MOVE.W   D6,9A(A6)              ; DFF09A=Conf. INTENA // Disable all interrupts
20058    MOVE.W   D6,96(A6)              ; DFF096=Conf. DMACON // Disable all DMA
2005C    LEA      20068(PC),A0           ; A0=20068(PC)
20060    MOVE.L   A0,20.S                ; A0=20
20064    MOVE.W   #2700,SR               ; Conf Status Register
20068    MOVE.W   #2700,SR               ; Conf Status Register
2006C    LEA      BFDA00,A1              ; CIAB, A1=todhi (Horizontal sync event bit 23-16)
20072    BSET     #7,400(A1)             ; CIAB, cra (Control register A), set bit à 1
20078    MOVEQ    #0,D2                  ; D2=00
2007A    MOVEQ    #0,D3                  ; D3=00

                                         ; #Conf Sync #1
2007C    MOVE.W   D6,9C(A6)              ; Conf INTREQ à $7FFF // Clear all pending interrupts.
20080    BTST     #5,1F(A6)              ; ➔ Test bit 5 Intreq register (vbl)  (Vertical Blank Line)
20086    BEQ      20080                  ; ➔ tant que la VLB est pas atteinte, on boucle
20088    MOVE.B   (A1),D3                ; D3=(A1), CIAB, A1=todhi (Horizontal sync event)
2008A    SWAP     D3                     ; Inverse en longword D3 00001111 devient 11110000 par exemple
2008C    MOVE.B   -100(A1),D3            ; Adresse pointée par A1 (BFDA00) -100 = BFD900 = CIAB, todmid mis dans D3
20090    LSL.W    #8,D3                  ; Décalage de 8bit vers la gauche de D3
20092    MOVE.B   -200(A1),D3            ; Adresse pointée par A1 (BFDA00) -200 = BFD800 = CIAB, todlo mis dans D3

                                         ; #Conf Sync #2
20096    MOVE.W   D6,9C(A6)              ; Conf INTREQ à $7FFF // Clear all pending interrupts.
2009A    BTST     #5,1F(A6)              ; ➔ Test bit 5 Intreq register (vbl)  (Vertical Blank Line)
200A0    BEQ      2009A                  ; ➔ tant que la VLB est pas atteinte, on boucle
200A2    MOVE.B   (A1),D2                ; D2=(A1), CIAB, A1=todhi (Horizontal sync event)
200A4    SWAP     D2                     ; Inverse en longword D2 00001111 devient 11110000 par exemple
200A6    MOVE.B    -100(A1),D2           ; Adresse pointé par A1 (BFDA00) -100 = BFD900 = CIAB, todmid
200AA    LSL.W    #8,D2                  ; Décalage de 8bit vers la gauche de D2
200AC    MOVE.B   -200(A1),D2            ; Adresse pointée par A1 (BFDA00) -200 = BFD800 = CIAB, todlo mis dans D3
200B0    SUB.L    D3,D2                  ; D2=D2-D3
200B2    CMP.W    #11F, D2               ; Compare 011F avec D2. Flag N est défini si résultat est plus 'petit que'
200B6    SLT      C0.S                   ; Si résultat 'plus petit que' alors on branche en C0

                                         ; #Copie_code_vers_70000
200BA    LEA      70000,A7               ; A7=70000
200C0    MOVEA.L  A7,A2                  ; A2=A7=70000
200C2    MOVE.W   #47F,D0                ; D0=47F (compteur)
200C6    MOVE.L   (A4)+,(A2)+            ; ➔ copie en LongWord en (A4) vers (A2), puis A4=A4+4 et A2=A2+4
                                         ;    Donc copie les données lue et copier en $59E8 vers la zone mémoire $70000
200C8    DBF      D0,200C6               ; ➔ tant que D0 est diffèrent de -1, on boucle
                                         ;    ($47F * 4) + 4 = $1200  Bytes copiés vers $70000
                                         ;
200CC    MOVE.L   A7,10.S                ; Adresse de la pile copiée à l'adresse $10
200D0    MOVEQ    #8,D0                  ; D0=8
         ---------------------------------- Bon, ça l'AR ne comprends pas, en vérité ça nous donne ça ➔
         200D2    LINEF                  ; 4E 7B          ➔ 200D2   MOVEC D0,CACR   ;
         200D4    ORI.B    #D7,D2        ; 00 02 4E D7    ➔ 200D6   JMP (A7)        ; On saute en $70000
         ------------------------------------------------------------------------
200D8    MOVE.W   #F00,180(A6)           ; ➔ #Crash_red_screen
200DE    BRA      200D8                  ; ➔ Dead Loop
================================
```

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

#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 'Track', ak $1600

Puisque qu'après un appel du *Trackdisk.device* il copie et exécute le code en *$70000*, on va changer celui-ci à cette adresse.

Taper :

```
A 20000+400
$020400  MOVE.W   #0F0,DFF180          ; ➔ Fond d'écran en vert
$020408  BRA      20400                ; ➔ Notre Dead-Loop
$02040A  <RETURN>
```

Pour info ➔

```
d 20000+400
~020400 MOVE.W  #2700,SR
~020404 MOVE.W  #7FFF,00DFF096
~02040C MOVE.W  #8210,00DFF096
~020414 MOVE.W  #7FFF,00DFF09A
~02041C MOVE.W  #7FFF,00DFF09C
~020424 CLR.W   00DFF180
```

On re-calcul le checksum

```
BOOTCHK 20000
```

et **toujours** avec la **disquette de backup** dans le lecteur, on écrit ce nouveau *bootblock*

```
WT 0 1 20000
```

Rebooter ensuite votre Amiga normalement, on arrive très rapidement sur notre fond d'écran vert.
Entrer dans l'AR, on remet le code original.

Taper :

```
A 70000
$070000  MOVE.W #2700,SR
$070004  MOVE.W #7FFF,DFF096
$07000C  <RETURN>
```

Part 6 Analyse du TrackLoader #1

On jette un coup d'œil au code en *$70000*, c'est un **TrackLoader**.
Après étude du code, voilà ce que l'on peut retenir.

Taper D 70000

#Pre_Conf_TrackLoader

```
70000    MOVE.W   #2700,SR          ; Conf Status Register
70004    MOVE.W   #7FFF,DFF096      ; Conf. DMACON // Clear all pending interrupts.
7000C    MOVE.W   #8210,DFF096      ; Conf. DMACON
70014    MOVE.W   #7FFF,DFF09A      ; Conf INTENA // Disable all interrupts.
7001C    MOVE.W   #7FFF,DFF09C      ; Conf INTENA // Clear all pending interrupts.
70024    CLR.W    DFF180            ; Fond d'écran en Noir
7002A    BSET     #1,BFE001         ; CIA-A / PRA, On test le bit5, lecteur prêt ?
70032    LEA      256.S,A7          ; A7=256, changement d'adresse de la pile
70036    BSR      70078             ; GoSub ➔ #Trackloader_Init
                                    ;
7003A    LEA      45610,A0          ; A0=45610
                                    ; A0 est utilisé dans routine de #Decrypt_decomp_Init
                                    ; donc A0=adresse de destination pour les données lues brut (donc non décryptée)
                                    ;
                                    ; $45610->$69894  Donnée non décryptée/décompressée = $24284 = !148100
                                    ;
70040    LEA      70060(PC),A1      ; A1=70060(PC)                    A1=70060 (position dans la table)
70044    BSR      701E2             ; GoSub ➔ #Trackloader_Start
                                    ; A0=Adr_dest_mémoire             A1=Pointeur table Adress_Start_raw
                                    ;
                                    ; Les données sont maintenant chargées, il faut les décompacter/décrypter
70048    LEA      2B0.S,A1          ; A1=$2B0                         A1=Position mémoire final (après décrypt/decomp)
7004C    BSR      70310             ; GoSub ➔ #Decrypt_decomp_Init
                                    ;
                                    ; MAJ de la table pour les prochains appels
70050    LEA      70068(PC),A0      ; A0=70068(PC)                    A0=position dans la table
70054    MOVE.L   (A0)+,C2.S        ; Copie du contenue de A0 vers $C2, donc $C2= 0006CA94          /!\ important
70058    MOVE.L   (A0)+,C6.S        ; Copie du contenue de A0 vers $C6, donc $C6= 00009B78          /!\ important
7005C    JMP      2B0.S             ; On saute en $2B0                          ➔ // Lancement de l'animation psygnosis
```

#Table_#01

```
70060    00 04 88 10               ← Adresse_Start_raw, (Minimum $189C, taille d'une Track custom)
                                     (Obliger de commencer en Track 1 minimum (car Track en 0 Spécifique)
70064    00 02 42 84               ← Length_To_Read
70068    00 06 CA 94               ← Mis en $C2, aucune idée.
7006C    00 00 9B 78               ← Mis en $C6, aucune idée.
70070    00 00                     ← Pointeur Track en cours.
70072    00 00                     ← Marqueur pour Trackloader ou pas.
70074    00 06 A0 00               ← Conf DSKPTH
```

#Trackloader_Init

```
70078    MOVE.W   #10,DFF096        ; Conf DMACON
70080    MOVE.W   #2,DFF09C         ; Conf INTENA
70088    MOVE.W   #7FFF,DFF09E      ; Conf ADKCON
70090    MOVE.W   #8100,DFF09E      ; Conf ADKCON
70098    ORI.B    #78,BFD100        ; 'Ou' binaire entre 0111 1000 et  $BFD100, donc DF0 a DF3 select et Motor ON
700A0    BCLR     #7,BFD100         ; CIA-B / PRB, bit7 mis à 0, motor On
700A8    BCLR     #3,BFD100         ; CIA-B, bit3 à 0, DF0 sélectionné
700B0    MOVE.W   #BB8,D6           ; ➔ D6=BB8
700B4    DBF      D6,700B4          ; ← D6=D6-1, boucle tant que D6 est différent de -1. Nous avons ici une pause
700B8    BTST     #5,BFE001         ; ➔ CIA-A / PRA, On test le bit5, lecteur prêt ?
700C0    BNE      700B8             ; ← Tant que lecteur pas prêt, on boucle pour attendre qu'il le soit.
700C2    LEA      70072(PC),A3      ; A3=70072(PC) Adresse dans la table.
700C6    MOVE.W   #1,(A3)           ; Copie le word 1 dans (A3), donc met 00 01 dans la table en 70072
700CA    RTS                        ; E.T Retour maison
```

#Move Inter.

```
700CC    BCLR     #1,BFD100         ; CIA-B / PRB, bit1 mis à 0, DIR=vers l'intérieur
700D4    BCLR     #0,BFD100         ; CIA-B / PRB, bit0 mis à 0, STEP, pré/déplacement tête
700DC    BSET     #0,BFD100         ; CIA-B / PRB, bit0 mis à 0, STEP, déplacement tête
700E4    BSR      70170             ; GoSub ➔ #WAIT
700E8    MOVE.L   A0,-(A7)          ; On Sauve A0 dans la pile
700EA    LEA      70070(PC),A0      ; 70070(PC)=A0 Adresse dans la table
700EE    ADDQ.W   #1,(A0)           ; (A0)=A0+1 Word
                                    ; On s'est déplacé d'une Track vers l'intérieur et (A0), ak $70070 est incrémenté de 1
                                    ; $70070 est donc un pointeur de Track
                                    ; On peut parler aussi en Cylindre vue que l'on se déplace que sur une face.
                                    ; Donc, ici, quand on avance d'une track, on avance aussi d'1 cylindre
                                    ;
700F0    MOVEA.L  (A7)+,A0          ; On restaure A0 de la pile
700F2    BRA      70164             ; GoTo #Update_table
```

#Move Exter.

```
700F6    BSET     #1,BFD100          ; CIA-B / PRB, bit1 mis à 0, DIR=vers l'extérieur
700FE    BCLR     #0,BFD100          ; CIA-B / PRB, bit0 mis à 0, STEP, pré/déplacement tête
70106    BSET     #0,BFD100          ; CIA-B / PRB, bit0 mis à 0, STEP, déplacement tête
7010E    BSR      70170              ; GoSub ➔ #WAIT
70112    MOVE.L   A0,-(A7)           ; On Sauve A0 dans la pile
70114    LEA      70070(PC),A0       ; 70070(PC)=A0 Adresse dans la table
70118    SUBQ.W   #1,(A0)            ; On s'est déplacé d'une Track vers l'extérieur et (A0), ak $70070 est décrémenté de 1
                                     ; $70070 est donc un pointeur de Track
                                     ; On peut parler aussi en Cylindre vue que l'on se déplace que sur une face.
                                     ; Donc, ici, quand on avance d'une track, on avance aussi d'1 cylindre
                                     ;
7011A    MOVEA.L  (A7)+,A0           ; On restaure A0 de la pile
7011C    BRA      70164              ; GoTo ➔ #Update_table
```

#Position Atteinte?

```
70120    CMP.W    70070(PC),D0       ; Compare D0 avec $70070(PC), l'adresse lue dans la table
70124    BEQ      70138              ; Si égale alors GoTo ➔ #Avance_Recule
70126    BGT      7013C              ; si N=0, (plus grand que), alors GoTo #GoTo_Position
70128    MOVE.W   70070(PC),D6       ; D6=70070(PC) = Adresse dans la table
7012C    SUB.W    D0,D6              ; D6=D6-D0
7012E    SUBQ.W   #1,D6              ; D6=D6-1 en word
70130    BSR      700F6              ; ➔ GoSub ➔ #Move Exter.
70132    DBF      D6,70130           ; ← D6=D6-1, tant que D6 est différent de -1 on boucle
70136    RTS                         ; E.T Retour maison
```

#Avance_Recule

```
70138    BSR      700CC              ; GoSub ➔ #Move Inter.
7013A    BRA      700F6              ; GoTo ➔ #Move Exter.     (Ceci explique les bruits de saccades lors des chargements)
```

#GoTo_Position

```
7013C    SUB.W    70070(PC),D0       ; D0 'word' = Word Du tableau en 70070(PC)-D0
70140    SUBQ.W   #1,D0              ; D0=D0-1
70142    BSR      700CC              ; ➔ GoSub ➔ #Move Inter.
70144    DBF      D0,70142           ; ← D0=D0-1, tant que D0 est différent de -1, on boucle
70148    RTS                         ; E.T Retour maison
```

#Retour_T00

```
7014A    MOVE.B   BFE001,D           ; D0=CIA PRA
70150    BTST     #4,D0              ; On teste le bit4, TK0, Tête sur T00 ?
70154    BEQ      7015E              ; Oui ? GoTo ➔ #Update_table
70156    BSR      700F6              ; Non, GoSub ➔ #Move Exter.
70158    BSR      70170              ; GoSub #WAIT
7015C    BRA      7014A              ; GoTo ➔ #Retour_T00                    // on Boucle sur Retour_T00
```

#Update_table

```
7015E    LEA      70070(PC),A3       ; A3=position dans la table
70162    CLR.W    (A3)               ; Efface le Word en (A3)
70164    BTST     #5,BFE001          ; ➔ CIA-A / PRA, On teste le bit5, lecteur prêt ?
7016C    BNE      70164              ; ← tant que le lecteur n'est pas prêt, on boucle
7016E    RTS                         ; E.T Retour Maison
```

#WAIT

```
70170    MOVE.W   D7,-(A7)           ; Sauvegarde D7 dans la pile
70172    MOVE.W   #1388,D7           ; ➔ D7=1388
70176    DBF      D7,70176           ; ← D7=D7-1, on boucle tant que D7 est différent de -1
7017A    MOVE.W   (A7)+,D7           ; Restaure D7 de la pile
7017C    RTS                         ; E.T Retour Maison
```

#DF0__SIDE_DOWN__MOTOR_OFF__DIR_EXT

```
7017E    MOVE.B   #FD,BFD100        ; Conf CIA-B / PRB
                                    ; MOTOR OFF, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 UNSELECT, SIDE DOWN, DIR EXT., STEP TRACK=1
                                    ;
70186    MOVE.W   #100,D0           ; ➔ D0=100
7018A    DBF      D0,7018A          ; ← D0=D0-1, on boucle tant que D0 est différent de -1 (c'est juste une pause)
7018E    MOVE.B   #F5,BFD100        ; Conf CIA-B / PRB
                                    ; MOTOR OFF, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 SELECT, SIDE DOWN, DIR EXT., STEP TRACK=1
                         ;
70196    MOVE.W   #B000,D0          ; D0=B000
7019A    DBF      D0,7019A          ; ←→ D0=D0-1, on boucle tant que D0 est différent de -1 (c'est encore une pause)
7019E    LEA      70072(PC),A3      ; A3=position dans la table
701A2    CLR.W    (A3)              ; Efface le Word en (A3)
701A4    RTS                        ; E.T Retour maison
```

#DF0__SIDE_DOWN__MOTOR_ON__DIR_EXT

```
701A6    MOVE.B   #7D,BFD100        ; conf CIA-B / PRB
                                    ; MOTOR ON, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 UNSELECT, SIDE DOWN, DIR EXT., STEP TRACK=1
701AE    NOP                        ;
701B0    NOP                        ;
701B2    MOVE.B   #75,BFD100        ; conf CIA-B / PRB
                                    ; MOTOR ON, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 SELECT, SIDE DOWN, DIR EXT., STEP TRACK=1
701BA    MOVE.W   #B000,D7          ; D7=B000
701BE    DBF      D7,701BE          ; ←→ D7=D7-1, on boucle tant que D7 est différent de -1 (c'est encore une pause)
701C2    RTS                        ; E.T Retour maison
```

#DF0__SIDE_UP__MOTOR_ON__DIR_EXT

```
701C4    MOVE.B   #79,BFD100        ; conf CIA-B / PRB
                                    ; MOTOR ON, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 UNSELECT, SIDE UP, DIR EXT., STEP TRACK=1
701CC    NOP                        ;
701CE    NOP                        ;
701D0    MOVE.B   #71,BFD100        ; conf CIA-B / PRB
                                    ; MOTOR ON, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 SELECT, SIDE UP, DIR EXT., STEP TRACK=1
701D8    MOVE.W   #B000,D7          ; D7= B000
701DC    DBF      D7,701DC          ; ←→ D7=D7-1, on boucle tant que D7 est différent de -1 (c'est encore une pause)
701E0    RTS                        ; E.T Retour maison
```

#Trackloader_Start **Avec** **A0**=Adr_dest_mémoire **A1**=Pointeur tableau(Adress_Start_raw)

```
701E2    MOVE.L   A0,-(A7)          ; Sauvegarde de A0 dans la pile
701E4    BSR      701EC             ; GoSub ➔ #Lecture_Table_and_Start_Trackload_Or_Not
701E8    MOVEA.L  (A7)+,A0          ; Restaure A0 de la pile
701EA    RTS                        ; E.T Retour maison
```

#Lecture_Table_and_Start_Trackload_Or_Not

```
701EC    LEA      70072(PC), A3     ; A3=70072       (Position dans la table)          // Marqueur prêt à Trackloader ou pas
701F0    TST.W    (A3)              ; Test contenu de A3 avec zero
701F2    BNE      701F8             ; Si est différent de zero alors GoTo ➔ #Side_Select
701F4    BSR      70078             ; sinon, GoSub ➔ #Trackloader_Init

#Side_Select
701F8    MOVE.L   (A1)+,D0          ; Copie du LongWord contenu en A1 dans D0 puis A1=A1+4
701FA    CMP.L    #84468,D0         ; D0-84468 et change les Flag en conséquence.
                                    ; Test pour connaitre la Side à utiliser.

                                    ; Selon résultat, on ira sur la routine Face_UP ou Face_Down
                                    ; donc D0, alias A1, à savoir $70060 indique une position de départ sur le disk
                                    ; ak : Adresse_Start_raw

70200    BGE      70206             ; Si plus grand que, alors
                                    ; GoTo ➔ #Recup_Info_pour_Trackloader_1 et #DF0__SIDE_UP__MOTOR_ON__DIR_EXT
                                    ;
70202    BSR      701A6             ; Sinon GoSub ➔ #DF0__SIDE_DOWN__MOTOR_ON__DIR_EXT
70204    BRA      70208             ; Et ensuite, GoTo ➔ #Recup_Info_pour_Trackloader_2
```

#Recup_Info_pour_Trackloader_1

```
70206    BSR      701C4            ; GoSub ➔ #DF0__SIDE_UP__MOTOR_ON__DIR_EXT

#Recup_Info_pour_Trackloader_2
70208    MOVE.L   D0,D3            ; A cette étape D0=(LongWord de $70060) donc valeur du tableau=$48810
                                   ; à savoir Adress_Start_raw
                                   ; D3=D0, donc D3=$48810
                         ;
7020A    DIVU.W   #189C,D0         ; D0=D0/$189C, cela valide deux choses :
                                   ; + La taille d'une Track custom est bien $189C
                                   ; + Que D0 contient une adresse physique sur le disque.
                                   ; D0 indique maintenant Start_Track
                                   ;
7020E    CMP.W    #56,D0           ; Compare D0 avec la valeur $56
70212    BLT      7021E            ; Si plus petit que, alors GoTo $7021E
70214    SUBI.L   #84468,D3        ; sinon, on D3=D3-$84468
7021A    SUBI.W   #56,D0           ; D0=D0-$56
                                   ;
7021E    MOVEQ    #0,D2            ; D2=00
70220    MOVE.W   D0,D2            ; D2=D0
70222    BSR      70120            ; GoSub ➔ #Position Atteinte?
                                   ;
70226    MULU.W   #189C,D2         ; A ce moment, D2 = $2F = !47 = position atteinte=position de la tête=Track en cours
                                   ; On le multiplie par la taille d'une Track, il devient la taille en Position_raw_atteinte
                                   ; Position où l'on est, en l'occurrence : $484A4
                                   ;
7022A    SUB.L    D2,D3            ; à cette étape D3=(LongWord de $70060) donc valeur du tableau=$48810
                                   ; ou alors si D0 Start_Track > $56, on soustrait la position_raw_atteinte de D3
                                   ; En fait, selon l'Adresse_Start_raw dans le tableau, on va utiliser une side précise.
                                   ;
                                   ; Donc cette opération effectuée sur D3 permet de repositionner
                                   ; l'Adresse_Start_raw tout en changeant de face.
                                   ; Exemple, D3=$48810(valeur tableau en $70060)-$484A4(position atteinte)=$36C
                                   ;
7022C    MOVE.L   (A1),D4          ; Copie le longWord contenu à l'adresse A1 dans D4
                                   ; A1=$70064=(Length_To_Read) = $24284 donc D4=$24284
                                   ;
7022E    LSR.L    #2,D4            ; Décale de 2 Bits vers la droite le LongWord D4 ce qui revient à diviser D4 par 4,
                                   ; Dans notre cas, nous donnes D4=90A1
                                   ;
70230    SUBQ.L   #1,D4            ; D4=D4-1, On lui enlève 1 : D4=$90A0 (cela permet de tomber sur un nombre pair)
                                   ;
70232    BSR      7023A            ; ➔      GoSub ➔ #Trackloader_Base
70234    BSR      700CC            ;        GoSub ➔ #Move Inter.
70238    BRA      70232            ; ➔      GoTo ➔ #Trackloader_Base (Indirect)
```

#Trackloader_Base

```
7023A    MOVEQ    #F,D2            ; D2=F
7023C    MOVE.L   #18B8,D0         ; ➔ D0=$18B8
70242    MOVEA.L  70074(PC),A6     ;    A6=Adresse dans la table=0006A000
70246    MOVE.W   #2,DFF09C        ;    Conf INTREQ, Disk Block Finished Interupt
7024E    MOVE.L   A6,DFF020        ;    Conf DSKPTH  (conf qui provient de la table donc), Donc DSKPTH=$6A000
70254    MOVE.W   #8210,DFF096     ;    Conf DMACON, DSKEN enable, ALL DMA enable
7025C    MOVE.W   #4489,DFF07E     ;    Conf DSKSYNC = $4489 (standard AmigaDos)
70264    MOVE.W   #7F00,DFF09E     ;    Conf ADKCON spécifique
7026C    MOVE.W   #B500,DFF09E     ;    Encore une fois pour démarrer le transfert
                                   ;    FAST=2us, WORDSYNC=active, MFM precomp, recomp=140ns
                                   ;
70274    MOVE.W   #4000,DFF024     ;     Conf DSKLEN, Write enable (ram or disk)
                                   ;
                                   ;     Tst CIA Ready
7027C    MOVE.B   BFDD00,D7        ;     D7=BFDD00
70282    MOVE.B   BFDD00,D7        ;     D7=icr register de CIA-B
70288    BTST     #4,D7            ;     ➔ Test du bit4 (FLAG) de icr
7028C    BEQ      70282            ;     ➔ Interruption générée ? On boucle
7028E    ADDI.W   #8001,D0         ;     ADD signé sur D0 (qui est a $18B8 voir quelques lignes au-dessus)
                                   ;     Avec $8001, ce qui nous donnes : $98B9 et flag C=0
                                   ;
70292    MOVE.W   D0,DFF024        ;     CONF DSKLEN, Disk DMA Enable, Length of DMA data=$18B9
70298    MOVE.W   D0,DFF024        ;     Encore une fois pour déclencher la lecture
7029E    MOVE.W   DFF01E,D0        ;     ➔ D0=INTREQR
702A4    BTST     #1,D0            ;        Test du Bit1 de INTREQR, Level 1 Disk Block Finished Interupt
702A8    BEQ      7029E            ;     ➔ Test ?, On boucle
702AA    MOVE.L   (A6)+,D0         ;     Valeur de DSKPTH dans D0, puis A6=A6+4 (donc A6=DFF024)
702AC    MOVE.L   (A6)+,D7         ;     Valeur de DSKLEN dans D7, puis A6=A6+4 (donc A6=DFF028)
702AE    ADD.L    D0,D0            ;     D0=D0+D0
702B0    ANDI.L   #AAAAAAAA,D0     ;     Post_Traitement MFM bit impair dans D0
702B6    ANDI.L   #55555555,D7     ;     Post_Traitement MFM bit pair dans D7
702BC    OR.L     D7,D0            ;     Traitement MFM
702BE    CMP.L    #42535432,D0     ;     Compare D0 avec 42 53 54 32, En Ascii = BST2.
                                   ;     Si la signature n'est pas trouvée, on re-check
                                   ;     et finalement, si tjs pas bon, partira sur la routine d'écran rouge plus bas.
                                   ;
702C4    BEQ      702D4            ;     Oui ? Signature trouvé GoTo ➔ #Traitement_MFM_BASE
                                   ;     On commence le traitement réel des données et le Trackload
702C6    DBF      D2,7023C         ; ➔   D2=D2-1, tant que D2 est différent de -1, on boucle en 7023C
                                   ;     (c'est reparti pour un tour)
                                   ;
702CA    MOVE.W   #F00,DFF180      ; ➔ RRRR, mauvaise pioche, fond d'écran en rouge.
702D2    BRA      702CA            ; ➔ DeadLoop sur fond d'écran en rouge
```

#Traitement_MFM_BASE

```
702D4    MOVEQ    #0,D2            ; D2=00
702D6    MOVE.W   #626,D2          ; D2=$626
                                   ;
702DA    TST.W    D3               ; Test contenu de D3 avec zéro                    // D3=Delta calculé préalablement.
702DC    BEQ      702E8            ; Si égal alors GoTo ➔ #Check_Traitement_MFM
702DE    ADD.W    D3,D3            ; Sinon D3=D3+D3                                  // on fait x2 sur D3
                                   ; Rappel, D3=((Valeur tableau 70060)-(valeur position atteinte))
                                   ;
702E0    ADDA.L   D3,A6            ; A6=A6+D3                                        // A6=DSKPTH+décodage en cour
702E2    LSR.W    #3,D3            ; Décale de 3 Bits vers la gauche le LongWord D3  // Revient à diviser par 8 D3
                                   ; Donc delta entre position atteinte et (valeur en 70064)/8
                                   ;
702E4    SUB.W    D3,D2            ; D2=D2-D3
                                   ;                                    // D2(position_raw actuel)-Delta calculé ci-dessus
702E6    MOVEQ    #0,D3            ; D3=0                                            // On remet D3 à Zero
                                   ;
#Check_Traitement_MFM
702E8    CMP.L    D2,D4            ; Compare D4-D2
702EA    BGT      702F2            ; si résultat plus grand que GoTo ➔ #Traitement MFM bit pair
702EC    MOVE.W   D4,D2            ; D2=D4
702EE    ADDQ.W   #4,A7            ; A7=A7+4
702F0    BRA      702F6            ; GoTo ➔ #Start_Decodage
```

#Traitement MFM bit pair

```
702F2    SUB.L    D2,D4            ; D4=D4-D2
702F4    SUBQ.L   #1,D4            ; D4=D4-1
#Start_Decodage
702F6    MOVE.L   #55555555,D5     ; D5=55555555, masque MFM bit pair
702FC    MOVE.L   (A6)+,D0         ; ➔  Copie contenu de A6 dans D0 puis A6=A6+4
702FE    MOVE.L   (A6)+,D7         ;     Copie contenu de A6 dans D7 puis A6=A6+4
70300    AND.L    D5,D0            ;     Traitement MFM
70302    AND.L    D5,D7            ;     Traitement MFM
70304    ADD.L    D0,D0            ;     Traitement MFM
70306    OR.L     D7,D0            ;     Traitement MFM
70308    MOVE.L   D0,(A0)+         ;     Copie de D0 à l'adresse pointée par A0 puis A0=A0+4
7030A    DBF      D2,702FC         ; ➔ D2=D2-1 et tant que D2 est différente de -1, on boucle
7030E    RTS                       ; E.T retour maison
```

#Decrypt_decomp_Init

```
70310    MOVEQ    #0,D7            ; D7=0
70312    MOVEA.L  A0,A2            ; A2=A0
70314    MOVE.L   (A0),D0          ; Copie du LongWord pointé par A0 dans D0
70316    BTST     #0,D0            ; Test de D0 avec 0
7031A    BEQ      70324            ; Si égale à Zero alors GoTo ➔ #Decrypt_Decomp_Base
7031C    MOVEA.L  A1,A3            ; A3=A1
7031E    NOT.W    D7               ; ...
70320    ANDI.W   #FFFE,D0         ; ... Toute une phase de traitement des datas

#Decrypt_Decomp_Base
70324    ADDA.L   D0,A0            ; ...
70326    MOVE.L   -(A0),(A2)       ; ...
70328    MOVEA.L  -(A0),A2         ; ...
7032A    ADDA.L   A1,A2            ; ...
7032C    MOVE.L   -(A0),D5         ; ...
7032E    MOVE.L   -(A0),D0         ; ...
70330    MOVEQ    #10,D6           ; ...
70332    EOR.L    D0,D5            ; ...

#Decrypt_Start
70334    LSR.L    #1,D0            ; Décalage de 1 bit vers la droite de D0
70336    BNE      7033A            ; GoTo ➔ #Decrypt_phase1 (Indirect)
70338    BSR      703AE            ; GoSub ➔ #Decrypt_D5
7033A    BCS      7036E            ; GoTo ➔ #Decrypt_phase1
7033C    MOVEQ    #8,D1            ; D1=08
7033E    MOVEQ    #1,D3            ; D3=01
70340    LSR.L    #1,D0            ; Décalage de 1 bit vers la droite de D0
70342    BNE      70346            ; GoTo ➔ #Decrypt_D0_D2 (Indirect)
70344    BSR      703AE            ; GoSub ➔ #Decrypt_D5
70346    BCS      70390            ; GoTo ➔ #Decrypt_D0_D2 (Indirect)
70348    MOVEQ    #3,D1            ; D1=03
7034A    MOVEQ    #0,D4            ; D4=00
7034C    BSR      703B8            ; GoSub ➔ #Decrypt_D0_D2
7034E    MOVE.W   D2,D3            ; D3=D2 en word
70350    ADD.W    D4,D3            ; D3=D3+D4

#Traitement_D0_D1
70352    MOVEQ    #7,D1            ; ➔
70354    LSR.L    #1,D0            ;       ➔ Décalage vers la droite d'1 bit de D0
70356    BNE      7035A            ;          GoTo ➔ #Decrypt_D2
70358    BSR      703AE            ;          GoSub ➔ #Decrypt_D5
7035A    ROXL.L   #1,D2            ;          Rotation étendue vers la gauche de 1 bit de D2
7035C    DBF      D1,70354         ;       ➔
70360    MOVE.B   D2,-(A2)         ;
70362    DBF      D3,70352         ; ➔ GoTo ➔ #Traitement_D0_D1
70366    BRA      7039C            ; GoTo ➔ #Check_A1_A2
```

#D1_D4_8Byte

```
70368    MOVEQ    #8,D1              ; D1=08
7036A    MOVEQ    #8,D4              ; D4=08
7036C    BRA      7034C              ; GoTo ➔ #Decrypt_D0_D2
```

#Decrypt_phase1

```
7036E    MOVEQ    #2,D1              ; D1=2
70370    BSR      703B8              ; GoSub ➔ #Decrypt_D0_D2
70372    CMP.B    #2,D2              ; Compare le Byte 02 avec D2
70376    BLT      70388              ; GoTo ➔ #Decrypt_phase2
70378    CMP.B    #3,D2              ; Compare le Byte 03 avec D2
7037C    BEQ      70368              ; GoTo ➔ #D1_D4_8Byte
7037E    MOVEQ    #8,D1              ; D1=8
70380    BSR      703B8              ; GoSub ➔ #Decrypt_D0_D2
70382    MOVE.W   D2,D3              ; D3=D2 en word
70384    MOVEQ    #C,D1              ; D1=$0C
70386    BRA      70390              ; GoTo ➔ #Decrypt_D0_D2 (Indirect)
```

#Decrypt_phase2

```
70388    MOVEQ    #9,D1              ; ...
7038A    ADD.W    D2,D1              ; ...
7038C    ADDQ.W   #2,D2              ; ...
7038E    MOVE.W   D2,D3              ; ...
70390    BSR      703B8              ; GoSub ➔ #Decrypt_D0_D2

70392    SUBQ.L   #1,A2              ; ➔
70394    MOVE.B   0(A2,D2.W),(A2)    ;   Copie le résultat dans (A2)
70398    DBF      D3,70392           ; ➔

#Check_A1_A2
7039C    CMPA.L   A2,A1              ;
7039E    BLT      70334              ; GoTo ➔ #Decrypt_Start
703A0    TST.L    D5                 ;
703A2    BNE      703D0              ; GoTo ➔ #D0=-1
703A4    TST.W    D7                 ;
703A6    BEQ      703AA              ; GoTo ➔ #RAZ_D0
703A8    BRA      703D4              ; GoTo ➔ #Move_Decrypt_base
```

#RAZ_D0

```
703AA MOVEQ      #0, D0             ; RAZ de D0
703AC RTS                           ; E.T Retour maison
```

#Decrypt_D5

```
703AE    MOVE.L   -(A0),D0           ;
703B0    EOR.L    D0,D5              ;
703B2    MOVE.W   D6,CCR             ;
703B4    ROXR.L   #1,D0              ;
703B6    RTS                        ; E.T Retour maison
```

#Decrypt_D0_D2

```
703B8    SUBQ.W   #1,D1              ; D1=D1-1
703BA    CLR.W    D2                 ; D2=0 en word
703BC    LSR.L    #1, D0             ; ➔ Décalage d'1 bit vers la droite de D0
703BE    BNE      703C8              ; GoTo ➔ #Decrypt_D0_D2b
703C0    MOVE.L   -(A0),D0           ;
703C2    EOR.L    D0,D5              ;
703C4    MOVE.W   D6,CCR             ;
703C6    ROXR.L   #1,D0              ;

#Decrypt_D0_D2b
703C8    ROXL.L   #1,D2              ;
703CA    DBF      D1,703BC           ; ➔ D1=D1-1, tant que D1 est diffèrent de -1 on boucle
703CE    RTS                        ; E.T Retour maison
```

#D0=-1

```
703D0    MOVEQ    #FFFFFFFF,D0       ; D0=-1
703D2    RTS                         ; E.T Retour maison
```

#Move_Decrypt_base

```
703D4    MOVEA.L  A3,A0             ;
703D6    MOVEA.L  A0,A1             ;
703D8    MOVEA.L  A0,A2             ;
703DA    MOVE.L   (A0),D0           ;
703DC    LSR.L    #8,D0             ;
703DE    ADDA.L   D0,A0             ;
703E0    MOVE.B   -(A0),(A3)+       ;
703E2    MOVE.B   -(A0),(A3)+       ;
703E4    MOVE.B   -(A0),(A3)        ;
703E6    MOVEQ    #0,D1             ;
703E8    MOVE.B   -(A0),D1          ;
703EA    LSL.W    #8,D1             ;
703EC    MOVE.B   -(A0),D1          ;
703EE    LSL.L    #8,D1             ;
703F0    MOVE.B   -(A0),D1          ;
703F2    ADDA.L   D1,A1             ;
703F4    MOVE.B   -(A0),D4          ;
703F6    MOVE.B   -(A0),D5          ;
703F8    MOVE.B   -(A0),D6          ;
703FA    MOVE.B   -(A0),D7          ;
703FC    MOVEQ    #0,D2             ;
703FE    MOVEQ    #FFFFFFFF, D3     ; D3=-1

#Move_Decrypt_Start
70400    CMPA.L   A2,A1             ; Compare A1 avec A2 (test si décrypt fini ou pas)
70402    BLE      7043E             ; GoTo ➔ #Decrypt_Done, A2 ou A1 contient à ce moment l'adr. Début des données decomp/dec
rypt
70404    MOVE.B   -(A0),D0          ;
70406    CMP.B    D0,D4             ;
70408    BEQ      70430             ; GoTo ➔ #COPIE_Decrypt
7040A    CMP.B    D0,D5             ;
7040C    BEQ      7041A             ; GoTo ➔ #Move_Decrypt_D0
7040E    CMP.B    D0,D6             ;
70410    BEQ      70424             ; GoTo ➔ #Move_Decrypt_D2
70412    CMP.B    D0,D7             ;
70414    BEQ      7042A             ; GoTo ➔ #Move_Decrypt_D3
70416    MOVE.B   D0,-(A1)          ;
70418    BRA      70400             ; GoTo ➔ #Move_Decrypt_Start
```

#Move_Decrypt_D0

```
7041A    MOVE.B   -(A0),D0          ;
7041C    MOVE.B   D0,-(A1)          ;
7041E    MOVE.B   D0,-(A1)          ;
70420    MOVE.B   D0,-(A1)          ;
70422    BRA.B    70400             ; GoTo ➔ #Move_Decrypt_Start
```

#Move_Decrypt_D2

```
70424    MOVE.B   D2,-(A1)          ;
70426    MOVE.B   D2,-(A1)          ;
70428    BRA      70400             ; GoTo ➔ #Move_Decrypt_Start
```

#Move_Decrypt_D3

```
7042A    MOVE.B   D3,-(A1)          ;
7042C    MOVE.B   D3,-(A1)          ;
7042E    BRA      70400             ; GoTo ➔ #Move_Decrypt_Start
```

#COPIE_Decrypt

```
70430    MOVEQ    #0,D0             ; RAZ de D0
70432    MOVE.B   -(A0),D1          ; Mise à jour de D1, nos données à copier
70434    MOVE.B   -(A0),D0          ; Mise à jour de D0, notre compteur

70436    MOVE.B   D1,-(A1)          ; ➔ Copie D1 dans A1 puis A1=A1-1
70438    DBF      D0,70436          ; ➔ D0=D0-1, tant que D0 est différent de -1, on boucle
7043C    BRA      70400             ; GoTo ➔ #Move_Decrypt_Start
```

#Decrypt_Done

```
7043E    MOVEQ    #0,D0             ; D0=0
70440    RTS                        ; E.T Retour Maison
```

Conclusion :

Nous avons ici un **TrackLoader** qui :

- Fonctionne par 'Side', Upper ou Lower (valeur de limite de sélection=*$84468*)
- Fonctionne avec une table qui est mise à jour avant chaque TrackLoad. (du moins, pour l'instant)
- Une taille de Track Custom de *$189C*
- 2 tracks à éviter sur le 1er Disque (Track 00=BootSecteur, Track 01=SignatureDisk)
- 1 track à éviter sur le second Disque (Track 00=SignatureDisk)
- Une configuration de **DSKPTH** effectuée spécifique à chaque TrackLoad via une table.
- Une adresse de DépartDisk spécifique à chaque TrackLoad via une table.
- Une Longueur de donnée à Trackloader spécifique à chaque TrackLoad via une table.
- Une adr. de destination mémoire en **A0** avant l'appel à la routine de TrackLoad.
- Des données compressées/cryptées sur le disque original.

La table en question, présente en *$70060* et mise à jour méthodiquement.

#Table_#01

```
70060    00 04 88 10      ← Adresse_Start_raw, (Minimum $189C, taille d'une Track custom)
                             Obligation de commencer en cylindre 1 minimum (Track en 0 Spécifique)
70064    00 02 42 84      ← Length_To_Read
70068    00 06 CA 94      ← Mis en $C2, aucune idée pour l'instant.
7006C    00 00 9B 78      ← Mis en $C6, aucune idée pour l'instant.
70070    00 00            ← Pointeur Track en cours. (piste car on fonctionne par Side)
70072    00 00            ← Marqueur pour Trackloader ou pas.
70074    00 06 A0 00      ← Conf DSKPTH
```

La procédure vue en *$70000*

1	**A0** pour définir l'adr. de destination pour le trackload à venir	7003A	LEA	45610,A0
2	**A1** pour définir l'adr. de la table pour récupérer les infos **DSKPTH** et **Longueur à lire**	70040	LEA	70060(PC),A1
3	Exécution du TrackLoad	70044	BSR.W	701E2
	Donnée Trackloadée et Disponible en mémoire à l'adresse A0			
4	**A1** pour définir l'adr. de destination pour la décompression	70048	LEA	2B0.S,A1
5	Exécution de la décompression	7004C	BSR.W	70310
	Donnée Décompressée et Disponible en mémoire à l'adresse A1			
6	Maj. De **A0** pour le prochain TrackLoad à venir.	70050	LEA	70068(PC),A0
7	Sans oublier de mettre à jour aussi les marqueurs en mémoire $C2 et $C6 Données qui seront récupérées avant le prochain TrackLoad	70054 70058	MOVE.L MOVE.L	(A0)+,C2.S (A0)+,C6.S
8	Exécution du code à l'adresse de décompression.	7005C	JMP	2B0.S

Lancement de l'Animation Psygnosis

Part 7 Test TrackLoader #1 et Rip de l'animation Psygnosis

Rebooter l'Amiga sur notre disquette de backup réalisée précédemment sous **X-Copy**.
Très vite on arrive sur notre fond d'écran en vert, **échanger la disquette par celle de l'original Disk1** et **Entrer dans l'AR**.

On remet le code d'origine, **Taper :** `A 70000`

```
A 70000
$070000  MOVE.W #2700,SR
$070004  MOVE.W #7FFF,DFF096
$07000C  <RETURN>
```

Petit rappel : Normalement le prochain **TrackLoad** s'effectuera en :

Adr. Source disk : *$48810*
Taille : *$24284*
DSKPTH : *$6A000*
Dest. Mémoire : *$45610*

On pause un *BreakPoint* juste avant la phase de décompression/décrypt, **Taper :** `BS 7004C`
On remplit la Zone mémoire de destination avec une petite marge : ($45610-$20) → ($45610+$24284+$20) ***$455F0 → $698B4***
Taper : `O "PaTtErN", 455F0 698B4`

On peut aussi vérifier si vous le voulez comme le montre l'image ci-dessous.

```
o "PaTtErN", 455F0 698B4
Ready.

m 455F0
:0455F0 50 61 54 74 45 72 4E 50 61 54 74 45 72 4E 50 61 PaTtErNPaTtErNPa
:045600 54 74 45 72 4E 50 61 54 74 45 72 4E 50 61 54 74 TtErNPaTtErNPaTt
:045610 45 72 4E 50 61 54 74 45 72 4E 50 61 54 74 45 72 ErNPaTtErNPaTtEr
:045620 4E 50 61 54 74 45 72 4E 50 61 54 74 45 72 4E 50 NPaTtErNPaTtErNP

m 698B4-40
:069874 61 54 74 45 72 4E 50 61 54 74 45 72 4E 50 61 54 aTtErNPaTtErNPaT
:069884 74 45 72 4E 50 61 54 74 45 72 4E 50 61 54 74 45 tErNPaTtErNPaTtE
:069894 72 4E 50 61 54 74 45 72 4E 50 61 54 74 45 72 4E rNPaTtErNPaTtErN
:0698A4 50 61 54 74 45 72 4E 50 61 54 74 45 72 4E 50 61 PaTtErNPaTtErNPa
```

Et on retourne au code avec la commande `X`
Après un trackload, notre *BreakPoint* est atteint, on entre automatiquement dans l'AR.

On vérifie de nouveau la zone mémoire, **taper :** `M 455F0` puis `M 698B4-40`

```
m 455F0
:0455F0 50 61 54 74 45 72 4E 50 61 54 74 45 72 4E 50 61 PaTtErNPaTtErNPa
:045600 54 74 45 72 4E 50 61 54 74 45 72 4E 50 61 54 74 TtErNPaTtErNPaTt
:045610 00 02 41 05 35 D8 00 35 96 00 F8 13 20 0C 62 F7 ..A.5..5.... .b.
:045620 29 F3 5C 02 49 D2 D0 86 CA 06 10 20 83 80 0C 64 ).\.I...... ...d

m 698B4-40
:069874 00 04 4D 44 7F FF 00 00 00 00 00 4E 66 41 C0 ..MD.......NfA.
:069884 41 C1 00 04 4D 44 42 20 00 00 00 00 00 7F FF A...MDB ......
:069894 72 4E 50 61 54 74 45 72 4E 50 61 54 74 45 72 4E rNPaTtErNPaTtErN
:0698A4 50 61 54 74 45 72 4E 50 61 54 74 45 72 4E 50 61 PaTtErNPaTtErNPa
```

Des données ont bien été chargées de ***$45610*** jusqu'à ***$69894*** comme prévu. Cela valide notre analyse du fonctionnement du **TrackLoader.**

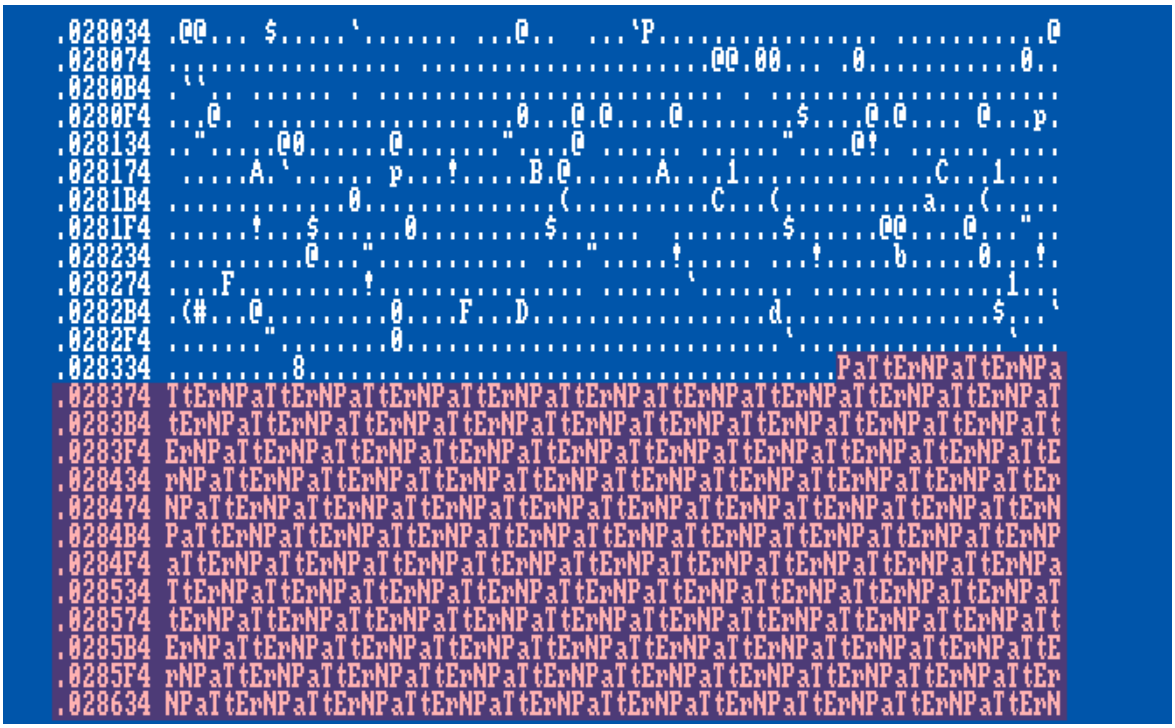
Il nous reste la partie décompression/décrypt :
On remplit la Zone mémoire de destination avec une petite marge : ($2B0-$20) → ($45610) ***$290 → $45610***
Taper : `O "PaTtErN", 290 45610`

On pause un *BreakPoint* juste après la phase de décompression/décrypt, **taper :** `BS 70050` **e**t on retour au code avec la commande `X`
Après quelques secondes, notre **BreakPoint** est atteint, la phase de décompression/décrypt.

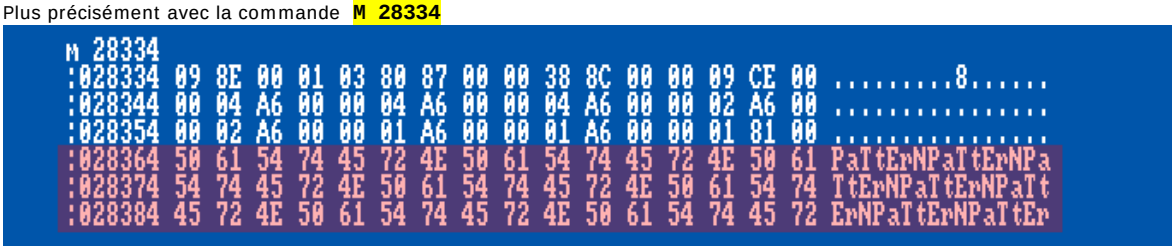
On vérifie la zone mémoire de décompression, **taper :** `M 455F0` puis `M 698B4-40`

```
m 290
:000290 50 61 54 74 45 72 4E 50 61 54 74 45 72 4E 50 61 PaTtErNPaTtErNPa
:0002A0 54 74 45 72 4E 50 61 54 74 45 72 4E 50 61 54 74 TtErNPaTtErNPaTt
:0002B0 00 00 00 00 00 00 00 00 00 00 00 26 61 00 00 F4 ...........&a...
:0002C0 4E F9 00 00 10 72 41 F9 00 04 00 00 43 F8 02 56 N....rA.....C..V

m 45610-40
:0455D0 72 4E 50 61 54 74 45 72 4E 50 61 54 74 45 72 4E rNPaTtErNPaTtErN
:0455E0 50 61 54 74 45 72 4E 50 61 54 74 45 72 4E 50 61 PaTtErNPaTtErNPa
:0455F0 54 74 45 72 4E 50 61 54 74 45 72 4E 50 61 54 74 TtErNPaTtErNPaTt
:045600 45 72 4E 50 61 54 74 45 72 4E 50 61 54 74 45 72 ErNPaTtErNPaTtEr
:045610 40 96 4C 03 35 D8 00 35 96 00 F8 13 20 0C 62 F7 @.L.5..5.... .b.
```

Des données sont maintenant présentes à partir de $2B0, mais elles ne vont pas jusqu'à $45610
$2B0+$24284=$24534
Taper : **N 290** et descendez dans le code jusqu'à apercevoir notre pattern '**PaTtErN**'



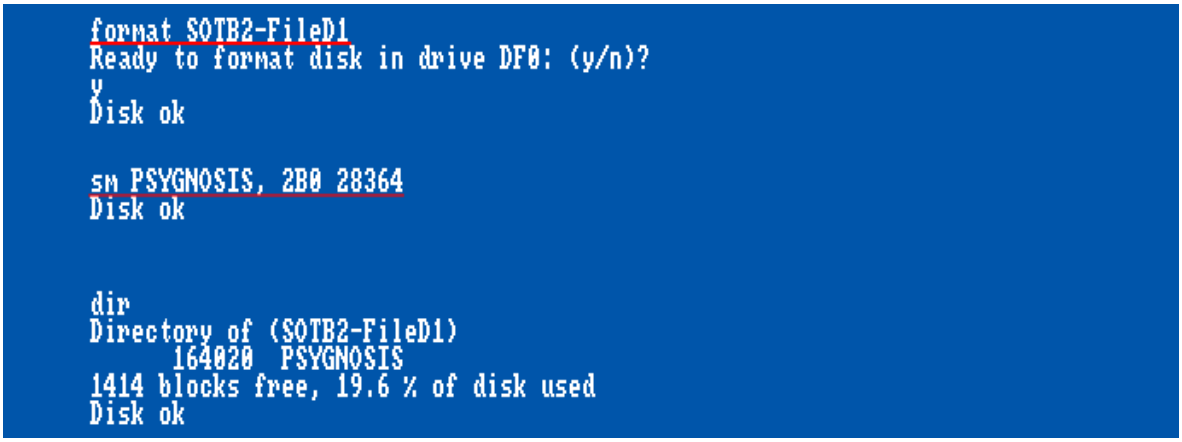
Plus précisément avec la commande **M 28334**



On a donc une fin des données Décompressées/décrypt en $28364 en partant de $2B0 (donc $28364-$2B0=$280B4)
Donc on part d'une taille de donnée **trackLoadées** de : $24284 pour arriver à une taille décompressée de $280B4
Soit $3E30 (15920 Octets, environ 15,5 Ko) de plus, (Ce qui nous donne un gain d'environ 10% du fichier) ce qui est vraiment pas mal.
On verra plus tard si on arrive à avoir mieux avec une autre routine de compression. Le but étant de gagner le plus de place possible.

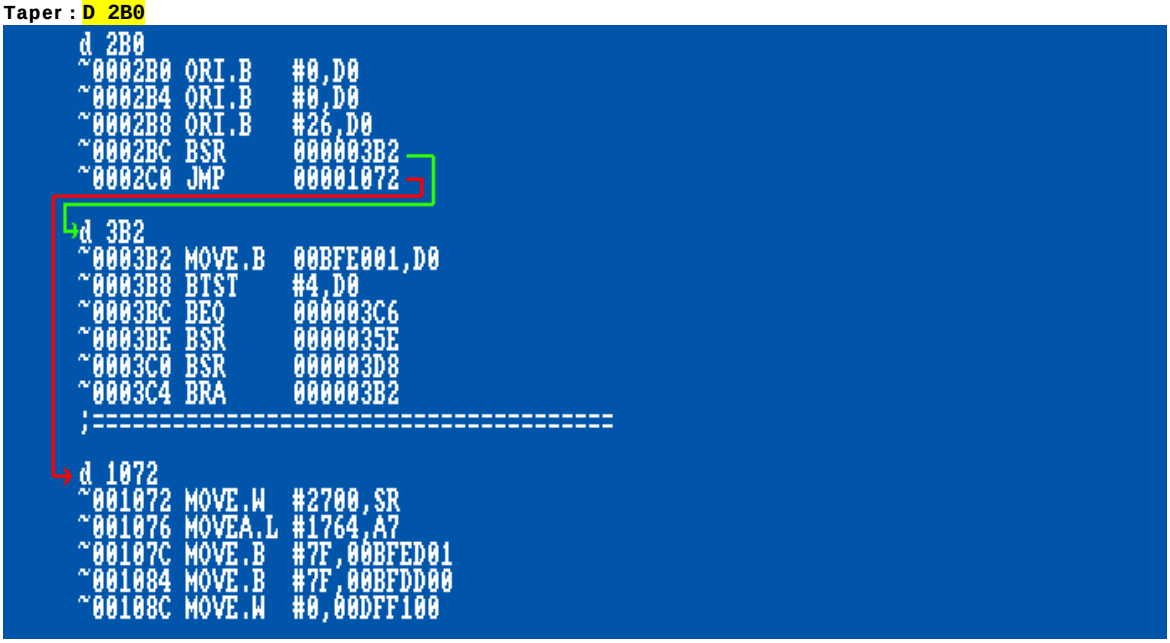
On en profite pour sauver ces données qui correspondent à la 1er animation **Psygnosis**
#FORMAT, Permet de formater une disquette au format AmigaDOS.
#SM, alias SaveMemory. Permet donc de sauver une zone mémoire vers un fichier.

Enlever la disquette originale du lecteur de disquette et insérez une disquette vierge.
Taper : **FORMAT SOTB2-FileD1** et on sauve le tout **SM PSYGNOSIS, 2B0 28364**



Part 8 Analyse du TrackLoader #2

Il nous reste plus qu'à jeter un coup d'œil au code en *$2b0*, et visiblement on a encore affaire à un **TrackLoader**
Taper : D 2B0



Plus en détail et voilà ce que l'on peut retenir.

#2B0

```
002B0    ORI.B    #0,D0
002B4    ORI.B    #0,D0
002B8    ORI.B    #26,D0                ; 26 ? Marqueur de quelque chose ?
002BC    BSR      3B2                   ; GoSub ➔ #Retour_T00
002C0    JMP      1072                  ; Goto ➔ #Base_Pre-Anim
```

#Base_Pre-Anim

```
1072     MOVE.W   #2700,SR              ; Conf. SR
1076     MOVEA.L  #1764,A7              ; A7=$1764
107C     MOVE.B   #7F,BFED01            ; Conf CIA-A icr
1084     MOVE.B   #7F,BFDD00            ; Conf CIA-B icr
108C     MOVE.W   #0,DFF100             ; Conf BPLCON0

1094     LEA      14E4,A0               ; A0=$14E4
109A     MOVE.W   #1F,D7                ; D7=1F
109E     MOVE.L   #0,(A0)+              ; ➔ Efface la zone (A0) puis A0=A0+4
10A4     DBF      D7,109E               ; ➔ D7=D7-1, tant que D7 est différent de -1, on boucle
                                       ; Soit $20 fois donc $20*4=$80 Bytes effacé depuis 14E4  (14E4 -> 15E4)

10A8     MOVE.L   #1142,64.S            ; Maj. D'une table.
10B0     MOVE.L   #1186,68.8            ;
10B8     MOVE.L   #11FC,6C.S            ;
10C0     MOVE.L   #1238,70.8            ;
10C8     MOVE.L   #12D2,74.S            ;
10D0     MOVE.L   #130A,78.8            ;
10D8     MOVE.L   #13E6,1764            ;

10E2     MOVE.B   #7F,BFED01            ; Conf CIA-A icr
10EA     MOVE.B   #0,BFEC01             ; Conf CIA-A sdr (connected to keyboard)
10F2     BCLR     #6,BFEE01             ; Conf CIA-A cra
10FA     MOVE.B   #98,BFED01            ; Conf CIA-A icr
1102     MOVE.B   BFED01,D0             ; D0=Conf CIA-A icr
1108     MOVE.L   #113E,DFF080          ; Conf COP1LCH    //Conf. De l'adresse de la Copperlist(1) = $113E
1112     MOVE.W   #C028,DFF09A          ; Conf POTINP
111A     MOVE.W   #8300,DFF096          ; Conf DMACON

1122     LEA      763C0,A0              ; A0=$763C0
1128     BSR      EFC                   ; GoSub ➔ #ERASE_A0
112C     LEA      6C780,A0              ; A0=$6C780
1132     BSR      EFC                   ; GoSub ➔ #ERASE_A0
1136     MOVE.W   #2000,SR              ; Conf. Status Register
113A     JMP      598.S                 ; GoTo ➔ #Base_Anim
```

#ERASE_A0

```
00EFC    MOVE.W   #270F,D7              ; D7=270F
00F00    MOVE.L   #0,(A0)+              ; ➔ Efface la zone (A0) puis A0=A0+4
00F06    DBF      D7,F00                ; ➔ D7=D7-1, tant que D7 est différent de -1, on boucle
00F0A    RTS                            ;
```

#Base_Anim

```
00598    MOVE.W   #4,D88                  ; copie 0004 à l'adresse D88
005A0    MOVE.W   #96,D86                 ; copie 0096 à l'adresse D86
005A8    MOVE.W   #FFFF,D8A               ; copie FFFF à l'adresse D8A
005B0    CLR.W    1DD2                    ; Nettoie le World en 1DD2
005B6    BSR      CE6                     ; GoSub $CE6
005BA    JSR      1D0A                    ; GoTo $1D0A
005C0    MOVE.W   #FFFF,ABE               ; copie FFFF à l'adresse $ABE
005C8    MOVE.W   #1,D0                   ; D0=0001
005CC    BSR      F44                     ; GoTo $F44
005D0    BTST     #7,BFE001               ; Bouton Fire appuyé ?
005D8    BNE      5E4                     ; - Si pas appuyé, alors GoTo ➔ #Start_Animation
005DA    MOVE.B   #1,176B                 ; - Si appuyé alors, copie 01 à l'adresse $176B
005E2    BRA      638                     ; puis, GoTo ➔ #Chargement Last_Load_Disk1 - InsertDisk2
```

#Start_Animation

```
005E4    LEA      16B14,A0                ; A0=16B14, Adresse_source_à_traiter
005EA    BSR      6B4                     ; GoSub ➔ #Decrypt/Decomp_&_execution

005EE    BTST     #7,BFE001               ; Bouton Fire appuyé ?
005F6    BEQ      638                     ; GoTo ➔ #Chargement Last_Load_Disk1 - InsertDisk2

#Chargement Animation Part #1             // Fin Animation Psyngosis, Début chargement Anim Part #1
005F8    MOVE.L   1DCA,578.S              ; Mise à jour du tableau #1, DSKPTH par rapport au contenu mémoire en $1DCA

00600    LEA      16B14,A0                ; A0=16B14,  Adr_dest_mémoire
00606    LEA      580.S,A1                ; A1=$580    Adresse du tableau : Adress_Start_raw
0060A    BSR      44A                     ; GoSub ➔ #Trackloader_Start

0060E    LEA      16B14,A0                ; A0=16B14, Adresse_source_à_traiter
00614    BSR      6B4                     ; GoSub ➔ #Decrypt/Decomp_&_execution

#Chargement Animation Part #2             // Fin Anim Part #1, Début chargement Anim Part #2
00618    MOVE.L   1DCA,578.S              ; Mise à jour du tableau #1, DSKPTH par rapport au contenu mémoire en $1DCA

00620    LEA      16B14,A0                ; A0=16B14,  Adr_dest_mémoire
00626    LEA      588.S,A1                ; A1=$588    Adresse du tableau : Adress_Start_raw
0062A    BSR      44A                     ; GoSub ➔ #Trackloader_Start

0062E    LEA      16B14,A0                ; A0=16B14, Adresse_source_à_traiter
00634    BSR      6B4                     ; GoSub ➔ #Decrypt/Decomp_&_execution

#Chargement Last_Load_Disk1 - InsertDisk2  // Fin Anim Part #2, Début chargement Last_Load
00638    LEA      256.S,A7                ; Change l'adresse de la pile en $256

0063C    MOVE.L   #50000,578.S            ; Fin des animations. Avant prochain, mise à jour de la table #02
00644    MOVE.L   C2.S,590.S              ; Copie du Word préalablement mis en C2 dans le tableau actuel
0064A    MOVE.L   C6.S,594.S              ; Copie du Word préalablement mis en C6 dans le tableau actuel

00650    LEA      40000,A0                ; A0=Destination mémoire pour le TrackLoader
00656    LEA      590(PC),A               ; A1=$590, adr. du tableau qui va contenir ce que l'on a préalablement mis en $C2
                                         ; valeur qui était en $70068 dans le précédent Trackloader

0065A    BSR      44A                     ; GoSub ➔ #Trackloader_Start
                                         ; A0=Adr_dest_mémoire              A1=Pointeur tableau(Adress_Start_raw)
                                           $590=(Pos.Disque=0006CA94        Size=00009B78)
                                           Donnée chargées : $40000 ➔ $49B78

#Recopie_2C6_vers_4F000                  // Fin dernier chargement, recopie code vers 4F000
0065E    LEA      2C6(PC),A0              ; A0=Adresse source pour boucle de copie=2C6
00662    LEA      4F000,A1                ; A1=Adresse dest. =4F000
00668    MOVE.W   #6,D0                   ; D0=6, compteur

0066C    MOVE.L   (A0)+,(A1)+             ; ➔ Boucle de copie de (A0) vers (A1)
0066E    DBF      D0,66C                  ; ➔ D0=D0-1, tant que D0 est différent de -1 on boucle
                                         ;   copie donc de 7 long word, soit 28 Bytes

00672    MOVE.W   #15E,D1                 ; D1=$15E
```

#Fin de la boucle de recopie, on attend la fin du compteur pour afficher le message 'INSERT DISK 2'

```
00676    CLR.B    176A            ; ➔ On efface le Byte en $176A
0067C    TST.B    176A            ;       ➔ On test le Byte en $176A
00682    BEQ      67C             ;       ➔ Si égale à zéro, on boucle, doit être un pointeur de fin de quelque chose
                                  En l'occurrence, timer de quelques secondes.

00684    BTST     #7,BFE001       ; CIA-A / PRA, On test le bit7, Bouton Fire1 appuyé ?
0068C    BEQ      692             ; Oui ?, On saute après cette boucle.
0068E    DBF      D1,676          ; ➔ Nop, pas appuyé, on boucle (on attend quelque seconde avant de bypasse ce test)

00692    TST.B    176B            ; Encore le test de $176B
                                  ; visiblement il se passe quelque chose en arrière-plan à cette adresse

00698    BNE      69E             ; Si ce n'est pas égale à Zero, on va en $69E
0069A    BSR      988             ; GoSub #$988                                    // Hors sujet du hack, aucun intérêt

0069E    MOVE.W   #7FFF,DFF09A    ; DFF09A=Conf. INTENA
006A6    MOVE.W   #7FFF,DFF096    ; DFF096=Conf. DMACON
006AE    JMP      4F000           ; saute en 4F000 ====> Affichage du message 'INSERT DISK 2'
```

 # Table_#02 Info du tableau récupérée avec la commande **BS 494** avant chaque chargement.

```
00578    00 05 00 00       ← Conf DSKPTH 'Last_Load', DMA DATA=50000
0057C    00 00             ← Marqueur pour Trackloader ou pas
0057E    00 00             ← Pointeur Track en cours

00580    00 00 18 9C       ← Adresse_Start_raw Anim_Part_#01,              (Minimum $189C, taille d'une Track)
                              on ne peut pas commencer en Cylindre 0, obliger de commencer en cylindre 1 minimum

00584    00 04 6F 74       ← Length_To_Read    Anim_Part_#01

00588    00 08 5D 04       ← Adresse_Start_raw Anim_Part_#02

0058C    00 04 8D DE       ← Length_To_Read    Anim_Part_#02

00590    00 04 88 10       ← Adresse_Start_raw Last_Load_D1 →en Fin d'anim = 00 06 CA 94 préalablement copié de l'ancienne table

00594    00 00 BB 80       ← Length_To_Read    Last_Load_D1 →en Fin d'anim = 00 00 9B 78 préalablement copié de l'ancienne table
```

#Copie_Post_LastLoad // code exécuté après **#Chargement dernier Data du Disk1 - InsertDisk2**

```
002C6    LEA      40000,A0          ; A0=$40000               // Adresse source
002CC    LEA      256.S,A1          ; A1=$256                 // Adresse de destination
002D0    MOVE.L   C6.S,D0           ; D0=C6.S
002D4    LSR.L    #2,D0             ; Décalage de 2 bits vers la droite de D0, ce qui nous donnes D0=26DE

002D6    MOVE.L   (A0)+,(A1)+       ; → Copie (A0) vers (A1) puis A0=A0+4 et A1=A1+4
                                    ; Comme on copie en LongWord, cela nous donne (26DE*4)+4=$9B7C de donnée copiée.
                                    ; $256+$9B7C=$9DD2        // Zone copiée (destination) = $256 → $9DD2

002D8    DBF      D0,2D6            ; ← D0=D0-1, tant que D0 est différent de -1, on boucle
002DC    JMP      256.S             ; Saute en $256
```

#Trackloader_Init

```
0002E0   MOVE.W   #10,DFF096        ; Conf DMACON
0002E8   MOVE.W   #2,DFF09C         ; Conf INTREQ
0002F0   MOVE.W   #7FFF,DFF09E      ; Conf ADKCON
0002F8   MOVE.W   #8100,DFF09E      ; Conf ADKCON
000300   ORI.B    #78,BFD100        ; 'Ou' binaire entre 0111 1000 et  $BFD100, donc DF0 a DF3 select et Motor ON
000308   BCLR     #7,BFD100         ; CIA-B / PRB, bit7 mis à 0, motor On
000310   BCLR     #3,BFD100         ;

000318   MOVE.W   #BB8,D6           ; → D6=BB8
00031C   DBF      D6,31C            ; ← D6=D6-1, boucle tant que D6 est différent de -1, nous avons ici une pause

000320   BTST     #5,BFE001         ; → CIA-A / PRA, On test le bit5, lecteur prêt ?
000328   BNE      320               ; → Tant que lecteur n'est pas prêt, on boucle.

00032A   LEA      57C(PC),A3        ; A3=57C(PC) Adresse dans la table, met 70072 dans A3 (l'adresse de la table)
0003EE   MOVE.W   #1,(A3)           ; Copie le word 01 dans (A3), donc met 00 01 dans la table en $57C
000332   RTS                       ; E.T Retour maison
```

#Move Inter.

```
00334    BCLR     #1,BFD100         ; CIA-B / PRB, bit1 mis à 0, DIR=vers l'intérieur
0033C    BCLR     #0,BFD100         ; CIA-B / PRB, bit0 mis à 0, STEP, pré/déplacement tête
00344    BSET     #0,BFD100         ; CIA-B / PRB, bit0 mis à 0, STEP, déplacement tête
0034C    BSR.W    3D8               ; GoSub → #WAIT
00350    MOVE.L   A0,-(A7)          ; Sauvegarde de A0 dans la pile
00352    LEA      57E(PC),A0        ; Pointeur Cylindre en cours mis en A0
00356    ADDQ.W   #1,(A0)           ; (A0)=A0+1 Word
                                    ; On s'est déplacé d'une Track vers l'int et (A0), $57E est incrémenté de 1
                                    ; $57E est donc un pointeur de Track

00358    MOVEA.L (A7)+, A0         ; On restaure A0 de la pile
0035A    BRA      3CC               ; GoTo → #Disque Ready? sans passé par la modification de A3
```

#Move Exter.

```
00035E   BSET     #1,BFD100         ; CIA-B / PRB, bit1 mis à 1, DIR=vers l'extérieur
000366   BCLR     #0,BFD100         ; CIA-B / PRB, bit0 mis à 0, STEP, pré/déplacement tête
00036E   BSET     #0,BFD100         ; CIA-B / PRB, bit0 mis à 0, STEP, déplacement tête
000376   BSR      3D8               ; GoSub → #WAIT
00037A   MOVE.L   A0,-(A7)          ; Sauve A0 dans la pile
00037C   LEA      57E(PC),A0        ; A0=57E(PC), ...encore, doit être un tableau.
000380   SUBQ.W   #1,(A0)           ; (A0)=A0-1 Word, On s'est déplacé d'une Track vers l'ext et (A0), $57E est décrémenté de 1
                                    ; $57E est donc un pointeur de Track
                                    ;
000382   MOVEA.L  (A7)+,A0         ; Restaure A0 de la pile
000384   BRA      3CC               ; GoTo → #Disque Ready? sans passé par la modification de A3
```

#Position Reach?

```
00388    CMP.W    57E(PC),D0        ; Compare $57E=Pointeur De Track en cours avec D0
0038C    BEQ      3A0               ; Si égale alors GoTo ➔ #Avance_Recule
0038E    BGT      3A4               ; si N=0, (plus grand que), alors GoTo ➔ #Go_To_Track
00390    MOVE.W   57E(PC),D6        ; D6=$57E=Adresse du Pointeur de Track en cours
00394    SUB.W    D0,D6             ; D6=D6-D0
00396    SUBQ.W   #1,D6             ; D6=D6-1
00398    BSR      35E               ; ➔ GoSub #Move Exter.
0039A    DBF      D6,398            ; ➔ D6=D6-1, tant que D6 est différent de -1, on boucle
0039E    RTS                        ; E.T Retour maison
```

#Avance_Recule

```
003A0    BSR      334               ; GoSub ➔ #Move Inter.
003A2    BRA      35E               ; GoTo ➔ #Move Exter.
```

#GoTo_Position

```
003A4    SUB.W    57E(PC),D0        ; D0=D0-(contenu de $57E), ak, position en cours
003A8    SUBQ.W   #1,D0             ; D0=D0-1
003AA    BSR      334               ; ➔ GoSub ➔ #Move Inter.
003AC    DBF      D0,3AA            ; ➔ D0=D0-1, tant que D0 est différent de -1, on boucle
003B0    RTS                        ; E.T Retour maison
```

#Retour_T00

```
0003B2   MOVE.B   BFE001,D0         ; ➔ PRA CIA-A dans D0
0003B8   BTST     #4,D0             ;    Test du Bit 4 de D0, à savoir Disque sur T00?
0003BC   BEQ      3C6               ;    Oui, GoTo ➔ #Disque Ready?
0003BE   BSR      35E               ;    GoSub ➔ #Move Exter.
0003C0   BSR      3D8               ;    GoSub ➔ #WAIT
0003C4   BRA      3B2               ; ➔ on boucle sur #Retour_T00
```

#Disque Ready?

```
0003C6   LEA      57E(PC),A3        ; A3=57E(PC), encore un tableau ?
0003CA   CLR.W    (A3)              ; Efface le Word à l'adresse 57E
0003CC   BTST     #5,BFE001         ; ➔ Test du Bit 8 du PRA, à savoir Disque_Ready ?
0003D4   BNE      3CC               ; ➔ Nop ? Non boucle jusqu'à que ce soit le cas.
0003D6   RTS                        ; E.T retour maison
```

#WAIT

```
003D8    MOVE.W   D7,-(A7)          ; Sauvegarde de D7 dans la pile
003DA    MOVE.W   #1388,D7          ; D7=1388
003DE    DBF      D7,3DE            ; ➔➔ Décrémente D7, tan que D7 est différent de -1, on boucle ici
003E2    MOVE.W   (A7)+,D7          ; On Restaure D7 de la pile
003E4    RTS                        ; E.T Retour maison
```

#DF0__SIDE_DOWN__MOTOR_OFF__DIR_EXT

```
003E6    MOVE.B   #FD,BFD100        ; Conf CIA-B / PRB
                                    ; MOTOR OFF, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 UNSELECT, SIDE DOWN, DIR EXT., STEP TRACK=1
003EE    NOP                        ;
003F0    NOP                        ;
003F2    NOP                        ;
003F4    NOP                        ;
003F6    MOVE.B   #F5,BFD100        ; conf CIA-B / PRB
                                    ; MOTOR OFF, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 SELECT, SIDE DOWN, DIR EXT., STEP TRACK=1
003FE    MOVE.W   #B000,D0          ; D0=B000
00402    DBF      D0,402            ; ➔➔ D0=D0-1, tant que D0 est différent de -1, on boucle, c'est une pause
00406    LEA      57C(PC),A3        ; A3=57C, Marqueur pour Trackloader ou pas
0040A    CLR.W    (A3)              ; On efface le marqueur
0040C    RTS                        ; E.T retour maison
```

#DF0__SIDE_DOWN__MOTOR_ON__DIR_EXT

```
0040E    MOVE.B   #7D,BFD100        ; conf CIA-B / PRB
                                    ; MOTOR ON, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 UNSELECT, SIDE DOWN, DIR EXT., STEP TRACK=1
00416    NOP                        ;
00418    NOP                        ;
0041A    MOVE.B   #75,BFD100        ; conf CIA-B / PRB
                                    ; MOTOR ON, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 SELECT, SIDE DOWN, DIR EXT., STEP TRACK=1
00422    MOVE.W   #B000,D7          ; D7=B000, compteur pause
00426    DBF      D7,426            ; ➔➔ D7=D7-1, tant que D7 est différent de -1, on boucle, c'est une pause
0042A    RTS                        ; E.T Retour maison
```

```
#DF0__SIDE_UP__MOTOR_ON__DIR_EXT
0042C    MOVE.B   #79,BFD100        ; conf CIA-B / PRB
                                    ; MOTOR ON, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 UNSELECT, SIDE UP, DIR EXT., STEP TRACK=1
00434    NOP                        ;
00436    NOP                        ;
00438    MOVE.B   #71,BFD100        ; conf CIA-B / PRB
                                    ; MOTOR ON, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 SELECT, SIDE UP, DIR EXT., STEP TRACK=1
00440    MOVE.W   #B000,D7          ; D7=B000, compteur pause
00444    DBF      D7,444            ; ←→ D7=D7-1, tant que D7 est différent de -1, on boucle, c'est une pause
00448    RTS                        ; E.T Retour maison
```

```
#Trackloader_Start                  A0=Adr_dest_mémoire              A1=Pointeur tableau(Adress_Start_raw)
0044A    MOVE.L   A0,-(A7)          ; Sauvegarde de A0 dans la pile
0044C    BSR      454               ; GoSub → #Lecture_Table_and_Start_Trackload_Or_Not
00450    MOVEA.L  (A7)+,A0          ; Restauration de A0 de la pile
00452    RTS                        ; E.T Retour Maison
```

```
#Lecture_Table_and_Start_Trackload_Or_Not
00454    LEA      57C(PC),A3        ; A3=57C (Adresse dans la table)                  // Marqueur prêt à Trackloader ou pas
00458    TST.W    (A3)              ; Test contenu de A3 avec zero
0045A    BNE      460               ; Si différent alors GoTo → #Side_Select
0045C    BSR      2E0               ; GoSub → #Trackloader_Init

#Side_Select
00460    MOVE.L   (A1)+,D0          ; D0=A1 puis A1=A1+4
00462    CMP.L    #84468,D0         ; D0-84468 et modifie les Flags en conséquence
                                    ; Test pour savoir sur quel Side on va utiliser
                                    ; Même valeur que le 1er TrackLoader à savoir 84468

                                    ; Selon résultat, on ira sur la routine Face_UP ou Face_Down
00468    BGE      46E               ; Si > GoTo → #Recup_Info_pour_Trackloader_1 et #DF0__SIDE_UP__MOTOR_ON__DIR_EXT

0046A    BSR      40E               ; sinon GoTo ---------------------------------->  #DF0__SIDE_DOWN__MOTOR_ON__DIR_EXT

0046C    BRA      470               ; Et ensuite GoTo → #Recup_Info_pour_Trackloader_2
```

```
#Recup_Info_pour_Trackloader_1
0046E    BSR      42C               ; GoSub → #DF0__SIDE_UP__MOTOR_ON__DIR_EXT

#Recup_Info_pour_Trackloader_2
00470    MOVE.L   D0,D3             ; A cette étape D0=(LongWord de $580) donc valeur du tableau=$189C
                                    ; à savoir 'Adress_Start_raw'
                                    ; D3=D0, donc D3=$189C, permet de savoir quel Track est concernée.

00472    DIVU     #189C,D0          ; D0/$189C et met le résultat dans D0
00476    CMP.W    #56,D0            ; Compare D0 avec la valeur $56
0047A    BLT      486               ; Si plus petit que, on branche en 846
0047C    SUBI.L   #84468,D3         ; Sinon, D3=D3-$84468
00482    SUBI.W   #56,D0            ; D0=D0-$56
00486    MOVEQ    #0,D2             ; D2=00
00488    MOVE.W   D0,D2             ; D2=D0
0048A    BSR      $388              ; GoSub → #Position Reach?
0048E    MULU     #189C,D2          ; A ce moment, D2=position atteinte=position de la tête=Track en cours
                                    ; On le multiplie par la taille d'une Track,
                                    ; il devient la taille en position_raw où l'on se trouve.

00492    SUB.L    D2,D3             ; cette étape D3=(LongWord de $580) donc valeur du tableau=$189C
                                      ou si D0 Start_Track était plus grand que $56
                                      on soustrait la position_raw_atteinte de D3

                                      En fait, selon l Adresse_Start_raw dans le tableau, on va utiliser une face précise.
                                      Et donc cette opération effectuée sur D3 permet
                                      de repositionner l' Adresse_Start_raw au départ tout en changeant de face.
                                      D3=(valeur tableau en $580)-(position atteinte)

00494    MOVE.L   (A1),D4           ; Copie le LongWord contenu à l'adresse A1 dans D4
                                    ; 1er Appel (chargement Anim_Part_#01,      A1=584=(Length_To_Read) = $46F74
                                    ; 2em Appel (chargement Anim_Part_#02,      A1=58C=(Length_To_Read) = $48DDE
                                    ; 3em Appel (chargement Last_Load,          A1=594=(Length_To_Read) = $9B78

00496    LSR.L    #2,D4             ; Décalage de 2 bits vers la droite de D4, revient à diviser par 4 D4
00498    SUBQ.L   #1,D4             ; D4=D4-1

0049A    BSR      4A2               ; → GoSub → #Trackloader_Base
0049C    BSR      334               ;    GoSub → #Move Inter.
004A0    BRA      49A               ; ← Branche en 49A, on boucle sur #Trackloader_Base
```

#Trackloader_Base

```
004A2    MOVEQ    #0F,D2           ; D2=F
004A4    MOVE.L   #18B8,D0         ; ➔ D0=$18B8
004AA    MOVEA.L  578(PC),A6       ; A6=Adresse dans la table=000578
004AE    MOVE.W   #2,DFF09C        ; Conf INTREQ, Disk Block Finished Interupt
004B6    MOVE.L   A6,DFF020        ; Conf DSKPTH     (conf qui provient de la table)
004BC    MOVE.W   #8210,DFF096     ; Conf DMACON, DSKEN enable, ALL DMA enable
004C4    MOVE.W   #4489,DFF07E     ; Conf DSKSYNC = $4489 (standard AmigaDos)
004CC    MOVE.W   #7F00,DFF09E     ; Conf ADKCON spécifique
004D4    MOVE.W   #B500,DFF09E     ; Encore une fois pour démarrer le transfert
                                     FAST=2us, WORDSYNC=active, MFM precomp, recomp=140ns

004DC    MOVE.W   #4000,DFF024     ; Conf DSKLEN, Write enable (ram or disk)

#Test CIA Ready
004E4    MOVE.B   BFDD00,D7        ; D7=BFDD00
004EA    MOVE.B   BFDD00,D7        ;       ➔ D7=icr register de CIA-B
004F0    BTST     #4,D7            ;          Test du bit4 (FLAG) de icr
004F4    BEQ      4EA              ;       ➔ Interruption générée ? On boucle

004F6    ADDI.W   #8001,D0         ; ADD signé sur D0 (qui est a $18B8 voir quelques lignes au-dessus) avec $8001
                                     ce qui nous donne : $98B9 et flag C=0

004FA    MOVE.W   D0,DFF024        ; CONF DSKLEN, Disk DMA Enable, Length of DMA data=$18B9
00500    MOVE.W   D0,DFF024        ; Encore une fois pour déclencher la lecture

00506    MOVE.W   DFF01E,D0        ;       ➔ D0=INTREQR
0050C    BTST     #1,D0            ;          Test du Bit1 de INTREQR, Level 1 Disk Block Finished Interupt
00510    BEQ      506              ;       ➔ Test ?, On boucle

00512    MOVE.L   (A6)+,D0         ; Valeur de DSKPTH dans D0, (A6=DFF024) puis A6=A6+4
00514    MOVE.L   (A6)+,D7         ; Valeur de DSKLEN dans D7, (A6=DFF028) puis A6=A6+4
00516    ADD.L    D0,D0            ; D0=D0+D0
00518    ANDI.L   #AAAAAAAA,D0     ; Post_Traitement MFM bit impair dans D0
0051E    ANDI.L   #55555555,D7     ; Post_Traitement MFM bit pair dans D7
00524    OR.L     D7,D0            ; Traitement MFM
00526    CMP.L    #42535432,D0     ; Compare D0 avec 42535432
                                     42 53 54 32 En Ascii = BST2
                                     Si la signature n'est pas trouvée, on re-check et finalement,
                                     si toujours pas bon, partira sur la routine d'écran rouge plus bas.

0052C    BEQ      53C              ; Oui ? Signature trouvée, GoTo ➔ #Traitement_MFM_BASE
                                     On commence le traitement réel des données et le Trackload

0052E    DBF      D2,4A4           ; ➔ D2=D2-1, tant que D2 est différent de -1, on boucle en 4A4
                                     C'est reparti pour un tour

00532    MOVE.W   #F00,DFF180      ; ➔ Fond d'écran en rouge
0053A    BRA      532              ; ➔ DeadLoop fond rouge
```

#Traitement_MFM_BASE

```
0053C    MOVEQ    #0,D2            ; D2=00
0053E    MOVE.W   #626,D2          ; D2=626

00542    TST.W    D3               ; Test contenu de D3 avec zero                    // D3=Delta calculé préalablement.
00544    BEQ      550              ; Si égal alors GoTo ➔ #Check_Traitement_MFM
00546    ADD.W    D3, D3           ; Sinon D3=D3+D3                                  // on fait x2 sur D3

                                     Rappel, D3=((Valeur tableau Adresse_Start_raw)-(valeur position atteinte))

00548    ADDA.L   D3,A6            ; A6=A6+D3                                        // A6=DSKPTH+décodage en cour
0054A    LSR.W    #3,D3            ; Décale de 3 Bits vers la gauche le LongWord D3// Revient à diviser par 8 D3
                                     Donc delta entre position atteinte et (valeur en Length_To_Read)/8

0054C    SUB.W    D3,D2            ; D2(position_raw actuel)-Delta calculé ci-dessus
0054E    MOVEQ    #0,D3            ; D3=0                                            // On remet D3 à Zero

#Check_Traitement_MFM
00550    CMP.L    D2,D4            ; Compare D4-D2
00552    BGT      55A              ; si résultat plus grand que GoTo ➔ #Traitement MFM bit pair
00554    MOVE.W   D4,D2            ; D2=D4
00556    ADDQ.W   #4,A7            ; A7=A7+4
00558    BRA      55E              ; GoTo ➔ #Start_Decodage
```

#Traitement MFM bit pair

```
0055A    SUB.L    D2,D4            ; D4=D4-D2
0055C    SUBQ.L   #1,D4            ; D4=D4-1

#Start_Decodage
0055E    MOVE.L   #55555555,D5     ; D5=55555555, masque MFM bit pair
00564    MOVE.L   (A6)+,D0         ; ➔ Copie contenue de A6 dans D0 puis A6=A6+4
00566    MOVE.L   (A6)+,D7         ;     Copie contenue de A6 dans D7 puis A6=A6+4
00568    AND.L    D5,D0            ;     Traitement MFM
0056A    AND.L    D5,D7            ;     Traitement MFM
0056C    ADD.L    D0,D0            ;     Traitement MFM
0056E    OR.L     D7,D0            ;     Traitement MFM
00570    MOVE.L   D0,(A0)+         ;     Copie de D0 à l'adresse pointé par A0 puis A0=A0+4
00572    DBF      D2,564           ; ➔ D2=D2-1 et tant que D2 est différente de -1, on boucle
00576    RTS                       ; E.T  Retour maison
```

#Decrypt/Decomp_&_execution

```
006B4    MOVE.L   (A0)+,800          ; Copie Le LongWord de l'adresse pointée par A0 à l'adresse $800
                                       $800=$46F74     et $5DA88 decomp anim_part_#01

006BA    ADDI.L   #16B14,800         ; Ajoute $16B14 à celle-ci à l'adresse $800
006C4    MOVE.L   A0,7F8             ; Copie A0 à l'adresse $7F8
006CA    MOVE.L   #93C,7FC           ; Met Le LongWord 93C à l'adresse $7FC
006D4    CLR.W    AAE                ; Efface le word à l'adresse $AAE
006DA    CLR.W    AAC                ; Efface le word à l'adresse $AAC
006E0    CLR.W    AB0                ; Efface le word à l'adresse $AB0
006E6    CLR.W    AB2                ; Efface le word à l'adresse $AB2
006EC    CLR.W    AB4                ; Efface le word à l'adresse $AB4
006F2    CLR.W    AB8                ; Efface le word à l'adresse $AB8
006F8    MOVE.W   #3,AB6             ; Met le Word $0003 à l'adresse $AB6

00700    BTST     #6,BFE001          ; Bouton Fire0 appuyé ?
00708    BNE      71A                ; Pas appuyé ? GoTo ➔ $71A

0070A    MOVE.W   #FFF,DFF180        ; Fond d'écran en blanc quand on appuie sur la souris.
00712    MOVE.B   #1,1ABE            ; met le Byte 01 en $1ABE

0071A    TST.B    1529               ; Compare 0 avec le contenu de l'adresse $1529
00720    BEQ      730                ; Si égale alors GoTo ➔ $730
00722    CLR.B    1529               ; Efface le Byte en $1529

00728    TST.B    1529               ; ➔ Compare 0 avec le contenu de l'adresse $1529
0072E    BEQ      728                ; ➔ Si égale alors GoTo ➔ $728

00730    TST.W    AAC                ; Compare 0 avec le contenu de l'adresse $AAC
00736    BNE      700                ; Si différent alors on GoTo ➔ $700

00738    TST.W    AB8                ; Compare 0 avec le contenu de l'adresse $AB8
0073E    BNE      752                ; Si différent alors GoTo ➔ $752

00740    MOVE.W   AB6,AB8            ; Copie le word en $0AB6 à l'adresse $AB8
0074A    MOVE.L   7FC,A1             ; A1=$7FC
00750    JSR      (A1)               ; JSR (A1), saute à l'adresse pointée en A1 (défini plus haut dans le code)

00752    TST.W    AB2                ; Compare 0 avec le contenue de l'adresse $AB2
00758    BEQ      772                ; Si égale alors GoTo ➔ $772

0075A    TST.W    AB0                ; Compare 0 avec le contenu de l'adresse $AB0
00760    BNE      772                ; Si différent alors GoTo ➔ $772

00762    MOVE.W   AA8,D0             ; Met le Word $0AA8 en D0
00768    BSR      F44                ; GoSub ➔ $F44
0076C    CLR.W    AB2                ; Efface le Word en $AB2

00772    TST.W    AAE                ; Compare 0 avec le contenu de l'adresse $AAE
00778    BPL      786                ;

0077A    MOVE.L   7FC, D0            ; Met le LongWord $7FC en D0
00780    CMP.L    93C, D0            ; Compare le LongWord $93C avec D0
00786    BNE      700                ; Si différent, GoTo $700

0078A    CLR.W    AAE                ; Efface le Word en $AAE
00790    MOVE.L   7F8,A6             ; Met le LongWord 7F8 en A6
00796    MOVE.W   (A6)+,D0           ; Copie le Word de (A6) dans D0 puis A6=A6+2
00798    BMI      7AA                ;
0079A    MOVEA.L  10(PC,D0.W),A1     ;
0079E    JSR      (A1)               ; A1=$16B18
007A0    MOVE.L   A6,7F8             ; Copie A6 à l'adresse $7F8
007A6    BRA      700                ; Branche en $700

007AA    RTS                         ;

On arrête ici notre désassemblage car cela devient trop long, nous avons à peu prêt tout ce qu'il nous faut.
Théoriquement la routine de décompression se situe de $6B4 → $FBC     soit 2312 Octets, mais bon, peu importe...
```

A noter que l'on peut ajouter une option pour sauter les animations.
Au lieu d'avoir un **fond d'écran blanc**, on peut modifier pour sauter directement en *$638* pour effectuer le dernier chargement.

Donc remplacer

```
0070A    MOVE.W   #FFF,DFF180        ; Fond d'écran en blanc quand on appuie sur la souris pendant l'animation.
```

par

```
0070A    BRA      638                ; GoTo ➔ #Chargement dernier Data du Disk1 - InsertDisk2
```

Voilà un tableau des principales Sous-Routine du *Trackloader*.
Le but étant de voir si on peut écraser celles-ci sans à avoir à faire trop de modification dans le code original.

Adr Mémoire	Sous Routine	Appelé par	Info	Dans la zone Potentielle ?	Zone Potentielle
2E0-332	#Trackloader_Init	45C	Pas nécessaire	**dedans**	
334-35A	#Move Inter.	3A0 3AA 49C	Pas nécessaire	**dedans**	
35E-384	#Move Exter.	398 3A2 3B2	Pas nécessaire	**dedans**	
388-39E	#Position Reach?	48A	Pas nécessaire	**dedans**	
3A0-3A2	#Avance_Recule	38C	Pas nécessaire	**dedans**	
3A4-3B0	#GoTo_Position	38E	Pas nécessaire	**dedans**	
3B2-3C4	#Retour_T00	**2BC** 3C4	Pas nécessaire	**A PATCHER**	
3C6-3D6	#Disque Ready?	3BC	Pas nécessaire	**dedans**	
3D8-3E4	#WAIT	34C 376 3C0	Pas nécessaire	**dedans**	
3E6-40C	#DF0__SIDE_DOWN__MOTOR_OFF__DIR_EXT	**NONE**	Pas nécessaire	**dedans**	
40E-42A	#DF0__SIDE_DOWN__MOTOR_ON__DIR_EXT	46A	Pas nécessaire	**dedans**	
42C-448	#DF0__SIDE_UP__MOTOR_ON__DIR_EXT	46E	Pas nécessaire	**dedans**	
44A-452	#Trackloader_Start	60A 62A 65A	**PLUS** nécessaire	**dedans**	
454-46C	#Lecture_Table_and_Start_Trackload_Or_Not	44C	Pas nécessaire	**dedans**	
46E-470	#Recup_Info_pour_Trackloader_1	468	Pas nécessaire	**dedans**	
470-4A0	#Recup_Info_pour_Trackloader_2	46C	Pas nécessaire	**dedans**	
4A2-4DC	#Trackloader_Base	49A	Pas nécessaire	**dedans**	
4E4-53A	#Test CIA Ready	**NONE**	Pas nécessaire	**dedans**	
53C-54E	#Traitement_MFM_BASE	52C	Pas nécessaire	**dedans**	
550-558	#Check_Traitement_MFM	544	Pas nécessaire	**dedans**	
55A-55E	#Traitement MFM bit pair	552	Pas nécessaire	**dedans**	
55E-576	#Start_Decodage	558	Pas nécessaire	**dedans**	

On peut donc assez facilement ‘écraser’ le code original par notre propre *Trackloader*.

Part 9 Analyse du dernier code chargé : Last_Load et du TrackLoader #3

Si on pose un **BreakPoint** avant le dernier trackload, c'est-à-dire en **$656**
Que l'on remplit la zone de destination avec un pattern quelconque en prenant une petite marge : **O "PaTtErN", 3FFF0 49B88**
et en posant bien sûr un **BreakPoint** juste après le Trackload, c'est-à-dire en **$65E**, que l'on retourne au code.
Une fois le trackload fini et notre breakPoint *$65E* atteint, on peut clairement voir que les DATA trackloader sont du code pure, pas de phase de compression ou d'encryption et que la zone remplie est bien celle attendue *$40000 → $49B78*

Puis quelques lignes de code plus tard, on exécute ce code copié : **006AE JMP 4F000**
Ce qui revient à analyser le code précédemment trackloadé en *$40000* qui est maintenant en place en *$4F000*

L'exécution de la routine **#Recopie_2C6_vers_4F000** qui intervient juste après, copie la routine **#Copie_Post_LastLoad** en *$4F000*

```
#Recopie_Copie_Post_LastLoad_vers_4F000
4F000   LEA     40000,A0        ; A0=$40000             // Adresse source
4F006   LEA     256.S,A1        ; A1=$256               // Adresse de destination
4F00A   MOVE.L  C6.S,D0         ; D0=Compteur pour la boucle DBF suivante
                                ; On comprend maintenant l'importance de la valeur de $C6

4F00E   LSR.L   #2,D0           ; Décalage de 2 bits vers la droite de D0, ce qui nous donnes D0=$26DE

4F010   MOVE.L  (A0)+,(A1)+     ; → Copie (A0) vers (A1) puis A0=A0+4 et A1=A1+4
                                ;   Copie en LongWord, cela nous donne (26DE*4)+4=$9B7C de donnée copiée.
                                ;   $256+$9B7C=$9DD2     // Zone copiée (destination) = $256 → $9DD2
                                ;   Rappel : Taille du code préalablement trackloadé en $40000 = $BB80
                                ;   Donc l'intégralité du code trackload est déplacé en $256

4F012   DBF     D0,4F010        ; ← D0=D0-1, tant que D0 est différent de -1, on boucle
4F016   JMP     256.S           ; Saute en $256
```

Après étude du code, voilà ce que l'on peut retenir. (à savoir, encore un **TrackLoader,** quasiment identique)
On va remonter l'analyse jusqu'au Menu principal du jeu ET du chargement du 1er Level
Le tout étant effectué avec les commandes **BS** et **ST**

Taper : D 256

```
256     BRA     1DE0                    ; GoTo → #Base_LastLoad
```

#WaitSyncV

```
25A     MOVEQ   #0,D1                   ; D1=00
25C     MOVE.B  BFE801,D1               ; D1=todlo (vsync)
262     BSR     26A                     ; → GoSub #Working_on_BE2_Mark
264     DBF     D1,262                  ; ← On boucle tant que c'est nécessaire.
268     RTS                             ; E.T retour maison
```

#Working_on_BE2_Mark

```
0026A   MOVE.L  BE2, D0                 ; Récupère le LongWord de $BE2 dans D0
00270   ADD.L   D0,D0                   ; D0= D0+D0
00272   BTST    #17, D0                 ; ...
00276   BNE     286                     ; GoSub #Working_on_BE2_Mark_02
00278   BTST    #1, D0                  ;
0027C   BNE     28C                     ; GoSub #Working_on_BE2_Mark_03
0027E   MOVE.L  D0,BE2                  ;
00284   RTS                             ;
```

#Working_on_BE2_Mark_#2

```
00286   BTST    #1,D0                   ; test du Bit1 de D0 avec zero
0028A   BNE     28E                     ; Si pas égale alors GoTo → #Working_on_BE2_Mark_#4

#Working_on_BE2_Mark_#3
0028C   ADDQ.L  #1,D0                   ; D0=D0+1

#Working_on_BE2_Mark_#4
0028E   MOVE.L  D0,BE2                  ; Copie le LongWord D0 à l'adresse $BE2
00294   RTS                             ; E.T retour maison
```

#Base_LastLoad

```
1DE0    MOVE.B   C0.S,D0                ; D0=$C0
1DE4    MOVE.B   D0,BBF.S               ; copie D0 à l'adresse $BBF
1DE8    MOVE.W   #2700,SR               ; Conf. Status register
1DEC    MOVE.W   #7FFF,D0               ; D0=$7FFF
1DF0    MOVE.W   #7C7F,DFF096           ; Conf DMAcon
1DF8    MOVE.W   D0,DFF09A              ; Conf INTENA
1DFE    MOVE.W   D0,DFF09C              ; Conf INTREQ
1E04    LEA      256.S,A7               ; A7=256
1E08    BSR      25A                    ; GoSub ➔ #WaitSyncV
1E0C    MOVE.B   #7F,BFDD01             ; ??, astuce de programmation pour accéder au bits 0 à 7 du CIA-A ? ou bug ?
                                        ; En tout cas, désactivation toutes les Interupts. Du CIA-B
                                        ;
1E14    MOVE.B   #88,BFED01             ; Conf CIAA / Icr
1E1C    MOVE.B   #0,BFEE01              ; Conf CIAA / Cra

1E24    MOVE.L   #99A8,6C.S             ; Copie $99A8 en $6C      ?? // marqueur servant a ?
1E2C    MOVE.L   #9A70,68.S             ; Copie $9A70 en $68      ?? // marqueur servant a ?

1E34    MOVE.W   #2000,SR               ; Conf. Status register
1E38    MOVE.W   #C028,DFF09A           ; Conf. INTENA
1E40    MOVE.W   #8650,DFF096           ; Conf. DMACON

1E48    JSR      19C4.S                 ; GoSub ➔ #Trackloader_2_Init
1E4C    JSR      1A96.S                 ; GoSub ➔ #Retour_T00
1E50    JSR      8C30                   ; GoSub ➔ #BASE_INSERT_DISK_ASKED
1E56    BSR      9D3E                   ; GoSub ➔ #Wait_BB6

1E5A    MOVE.W   #1A0,DFF096            ; Conf. DMACON

1E62    BSR      1C5C                   ; GoSub ➔ #Phase_De_Chargement_#1
```

#Trackloader_2_Init

```
19C4    MOVE.W   #10,DFF096             ; Conf DMACON
19CC    MOVE.W   #2,DFF09C              ; Conf INTREQ
19D4    MOVE.W   #7FFF,DFF09E           ; Conf ADKCON
19DC    MOVE.W   #8100,DFF09E           ; Conf ADKCON
19E4    ORI.B    #78,BFD100             ; 'Ou' binaire entre 0111 1000 et  $BFD100, donc DF0 a DF3 select et Motor ON
19EC    BCLR     #7,BFD100              ; CIA-B / PRB, bit7 mis à 0, motor On
19F4    BCLR     #3,BFD100              ; CIA-B, bit3 à 0, DF0 sélectionné
19FC    MOVE.W   #BB8,D6                ; D6=BB8
1A00    DBF      D6,1A00                ; ➔➔ D6=D6-1, boucle tant que D6 est différent de -1, nous avons ici une pause

1A04    BTST     #5,BFE001              ; ➔ CIA-A / PRA, On test le bit5, lecteur prêt ?
1A0C    BNE      1A04                   ; ➔ Tant que lecteur pas prêt, on boucle pour attendre qu'il le soit

1A0E    LEA      C1E(PC),A3             ; A3=$C1E(PC) Adresse dans la table.
1A12    MOVE.W   #1,(A3)                ; Copie le word 01 dans (A3), donc met 00 01 dans la table en $C1E
1A16    RTS                             ; E.T Retour maison
```

#Retour_T00

```
1A96    MOVE.B   BFE001,D0              : D0=CIA PRA
1A9C    BTST     #4,D0                  ; On test le bit4, TK0, Tête sur T00 ?
1AA0    BEQ      1AAA                   ; Oui ? GoTo ➔ #Disque Ready?
1AA2    BSR      1A42                   ; Non ? GoSub ➔ #Move Exter.
1AA4    BSR      1ABC                   ; GoSub ➔ #WAIT
1AA8    BRA      1A96                   ; GoTo ➔ #Retour_T00
```

#WAIT

```
1ABC    MOVE.W   D7,-(A7)               ; Sauvegarde de D7 dans la pile
1ABE    MOVE.W   #1388,D7               ; D7=$1388
1AC2    DBF      D7,1AC2                ; ➔➔ Décrémente D7, tan que D7 est différent de -1, on boucle ici
1AC6    MOVE.W   (A7)+, D7              ; On Restaure D7 de la pile
1AC8    RTS                             ; E.T Retour maison
```

#Disque Ready?

```
1AAA    LEA      C20(PC),A3             ; A3=position dans la table
1AAE    CLR.W    (A3)                   ; Efface le Word à l'adresse $C20                    // Marqueur Track en cours

#Disque Ready?
1AB0    BTST     #5,BFE001              ; ➔ Test du Bit 8 du PRA, à savoir Disque_Ready
1AB8    BNE      1AB0                   ; ➔ Nop ? GoTo ➔ $1AB0
1ABA    RTS                             ; E.T Retour maison
```

#Move Exter.

```
1A42    BSET    #1,BFD100               ; CIA-B / PRB, bit1 mis à 1, DIR=vers l'extérieur
1A4A    BCLR    #0,BFD100               ; CIA-B / PRB, bit0 mis à 0, STEP, pré/déplacement tête
1A52    BSET    #0,BFD100               ; CIA-B / PRB, bit0 mis à 0, STEP, déplacement tête
1A5A    BSR     1ABC                    ; GoSub ➔ #WAIT
1A5E    MOVE.L  A0,-(A7)                ; Sauvegarde de A0 dans la pile
1A60    LEA     C20(PC),A0              ; Pointeur Cylindre en cours mis en A0
1A64    SUBQ.W  #1,(A0)                 ; (A0)=A0-1 Word, On s'est déplacé d'une Track vers l'int et (A0)
                                          $C20 est décrémenté de 1
                                          $C20 est donc un pointeur de TRACK, en l'occurrence 'PISTE' car
                                          le trackloader fonctionne en SIDE.

1A66    MOVE.L  (A7)+,A0                ; Restaure A0 de la pile
1A68    BRA     1AB0                    ; On branche dans la sous-routine GoTo ➔ #Disque Ready?
```

#BASE_INSERT_DISK_ASKED

```
8C30    MOVEQ   #1,D0                   ; D0=01                         // Choix du disk demandé (00=Disk1, 01=Disk2)

#BASE_INSERT_DISK_ASKED_WITHOUT_FLOPPY_DISK_MARKER
8C32    MOVE.W  D0,A0.S                 ; Copie de D0 l'adresse mémoire $A0         (marqueur Disk demandée)
8C36    BSR     9D3E                    ; GoSub ➔ #Wait_BB6
8C3A    LEA     DFF000,A6               ; A6=DFF000
8C40    MOVE.W  #1A0,96(A6)             ; Conf de DMACON
8C46    MOVE.L  #8D1E,DFF080            ; Conf de Coperlist / COP1LCH //Conf. De l'adresse de la Copperlist(1) = $8D1E

8C50    MOVE.W  #1200,100(A6)           ; Conf BitPlan / BPLCON0
8C56    CLR.L   102(A6)                 ; Conf BitPlan / BPLCON1
8C5A    CLR.L   108(A6)                 ; Conf BitPlan / BPL1MOD
8C5E    MOVE.L  #2C81F4C1,8E(A6)        ; Conf Display / DIWSTRT
8C66    MOVE.L  #3800D0,92(A6)          ; Conf Display / DDFSTRT

8C6E    LEA     30000, A0               ; A0=30000
8C74    MOVE.W  #7CF,D1                 ; D1=7CF
8C78    CLR.L   (A0)+                   ; ➔ Boucle d'effacement de la zone Mémoire (A0)
8C7A    DBF     D1,8C78                 ; ➔ Tant que D1 est différent de -1, on boucle

8C7E    TST.W   A0.S                    ; Test Du Word en $A0               // Zone tampon de lecture du Disk (signature)
8C82    BEQ     8C8C                    ; Si égale à Zero alors GoSub ➔ #CHECK_DISK
8C84    LEA     8D2A,A0                 ; A0=8D2A
8C8A    BRA     8C92                    ; GoTo ➔ #CHECK_DISK_A0
```

#Wait_BB6

```
9D3E    CLR.B   BB6.S                   ; On efface le Byte en BB6
9D42    TST.B   BB6.S                   ; ➔ On test le Byte en BB6
9D46    BEQ     9D42                    ; ➔ Tant que différent de zero, on boucle
9D48    RTS                             ; E.T Retour Maison
```

```
#CHECK_DISK

8C8C    LEA     8DEE,A0                 ; A0=8DEE

#CHECK_DISK_A0
8C92    LEA     30E16,A1                ; A1=30E16
8C98    MOVEQ   #6,D1                   ; D1=06, compteur #1
8C9A    MOVEQ   #0D,D2                  ; ➔ D2=0D, compteur #2
8C9C    MOVE.W  (A0)+,(A1)+             ;     ➔ Boucle de copie de (A0) vers (A1) défini juste au-dessus
8C9E    DBF     D2,8C9C                 ;     ← tant que D2 est différent de -1, on boucle
8CA2    ADDA.W  #C,A1                   ;     A1=A1+$C
8CA6    DBF     D1,8C9A                 ; ← tant que D1 est différent de -1, on boucle

8CAA    LEA     31021,A1                ; A1=31021
8CB0    LEA     8EB2,A0                 ; A0=8EB2
8CB6    MOVEQ   #6,D1                   ; D1=06, compteur #1
8CB8    MOVEQ   #15,D2                  ; ➔ D2=15, compteur 2
8CBA    MOVE.B  (A0)+,(A1)+             ;    ➔ Boucle de copie de (A0) vers (A1) défini juste au-dessus
8CBC    DBF     D2,8CBA                 ;    ← tant que D2 est différent de -1, on boucle
8CC0    ADDA.W  #12,A1                  ;       A1=A1+$C
8CC4    DBF     D1,8CB8                 ; ← tant que D1 est différent de -1, on boucle

8CC8    MOVE.W  #F00,DFF182             ; Conf Palette Color01    // Affiche du message 'PLEASE INSER BEAST DISK'
8CD0    BSR     9D3E                    ; GoSub ➔ #Wait_BB6
8CD4    MOVE.W  #8180,DFF096            ; Conf DMACON

#DISK2_OR_DISK1_INSERTED?
8CDC    BSR     1ACA                    ; GoSub ➔ #DF0__SIDE_DOWN__MOTOR_OFF__DIR_EXT
8CE0    TST.B   BB4.S                   ; ➔ On attend que l'on appuie sur FIRE
8CE4    BEQ     8CE0                    ; ← Tant que l'on n'appuie pas, on boucle
8CE6    BSR     19C4                    ; GoSub ➔ #Trackloader_2_Init
8CEA    BSR     1A96                    ; GoSub ➔ #Retour_T00

8CEE    LEA     B0.S,A0                 ; A0=B0                   Adresse Mémoire Destination
8CF2    LEA     8D16,A1                 ; A1=8D16                 Pos. Tableau      // Check_Signature DISK, 1 WORD
8CF8    BSR     1B2E                    ; GoSub ➔ #TrackLoader_2

8CFC    TST.W   A0.S                    ; Test des bits de l'adresse $A0
                                          $A0 = Zone tampon de lecture du Disk (signature)
8D00    BEQ     8D0C                    ; Si bits à zéro alors GoTo ➔ #Check_Signature_DISK_01

#Check_Signature_DISK_02
8D02    CMPI.W  #D0D2,B0.S              ; Sinon, compare le word D0D2 (qui est la signature du Disk Original N°2)
8D08    BNE     8CDC                    ; Si signature Disk2 pas trouvée, on boucle sur #DISK2_OR_DISK1_INSERTED?
8D0A    RTS                             ; Signature trouvée, on retourne au code
```

```
#Check_Signature_DISK_01

8D0C    CMPI.W  #4,B0.S                 ; Compare le word 0004 avec le contenue lue en l'adresse $B0 (dernier chargement)
8D12    BNE.B   8CDC                    ; Si différent, on repart pour un tour
8D14    RTS                             ; E.T Retour Maison
```

```
#DF0__SIDE_DOWN__MOTOR_OFF__DIR_EXT

1ACA    MOVE.B  #FD,BFD100              ; Conf CIA-B / PRB
                                          MOTOR OFF, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 UNSELECT, SIDE DOWN, DIR EXT., STEP TRACK=1

1AD2    MOVE.W  #100,D0                 ; D0=100
1AD6    DBF     D0,1AD6                 ; ←➔ D0=D0-1, tant que D0 est différent de -1, on boucle, c'est une pause
1ADA    MOVE.B  #F5,BFD100              ; Conf CIA-B / PRB
                                          MOTOR OFF, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 SELECT, SIDE DOWN, DIR EXT., STEP TRACK=1

1AE2    MOVE.W  #B000,D0                ; D0=B000
1AE6    DBF     D0,1AE6                 ; ←➔ D0=D0-1, tant que D0 est différent de -1, on boucle, c'est une pause
1AEA    LEA     C1E(PC),A3              ; A3=position dans la table
1AEE    CLR.W   (A3)                    ; On efface le marqueur
1AF0    RTS                             ; E.T Retour maison
```

```
#Trackloader_Start              A0=Adr_dest_mémoire             A1=Pointeur tableau(Adress_Start_raw)

1B2E    MOVE.L  A0,-(A7)                ; Sauvegarde de A0 dans la pile
1B30    BSR     1B38                    ; GoSub ➔ #Lecture_Table_and_Start_Trackload_Or_Not
1B34    MOVE.L  (A7)+,A0                ; Restauration de A0 de la pile
1B36    RTS                             ; E.T Retour maison
```

#Lecture_Table_and_Start_Trackload_Or_Not

```
1B38    LEA     C1E(PC),A3      ; A3=C1E (Adresse dans la table)
1B3C    TST.W   (A3)            ; Test contenu de A3 avec zéro
1B3E    BNE     1B44            ; Si pas égal alors GoTo ➔ #Side_Select
1B40    BSR     19C4            ; GoSub ➔ #Trackloader_2_Init

#Side_Select
1B44    MOVE.L (A1)+, D0        ; D0=A1 puis A1=A1+4
1B46    CMP.L   #84468,D0       ; D0-84468 et modifie les Flag en conséquence
                                ; Test pour savoir sur quel Side on va utiliser
                                ; Même valeur que les autres TrackLoader à savoir 84468
                                ; (Selon résultat, on ira sur la routine Face_UP ou face_Down


1B4C    BGE     1B52            ; Si > alors GoTo  ➔ #Recup_Info_pour_Trackloader_2 et #DF0__SIDE_UP__MOTOR_ON__DIR_EXT

1B4E    BSR     1AF2            ; sinon GoSub     ----------------------------------> #DF0__SIDE_DOWN__MOTOR_ON__DIR_EXT

1B50    BRA     1B54            ; Et ensuite GoTo ➔ #Recup_Info_pour_Trackloader_2
```

#Recup_Info_pour_Trackloader_1

```
1B52    BSR     1B10            ; GoSub ➔ #DF0__SIDE_UP__MOTOR_ON__DIR_EXT

#Recup_Info_pour_Trackloader_2
1B54    MOVE.L  D0,D3           ; A cette étape, 1er appel, D0=(LongWord de $189C) donc valeur du tableau=$189C
                                ; à savoir 'Adress_Start_raw'
                                ; D3=D0, donc D3=$189C

1B56    DIVU.W  #$189C,D0       ; D0=D0/$189C et met le résultat dans D0

1B5A    CMP.W   #56,D0          ; Compare D0 avec la valeur $56
1B5E    BLT     1B6A            ; Si plus petit que, on branche en 1B6A
1B60    SUBI.L  #84468,D3       ; Sinon, D3=D3-$84468
1B66    SUBI.W  #56,D0          ; D0=D0-$56
1B6A    MOVEQ   #0,D2           ; D2=00
1B6C    MOVE.W  D0,D2           ; D2=D0
1B6E    BSR     1A6C            ; GoSub ➔ #Position Reach?

1B72    MULU    #189C,D2        ; A ce moment, D2=$189C = position atteinte = position de la tête
                                ; On le multiplie par la taille d'une Track
                                ; il devient la taille en position_raw_atteinte
                                ; Position où l'on est, en l'occurrence : $189C

1B72    SUB.L   D2,D3           ; Cette étape D3=(LongWord de 189C)
                                  ou si D0 Start_Track était plus grand que $56
                                  on soustrait la position_raw_atteinte de D3

                                  En fait, selon l 'Adresse_Start_raw' dans le tableau, on va utiliser une face précise.
                                  Et donc cette opération effectuée sur D3 permet
                                  de repositionner l' Adresse_Start_raw tout en changeant de face.

1B78    MOVE.L  (A1),D4         ; Copie le longWord contenu à l'adresse A1 dans D4

1B7A    LSR.L   #2,D4           ; Décalage de 2 bits vers la droite de D4, revient a diviser par 4 D4
1B7C    SUBQ.L  #1,D4           ; D4=D4-1

1B7E    BSR     1B86            ; ➔ GoSub ➔ #Trackloader_Base
1B80    BSR     1A18            ;    GoSub ➔ #Move Inter.
1B84    BRA     1B7E            ; ← Branche en 1B7E, on boucle sur #Trackloader_Base
```

#Trackloader_Base

```
1B86    MOVEQ   #0F,D2              ; D2=F
1B88    MOVE.L  #18B8,D0            ; ➔ D0=$18B8
1B8E    MOVE.L  C1A(PC),A6          ; A6=Adresse dans la table=0C1A=$000018000   //Adr. mémoire pour le DSKPTH
1B92    MOVE.W  #2,DFF09C           ; Conf INTREQ, Disk Block Finished Interupt
1B9A    MOVE.L  A6,DFF020           ; Conf DSKPTH = $18000
1BA0    MOVE.W  #8210,DFF096        ; Conf DMACON, DSKEN enable, ALL DMA enable
1BA8    MOVE.W  #$4489,DFF07E       ; Conf DSKSYNC = $4489 (standard AmigaDos)
1BB0    MOVE.W  #7F00,DFF09E        ; Conf ADKCON spécifique
1BB8    MOVE.W  #B500,DFF09E        ; Encore une fois pour démarrer le transfert
                                      FAST=2us, WORDSYNC=active, MFM precomp, recomp=140ns

1BC0    MOVE.W  #4000,FF024         ; Conf DSKLEN, Write enable (ram or disk)

#Test CIA Ready
1BC8    MOVE.B  BFDD00,D7           ; D7=BFDD00
1BCE    MOVE.B  BFDD00,D7           ;       ➔ D7=icr register de CIA-B
1BD4    BTST    #4,D7               ;          Test du bit4 (FLAG) de icr
1BD8    BEQ     1BCE                ;       ➔ Interruption générée ? On boucle

1BDA    ADDI.W  #8001,D0            ; ADD signé sur D0 (qui est a $18B8 voir quelques lignes au-dessus) avec $8001
                                      ce qui nous donnes : $98B9 et flag C=0

1BDE    MOVE.W  D0,DFF024           ; CONF DSKLEN, Disk DMA Enable, Length of DMA data=$18B9
1BE4    MOVE.W  D0,DFF024           ; Encore une fois pour déclencher la lecture

1BEA    MOVE.W  DFF01E,D0           ;       ➔ D0=INTREQR
1BF0    BTST    #1,D0               ;         Test du Bit1 de INTREQR, Level 1 Disk Block Finished Interupt
1BF4    BEQ     1BEA                ;       ➔ Test ?, On boucle

1BF6    MOVE.L (A6)+, D0            ; Valeur de DSKPTH dans D0, (A6=DFF024) puis A6=A6+4
1BF8    MOVE.L (A6)+, D7            ; Valeur de DSKLEN dans D7, (A6=DFF028) puis A6=A6+4
1BFA    ADD.L   D0,D0               ; D0=D0+D0
1BFC    ANDI.L  #AAAAAAAA,D0        ; Post_Traitement MFM bit impair dans D0
1C02    ANDI.L  #55555555,D7        ; Post_Traitement MFM bit pair dans D7
1C08    OR.L    D7,D0               ; Traitement MFM
1C0A    CMP.L   #42535432,D0        ; Compare D0 avec 42535432
                                      42 53 54 32 En Ascii = BST2

1C10    BEQ     1C20                ; Oui ? 'Signature' trouvée GoSub ➔ #Traitement_MFM_BASE
                                      On commence le traitement réel des données et le Trackload

1C12    DBF     D2,1B88             ; ➔ D2=D2-1, tant que D2 est différent de -1, on boucle en 1B88
                                        C'est reparti pour un tour

1C16    MOVE.W  #F00,DFF180         ; ➔ Fond d'écran en rouge
1C1E    BRA     1C16                ; ➔ DeadLoop fond rouge
```

#Move Inter.

```
1A18    BCLR    #1,BFD100           ; CIA-B / PRB, bit1 mis à 0, DIR=vers l'intérieur
1A20    BCLR    #0,BFD100           ; CIA-B / PRB, bit0 mis à 0, STEP, pré/déplacement tête
1A28    BSET    #0,BFD100           ; CIA-B / PRB, bit0 mis à 0, STEP, déplacement tête
1A30    BSR     1ABC                ; GoSub ➔ #WAIT
1A34    MOVE.L  A0,-(A7)            ; On Sauve A0 dans la pile
1A36    LEA     C20(PC),A0          ; 00C20(PC)=A0 Adresse dans la table
1A3A    ADDQ.W  #01,(A0)            ; (A0)=A0+1 Word
                                    ; On s'est déplacé d'une Track vers l'intérieur et (A0),$C20 est incrémenté de 1
                                    ; Ici c'est donc $C20 qui est notre pointeur de Track (plus exactement Piste)

1A3C    MOVE.L  (A7)+,A0            ; On restaure A0 de la pile
1A3E    BRA     1AB0                ; GoTo ➔ #Disque Ready? sans passer par la modification de A3
```

#Position Reach?

```
1A6C    CMP.W   C20(PC),D0          ; Compare D0 avec 00C20(PC), l'adresse lue dans la table
1A70    BEQ     1A84                ; Si égale alors GoTo ➔ #Avance_Recule
1A72    BGT     1A88                ; si N=0, (plus grand que), alors GoTo ➔ #GoTo_To_Track
1A74    MOVE.W  C20(PC),D6          ; D6=$C20=Adresse du Pointeur Cylindre en cours
1A78    SUB.W   D0,D6               ; D6=D6-D0
1A7A    SUBQ.W  #1, D6              ; D6=D6-1
1A7C    BSR     1A42                ; ➔ GoSub ➔ #Move Exter.
1A7E    DBF     D6,1A7C             ; ➔ D6=D6-1, tant que D6 est différent de -1 on boucle
1A82    RTS                         ; E.T Retour maison
```

#Avance_Recule

```
1A84    BSR     1A18                ; GoSub ➔ #Move Inter.
1A86    BRA     1A42                ; GoTo ➔ #Move Exter.
```

#GoTo_To_Track

```
1A88    SUB.W   C20(PC),D0      ; D0=D0-(contenu de $C20), ak, position en cours
1A8C    SUBQ.W  #1,D0           ; D0=D0-1
1A8E    BSR     1A18            ; ➔ GoSub ➔ #Move Inter.
1A90    DBF     D0,1A8E         ; ← D0=D0-1, tant que D0 est différent de -1, on boucle
1A94    RTS                     ; E.T Retour maison
```

#DF0__SIDE_DOWN__MOTOR_ON__DIR_EXT

```
1AF2    MOVE.B  #7D,BFD100      ; Conf CIA-B / PRB
                                  MOTOR ON, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 UNSELECT, SIDE DOWN, DIR EXT., STEP TRACK=1
1AFA    NOP
1AFC    NOP
1AFE    MOVE.B  #75,BFD100      ; Conf CIA-B / PRB
                                  MOTOR ON, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 SELECT, SIDE DOWN, DIR EXT., STEP TRACK=1

1B06    MOVE.W  #B000,D7        ; D7=B000, compteur pause
1B0A    DBF     D7,1B0A         ; ←→ D7=D7-1, tant que D7 est différent de -1, on boucle, c'est une pause
1B0E    RTS                     ; E.T Retour maison
```

#DF0__SIDE_UP__MOTOR_ON__DIR_EXT

```
1B10    MOVE.B  #79,BFD100      ; Conf CIA-B / PRB
                                  MOTOR ON, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 UNSELECT, SIDE UP, DIR EXT., STEP TRACK=1
1B18    NOP
1B1A    NOP
1B1C    MOVE.B  #71,BFD100      ; Conf CIA-B / PRB
                                  MOTOR ON, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 SELECT, SIDE UP, DIR EXT., STEP TRACK=1

1B24    MOVE.W  #B000,D7        ; D7=B000, compteur pause
1B28    DBF     D7,1B28         ; ←→ D7=D7-1, tant que D7 est différent de -1, on boucle, c'est une pause
1B2C    RTS                     ; E.T Retour maison
```

#Traitement_MFM_BASE

```
1C20    MOVEQ   #0,D2           ; D2=00
1C22    MOVE.W  #626,D2         ; D2=626
1C26    TST.W   D3              ; Test contenu de D3 avec zero              // D3=Delta calculé préalablement.
1C28    BEQ     1C34            ; Si égal alors GoTo ➔ #Check_Traitement_MFM
1C2A    ADD.W   D3,D3           ; Sinon D3=D3+D3                            // on fait x2 sur D3

                                  Rappel, D3=((Valeur tableau Adresse_Start_raw)-(valeur position atteinte))

1C2C    ADDA.L  D3, A6          ; A6=A6+D3                                  // A6=DSKPTH+décodage en cour
1C2E    LSR.W   #3,D3           ; Décale de 3 Bits vers la gauche le LongWord D3// Revient a diviser par 8 D3
                                  donc delta entre position atteinte et (valeur en Length_To_Read)/8

1C30    SUB.W   D3,D2           ; D2(position_raw actuel)-Delta calculé ci-dessus
1C32    MOVEQ   #0,D3           ; D3=0                                      // On remet D3 à Zero

#Check_Traitement_MFM
1C34    CMP.L   D2,D4           ; Compare D4-D2
1C36    BGT.B   1C3E            ; si résultat plus grand que GoTo ➔ #Traitement MFM bit pair
1C38    MOVE.W  D4,D2           ; D2=D4
1C3A    ADDQ.W  #4,A7           ; A7=A7+4
1C3C    BRA     1C42            ; GoTo ➔ #Start_Decodage
```

#Traitement MFM bit pair

```
01C3E   SUB.L   D2,D4           ; D4=D4-D2
01C40   SUBQ.L  #1,D4           ; D4=D4-1

#Start_Decodage
01C42   MOVE.L  #55555555,D5    ; D5=55555555, masque MFM bit pair
01C48   MOVE.L  (A6)+,D0        ; ➔ Copie contenue de A6 dans D0 puis A6=A6+4
01C4A   MOVE.L  (A6)+,D7        ;     Copie contenue de A6 dans D7 puis A6=A6+4
01C4C   AND.L   D5,D0           ;     Traitement MFM
01C4E   AND.L   D5,D7           ;     Traitement MFM
01C50   ADD.L   D0,D0           ;     Traitement MFM
01C52   OR.L    D7,D0           ;     Traitement MFM
01C54   MOVE.L  D0,(A0)+        ;     Copie de D0 à l'adresse pointé par A0 puis A0=A0+4
01C56   DBF     D2,1C48         ; ← D2=D2-1 et tant que D2 est différente de -1, on boucle
01C5A   RTS                     ; E.T  Retour maison
```

#Decomp/Decrypt

```
                                ##Adr. de décompression = Identique à l'adresse Source du Trackload (donc A0)
1824    MOVEA.L  A0,A1          ; A1=A0=Adr. Source, on décompresse à la même adr. Mémoire que le TrackLoad
                                  On écrase donc la zone 'source'.
```

#Decomp/Decrypt_By_Adress

```
                                ##Adr. de décompression = A1
1826    MOVEQ    #0,D7          ; D7=00
1828    MOVE.L   A0,A2          ; A2=A0          A2=A0=Adr. Source
182A    MOVE.L   (A0),D0        ; Copie le 1er longWord de A0 dans D0 // (Length compressé+1 pour info)
182C    BTST     #0,D0          ; On test celui-ci
1830    BEQ      183A           ; Si=0000 on branche en $138A
1832    MOVE.L   A1,A3          ; A3=A1=Adresse Source
1834    NOT.W    7              ; Opération binaire de NOT sur le Word D7
1836    ANDI.W   #FFFE,D0       ; Opération binaire de AND entre et D0 (résultat mis dans D0)
```

#Lenght_Indicated_in_D0

```
183A    ADDA.L   D0,A0          ; Dans ce cas, on ajoute D0 a A0 (donc théoriquement, c'est la Longueur des données.
183C    MOVE.L   -(A0),(A2)     ; Copie le longword de A0 vers A2 puis A0=A0-4
183E    MOVE.L   -(A0),A2       ; Copie le longword de A0 sur A2, Donc le second LongWord contient l'information mis dans A2
1840    ADDA.L   A1,A2          ; Copie de A1 sur A2
1842    MOVE.L   -(A0),D5       ; Copie du longword de A0 dans D5 puis A0=A0-4
1844    MOVE.L   -(A0),D0       ; Copie du longword de A0 dans D0 puis A0=A0-4
1846    MOVEQ    #10,D6         ; D6=$10
1848    EOR.L    D0,D5          ; Opération binaire de EOR entre D0 et D5
184A    LSR.L    #1,D0          ; Décalage de 1 bit de D0 vers la droite
184C    BNE      1850           ; test flag
184E    BSR      18C4           ; ...
1850    BCS      1884           ; ... ...
1852    MOVEQ    #8,D1
1854    MOVEQ    #1,D3
1856    LSR.L    #1,D0
1858    BNE      185C
185A    BSR      18C4
185C    BCS      18A6
185E    MOVEQ    #3,D1
1860    MOVEQ    #0,D4
1862    BSR      18CE
1864    MOVE.W   D2,D3
1866    ADD.W    D4,D3
1868    MOVEQ    #7,D1
186A    LSR.L    #1,D0
186C    BNE      1870
186E    BSR      18C4
1870    ROXL.L   #1,D2
1872    DBF      D1,186A
1876    MOVE.B   D2,-(A2)
1878    DBF      D3,1868
187C    BRA      18B2
================================================================================
187E    MOVEQ    #8,D1
1880    MOVEQ    #8,D4
1882    BRA      1862
================================================================================
1884    MOVEQ    #2,D1
1886    BSR      18CE
1888    CMP.B    #2,D2
188C    BLT      189E
188E    CMP.B    #03,D2
1892    BEQ      187E
1894    MOVEQ    #8,D1
1896    BSR      18CE
1898    MOVE.W   D2,D3
189A    MOVEQ    #C,D1
189C    BRA      18A6
================================================================================
189E    MOVEQ    #9,D1
18A0    ADD.W    D2,D1
18A2    ADDQ.W   #2,D2
18A4    MOVE.W   D2,D3
18A6    BSR      18CE
18A8    SUBQ.L   #1,A2
18AA    MOVE.B   0(A2,D2),(A2)
18AE    DBF      D3,18A8
18B2    CMPA.L   A2,A1
18B4    BLT      184A
18B6    TST.L    D5
18B8    BNE      18E6
18BA    TST.W    D7
18BC    BEQ      8C0
18BE    BRA      18EA
================================================================================
18C0    MOVEQ    #0,D0
18C2    RTS
================================================================================
18C4    MOVE.L   -(A0),D0
18C6    EOR.L    D0,D5
18C8    MOVE.W   D6,CCR
18CA    ROXR.L   #1,D0
18CC    RTS
================================================================================
```

```
18CE     SUBQ.W   #1,D1
18D0     CLR.W    D2
18D2     LSR.L    #1,D0
18D4     BNE      18DE
18D6     MOVE.L   -(A0),D0
18D8     EOR.L    D0,D5
18DA     MOVE.W   D6,CCR
18DC     ROXR.L   #1,D0
18DE     ROXL.L   #1,D2
18E0     DBF      D1,18D2
18E4     RTS
================================================================================
18E6     MOVEQ    #FFFFFFFF,D0
18E8     RTS
================================================================================
18EA     MOVE.L   A3,A0
18EC     MOVE.L   A0,A1
18EE     MOVE.L   A0,A2
18F0     MOVE.L   (A0),D0
18F2     LSR.L    #8,D0
18F4     ADDA.L   D0,A0
18F6     MOVE.B   -(A0),(A3)+
18F8     MOVE.B   -(A0),(A3)+
18FA     MOVE.B   -(A0),(A3)
18FC     MOVEQ    #0,D1
18FE     MOVE.B   -(A0),D1
1900     LSL.W    #8,D1
1902     MOVE.B   -(A0),D1
1904     LSL.L    #8,D1
1906     MOVE.B   -(A0),D1
1908     ADDA.L   D1,A1
190A     MOVE.B   -(A0),D4
190C     MOVE.B   -(A0),D5
190E     MOVE.B   -(A0),D6
1910     MOVE.B   -(A0),D7
1912     MOVEQ    #0,D2
1914     MOVEQ    #FFFFFFFF,D3
1916     CMPA.L   A2,A1
1918     BLE      1954
191A     MOVE.B   -(A0),D0
191C     CMP.B    D0,D4
191E     BEQ      1946
1920     CMP.B    D0,D5
1922     BEQ      1930
1924     CMP.B    D0,D6
1926     BEQ      193A
1928     CMP.B    D0,D7
192A     BEQ.B    1940
192C     MOVE.B   D0,-(A1)
192E     BRA      1916
================================================================================
1930     MOVE.B   -(A0),D0
1932     MOVE.B   D0,-(A1)
1934     MOVE.B   D0,-(A1)
1936     MOVE.B   D0,-(A1)
1938     BRA      1916
================================================================================
193A     MOVE.B   D2, -(A1)
193C     MOVE.B   D2, -(A1)
193E     BRA      1916
================================================================================
1940     MOVE.B   D3,-(A1)
1942     MOVE.B   D3,-(A1)
1944     BRA      1916
================================================================================
1946     MOVEQ    #0,D0
1948     MOVE.B   -(A0),D1
194A     MOVE.B   -(A0),D0
194C     MOVE.B   D1,-(A1)
194E     DBF      D0,194C
1952     BRA      1916
================================================================================
1954     MOVEQ    #0,D0
1956     RTS
================================================================================
```

#Decomp/Decrypt_02

```
                                ##Adr. de décompression = Identique à l'adresse Source du Trackload (donc A0)
1958    MOVE.L  A0,A1           ; A1=A0=Adr. Source
195A    MOVE.L  A1,A2           ; A2=A1
195C    MOVE.L  A0,A3           ; A3=A0          Bref… tout au même endroit
195E    MOVE.L  (A0),D0         ; 1er LongWord dans D0

1960    LSR.L   #8,D0           ; Décale le résultat d'un Octet vers la droite
1962    ADDA.L  D0,A0           ; Et on ajoute le tout à A0
1964    MOVE.B  -(A0),(A3)+
1966    MOVE.B  -(A0),(A3)+
1968    MOVE.B  -(A0),(A3)
196A    MOVEQ   #0,D1
196C    MOVE.B  -(A0),D1

196E    LSL.W   #8,D1
1970    MOVE.B  -(A0),D1

1972    LSL.L   #8,D1
1974    MOVE.B  -(A0),D1
1976    ADDA.L  D1,A1
1978    MOVE.B  -(A0),D4
197A    MOVE.B  -(A0),D5
197C    MOVE.B  -(A0),D6
197E    MOVE.B  -(A0),D7
1980    MOVEQ   #0,D2
1982    MOVEQ   #FFFFFFFF,D3

#Base_Traitement
1984    CMPA.L  A2,A1
1986    BLE     19C2            ; GoTo ➔ Fin_de_routine_et_RTS
1988    MOVE.B  -(A0),D0
198A    CMP.B   D0,D4
198C    BEQ     19B4            ; GoTo ➔ #1em_Traitement
198E    CMP.B   D0,D5
1990    BEQ     199E            ; GoTo ➔ #2em_Traitement
1992    CMP.B   D0,D6
1994    BEQ     19A8            ; GoTo ➔ #3em_Traitement
1996    CMP.B   D0,D7
1998    BEQ     19AE            ; GoTo ➔ #4em_Traitement
199A    MOVE.B  D0,-(A1)
199C    BRA     1984            ; GoTo ➔ #Base_Traitement
================================================================================
                                #2em_Traitement
199E    MOVE.B  -(A0),D0
19A0    MOVE.B  D0,-(A1)
19A2    MOVE.B  D0,-(A1)
19A4    MOVE.B  D0,-(A1)
19A6    BRA     1984            ; GoTo ➔ #Base_Traitement
================================================================================
                                #3em_Traitement
19A8    MOVE.B  D2,-(A1)
19AA    MOVE.B  D2,-(A1)
19AC    BRA     1984            ; GoTo ➔ #Base_Traitement
================================================================================
                                #4em_Traitement
19AE    MOVE.B  D3,-(A1)
19B0    MOVE.B  D3,-(A1)
19B2    BRA     1984            ; GoTo ➔ #Base_Traitement
================================================================================
                                #1er_Traitement
19B4    MOVEQ   #0,D0
19B6    MOVE.B  -(A0),D1
19B8    MOVE.B  -(A0),D0
19BA    MOVE.B  D1,-(A1)        ; ➔ Boucle vers A1
19BC    DBF     D0,19BA         ; ➔
19C0    BRA     1984            ; GoTo ➔ #Base_Traitement
===========
============================================================
19C2    RTS                     ; E.T Retour maison, Fin Sous-routine decomp/decrypt
================================================================================
```

#Phase_De_Chargement_#1

```
1C5C    LEA     46FB4,A0        ; A0=46FB4                          Adr Mémoire destination
1C62    LEA     17D6.S,A1       ; A1=17D6                           Pos. Tableau            // Chargement Phase#1 01/16
1C66    BSR     1B2E            ; GoSub → #Trackloader_Start
1C6A    BSR     1824            ; GoSub → #Decomp/Decrypt

1C6E    LEA     70000,A0        ; A0=70000                          Adr Mémoire destination
1C74    LEA     17DE.S,A1       ; A1=17DE                           Pos. Tableau            // Chargement Phase#1 02/16
1C78    BSR     1B2E            ; GoSub → #Trackloader_Start
1C7C    BSR     1824            ; GoSub → #Decomp/Decrypt

1C80    LEA     45F34,A0        ; A0=45F34                          Adr Mémoire destination
1C86    LEA     17E6.S,A1       ; A1=17E6                           Pos. Tableau            // Chargement Phase#1 03/16
1C8A    BSR     1B2E            ; GoSub → #Trackloader_Start
1C8E    BSR     1824            ; GoSub → #Decomp/Decrypt

1C92    LEA     50000,A0        ; A0=50000                          Adr Mémoire destination
1C98    LEA     17EE.S,A1       ; A1=17EE                           Pos. Tableau            // Chargement Phase#1 04/16
1C9C    BSR     1B2E            ; GoSub → #Trackloader_Start
1CA0    BSR     1824            ; GoSub → #Decomp/Decrypt

1CA4    LEA     515DC,A0        ; A0=515DC                          Adr Mémoire destination
1CAA    LEA     $17F6.S,A1      ; A1=17F6                           Pos. Tableau            // Chargement Phase#1 05/16
1CAE    BSR     1B2E            ; GoSub → #Trackloader_Start
------------------------------------------------------------------------
1CB2    LEA     42000,A1        ; A1=42000
1CB8    BSR     1826            ; GoSub → #Decomp/Decrypt_By_Adress
------------------------------------------------------------------------
1CBC    LEA     50398,A0        ; A0=50398                          Adr Mémoire destination
1CC2    LEA     17FE.S,A1       ; A1=17FE                           Pos. Tableau            // Chargement Phase#1 06/16
1CC6    BSR     1B2E            ; GoSub → #Trackloader_Start
1CCA    BSR     1824            ; GoSub → #Decomp/Decrypt

1CCE    LEA     457A4,A0        ; A0=457A4                          Adr Mémoire destination
1CD4    LEA     1806.S,A1       ; A1=1806                           Pos. Tableau            // Chargement Phase#1 07/16
1CD8    BSR     1B2E            ; GoSub → #Trackloader_Start
1CDC    JMP     1824            ; GoTo → #Decomp/Decrypt            A7=$252         (A7)=1E66

                                  Un BreakPoint en $1956 suivi de la commande ST
                                  finira bien par un retour prévu en $1E66 à savoir, #Phase_De_Chargement_#2_1/2
```

#Phase_De_Chargement_#2_1/2

```
1E66    LEA     4324C,A0        ; A0=4324C                          Adr Mémoire destination
1E6C    LEA     180E.S,A1       ; A1=180E                           Pos. Tableau            // Chargement Phase#1 08/16
1E70    BSR     1B2E            ; GoSub → #Trackloader_Start

#Phase_De_Chargement_#2_1/2_bis
1E74    MOVE.L  #7C000,C1A.S    ; Copie $7C000 à l'adresse $C1A, MAJ du DSKPTH
1E7C    BSR     1D48            ; GoSub → #LoadPhase_#1_End_Part1/2
1E80    BSR     1A96            ; GoSub → #Retour_T00

1E84    LEA     641DC,A0        ; A0=641DC                          Adr Mémoire destination
1E8A    LEA     1676.S,A1       ; A1=1676                           Pos. Tableau            // Chargement Phase#1 10/16
1E8E    BSR     1B2E            ; GoSub → #Trackloader_Start
1E92    BSR     1824            ; GoSub → #Decomp/Decrypt

1E96    LEA     70400,A0        ; A0=70400                          Adr Mémoire destination
1E9C    LEA     167E.S,A1       ; A1=167E                           Pos. Tableau            // Chargement Phase#1 11/16
1EA0    BSR     1B2E            ; GoSub → #Trackloader_Start
1EA4    BSR     1824            ; GoSub → #Decomp/Decrypt


1EA8    BSR     1CE0            ; GoSub → #Donnée_déja_chargé?      si ce n'est pas le cas, continue la phase #1
                                                                    de chargement en 1CE6, 'Phase_De_Chargement_#2_2/2'

1EAC    JSR     1ACA.S          ; GoSub → #DF0__SIDE_DOWN__MOTOR_OFF__DIR_EXT
1EB0    LEA     DFF180,A0       ; A0=DFF180
1EB6    MOVEQ   #F,D0           ; D0=$F, compteur
1EB8    CLR.L   (A0)+           ; → Efface (A0)
1EBA    DBF     D0,1EB8         ; ← D0=D0-1 et tant que D0 est différent de -1, on boucle (donc 16 fois)

1EBE    BSR     5710            ; GoSub → #Decrypt_RAW
```

```
1EC2    LEA     DFF000,A6           ; A6=DFF000
1EC8    MOVE.L  #441B8,80(A6)       ; Conf copper  // COP1LCH //Conf. De l'adresse de la Copperlist(1) = $441B8
1ED0    MOVE.W  #5200,100(A6)       ; Conf BitPlan // BPLCON0
1ED6    CLR.L   102(A6)             ; Conf BitPlan // BPLCON1
1EDA    CLR.L   108(A6)             ; Conf BitPlan // BPL1MOD
1EDE    MOVE.L  #2C81F4C1,8E(A6)    ; Conf Display // DIWSTRT
1EE6    MOVE.L  #3800D0,92(A6)      ; Conf Display // DDFSTRT
1EEE    MOVEQ   #FFFFFFFF,D0        ; D0=FFFFFFFF
1EF0    MOVE.L  D0,44(A6)           ; Conf BLTAFWM=D0
1EF4    LEA     BC2.S,A0            ; A0=$BC2
1EF8    MOVE.L  #ED04,(A0)+         ; Copie ED04 en (A0) (table ?), puis A0=A0+4
1EFE    MOVE.L  #F66E,(A0)+         ; Copie F66E en (A0), puis A0=A0+4
1F04    MOVE.L  #F67C,(A0)+         ; Copie F67C en (A0), puis A0=A0+4
1F0A    MOVE.L  #F68A,(A0)+         ; Copie F68A en (A0), puis A0=A0+4
1F10    MOVE.L  #F698,(A0)+         ; Copie F698 en (A0), puis A0=A0+4
1F16    MOVE.L  #F3BE,(A0)+         ; Copie F3BE en (A0), puis A0=A0+4
1F1C    MOVE.L  #F3C2,(A0)          ; Copie F3C2 en (A0)
1F22    TST.B   BE0.S               ; Test du Byte en BE0
1F26    BEQ     1F48                ; Si égale à Zero alors GoTo #1F48
1F28    MOVEQ   #1,D0               ; D0=1
1F2A    MOVE.B  D0,F3C6             ; Copie D0 à l'adresse $F3C6
1F30    MOVE.B  D0,BC0.S            ; Copie D0 à l'adresse $BC0
1F34    MOVE.B  BBF.S,F3C3          ; Copie de BBF à l'adresse $F3C3
1F3C    JSR     EB7E                ; GoSub ➔ #EB7E
1F42    MOVE.B  #1,BBE.S            ;
1F48    MOVE.L  #794C4,42E.S        ;
1F50    BSR     9D3E                ; GoSub ➔ #ERASE_BB6
1F54    MOVE.W  #8180,DFF096        ;
1F5C    LEA     44FD2,A0            ;
1F62    JSR     4504E               ; GoSub ➔ #COLOR_TABLE_AND_CO
1F68    MOVEQ   #2,D5               ;
1F6A    MOVE.W  #12C,D0             ;
1F6E    TST.B   BB4.S               ; ➔ Test Appuie du bouton FIRE
1F72    BNE     24CC                ;    Appuyé ? alors GoTo ➔ #CHARGEMENT_LEVEL1
1F76    BSR     9D3E                ;    Sinon -----➔  GoSub ➔ #ERASE_BB6
1F7A    DBF     D0,1F6E             ; ➔ D0=D0-1 et on boucle tant que D0 est différent de -1
1F7E    LEA     23E0,A0             ;
1F84    MOVE.W  #4CD4,2(A0)         ;
1F8A    BSR     22FE                ; GoSub ➔ #BLITTER_CONF_#01
1F8E    BSR     2334                ; GoSub ➔ #BLITTER_CONF_#02
1F92    BSR     23B4                ; GoSub ➔ #START_LEVEL1_OR_NOT
1F96    MOVEQ   #28,D7              ;
1F98    TST.B   BB4.S               ; ➔ Test Appuie du bouton FIRE
1F9C    BNE     24CC                ;    Appuyé ? alors GoTo ➔ #CHARGEMENT_LEVEL1
1FA0    BSR     9D3E                ;    Sinon -----➔  GoSub ➔ #ERASE_BB6
1FA4    DBF     D7,1F98             ; ➔ D1=D1-1 et on boucle tant que D1 est différent de -1
1FA8    MOVEQ   #3C,D7              ;

1FAA    MOVE.L  DFF004,D0           ; ➔
1FB0    LSR.L   #8,D0               ;
1FB2    ANDI.W  #1FF,D0             ;
1FB6    CMP.W   #8E,D0              ;
1FBA    BLT     1FAA                ;       ➔ On boucle tant que D0 ne convient pas

1FBC    BSR     237E                ;       GoSub ➔ #BLITTER_CONF_#00
1FC0    SUBI.W  #28,2(A0)           ;
1FC6    BSR     22FE                ;       GoSub ➔ #BLITTER_CONF_#01
1FCA    BSR     2334                ;       GoSub ➔ #BLITTER_CONF_#02
1FCE    MOVE.L  DFF004,D0           ;
1FD4    LSR.L   #8,D0               ;
1FD6    ANDI.W  #1FF,D0             ;
1FDA    CMP.W   #1E,D0              ;
1FDE    BLT     1FCE                ;       ➔ On boucle tant que D0 ne convient pas
1FE0    CMP.W   #32,D0              ;
1FE4    BGT     1FCE                ;       ➔ On boucle tant que D0 ne convient pas
1FE6    TST.B   BB4.S               ;       Test Appuie du bouton FIRE
1FEA    BNE     24CC                ;       Appuyé ? alors GoTo ➔ #CHARGEMENT_LEVEL1
1FEE    DBF     D7,1FAA             ; ➔ D1=D1-1 et on boucle tant que D1 est différent de -1
1FF2    MOVEQ   #4B,D7              ;
1FF4    TST.B   BB4.S               ; ➔ Test Appuie du bouton FIRE
1FF8    BNE     24CC                ;    Appuyé ? alors GoTo ➔ #CHARGEMENT_LEVEL1
1FFC    BSR     9D3E                ;    Sinon -----➔  GoSub ➔ #ERASE_BB6
2000    DBF     D7,1FF4             ; ➔ D1=D1-1 et on boucle tant que D1 est différent de -1
2004    LEA     23F4,A0             ;
200A    MOVE.W  #491C,2(A0)         ;
2010    MOVE.W  #0400,A(A0)         ;
2016    MOVE.W  #DE1C,E(A0)         ;
201C    MOVEQ   #7,D7               ;
201E    BRA     2030                ; GoTo ➔ $2030
...
```

#Decrypt_RAW

```
5710     LEA      5754,A0          ; A0=5754
5716     LEA      6DE1C,A1         ; A1=6DE1C
571C     LEA      70400,A2         ; A2=70400
5722     MOVEQ    #D,D3            ; D3=0D

5724     MOVE.W   (A0)+,D0         ; ➔ Copie (A0) en D0 puis A0=A0+2
5726     MOVE.W   D0,D1            ;    D1=D0
5728     LSR.W    #1,D0            ;    Décale de 1 Bits vers la droite le LongWord D0// Revient à diviser par 2 D0
572A     SUBQ.W   #1,D0            ;    D0=D0-1
-------------------------------------------------
572C     MOVE.L   A2,A3            ;    A3=A2
572E     ADDA.W   D1,A3            ;    A3=A3+D1

5730     MOVE.L   A3,A4            ;    A4=A3
5732     ADDA.W   D1,A4            ;    A4=A4+D1

5734     MOVE.L   A4,A5            ;    A5=A4
5736     ADDA.W   D1,A5            ;    A5=A5+D1

5738     MOVE.L   A5,A6            ;         A6=A5
573A     ADDA.W   D1,A6            ;         A6=A6+D1
-------------------------------------------------
573C     MOVE.W   (A2)+,D2         ;       ➔ D2=(A2) puis A2=A2+2
573E     OR.W     (A3)+,D2         ;          OR de (A3) avec D2, résultat en D2 puis A3=A3+2
5740     OR.W     (A4)+,D2         ;          OR de (A4) avec D2, résultat en D2 puis A3=A3+2
5742     OR.W     (A5)+,D2         ;          OR de (A5) avec D2, résultat en D2 puis A3=A3+2
5744     OR.W     (A6)+,D2         ;          OR de (A6) avec D2, résultat en D2 puis A3=A3+2
5746     MOVE.W   D2,(A1)+         ;          Copie de D2 dans (A1) puis A1=A1+2
5748     DBF      D0,573C          ;       ➔ D0=D0-1, tant que D0 est différent de -1, on boucle
574C     MOVE.L   A6,A2            ;    A2=A6
574E     DBF      D3,5724          ; ➔ D3=D3-1, tant que D3 est différent de -1, on boucle
5752     RTS                       ; E.T retour maison
```

#Donnée_déja_chargé?

```
1CE0     TST.B    BE0.S            ; Test des Bits de $BE0, marqueur Data déjà chargé
1CE4     BEQ      1D46             ; Si égale à 0, alors GoTo ➔ 1D46 (qui est un RTS)

#Phase_De_Chargement_#2_2/2
1CE6     LEA      EB7E,A0          ; A0=EB7E                          Adr Mémoire destination
1CEC     LEA      1686.S,A1        ; A1=1686                          Pos. Tableau              // Chargement Phase#1 12/16
1CF0     BSR      1B2E             ; GoSub ➔ #Trackloader_Start
1CF4     BSR      1824             ; GoSub ➔ #Decomp/Decrypt

1CF8     LEA      F920,A0          ; A0=F920                          Adr Mémoire destination
1CFE     LEA      168E.S,A1        ; A1=168E                          Pos. Tableau              // Chargement Phase#1 13/16
1D02     BSR      1B2E             ; GoSub ➔ #Trackloader_Start
1D06     BSR      1824             ; GoSub ➔ #Decomp/Decrypt

1D0A     LEA      26488,A0         ; A0=26488                         Adr Mémoire destination
1D10     LEA      1696.S,A1        ; A1=1696                          Pos. Tableau              // Chargement Phase#1 14/16
1D14     BSR      1B2E             ; GoSub ➔ #Trackloader_Start
1D18     BSR      1824             ; GoSub ➔ #Decomp/Decrypt

1D1C     LEA      27BC2,A0         ; A0=27BC2                         Adr Mémoire destination
1D22     LEA      169E.S,A1        ; A1=169E                          Pos. Tableau              // Chargement Phase#1 15/16
1D26     BSR      1B2E             ; GoSub ➔ #Trackloader_Start
1D2A     BSR      1958             ; GoSub ➔ #Decomp/Decrypt_02

1D2E     LEA      28AB2,A0         ; A0=28AB2                         Adr Mémoire destination
1D34     LEA      16A6.S,A1        ; A1=16A6                          Pos. Tableau              // Chargement Phase#1 16/16
1D38     BSR      1B2E             ; GoSub ➔ #Trackloader_Start
1D3C     BSR      1958             ; GoSub ➔ #Decomp/Decrypt_02

1D40     MOVE.B   #1,BE1.S         ; Copie le Byte 01 à l'adresse $BE1, c'est à coup sûr un marqueur de fin de chargement
1D46     RTS                       ; E.T Retour Maison
```

#LoadPhase_#1_End_Part1/2

```
01D48    MOVE.W   #775,D0          ; D0=0775, compteur
01D4C    LEA      43C.S,A0         ; A0=$43C
01D50    CLR.B    (A0)+            ; ➔ On efface le Byte (A0)
01D52    DBF      D0,1D50          ; ← Tant que D0 est différent de -1, on boucle
-------------------------------------------------------------------------
01D56    LEA      2A8.S,A0         ; A0=2A8                          Adr Mémoire destination
01D5A    LEA      17C6.S,A1        ; A1=17C6                         Pos. Tableau            // Chargement Phase#1 09/16
01D5E    BSR      1B2E             ; GoTo ➔ #Trackloader_Start
-------------------------------------------------------------------------


=========================================================================
01D62    MOVE.W   #0,DFF180        ; Fond d'écran en noir
01D6A    BSR      1824             ; GoSub ➔ #Decomp/Decrypt

01D6E    MOVE.L   #42000,398.S     ; Update table
01D76    MOVE.B   #4D,4D7E5        ; Update table
01D7E    MOVE.W   #840,1164.S      ; Update table
01D84    MOVE.W   #520,1168.S      ; Update table
01D8A    MOVE.W   #300,116C.S      ; Update table
01D90    MOVE.W   #840,DAC.S       ; Update table
01D96    MOVE.W   #520,DB0.S       ; Update table
01D9C    MOVE.W   300,DB4.S        ; Update table

01DA2    BSR      7B2E             ; GoSub ➔ #RAZ_Tables
01DA6    BSR      5B82             ;
01DAA    LEA      44E90,A2         ;
01DB0    BSR      77AC             ;
01DB4    LEA      44B80,A0         ;
01DBA    LEA      44680, A1        ;
01DC0    MOVEQ    #13,D0           ;
01DC2    MOVE.L   (A0)+,1A(A1)     ;
01DC6    MOVE.W   (A0)+,22(A1)     ;
01DCA    MOVE.L   (A0)+,34(A1)     ;
01DCE    MOVE.L   (A0)+,38(A1)     ;
01DD2    MOVE.L   (A0)+,3C(A1)     ;
01DD6    ADDA.W   #40,A1           ;
01DDA    DBF      D0,1DC2          ;
01DDE    RTS                       ;  E.T Retour maison
```

#RAZ_Tables

```
7B2E     MOVEQ    #0,D0                   ; D0=00
7B30     MOVE.W   D0,390.S                ; RAZ Table
7B34     MOVE.W   D0,392.S                ; RAZ Table
7B38     MOVE.W   D0,394.S                ; RAZ Table
7B3C     MOVE.W   D0,396.S                ; RAZ Table
7B40     MOVE.B   D0,357.S                ; RAZ Table
7B44     MOVE.B   D0,38E.S                ; RAZ Table

7B48     MOVE.L   398.S,A0                ; A0=398
7B4C     MOVE.L   D0,(A0)                 ; RAZ (A0)
7B4E     MOVE.L   D0,4C(A0)               ; RAZ (A0)+4C
7B52     MOVE.L   D0,60(A0)               ; RAZ (A0)+60
7B56     MOVE.L   D0,74(A0)               ; RAZ (A0)+74

7B5A     LEA      35E.S,A0                ; A0=35E
7B5E     MOVEQ    #3,D1                   ; D1=03, compteur
7B60     MOVE.L   D0,(A0)+                ; ➔ RAZ (A0)
7B62     MOVE.L   (A0),A1                 ;    Copie de (A0) en A1
7B64     MOVE.L   D0,(A1)                 ;    Copie de D0 en (A1)
7B66     MOVE.L   #515DC,(A0)+            ;    Copie $515DC en (A0) puis A0=A0+4
7B6C     MOVE.L   D0,(A0)+                ;    RAZ (A0)
7B6E     DBF      D1,7B60                 ; ←  D1=D1-1, boucle tant que D1 est différent de -1

7B72     BSR      41FA                    ; GoSub ➔ #Update_Table_CDC_CEC__D2C_D38
7B76     BSR      3FCC                    ; GoSub ➔ #Update_Table_10E4_10F0__1094_10A0
7B7A     BSR      5B10                    ; GoSub ➔ #JSR(A0)_or_Erase

7B7E     CLR.B    3AB.S                   ; RAZ Table
7B82     CLR.B    357.S                   ; RAZ Table
7B86     CLR.L    358.S                   ; RAZ Table
7B8A     CLR.B    35C.S                   ; RAZ Table
7B8E     RTS                              ; E.T Retour maison
```

#Update_Table_CDC_CEC__D2C_D38

```
41FA     MOVE.W   362.S,D2C.S             ; D2C → D38
4200     MOVE.W   364.S,D30.S             ;
4206     MOVE.W   36E.S,D34.S             ;
420C     MOVE.W   370.S,D38.S             ;

4212     MOVE.W   37A.S,CDC.S             ; CDC → CEC
4218     MOVE.W   37C.S,CE0.S             ;
421E     MOVE.W   386.S,CE4.S             ;
4224     MOVE.W   388.S,CE8.S             ;
422A     RTS                              ; E.T retour maison
```

#Update_Table_10E4_10F0__1094_10A0

```
3FCC    MOVE.W   362.S,10E4.S          ; 10E4 → 10F0
3FD2    MOVE.W   364.S,10E8.S          ;
3FD8    MOVE.W   36E.S,10EC.S          ;
3FDE    MOVE.W   370.S,10F0.S          ;
3FE4    MOVE.W   37A.S,1094.S          ; 1094 → 10A0
3FEA    MOVE.W   37C.S,1098.S          ;
3FF0    MOVE.W   386.S,109C.S          ;
3FF6    MOVE.W   388.S,10A0.S          ;
3FFC    RTS                            ; E.T Retour maison
```

#JSR(A0)_or_Erase

```
5B10    MOVE.L   3A2.S,A0              ; A0=$3A2
5B14    CMPA.L   #0,A0                 ; Compare A0 avec la valeur $00000000
5B1A    BEQ      5B1E                  ; Si identique alors GoSub $5B1E
5B1C    JSR      (A0)                  ; Sinon, GoSub (A0)

5B1E    CLR.L    3A6.S                 ; On efface le LongWord en $3A6
5B22    CLR.L    3A2.S                 ; On efface le LongWord en $3A2
5B26    LEA      4D5F4,A0              ; A0=4D5F4
5B2C    MOVE.L   A0,39E.S              ; A0=39E
5B30    CLR.B    1(A0)                 ; On efface le Byte en (A0)+1
5B34    CLR.B    51(A0)                ; On efface le Byte en (A0)+51
5B38    BRA      5AC4                  ; GoTo → #Update_Table_A0_With_3A0&3E0__E58_1210_E5C_1214
```

#Update_Table_A0_With_3A0&3E0__E58_1210_E5C_1214

```
5AC4    MOVE.W   3A0.S,A0              ; Copie le Word contenu en $3A0 dans A0
5AC8    MOVE.W   A0,E58.S              ; Copie A0 à l'adresse $E58
5ACC    MOVE.W   A0,1210.S             ; Copie A0 à l'adresse $1210

5AD0    ADDA.W   #50,A0                ; A0=A0+50        // $3E0
5AD4    MOVE.W   A0,E5C.S              ; Copie A0 à l'adresse $E5C
5AD8    MOVE.W   A0,1214.S             ; Copie A0 à l'adresse $1214
5ADC    RTS                            ; E.T Retour maison
```

#EB7E_Post_Traitement

```
EB7E    MOVE.B   #1,F3C2               ;
EB86    MOVE.W   #28,F3C0              ;
EB8E    CLR.B    F3BE                  ;
EB94    BSET     #1,BFE001             ;
EB9C    LEA      F920,A0               ;
EBA2    ADDA.L   #1D8,A0               ;
EBA8    MOVE.L   #80,D0                ;
EBAE    MOVEQ    #0,D0                 ;
EBB0    MOVE.L   D1,D2                 ;
EBB2    SUBQ.W   #1,D0                 ;
EBB4    MOVE.B   (A0)+,D1              ; →
EBB6    CMP.B    D2,D1                 ;
EBB8    BGT      EBB0                  ;   GoTo → $EBB0
EBBA    DBF      D0,EBB4               ; ←
EBBE    ADDQ.B   #1,D2                 ;
EBC0    LEA      F920(PC),A0           ;
EBC4    LEA      F37C(PC),A0           ;
EBC8    ASL.L    #8,D2                 ;
EBCA    ASL.L    #2,D2                 ;
EBCC    ADDI.L   #258,D2               ;
EBD2    ADD.L    A0,D2                 ;
EBD4    MOVEQ    #E,D0                 ;
EBD6    MOVE.L   D2,(A1)+              ; →
EBD8    MOVEQ    #0,D1                 ;
EBDA    MOVE.W   2A(A0),D1             ;
EBDE    ASL.L    #1,D1                 ;
EBE0    ADD.L    D1,D2                 ;
EBE2    ADDA.L   #1E,A0                ;
EBE8    DBF      D0,EBD6               ; ←
EBEC    LEA      F37C(PC),A0           ;
EBF0    MOVEQ    #0,D0                 ;
EBF2    MOVEA.L  0(A0,D0.W),A1         ;
EBF6    CLR.L    (A1)                  ;
EBF8    ADDQ.L   #4,D0                 ;
EBFA    CMP.L    #3C,D0                ;
EC00    BNE      EBF2                  ; GoTo → $EBF2
EC02    LEA      DFF0A8,A4             ;
EC08    CLR.W    (A4)                  ;
EC0A    CLR.W    10(A4)                ;
EC0E    CLR.W    20(A4)                ;
EC12    CLR.W    30(A4)                ;
EC16    CLR.L    F36E                  ;
EC1C    CLR.L    F63A                  ;
EC22    MOVE.B   FAF6,F3B9             ;
EC2C    RTS                            ;
```

#ERASE_BB6

```
9D3E    CLR.B   BB6.S                   ;
9D42    CLR.B   BB6.S                   ;
9D46    BEQ     9D42                    ;
9D48    RTS                             ;
```

#COLOR_TABLE_AND_CO

```
4504E   MOVEA.L 42E.S,A1                ;
45052   MOVEQ   #F,D7                   ;
45054   MOVEQ   #1E,D0                  ; ➔
45056   MOVE.W  (A0)+,D1                ;       ➔
45058   MOVE.W  D1,D2                   ;
4505A   ANDI.W  #F,D2                   ;
4505E   BEQ     45062                   ;           GoTo ➔ #45062
45060   SUBQ.W  #1,D2                   ;
45062   MOVE.W  D2,D3                   ;
45064   MOVE.W  D1,D2                   ;
45066   ANDI.W  #F0,D2                  ;
4506A   BEQ     45070                   ;
4506C   SUBI.W  #10,D2                  ;
45070   OR.W    D2,D3                   ;
45072   ANDI.W  #F00,D1                 ;
45076   BEQ     4507C                   ;
45078   SUBI.W  #100,D1                 ;
4507C   OR.W    D1,D3                   ;
4507E   MOVE.W  D3,(A1)+                ;
45080   DBF     D0,45056                ;       ➔
45084   MOVEA.L A1,A0                   ;
45086   SUBA.W  #3E,A0                  ;
4508A   DBF     D7,45054                ; ➔
4508E   MOVEA.L 42E.S,A0                ;
45092   ADDA.W  #3A2,A0                 ;
45096   MOVEQ   #F,D1                   ;
45098   JSR     9D3E                    ;       ➔ GoSub ➔ #ERASE_BB6
4509E   JSR     9D3E                    ;         GoSub ➔ #ERASE_BB6
450A4   JSR     9D3E                    ;         GoSub ➔ #ERASE_BB6
450AA   JSR     9D3E                    ;         GoSub ➔ #ERASE_BB6
450B0   JSR     9D3E                    ;         GoSub ➔ #ERASE_BB6
450B6   LEA     DFF182,A1               ;
450BC   MOVEQ   #1E,D0                  ;
450BE   MOVE.W  (A0)+,(A1)+             ; ➔
450C0   DBF     D0,450BE                ; ➔
450C4   SUBA.W  #7C,A0                  ;
450C8   DBF     D1,45098                ;       ➔
480CC   RTS                             ;
```

#CHARGEMENT_LEVEL1

```
24CC    MOVE.L  #70400,C1A.S            ; Mise à jour du DSKPTH en $70400
24D4    LEA     515DC.A0                ; A0=515DC                  Adr Mémoire destination
24DA    LEA     16AE.S,A1               ; A1=16AE                   Pos. Tableau            // Chargement Level#1_01/06
24DE    BSR     1B2E                    ; GoSub ➔ #Trackloader_Start
24E2    BSR     1824                    ; GoSub ➔ #Decomp/Decrypt
24E6    LEA     4D800,A0                ; A0=4D800                  Adr Mémoire destination
24EC    LEA     16B6.S,A1               ; A0=16B6                   Pos. Tableau            // Chargement Level#1_02/06
24F0    BSR     1B2E                    ; GoSub ➔ #Trackloader_Start
24F4    BSR     1824                    ; GoSub ➔ #Decomp/Decrypt
24F8    TST.B   C0.S                    ; Test de $C0                                       Marqueur donnée déjà chargé
24FC    BNE     24CC                    ; GoTo ➔ #CHARGEMENT_LEVEL1
24FE    LEA     2AFBE,A0                ; A0=2AFBE                  Adr Mémoire destination
2504    LEA     16BE.S,A1               ; A1=16BE                   Pos. Tableau            // Chargement Level#1_03/06
2508    BSR     1B2E                    ; GoSub ➔ #Trackloader_Start
250C    BSR     1824                    ; GoSub ➔ #Decomp/Decrypt
2510    LEA     AC00,A0                 ; A0=AC00                   Adr Mémoire destination
2516    LEA     16C6.S,A1               ; A1=16C6                   Pos. Tableau            // Chargement Level#1_04/06
251A    BSR     1B2E                    ; GoSub ➔ #Decomp/Decrypt
251E    MOVE.B  #1,F3BE                 ;
2526    LEA     29654,A0                ; A0=29654                  Adr Mémoire destination
252C    LEA     16D6.S,A1               ; A1=16D6                   Pos. Tableau            // Chargement Level#1_05/06
2530    BSR     1B2E                    ; GoSub ➔ #Trackloader_Start
2534    BSR     1824                    ; GoSub ➔ #Decomp/Decrypt
2538    LEA     6DE1C,A0                ; A0=6DE1C                  Adr Mémoire destination
253E    LEA     16CE.S,A1               ; A1=16CE                   Pos. Tableau            // Chargement Level#1_06/06
2542    BSR     1B2E                    ; GoSub ➔ #Trackloader_Start
2546    BSR     1824                    ; GoSub ➔ #Decomp/Decrypt
254A    MOVEA.L C1A.S,A0                ; Récupère le DSKPTH et le met dans A0
254E    BSR     9B24                    ; GoSub ➔ #RAZ_Trackloader
...
```

#BLITTER_CONF_#00

```
0237E    MOVE.W   #9F0,40(A6)              ; A6=DFF000, donc Conf de BLTCON0
02384    CLR.W    42(A6)                   ; Conf BLTCON1
02388    CLR.W    64(A6)                   ; Conf BLTAMOD
0238C    MOVE.W   10(A0),66(A6)            ; Conf BLTDMOD
02392    MOVE.L   4(A0), 50(A6)            ; Conf BLTAPTH
02398    MOVE.L   (A0),A1                  ; A1=(A0)
0239A    MOVEQ    #4,D0                    ; Compteur en D0
0239C    MOVE.L   A1,54(A6)                ; ➔ Conf BLTDPTH
023A0    BSR      5B42                     ; GoTo ➔
023A4    MOVE.W   12(A0),58(A6)            ; Conf BLTSIZE
023AA    ADDA.W   #1F40,A1                 ; A1=A1+1F40
023AE    DBF      D0,239C                  ; ➔ D0=D0-1, on boucle tant que D0 est diffèrent de -1
023B2    RTS                               ; E.T retour maison
```

#BLITTER_CONF_#01

```
022FE    MOVE.W   #9F0,40(A6)              ; A6=DFF000, donc Conf de BLTCON0
02304    CLR.W    42(A6)                   ; Conf BLTCON1
02308    MOVE.W   10(A0),64(A6)            ; Conf BLTAMOD
0230E    CLR.W    66(A6)                   ; Conf BLTDMOD
02312    MOVE.L   4(A0),54(A6)             ; Conf BLTDPTH
02318    MOVE.L   (A0),A1                  ; A1=(A0)
0231A    MOVEQ    #4,D0                    ; Compteur en D0
0231C    MOVE.L   A1,50(A6)                ; ➔ Conf BLTAPTH
02320    BSR      5B42                     ;    GoSub ➔
02324    MOVE.W   12(A0),58(A6)            ;    Conf BLTSIZE
0232A    ADDA.W   #1F40,A1                 ;    A1=A1+1F40
0232E    DBF      D0,231C                  ; ➔ D0=D0-1, on boucle tant que D0 est diffèrent de -1
02332    RTS                               ; E.T retour maison
```

#BLITTER_CONF_#02

```
02334    MOVE.W   #FCA,40(A6)              ; A6=DFF000, donc Conf de BLTCON0
0233A    CLR.W    42(A6)                   ; Conf BLTCON1
0233E    CLR.W    64(A6)                   ; Conf BLTAMOD
02342    CLR.W    62(A6)                   ; Conf BLTBMOD
02346    MOVE.W   10(A0),60(A6)            ; Conf BLTCMOD
0234C    MOVE.W   10(A0),66(A6)            ; Conf BLTDMOD
02352    MOVE.L   8(A0),4C(A6)             ; Conf BLTBPTH
02358    MOVE.L   (A0),A1                  ; A1=(A0)
0235A    MOVEQ    #4,D0                    ; Compteur en D0
0235C    MOVE.L   C(A0),50(A6)             ; ➔ Conf BLTAPTH
02362    MOVE.L   A1,48(A6)                ;    Conf BLTCPTH
02366    MOVE.L   A1,54(A6)                ;    Conf BLTDPTH
0236A    BSR      5B42                     ;    GoSub ➔
0236E    MOVE.W   12(A0),58(A6)            ;    Conf BLTSIZE
02374    ADDA.W   #1F40,A1                 ;    A1=A1+1F40
02378    DBF      D0,235C                  ; ➔ D0=D0-1, on boucle tant que D0 est diffèrent de -1
0237C    RTS                               ; E.T retour maison
```

#START_LEVEL1_OR_NOT

```
023B4    MOVEQ    #F,D0                    ; Compteur en D0
023B6    MOVE.W   #FFF,D1                  ;
023BA    MOVE.W   D1,196(A6)               ; ➔
023BE    BSR      9D3E                     ;   GoSub ➔ #ERASE_BB6
023C2    TST.B    BB4.S                    ;   Test l'appuie sur FIRE
023C6    BNE      24CC                     ;   Appuyé ? alors GoTo ➔ #CHARGEMENT_LEVEL1
023CA    BSR      9D3E                     ;   Sinon -----➔  GoSub ➔ #ERASE_BB6
023CE    TST.B    BB4.S                    ;   Test l'appuie sur FIRE
023D2    BNE      24CC                     ;   GoTo ➔ #CHARGEMENT_LEVEL1
023D6    SUBI.W   #111, D1                 ;   D1=D1-111
023DA    DBF      D0,23BA                  ; ➔ D0=D0-1, on boucle tant que D0 est diffèrent de -1
023DE    RTS                               ; E.T retour maison
```

#RAZ_TRACKLOADER

```
09B24    MOVE.L   A0,1818.S                ; MAJ de DSKPTH à l'adresse $1818 de la table      ? Etrange de passer par A0
                                                                                                  pour faire ça…
09B28    BSR      9B40                     ; GoSub ➔ #Pre_Base_TrackLoadX
09B2C    BSR      9B8A                     ; GoSub ➔ #Motor_ON
09B30    BSR      9BBA                     ; GoSub ➔ #Retour_T00
09B34    BSR      9C24                     ; GoSub ➔ #Pre_Base_TRK_X2
09B38    BSR      9C94                     ; GoSub ➔ #Base_Conf_Interupt
09B3C    BRA      9CDC                     ; GoTo ➔ #Traitement_Trait_01
```

#Pre_Base_TrackLoadX

```
09B40    MOVE.W   DFF01C,181C.S          ; Update_Table_X
09B48    MOVE.W   DFF01E,181E.S          ; ...
09B50    MOVE.W   DFF002,1820.S          ; ... ...
09B58    MOVE.W   DFF010,1822.S          ; ... ... ...

09B60    MOVE.W   #10,DFF096             ; Conf DMACON
09B68    MOVE.W   #2,DFF09C              ; Conf INTREQ
09B70    MOVE.W   #7FFF,DFF09E           ; Conf ADKCON
09B78    MOVE.W   #8100,DFF09E           ; Conf ADKCON
09B80    ORI.B    #$78,BFD100            ; 'Ou' binaire entre 0111 1000 et  $BFD100, donc DF0 à DF3 select et Motor ON
09B88    RTS                             ; E.T Retour maison
```

#Motor_ON

```
09B8A    BCLR     #7,BFD100              ; CIA-B / PRB, bit7 mis à 0, motor On
09B92    MOVEQ    #64,D0                 ; D0=64, compteur
09B94    BCLR     #3,$BFD100             ; ➔ CIA-B, bit3 à 0, DF0 sélectionné
09B9C    DBF      D0,9B94                ; ← D0=D0-1, tant que D1 est différent de -1, on boucle
09BA0    BSR      9BA6                   ; GoSub ➔ #Floppy_Ready?
09BA4    RTS                             ; E.T Retour maison
```

#Floppy_Ready?

```
09BA6    MOVE.W   #BB8, D6               ; D6=BB8, compteur
09BAA    DBF      D6,9BAA                ; ←➔ D6=D6-1, tant que D6 est différent de -1, on boucle. (C'est une pause)
09BAE    BTST     #5,BFE001              ; ➔ Test de Bit 5 de BFE001=Floppy Ready?
09BB6    BNE      9BAE                   ; ← tant que lecteur n'est pas prêt, on boucle
09BB8    RTS                             ; E.T Retour maison
```

#Retour_T00

```
09BBA    BTST     #4,BFE001              ; CIA-A / PRA, On test le bit4, Tête sur T00 ?
09BC2    BEQ      9BE0                   ; Si égale à Zero Alors GoTo $9BE0 qui fait un RTS
09BC4    BSET     #1,BFD100              ; CIA-B / PRB, bit1 mis à 0, DIR=vers l'intérieur
09BCC    BCLR     #0,BFD100              ; CIA-B / PRB, bit0 mis à 0, STEP, pré-déplacement tête
09BD4    BSET     #0,BFD100              ; CIA-B / PRB, bit0 mis à 0, STEP, déplacement tête
09BDC    BSR      9BA6                   ; GoTo ➔ #Floppy_Ready?
09BDE    BRA      9BBA                   ; On boucle, GoSub ➔ #Move Exter.
```

```
09BE0    RTS
```

#Retour_Maison

```
09BE2    BTST     #4,BFE001              ; CIA-A / PRA, On test le bit4, Tête sur T00 ?
09BEA    BEQ      9BF0                   ; GoTo ➔ #GoTo_Position_base
09BEC    BSR      9BBA                   ; GoSub ➔ #Retour_T00
09BEE    BRA      9BE2                   ; On Boucle, GoTo ➔ #Retour_Maison
```

#GoTo_Position_base

```
09BF0    MOVE.W #0,D2                    ; D2=0

#GoTo_Position_delta_D2
09BF4    TST.W    D2                     ; Test du Word D2
09BF6    BEQ      9C22                   ; Si égale à Zero Alors GoTo $9C22 qui fait un RTS
09BF8    BCLR     #1,BFD100              ; CIA-B / PRB, bit1 mis à 0, DIR=vers l'intérieur
09C00    NOP                             ;
09C02    NOP                             ;
09C04    NOP                             ;
09C06    BCLR     #0,BFD100              ; CIA-B / PRB, bit0 mis à 0, STEP, pré/déplacement tête
09C0E    NOP                             ;
09C10    NOP                             ;
09C12    NOP                             ;
09C14    BSET     #0,BFD100              ; CIA-B / PRB, bit0 mis à 0, STEP, déplacement tête
09C1C    BSR      9BA6                   ; GoSub ➔ #Floppy_Ready?
09C1E    SUBQ.W   #1,D2                  ; D2=D2-1
09C20    BRA      9BF4                   ; GoTo ➔ #GoTo_Position_delta_D2
```

```
09C22    RTS                             ; E.T Retour maison
```

#Pre_Base_TRK_X2

```
09C24    BSR      9BE2                        ; → GoSub → #Retour_Maison

#Pre_Base_TRK_X2_bis
09C26    BCLR     #2,$00BFD100                ; CIA-B / PRB, bit2 mis à 0, /SIDE=0, Side Upper
09C2E    MOVE.W   #8210,DFF096                ; Conf DMACON, DSKEN enable, ALL DMA enable
09C36    MOVE.L   1818.S,DFF020               ; DSKPTH=dans la table $1818
09C3E    CLR.W    DFF024                      ; Nettoie DSKLEN
09C44    BSR      9C80                        ; GoSub → #ICR_TEST
09C48    MOVE.W   #4000,DFF024                ; Conf DSKLEN, Write enable (ram or disk)
09C50    MOVE.W   #B778,DFF024                ; Conf DSKLEN, Disk DMA enable, Write disable, DSKLEN=3778
09C58    MOVE.W   #B778,DFF024                ; Conf DSKLEN, Disk DMA enable, Write disable, DSKLEN=3778
09C60    MOVE.L   #30D40,D1                   ; D1=30D40

#INTREQ_TEST
09C66    MOVE.W   DFF01E,D0                   ; D0=INTREQR
09C6C    SUBQ.L   #1,D1                       ; D1=D1-1
09C6E    BEQ      9C26                        ; si Flag Z=1 alors on repart pour un tour, GoTo → #Pre_Base_TRK_X2_bis
09C70    BTST     #1,D0                       ; Test du Bit 1 de D0
09C74    BEQ      9C66                        ; Si égale à Zero, on boucle GoTo → #INTREQ_TEST
09C76    MOVE.W   #2,DFF09C                   ; Conf INTREQ, Clear DSKBLK (Disk Block Finished Interupt)
09C7E    RTS                                  ; E.T Retour maison
```

#ICR_TEST

```
09C80    MOVE.B   BFDD00,D6                   ; D6=CIA-B ICR
09C86    MOVE.B   BFDD00,D6                   ; →  D6=CIA-B ICR
09C8C    BTST     #4,D6                       ;    test bit 4 du ICR
09C90    BEQ      9C86                        ; ← on boucle si nécessaire
09C92    RTS                                  ; E.T Retour maison
```

#Base_Conf_Interupt

```
09C94    BSET     #7,BFD100                   ; CIA-B PRB  // Motor Off
09C9C    BSET     #3,BFD100                   ; CIA-B SEL0 // Unselect DF0
09CA4    BCLR     #3,BFD100                   ; CIA-B SEL0 // select DF0
09CAC    MOVE.W   181C.S,D0                   ; D0=DFF01C
09CB0    BSET     #F,D0                       ; Conf. INTENAR
09CB4    MOVE.W   D0,DFF09A                   ; Conf. INTENA
09CBA    MOVE.W   1820.S,D0                   ; Conf. INTENAR
09CBE    BSET     #F,D0                       ; Conf. INTENAR
09CC2    MOVE.W   D0,DFF096                   ; Conf. DMACON
09CC8    MOVE.W   1822.S,D0                   ; Conf. INTENAR
09CCC    BSET     #F,D0                       ; Conf. INTENAR
09CD0    MOVE.W   D0,DFF09E                   ; Conf ADKCON
09CD6    CLR.W    C20.S                       ; On efface le Word en C20
09CDA    RTS                                  ; E.T Retour maison
```

#Traitement_Trait_01

```
09CDC    MOVE.L   1818.S,A0                   ; A0=1818
09CE0    MOVE.L   #3778,D3                    ; Compteur en D3
09CE6    MOVE.L   (A0),D0                     ; →
09CE8    ADDQ.L   #2,A0                       ; A0=A0+2
09CEA    MOVE.L   #F,D2                       ; Compteur en D2
09CF0    MOVE.L   D0,D1                       ;                → 
09CF2    SWAP     D1                          ;                Inverse en longword D1
09CF4    CMPI.W   #4454, D1                   ;                On compare avec 4454 (Word de synchro Custom?)
09CF8    BEQ      9D08                        ;                Si identique alors on GoTo → #Traitement_Trait_02
09CFA    ADD.L    D0,D0                       ;                D0=D0+D0
09CFC    DBF      D2,9CF0                     ;                ← D2=D2-1, on boucle tant que D2 est diffèrent de -1
09D00    DBF      D3,9CE6                     ; ← D3=D3-1, on boucle tant que D3 est diffèrent de -1
09D04    BRA      9D38                        ; GoTo → #SET_D0_To_1
```

#Traitement_Trait_02

```
09D08    MOVEQ    #0,D5                       ;
09D0A    MOVE.L   (A0),D0                     ; → D0=(A0)
09D0C    ADDQ.L   #2,A0                       ;   A0=A0+2
09D0E    MOVE.L   #F,D2                       ;   Compteur en D2
09D14    MOVE.L   D0,D1                       ;                →
09D16    SWAP     D1                          ;                 Inverse en longword D1
09D18    CMPI.W   #4454,D1                    ;                On compare avec 4454 (Word de synchro Custom?)
09D1C    BEQ      9D2C                        ;                Si identique alors on GoTo → #Traitement_Trait_03
09D1E    ADD.L    D0,D0                       ;                D0=D0+D0
09D20    DBF      D2,9D14                     ;                ← D2=D2-1, on boucle tant que D2 est diffèrent de -1
09D24    ADDQ.L   #1,D5                       ;    D5=D5+1
09D26    DBF      D3,9D0A                     ; ← D3=D3-1, on boucle tant que D3 est diffèrent de -1
09D2A    BRA      9D38                        ; GoTo → #SET_D0_To_1
```

#Traitement_Trait_03

```
09D2C    SUBI.L   #1A2C,D5                    ; D5=D5-1A2C
09D32    BMI      9D38                        ; Si résultat négatif, alors GoTo → #SET_D0_To_1
09D34    CLR.W    D0                          ; Sinon On efface le Word D0
09D36    RTS                                  ; E.T Retour Maison
```

#SET_D0_To_1

```
09D38    MOVE.W   #1,D0                          ; D0=1
09D3C    RTS                                     ; E.T Retour Maison
```

Ce qui finalement nous donne la table suivante des appels pour arriver au **menu** principal.

#Table_#02

Ordre D'appel	Adr. D'appel	A0 Adr. mémoire	A1 Pos. Table	DSKPTH	(A1) LENGTH	Adr. Zone de Chargement en mémoire	Decomp/ Decrypt	NFO	Adr. Zone de Décomp en mémoire	LENGTH Décomp
00	8CF8	000B0	8D16	7C100/18000	00004	000B0-000B4	NON	Check Signature Disk	N/A	N/A
01	1C66	46FB4	17D6	18000	03A90	46FB4-4AA44	BSR 1824	Load_Menu_01/16	46FB4-4D800	0684C
02	1C78	70000	17DE	18000	00250	70000-70250	BSR 1824	Load_Menu_02/16	70000-70390	00390
03	1C8A	45F34	17E6	18000	00438	45F34-4636C	BSR 1824	Load_Menu_03/16	45F34-46AD4	00BA0
04	1C9C	50000	17EE	18000	00124	50000-50124	BSR 1824	Load_Menu_04/16	50000-5019C	0019C
05	1CAE	515DC	17F6	18000	00664	515DC-51C40	BSR 1826	Load_Menu_05/16	42000-4324B	0124B
06	1CC6	50398	17FE	18000	00268	50398-50600	BSR 1824	Load_Menu_06/16	50398-509D7	0063F
07	1CD8	457A4	1806	18000	003FC	457A4-45BA0	BSR 1824	Load_Menu_07/16	457A4-45F34	00790
08	1E70	4324C	180E	18000	01EBC	4324C-45108	NON	Load_Menu_08/16	N/A	N/A
09	1D5E	002A8	17C6	7C000	00060	002A8-00308	BSR 1824	Load_Menu_09/16	002A8-0043C	00194
10	1E8E	641DC	1676	7C000	04C18	641DC-68DF4	BSR 1824	Load_Menu_10/16	641DC-6DE1C	9C40
11	1EA0	70400	167E	7C000	0271C	70400-72B1C	BSR 1824	Load_Menu_11/16	70400-794C4	090C4
12	1CF0	0EB7E	1686	7C000	007FC	0EB7E-0F37A	BSR 1824	Load_Menu_12/16	0EB7E-0F920	00DA2
13	1D02	0F920	168E	7C000	0E9D8	0F920-1F240	BSR 1824	Load_Menu_13/16	0F920-26488	16B68
14	1D14	26488	1696	7C000	01408	26488-27890	BSR 1824	Load_Menu_14/16	26488-27BC2	0173A
15	1D26	27BC2	169E	7C000	00E10	27BC2-289D2	BSR 1958	Load_Menu_15/16	27BC2-28AB2	00EF0
16	1D38	28AB2	16A6	7C000	00B48	28AB2-295FA	BSR 1958	Load_Menu_16/16	28AB2-29654	00BA2

Sans oublier le tableau suivant pour le chargement suivant, à savoir le TrackLoad du **Level #01**

#Table_#03

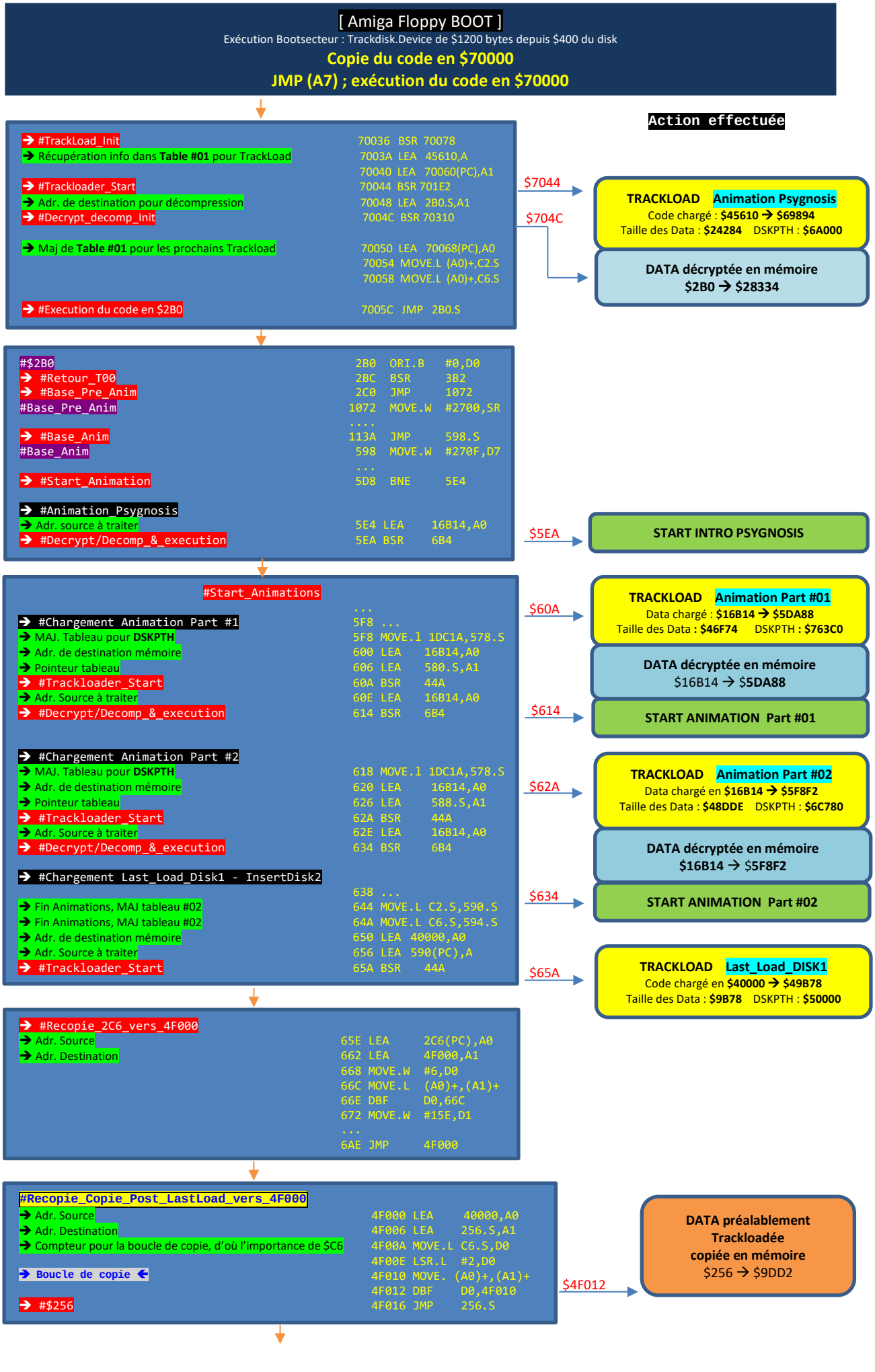
Ordre D'appel	Adr. D'appel	A0 Adr. mémoire	A1 Pos. Table	DSKPTH	(A1) LENGTH	Adr. Zone de Chargement en mémoire	Decomp/ Decrypt	NFO	Adr. Zone de Décomp. en mémoire	LENGTH Décomp
17	24DE	515DC	16AE	70400	0CAA8	515DC-5E084	BSR 1824	Level#1_01/06	515DC-641DC	12C00
18	24F0	4D800	16B6	70400	00E84	4D800-4E684	BSR 1824	Level#1_02/06	4D800-50000	02800
19	2508	2AFBE	16BE	70400	0DDC8	2AFBE-38D86	BSR 1824	Level#1_03/06	2AFBE-42000	17042
20	251A	0AC00	16C6	70400	03C28	0AC00-0E828	Non	Level#1_04/06	N/A	N/A
21	2530	29654	16D6	70400	01178	29654-2A7CC	BSR 1824	Level#1_05/06	29654-2AFBE	0196A
22	2542	6DE1C	16CE	70400	003A0	6DE1C-6E1BC	BSR 1824	Level#1_06/06	6DE1C-6FD9C	01E80

On notera aussi qu'il existe d'origine un Compteur de Track positionné en mémoire **$C20**
/! Il serait intéressant de s'attarder un peu plus tard dessus pour voir si il n'est pas utilisé pendant le jeu Hors la routine interne de TrackLoad.
D'ailleurs pourquoi attendre ? Il suffit d'entre dans l'AR une fois le Level chargé et de regarder.

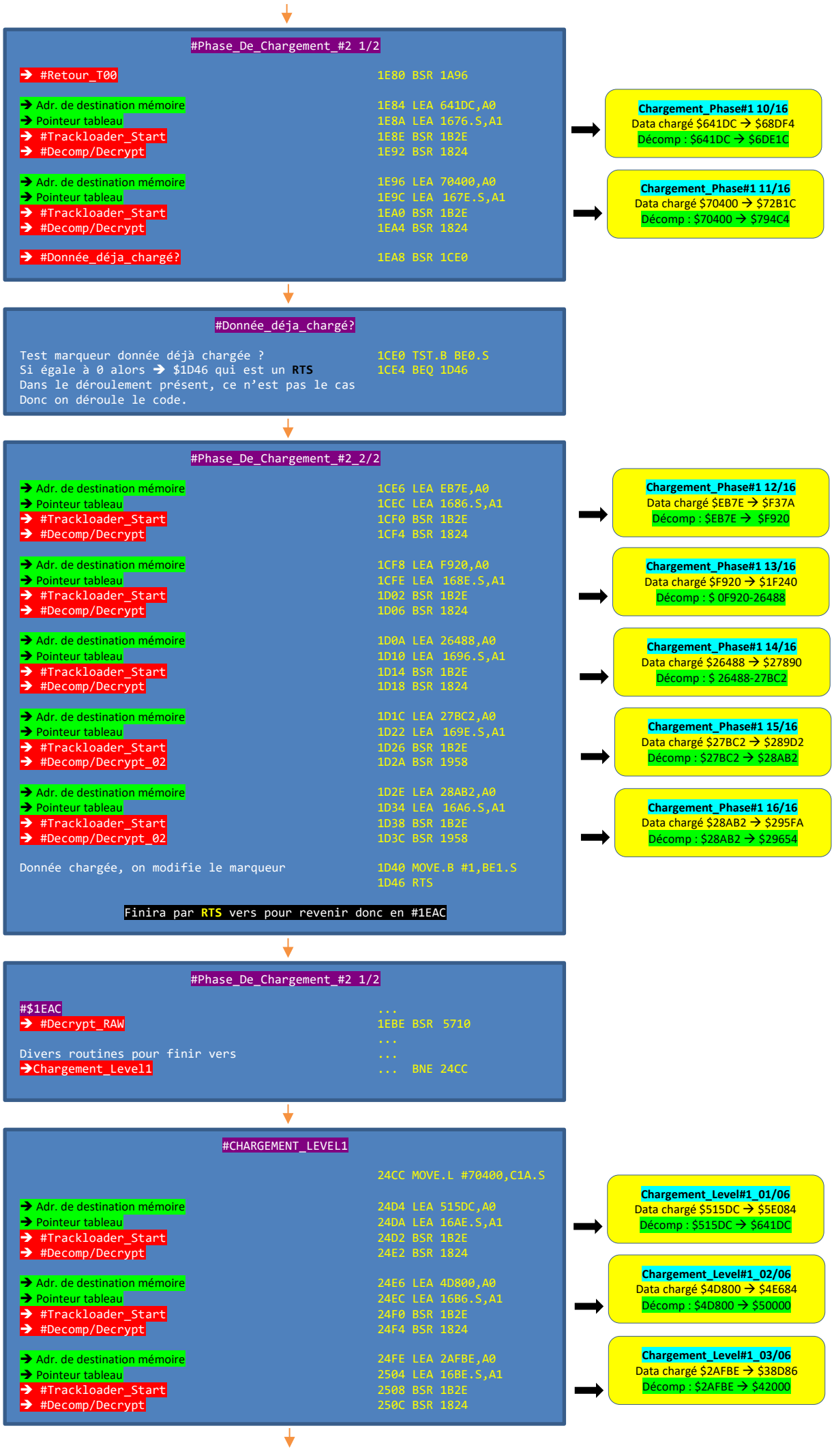
```
f 0C 20
Search from: 000000 to: 080000
006C5B  006C69  00716A  007236  007FD2  009CD8  014052  014272  014352  0148B6
0148F6  014936  014976  014CB6  014CF6  014D36  014D76  0150B6  0150F6  015136
015176  02F902  03014E  030550  03096A  030D60  031162  03157C  041124  045B48
046833  0468F3  046F14  047C8A  04A6CA  05A1E9  05D0EF
Ready.
d 716A-4
~007166 CMPI.W  #A,00000C20.S
~00716C BNE     0000717E
~00716E BSR     00007A90
~007172 JSR     00001ACA.S

d 7236-4
~007232 CMPI.W  #35,00000C20.S
~007238 BNE     0000724A
~00723A BSR     00007A90
```

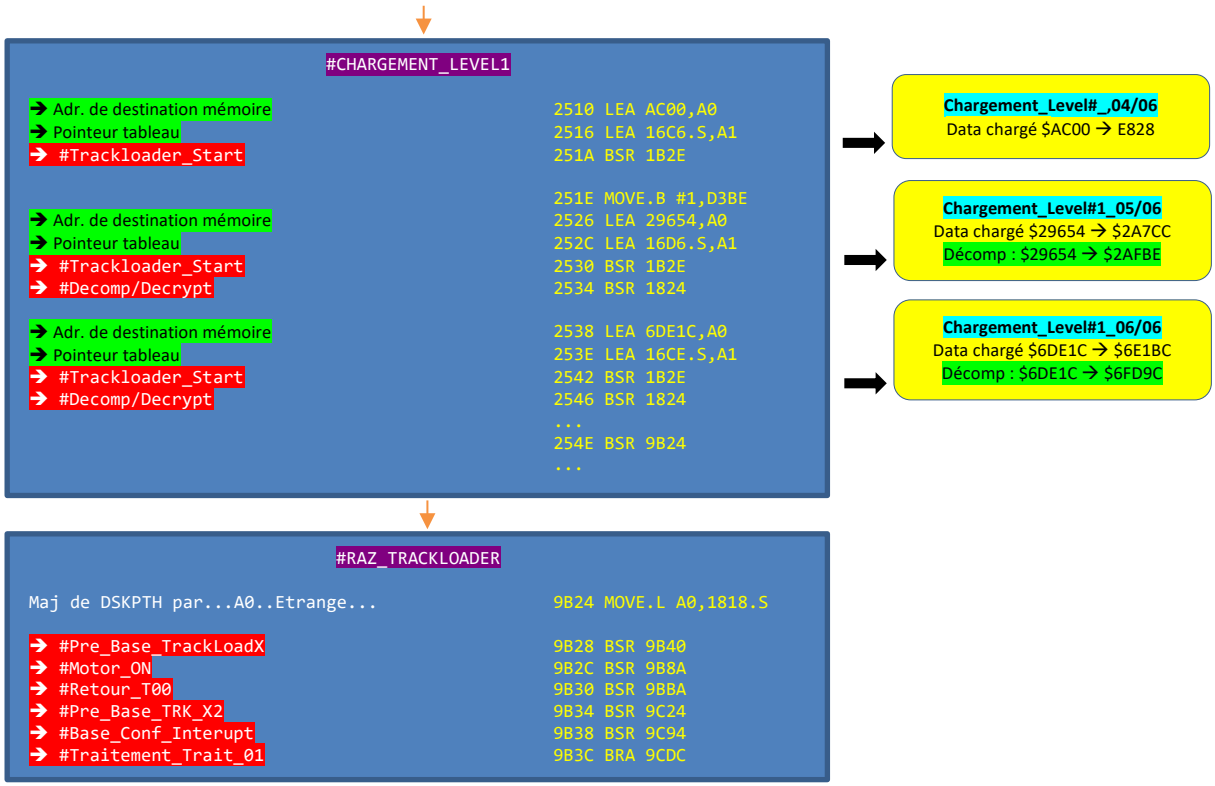
Et effectivement, le compteur semble bien utilisé en dehors des routines de Trackload, il va falloir en tenir compte et le recréer si nécessaire.

Part 10 Schéma Part #01 (ak : l'organigramme qui fait peur...)




#$256
→ #Base_LastLoad
256 BRA 1DE0
#Base_LastLoad
1DE0 MOVE.B C0.S,D0
...
→ #Trackloader_2_Init
1E48 JSR 19C4.S
→ #Retour_T00
1E4C JSR 1A96.S
→ #BASE_INSERT_DISK_ASKED
1E50 JSR 8C30
#CHECK_DISK
DISK 2 insérée ? // Fire appuyé
NON
OUI
→ #Wait_BB6
1E56 BSR 9D3E
→ #Phase_De_Chargement_#1
1E62 BSR 1C5C
ZONE DE TRACKLOAD
#Phase_De_Chargement_#1
→ Adr. de destination mémoire
→ Pointeur tableau
→ #Trackloader_Start
→ #Decomp/Decrypt
1C5C LEA 46FB4,A0
1C62 LEA 17D6.S,A1
1C66 BSR 1B2E
1C6A BSR 1824
Chargement_Phase#1 01/16
Data chargé $46FB4 → $4AA4A
Décomp : $46FB4 → $4D800
→ Adr. de destination mémoire
→ Pointeur tableau
→ #Trackloader_Start
→ #Decomp/Decrypt
1C6E LEA 70000,A0
1C74 LEA 17DE.S,A1
1C78 BSR 1B2E
1C7C BSR 1824
Chargement_Phase#1 02/16
Data chargé $70000→ $70250
Décomp : $70000 → $70390
→ Adr. de destination mémoire
→ Pointeur tableau
→ #Trackloader_Start
→ #Decomp/Decrypt
1C80 LEA 45F34,A0
1C86 LEA 17E6.S,A1
1C8A BSR 1B2E
1C8E BSR 1824
Chargement_Phase#1 03/16
Data chargé $45F34→ $4636C
Décomp : $45F34 → $46AD4
→ Adr. de destination mémoire
→ Pointeur tableau
→ #Trackloader_Start
→ #Decomp/Decrypt
1C95 LEA 50000,A0
1C98 LEA 17EE.S,A1
1C9C BSR 1B2E
1CA0 BSR 1824
Chargement_Phase#1 04/16
Data chargé $50000 → $50124
Décomp : $50000 → $5019C
→ Adr. de destination mémoire
→ Pointeur tableau
→ #Trackloader_Start
→ Adr. de destination mémoire pour décomp/décrypt
→ #Decomp/Decrypt_By_Adress
1CA4 LEA 515DC,A0
1CAA LEA 17F6.S,A1
1CAE BSR 1B2E
1CB2 LEA 42000,A1
1CB8 BSR 1826
Chargement_Phase#1 05/16
Data chargé $515DC → 51C40
Décomp : $42000 → $4324B
→ Adr. de destination mémoire
→ Pointeur tableau
→ #Trackloader_Start
→ #Decomp/Decrypt
1CBC LEA 50398,A0
1CC2 LEA 17FE.S,A1
1CC6 BSR 1B2E
1CCA BSR 1824
Chargement_Phase#1 06/16
Data chargé $50398 → 50600
Décomp : $50398 → $509D7
→ Adr. de destination mémoire
→ Pointeur tableau
→ #Trackloader_Start
→ #Decomp/Decrypt
1CCE LEA 457A4,A0
1CD4 LEA 1806.S,A1
1CD8 BSR 1B2E
1CDC JMP 1824
Chargement_Phase#1 07/16
Data chargé $457A4 → 45BA0
Décomp : $457A4 → $45F34
Finira par un RTS pour revenir en $E166
#Phase_De_Chargement_#2 1/2
→ Adr. de destination mémoire
→ Pointeur tableau
→ #Trackloader_Start
1E66 LEA 434C,A0
1E6C LEA 180E.S,A1
1E70 BSR 1B2E
Chargement_Phase#1 08/16
Data chargé $4324C → $45108
1E74 MOVE.L #7C000,C1A.S
→ #LoadPhase_#1_End_Part1/2
1E7C BSR 1D48
#LoadPhase_#1_End_Part1/2
#$1D48
1D48 MOVE.W #775,D0
...
→ Adr. de destination mémoire
→ Pointeur tableau
→ #Trackloader_Start
1D56 LEA 2A8.S,A0
1D5A LEA 17C6.S,A
1D5E BSR 1B2E
...
→ #Decomp/Decrypt
1D6A BSR 1824
Chargement_Phase#1 09/16
Data chargé $2A8 → $308
Décomp : $2A8 → $43C
Finira par RTS vers pour revenir donc en #1E80


#Phase_De_Chargement_#2 1/2
→ #Retour_T00 1E80 BSR 1A96
→ Adr. de destination mémoire 1E84 LEA 641DC,A0
→ Pointeur tableau 1E8A LEA 1676.S,A1
→ #Trackloader_Start 1E8E BSR 1B2E
→ #Decomp/Decrypt 1E92 BSR 1824
→ Adr. de destination mémoire 1E96 LEA 70400,A0
→ Pointeur tableau 1E9C LEA 167E.S,A1
→ #Trackloader_Start 1EA0 BSR 1B2E
→ #Decomp/Decrypt 1EA4 BSR 1824
→ #Donnée_déja_chargé? 1EA8 BSR 1CE0
Chargement_Phase#1 10/16
Data chargé $641DC → $68DF4
Décomp : $641DC → $6DE1C
Chargement_Phase#1 11/16
Data chargé $70400 → $72B1C
Décomp : $70400 → $794C4
#Donnée_déja_chargé?
Test marqueur donnée déjà chargée ? 1CE0 TST.B BE0.S
Si égale à 0 alors → $1D46 qui est un RTS 1CE4 BEQ 1D46
Dans le déroulement présent, ce n'est pas le cas
Donc on déroule le code.
#Phase_De_Chargement_#2_2/2
→ Adr. de destination mémoire 1CE6 LEA EB7E,A0
→ Pointeur tableau 1CEC LEA 1686.S,A1
→ #Trackloader_Start 1CF0 BSR 1B2E
→ #Decomp/Decrypt 1CF4 BSR 1824
→ Adr. de destination mémoire 1CF8 LEA F920,A0
→ Pointeur tableau 1CFE LEA 168E.S,A1
→ #Trackloader_Start 1D02 BSR 1B2E
→ #Decomp/Decrypt 1D06 BSR 1824
→ Adr. de destination mémoire 1D0A LEA 26488,A0
→ Pointeur tableau 1D10 LEA 1696.S,A1
→ #Trackloader_Start 1D14 BSR 1B2E
→ #Decomp/Decrypt 1D18 BSR 1824
→ Adr. de destination mémoire 1D1C LEA 27BC2,A0
→ Pointeur tableau 1D22 LEA 169E.S,A1
→ #Trackloader_Start 1D26 BSR 1B2E
→ #Decomp/Decrypt_02 1D2A BSR 1958
→ Adr. de destination mémoire 1D2E LEA 28AB2,A0
→ Pointeur tableau 1D34 LEA 16A6.S,A1
→ #Trackloader_Start 1D38 BSR 1B2E
→ #Decomp/Decrypt_02 1D3C BSR 1958
Donnée chargée, on modifie le marqueur 1D40 MOVE.B #1,BE1.S
1D46 RTS
Finira par RTS vers pour revenir donc en #1EAC
Chargement_Phase#1 12/16
Data chargé $EB7E → $F37A
Décomp : $EB7E → $F920
Chargement_Phase#1 13/16
Data chargé $F920 → $1F240
Décomp : $ 0F920-26488
Chargement_Phase#1 14/16
Data chargé $26488 → $27890
Décomp : $ 26488-27BC2
Chargement_Phase#1 15/16
Data chargé $27BC2 → $289D2
Décomp : $27BC2 → $28AB2
Chargement_Phase#1 16/16
Data chargé $28AB2 → $295FA
Décomp : $28AB2 → $29654
#Phase_De_Chargement_#2 1/2
#$1EAC ...
→ #Decrypt_RAW 1EBE BSR 5710
...
Divers routines pour finir vers ...
→Chargement_Level1 ... BNE 24CC
#CHARGEMENT_LEVEL1
24CC MOVE.L #70400,C1A.S
→ Adr. de destination mémoire 24D4 LEA 515DC,A0
→ Pointeur tableau 24DA LEA 16AE.S,A1
→ #Trackloader_Start 24D2 BSR 1B2E
→ #Decomp/Decrypt 24E2 BSR 1824
→ Adr. de destination mémoire 24E6 LEA 4D800,A0
→ Pointeur tableau 24EC LEA 16B6.S,A1
→ #Trackloader_Start 24F0 BSR 1B2E
→ #Decomp/Decrypt 24F4 BSR 1824
→ Adr. de destination mémoire 24FE LEA 2AFBE,A0
→ Pointeur tableau 2504 LEA 16BE.S,A1
→ #Trackloader_Start 2508 BSR 1B2E
→ #Decomp/Decrypt 250C BSR 1824
Chargement_Level#1_01/06
Data chargé $515DC → $5E084
Décomp : $515DC → $641DC
Chargement_Level#1_02/06
Data chargé $4D800 → $4E684
Décomp : $4D800 → $50000
Chargement_Level#1_03/06
Data chargé $2AFBE → $38D86
Décomp : $2AFBE → $42000



Part 11 Appel disque depuis le jeu. (ak : Le tableau de la mort)

Bravo à vous si vous êtes encore là et que vous avez bien tout suivi, ce qui n'est pas forcément évident au vue du nombre d'informations.

Alors… plusieurs façons de procéder pour lister tous les appels au TrackLoader et les ripper.
La 1er est d'utiliser la commande **FA 1B2E** et de remplir en conséquence une table en analysant le code autour de ces appels.
La seconde, utilisée ci-dessous, c'est d'utiliser les *Cheat-Modes* (Energie et money) pour finir le jeu le tout avec des *BreakPoints* bien placé.
à savoir en appel du TrackLoader : **$1B2E**
En sortie du TrackLoader : **$1B36**

Et à chaque appel, regarder les valeurs des registres **A0** et **A1**
Le Program *Counter* (***PC***)
La valeur en mémoire de l'adresse demandée.
Les instructions qui précédent et suivent l'appel à *1B2E*

TIPS : Les breakpoints vont interférer avec la routine de changement de disk pendant le jeu (qui arrive 2 ou 3 fois)
Dans ce cas, il faudra insérer le disk demandé dans le lecteur puis entrer dans l'AR et sauter directement vers le retour au code.

Si c'est le **disk1** demandé ➔ tapez : **G8D14**
Si c'est le **disk2** demandé ➔ tapez : **G8D0A**

Plus d'info, voir le code en question plus haut part 9 du Tuto

Vous devriez retrouver le tableau suivant :

Disk2

Ordre D'appel	Disk Side	Adr. D'appel	A0 Adr. mém	A1 Pos. Table	DSKPTH	(A1)+ LENGTH	(A1) Pos. Disk Réel	Adr. Zone de Chargement en mémoire	Decomp/ Decrypt	Adr. Zone de Décomp. en mémoire	LENGTH Décomp	NFO
Load_Menu_10	D2-LOW	1E8E	641DC	1676	7C000	04C18	00000-04C17	641DC-68DF4	BSR 1824	641DC-6DE1C	9C40	Load_Menu_10/16 // Return_Menu_02/08
00	D2-LOW	8CF8	000B0	8D16	7C100 8000	00004	00000-00004	000B0-000B4	Non	N/A	N/A	Check Signature Disk
Load_Menu_11	D2-LOW	1EA0	70400	167E	7C000	0271C	04C18-07333	70400-72B1C	BSR 1824	70400-794C4	090C4	Load_Menu_11/16, Return_Menu_03/08
Load_Menu_12 In_Game_50	D2-LOW	1CF0	0EB7E	1686	7C100 7C000	007FC	07334-07B2F	0EB7E-0F37A	BSR 1824	0EB7E-0F920	00DA2	Load_Menu_12/16, Retour_CRYSTAL_CAVERNS_08/12 Return_Menu_04/08
Load_Menu_13 In_Game_51	D2-LOW	1D02	0F920	168E	7C100 7C000	0E9D8	07B30-16507	0F920-1F240	BSR 1824	0F920-26488	16B68	Load_Menu_13/16, Retour_CRYSTAL_CAVERNS_09/12 Return_Menu_05/08
Load_Menu_14 In_Game_52	D2-LOW	1D14	26488	1696	7C100 7C000	01408	16508-1790F	26488-27890	BSR 1824	26488-27BC2	0173A	Load_Menu_14/16, Retour_CRYSTAL_CAVERNS_10/12 Return_Menu_06/08
Load_Menu_15 In_Game_53	D2-LOW	1D26	27BC2	169E	7C100 7C000	00E10	17910-1871F	27BC2-289D2	BSR 1958	27BC2-28AB2	00EF0	Load_Menu_15/16, Retour_CRYSTAL_CAVERNS_11/12 Return_Menu_07/08
Load_Menu_16 In_Game_54	D2-LOW	1D38	28AB2	16A6	7C100 7C000	00B48	18720-19267	28AB2-295FA	BSR 1958	28AB2-29654	00BA2	Load_Menu_16/16, Retour_CRYSTAL_CAVERNS_12/12 Return_Menu_08/08 Fin chargement Phase1, affichage Menu du jeu
Load_Level1_01	D2-LOW	24DE	515DC	16AE	70400	0CAA8	19268-25D0F	515DC-5E084	BSR 1824	515DC-641DC	12C00	Level#1_01/06
In_Game_61	D2-LOW	762C	515DC	16AE	7C100	0CAA8	19268-25D0F	515DC-5E084	BSR 1824	515DC-641DC	12C00	Retour_Level#1_02/07 (Venant de WATER_VORTEX)
In_Game_55-RET	D2-LOW	7800	515DC	16AE	74300	0CAA8	19268-25D0F	515DC-5E084	BSR 1824	515DC-641DC	12C00	Level#1_retour_du_1er_Pont_a_droite_01/04 (Venant de la cabane du vieille homme)
In_Game_31-A	D2-LOW	7966	515DC	16AE	74300	0CAA8	19268-25D0F	515DC-5E084	BSR 1824	515DC-641DC	12C00	Level#1_retour_vers_Arbre_agressif_01/04 (Venant de la gauche, du pont à requin)
In_Game_44	D2-LOW	7C80	515DC	16AE	7C100	0CAA8	19268-25D0F	515DC-5E084	BSR 1824	515DC-641DC	12C00	Retour_CRYSTAL_CAVERNS_02/12
Load_Level1_02	D2-LOW	24F0	4D800	16B6	70400	00E84	25D10-26B93	4D800-4E684	BSR 1824	4D800-50000	02800	Level#1_02/06
In_Game_63	D2-LOW	763E	4D800	16B6	7C100	00E84	25D10-26B93	4D800-4E684	BSR 1824	4D800-50000	02800	Retour_Level#1_03/07 (Venant de WATER_VORTEX)
In_Game_45	D2-LOW	7C9A	4D800	16B6	7C100	00E84	25D10-26B93	4D800-4E684	BSR 1824	4D800-50000	02800	Retour_CRYSTAL_CAVERNS_03/12
In_Game_56-RET	D2-LOW	7812	4D800	16B6	74300	00E84	25D10-26B93	4D800-4E684	BSR 1824	4D800-50000	02800	Level#1_retour_du_1er_Pont_a_droite_02/04 (Venant de la cabane du vieille homme)
In_Game_12	D2-LOW	8A9C	4D800	16B6	74300	00E84	25D10-26B93	4D800-4E684	BSR 1824	4D800-50000	02800	Retour_Level#2_04/06
Load_Level1_03	D2-LOW	2508	2AFBE	16BE	70400	0DDC8	26B94-3495B	2AFBE-38D86	BSR 1824	2AFBE-42000	17042	Level#1_03/06

Ordre D'appel	Side	Adr. D'appel	A0 Adr. mém	A1 Pos. Table	DSKPTH	(A1)+ LENGTH	(A1) Pos. Disk Réel	Adr. Zone de Chargement en mémoire	Decomp/ Decrypt	Adr. Zone de Décomp. en mémoire	LENGTH Décomp	NFO
In_Game_46	D2-LOW	7CAC	2AFBE	16BE	7C100	0DDC8	26B94-3495B	2AFBE-38D86	BSR 1824	2AFBE-42000	17042	Retour_CRYSTAL_CAVERNS_04/12
In_Game_57-RET	D2-LOW	7824	2AFBE	16BE	74300	0DDC8	26B94-3495B	2AFBE-38D86	BSR 1824	2AFBE-42000	17042	Level#1_retour_du_1er_Pont_a_droite_03/04 (Venant de la cabane du vieille homme)
In_Game_31-B	D2-LOW	7980	2AFBE	16BE	74300	0DDC8	26B94-3495B	2AFBE-38D86	BSR 1824	2AFBE-42000	17042	Level#1_retour_vers_Arbre_agressif_02/04 (Venant de la gauche, du pont a requin)
Load_Level1_04	D2-LOW	251A	0AC00	16C6	70400	03C28	3495C-38583	0AC00-0E828	Non	N/A	N/A	Level#1_04/06
In_Game_58-RET	D2-LOW	7836	0AC00	16C6	74300	03C28	3495C-38583	0AC00-0E828	Non	N/A	N/A	Level#1_retour_du_1er_Pont_a_droite_04/04 (Venant de la cabane du vieille homme)
In_Game_31-C	D2-LOW	7992	0AC00	16C6	74300	03C28	3495C-38583	0AC00-0E828	Non	N/A	N/A	Level#1_retour_vers_Arbre_agressif_03/04 (Venant de la gauche, du pont a requin)
In_Game_48	D2-LOW	7CD0	0AC00	16C6	7C100	03C28	3495C-38583	0AC00-0E828	Non	N/A	N/A	Retour_CRYSTAL_CAVERNS_06/12
Load_Level1_06	D2-LOW	2542	6DE1C	16CE	70400	003A0	38584-38923	6DE1C-6E1BC	BSR 1824	6DE1C-6FD9C	01E80	Level#1_06/06
In_Game_47	D2-LOW	7CBE	6DE1C	16CE	7C100	003A0	38584-38923	6DE1C-6E1BC	BSR 1824	6DE1C-6FD9C	01E80	Retour_CRYSTAL_CAVERNS_05/12
In_Game_30	D2-LOW	8244	6DE1C	16CE	641DE	003A0	38584-38923	6DE1C-6E1BC	BSR 1824	6DE1C-6FD9C	01E80	Retour_Level4_02/02
In_Game_22	D2-LOW	84C2	6DE1C	16CE	641DE	003A0	38584-38923	6DE1C-6E1BC	BSR 1824	6DE1C-6FD9C	01E80	Retour_Level#3_Cage_Lift_02/02
In_Game_14	D2-LOW	8AC0	6DE1C	16CE	74300	003A0	38584-38923	6DE1C-6E1BC	BSR 1824	6DE1C-6FC9C	01E80	Retour_Level#2_06/06
In_Game_65	D2-LOW	7668	6DE1C	16CE	7C100	003A0	38584-38923	6DE1C-6E1BC	BSR 1824	6DE1C-6FC9C	01E80	Retour_Level#1_05/07 (Venant de WATER_VORTEX)
Load_Level1_05	D2-LOW	2530	29654	16D6	70400	01178	38924-39A9B	29654-2A7CC	BSR 1824	29654-2AFBE	0196A	Level#1_05/06
In_Game_49	D2-LOW	7CDE	29654	16D6	7C100	01178	38924-39A9B	29654-2A7CC	BSR 1824	29654-2AFBE	0196A	Retour_CRYSTAL_CAVERNS_07/12
In_Game_31-D	D2-LOW	79A0	29654	16D6	74300	01178	38924-39A9B	29654-2A7CC	BSR 1824	29654-2AFBE	0196A	Level#1_retour_vers_Arbre_agressif_04/04 (Venant de la gauche, du pont a requin)
In_Game_18	D2-LOW	7A3C	291D2	16D6	74300	01178	38924-39A9B	291D2-2A34A	BSR 1824	29654-2AB3C	014E8	Level#1_Après_block_qui_tombe_04/04 (vers la gauche, après l'ennemi 'pousse-pierre')
In_Game_10	D2-LOW	8A70	515DC	16DE	74300	04F08	39A9C-3E9A3	515DC-564E4	BSR 1824	515DC-5A5DC	09000	Retour_Level#2_02/06
In_Game_01	D2-LOW	719A	515DC	16DE	74300	04F08	39A9C-3E9A3	515DC-564E4	BSR 1824	515DC-5A5DC	09000	Level#1_Arbre_agressif_01/04
In_Game_02	D2-LOW	71AC	5A5DC	16E6	74300	054A0	3E9A4-43E43	5A5DC-5FA7C	BSR 1824	5A5DC-641DC	09C00	Level#1_Arbre_agressif_02/04 (vers_la_gauche, vers pont requin)
In_Game_11	D2-LOW	8A82	5A5DC	16E6	74300	054A0	3E9A4-43E43	5A5DC-5FA7C	BSR 1824	5A5DC-641DC	09C00	Retour_Level#2_03/06
In_Game_29	D2-LOW	8232	5A5DC	16E6	641DE	054A0	3E9A4-43E43	5A5DC-5FA7C	BSR 1824	5A5DC-641DC	09C00	Retour_Level4_01/02

Ordre D'appel	Side	Adr. D'appel	A0 Adr. mém	A1 Pos. Table	DSKPTH	(A1)+ LENGTH	(A1) Pos. Disk Réel	Adr. Zone de Chargement en mémoire	Decomp/ Decrypt	Adr. Zone de Décomp. en mémoire	LENGTH Décomp	NFO
In_Game_21	D2-LOW	84B0	5A5DC	16E6	641DE	054A0	3E9A4-43E43	5A5DC-5FA7C	BSR 1824	5A5DC-641DC	09C00	Retour_Level#3_Cage_Lift_01/02
In_Game_23	D2-LOW	7AD0	2D2E0	16EE	74300	0B26C	43E44-4F0AF	2D2E0-3854C	BSR 1824	2D2E0-42000	14D20	Level#1_Retour_block_qui_tombe_01/02
In_Game_13	D2-LOW	8AAE	2D2E0	16EE	74300	0B26C	43E44-4F0AF	2D2E0-3854C	BSR 1824	2D2E0-42000	14D20	Retour_Level#2_05/06 (Venant de la gauche)
In_Game_03	D2-LOW	71C6	2D2E0	16EE	74300	0B26C	43E44-4F0AF	2D2E0-3854C	BSR 1824	2D2E0-42000	14D20	Level#1_Arbre_agressif_03/04 (vers_la_gauche, vers pont requin)
In_Game_04	D2-LOW	71D8	0AA00	16F6	74300	03A34	4F0B0-52AE3	0AA00-0E434	BSR 8C0A Non	N/A	N/A	Level#1_Arbre_agressif_04/04 (vers_la_gauche, vers pont requin)
In_Game_24	D2-LOW	7AE2	0AA00	16F6	74300	03A34	4F0B0-52AE3	0AA00-0E434	Non	N/A	N/A	Level#1_Retour_block_qui_tombe_02/02
In_Game_15	D2-LOW	7A0A	32A4C	16FE	74300	07DC8	52AE4-5A8AB	32A4C-3A814	BSR 1824	32A4C-42000	0F5B4	Level#1_Après_block_qui_tombe_01/04 (Venant de la gauche)
In_Game_16	D2-LOW	7A1C	2AB3C	1706	74300	041C8	5A8AC-5EA73	2AB3C-2ED04	BSR 1824	2AB3C-32A4C	07F10	Level#1_Après_block_qui_tombe_02/04 (vers la gauche, après l'ennemi 'pousse-pierre')
In_Game_17	D2-LOW	7A2E	0A300	170E	74300	045EC	5EA74-6305F	0A300-0E8EC	Non	N/A	N/A	Level#1_Après_block_qui_tombe_03/04 (vers la gauche, après l'ennemi 'pousse-pierre')
In_Game_19	D2-LOW	8360	5A5DC	1716	641DE	0353C	63060-6659B	5A5DC-5DB18	BSR 1824	5A5DC-5DBE7	0360B	Level#3_Cage_Lift_01/02
In_Game_28	D2-LOW	8150	6DE1C	171E	641DE	00588	6659C-66B23	6DE1C-6E3A4	BSR 1824	6DE1C-6FD9C	01E80	Level#4_After_Password_of_Barloom_Projection_04/04
In_Game_20	D2-LOW	8372	6DE1C	171E	641DE	00588	6659C-66B23	6DE1C-6E3A4	BSR 1824	6DE1C-6FD9C	01E80	Level#3_Cage_Lift_02/02
In_Game_25	D2-LOW	811A	5A5DC	1726	641DE	01CE4	66B24-68807	5A5DC-5C2C0	BSR 1824	5A5DC-5D5DC	03000	Level#4_After_Password_of_Barloom_Projection_01/04
In_Game_26	D2-LOW	812C	5E1DC	172E	641DE	001E0	68808-689E7	5E1DC-5E3BC	BSR 1824	5E1DC-5E7DC	00600	Level#4_After_Password_of_Barloom_Projection_02/04
In_Game_27	D2-LOW	813E	61DDC	1736	641DE	00B5C	689E8-69543	61DDC-62938	BSR 1824	61DDC-63D5C	01F80	Level#4_After_Password_of_Barloom_Projection_03/04
In_Game_05 In_Game_09-RET In_Game_59 In_Game_60	D2-LOW	7E68	30000	173E	74300 7C100	02EEC	69544-6C42F	30000-32EEC	BSR 1826	641DC-6A5DC	06400	Entrer_dans_maison_GoTo_Level#2_01/04, Retour_Level#2_01/06 CRYSTAL_CAVERNS_01/11 (Venant de Down Stairs et I/O sur Disk1) Level_6_01/04 (Water Vortex) et Retour_Level#1_01/07
In_Game_06	D2-LOW	8918	515DC	1746	74300	02C44	6C430-6F073	515DC-54220	BSR 1824	515DC-58C5D	07681	Entrer_dans_maison_GoTo_Level#2_02/04
In_Game_07	D2-LOW	892A	5F9DC	174E	74300	00B70	6F074-6FBE3	5F9DC-6054C	BSR 1824	5F9DC-641DC	04800	Entrer_dans_maison_GoTo_Level#2_03/04
In_Game_08	D2-LOW	893C	4E0C0	1756	74300	0011C	6FBE4-6FCFF	4E0C0-4E1DC	BSR 1824	4E0C0-4E700	00640	Entrer_dans_maison_GoTo_Level#2_04/04

Disk2

Ordre D'appel	Side	Adr. D'appel	A0 Adr. mém	A1 Pos. Table	DSKPTH	(A1)+ LENGTH	(A1) Pos. Disk Réel	Adr. Zone de Chargement en mémoire	Decomp/ Decrypt	Adr. Zone de Décomp. en mémoire	LENGTH Décomp	NFO
In_Game_54b	D1-LOW	7E68	30000	175E	7C100	4228	6FD00-73F87	30000-34288	BSR 1826	641DC-6A5DC	6400	Entrée Vortex
In_Game_60	D1-LOW	74FA	5955C	1766	74300	04228	73F88-781AF	5955C-5D784	BSR 1824	5955C-641DC	0AC80	Level_6_02/04
In_Game_61	D2-LOW	750C	4E840	176E	74300	001D0	781B0-7837F	4E840-4EA10	BSR 1824	4E840-4FA50	01210	Level_6_03/04
In_Game_62	D2-LOW	751E	6DE1C	1776	74300	00628	78380-789A7	6DE1C-6E444	BSR 1824	6DE1C-6FE5C	02040	Level_6_04/04
						D35C	789A8-85D03			ZONE NON UTILISEE ~52Ko – Spécifique changement de Side		
In_Game_67	D2-UP	768C	2E6EC	177E	7C100	0B00C	85D04-90D0F	2E6EC-396F8	BSR 1824	2E6EC-42000	13914	Retour_Level#1_07/07 (Venant de WATER_VORTEX)
In_Game_57	D2-UP	709E	2E6EC	177E	74300	0B00C	85D04-90D0F	2E6EC-396F8	BSR 1824	2E6EC-42000	13914	Level#1_Pont_a_droite 03/04 (vers cabane vieille homme)
In_Game_58	D2-UP	70B0	0AC00	1786	74300	030D4	90D10-93DE3	0AC00-0DCD4	Non	N/A	N/A	Level#1_Pont_a_droite 04/04 (vers cabane vieille homme)
In_Game_56	D2-UP	708C	4D800	178E	74300	00478	93DE4-9425B	4D800-4DC78	BSR 1824	4D800-4F740	01F40	Level#1_Pont_a_droite 02/04 (vers cabane vieille homme)
In_Game_66	D2-UP	767A	4D800	178E	7C100	00478	93DE4-9425B	4D800-4DC78	BSR 1824	4D800-4F740	01F40	Retour_Level#1_06/07 (Venant de WATER_VORTEX)
In_Game_55	D2-UP	7074	38000	1796	74300	05914	9425C-99B6F	38000-3D914	BSR 1826	55DDC-5F85C	09A80	Level#1_Pont_a_droite 01/04 (vers cabane vieille homme)
In_Game_64	D2-UP	7650	6A5DC	1796	7C100	05914	9425C-99B6F	6A5DC-6FEF0	BSR 1826	55DDC-5BF50	06174	Retour_Level#1_04/07 (Venant de WATER_VORTEX)
N/A	D2-UP	78E4	38000	1796		05914	9425C-99B6F	38000-3D914	BSR 1826	55DDC-5F85C	9A80	Jamais appelé. Position dans map en $2BE, Jamais atteinte car chateau apparait et empêche d'aller à cette position (10 pas de plus que l'entrée du chateau)
END_01	D2-UP	4F7E	0F0C6	179E	7C100	00308	99B70-99E77	0F0C6-0F3CE	BSR 1824	0F0C6-0F57C	04B6	Quit_Game_01/04
END_02	D2-UP	4F90	0F57C	17A6	7C100	32A84	99E78-CC8FB	0F57C-42000	Non	N/A	N/A	Quit_Game_02/04
END_03	D2-UP	5010	70400	17AE	6AFE0	0F778	CC8FC-DC073	70400-7FB78	Non	N/A	N/A	Quit_Game_03/04
END_04	D2-UP	501E	641DC	17B6	6AFE0	06E04	DC074-E2E77	641DC-6AFE0	BSR 1958	641DC-6DE1B	09C3F	Quit_Game_04/04
In_Game_68	D2-UP	78B0	58DDC	17BE	7C100	04C98	E2E78-E7B0F	58DDC-5DA74	BSR 1824	58DDC-6405C	0B280	CASTLE_01/01 (Fait apparaitre l'entré du chateau)
Load_Menu_09	D2-UP	1D5E	002A8	17C6	7C000	00060	E7B10-E7B6F	002A8-00308	BSR 1824	002A8-0043C	00194	Load_Menu_09/16 et Return_Menu_01/08
In_Game_69 [END]	D2-UP	85EE	515DC	17CE	74300	04520	E7B70-EC08F	515DC-55AFC	BSR 1824	515DC-5B21C	09C40	END_GAME_01/01
Load_Menu_01	D2-UP	1C66	46FB4	17D6	18000	03A90	EC090-EFB1F	46FB4-4AA44	BSR 1824	46FB4-4D800	0684C	Load_Menu_01/16
Load_Menu_02	D2-UP	1C78	70000	17DE	18000	00250	EFB20-EFD6F	70000-70250	BSR 1824	70000-70390	00390	Load_Menu_02/16

Ordre D'appel	Side	Adr. D'appel	A0 Adr. mém	A1 Pos. Table	DSKPTH	(A1)+ LENGTH	(A1) Pos. Disk Réel	Adr. Zone de Chargement en mémoire	Decomp/ Decrypt	Adr. Zone de Décomp. en mémoire	LENGTH Décomp	NFO
Load_Menu_03	D2-UP	1C8A	45F34	17E6	18000	00438	EFD70-F01A7	45F34-4636C	BSR 1824	45F34-46AD4	00BA0	Load_Menu_03/16
Load_Menu_04	D2-UP	1C9C	50000	17EE	18000	00124	F01A8-F02CB	50000-50124	BSR 1824	50000-5019C	0019C	Load_Menu_04/16
Load_Menu_05	D2-UP	1CAE	515DC	17F6	18000	00664	F02CC-F092F	515DC-51C40	BSR 1826	42000-4324B	0124B	Load_Menu_05/16
Load_Menu_06	D2-UP	1CC6	50398	17FE	18000	00268	F0930-F0B97	50398-50600	BSR 1824	50398-509D7	0063F	Load_Menu_06/16
Load_Menu_07	D2-UP	1CD8	457A4	1806	18000	003FC	F0B98-F0F93	457A4-45BA0	BSR 1824	457A4-45F34	00790	Load_Menu_07/16
Load_Menu_08	D2-UP	1E70	4324C	180E	18000	01EBC	F0F94-F2E4F	4324C-45108	Non	N/A	N/A	Load_Menu_08/16

Disk1

Ordre D'appel	SIDE	Adr. D'appel	A0 Adr. mém	A1 Pos. Table	DSKPTH	(A1)+ LENGTH	(A1) Pos. Disk Réel	Adr. Zone de Chargement en mémoire	Decomp/ Decrypt	Adr. Zone de Décomp. en mémoire	LENGTH Décomp	NFO
Anim part #01	D1-LOW	60A	16B14	580	763C0	46F74	0189C-4880F	16B14-5DA88	BSR 6B4	N/A	N/A	Animation Part #01 de Shadow Of The Beast II
Anim Psygnosis	D1-LOW	70044	45610	N/A	6A000	24284	48810-6CA93	45610-69894	BSR 70310	002B0-28364	280B4	Animation Psygnosis
Last_Load_Disk1	D1-LOW	65A	40000	590	50000	9B78	6CA94-7660B	40000-49B78	N/A	N/A	N/A	Dernier code chargé après fin animation
In_Game_37	D1-LOW	72F0	0A800	161E	7C100	04074	76FC0-7B033	0A800-0E874	Non	N/A	N/A	CRYSTAL_CAVERNS_06/11 (Venant de Down Stairs)
						ACCF	7B034-85D03					ZONE NON UTILISEE Fin de Disquette
Anim part #02	D1-UP	62A	16B14	588	6C780	48DDE	85D04-CEAE1	16B14-5F8F2	BSR 6B4	N/A	N/A	Animation Part #02 de Shadow Of The Beast II
						1476	CEAE1-CFF57					ZONE NON UTILISEE ~5Ko
In_Game_43	D1-UP	7C44	31F40	1626	7C100 74300	0280C	CFF58-D2763	31F40-3474C	Non	N/A	N/A	Retour_CRYSTAL_CAVERNS_01/12 (puis demandera d'insérer le Disk2)
In_Game_32	D1-UP	7E68	30000	1626	7C100	0280C	CFF58-D2763	30000-3280C	BSR 1826	641DC-6A5DC	06400	CRYSTAL_CAVERNS_01/11 (Venant de Down Stairs)
In_Game_33	D1-UP	72A8	515DC	162E	7C100	0C3C8	D2764-DEB2B	515DC-5D9A4	BSR 1824	515DC-641DC	12C00	CRYSTAL_CAVERNS_02/11 (Venant de Down Stairs)
In_Game_34	D1-UP	72BA	4E700	1636	7C100	00524	DEB2C-DF04F	4E700-4EC24	BSR 1824	4E700-4F600	00F00	CRYSTAL_CAVERNS_03/11 (Venant de Down Stairs)
In_Game_35	D1-UP	72CC	6DE1C	163E	7C100	0151C	DF050-E056B	6DE1C-6F338	BSR 1824	6DE1C-6FC9C	01F80	CRYSTAL_CAVERNS_04/11 (Venant de Down Stairs)
In_Game_36	D1-UP	72DE	2E248	1646	7C100	0A73C	E056C-EACA7	2E248-38984	BSR 1824	2E248-42000	13DB8	CRYSTAL_CAVERNS_05/11 (Venant de Down Stairs)
In_Game_38	D1-UP	7308	0F38A	164E	7C100	007F0	EACA8-EB497	0F38A-FB7A	BSR 1824	0F38A-1013A	00DB0	CRYSTAL_CAVERNS_07/11 (Venant de Down Stairs)
In_Game_39	D1-UP	731A	1013A	1656	7C100	10410	EB498-FB8A7	1013A-2054A	BSR 1824	1013A-2AF4A	1AE10	CRYSTAL_CAVERNS_08/11 (Venant de Down Stairs)
In_Game_40	D1-UP	732C	2AF4A	165E	7C100	01828	FB8A8-FD0CF	2AF4A-2C772	BSR 1958	2AF4A-2C7B6	0186C	CRYSTAL_CAVERNS_09/11 (Venant de Down Stairs)
In_Game_41	D1-UP	733E	2C7B6	1666	7C100	00E10	FD0D0-FDEDF	2C7B6-2D5C6	BSR 1958	2C7B6-2D6A6	00EF0	CRYSTAL_CAVERNS_10/11 (Venant de Down Stairs)
In_Game_42	D1-UP	7350	2D6A6	166E	7C100	00B48	FDEE0-FEA27	2D6A6-2E1EE	BSR 1958	2D6A6-2E247	00BA1	CRYSTAL_CAVERNS_11/11 (Venant de Down Stairs)

Part 12a Rip Full Disk1

Rip Disk1 Side Lower :

Le disque 1 utilise aussi un format *MFM custom* (voir **part 2** Analyse de l'image IPF)

Question : Est-il vraiment utile de riper le Disk1 au complet ?
Nous savons qu'il est principalement utilisé pour les animations du début du jeu, d'ailleurs nous allons utiliser une autre méthode pour ripper celles-ci.
*(nous avons déjà rippé l'animation Psygnosis donc il nous restera plus que 2 animations à ripper)

Mais pour l'instant, occupons-nous du disque complet Side par Side. (Le Trackloader fonctionne par Side)

Le disque 1 est bootable, donc la Track 0.0 est au format standard 'AmigaDos' elle ne pourra être lue par notre Trackloader Custom.
Donc nous avons à riper : **(!80-1)*!6300 = $79824 = !497 700 Bytes**

Rappel du fonctionnement du Trackloader Custom de SOTB2 :

A0 = Destination memory
A1 = Adr dans le tableau (et modifier la length dans le tableau donc)

Le buffer du trackloader custom est situé en **$6A000**

Sur un A500, la zone mémoire potentiel libre avec marge est donc de : **$1000 → $69000 = $68000** soit '425 984 Bytes'
Cella ne suffira pas pour le rip complet d'une side, voyons combien on peut lire de 'track custom' dans '425 984 Bytes'
!425984 / !6300 = !67
!67 * !6300 = $68000 = !422 100 Bytes

Let's Rock'n roll
Booter sur le Disk1 original de SOTB2 et **entrer** dans l'AR lors du trackload (avant affichage de la 1er animation)

Tapez :
```
A 7003A
$7003A  LEA     $1000,A0              ; Destination mémoire

A 70048
$70048  MOVE.W  #F0,DFF180            ; → Ecran vert en fin de trackload
$70050  NOP                           ;    Wait
$70052  MOVE.W  #0,DFF180             ;    Ecran noir
$7005A  BRA     70048                 ; ← Dead loop
```

Puis, Tapez :
```
M 70060         mettre 00 00 18 9C    ; adr physique départ sur disque (Start pos.+1 track Custom)*
M 70064         mettre 00 06 80 00    ; Longueur à lire
```

*Comme expliqué au-dessus, on commence à la 1er Track Custom qui est la seconde track de la side.
La 1er Track étant non-custom car c'est celle de boot, elle est au format AmigaDOS.

Et on lance le *trackload*, **Tapez** : **G 7003A**

Trackload des Tracks de 01 → 68 sans aucun problème et on arrive sur notre fond d'écran en vert, le trackload est terminé.
Entrez dans l'AR.

On sauve le tout sur une nouvelle disquette vierge insérée dans DF1.
Tapez :SM 1:Disk1_lower_00, 1000 1000+68000

On fait le reste :
```
Total a riper     : $79824
Déjà ripé         : $68000
Reste             : $79824-$68000   = $11824
```

Tapez :
```
M 70060         Mettre 00 06 98 9C    ; adr physique départ sur disque ($189C+$68000)
M 70064         Mettre 00 01 18 24    ; Long à lire
```

Et on lance notre trackloader, **Tapez** : **G 7003A**
Trackload des dernières Tracks sans aucun problème et on arrive sur notre fond d'écran en vert, le trackload est terminé.
Entrez dans l'AR.

```
SM 1:Disk1_lower_01, 1000 1000+11824
```

On joint les deux :
Enlever la disquette dans DF1 et insérez la dans DF0 à la place du Disk Original N°1 de Shadow Of The Beast.

```
O 00, 1000 80000                      ; Petit nettoyage
LM Disk1_lower_00, 1000               ; Rip 1er disquette side Lower phase 1
LM Disk1_lower_01, 69000              ; Rip 1er disquette side Lower phase 2
Format SOTB_Disk1_lower               ; Formatage si nécessaire d'une nouvelle disquette
SM Disk1_Lower.bin,1000 7A824         ; Rip Full side Lower 1er disquette
```

Reste à faire l'autre side.

Rip Disk1 Side Upper :

Sur la seconde face il n'y a pas de Track AmigaDOS MAIS il y a notre fameuse Track de protection/signature
Donc finalement, cela revient à riper exactement la même quantité de donnée sur cette face.

Rappel du fonctionnement du code pour la sélection de Side, **Upper** ou **Lower** = $08 44 68

```
701FA CMP.L #$84468, D0          ; Compare D0 avec $84468
70200 BGE.B $70206               ; Si plus grand que       alors GoTo #DF0__SIDE_UP__MOTOR_ON__DIR_EXT
70202 BSR.B $701A6               ; sinon                         GoTo #DF0__SIDE_DOWN__MOTOR_ON__DIR_EXT
```

La 1er Track en side Upper commence donc à l'adresse $84468
*Nous allons commencer à la seconde track comme expliqué juste au-dessus donc à l'adresse : $84468 + $189C = $85D04

Let's Go
Booter sur le Disk1 original de SOTB2 et **entrer** dans l'AR lors du trackload (avant affichage de la 1er animation)

Tapez :

```
A 7003A
$7003A  LEA     $1000,A0            ; Destination mémoire

A 70048
$70048  MOVE.W  #F0,DFF180          ; → Ecran vert en fin de trackload
$70050  NOP                         ;   Wait
$70052  MOVE.W  #0,DFF180           ;   Ecran noir
$7005A  BRA     70048               ; ← Dead loop
```

Puis, Tapez :

```
M 70060         mettre 00 08 5D 04  ; adr physique départ sur disque *(voir explication au-dessus)
M 70064         mettre 00 06 80 00  ; Longueur à lire
```

Et on lance le *trackload*, **Tapez** : **G 7003A**

Trackload des Tracks de 01 → 68 sans aucun problème et on arrive sur notre fond d'écran en vert, le trackload est terminé.
Entrez dans l'AR.

On sauve le tout sur une nouvelle disquette vierge inseré dans DF1.
Tapez : SM 1:Disk1_upper_00, 1000 1000+68000

On fait le reste :

```
Total a riper     : $79824 (79 Tracks de $189C, voir plus haut)
Déjà ripé         : $68000
Reste             : $79824-$68000   = $11824
```

Tapez :

```
M 70060         Mettre 00 0E DD 04  ; adr physique départ sur disque ($85D04+$68000)
M 70064         Mettre 00 01 18 24  ; Long à lire
```

Et on lance notre trackloader, **Tapez** : **G 7003A**
Trackload des dernières Tracks sans aucun problème et on arrive sur notre fond d'écran en vert, le trackload est terminé.
Entrez dans l'AR.

```
SM 1:Disk1_upper_01, 1000 1000+11824
```

On joint les deux :
Enlever la disquette dans DF1 et insérez la dans DF0 à la place du Disk Original N°1 de Shadow Of The Beast.

```
O 00, 1000 80000                    ; Petit nettoyage
LM Disk1_upper_01, 1000             ; Rip 1er disquette side Upper phase 1
LM Disk1_upper_02, 69000            ; Rip 1er disquette side Upper phase 2
Format SOTB2_Disk1_upper            ; Formatage si nécessaire d'une nouvelle disquette
SM Disk1_Upper.bin,1000 7A824       ; Rip Full side Upper 1er disquette
```

Part 12b Rip Full Disk2

On retrouve exactement le même format MFM custom sur le disk2, nous allons utiliser toujours la même méthode pour ripper ses données.

Rip Disk2 Side Lower :

Le disque 2 n'est pas bootable, donc la Track 0.0 n'est pas au format standard 'AmigaDos' mais dans un format *custom MFM*
Ce qui nous donnes un total de track a ripper :

```
!80*!6300 = $7B0C0 = !504 000 Bytes
```

Booter sur le Disk1 original de SOTB2 et **entrer** dans l'AR lors du trackload (avant affichage de la 1er animation)
Swaper dans le lecteur interne le disk1 par le disk2 original de SOTB2 et insérez une **nouvelle disquette** vierge dans le lecteur externe.

Tapez :

```
A 7003A
$7003A  LEA     $1000,A0              ; Destination mémoire

A 70048
$70048  MOVE.W  #F0,DFF180            ; → Ecran vert en fin de trackload
$70050  NOP                           ;   Wait
$70052  MOVE.W  #0,DFF180             ;   Ecran noir
$7005A  BRA     70048                 ; ← Dead loop
```

Puis

```
M 70060         Mettre 00 00 00 00    ; adr physique départ sur disque
M 70064         Mettre 00 06 80 00    ; Long à lire
```

Et on lance le *trackload*, tapez : **G 7003A**

Trackload des Tracks de 01 → 68 sans aucun problème et on arrive sur notre fond d'écran en vert, le trackload est terminé.
Entrez dans l'AR.
On sauve le tout sur notre nouvelle disquette vierge préalablement insérée dans DF1.
Tapez : SM 1:Disk2_lower_01, 1000 1000+68000

Sans oublier le reste :

```
Total a riper     : $7B0C0
Déjà ripé         : $68000
Reste             : $7B0C0-$68000   = $130C0
```

Tapez :

```
M 70060         Mettre 00 06 80 00    ; adr physique départ sur disque
M 70064         Mettre 00 01 30 C0    ; Long à lire
```

Et on lance notre trackloader, **Tapez** : **G 7003A**
Trackload des dernières Tracks sans aucun problème et on arrive sur notre fond d'écran en vert, le trackload est terminé.

Entrez dans l'AR.
Tapez : SM 1:Disk2_lower_02, 1000 1000+130C0

On joint les deux :
Enlever la disquette dans DF1 et insérez la dans DF0 à la place du Disk Original N°2 de Shadow Of The Beast.

```
O 00, 1000 80000                      ; Petit nettoyage
LM Disk2_lower_01, 1000               ; Rip 1er disquette side Lower phase 1
LM Disk2_lower_02, 69000              ; Rip 1er disquette side Lower phase 2
Format SOTB2_Disk2_lower              ; Formatage si nécessaire d'une nouvelle disquette
SM Disk2_Lower.bin,1000 7A824         ; Rip Full side Lower 2eme disquette
```

Rip Disk2 Side Upper :

Donc….normalement, 1 disque = 80 cylindre (0 à 79)
Sauf que l'on a la track de protection/signature : Cylindre 0.1 (voir code plus haut ou analyseur web)
Ce qui nous donnes donc finalement 79 track a ripper sur la face **'Upper'**
!79*!6300 = *$79824* = **!497 700 Bytes**

La 'base' pour 'trackloader' en side **Upper** est *$84468* (voir code plus haut);

On lui ajoute donc 1 Track Custom pour que le trackloader fonctionne et commence la lecture sur la seconde Track de la side upper.
Donc **$84468+$189C=***$85D04*, c'est la valeur a mettre dans le tableau en $70060 comme *'Adresse_Start_raw'* et comme précédemment, on rip en deux phases.

Booter sur le Disk1 original de SOTB2 et **entrer** dans l'AR lors du trackload (avant affichage de la 1er animation)
Swaper dans le lecteur interne le disk1 par le disk2 original de SOTB2 et insérez une **nouvelle disquette** vierge dans le lecteur externe.

Tapez :

```
A 7003A
$7003A  LEA     $1000,A0              ; Destination mémoire

A 70048
$70048  MOVE.W  #F0,DFF180            ; → Ecran vert en fin de trackload
$70050  NOP                           ;    Wait
$70052  MOVE.W  #0,DFF180             ;    Ecran noir
$7005A  BRA     70048                 ; ← Dead loop
```

Puis

```
M 70060         Mettre 00 08 5D 04    ; adr physique départ sur disque (limite+1 track)=84468+189C=85D04)
M 70064         Mettre 00 06 80 00    ; Long à lire
```

Et on lance le *trackload*, **Tapez** : `G 7003A`

Trackload des Tracks de 01 → 68 sans aucun problème et on arrive sur notre fond d'écran en vert, le trackload est terminé.
Entrez dans l'AR.

On sauve le tout sur notre nouvelle disquette vierge inséré dans DF1.
Tapez :`SM 1:Disk2_Upper_00, 1000 1000+68000`

On fait le reste :

```
Total a riper     : $79824 (79 Tracks de $189C, voir plus haut)
Déjà ripé         : $68000
Reste             : $79824-$68000   = $11824
```

Tapez :

```
M 70060         Mettre 00 0E DD 04    ; adr physique départ sur disque ($85D04+$68000)
M 70064         Mettre 00 01 18 24    ; Long à lire
```

Et on lance notre trackloader, **Tapez** : `G 7003A`
Trackload des dernières Tracks sans aucun problème et on arrive sur notre fond d'écran en vert, le trackload est terminé.
Entrez dans l'AR.

```
SM 1:Disk1_upper_01, 1000 1000+11824
```

On joint les deux :
Enlever la disquette dans DF1 et insérez la dans DF0 à la place du Disk Original N°1 de Shadow Of The Beast.

```
O 00, 1000 80000                      ; Petit nettoyage
LM Disk1_upper_00, 1000               ; Rip 1er disquette side Upper phase 1
LM Disk1_upper_01, 69000              ; Rip 1er disquette side Upper phase 2
Format SOTB2_Disk1_upper              ; Formatage si nécessaire d'une nouvelle disquette
SM Disk1_Upper.bin,1000 7A824         ; Rip Full side Upper 1er disquette
```

Part 13 Rip Disk1 <FILE>

RIP Anim Part 01 et 02 ainsi que le dernier bout de code

Rebooter votre Amiga avec l'original Disque1 dans lecteur Interner et une nouvelle disquette vierge (oui encore) dans le lecteur externe.
Entrer dans l'AR pendant l'animation de psygnosis.

Nous avons déjà noté toutes les valeurs (revoir Part 10 Schéma Part #01 (aka : l'organigramme qui fait peur…))
Mais bon, pour le fun :
On pause un *breakpoints* juste avant la phase de décompression :

Tapez : **BS 6B4** et on retourne au code du jeu, tapez : **X**

Notre *breakpoint* est atteint une fois le trackload terminé et on entre automatiquement dans l'AR et comme prévu nous allons regarder nos valeurs.
Tapez : **R** on note **A0** qui est notre adresse Source des données.
On regarde ensuite à l'adresse pointée en **A1** pour connaitre la longueur à sauvegarder.

```
bs 6B4
Breakpoint inserted
Ready.
x
No known virus in memory!
Ready.
Breakpoint raised at address: 000006B4
r
D0=01AC2991 00000002 0000FFFF 00000000  000000DA 55555555 0000FFFF 01040111
A0=00016B14 00000584 0007447C 0000057C  0000B698 00001E14 00076AA0 00001760
PC = 000006B4 USP = 000018B2 SR = 2000 T=0 S=1 I=000 X=0 N=0 Z=0 V=0 C=0
m 584
:000584 00 04 6F 74 00 08 5D 04 00 04 8D DE 00 04 88 10 ..ot..]..........

sm 1:Anim_01, 16B14 16B14+46F74
```

Et on sauve le tout sur notre disquette vierge en DF1, **tapez : SM 1:Anim_01, 16B14 16B14+46F74**

On efface notre *breakpoint,* **tapez** : **BDA** et on retourne au code du jeu avec la commande **X**
La décompression originale s'effectue et la 1ère partie de la l'animation est jouée.
Entrez dans l'AR, on en profite pour poser nos deux *breakpoints.*
Tapez : BS 634 et **BS 65A** et on retourne au code du jeu avec la commande **X**

Notre breakpoint est atteint une fois le trackload terminé et on entre automatiquement dans l'AR et comme prévu nous allons regarder nos valeurs.
Tapez : **R** et on regarde la même chose que précédemment.

```
bs 634
Breakpoint inserted
Ready.
bs 65A
Breakpoint inserted
Ready.
x
No known virus in memory!
Ready.
Breakpoint raised at address: 00000634
r
D0=0500FDFC 00000002 0000FFFF 00000000  0000024D 55555555 0000FFFF 05005554
A0=00016B14 0000058C 0007E0BC 0000057C  0000B698 00001E14 0006D9F8 00001764
PC = 00000634 USP = 000018B2 SR = 2000 T=0 S=1 I=000 X=0 N=0 Z=0 V=0 C=0
m 58C
:00058C 00 04 8D DE 00 04 88 10 00 00 BB 80 33 FC 00 04 ............3ü..
```

On sauve le tout sur notre disquette toujours présente en DF1, **tapez : SM 1:Anim_02, 16B14 16B14+48DDE**
Ok ? On retourne au code du jeu avec la commande **X**

L'animation se poursuit et le dernier Trackload arrive et enfin, notre dernier *breakpoints* est atteint, on entre automatiquement dans l'AR.
Tapez : **R** et on regarde la même chose que précédemment avec une variante sur l'adr. mémoire contenant la longueur (revoir analyse du code)

```
Breakpoint raised at address: 0000065A
r
D0=FFFFFFFF 00000002 000000F0 000000F0  0000024D 55555555 00000000 0000FFFF
A0=00040000 00000590 0007447C 0000057C  0000B698 00001E14 00016FDA 00000256
PC = 0000065A USP = 000018B2 SR = 2000 T=0 S=1 I=000 X=0 N=0 Z=0 V=0 C=0
m 590
:000590 00 06 CA 94 00 00 9B 78 33 FC 00 04 00 00 0D 88 .......x3ü......

sm 1:Last_Load.bin, 40000 40000+9B78
```

On sauve le tout sur notre disquette toujours présente en DF1, **tapez : SM 1:Last_Load, 40000 40000+9B78**

Part 13b Rip Disk2 <fichier par fichier>

Il existe une autre possibilité, parfaite dirons-nous.
Le rip de fichier en posant des Breakpoints pour chaque appel du TrackLoader (comme vu plus haut)

Exemple, on commence sur le message 'insert Disk2' juste après la fin des animations.
Rappel : Adr du trackloader *$1B2E*

Voir analyse du Decomp_Decrypt :
A0=adr. Source
A1=adr. dest ; Mais qui est biaisé par la 1er commande en 01824 MOVE.L A0, A1
Il va décompresser au même endroit que la source (il fonctionne à l'envers donc pas de soucis pour lui)

On va utiliser l'appel : $1E8E

```
1E96    LEA     70400,A0        ; Adresse de destination
1E9C    LEA     1676.S,A1       ; Position dans tableau          (Longueur data=271C)     2 tracks lue
1EA0    BSR     1B2E            ; GoTo #TrackLoader
1EA4    BSR     1824            ; GoTo #Decomp_Decrypt
1EA8    ...
```

Le dskpth est en $18000, on se laisse une bonne marche, on va utiliser la mémoire à partir de $30000

```
R A0 30000
O "PaTtErN", 30000 50000        ; On remplit la zone de destination d'un pattern
BS 1EA4
G 1E9C
```

Le chargement s'effectue et notre breakpoint est atteint.
On sauve tout ça :

```
SM 1:Beast2_4C18, 30000 30000+271C
```

```
1er LongWord              = 00 00 27 1D           ; Longueur de la zone compressée
Dernier LongWord          = 00 1C 9B 46           ;
Av Dernier LongWord       = 00 00 38 D9           ; A2 avant ajout
Av Av dern. LongWord      = 98 D7 B8 1D           ; D5
Av Av Av dern. LongWord   = 00 0C 94 07           ; D0
```

On va décrypter le tout :

```
BDA
BS 1EA8                         ; BreakPoint en fin de decomp_decryt
R A0 30000                      ; Devrait être déjà à cette valeur
X
```

```
F "PaTtErN", 30000
```

Fin des données en 390C4, ce qui nous donne une longueur de 390C4-30000=$90C4
On sauve le tout :

```
SM 1:Beast2D_4C18, 30000 390C4
```

```
10012    Beast2_4C18
37060    Beast2D_4C18
```

Il est aussi possible d'automatiser le rip, il existe une Zone de texte en *$2D90*, on peut l'utiliser pour l'instant :

```
2D90    MOVE.W  #$00F,$DFF180   ; Fond en bleu, on note A1
2D98    BTST    #6,$BFE001      ; Left Button
2DA0    BNE     2D90
2DA2    BSR     1B38
2DA6    MOVE.W  #$0F0,$DFF180   ; Fond vert, on sauve les données non décompressé
2DAE    BTST    #A,$DFF016      ; Right Button
2DB6    BNE     2DA6
2DB8    MOVE.W  #0,$DFF180      ; Retour écran noir
2DC0    RTS


1B30    BSR     1B38            ; à changer par un bsr 2D90      (opcode original=61 00 00 06 20)
1B34    MOVEA.L (A7)+,A0
1B36    RTS

8CE4    NOP                     ; On désactive l'attente d'appuie sur fire pour l'insertion du Disk2
```

(Rappel : La pose du BS bloque la routine de check de signature du disk....(voir plus haut))
On pose un BS en *1840* pour relever **A2** qui est l'adresse de fin de donnée compressé (si on passe par la routine de décompression bien sûr)

Fond Bleu (bouton gauche)
Début appel routine trackloader, **regarder A0** et regarder a quoi il correspond dans la table.
(ou regarder directement dans le fichier Last_Load_Original)

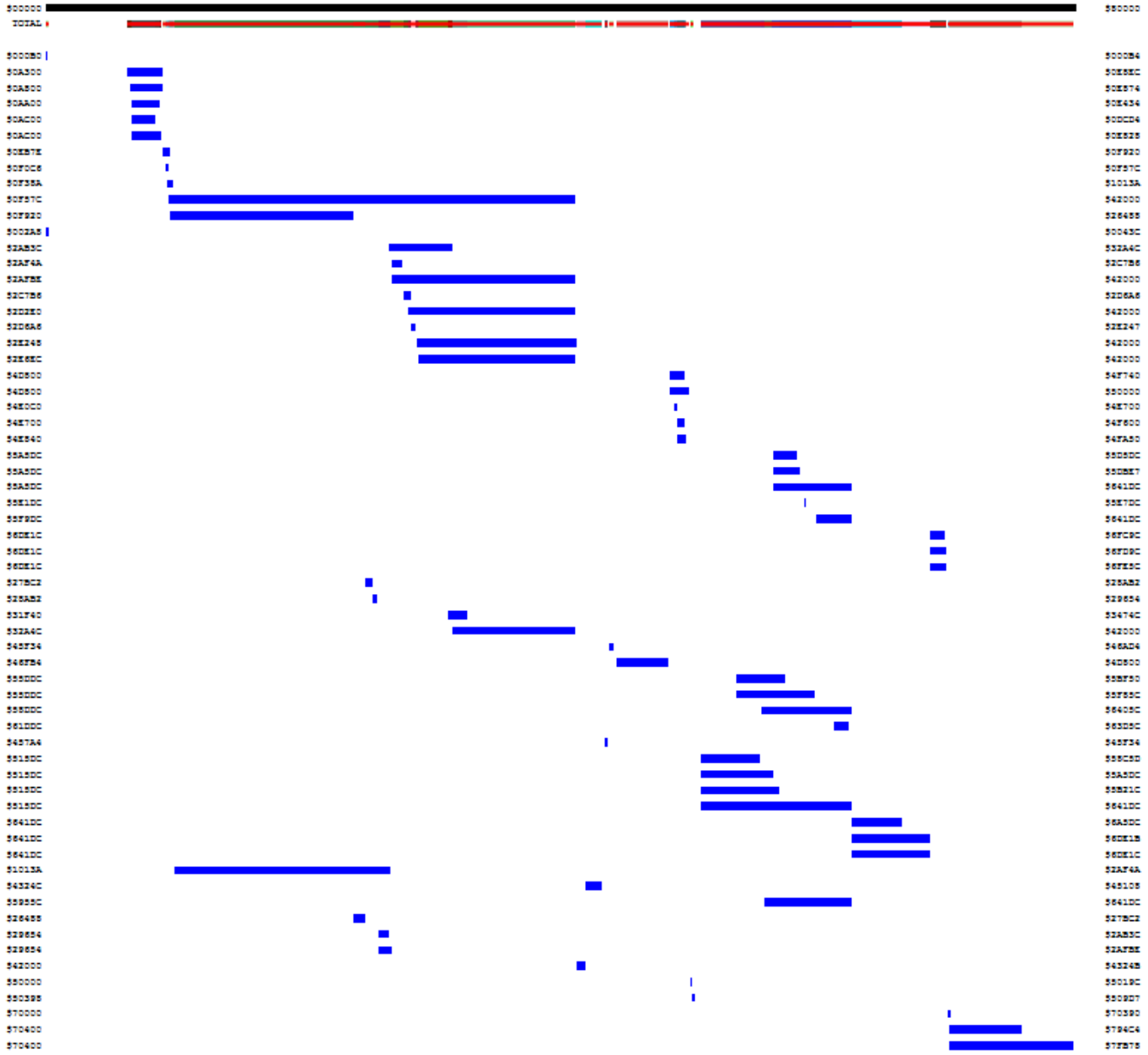
A0 indique l'adresse de destination, où sera copié les données lues du disque.
Au vue de ce que l'on a noté, toutes les données seront copiées après la zone de code située en mémoire basse
On peut donc à chaque écran bleu remplir la zone mémoire depuis **A0** jusqu'à la fin avec un pattern spécifique.

On appuie sur le bouton gauche de la souris, le trackloader charge les données du disque et les décompresse/décrypte

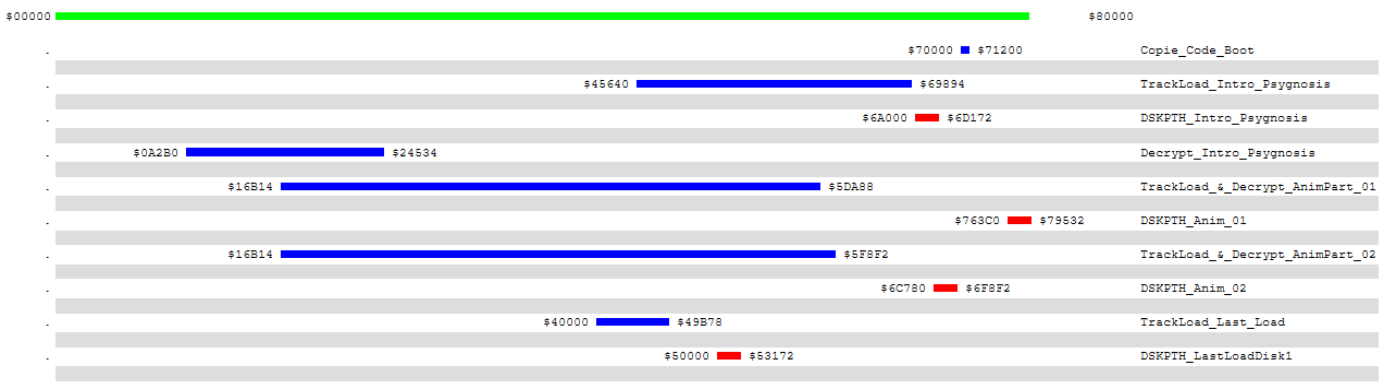
Fond Vert (bouton droit)
Data non décompressée chargée.
On recherche notre pattern en mémoire et on sauve le tout depuis A0 jusqu'a notre pattern.
On appuie sur le bouton droit de la souris, et on recommence.

Part 14 Tableau Plage d'occupation mémoire

En regardant le précédent tableau de tous les appels (Part11), il est possible d'établir un graphique de l'occupation mémoire pendant le jeu.
Ce qui nous donne :



Un 'focus' sur la zone 'spécifique' du 1er Disque



Il parait clair que les 512Ko de l'Amiga sont bien utilisés…

Part 14B Analyse du code pendant une partie

Avant de se pencher sur l'organisation que l'on va devoir mettre en place au niveau des données sur notre Crack.
Et oui car il parait évident que l'on ne peut pas copier 'in place' toutes les données ripées sur une disk AmigaDos sans faire des modifications.
On va regarder le code pendant la phase de jeu et voir si il n'y a rien de spécial qui pourrait nous poser problème.

Pour cela on va 'analyser' 2 chargements en partant du point de départ.
- Celui en partant vers la gauche.
- Celui en partant vers la droite vers 'Crystal Cavern'.

ça permettra de nous donner une idée de la 'chose'.
On lance le jeu original jusqu'à arriver au début du LEVEL1, on entre dans l'AR et on pose un *BreakPoint* sur le *TrackLoader*.
BS 1B2E et **BS 1B36**

On commence vers la gauche, très vite notre *BreakPoint* est atteint et on entre automatiquement dans l'AR
On jette un coup d'œil pour voir l'adresse de retour située dans la pile
R
M 24E → **719E**

Et on jette un coup d'œil à ce code, **tapez : D 719E-E**

```
07190    LEA $000515DC, A0                ; On retrouve nos chargement prévus avec les phases Trackload et Decrypt
07196    LEA $16DE.W, A1                  ; Arbre Agressif 1/4
0719A    BSR.W $00001B2E                  ; GoTo #Trackloader_Start
0719E    BSR.W $00001824                  ; GoTo #Decomp/Decrypt

071A2    LEA $0005A5DC, A0                ; etc
071A8    LEA $16E6.W, A1                  ; Arbre Agressif 2/4
071AC    BSR.W $00001B2E                  ; GoTo #Trackloader_Start
071B0    BSR.W $00001824                  ; GoTo #Decomp/Decrypt

071B4    BSR.W $000054E2                  ; On ne s'attarde pas sur les sous-routines, le but n'est pas de
071B8    BSR.W $00005588                  ; comprendre toutes les phases du jeux mais les points important de 'protection'
                                          ; ici, pas de CMP ou autre, donc on zap leur analyse.

071BC    LEA $0002D2E0, A0                ;
071C2    LEA $16EE.W, A1                  ; Arbre Agressif 3/4
071C6    BSR.W $00001B2E                  ; GoTo #Trackloader_Start
071CA    BSR.W $00001824                  ; GoTo #Decomp/Decrypt

071CE    LEA $0000AA00, A0                ;
071D4    LEA $16F6.W, A1                  ; Arbre Agressif 4/4
071D8    BSR.W $00001B2E                  ; GoTo #Trackloader_Start
                                          ; Bon… jusqu'ici, rien de transcendant
071E0    LEA      46A4.S,A0
071E4    LEA      466E.S,A1
071E8    BSR      4654                    ; Bla bla

071EC    LEA      4710.S,A0
071F0    LEA      46EA.S,A1

071F4    BSR      4654                    ; Bla bla
071F8    LEA      4790.S,A0               ; Bla bla
071FC    LEA      476A.S,A1
07200    BSR      4654

07204    LEA      491A.S,A0
07208    LEA      4904.S,A1
0720C    BSR      4654                    ; Bla bla

07210    MOVE.W #$00DC, $00007F82
========================================  ; 1er Routine de CRC
07218    LEA $000099A8, A0                ; A0=99A8
0721E    LEA $00009D4A, A1                ; A1=9D4A
07224    MOVEQ #$00, D0

07226    ADD.W (A0)+, D0                  ; → Oh la jolie boucle de calcul d'un CRC.
07228    CMPA.L A0, A1                    ;
0722A    BGT.B $00007226                  ; ← En gros, on boucle tant qu'on est pas arrivé jusqu'à A1 défini plus haut
                                          ;    à savoir 9D4A. Impact sur le code ? Aucune idée mais par précaution on
                                          ;    va la désactiver.


0722C    CMP.W #$3D69, D0                 ; On compare le total, D0 avec 3D69
                                            ce qui si c'est bon, nous donnera un couple A0, A1 et D0 attendu

07230    BNE.B $00007226                  ; ← Si ce n'est pas égal (c'est que le CRC est pas validé donc)
                                            on branche en 7226 (et c'est repartie pour un petit tour.)
========================================
```

Pour désactiver ce 'check' on va faire simple, au lieu de tester **D0** on va le définir.
Donc on remplace :
~~0722C CMP.W #$3D69, D0~~ ; OPCODE = **B0 7C 3D 69**
Par
0722C MOVE.W #$3D69,D0
0730 NOP

Il est assez courant qu'une routine de CRC en cache une autre (bref, en général il y en a plus d'une et il est assez courant que les 'autres' soient calquées sur le même fonctionnement que celle préalablement trouvée.

Donc on cherche en mémoire si il y en a pas une autre du même type.
Tapez : F B0 7C 3D 69 → **722C** (celle que l'on vient de trouver) et **7D52**
On jette un coup d'œil sur la routine en **7D52**

```
========================================
07D3E    LEA $000099A8, A0              ; Exactement la même routine de CRC vue en 7218
07D44    LEA $00009D4A, A1              ; lol
07D4A    MOVEQ #$00, D0                 ;
07D4C    ADD.W (A0)+,D0                 ;
07D4E    CMPA.L A0, A1                  ;
07D50    BGT.B $00007D4C                ;
07D52    CMP.W #$3D69, D0               ;
07D56    BNE.B $00007D4C                ;
========================================
```

Bon...même solution.
On remplace :
~~07D52 CMP.W #$3D69, D0~~
Par
07D52 MOVE.W #$3D69,D0
07D56 NOP

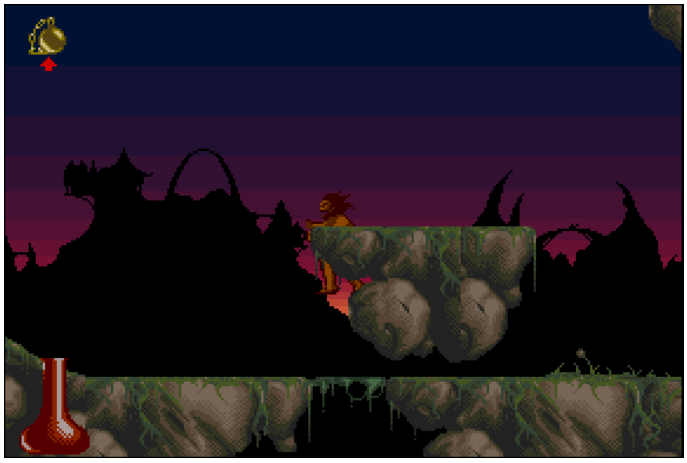
Et on continue notre analyse et pas besoin d'aller bien loin pour voir encore quelque chose d'intéressant (décidément)

```
07232    CMPI.W #$0035, $0C20.W         ; Comparer le Pointeur de 'piste' en cours avec la valeur $35 (piste !53)
07238    BNE.B $0000724A                ; Non ? GoTo 724A
0723A    BSR.W $00007A90                ; Oui ? GoTo 7A90
0723E    JSR $1ACA.W
07242    LEA $0256.W, A7
07246    BRA.W $00002758
```

Alors ici, on ne va pas regarder les sous-routine en 724A et 7A90, on notera juste, et c'est IMPORTANT, que le code effectue des tests sur le *marqueur de piste* en cours (aka $C20).
ça c'est TRES PROBLématique car comme nous allons recréer complétement nos disks de crack sur une disquette AmigaDOS standard et donc Fonctionner en side 'normal' et pas que sur une side comme le trackload d'origine.
De plus, il est plus que probable que nous allons bouger aussi nos données, elles ne vont plus se retrouver forcément au même endroit que l'original.

Nous verrons plus tard (en fin de cette section), comme bypasser ce type de 'protection' (bbaaa si, c'en une protection).

Bon...on retourne au jeu et on se dirige vers Crystal Cavern tout à droite ? Je pense que c'est une bonne idée de regarder le code à cette endroit.
Tapez : BDA puis **X** et diriger vous tout à droite.



Sautez plusieurs fois pour faire céder le sol sous vos pieds.

Continuez la descente des escaliers qui va nous mener tout droit au '**Crystal Level**' et nous demander le Disk1.

Et on entre dans l'AR, on regarde où l'on se trouve et qu'elle est l'adresse de retour stocké dans la Pile. **Tapez : R**
Sans surprise on se trouve en **8CE0** dans la routine de **#DISK2_OR_DISK1_INSERTED?**
Avec une adr. de retour située en **7254**, il est logique de s'attendre au chargement en cascade des 11 'fichiers' de ce Level, sauf que… pas forcément.

```
07254    BSR.W $00009D3E
07258    MOVE.W #$01A0, $00DFF096
07260    LEA $000515DC, A0
07266    BSR.W $00009B24                ; GoSub ➔ #RAZ_TRACKLOADER
                                        ; /!\ Routine à patcher car l'ensemble de ces Sous-routines n'existeront plus.

0726A    BNE.B $000072B0                ; Tiens, un test sur le Flag Z, qui, le cas échéant branchera en 72B0
                                        ; On trouve en 72B0, c'est le trackload de Crystal Cavern #03/11
                                        ; Donc si ce test est valide, on trackloadera directement la Part 03/11
                                        ; sans passer par la 01/11 et 02/11
                                        ;
                                        ; Dans notre crack, c'est à coup sûr le Bug garanti puisque l'on va patcher
                                        ; Les routines en 72B0 et donc pas de modification de Z
                                        ; Il faudra le désactiver/modifier
0726C    MOVE.B #$01, $0000F3BE
07274    MOVEQ #$02, D0
07276    MOVEQ #$01, D1
07278    LEA $1626.W, A1
0727C    LEA $00044EFE, A2

07282    BSR.W $00007E58                ; Chargement du 1er 'fichiers'
         ;-------------------------------------------------------------
         07E58    MOVE.B D0, $02C9.W
         07E5C    MOVE.L $02CA.W, $0C1A.W
         07E62    LEA $00030000, A0     ; Crystal Cavern #01/11
         07E68    BSR.W $00001B2E       ; GoTo #Trackloader_Start
         ...
         ;-------------------------------------------------------------

07286    TST.B $0BE0.W
0728A    BEQ.B $00007298
0728C    TST.B $0000F3C2                ; au vue du code, Bof..pas plus parlant que ça celui-là.
07292    BEQ.B $0000728C
07294    CLR.B $0BBE.W                  ; Je n'accorde pas vraiment d'intérêt au test de BBE
07298    TST.B $0BBF.W                  ; ou de BBF, ils sont vraiment trop nombreux pour être une 'protection'
                                        ; proprement dite, cela semble plutôt des marqueurs de chargement déjà fait
0729C    BNE.B $0000728C

0729E    LEA $000515DC, A0              ;
072A4    LEA $162E.W, A1                ; Crystal Cavern #02/11
072A8    BSR.W $00001B2E                ; GoTo #Trackloader_Start
072AC    BSR.W $00001824                ; GoTo #Decomp/Decrypt

072B0    LEA $0004E700, A0              ;
072B6    LEA $1636.W, A1                ; Crystal Cavern #03/11
072BA    BSR.W $00001B2E                ; GoTo #Trackloader_Start
072BE    BSR.W $00001824                ; GoTo #Decomp/Decrypt

072C2    LEA $0006DE1C, A0              ;
072C8    LEA $163E.W, A1                ; Crystal Cavern #04/11
072CC    BSR.W $00001B2E                ; GoTo #Trackloader_Start
072D0    BSR.W $00001824                ; GoTo #Decomp/Decrypt

072D4    LEA $0002E248, A0              ;
072DA    LEA $1646.W, A1                ; Crystal Cavern #05/11
072DE    BSR.W $00001B2E                ; GoTo #Trackloader_Start
072E2    BSR.W $00001824                ; GoTo #Decomp/Decrypt

072E6    LEA $0000A800, A0              ;
072EC    LEA $161E.W, A1                ; Crystal Cavern #06/11
072F0    BSR.W $00001B2E                ; GoTo #Trackloader_Start

072F4    TST.B $0BE0.W
072F8    BEQ.B $00007358
072FA    CLR.B $0BE1.W
072FE    LEA $0000F38A, A0              ;
07304    LEA $164E.W, A1                ; Crystal Cavern #07/11
07308    BSR.W $00001B2E                ; GoTo #Trackloader_Start
0730C    BSR.W $00001824                ; GoTo #Decomp/Decrypt

07310    LEA $0001013A, A0              ;
07316    LEA $1656.W, A1                ; Crystal Cavern #08/11
0731A    BSR.W $00001B2E                ; GoTo #Trackloader_Start
0731E    BSR.W $00001824                ; GoTo #Decomp/Decrypt

07322    LEA $0002AF4A, A0              ;
07328    LEA $165E.W, A1                ; Crystal Cavern #09/11
0732C    BSR.W $00001B2E                ; GoTo #Trackloader_Start

07330    BSR.W $00001958                ; GoTo #Decomp/Decrypt_02
07334    LEA $0002C7B6, A0              ;
0733A    LEA $1666.W, A1                ; Crystal Cavern #10/11
0733E    BSR.W $00001B2E                ; GoTo #Trackloader_Start
07342    BSR.W $00001958                ; GoTo #Decomp/Decrypt_02

07346    LEA $0002D6A6, A0              ;
0734C    LEA $166E.W, A1                ; Crystal Cavern #11/11
07350    BSR.W $00001B2E                ; GoTo #Trackloader_Start
07354    BSR.W $00001958                ; GoTo #Decomp/Decrypt_02
```

On revient quelques minutes sur le test vue sur **C20** juste au-dessus. A savoir donc, un test sur le compteur de 'Piste en cours'.

Tapez : FA 0C20

```
7166    CMPI.W  #A,C20.S            ; Inconnue pour l'instant
7232    CMPI.W  #35,C20.S           ; Celui que l'on a détecté juste au-dessus.
7FCE    CMPI.W  #4F,C20.S           ; Inconnue pour l'instant
9CD6    CLR.W   C20.S               ; Raz du compteur de track, aka #Base_Conf_Interupt,ne nous gêne pas plus que çà.
46F10   CMPI.W  #A,C20.S            ; Inconnue pour l'instant
```

On jette un coup d'œil à ces 3 adresses à l'aide de la commande **D 7166**

```
================================
07166   CMPI.W  #$000A, $0C20.S     ; Check sur le compteur de piste
0716C   BNE     $0000717E           ; GoSub ➔ TrackLoad_ Arbre Agressif 1/4
0716E   BSR     $00007A90           ; On retrouve encore notre sous-routine en 7A90
07172   JSR     $1ACA.S
07176   LEA     $0256.S, A7
0717A   BRA     $00002758
================================

D 717E
================================
0717E   MOVE.W  #$7910, $7036.S
07184   MOVE.B  #$03, $02C9.S
0718A   MOVE.L  $02CA.W, $0C1A.S

07190   LEA     $000515DC, A0       ;
07196   LEA     $16DE.S, A1         ; Arbre Agressif 1/4
0719A   BSR     $00001B2E           ; GoTo #Trackloader_Start
0719E   BSR     $00001824           ; GoTo #Decomp/Decrypt
================================
```

On continu : **D 7FCE** (Vu le code, on remonte au début de la sous-routine, cela semble intéressant.)

```
================================
07FC4   MOVE.W  $02BE.W, D1         ; D1=$2BE
07FC8   CMP.W   #$454A, D1          ; Un check fait sur 2BE
07FCC   BNE     $00007FF4           ; Tiens… suivant l'état de 2BE , il va zapper les tests ci-dessous.

07FCE   CMPI.W  #$004F, $0C20.S     ; Encore un check sur le 'compteur de piste'
07FD4   BNE     $00007FC4
07FD6   CMPI.W  #$4150, $02C0.S     ; Et un autre
07FDC   BGT     $0000805C
07FDE   CMPI.B  #$01, $0BC0.S
07FE4   BEQ     $0000805C
07FE6   MOVEQ   #$01, D0            ; D0=01
07FE8   MOVE.B  D0, $0BC0.S         ; Copie D0 à l'adresse $BC0
07FEC   MOVE.B  D0, $0000FBD4       ; Copie D0 à l'adresse $FBD4
07FF2   RTS
================================
07FF4   CMP.W   #$422C, D1          ; Et encore un test sur 2BE
07FF8   BNE     $00008018
07FFA   CMPI.W  #$4142, $02C0.S     ; Et encore un autre sur le 'compteur de piste'
08000   BGT     $0000805C
08002   CMPI.B  #$04, $0BC0.S
08008   BEQ     $0000805C
0800A   MOVEQ   #$04, D0            ; J'accorde peu d'intérêt
0800C   MOVE.B  D0, $0BC0.S         ; Car il n'est pas que 'tester', il est aussi positionner (comme ici)
08010   MOVE.B  D0, $0000FBDE       ; et les branchements effectués après leurs tests ne sont pas parlants.
08016   RTS
================================
805C    RTS
================================
```

Et le dernier, **D F46F10** (Vu le code, on remonte au début de la sous-routine, cela semble intéressant.)

```
================================
46F10   CMPI.W  #A,C20.S            ; Again, un check sur le 'compteur de piste'
46F16   BNE     46F28
46F18   BSR     4783A
46F1C   JSR     1ACA.S
46F20   LEA     256.S,A7
46F24   BRA     42502

================================
```

Bon…Il est clair que le marqueur en **C20** est important pour le bon fonctionnement du jeu.
Comme vu plus haut, nous allons devoir trouver une méthode pour que l'ensemble des tests fonctionnent bien.
Je n'ai pas trop envie de patcher à l'aveugle les tests fait sur **C20,** de plus qui sait, il y en aura peut-être d'autres !

On a vu aussi un test sur **2BE**, on s'attarde 2s sur celui-ci, **Tapez : M 2BE**
De mon côté, cette valeur est positionnée à **42 C8**

On retourne au code du jeu, on ne touche à rien, on attends quelques secondes et on **retourne dans l'AR** pour regarder de nouveau en **2BE**
La valeur n'a pas bougé.

Ok, on retourne de nouveau au jeu et on avance ou recule un peu, on retourne dans l'AR et on regarde de nouveau la valeur de **2BE**
La valeur a cette fois ci changée.

Vous pouvez faire quelques tests dans ce sens, et même revenir exactement au même endroit physiquement avec votre personnage,
Vous vous apercevrez qu'en fait, le marqueur en **2BE** semble **indiquer une position** du personnage **dans la MAP** du jeu.

Théoriquement, cela ne devrait avoir aucune incidence sur notre crack (une position reste une position même dans un crack).
Néanmoins, c'est bon de connaitre qu'il existe (ce marqueur) et qu'il est positionné en **2BE**

On revient un petit moment sur l'appel vers 9B24 vu juste au-dessus.

```
7266    BSR.W $00009B24              ; GoSub ➔ #RAZ_TRACKLOADER
726A    BNE.B $000072B0              ; Un test sur le Flag Z, qui, le cas échéant branchera en 72B0
                                     ; Qui, comme nous l'avons vu, va faire planter le jeu.
```

Il serait intéressant de savoir quand est appelé la routine vers **#RAZ_TRACKLOADER**
Si après chaque appel à celle-ci, est effectuée un **BNE**, et si c'est le cas, si ce **BNE** part sur un plantage du jeu.
Auquel cas, on serait en présence d'une protection et il faudra regarder plus en détail le code en **9B24**
On commence les investigations, **entrez dans l'AE** pendant une partie et **tapez** : **FA 9B24**
Cela nous donnes : (et on regarde le code qui gravite autour)

```
254E    BSR 9B24        Dans la zone chargement Level#1
7266    BSR 9B24        Dans la zone chargement Crystal_Cavern (vue plus haut)
74E8    BSR 9B24        Dans la zone de chargement vortex (merci au tableau de la mort et A1)
```

TOUS ont un **BNE** après cette instruction, et TOUS avec ce BNE partent sur une partie du code non attendu si c'est le cas (Z=0)

```
245E    Si Z=0 alors il part dans une boucle de la mort de rechargement.
7266    Si Z=0 alors zap tout une partie du chargement.. voir Chapitre 14B du tuto.
74E8    Si Z=0 alors zap aussi tout une partie (voir tout) le chargement du level en question.
```

Je ne sais pas pour vous, mais ça ressemble de plus en plus à une protection cette histoire. On regarde la code en 9B24 ? Let's go !

#RAZ_TRACKLOADER

```
9B24  MOVE.L A0,1818.S     ;
9B28  BSR    9B40          ; GoSub ➔ #Pre_Base_TrackLoadX     A supprimer, c'est déjà intégré dans notre TrackLoader.
9B2C  BSR    9B8A          ; GoSub ➔ #Motor_ON                A supprimer, pareil.
9B30  BSR    9BBA          ; GoSub ➔ #Retour_T00              A supprimer, pareil.
9B34  BSR    9C24          ; GoSub ➔ #Pre_Base_TRK_X2         A supprimer, pareil.
9B38  BSR    9C94          ; GoSub ➔ #Base_Conf_Interupt      On garde que le RAZ de $C20, tout le reste poubelle.
9B3C  BRA    9CDC          ; GoTo ➔ #Traitement_Trait_01      Voir en dessous ➔
```

Rappel de ce que l'on a vu beaucoup plus haut dans l'analyse du Trackloader 3 **part 9 du Tuto**

#Traitement_Trait_01

```
09CDC    MOVE.L   1818.S,A0          ; A0=1818
09CE0    MOVE.L   #3778,D3           ; Compteur en D3
09CE6    MOVE.L   (A0),D0            ; ➔
09CE8    ADDQ.L   #2,A0              ; A0=A0+2
09CEA    MOVE.L   #F,D2              ; Compteur en D2
09CF0    MOVE.L   D0,D1              ;                ➔
09CF2    SWAP     D1                 ;                Inverse en longword D1
09CF4    CMPI.W   #4454, D1          ;                On compare avec 4454 (Word de synchro Custom?)
09CF8    BEQ      9D08               ;                Si identique alors on GoTo ➔ #Traitement_Trait_02
09CFA    ADD.L    D0,D0              ;                D0=D0+D0
09CFC    DBF      D2,9CF0            ;                ➔ D2=D2-1, on boucle tant que D2 est diffèrent de -1
09D00    DBF      D3,9CE6            ; ➔ D3=D3-1, on boucle tant que D3 est diffèrent de -1
09D04    BRA      9D38               ; GoTo ➔ #SET_D0_To_1
```

#Traitement_Trait_02

```
09D08    MOVEQ    #0,D5              ;
09D0A    MOVE.L   (A0),D0            ; ➔ D0=(A0)
09D0C    ADDQ.L   #2,A0              ;   A0=A0+2
09D0E    MOVE.L   #F,D2              ;   Compteur en D2
09D14    MOVE.L   D0,D1              ;                ➔
09D16    SWAP     D1                 ;                 Inverse en longword D1
09D18    CMPI.W   #4454,D1           ;                On compare avec 4454 (Word de synchro Custom?)
09D1C    BEQ      9D2C               ;                Si identique alors on GoTo ➔ #Traitement_Trait_03
09D1E    ADD.L    D0,D0              ;                D0=D0+D0
09D20    DBF      D2,9D14            ;                ➔ D2=D2-1, on boucle tant que D2 est diffèrent de -1
09D24    ADDQ.L   #1,D5              ;    D5=D5+1
09D26    DBF      D3,9D0A            ; ➔ D3=D3-1, on boucle tant que D3 est diffèrent de -1
09D2A    BRA      9D38               ; GoTo ➔ #SET_D0_To_1
```

#Traitement_Trait_03

```
09D2C    SUBI.L   #1A2C,D5                    ; D5=D5-1A2C
09D32    BMI      9D38                        ; Si résultat négatif, alors GoTo ➔ #SET_D0_To_1
09D34    CLR.W    D0                          ; Sinon On efface le Word D0
09D36    RTS                                  ; E.T Retour Maison
```

#SET_D0_To_1

```
09D38    MOVE.W   #1,D0                       ; D0=1
09D3C    RTS                                  ; E.T Retour Maison
```

Bon, c'est bien joli toutes ces sous-routines mais ça fait quoi ? ça n'écrit rien du tout à une adresse mémoire et ça semble plutôt calculer quelque chose et selon le résultat sortir avec **D0**=0 ou **D0**=1

Il serait intéressant de connaitre le déroulement 'normal' de ces sous-routines et de l'état en sortie de **D0** et du Flag Z du coup.
pour ça c'est simple, il suffit de jouer à l'endroit du/des test(s).

254E Chargement Level#1, il suffira de poser un BS en 9D34 et en 9D38 depuis le menu du jeu et de lancer le chargement.
7266 Pareille au niveau des BS sauf que la manip est à faire avant d'entrer dans le niveau Crystal Level
74E8 Bon, la ça va être plus dur car il faut avancer bcp dans le jeu pour arriver à cette étape, vous allez donc me croire sur parole pour celle-ci

Sur chaque *Breakpoint* posé ci-dessus, on est **TOUJOURS** dans la zone **9D34**, jamais on atteint **9D38**
Donc la sortie 'normale' de la routine **#Traitement_Trait_01** en **9CDC** devrait toujours être **D0=0 et le flag Z=1**

Donc, là pour le coup on est sûr, c'est une protection.

On ne va pas continuer plus loin notre investigation et passer directement à la conclusion de ce chapitre.
Nous avons vu :

- 1er routine de CRC 7218 → 07230 à désactiver en 722C
- 2eme routine de CRC 7D3E → 07D56 à désactiver en 7D52
- Fonctionnement du jeux avec des Checks effectuées sur le marqueur de Piste en $C20
- Un **BNE** à faire sauter en 726A ~~BNE.B $000072B0~~
- Un ensemble de sous-routine à désactiver lorsqu'on aura remplacé le trackloader d'origine.
- Marqueur de position dans la MAP en $2BE
- Tout un ensemble de sous-routines à désactiver dans $9B24
- Mais garder dans celle-ci un **RAZ** de $C20, et un **RAZ** de **D0** (**/!**s'assurer d'avoir un **Z=0** en sortie)

L'ensemble de ces modification seront à effectuer lors du HACK des données **Last_Load_Disk1**

Pour le fonctionnement global du jeu avec le marqueur de piste en C20,
il serait **plus simple** de **garder les positions d'origines** relevées dans notre tableau de la mort listant tous les appels.
Rappel : 1er long word de (A1), A1 qui contient l'adresse du tableau pour le prochain Trackload.

Ensuite..à voir selon le test de notre crack, il y aura peut-être d'autre choses à modifier ?!

Part 15 Réorganisation des données pour création de nos disks

L'idée : Essayer de créer une version Cracké contenant l'intégralité du jeu et de ses animations le tout sur 2 disquettes.

Le hic, c'est qu'en regardant le tableau des appels (Part 11 du tuto), il apparait clair que l'on est ici sur un dépassement significatif des données.
En étudiant ce tableau on peut s'apercevoir que les données sont contiguës et utilisent quasiment l'intégralité des disques. (au passage, incroybale…)

Si on déplace les chargements **Load_Menu_01 -> Load_Menu_07** du disk2 vers le disk1, les données du coup rentreront sur une disquette *AmigaDOS standard*.
Il faudra bien sûr modifier les appels pour ses chargements (mais ils ne sont appelés qu'une fois, au début du chargement du Menu) et modifier aussi l'appel du dernier : **Load_Menu_08**
Utiliser un trackloader 'standard' en modifiant sont entrée pour utiliser exactement les mêmes types d'appels et utiliser les tables d'origines contenues en mémoire voir modifier certains appels pour coller à notre disk.
Ça fera beaucoup moins de travail à faire d'une part, et cela sera bcp plus simple (en théorie)

Tous ses appels sont effectués via le dernier chargement après les animations, à savoir : **Last_Load_Disk1** qui contient clairement du code et pas des Data, il faudra le modifier.
Cela nous donne ceci :

DISK2 – Notre version cracké

					ORIGINAL		Position dans		
Ordre D'appel	Adr. D'appel	LENGTH	Decomp/ Decrypt	Chaine Hexa // Signature	Disk	SIDE	Disk Réel	Fichier Rippé	Notre Disk Cracké
Load_Menu_11	1EA0	0271C	BSR 1824	0000271DF050378932041AB57E0081AF0503A82E68250321D001030407C05839	D2	LOW	04C18-07333	04C18-07333	00000-0271B
Load_Menu_12 In_Game_50	1CF0	007FC	BSR 1824	000007FD55536181F6C1031007350001FA0F01054BEA7C121795BC00D821E107	D2	LOW	07334-07B2F	07334-07B2F	0271C-02F17
Load_Menu_13 In_Game_51	1D02	0E9D8	BSR 1824	0000E9D9B686A6042EEEF60446D60E20E0EA16717274B3B734631FA61C8AA478	D2	LOW	07B30-16507	07B30-16507	02F18-118EF
Load_Menu_14 In_Game_52	1D14	01408	BSR 1824	00001408EFF1915FC81C0C34284FF8280FFF84107F800070797962B70F2F9004	D2	LOW	16508-1790F	16508-1790F	118F0-12CF7
Load_Menu_15 In_Game_53	1D26	00E10	BSR 1958	000E0D18191A150C02FEFDFE9A010200FDFAF9F7F8FD00030403020200010408	D2	LOW	17910-1871F	17910-1871F	12CF8-13B07
Load_Menu_16 In_Game_54	1D38	00B48	BSR 1958	000B48F6F7F7FC00030A110D17161A2120212016120E0702FEF6F2EEE8E8E6E4	D2	LOW	18720-19267	18720-19267	13B08-1464F
Load_Level1_01	24DE	0CAA8	BSR 1824	0000CAA90069A08715B3F1AC0E03FE53F1FC09DE3F0E71DA0EB103F10FA1824F	D2	LOW	19268-25D0F	19268-25D0F	14650-210F7
In_Game_61	762C	0CAA8	BSR 1824						
In_Game_55-RET	7800	0CAA8	BSR 1824						
In_Game_31-A	7966	0CAA8	BSR 1824						
In_Game_44	7C80	0CAA8	BSR 1824						

Load_Level1_02	24F0	00E84	BSR 1824	00000E8598683F015B0D07E034E1A0FC063C341E403C2400A077E140F80422BC	D2	LOW	25D10-26B93	25D10-26B93	210F8-21F7B
In_Game_63	763E	00E84	BSR 1824						
In_Game_45	7C9A	00E84	BSR 1824						
In_Game_56-RET	7812	00E84	BSR 1824						
In_Game_12	8A9C	00E84	BSR 1824						
Load_Level1_03	2508	0DDC8	BSR 1824	0000DD7976FF8B66043E0C083C00F34501FCC1A641FAFF0712FAA50AE184EA4E	D2	LOW	26B94-3495B	26B94-3495B	21F7C-2FD43
In_Game_46	7CAC	0DDC8	BSR 1824						
In_Game_57-RET	7824	0DDC8	BSR 1824						
In_Game_31-B	7980	0DDC8	BSR 1824						
Load_Level1_04	251A	03C28	Non	428E30004020444A3000402000050000046602130311042205320643075400 01	D2	LOW	3495C-38583	3495C-38583	2FD44-3396B
In_Game_58-RET	7836	03C28	Non						
In_Game_31-C	7992	03C28	Non						
In_Game_48	7CD0	03C28	Non						
Load_Level1_06	2542	003A0	BSR 1824	000003A1002C3FF00801031FFC8080EF8042E7F38080503F9CF8080C7FF97FE4	D2	LOW	38584-38923	38584-38923	3396C-33D0B
In_Game_47	7CBE	003A0	BSR 1824						
In_Game_30	8244	003A0	BSR 1824						
In_Game_22	84C2	003A0	BSR 1824						
In_Game_14	8AC0	003A0	BSR 1824						
In_Game_65	7668	003A0	BSR 1824						
Load_Level1_05	2530	01178	BSR 1824	00001178C20F103FFF08100C7038C5770A0417D9AC1F6F7E2FCFFF7CBA435DF0	D2	LOW	38924-39A9B	38924-39A9B	33D0C-34E83
In_Game_49	7CDE	01178	BSR 1824						
In_Game_31-D	79A0	01178	BSR 1824						
In_Game_18	7A3C	01178	BSR 1824						
In_Game_10	8A70	04F08	BSR 1824	00004F0900B2A10715B3F1AC0E03FE53F1FC09DE3F0E71DA0EB103F10FA1824F	D2	LOW	39A9C-3E9A3	39A9C-3E9A3	34E84-39D8B
In_Game_01	719A	04F08	BSR 1824						

In_Game_02	71AC	054A0	BSR 1824	000054A1D4FFF00021891B032AFD0E73C00180E0390B65BFDCAFDFFE17900203	D2	LOW	3E9A4-43E43	3E9A4-43E43	39D8C-3F22B
In_Game_11	8A82	054A0	BSR 1824						
In_Game_29	8232	054A0	BSR 1824						
In_Game_21	84B0	054A0	BSR 1824						
In_Game_23	7AD0	0B26C	BSR 1824	0000B26C0708A07FD8507086060A07086261C0708A068003C180BC84A0FD01C2	D2	LOW	43E44-4F0AF	43E44-4F0AF	3F22C-4A497
In_Game_13	8AAE	0B26C	BSR 1824						
In_Game_03	71C6	0B26C	BSR 1824						
In_Game_04	71D8	03A34	BSR 8C0A Non	39F43000400000003000400000030000046603120423062308430964 0A870000	D2	LOW	4F0B0-52AE3	4F0B0-52AE3	4A498-4DECB
In_Game_24	7AE2	03A34	Non						
In_Game_15	7A0A	07DC8	BSR 1824	00007DC9F6103400606800400506401 20C043FD8087FB010E16041B0C883C1B8	D2	LOW	52AE4-5A8AB	52AE4-5A8AB	4DECC-55C93
In_Game_16	7A1C	041C8	BSR 1824	000041C97C2938E07BCEF033E181DF8060404F93FE73E60019F3800CF9C0067C	D2	LOW	5A8AC-5EA73	5A8AC-5EA73	55C94-59E5B
In_Game_17	7A2E	045EC	Non	348C400040B030E6300040600001000004660314021203220432054206550000	D2	LOW	5EA74-6305F	5EA74-6305F	59E5C-5E447
In_Game_19	8360	0353C	BSR 1824	0000353D1A34BFE021A3037FC0FF57C494A1A103281BD90D281BBE0C24243420	D2	LOW	63060-6659B	63060-6659B	5E448-61983
In_Game_28	8150	00588	BSR 1824	00000589301A3A4016510360004 6DF1815807901CFC40D80011BACFEC908B220	D2	LOW	6659C-66B23	6659C-66B23	61984-61F0B
In_Game_20	8372	00588	BSR 1824		D2	LOW	6659C-66B23		
In_Game_25	811A	01CE4	BSR 1824	00001CE58A713BA1D1B45181A3488040F452499015D200E0868B5881811683E6	D2	LOW	66B24-68807	66B24-68807	61F0C-63BEF
In_Game_26	812C	001E0	BSR 1824	000001E1BF00A3648003095F041DF95C07E206A003D2C81FF54A18657FF21CB5	D2	LOW	68808-689E7	68808-689E7	63BF0-63DCF
In_Game_27	813E	00B5C	BSR 1824	00000B5D0F77FC7DE6F729300E96E4FFF26F7503007DEFDEC3C5A1618821C01E	D2	LOW	689E8-69543	689E8-69543	63DD0-6492B
In_Game_05 In_Game_09-RET In_Game_59 In_Game_60	7E68	02EEC	BSR 1826	00002EEDE25501646D52E50701E29085D2095985629099504420E14940000AA9	D2	LOW	69544-6C42F	69544-6C42F	6492C-67817
In_Game_06	8918	02C44	BSR 1824	00002C4538E3F3FFE0E383806071DAB43E7037F0206FD01C2FE880C81F3E0C2F	D2	LOW	6C430-6F073	6C430-6F073	67818-6A45B
In_Game_07	892A	00B70	BSR 1824	00000B71644137B830701787CF8FF87DDB4FC313F33FDC1FFCF03FEFF7905F8E	D2	LOW	6F074-6FBE3	6F074-6FBE3	6A45C-6AFCB
In_Game_08	893C	0011C	BSR 1824	0000011DC300643E080808010380A07DFFC680303342E80303B42F40340181F4	D2	LOW	6FBE4-6FCFF	6FBE4-6FCFF	6AFCC-6B0E7
In_Game_54b	7E68	04228	BSR 1824	0000428980202178 01A4D81E55060FF8B0762BC0E078AFC1D11789822C62807C	D2	LOW	6FD00-73F87	6FD00-73F87	6B0E8-6F36F
In_Game_60	74FA	04228	BSR 1824	0000422970303260606 8FD40E401681C0C20180040E405684B0640281980B008	D2	LOW	73F88-781AF	73F88-781AF	6F370-73597

In_Game_61	750C	001D0	BSR 1824	000001D1C2163A3680184590C0CA01D1F604831789076F140C12B786064F040D	D2	LOW	781B0-7837F	781B0-7837F	73598-73767
In_Game_62	751E	00628	BSR 1824	00000629C02705880D3AA07C089084AF87FBFA7FE13406807FEBE05920806002	D2	LOW	78380-789A7	78380-789A7	73768-73D8F
In_Game_67	768C	0B00C	BSR 1824	0000B00D9227AD0DFE9BFD3BFA67F617ED17DC67B3C5B16B6757786ED71EDB9F	D2	UP	85D04-90D0F	00000-0B00B	73D90-7ED9B
In_Game_57	709E	0B00C	BSR 1824						
In_Game_58	70B0	030D4	Non	5040300040F85124300040F8000100000466021203220432064307650888000 1	D2	UP	90D10-93DE3	0B00C-0E0DF	7ED9C-81E6F
In_Game_56	708C	00478	BSR 1824	00000479B08A01039602079C040E58081B20103960206C8040D4C081B2010353	D2	UP	93DE4-9425B	0E0E0-0E557	81E70-822E7
In_Game_66	767A	00478	BSR 1824						
In_Game_55	7074	05914	BSR 1826	0000591594AD694FAD804930120486007B3977B3FBFF1A100D24621561DECCFF	D2	UP	9425C-99B6F	0E558-13E6B	822E8-87BFB
In_Game_64	7650	05914	BSR 1826						
N/A	78E4	05914	BSR 1826				9425C-99B6F		
END_01	4F7E	00308	BSR 1824	00000309C54888005AA3BF41E01008040DF2801B043CC054020D489203280920	D2	UP	99B70-99E77	13E6C-14173	87BFC-87F03
END_02	4F90	32A84	Non	67616D656F7665720000000000000000000000000623100000000000000000000	D2	UP	99E78-CC8FB	14174-46BF7	87F04-BA987
END_03	5010	0F778	Non	060602FE040A0E120E060404FCFC060A0E0E08060804FCFE02060E0A08080802 (-B0)	D2	UP	CC8FC-DC073	46BF8-5636F	BA988-CA0FF
END_04	501E	06E04	BSR 1958	006E01073FCF8FDE3CE551901E005A0F8F021FBFF9CAA3201B005A01FC01FE02	D2	UP	DC074-E2E77	56370-5D173	CA100-D0F03
In_Game_68	78B0	04C98	BSR 1824	00004C981087B1FBEF784E70A83D19E00C30C050614020109787F2EF78780D15	D2	UP	E2E78-E7B0F	5D174-61E0B	D0F04-D5B9B
Load_Menu_09	1D5E	00060	BSR 1824	000000603790606040FF807848B5D9E050211706142C24000609274211C42570	D2	UP	E7B10-E7B6F	61E0C-61E6B	D5B9C-D5BFB
In_Game_69 [END]	85EE	04520	BSR 1824	00004521C3140C56F148B57CE0B2349645E52DD0E0698660185BB9C2C02CF1C8	D2	UP	E7B70-EC08F	61E6C-6638B	D5BFC-DA11B
		4F03		ZONE NON UTILISEE ~20Ko			EC090-F0F93		
Load_Menu_08	1E70	01EBC	Non	4AB8041E67162078041E20B804262078042220B8042A700021C0041E41F80B1A	D2	UP	F0F94-F2E4F	6B290-68247	DA11C-DBFD7

Comme indiqué dans le tableau ci-dessus, l'on va copier tout simplement les données ripées du **Disk2** sur notre **Disk2** mais l'on ne va pas partir de la position **raw** $4C18 mais $0000
On va juste bypasser la **ZONE NON UTILISEE** en fin de disquette (*EC090-F0F93*) afin de pouvoir faire entrer toutes nos données sur notre disk.

Comme on peut le remarquer, la différence est donc de **-4C18**.
Il faudra donc faire cette soustraction avant notre routine de *TrackLoad* ou bien sûr, directement dans le code de notre *TrackLoader* 'solution plus simple'
à noter qu'à partir des appels originaux de la seconde face (UP), on se prend un **delta** plus grand et c'est normal.
Delta qui correspond au marqueur de fin du disque d'origine : *$85D04* (revoir **Part 12a Rip Full Disk1** si nécessaire) moins l'adresse du début des prochaines données : *$73D8F* (voir tableau ci-dessus)
Ce qui nous donne : **85D04-73D8F=$11F74**

A voir plus tard comment on peut gérer ces décalages simplement avec des **CMP**.

DISK1 – Notre version cracké

					ORIGINAL		Position dans		
Ordre D'appel	Adr. D'appel	LENGTH	Decomp/ Decrypt	Chaine Hexa // Signature	Disk	SIDE	Disk Réel	Fichier Rippé	Notre Disk Cracké
Boosecteur + code									
Load_Menu_10	1E8E	04C18	BSR 1824	00004C19441E01FB52EEBFB9E1A93F80945B24B05FC0EB206DBC0074CF981007	D2	LOW	00000-04C18	00000-04C18	00400-05017
Load_Menu_01	1C66	03A90	BSR 1824	00003A91C46244304A924783188E0649088F1EB8400482075E0B6AA411301800	D2	UP	EC090-EFB1F	6638C-69E1B	05018-08AA7
Load_Menu_02	1C78	00250	BSR 1824	00000250C1C0303200A06617C008008188070338O1003A3ADBF0040020CF803F	D2	UP	EFB20-EFD6F	69E1C-6A06B	08AA8-08CF7
Load_Menu_03	1C8A	00438	BSR 1824	0000043980D7046478DA0F679AE33C6E0448C02100B33BDCC79A03DDDB9EBE7D	D2	UP	EFD70-F01A7	6A06C-6A4A3	08CF8-0912F
Load_Menu_04	1C9C	00124	BSR 1824	000001242180904B8A0A2188008C4042100048C0022183430283888A8181CD83	D2	UP	F01A8-F02CB	6A4A4-6A5C7	09130-09253
Load_Menu_05	1CAE	00664	BSR 1826	00000665E08050002000380BC0003E024005BD05C203FE030107FE8A81077F99	D2	UP	F02CC-F092F	6A5C8-6AC2B	09254-098B7
Load_Menu_06	1CC6	00268	BSR 1824	00000269AC054004C00BDEA00BC9A8A01B90782D2C023041CA05B443CC737227	D2	UP	F0930-F0B97	6AC2C-6AE93	098B8-09B1F
Load_Menu_07	1CD8	003FC	BSR 1824	000003FDE5FF07F9078B40A078967827ED28B08F40700463F81FB0904C6C0482	D2	UP	F0B98-F0F93	6AE94-6B28F	09B20-09F1B
In_Game_37	72F0	04074	Non	483E40704170454C40E04150000B000004660013002501360347056808980000	D1	LOW	76FC0-7B033	75724-79797	09F1C-0DF8F
In_Game_43	7C44	0280C	Non	0000280D1FA620D8E704C7069F08E04AF1D007055F823F9D84811F065C960EC0	D1	UP	CFF58-D2763	4A254-4CA5F	0DF90-1079B
In_Game_32	7E68	0280C	BSR 1826						
In_Game_33	72A8	0C3C8	BSR 1824	0000C3C90036A00C583E4000C8C80064780033E8B7761A16401908000CAA8DFD	D1	UP	D2764-DEB2B	4CA60-58E27	1079C-1CB63
In_Game_34	72BA	00524	BSR 1824	0000052500683F6BFC1A029020908888F88A007F840E515082EB1D336034833A	D1	UP	DEB2C-DF04F	58E28-5934B	1CB64-1D087
In_Game_35	72CC	0151C	BSR 1824	0000151D45FDFF008B81FC01010120D7E7E01149DC1DFA8278761FCE6A24F9D0	D1	UP	DF050-E056B	5934C-5A867	1D088-1E5A3
In_Game_36	72DE	0A73C	BSR 1824	0000A73D2D07F1E3E20380E61205A207AA07825B781F3910F01F801FFC3FF96F	D1	UP	E056C-EACA7	5A868-64FA3	1E5A4-28CDF
In_Game_38	7308	007F0	BSR 1824	000007F1D55B6181EEC1031002A0807E83C0021368881FE49F0036087845C804	D1	UP	EACA8-EB497	64FA4-65793	28CE0-294CF
In_Game_39	731A	10410	BSR 1824	00010411B6A6A680A40303CC02409823C1203818701B2C62C8710E8164110041	D1	UP	EB498-FB8A7	65794-75BA3	294D0-398DF
In_Game_40	732C	01828	BSR 1958	0018262AC0E5EADFF6F6EE00FF13EF2823215040121F28281F1F152222003BF5	D1	UP	FB8A8-FD0CF	75BA4-773CB	398E0-3B107
In_Game_41	733E	00E10	BSR 1958	000E0D18191A150C02FEFDFE9A010200FDFAF9F7F8FD0003040302020001040 8	D1	UP	FD0D0-FDEDF	773CC-781DB	3B108-3BF17
In_Game_42	7350	00B48	BSR 1958	000B48F6F7F7FC00030A110D17161A2120212016120E0702FEF6F2EEE8E8E6E4	D1	UP	FDEE0-FEA27	781DC-78D23	3BF18-3CA5F

Ensuite viendra à la suite, les animations.
De préférence d'abord **Anim_part#01** et **Anim_part#02** car nous savons qu'il n'y a aucune modification à faire dans celles-ci donc nous connaissons exactement leurs tailles compactées.
Alors que dans **Anim_Psygnosis** et **Last_Load_Disk1**, on en aura à faire. Il est préférable alors de les laisser en dernier, cela sera plus simple si on doit faire plusieurs essais.
On aura juste à réécrire ces deux dernièrs 'fichiers'.

Le meilleur taux de compression est obtenu avec le compresseur **RNC PROPACK** dont les sources sont disponible sur aminet, donc impeccable.
Pour info, **Crunchmania** s'en tire aussi pas mal MAIS, pas aussi bien que RNC 😊

Après quelques tests, voilà ce que nous donne la compression en termes de taux/gain. (Je bypasse la partie compression de ces animations avec **PPAMI** sous workbench, la syntaxe est assez simple.)

Info	Size Réel	Size compressé RNC ProPacker	GainCalcul Octets	Extra Nfo
Anim Psygnosis	164 022	133 499	30 523	Chargé par le bootsecteur, pas dans la table du jeu. (Trackdisk.Device)
Anim sotb2 #1	290 677	236 710	53 967	Chargé par code dans Anim_Psygnosis (Trackloader Custom #1)
Anim sotb2 #2	298 460	235 956	62 504	Chargé par code dans Anim_Psygnosis (Trackloader Custom #1)
Last Code	39 800	~~21 532 *~~	~~18 268 *~~	Chargé par code dans Anim_Psygnosis (Trackloader Custom #1)

*** /!** afin de garder une meilleur souplesse pour des modifications éventuelles, l'on ne **compressera PAS les data** 'LAST CODE DISK1' **MAIS** cela reste 'possible'.

On va caler nos fichiers au plus près de chaque Secteur (donc multiple de 512 Octets) afin de gagner le plus de place possible et de garder le fonctionnement avec notre Trackloader Custom #01

Ce qui nous donne finalement :

Info	Ordre D'appel	LENGTH	Chaine Hexa // Signature Decomp	Notre Disk Cracké	Disk Crack	Position Track + Offset
	~~In_Game_42~~	~~00B48~~	~~000B48F6F7F7FC00030A110D17161A2120212016120E0702FEF6F2EEE8E8E6E4~~	~~3BF18-3CA5F~~	~~D1~~	~~N/A~~
	927 Octets de vide			3CA60-3CDFF		
RNC	Anim part #01	39CA6	00046F740030FFFFFFFF0010003400030038004600 1C0014000000000464000A	3CE00-76AA5	D1	Track !44 + $600
	159 Octets de vide			76AA6-76BFF		
RNC	Anim part #02	399B4	00048DDE0030FFFFFFFF0010003400030038004600 1C00140000000004C60000	76C00-B05B3	D1	Track !86 + $800
	75 Octets de vide			B05B4-B05FF		
RNC	Anim Psygnosis	~2097B	0002410535D800359600F813200C62F729F35C0249D2D086CA06102083800C64	B0600-D0F7A	D1	Track !128 + $600
	644 Octets de vide			D0F7B-D11FF		
CODE	Last_Load_Disk1	9B78	60001B887200123900BFE801610651C9FFFC4E75203900000BE2D08008000017	D1200-DAD77	D1	Track !152 + $1400
	4744 Octets de vide à la fin du DISK			DAD78-DC000		

Cela mérite quelques explications.

Rappel : taille d'une Track AmigaDos standard = $1600 ou !5632 taille d'un secteur AmigaDos standard = $200 ou !512 Nbr de secteur dans Track AmigaDos Standard = !11

Nous savons que la fin de nos données ripées de SOTB2 sur notre **Disk1** finissent en Raw.pos **$3CA60**
Afin donc de faciliter les choses avec notre trackloader Custom #01, on va **fonctionner en mode '**secteur', donc la question à se poser est.
Quel est l'adresse du prochain secteur après la position Raw.pos **$3CA60** ?
La réponse serait ... Encore faudrait t-il déjà savoir sur qu'elle Track et secteur on est en position **$3CA60 ... o_O'**
Facile : **$3CA60 / $1600 =** !44,11 Donc déjà, on a la Track c'est **!44**
Ensuite il suffit de calculer le delta : !44*$1600= $3C800
$3CA60 - $3C800 = $260
$260 / $200 = !1,19 (1 sect=512 Octets), Donc 1 secteur complet et encore des données sur un autre secteur, ce qui nous fait un total de 2 secteurs.

Donc assez logiquement**, le prochaine 'secteur'** sera le 3eme dans la Track !44
En fait ça sera le secteur 2 de la Track !44 car on compte les secteurs à partir de Zero. Donc le '3eme secteur' est le secteur N°2

Ce qui nous donne en hexa : (!44 * $1600) + ($200 * 3) = $3C800 + $600 = **$3CE00**
Donc nos premiers Data à enregistrer, à savoir **Anim part #01** devront être écrites à la Raw.pos **$3CE00**
Il suffit d'utiliser la longueur des datas pour calculer la prochaine position **et ainsi de suite.**

A un petit détail prêt, nous devons modifier notre fichier préalablement cracké* : **Anim Psygnosis,** donc nous ne connaissons pas 'exactement' sa taille finale (même si, cela ne va pas varier de bcp)
On va donc laisser 1 secteur de plus libre au cas dans notre tableau le concernant. (Barre Bleue)
Aussi, les prochaines données à écrire après celui-ci, à savoir : **Last_Load_Disk1** ne seront pas écrites 'au prochain secteur libre' mais 'au prochain secteur libre **+ 1**'
*et oui car ce 'tableau' est contenu dans le code **Anim Psygnosis** , nous allons devoir le mettre à jour et le re-compacter.

J'espère avoir été assez clair sur le sujet ☺
Non ? Dommage 😉

Ainsi notre futur tableau de chargement dans le code '**Anim Psygnosis**' sera :

8 Bytes (578 ->58F)

Track N°					
+Offset	En Dec.	Adr DSKPTH (A2)	Raw.Pos	SIZE	INFORMATION
$600	44	7 63 C0	00 03 CE 00	00 03 9C A6	RNC COMP – Anim Part #01
$800	86	06 C7 80	00 07 6C 00	00 03 99 B4	RNC COMP - Anim Part #02
$200	152	05 00 00	00 0D 12 00	00 00 9B 78	CODE DIRECT - Last_LoadDisk1

Part 15b Préparation des fichiers pour la création de notre Disk1 cracké

Insérez dans **DF0** la disquette contant le fichier de rip : **Disk2_Upper.bin**
Et dans **DF1** une nouvelle disquette vierge formatée. **Un petit reboot et on entre dans l'AR**

Tapez :

```
O 00, 1000 80000
LM Disk2_Lower.bin, 1000
SM 1:DISK1_P1, 1000 1000+4C18
```

Swapper la disquette dans **DF0** par celle contenant le fichier : **Disk2_Upper.bin**
Tapez :

```
LM Disk2_Upper.bin, 1000
SM 1:DISK1_P2, 1000+6638C 1000+6638C+4F04                    // $6B28F +1 - $6638C = $4F04
```

Swapper la disquette dans **DF0** par celle contenant le fichier : **Disk1_Lower.bin**
Tapez :

```
LM Disk1_Lower.bin, 1000
SM 1:DISK1_P3, 1000+75724 1000+75724+4074
```

Et pour finir cette section, swapper la disquette dans **DF0** par celle contenant le fichier : **Disk1_Upper.bin**
Tapez :

```
SM 1:DISK1_P4, 1000+4A254 1000+4A254+2EAD0                   // $78D23 + 1 - $4A254 = $2EAD0
```

Et on joint le tout, **Tapez :**

```
O 00, 10000 80000
LM 1:DISK1_P1, 10000
LM 1:DISK1_P2, 10000+4C18
LM 1:DISK1_P3, 10000+4C18+4F04
LM 1:DISK1_P4, 10000+4C18+4F04+4074
SM 1:DISK1_PRE, 10000 4C660
```

```
dir
Directory of (Action Replay Amiga)
      019480  DISK1_P1
      020228  DISK1_P2
      016500  DISK1_P3
      191184  DISK1_P4
1239 blocks free, 29.6 % of disk used
Disk ok
o 00, 10000 80000
Ready.
lm DISK1_P1, 10000
Loading from 010000 to 014C18
Disk ok
lm DISK1_P2, 10000+4C18
Loading from 014C18 to 019B1C
Disk ok
lm DISK1_P3, 10000+4C18+4F04
Loading from 019B1C to 01DB90
Disk ok
lm DISK1_P4, 10000+4C18+4F04+4074
Loading from 01DB90 to 04C660
Disk ok
sm DISK1_PRE, 10000 4C660
Disk ok
```

Part 16 Modification et compilation du Trackloader d'AlphaONE – Phase1

Le trackloader d'AlphanOne version 2004 fera très bien l'affaire, on va juste faire quelques modifications dessus.
On supprime **A5** du code source du Trackloader d'AlphaOne car à ce stade il interfère avec le code original de SOTB2

Rappel de leur fonctionnement respectif :

Fonctionement du trackloader original de SOTB2	Valeur du trackloader AlphaOne
A0=Destination mémoire **A1**=Position tableau	**A0**=Destination mémoire **A1**=Non Utilisé **A2**=DSKPTH **D0**=Longueur à lire **D1**=Numéro de track de départ **D2**=Offset dans la track, ZERO

Trackloader_SOTB2_#01.s

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

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

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

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

TRACKLOADER:
                MOVEM.L  D0-D7/A0-A6,-(A7)          ; Sauve tous les registres
                LEA      $DFF000,A6                 ; Nécessaire
 ;-------------------------------------------------------------------------------------------------
                MOVE.L   (A1),D2                    ; Recup Raw Pos
                MOVE.L   4(A1),D0                   ; Recup Length
                MOVE.L   D2,D1                      ; Copy Raw Pos. in D1
                DIVU.W   #$1600,D1                  ; D1 / Taille réel concernée
                AND.L    #$FF,D1                    ; Nettoyage D1 // TRACKNR.L
                MOVE.L   D1,D3                      ; Copy D1 in D3
                MULU.W   #$1600,D3                  ; Delta
                SUB.L    D3,D2                      ; Calcul Offset   // BYTEOFFSET.L
;-------------------------------------------------------------------------------------------------
                LEA      $BFD100,A4                 ; DRIVESELECT REGISTER
                ;LEA     $BFE001,A5
                MOVEQ    #0,D7                      ; D7 = BYTECOUNTER
                ADD.L    D2,D0                      ; BYTES TO READ + BYTEOFFSET

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

STEPHEADZERO:
                ;BTST    #4, (A5)
                BTST     #4, $BFE001                ; MOVE HEADS TO CYLINDER 0
                BEQ.B    HEADONZERO                 ; ========================
                BSET     #1,(A4)
                BSET     #0,(A4)
                NOP
                NOP
                BCLR     #0,(A4)
                NOP
                NOP
                BSET     #0,(A4)
                BSR.W    DELAY
                BSR.W    DISKREADY
                BRA.B    STEPHEADZERO

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

```
CHOOSEHEADDOWN:   BSET     #2,(A4)

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

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

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

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

```
READREADY:      MOVE.B   #$FD,(A4)              ; SWITCH MOTOR DRIVE 0 OFF
                NOP                             ; ========================
                NOP
                MOVE.B   #$E7,(A4)
                MOVEM.L  (A7)+,D0-D7/A0-A6      ; Restaure la pile
                RTS

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

DISKREADY:
                ;BTST     #5, (A5)
                BTST     #5, $BFE001            ; WAIT FOR DISK-READY
                BNE.B    DISKREADY              ; ====================
                RTS

TRACKLOADERENDE:
```

On tape/charge le tout sous ASM-ONE, on assemble, on sauve le binaire compilé : **TRACKLOADER_AlphaOne_P1**

Pour plus d'info sur l'utilisation d'ASM-one allez regarder dans mes autres tutos disponibles sur ma Home Page.
Par exemple celui de R-Type II

Ou si vous préférez la version déjà compilée, ci-joint sous forme de suite Hexa.

```
48 E7 FF FE 4D F9 00 DF F0 00 24 11 20 29 00 04 22 02 82 FC 16 00 02 81
00 00 00 FF 26 01 C6 FC 16 00 94 83 49 F9 00 BF D1 00 7E 00 D0 82 18 BC
00 7D 4E 71 4E 71 18 BC 00 75 61 00 01 76 08 39 00 04 00 BF E0 01 67 22
08 D4 00 01 08 D4 00 00 4E 71 4E 71 08 94 00 00 4E 71 4E 71 08 D4 00 00
61 00 01 44 61 00 01 4C 60 D4 83 FC 00 02 48 41 4A 41 67 06 08 94 00 02
60 04 08 D4 00 02 48 41 4A 01 67 24 08 94 00 01 08 D4 00 00 4E 71 4E 71
08 94 00 00 4E 71 4E 71 08 D4 00 00 61 00 01 08 61 00 01 10 51 C9 FF DA
61 00 01 08 3D 7C 82 10 00 96 3D 7C 7F 00 00 9E 3D 7C 85 00 00 9E 3D 7C
44 89 00 7E 3D 7C 40 00 00 24 2D 4A 00 20 3D 7C 99 00 00 24 3D 7C 99 00
00 24 3D 7C 00 02 00 9C 08 39 00 01 00 DF F0 1F 67 F6 3D 7C 40 00 00 24
7A 00 22 4A 28 3C 55 55 55 55 0C 59 44 89 66 FA 0C 51 44 89 67 F4 26 11
22 29 00 04 C6 84 C2 84 E3 83 86 81 E0 9B B6 05 67 08 D3 FC 00 00 04 3E
60 D8 D3 FC 00 00 00 38 2C 3C 00 00 00 7F 22 29 02 00 26 19 C6 84 C2 84
E3 83 86 81 B4 87 6E 06 48 43 30 C3 48 43 54 87 B4 87 6E 02 30 C3 54 87
B0 87 6F 00 00 40 51 CE FF D6 52 05 0C 05 00 0B 66 90 08 14 00 02 67 08
08 94 00 02 60 00 FF 3A 08 D4 00 02 08 94 00 01 08 D4 00 00 4E 71 4E 71
08 94 00 00 4E 71 4E 71 08 D4 00 00 61 00 00 18 60 00 FF 16 18 BC 00 FD
4E 71 4E 71 18 BC 00 E7 4C DF 7F FF 4E 75 28 3C 00 00 25 00 51 CC FF FE
4E 75 08 39 00 05 00 BF E0 01 66 F6 4E 75
```

Part 17 Compilation de la routine de Compression RNC PRO-PACK de Rob North.

Comme expliqué précédemment, il est tout simplement impossible de proposer un crack du jeu fonctionnant sur deux disquettes sans passer par une phase de compression de certains fichiers. Pour cela nous avons utilisé **RCN pro-pack méthode 1** pour la compilation de nos fichiers.
Il est maintenant temps de compiler la routine de décompression que l'on utilisera pour notre crack.
Nous allons l'adapter à nos besoins.

RNC_1.S

```
*---------------------------------------------------------------------------
* PRO-PACK Unpack Source Code - MC68000, Method 1
*
* Copyright (c) 1991,92 Rob Northen Computing, U.K. All Rights Reserved.
*
* File: RNC_1.S
*
* MOD pour adressage absolue de la taille et position du 'tampon'
* Date: 24.3.92
*---------------------------------------------------------------------------

*---------------------------------------------------------------------------
* Conditional Assembly Flags
*---------------------------------------------------------------------------

CHECKSUMS       EQU     0               ; set this flag to 1 if you require
                                        ; the data to be validated

PROTECTED       EQU     0               ; set this flag to 1 if you are unpacking
                                        ; a file packed with option "-K"

*---------------------------------------------------------------------------
* Return Codes
*---------------------------------------------------------------------------

NOT_PACKED      EQU     0
PACKED_CRC      EQU     -1
UNPACKED_CRC    EQU     -2

*---------------------------------------------------------------------------
* Other Equates
*---------------------------------------------------------------------------

PACK_TYPE       EQU     1
PACK_ID         EQU     "R"<<24+"N"<<16+"C"<<8+PACK_TYPE
HEADER_LEN      EQU     18
MIN_LENGTH      EQU     2
CRC_POLY EQU    $A001

;RAW_TABLE      EQU     0               ; désactivé. On va fonctionner en absolue

;POS_TABLE      EQU     RAW_TABLE+16*8  ; désactivé. On va fonctionner en absolue

POS_TABLE       EQU     $6A220          ; remplacé par : Adr. de la zone tampon en absolue
                                        ; En l'occurrence en $6A220, voir explication à la fin de ce chapitre.

LEN_TABLE       EQU     POS_TABLE+16*8  ; Ne change pas.

;               IFEQ    CHECKSUMS       ; Non utilisé
BUFSIZE         EQU     16*8*3          ; 384 Octets devrait effectivement suffire.

;               ELSE                    ; Non utilisé
;BUFSIZE        EQU     512             ; Non utilisé
;               ENDC                    ; Non utilisé

counts          EQUR    d4
key             EQUR    d5
bit_buffer      EQUR    d6
bit_count       EQUR    d7

input           EQUR    a3
input_hi EQUR   a4
output          EQUR    a5
output_hi       EQUR    a6

*---------------------------------------------------------------------------
* Macros
*---------------------------------------------------------------------------

getrawREP       MACRO
getrawREP2\@
                move.b  (input)+,(output)+
                IFNE PROTECTED
                eor.b   key,-1(output)
                ENDC
                dbra    d0,getrawREP2\@
                IFNE PROTECTED
                ror.w   #1,key
                ENDC
                ENDM
```

```
*------------------------------------------------------------------------------
* PRO-PACK Unpack Routine - MC68000, Method 1
*
* on entry,
*        d0.l = packed data key, or 0 if file was not packed with a key
*        a0.l = start address of packed file
*        a1.l = start address to write unpacked file
* on exit,
*        d0.l = length of unpacked file in bytes OR error code
*                  0 = not a packed file
*                 -1 = packed data CRC error
*                 -2 = unpacked data CRC error
*
*        all other registers are preserved
*------------------------------------------------------------------------------

Unpack
                movem.l  d0-d7/a0-a6,-(sp)
                lea      -BUFSIZE(sp),sp
                move.l   sp,a2
                movea.l  A0,A1             ; Add // Adr de destination = Adr source

                IFNE PROTECTED
                move.w   d0,key
                ENDC

                bsr      read_long
                moveq.l  #NOT_PACKED,d1
                cmp.l    #PACK_ID,d0
                bne      unpack16
                bsr      read_long
                move.l   d0,BUFSIZE(sp)
                lea      HEADER_LEN-8(a0),input
                move.l   a1,output
                lea      (output,d0.l),output_hi
                bsr      read_long
                lea      (input,d0.l),input_hi

                IFNE     CHECKSUMS
                move.l   input,a1
                bsr      crc_block
                lea      -6(input),a0
                bsr      read_long
                moveq.l  #PACKED_CRC,d1
                cmp.w    d2,d0
                bne      unpack16
                swap     d0
                move.w   d0,-(sp)
                ENDC

                clr.w    -(sp)
                cmp.l    input_hi,output
                bcc.s    unpack7
                moveq.l  #0,d0
                move.b   -2(input),d0
                lea      (output_hi,d0.l),a0
                cmp.l    input_hi,a0
                bls.s    unpack7
                addq.w   #2,sp

                move.l   input_hi,d0
                btst     #0,d0
                beq.s    unpack2
                addq.w   #1,input_hi
                addq.w   #1,a0
unpack2
                move.l   a0,d0
                btst     #0,d0
                beq.s    unpack3
                addq.w   #1,a0
unpack3
                moveq.l  #0,d0
unpack4
                cmp.l    a0,output_hi
                beq.s    unpack5
                move.b   -(a0),d1
                move.w   d1,-(sp)
                addq.b   #1,d0
                bra.s    unpack4
unpack5
                move.w   d0,-(sp)
                add.l    d0,a0
                IFNE PROTECTED
                move.w   key,-(sp)
                ENDC
```

```
unpack6
                lea     -8*4(input_hi),input_hi
                movem.l (input_hi),d0-d7
                movem.l d0-d7,-(a0)
                cmp.l   input,input_hi
                bhi.s   unpack6
                sub.l   input_hi,input
                add.l   a0,input
                IFNE PROTECTED
                move.w  (sp)+,key
                ENDC

unpack7
                moveq.l #0,bit_count
                move.b  1(input),bit_buffer
                rol.w   #8,bit_buffer
                move.b  (input),bit_buffer
                moveq.l #2,d0
                moveq.l #2,d1
                bsr     input_bits
unpack8
                move.l  a2,a0
                bsr     make_huftable
;               lea     POS_TABLE(a2),a0        ; On ne fonctionne plus avec le couple POS_TABLE(A2)
                lea     POS_TABLE,a0            ; mais directement avec POS_TABLE
                bsr     make_huftable
;               lea     LEN_TABLE(a2),a0        ; Idem
                lea     LEN_TABLE,a0            ;
                bsr     make_huftable
unpack9
                moveq.l #-1,d0
                moveq.l #16,d1
                bsr     input_bits
                move.w  d0,counts
                subq.w  #1,counts
                bra.s   unpack12
unpack10
;               lea     POS_TABLE(a2),a0        ; Idem
                lea     POS_TABLE,a0            ;
                moveq.l #0,d0
                bsr.s   input_value
                neg.l   d0
                lea     -1(output,d0.l),a1
                move.l  a1,$dff180              ; ADD COLOR BAR
;               lea     LEN_TABLE(a2),a0        ; On ne fonctionne plus avec le couple LEN_TABLE(A2)
                lea     LEN_TABLE,a0            ; mais directement avec LEN_TABLE
                bsr.s   input_value
                move.b  (a1)+,(output)+
unpack11
                move.b  (a1)+,(output)+
                dbra    d0,unpack11
unpack12
                move.l  a2,a0
                bsr.s   input_value
                subq.w  #1,d0
                bmi.s   unpack13
                getrawREP
                move.b  1(input),d0
                rol.w   #8,d0
                move.b  (input),d0
                lsl.l   bit_count,d0
                moveq.l #1,d1
                lsl.w   bit_count,d1
                subq.w  #1,d1
                and.l   d1,bit_buffer
                or.l    d0,bit_buffer
unpack13
                dbra    counts,unpack10
                cmp.l   output_hi,output
                bcs.s   unpack8

                move.w  (sp)+,d0
                beq.s   unpack15
                IFNE    CHECKSUMS
                move.l  output,a0
                ENDC
unpack14
                move.w  (sp)+,d1
                IFNE    CHECKSUMS
                move.b  d1,(a0)+
                ELSEIF
                move.b  d1,(output)+
                ENDC
                subq.b  #1,d0
                bne.s   unpack14
```

```
unpack15

                IFNE    CHECKSUMS
                move.l  BUFSIZE+2(sp),d0
                sub.l   d0,output
                move.l  output,a1
                bsr     crc_block
                moveq.l #UNPACKED_CRC,d1
                cmp.w   (sp)+,d2
                beq.s   unpack17
                ELSEIF
                bra.s   unpack17
                ENDC
unpack16
                move.l  d1,BUFSIZE(sp)
unpack17
                lea     BUFSIZE(sp),sp
                movem.l (sp)+,d0-d7/a0-a6
                rts

input_value
                move.w  (a0)+,d0
                and.w   bit_buffer,d0
                sub.w   (a0)+,d0
                bne.s   input_value
                move.b  16*4-4(a0),d1
                sub.b   d1,bit_count
                bge.s   input_value2
                bsr.s   input_bits3
input_value2
                lsr.l   d1,bit_buffer
                move.b  16*4-3(a0),d0
                cmp.b   #2,d0
                blt.s   input_value4
                subq.b  #1,d0
                move.b  d0,d1
                move.b  d0,d2
                move.w  16*4-2(a0),d0
                and.w   bit_buffer,d0
                sub.b   d1,bit_count
                bge.s   input_value3
                bsr.s   input_bits3
input_value3
                lsr.l   d1,bit_buffer
                bset    d2,d0
input_value4
                rts

input_bits
                and.w   bit_buffer,d0
                sub.b   d1,bit_count
                bge.s   input_bits2
                bsr.s   input_bits3
input_bits2
                lsr.l   d1,bit_buffer
                rts

input_bits3
                add.b   d1,bit_count
                lsr.l   bit_count,bit_buffer
                swap    bit_buffer
                addq.w  #4,input
                move.b  -(input),bit_buffer
                rol.w   #8,bit_buffer
                move.b  -(input),bit_buffer
                swap    bit_buffer
                sub.b   bit_count,d1
                moveq.l #16,bit_count
                sub.b   d1,bit_count
                rts

read_long
                moveq.l #3,d1
read_long2
                lsl.l   #8,d0
                move.b  (a0)+,d0
                dbra    d1,read_long2
                rts

make_huftable
                moveq.l #$1f,d0
                moveq.l #5,d1
                bsr.s   input_bits
                subq.w  #1,d0
                bmi.s   make_huftable8
                move.w  d0,d2
                move.w  d0,d3
                lea     -16(sp),sp
                move.l  sp,a1
```

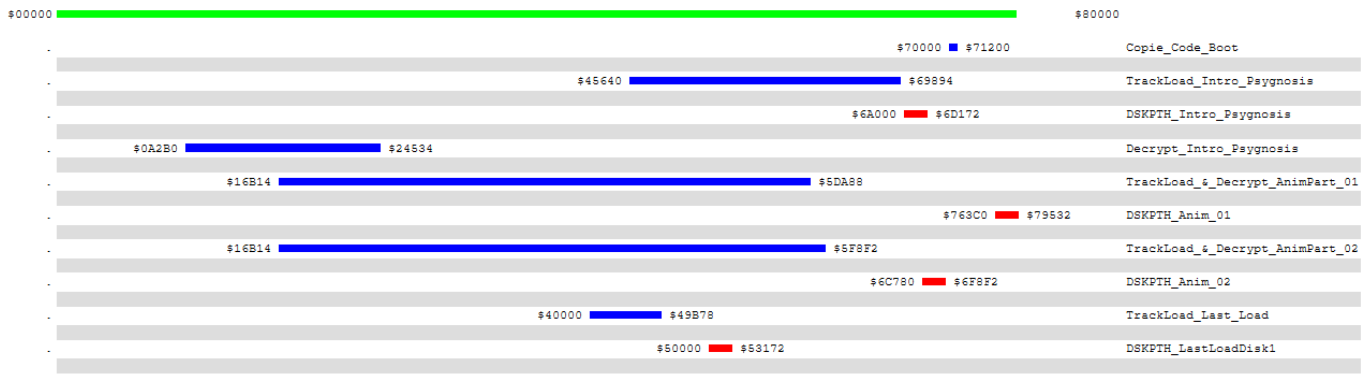
```
make_huftable3
                moveq.l  #$f,d0
                moveq.l  #4,d1
                bsr.s    input_bits
                move.b   d0,(a1)+
                dbra     d2,make_huftable3
                moveq.l  #1,d0
                ror.l    #1,d0
                moveq.l  #1,d1
                moveq.l  #0,d2
                movem.l  d5-d7,-(sp)
make_huftable4
                move.w   d3,d4
                lea      12(sp),a1
make_huftable5
                cmp.b    (a1)+,d1
                bne.s    make_huftable7
                moveq.l  #1,d5
                lsl.w    d1,d5
                subq.w   #1,d5
                move.w   d5,(a0)+
                move.l   d2,d5
                swap     d5
                move.w   d1,d7
                subq.w   #1,d7
make_huftable6
                roxl.w   #1,d5
                roxr.w   #1,d6
                dbra     d7,make_huftable6
                moveq.l  #16,d5
                sub.b    d1,d5
                lsr.w    d5,d6
                move.w   d6,(a0)+
                move.b   d1,16*4-4(a0)
                move.b   d3,d5
                sub.b    d4,d5
                move.b   d5,16*4-3(a0)
                moveq.l  #1,d6
                subq.b   #1,d5
                lsl.w    d5,d6
                subq.w   #1,d6
                move.w   d6,16*4-2(a0)
                add.l    d0,d2
make_huftable7
                dbra     d4,make_huftable5
                lsr.l    #1,d0
                addq.b   #1,d1
                cmp.b    #17,d1
                bne.s    make_huftable4
                movem.l  (sp)+,d5-d7
                lea      16(sp),sp
make_huftable8
                rts

                IFNE     CHECKSUMS
crc_block
                move.l   a2,a0
                moveq.l  #0,d3
crc_block2
                move.l   d3,d1
                moveq.l  #7,d2
crc_block3
                lsr.w    #1,d1
                bcc.s    crc_block4
                eor.w    #CRC_POLY,d1
crc_block4
                dbra     d2,crc_block3
                move.w   d1,(a0)+
                addq.b   #1,d3
                bne.s    crc_block2
                moveq.l  #0,d2
crc_block5
                move.b   (a1)+,d1
                eor.b    d1,d2
                move.w   d2,d1
                and.w    #$ff,d2
                add.w    d2,d2
                move.w   (a2,d2.w),d2
                lsr.w    #8,d1
                eor.b    d1,d2
                subq.l   #1,d0
                bne.s    crc_block5
                rts
                ENDC
```

On tape/charge le tout sous ASM-ONE, on assemble, on sauve le binaire compilé : **RNC_1**

Ou si vous préférez la version déjà compilée, ci-joint sous forme de suite Hexa.

```
48E7FFFE4FEFFE80244F22486100017672000C80524E43016600010261000166 2F400180
47E8000A2A494DF508006100015449F308004267BBCC644C7000102BFFFE41F60800B1CC
633E544F200C0800000067045 24C52482008080000006702524870 00BDC8670812203F01
520060F43F00D1C049ECFFE04CD400FF48E0FF00B9CB62F097CCD7C87E001C2B0001E15E
1C13700272026100 00C8204A610000F241F90006A220610000E841F90006A2A0610000DE
70FF7210610000A638005344602641F90006A2207000615E448043F508FF23C900DFF180
41F90006A2A0614A1AD91AD951C8FFFC204A613E53406B1A1ADB51C8FFFC102B0001E158
1013EFA87201EF695341CC818C8051CCFFB6BBCE6588301F6706321F530066FA2F410180
4FEF01804CDF7FFF4E753018C046905866F81228003C9E016C026130E2AE1028003D0C00
00026D16530012001400302800 3EC0469E016C026112E2AE05C04E75C0469E016C026104
E2AE4E75DE01EEAE4846584B1C23E15E1C23484692077E109E014E757203E188101851C9
FFFA4E75701F720561CA53406B7C340036004FEFFFF0224F700F720461B612C051CAFFF6
7001E2987201740048E70700380343EF000CB219663A7A01E36D534530C52A0248453E01
5347E355E25651CFFFFA7A109A01EA6E30C61141003C1A039A041145003D7C015305EB6E
53463146003ED48051CCFFC0E28852010C01001166AE4CDF00E04FEF00104E75
```

Petit rappel des phases de chargement des différentes parties du Disk1 de Shadow Of The Beast (déjà vue en fin de Part14)



A la question où copier notre *RNC decruncher* en mémoire la réponse au vu de schéma ci-dessus est assez complexe.

Rappelez-vous de l'analyse du trackloader du jeu, il fonctionne avec une ou plusieurs tables mémoire qui contiennent la position de départ sur le disque ainsi que la taille à trackloader, le tout sans oublier, du moins dans la 1er partie du jeu, les chargements du disk1, un DSKPTH variable qui est aussi contenu dans cette table.
En gros, la zone de 'tampon' qui sert pour la lecture MFM du disque bouge selon le trackload en cours.

Si on regarde bien, on peut voir que pour l'intro Psygnosis, elle est fixé à l'adresse mémoire $6A000
Revoir la section '#Trackloader_Base' dans section 'Part 6 Analyse du TrackLoader #1'.

La longueur (DFF024) est fixée à $18B9 et le registre DFF024 (ak DSKLEN), indique le nbr de words à transférer par appel.
Donc $18B9*2= $3172
$6A000+$3172= $6D172

Donc la zone du 'tampon' pour le 1er trackload 'officiel' de l'animation de SOTB2 est située : $6A000 → $6D172

Hors, si on part du principe que nous effectuerons ce trackload initial non plus avec le TrackLoader officiel du jeu mais tout simplement depuis le boot-secteur avec le TrackDisk.device, cela rend cette zone mémoire disponible.

De plus, comme vue sur le schéma ci-dessus des zone mémoires, elle est utilisé uniquement que sur le 1er Trackload.
Cela nous arrange ☺

Nous pouvons prévoir de placer notre décruncher en $6A200
Et comme il a une taille de 536 Octets, cela nous donne la zone mémoire utilisé : $6A000 → $6A218
On se laisse un peu de marge et on place la zone tampon du decruncher en $6A220

A ce stade vous devriez avoir les fichiers suivants :

```
-------------------------------------------------------
Anim_01                                290 676 Octets
Anim_01.RNC                            236 710 Octets
-------------------------------------------------------
Anim_02                                298 462 Octets
Anim_02.RNC                            235 956 Octets
-------------------------------------------------------
Psygnosys                              164 022 Octets              ; Devra être modifié
Psygnosys.RNC                        ~ 133 501 Octets
-------------------------------------------------------
Last_Load                               39 800 Octets              ; Juste pour info car on ne
Last_Load.RNC                        ~  21 532 Octets              ; compressera pas
-------------------------------------------------------
Disk1_Lower_01                         425 984 Octets
Disk1_Lower_02                          71 728 Octets
Disk1_Lower.bin                        497 700 Octets
-------------------------------------------------------
Disk1_Upper_01                         425 984 Octets
Disk1_Upper_02                          71 728 Octets
Disk1_Upper.bin                        497 700 Octets
-------------------------------------------------------
Disk2_Lower_01                         425 984 Octets
Disk2_Lower_02                          78 016 Octets
Disk2_Lower.bin                        504 000 Octets
-------------------------------------------------------
Disk1_Upper_01                         425 984 Octets
Disk1_Upper_02                          71 716 Octets
Disk2_Upper.bin                        497 700 Octets
-------------------------------------------------------
TRACKLOADER_AlphaOne_P1                    434 Octets
RNC_1                                      522 Octets
DISK1_P1                                 1 9480 Octets
DISK1_P2                                20 228 Octets
DISK1_P3                                16 500 Octets
DISK1_P4                               191 184 Octets
DISK1_PRE                              247 392 Octets
-------------------------------------------------------
```

Bon, je ne vais plus parler d'insérer la disquette trucmuche mais plutôt charger le fichier xxx.
A vous de jongler entre vos disquettes et si nécessaire d'en utiliser d'autres (et oui, encore).

Part 18 Modification du fichier principal : Psygnosys

Le trackload des animations et du dernier chargement avant le message 'Insert Disk 1' se fait directement dans le 'fichier' **Psygnosis**

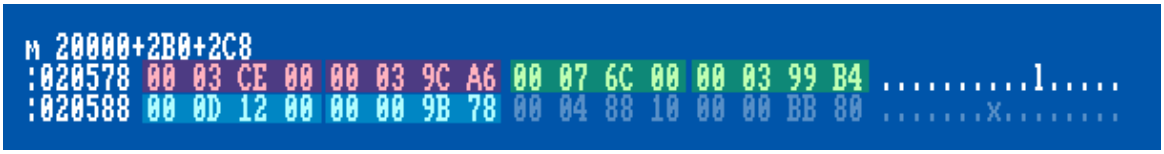
On va donc travailler sur celui-ci, **Booter sur l'AR** et **charger le fichier** préalablement rippé.
Normalement*, ce fichier est chargé en *$2B0*. Afin de faciliter la compréhension globale, on va le charger en *$20000+$2B0*
** Revoir 'Part 8 Analyse du TrackLoader #2' pour comprendre le fonctionnement/déroulement du fichier Psygnosys si nécessaire.*

Tapez :
```
LM Psygnosis_original, 20000+2B0
```

On nettoie le code, on supprime le **BSR** vers la routine *Retour_T00*, **Tapez :**
```
A 20000+2B0+C
~202BC  NOP                     ; Anciennement 'GoTo #Retour_T00'
~202BE  NOP                     ; 2 nop pour le désactiver.
~202C0  <RETURN>
```

On remplace le TrackLoader par celui que l'on a modifié d'AlphaOne.
```
LM TRACKLOADER_AlphaOne_P1, 20000+2B0+30                    // 202E0 →2049E
```

On modifie la table d'origine avec nos propres valeur le tout codé à chaque fois sur 8 Bytes
```
M 20000+2B0+2C8
```



Revoir explication tableau un peu plus haut.

On modifie maintenant les 3 appels et on supprime ce qui n'est plus nécessaire.
```
A 20000+2B0+348
```

```
#Chargement Animation Part #1 ($5F8 adresse original)
~205F8  LEA     763C0,A2        ; Conf de DSKPTH en A2 pour notre TrackLoader
~206FE  LEA     16B14,A0        ; A0=16B14, Adr_dest_mémoire
~20604  LEA     578.S,A1        ; A1=$578,  Nouveau Pointeur pour notre adresse de notre nouveau tableau.
~20608  BSR     202E0           ; Appel de notre nouveau Trackloader d'AlphaOne.
~2060C  JSR     6A000           ; Appel de notre décompacteur qui utilise donc aussi A0.

                                ; Pourquoi en 6A000 ? voir fin de la section 'partie 17' pour plus d'info.

~20612  BSR     206B4           ; Appel du decomp/décrypt original.
~20616  NOP                     ; Voir plus bas
~20618  NOP                     ; Voir plus bas


#Chargement Animation Part #2 ($618 adresse original)
~2061A  LEA     6C780,A2        ; Conf de DSKPTH en A2 pour notre TrackLoader
~20620  LEA     16B14,A0        ; A0=16B14, Adr_dest_mémoire, ne change pas.
~20626  LEA     580.S,A1        ; A1=$580,  Nouveau Pointeur pour notre adresse de notre nouveau tableau.
~2062A  BSR     202E0           ; Appel de notre nouveau Trackloader d'AlphaOne.
~2062E  JSR     6A000           ; Appel de notre décompacteur qui utilise donc aussi A0.
~20634  BSR     206B4           ; Appel du decomp/décrypt original.
~20636  NOP                     ; Voir plus bas
~20638  NOP                     ; Voir plus bas
```

```
Pour info :
~20638   LEA     256.S,A7       ; Code original, On va le déplacer APRES notre dernier TrackLoad.
                                ; Car ce n'est pas une bonne idée de changer A7 avant de faire appel à nos routines.
```

```
# Chargement dernier Data du Disk1     LastLoad-Disk1
~2063A  LEA     50000,A2                ; Conf de DSKPTH en A2 pour notre TrackLoader
~20640  LEA     40000,A0                ; A0=40000, Adr_dest_mémoire, ne change pas
~20646  LEA     588.S,A1                ; A1=$588,  Nouveau Pointeur pour notre l'adresse de notre tableau
~2064A  BSR     202E0           ; Appel de notre nouveau Trackloader d'AlphaOne
~2065E  JSR     6A000           ; Appel de notre décompacteur qui utilise donc aussi A0.
~20654  LEA     256.S,A7                ; On remet le code d'origine que l'on avait supprimé en 20638
~20658  NOP                     ;
~2065A  NOP                     ; Ensemble de NOP Pour retomber sur nos pattes
~2065C  NOP                     ; en $2065E
```

```
~2065E  LEA     202C6(PC),A0    ; Code original, ne change pas.

A 20000+70A                     ; Sans oublier d'ajouter la possibilité de sauter les animations
~2070A  BRA     20638           ; de remplacer l'apparition du fond d'écran blanc l'ors de l'appui de la souris
~2070E  NOP                     ; par un branchement directement sur la routine 'fin des animations'
~20710  NOP
```

Et bien sûr, on sauve le tout, **tapez :** `SM Psygnosis_mod, 20000+2B0 20000+2B0+280B6`
Et on le passe à la moulinette du compresseur RNC Propack pour obtenir le fichier **Psygnosys_mod.RNC** (taille !133502 Octets)

Part 19a Création de notre disk 1 de SOTB2

Rappel :

```
Anim part #01          Row.Pos ➔ 3CE00       taille ➔ 39CA6  //     (!43 Tracks)    Pos. Track !44  + $600
Anim part #02          Row.Pos ➔ 76C00       taille ➔ 399B4  //     (!42 Tracks)    Pos. Track !86  + $800
Anim Psygnosis         Row.Pos ➔ B0600       taille ➔ 2097D  //     (!24 Tracks)    Pos. Track !128 + $600
Last_Load_Disk1        Row.Pos ➔ D1200       taille ➔  9B78  //     (8 Tracks)      Pos. Track !152 + $200
```

Rebooter et entrer dans l'AR puis Tapez :
O 00, 20000 70000

Ensuite on écrit notre petit bout de code :
A 20000

```
2000C   NOP                         ; On se garde une petite marge, au cas où on voudrait ajouter une cracktro
2000E   NOP                         ;

                                    ; #TrackLoad de Psygnosis_mod.RNC
20010   MOVE.W  #2,1C(A1)           ; Trackdisk.device en mode lecture
20016   MOVE.L  #40000,28(A1)       ; Destination mémoire
2001E   MOVE.L  #21000,24(A1)       ; Taille à trackloadé (un petit peu plus)
20026   MOVE.L  #B0600,2C(A1)       ; Position Raw Disk          // Track !128 et  au début du 4eme secteur (donc le secteur 3, on part de 0)
2002E   MOVEA.L 4.S,A6              ; au ca où
20032   JSR     -1C8(A6)            ; Lance le TrackLoad du 'fichier' Psygnosis_MOD.RNC      OPCODE=4E AE FE 38

                                    ; #Re-copie du code de decrunch RNC en mémoire 6A200
20036   LEA     200E6(PC),A0        ; A0=Adr. Source mémoire             $200E6
2003A   LEA     6A200,A1            ; A1=Adr. Destination mémoire        $6A200
20040   MOVE.L  #85,D0              ; D0=Taille à copier (en nbr de Longword à copier)      // 522 Octets soit $82 LongWord
20046   MOVE.L  (A0)+,(A1)+         ; ➔ Copie Source vers destination
20048   DBF     D0,20046            ; ← D0=D0-1 et on boucle tant que D0 est différent de -1

                                    ; #Decompression de l'animation Psygnosis_MOD.RNC
2004C   LEA     40000,A0            ; A0= Adr. Source mémoire du 'fichier' Psygnosis_MOD.RNC
20052   JSR     6A200               ; GoSub #RNC_Decrunc

                                    ; #Re-copie du code d'origine de SOTB2
20058   LEA     2008C(PC),A0        ; A0= Adr. Source mémoire            $2008C
2005C   LEA     7FC08C,A1           ; A1=Adr. Destination mémoire        $7FC08C
20062   MOVE.W  #18,D0              ; D0=Taille à copier (en nbr de Longword à copier)
20066   MOVE.L  (A0)+,(A1)+         ; ➔ Copie Source vers destination
20068   DBF     D0,20066            ; ← D0=D0-1 et on boucle tant que D0 est différent de -1

2006C   MOVE.L  #6CA94,C2           ; On repositionne les deux valeurs importantes d'origine de SOTB2
20076   MOVE.L  #9B78,C6            ; Voir partie 'Part 6 Analyse du TrackLoader #1' dans la section analysée #Pre_Conf_TrackLoader
20080   JMP     7F08C               ; Et on saute sur le code d'origine de SOTB2 recopié en mémoire.
;=========================
                                    ; Code d'origine de SOTB2
20086   NOP                         ; On recopie le code original juste avant le lancement de l'intro Psygnosis
                                    ; Voir partie 'Part 6 Analyse du TrackLoader #1' dans la section analysée #Pre_Conf_TrackLoader
                                    ; Code qui va être exécuté AVANT de lancer le Code trackloadé.
20088   NOP
2008A   NOP                         ; Petite marge, au cas où.

2008C   MOVE.W  #$7FFF,DFF09A       ; Conf INTENA, clear all Level
20094   MOVE.W  #$7FFF,DFF096       ; Conf DMACON, clear all DMA channels and BBUSY, BZERO, BLTPRI
2009C   LEA     200A8(PC),A0        ; A0=80, adresse début conf pile superviseur
200A0   MOVE.L  A0,20.S             ; copie de A0 à l'adresse =$20 pour la conf superviser
200A4   MOVE.W  #2700,SR            ; Pile Superviseur =$2700
200A8   MOVE.W  #2700,SR            ; Pile Superviseur =$2700
200AC   LEA     40000,A0            ; A0=Adr source      = $40000
200B2   LEA     2B0.S,A1            ; A1=Adr de Destination= $2B0
200B6   MOVE.L  (A0)+,(A1)+         ; ➔       Copie
200B8   CMPA.L  #68200,A0           ; Compare $68200 avec A0, c'est la dernière adresse quand tout sera copié ($68200-$40000=$28200 size)
200BE   BNE     200B6               ; ← Tant que l'on n'est pas à la fin des données, on copie
200C0   MOVE.W  #8210,DFF096        ; Conf DMACON, Set Disk DMA, Enable all DMA
200C8   MOVE.W  #7FFF,DFF09A        ; Conf INTENA, clear all Level
200D0   MOVE.W  #7FFF,DFF09C        ; Conf INTREQ
200D8   CLR.W   DFF180              ; On efface le fond d'écran.
200DE   LEA     256.S,A7            ; A7=256, adr. de la pile (identique à l'original)
200E2   JMP     2B0.S               ; On lance l'animation

;=========================
                                    ; Code du décompacteur RNC      $200E6➔$202F0    Taille : 522 Octets
200E6   MOVEM.L D0-D7/A0-A6,-(A7)
200EA   LEA     -200(A7),A7
200EE   ...
```

Insérez le disk contenant les fichiers nécessaires dans le lecteur externe (donc **DF1**) et insérez une disquette formatée dans **DF0** : **SOTB2_D1_CRACK**
Swaper quand c'est nécessaire les disquettes dans le lecteur externe afin de charger le/les bon(s) fichier(s).

Comme prévu on charge notre **décompacteur RNC** en $200E6
```
LM 1:RNC_1, 200E6
```

On re-calcul le checksum
```
BOOTCHK 20000
```

Et on re-écrit le tout sur notre disquette
```
WT 0 1 20000
```

On s'attaque maintenant aux données préalablement préparées. *(Part 15b Préparation des fichiers pour la création de notre Disk1 cracké')*

Tapez :
```
O 00, 10000 80000
RT 0 1 10000
LM 1:DISK1_PRE, 10000+400
BOOTCHK 10000
WT 0 !45 10000                                          // Longueur 44 + 1 track de boot
```

```
o 00, 10000 80000
Ready.
rt 0 1 10000
Disk ok
lm 1:DISK1_PRE, 10000+400
Loading from 010400 to 04CA60
Disk ok
wt 0 !45 10000
```

On Passe maintenant aux animations.
```
O 00, 20000 60000
RT !44 1 20000
LM 1:Anim_01.RNC, 20000+600          ; Anim01 RNC Size= $39CA6+$600=$3A2A6      $3A2A6/$1600= !42,31 donc !43
WT !44 !43 20000

O 00, 20000 60000
RT !86 1 20000
LM 1:Anim_02.RNC, 20000+800          ; Anim02 RNC Size= $399B4+$800=$3A1B4      $3A1B4/$1600= !42,26 donc !43
WT !86 !43 20000
```

Sans oublier le code de l'animation Psygnosis modifié et compacté.
```
O 00, 20000 60000
RT !128 1 20000
LM 1:Psygnosis_mod.RNC, 20000+600    ; PsygnosisMod RNC Size=$2097E+$600=$20F7E  $20F7E/$1600= !23,98 donc !24
WT !128 !24 20000
```

/!\ Il nous restera plus que LastLoad_Disk1 que l'on fera plus tard !

Part 19b Création de notre disk 2 de SOTB2

Revoir : '*Part 15 Réorganisation des données pour création de nos disks*' si nécessaire.

***** : **Disk_lower_01, Disk_lower_02, Disk_upper_01, Disk_upper_02**

Insérez le disk contenant les fichiers nécessaires* dans le lecteur externe (donc **DF1**)
et **insérez** une disquette **vierge** dans **DF0** : **SOTB2_D2_CRACK**

Comme prévu on charge nos données ripées et on écrit le tout sur notre nouveau disk.

```
O 00, 10000 80000
LM 1:Disk2_Lower_01, 10000                    // Taille !425984   !425984-$4C18=!406504
WT 0 !73 10000+4C18                           // !406504/!5632 = 72.2 soit 73 Tracks
                                              // Sans oublier de 'sauter' les 1er $4C18 octets
```

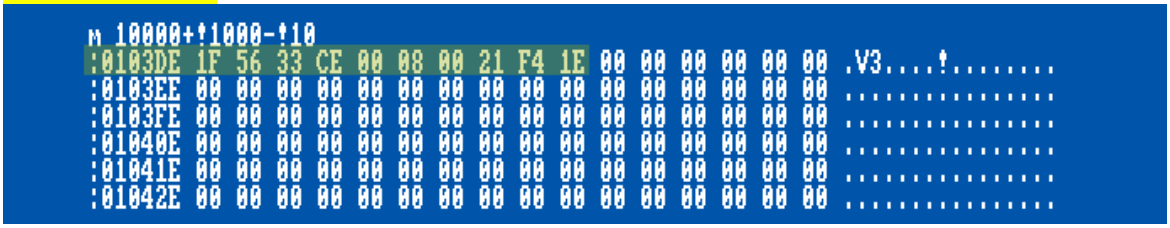
Alors maintenant c'est un peu plus compliqué dans le sens où **l'on doit** continuer **EXACTEMENT** à la suite de ces données et non pas de la '**track**'.
Hors, avec *l'action replay* on fonctionne uniquement en mode 'track'.

Ce que l'on va faire c'est charger la dernière track écrite -1 avec la commande ci-dessus et charger nos données à la suite.

```
O 00, 10000 80000
RT !72 1 10000
```

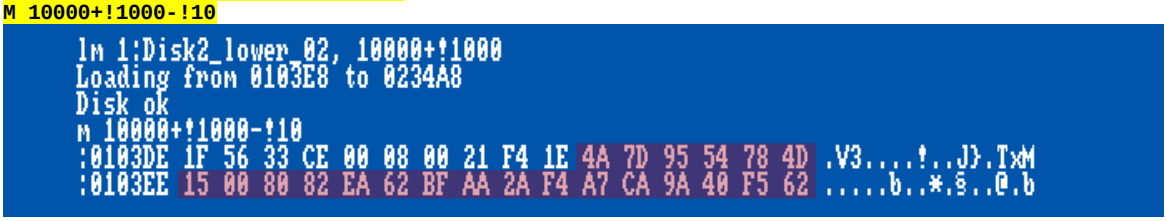
Petit calcul de l'offset : !5632*!72=**405504** et !406504 - !**405504**=!**1000**
Une petite vérification s'impose.

```
M 10000+!1000-!10
```



On charge les prochaines données à la suite.
Insérez maintenant le disk contant le fichier Disk_lower_02 dans le lecteur externe.
Tapez :

```
LM 1:Disk2_Lower_02, 10000+!1000              // Taille !78016
M 10000+!1000-!10
```



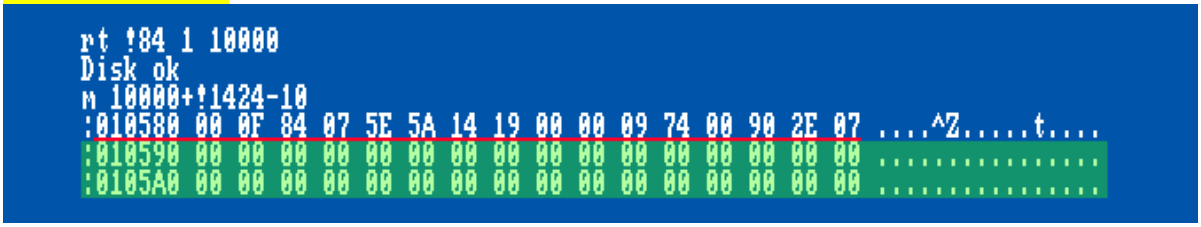
Donnée donc chargée jusqu'à $234A8 et présente en mémoire de $10000 → $234A8
Ce qui nous donne $234A8-$10000=$134A8 $134A8/$1600= !14.03 **donc !15**

```
WT !72 !15 10000
```

Maintenant nous avons besoin de copier nos données ripées du Disk original 2 UPPER à la suite.
Mais, il existe une grande zone vide d'origine à cet endroit, il faut donc bien sûr ne pas la laisser et copier nos données exactement à la suite de nos dernières données '**In_Game_62**' (revoir tableau si nécessaire en : *Part 15 Réorganisation des données pour création de nos disks*

Position de nos futures données '**In_Game_62**' en position raw-disk **$73D90-7ED9B**, ce qui nous donne :
$73D90 = !474512 474512/5632= !84,25 !84*5632= !473088 !474512-!473088= !1424
Donc début des prochaines données en Track !84, offset !1424

```
O 00, 10000 80000
RT !84 1 10000
M 10000+!1424-!10
```



On est bon, il nous reste plus qu'à charger à la suite de ces données.
Insérez maintenant le disk contant le fichier Disk_Upper_01 dans le lecteur externe.

```
LM 1:Disk2_Upper_01, 10000+!1424      // Taille fichier !425984 +donnée précédente !1424 = !427408
!427408 / !5632 = !75,9       soit 76 Tracks à enregistrer
```

MAIS, si on regarde bien ce que l'on a prévu de faire : *Part 15 Réorganisation des données pour création de nos disks*
On s'arrête sur les données '**In_Game_69 [END]'**, positionnée dans notre fichier de rip **Upper D2** en position **raw** : **$61E6C-$6638B**

On vérifie le début le bon positionnement des données données rippées en mémoire, le début des données **In_Game_69 [END]'**

M 10000+ !14242+61E6C // Début de la zone mémoire de Travail + Précèdent offset + Pos. Start Attendue dans fichier rip

```
m 10000+!1424+61E6C
:0723FC 00 00 45 21 C3 14 0C 56 F1 48 B5 7C E0 B2 34 96 ..E!...V.H.|..4.
```

On retombe bien sur nos 'signatures' hexa précédemment relevées, c'est bon.

Il nous reste plus qu'a 'nettoyer' notre zone mémoire à la fin de ces données, à savoir en : $1000 + !1424 +$6638B+1
(Zone mémoire de travaill + offset précédent + Fin des données de **In_Game_69 [END]** + 1)
O 00, 10000+!1424+6638B+1 80000 **// $7691C → $80000**

On calcule combien de track on doit écrire sur notre disque
($7691C-$10000)/$1600 = $4A exactement !74,6 donc !75 soit $4B, aux choix.

WT !84 !75 10000
comme prévu, on arrive presque à la fin du disque (Piste 79 en l'occurrence).
Il nous reste plus qu'à écrire les dernières données à la suite, à savoir celle de '**Load_Menu_08**'
Données qui se trouvent sur le Disk original 2 en face UPPER en fin du disk, donc dans notre fichier de rip : **Disk2_Upper_02**

Insérez maintenant le disk contant le fichier Disk_Upper_02 dans le lecteur externe.
Bon, on va faire plus simple pour ces dernières données.
Nous connaissons sa 'signature' Hexa et nous connaissons sa taille, On va donc charger notre fichier de Dump en mémoire
Chercher notre chaine hexa, ajouter sa 'taille' à sa position trouvée et sauver le tout sur notre disque externe.

Signature Hexa : **4AB8041E67162078041E20B804262078042220B8042A700021C0041E41F80B1A**
taille : $1EBC

Tapez :
O 00, 10000 80000
LM 1:Disk2_Upper_02, 10000
F 4A B8 04 1E 67 16 20 78 04 1E 20 B8 04 26 20 78

```
dir 1:
Directory of (empty)upper_02
      425984  Disk2_upper_01disk used
      071716  Disk2_upper_02
0722 blocks free, 58.1 % of disk used
Disk ok

lm 1:Disk2_upper_02, 10000
Loading from 010000 to 021824
Disk ok

f 4A B8 04 1E 67 16 20 78 04 1E 20 B8 04 26 20 78
Search from: 000000 to: 080000
013290
Ready.
sm 1:Load_Menu_08, 13290 13290+1EBC
Disk ok
```

Et on sauve le tout : **SM 1:Load_Menu_08, 13290 13290+1EBC**

Il faut maintenant sauver ces données à la suite de nos datas sur notre disk2.
Toujours basé sur notre tableau, il devrait être présent sur notre disk en position raw : $DA11C-$DBFD7
$DA11C = !893212 !893212 / !5632 = 158,6 !158 * !5632= !889856
!893212 - !889856 = !3356 **Donc Ofsset $D1C**

Tapez :
O 00, 10000 80000
RT !158 1 10000
M 10000+D1C-10

```
o 00, 10000 80000
Ready.
rt !158 1 10000
Disk ok
m 10000+D1C-10
:010D0C 01 83 F8 87 19 A5 D9 FD 00 00 53 3C 00 CA 3C C0 .........S<..<.
:010D1C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:010D2C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Ok, on est toujours bon.

On charge notre fichier à la suite et on sauve le tout, c'est partit.
LM 1:Load_Menu_08, 10000+D1C

$D1C + $1EBC = $2BD8 = !11224 !11224 / 5362 = 1,9 **Soit 2 Tracks**
WT !158 2 10000

```
dir 1:
Directory of (empty)
      425984  Disk2_upper_01
      071716  Disk2_upper_02
      007868  Load_Menu_08
0704 blocks free, 60.0 % of disk used
Disk ok

lm 1:Load_Menu_08, 10000+D1C
Loading from 0I0D1C to 012BD8
Disk ok

wt !158 2 10000
Disk ok
```

Pfiouuuu, C'est fini pour cette section 😊

Part 20a Analyse Occupation mémoire libre pour insérer notre TackLoader Final

Il faut revenir sur l'analyse du dernier chargement du Disk1.
Si on fait une synthèse des positions en mémoire des différentes sections de cette partie du code, cela nous donne le tableau suivant :
Le but étant de comprendre la place en mémoire qu'occupe chaque partie et de trouver un emplacement 'libre' où l'on pourra insérer notre *TrackLoader* modifié.

		Appelé par				
Adr Mémoire	Sous Routine	INSERT DISK 2	IN GAME	Info	Dans la zone Potentielle ?	Zone Potentielle
256-268	#WaitSyncV	4F016	4F016	OBLIGATOIRE	N/A	Non
26A-284	#Working_on_BE2_Mark	262	plein de fois	OBLIGATOIRE	N/A	Non
286-294	#Working_on_BE2_Mark_#x	276	276	OBLIGATOIRE	N/A	Non
446-BD6						Non
						Non
1824-1956	#Decomp/Decrypt	plein de fois	plein de fois	OBLIGATOIRE	N/A	Non
1958-19C2	#Decomp/Decrypt_02	plein de fois	plein de fois	OBLIGATOIRE	N/A	Non
19C4-1A16	#Trackloader_2_Init	1B40 1E48 4ECC 8CE6 41BF2	1B40 1E48 **4ECC** **8CE6** 41BF2	Pas nécessaire	A PATCHER	
1A18-1A3E	#Move Inter.	1A84 1A8E 1B80	1A84 1A8E 1B80	Pas nécessaire	Oui	
1A6C-1A82	#Position Reach?	1B6E	1B6E	Pas nécessaire	Oui	
1A84-1A86	#Avance_Recule	1A70	1A70	Pas nécessaire	Oui	
1A88-1A94	#GoTo_To_Track	1A72	1A72	Pas nécessaire	Oui	
1A96-1AA8	#Retour_T00	1AA8 1E4C 1E80 4ED0 8CEA **41BF6**	1AA8 1E4C 1E80 **4ED0** **8CEA**	Pas nécessaire	A PATCHER	
1ABC-1AC8	#WAIT	1A30 1A5A 1AA4	1A30 1A5A 1AA4		Oui	
1AAA-1ABA	#Disque Ready?	1AA0	1AA0	Pas nécessaire	Oui	
1A42-1A68	#Move Exter.	1A7C 1A86 1AA2	1A7C 1A86 1AA2	Pas nécessaire	Oui	
1ACA-1AF0	#DF0__SIDE_DOWN__MOTOR_OFF__DIR_EXT	plein de fois	plein de fois	Pas nécessaire	A PATCHER	
1AF2-1B0E	#DF0__SIDE_DOWN__MOTOR_ON__DIR_EXT	1B4E	1B4E	Pas nécessaire	Oui	
1B10-1B2C	#DF0__SIDE_UP__MOTOR_ON__DIR_EXT	1B52	1B52	Pas nécessaire	Oui	
1B2E-1B36	#Trackloader_Start	plein de fois	plein de fois	Pas nécessaire	A PATCHER	
1B38-1B40	#Lecture_Table_and_Start_Trackload_Or_Not	1B30	1B30	Pas nécessaire	Oui	
1B44-1B50	#Side_Select	1B3E	1B3E	Pas nécessaire	Oui	
1B52-1B84	#Recup_Info_pour_Trackloader_x	1B4C	1B4C	Pas nécessaire	Oui	
1B86-1BC0	#Trackloader_Base	1B7E	1B7E	Pas nécessaire	Oui	
1BC8-1C1E	#Test CIA Ready	N/A	N/A	Pas nécessaire	N/A	
1C20-1C32	#Traitement_MFM_BASE	1C10	1C10	Pas nécessaire	Oui	
1C34-1C3C	#Check_Traitement_MFM	1C28	1C28	Pas nécessaire	Oui	
1C3E-1C5A	#Traitement MFM bit pair	1C36	1C36	Pas nécessaire	Oui	
1C5C-1CDC	#Phase_De_Chargement_#1	1E62	1E62	OBLIGATOIRE	N/A	Non
1CE0-1CE4	#Donnée_déja_chargé?	1EA8 256C 7CE6	1EA8 256C 7CE6 **4D86C**	OBLIGATOIRE	N/A	Non
1CE6-1D46	#Phase_De_Chargement_#2_2/2	N/A	N/A	OBLIGATOIRE	N/A	Non
1D48-1DDE	#LoadPhase_#1_End_Part1/2	1E7C	1E7C	OBLIGATOIRE	N/A	Non
1DE0-1E62	#Base_LastLoad	256	256	OBLIGATOIRE	N/A	Non
1E66-201E	#Phase_De_Chargement_#2_1/2	N/A	N/A	OBLIGATOIRE	N/A	Non
1E74-201E	#Phase_De_Chargement_#2_1/2_bis	N/A	N/A	OBLIGATOIRE	N/A	Non

22FE-2332	#BLITTER_CONF_#01	1F8A 1FC6 2030 2080 210E 211C 212A 2138 2226 2266 22F2	1F8A 1FC6 2030 2080 210E 211C 212A 2138 2226 2266 22F2	**OBLIGATOIRE**	N/A	**Non**
2334-237C	#BLITTER_CONF_#02	1F8E 1FCA 2034 2084 2112 2120 212E 213C 222A 226A 22F6 **4C408**	1F8E 1FCA 2034 2084 2112 2120 212E 213C 222A 226A 22F6	**OBLIGATOIRE**	N/A	**Non**
237E-23B2	#BLITTER_CONF_#00	1FBC 2020 20FA 21B2 21BC 21C6 21D0 21E4 21EE 2216 2256 22A8 22C2	1FBC 2020 20FA 21B2 21BC 21C6 21D0 21E4 21EE 2216 2256 22A8 22C2	**OBLIGATOIRE**	N/A	**Non**
23B4-23DE	#START_LEVEL1_OR_NOT	1F92 2054	1F92 2054	**OBLIGATOIRE**	N/A	**Non**
24CC-254E	#CHARGEMENT_LEVEL1	**plein de fois**	N/A	**OBLIGATOIRE**	N/A	**Non**
						Non
3FCC-3FFC	#Update_Table_10E4_10F0__1094_10A0	7B76	7B76	**OBLIGATOIRE**	N/A	**Non**
41FA-422A	#Update_Table_CDC_CEC__D2C_D38	7B72	7B72	**OBLIGATOIRE**	N/A	**Non**
						Non
5710-5752	#Decrypt_RAW	1EBE	1EBE	**OBLIGATOIRE**	N/A	**Non**
5AC4-5ADC	#Update_Table_A0_With_3A0&3E0__E58_1210_E5C_1214	5B0E 5B38	5B0E 5B38	**OBLIGATOIRE**	N/A	**Non**
5B10-5B38	#JSR(A0)_or_Erase	5AB6 7B7A	5AB6 7B7A	**OBLIGATOIRE**	N/A	**Non**
						Non

Il semblerait que la zone **$19C4 → $1C5A** soit un bon candidat.

7B2E-7B8E	#RAZ_Tables	1DA2 4E3A 7EF4	1DA2 4E3A 7EF4	**OBLIGATOIRE**	N/A	**Non**
8C30-8C8A	#BASE_INSERT_DISK_ASKED...	1E50 4EDC 7C48 41BFA	1E50 4EDC 7C48 41BFA	**OBLIGATOIRE**	N/A	**Non**
8C8C-8D0A	#CHECK_DISK...	8C82	8C82	**OBLIGATOIRE**	N/A	**Non**
8C92-8CD4	#CHECK_DISK_A0	8C8A	8C8A	**OBLIGATOIRE**	N/A	**Non**
8CDC-8D00	#DISK2_OR_DISK1_INSERTED?	8D08 8D12	8D08 8D12	**OBLIGATOIRE**	N/A	**Non**
8D02-8D0A	#Check_Signature_DISK_02	N/A	N/A	**OBLIGATOIRE**	N/A	**Non**
8D0C-8D14	#Check_Signature_DISK_01	8D00	8D00	**OBLIGATOIRE**	N/A	**Non**
9B24-9B3C	#RAZ_TRACKLOADER	254E 7266 7E48	254E 7266 7E48	**OBLIGATOIRE**	**A PATCHER**	**Non**
9B40-9B88	#Pre_Base_TrackLoadX	9B28	9B28	Pas nécessaire	**A PATCHER**	
9BA6-9BB8	#Floppy_Ready?	9BA0 9BDC 9C1C	**9BA0** 9BDC **9C1C**	Pas nécessaire	**A PATCHER**	
9B8A-9BA4	#Motor_ON	9B2C	9B2C	Pas nécessaire	**A PATCHER**	
9BBA-9BE0	#Retour_T00	9B30 9BDA 9BEC	**9B30** 9BDA **9BEC**	Pas nécessaire	**A PATCHER**	

Il semblerait aussi que la zone **$9B40 → $9BA4** soit aussi disponible aussi mais de plus petite taille.

19C4 → $1C5A = $296 = !662 Octets Dispo
9B40 → $9BE0 =$A0 = !160 Octets Dispo

Part 20b Modification et compilation du Trackloader d'AlphaONE v2 – Phase2

Cette fois-ci nous allons prendre le **trackload D'AlphaOne v2 de 2005**, il permet un adressage sur les bytes pairs et impairs.
Comme nous avons réorganisé le positionnement des datas sur nos disquettes de crack, il va être nécessaire soit de modifier tous les appels.
OU **de garder les tableaux d'origines des appels et de patcher directement les nouvelles positions réel.**

DISK2 – Cracked Version

		ORIGINAL		Position	dans	
Ordre D'appel	Adr. D'appel	Disk	SIDE	Disk Réel	Notre Disk Cracké	Delta entre la Pos.Orig Et la Pos. Réel
Load_Menu_11	1EA0	D2	LOW	04C18-07333	00000-0271B	-4C18
Load_Menu_12 In_Game_50	1CF0	D2	LOW	07334-07B2F	0271C-02F17	-4C18
Load_Menu_13 In_Game_51	1D02	D2	LOW	07B30-16507	02F18-118EF	-4C18
Load_Menu_14 In_Game_52	1D14	D2	LOW	16508-1790F	118F0-12CF7	-4C18
Load_Menu_15 In_Game_53	1D26	D2	LOW	17910-1871F	12CF8-13B07	-4C18
Load_Menu_16 In_Game_54	1D38	D2	LOW	18720-19267	13B08-1464F	-4C18
Load_Level1_01	24DE	D2	LOW	19268-25D0F	14650-210F7	-4C18
In_Game_61	762C					
In_Game_55-RET	7800					
In_Game_31-A	7966					
In_Game_44	7C80					
Load_Level1_02	24F0	D2	LOW	25D10-26B93	210F8-21F7B	-4C18
In_Game_63	763E					
In_Game_45	7C9A					
In_Game_56-RET	7812					
In_Game_12	8A9C					
Load_Level1_03	2508	D2	LOW	26B94-3495B	21F7C-2FD43	-4C18
In_Game_46	7CAC					
In_Game_57-RET	7824					
In_Game_31-B	7980					
Load_Level1_04	251A	D2	LOW	3495C-38583	2FD44-3396B	-4C18
In_Game_58-RET	7836					
In_Game_31-C	7992					
In_Game_48	7CD0					
Load_Level1_06	2542	D2	LOW	38584-38923	3396C-33D0B	-4C18
In_Game_47	7CBE					
In_Game_30	8244					
In_Game_22	84C2					
In_Game_14	8AC0					
In_Game_65	7668					
Load_Level1_05	2530	D2	LOW	38924-39A9B	33D0C-34E83	-4C18
In_Game_49	7CDE					
In_Game_31-D	79A0					
In_Game_18	7A3C					
In_Game_10	8A70	D2	LOW	39A9C-3E9A3	34E84-39D8B	-4C18
In_Game_01	719A					

In_Game_02	71AC	D2	LOW	3E9A4-43E43	39D8C-3F22B	-4C18
In_Game_11	8A82					
In_Game_29	8232					
In_Game_21	84B0					
In_Game_23	7AD0	D2	LOW	43E44-4F0AF	3F22C-4A497	-4C18
In_Game_13	8AAE					
In_Game_03	71C6					
In_Game_04	71D8	D2	LOW	4F0B0-52AE3	4A498-4DECB	-4C18
In_Game_24	7AE2					
In_Game_15	7A0A	D2	LOW	52AE4-5A8AB	4DECC-55C93	-4C18
In_Game_16	7A1C	D2	LOW	5A8AC-5EA73	55C94-59E5B	-4C18
In_Game_17	7A2E	D2	LOW	5EA74-6305F	59E5C-5E447	-4C18
In_Game_19	8360	D2	LOW	63060-6659B	5E448-61983	-4C18
In_Game_28	8150	D2	LOW	6659C-66B23	61984-61F0B	-4C18
In_Game_20	8372	D2	LOW	6659C-66B23		
In_Game_25	811A	D2	LOW	66B24-68807	61F0C-63BEF	-4C18
In_Game_26	812C	D2	LOW	68808-689E7	63BF0-63DCF	-4C18
In_Game_27	813E	D2	LOW	689E8-69543	63DD0-6492B	-4C18
In_Game_05 In_Game_09-RET In_Game_59 In_Game_60	7E68	D2	LOW	69544-6C42F	6492C-67817	-4C18
In_Game_06	8918	D2	LOW	6C430-6F073	67818-6A45B	-4C18
In_Game_07	892A	D2	LOW	6F074-6FBE3	6A45C-6AFCB	-4C18
In_Game_08	893C	D2	LOW	6FBE4-6FCFF	6AFCC-6B0E7	-4C18
In_Game_54b	7E68	D2	LOW	6FD00-73F87	6B0E8-6F36F	-4C18
In_Game_60	74FA	D2	LOW	73F88-781AF	6F370-73597	-4C18
In_Game_61	750C	D2	LOW	781B0-7837F	73598-73767	-4C18
In_Game_62	751E	D2	LOW	78380-789A7	73768-73D8F	-4C18
In_Game_67	768C	D2	UP	85D04-90D0F	73D90-7ED9B	-11F74
In_Game_57	709E					
In_Game_58	70B0	D2	UP	90D10-93DE3	7ED9C-81E6F	-11F74
In_Game_56	708C	D2	UP	93DE4-9425B	81E70-822E7	-11F74
In_Game_66	767A					
In_Game_55	7074	D2	UP	9425C-99B6F	822E8-87BFB	-11F74
In_Game_64	7650					
N/A	78E4			9425C-99B6F		
END_01	4F7E	D2	UP	99B70-99E77	87BFC-87F03	-11F74
END_02	4F90	D2	UP	99E78-CC8FB	87F04-BA987	
END_03	5010	D2	UP	CC8FC-DC073	BA988-CA0FF	-11F74
END_04	501E	D2	UP	DC074-E2E77	CA100-D0F03	
In_Game_68	78B0	D2	UP	E2E78-E7B0F	D0F04-D5B9B	-11F74
Load_Menu_09	1D5E	D2	UP	E7B10-E7B6F	D5B9C-D5BFB	
In_Game_69 [END]	85EE	D2	UP	E7B70-EC08F	D5BFC-DA11B	-11F74
				EC090-F0F93		
Load_Menu_08	1E70	D2	UP	F0F94-F2E4F	DA11C-DBFD7	-16E78

DISK1 – Cracked Version

		ORIGINAL		Position	dans	
Ordre D'appel	Adr. D'appel	Disk	SIDE	Disk Réel	Notre Disk Cracké	Delta entre la Pos.original et la Pos. Réel
Boosecteur + code						
Load_Menu_10	1E8E	D2	LOW	00000-04C18	0400-05017	+400
Load_Menu_01	1C66	D2	UP	EC090-EFB1F	05018-08AA7	-E7078
Load_Menu_02	1C78	D2	UP	EFB20-EFD6F	08AA8-08CF7	-E7078
Load_Menu_03	1C8A	D2	UP	EFD70-F01A7	08CF8-0912F	-E7078
Load_Menu_04	1C9C	D2	UP	F01A8-F02CB	09130-09253	-E7078
Load_Menu_05	1CAE	D2	UP	F02CC-F092F	09254-098B7	-E7078
Load_Menu_06	1CC6	D2	UP	F0930-F0B97	098B8-09B1F	-E7078
Load_Menu_07	1CD8	D2	UP	F0B98-F0F93	09B20-09F1B	-E7078
In_Game_37	72F0	D1	LOW	76FC0-7B033	09F1C-0DF8F	-6D0A4
In_Game_43	7C44	D1	UP	CFF58-D2763	0DF90-1079B	-C1FC8
In_Game_32	7E68					
In_Game_33	72A8	D1	UP	D2764-DEB2B	1079C-1CB63	-C1FC8
In_Game_34	72BA	D1	UP	DEB2C-DF04F	1CB64-1D087	-C1FC8
In_Game_35	72CC	D1	UP	DF050-E056B	1D088-1E5A3	-C1FC8
In_Game_36	72DE	D1	UP	E056C-EACA7	1E5A4-28CDF	-C1FC8
In_Game_38	7308	D1	UP	EACA8-EB497	28CE0-294CF	-C1FC8
In_Game_39	731A	D1	UP	EB498-FB8A7	294D0-398DF	-C1FC8
In_Game_40	732C	D1	UP	FB8A8-FD0CF	398E0-3B107	-C1FC8
In_Game_41	733E	D1	UP	FD0D0-FDEDF	3B108-3BF17	-C1FC8
In_Game_42	7350	D1	UP	FDEE0-FEA27	3BF18-3CA5F	-C1FC8
					3CC60	

Info		Disk Crack	SIDE Orig.	Disk Réel	Notre Disk Cracké	Delta entre la Pos.original et la Pos. Réel
Anim part #01		D1	LOW	0189C-4880F	Non concerné - TrackDiskDevice	
Anim part #02		D1	UP	85D04-CEAE1	Non concerné - TrackDiskDevice	
Anim Psygnosis		D1	LOW	48810-6CA93	Non concerné - TrackDiskDevice	
Last_Load_Disk1		D1	LOW	6CA94-7660B	Non concerné - TrackDiskDevice	

Ce qui nous donne au final 8 possibilités : (7 + la signature disk qui ne change pas **D2**)

```
Si      (A1)           = 189C  alors   +0                    // Signature Disk, reste en, $189C
Si       A1            = 180E  alors   D2=D2-16E78
Si       A1            = 161E  alors   D2=D2-6D0A4
Si       A1            = 1676  alors   D2=D2+400
Si       A1 entre 1626 et 166E alors   D2=D2-C1FC8
Si       A1 entre 167E et 1776 alors   D2=D2-4C18
Si       A1 entre 177E et 17CE alors   D2=D2-11F74
Si       A1 entre 17D6 et 1806 alors   D2=D2-E7078
```

L'option choisie ici est donc de garder les **raw.pos** originales et de les patcher selon les conditions ci-dessus.
En plus, comme vu au *chapitre 14B*, cela perturbera beaucoup moins le fonctionnement du jeu qui se sert du marqueur de Piste.
Car, comme nous avons réorganiser les données sur nos disquettes de Hack, faire un compte de piste réel ne servirait à RIEN.
Ou alors il faudrait patcher chaque check effectué en **C20** dans tout le code du jeu.

L'astuce est simple, si on regarde bien, aucun check n'est fait sur **C20** pendant le trackload, il est juste mis à jour selon le déplacement de la tête +1 ou -1 selon le déplacement voir une remise à zéro si on est sur la T00

Il suffit alors de calculer sûr qu'elle piste le trackload original aurait dû finir.
C'est simple vue que l'on garde les **raw.pos** d'origine et que l'on connait la **SIZE** des données à trackload.
Il suffit d'additionner les deux pour connaitre la **raw.pos final** et de diviser le tout par la taille de track custom : 189C
Ensuite de copier cette valeur à l'adresse **$C20**

à noter que le jeu original, le trackload original fonctionne par **SIDE** avec une valeur de limite de $84468
Donc, tout ce qui est plus petit que $84468 n'aura besoin d'aucune correction dans notre futur routine de calcul de 'Piste Originale'
Et tout ce qui est plus grand que $84468 aura besoin d'un ajustement, à savoir d'effectuer une soustraction de 84468 de la **raw.pos final**
avant de diviser par 189C.

A noter aussi que si on peut intégrer dans notre code de trackloader custom les routines d'origines que l'on a écrasé et qui sont utile, ça serait bien.
A savoir :

1A96 Car appelé bcp de fois -> **#Retour_T00** → **Raz $C20 et RTS**
1ACA Car appelé bcp de fois -> **#DF0__SIDE_DOWN__MOTOR_OFF__DIR_EXT** → **Simple RTS**
1B2E Car appelé sur chaque Trackload -> **#TrackLoader_2** → **à rediriger vers le début de notre routine de trackload**

Nous allons utiliser la seconde partie de libre en mémoire en **9B40** → **$9BE0** pour nos raw.pos patchs. (car notre code est petit et cette zone aussi)
Ce qui nous donne le code suivant pour la modification de Raw.Pos comme vue plus haut.

DELTA.s **// 150 Octets**

```
                                                ; /!\ à  charger en $9B40
DELTA:                                          ; à  ce stade
                                                ; (A1) = Adr du tableau     et D2 = Raw.Pos du tableau
;---------------------------------------
CASE1:   CMP.W    #$1676,A1                     ; Compare A1 avec 1676 - LoadMenu_10
                  BNE     CASE2                 ; Si différent, on continue en CASE2
                  ADD.L   #$400,D2              ; Sinon, on ajoute $400 à  D2
;---------------------------------------
CASE2:   CMP.W    #$180E,A1                     ; Compare A1 avec 180E - LoadMenu_08
                  BNE     CASE3                 ; Si différent, on continue en CASE3
                  SUB.L   #$16E78,D2            ; Sinon, on soustrait $16E78 de D2
;---------------------------------------
CASE3:            CMP.W   #$161E,A1             ; Compare A1 avec 161E - Crystal Cavern 06/11
                  BNE     NEXT                  ; Si différent , on continue en CASE4
                  SUB.L   #$6D0A4,D2            ; Sinon, on soustrait $16E78 de D2
;---------------------------------------
NEXT:
                  CMP.L   (A1),D2               ; Check si D2 est différent (A1)
                  BNE     DONE                  ; Si c'est le cas, c'est qu'aucun des cas au-dessus a eu lieu
                                                ; et dans ce cas, on saute directement Ã  la fin du code.
;-------------------------------------------------------------------------------------------------------------
SIGNATUREDISK:
                  CMP.L   #$189C,(A1)       ; Compare Raw.Pos avec 189C qui correspond à  Raw.Pos de la signature Disk
                  BEQ     DONE              ; Si identique, on ne change pas D2 et on saute à  la fin de notre routine
;-------------------------------------------------------------------------------------------------------------

CASE4:            CMP.W   #$17D6,A1             ; Compare A1 avec 17D6
                  BLT     CASE5                 ; Si plus petit alors, on continue En CASE5
                  SUB.L   #$E7078,D2            ; Sinon D2=D2-E7078
                  BRA     DONE

CASE5:            CMP.W   #$177E,A1             ; Compare A1 avec 177E
                  BLT     CASE6                 ; Si plus petit alors, on continue En CASE6
                  SUB.L   #$11F74,D2            ; Sinon D2=D2-11F74
                  BRA     DONE

CASE6:            CMP.W   #$167E,A1             ; Compare A1 avec 167E
                  BLT     CASE7                 ; Si plus petit alors, on continue En CASE7
                  SUB.L   #$4C18,D2             ; Sinon D2=D2-4C18
                  BRA     DONE

                  NOP                           ; Pour retomber sur nos pattes en 9BBA
                  NOP
                  NOP
                  NOP
                  NOP
                  ;-------------------------------------------------------------------------------
                  CLR.W   $C20                  ; RAZ du compteur original de piste
                  RTS                           ; Annulation de la routine en 9BBA
                  ;-------------------------------------------------------------------------------

CASE7:            CMP.W   #$1626,A1             ; Compare A1 avec 1626
                  BLT     DONE                  ; Si plus petit alors, on continue En DONE
                  SUB.L   #$C1FC8,D2            ; Sinon D2=D2-C1FC8
                  BRA     DONE


DONE:             RTS                           ; retour avec en D2 la Raw.Pos. Réel

END:
```

Si vous préférez la version déjà compilée, ci-joint sous forme de suite Hexa.

DELTA

```
B2 FC 16 76 66 00 00 08 06 82 00 00 04 00 B2 FC 18 0E 66 00 00 08 04 82
00 01 6E 78 B2 FC 16 1E 66 00 00 08 04 82 00 06 D0 A4 B4 91 66 00 00 66
0C 91 00 00 18 9C 67 00 00 5C B2 FC 17 D6 6D 00 00 0C 04 82 00 0E 70 78
60 00 00 4A B2 FC 17 7E 6D 00 00 0C 04 82 00 01 1F 74 60 00 00 38 B2 FC
16 7E 6D 00 00 1E 04 82 00 00 4C 18 60 00 00 26 4E 71 4E 71 4E 71 4E 71
4E 71 42 79 00 00 0C 20 4E 75 B2 FC 16 26 6D 00 00 0C 04 82 00 0C 1F C8
60 00 00 02 4E 75
```

Et de l'utilisation de la plage mémoire **19C4 → $1C5A** pour notre **trackloader custom.**
Le code est commenté pour une meilleur compréhension ☺

TRACKLOADER_AlphaOnev2[2005].s **// 566 Octets**

```
init:   move.w  $dff01c,oldintena
        move.w  $dff01e,oldintreq
        bset    #7,oldintena
        bset    #7,oldintreq
        move.w  #$7fff,$dff09a          ; interrupts aus.
        move.w  #$7fff,$dff09c          ;
;*************************************************************
        lea     $dff000,a6
        lea     buffer(pc),a0
        lea     mfmbuffer,a2
        move.l  #5*$1600,d0
        move.l  #0,d1
        move.l  #$0,d2
        jsr     TRACKLOADER
;*************************************************************
        move.w  oldintena(pc),$dff09a
        move.w  oldintreq(pc),$dff09c
        moveq   #0,d0
        rts
oldintena:      dc.w    0
oldintreq:      dc.w    0
buffer:         blk.b   130000,0
mfmbuffer:      blk.w   6400,0


; ===========================================================================
; HARDWARE-DISKLOADER (c) ALPHA ONE 2005. v2.0
; ************************************
; IMPROVED TO READ FROM ODD BYTEPOSITIONS.
; PRO VERSION -> WITH TRACKCOUNTER.
; IN: A6=$DFF000
;     A2=MFMBUFFER.L
;     A0=BUFFER.L
;     D0=LENGTH.L
;     D1=TRACKNR.L
;     D2=BYTEOFFSET.L
TRACKLOADER:
        ;-------------------------------------------------------------------   /!\ à charger en $19FA
        ; MOD for Shadow Of The Beast 2
        ;-------------------------------------------------------------------
        MOVE.L  $C1A,A2                 ; Recup DSKPTH from C1A
        LEA             $DFF000,A6      ; Nécéssaire
        MOVE.L  (A1),D2                 ; Recup Raw Pos
        MOVE.L  D2,D4                   ; Copy Raw Pos. in D4
        MOVE.L  4(A1),D0                ; Recup Length
        ADD.L   D0,D4                   ; D4 = Final.Position apres Trackload

        CMP.L   #$84468,D4              ; Compare par rapport à la valeur du limiteur de 'side' d'origine
        BLT     CORRECTION              ; Si c'est plus petit, alors on saute les prochaines ?tapes
        SUB.L   #$84468,D4              ; On enlève de l'équation ‘une side complète’

CORRECTION:
        DIVU.W  #$189C,D4               ; D4 / Taille Custom
        AND.L   #$FF,D4                 ; Nettoyage D4
        MOVE.W  D4,$C20                 ; Sauvegarde 'PisteFinal' pour SOTB2
        ;-------------------------------------------------------------------
        JSR     $9B40                   ; GoSub #CalculDelta RawPos reel      (9B40 -> 9BB2)
        ;-------------------------------------------------------------------
        MOVE.L  D2,D1                   ; Copy Raw Pos. in D1
        DIVU.W  #$1600,D1               ; D1 / Taille réel concernée
        AND.L   #$FF,D1                 ; Nettoyage D1 // TRACKNR.L
        MOVE.L  D1,D3                   ; Copy D1 in D3
        MULU.W  #$1600,D3               ; Delta
        SUB.L   D3,D2                   ; Calcul Offset // BYTEOFFSET.L
        ;-------------------------------------------------------------------
                LEA     $BFD100,A4              ; DRIVESELECT REGISTER
                LEA     $BFE001,A5              ; DRIVESTATUS REGISTER
                LEA     CURRENTTRACK(PC),A3     ; HOLDS THE ACTUAL TRACKPOS
                MOVEQ   #0,D7                   ; D7 = BYTECOUNTER
                ADD.L   D2,D0                   ; BYTES TO READ + BYTEOFFSET
                MOVE.L  D0,D3
                DIVS.W  #$1600,D3
                SWAP    D3                      ; --> Add, bug correction on track position
                TST.W   D3                      ;
                BNE.B   WITHOUT                 ;
                SWAP    D3                      ;
                SUBQ.B  #$01,D3                 ;
                BRA     WITH                    ;
WITHOUT:        SWAP    D3                      ;
```

```
WITH:           MOVE.B  D3,1(A3)                ; <--
                MOVE.B  #$7D,(A4)               ; SWITCH MOTOR DRIVE 0 ON
                NOP                             ; ======================
                NOP
                MOVE.B  #$75,(A4)
                BSR.W   DELAY
                BSR.W   DISKREADY
                CMP.B   #$FF,(A3)
                BNE.B   MOVETOCYLINDER
                BRA             NEXT
                ;-------------------------------------------------------------------
                CLR.W   $C20                    ; RAZ Compteur Piste
                RTS                             ; Annulation $1A96
                ;-------------------------------------------------------------------
NEXT:
                MOVE.B  #0,(A3)
                BSET    #1,(A4)                 ; CHOSE DIRECTION TOWARDS 0
MOVETOZERO:     BTST    #4,(A5)                 ; HEAD ON CYLINDER 0?
                BEQ.B   MOVETOCYLINDER
                BSR.W   MOVEHEAD                ; MOVE HEAD + DELAY
                BRA.B   MOVETOZERO
MOVETOCYLINDER: BSET    #2,(A4)                 ; CHOOSE HEAD HIGH
                BTST    #0,D1                   ; EVEN OR ODD TRACK?
                BEQ.B   HEADOK                  ; TRACK IS EVEN, ALRIGHT!
                BCLR    #2,(A4)                 ; CHOOSE HEAD LOW
HEADOK:         MOVEQ   #0,D3
                MOVEQ   #0,D5
                MOVE.B  D1,D3                   ; NEW TRACKNUM -> D3

                BRA             NEXT2
                ;-----------------------------------------------------
                RTS                             ; Annulation $1ACA
                ;-----------------------------------------------------

NEXT2:
                MOVE.B  (A3),D5                 ; CURRENT TRACK -> D5
                MOVE.B  D1,(A3)                 ; REPLACE CURRENT TRACK
                LSR.W   #1,D3                   ; GET CYLINDER NUM NEW TRACK
                LSR.W   #1,D5                   ; GET CYLINDER NUM CUR TRACK
                SUB.W   D3,D5
                BEQ.B   READTRACK
                BMI.B   OTHERDIRECTION
                BSET    #1,(A4)                 ; CHOOSE DIRECTION TOWARDS 0
                BRA.B   CHOSEN

OTHERDIRECTION: BCLR    #1,(A4)                 ; CHOOSE DIRECTION OUTWARDS 0
                NEG.W   D5
CHOSEN:         BSR.W   MOVEHEAD
                SUBQ.B  #1,D5
                BNE.B   CHOSEN
READTRACK:      BSR.W   DISKREADY               ; EINEN TRACK LESEN
                MOVE.W  #$8210,$96(A6)          ; =================
                MOVE.W  #$7F00,$9E(A6)
                MOVE.W  #$8500,$9E(A6)
                MOVE.W  #$4489,$7E(A6)
                MOVE.W  #$4000,$24(A6)
                MOVE.L  A2,$20(A6)
                MOVE.W  #$9900,$24(A6)
                MOVE.W  #$9900,$24(A6)
                MOVE.W  #$2,$9C(A6)
                NOP             ; Pour retomber sur nos pattes en 1B2E
                NOP             ;
                BRA             TRACKREADY
                ;-----------------------------------------------------------
                MOVEM.L D0-D7/A0-A6,-(A7)       ; On sauve tous les registres
                BSR     TRACKLOADER             ; Patch pour garder les appels en $1B2E d'origine de SOTB2
                MOVEM.L (A7)+,D0-D7/A0-A6       ; et on restaure le tout.
                RTS
                ;-----------------------------------------------------------

TRACKREADY:     BTST    #1,$DFF01F
                BEQ.B   TRACKREADY
                MOVE.W  #$4000,$24(A6)
                MOVEQ   #0,D5                   ; DECODE TRACK

DECODE:         MOVE.L  A2,A1                   ; ============
                MOVE.L  #$55555555,D4
```

```
FINDSYNC:       CMP.W   #$4489,(A1)+
                BNE.B   FINDSYNC
                CMP.W   #$4489,(A1)
                BEQ.B   FINDSYNC
                MOVE.L  (A1),D3
                MOVE.L  4(A1),D1
                AND.L   D4,D3
                AND.L   D4,D1
                ASL.L   #1,D3
                OR.L    D1,D3
                ROR.L   #8,D3
                CMP.B   D5,D3
                BEQ.B   SECTORFOUND
                ADD.L   #1086,A1
                BRA.B   FINDSYNC

SECTORFOUND:    ADD.L   #56,A1
                MOVE.L  #(512/4)-1,D6
DECODESECTOR:   MOVE.L  512(A1),D1
                MOVE.L  (A1)+,D3
                AND.L   D4,D3
                AND.L   D4,D1
                ASL.L   #1,D3
                OR.L    D1,D3
                LEA     STORE(PC),A3
                MOVE.L  D3,(A3)
                MOVEQ   #4-1,D3

LOOP:           CMP.L   D7,D0
                BEQ.B   READREADY
                CMP.L   D7,D2
                BGT.B   BELOW
                MOVE.B  (A3),(A0)+
BELOW:          ADDQ.L  #1,A3
                ADDQ.L  #1,D7
                DBF     D3,LOOP
                DBF     D6,DECODESECTOR
                ADDQ.B  #1,D5
                CMP.B   #11,D5
                BNE.W   DECODE
TRACKDONE:      CMP.L   D7,D0                   ; TRACK DONE, GET ONTO NEXT.
                BEQ.B   READREADY               ; ==========================
                BTST    #2,(A4)
                BEQ.B   ONTONEXT
                BCLR    #2,(A4)
                BSR.W   DELAY
                BRA.W   READTRACK
ONTONEXT:       BSET    #2,(A4)
                BSR.W   DELAY
                BCLR    #1,(A4)
                BSR.W   MOVEHEAD
                BRA.W   READTRACK
READREADY:      MOVE.B  #$FD,(A4)               ; SWITCH MOTOR DRIVE 0 OFF
                NOP                             ; ========================
                NOP
                MOVE.B  #$E7,(A4)
                LEA     CURRENTTRACK(PC),A3
                MOVE.B  1(A3),D0
                ADD.B   D0,(A3)
                RTS
DELAY:          CLR.B   $300(A4)                ; DELAY ROUTINE
                MOVE.B  #$19,$400(A4)           ; =============
                MOVE.B  #$1,$D00(A4)
WAIT:           BTST    #0,$C00(A4)
                BEQ.B   WAIT
                BCLR    #0,$D00(A4)
                RTS
DISKREADY:      BTST    #5,(A5)                 ; WAIT FOR DISK-READY
                BNE.B   DISKREADY               ; ====================
                RTS
MOVEHEAD:       BCLR    #0,(A4)
                BSR.W   DELAY
                BSET    #0,(A4)
                BSR.W   DELAY
                RTS
CURRENTTRACK:   DC.B    $FF,0
STORE:          DC.L    0

END:
; ===========================================================================
```

On tape/charge le tout sous ASM-ONE, on assemble, on sauve les binaires compilé : **TRACKLOADER_AlphaOne_P2** et **DELTA**

Si vous préférez la version déjà compilée, ci-joint sous forme de suite Hexa.

TRACKLOADER_AlphaOnE_P2

```
24 79 00 00 0C 1A 4D F9 00 DF F0 00 24 11 28 02 20 29 00 04 D8 80 0C 84
00 08 44 68 6D 00 00 08 04 84 00 08 44 68 88 FC 18 9C 02 84 00 00 00 FF
33 C4 00 00 0C 20 4E B9 00 00 9B 40 22 02 82 FC 16 00 02 81 00 00 00 FF
26 01 C6 FC 16 00 94 83 49 F9 00 BF D1 00 4B F9 00 BF E0 01 47 FA 01 E4
7E 00 D0 82 26 00 87 FC 16 00 48 43 4A 43 66 08 48 43 53 03 60 00 00 04
48 43 17 43 00 01 18 BC 00 7D 4E 71 4E 71 18 BC 00 75 61 00 01 7C 61 00
01 98 0C 13 00 FF 66 20 60 00 00 0A 42 79 00 00 0C 20 4E 75 16 BC 00 00
08 D4 00 01 08 15 00 04 67 06 61 00 01 7C 60 F4 08 D4 00 02 08 01 00 00
67 04 08 94 00 02 76 00 7A 00 16 01 60 00 00 04 4E 75 1A 13 16 81 E2 4B
E2 4D 9A 43 67 16 6B 06 08 D4 00 01 60 06 08 94 00 01 44 45 61 00 01 42
53 05 66 F8 61 00 01 32 3D 7C 82 10 00 96 3D 7C 7F 00 00 9E 3D 7C 85 00
00 9E 3D 7C 44 89 00 7E 3D 7C 40 00 00 24 2D 4A 00 20 3D 7C 99 00 00 24
3D 7C 99 00 00 24 3D 7C 00 02 00 9C 4E 71 4E 71 60 00 00 10 48 E7 FF FE
61 00 FE C6 4C DF 7F FF 4E 75 08 39 00 01 00 DF F0 1F 67 F6 3D 7C 40 00
00 24 7A 00 22 4A 28 3C 55 55 55 55 0C 59 44 89 66 FA 0C 51 44 89 67 F4
26 11 22 29 00 04 C6 84 C2 84 E3 83 86 81 E0 9B B6 05 67 08 D3 FC 00 00
04 3E 60 D8 D3 FC 00 00 00 38 2C 3C 00 00 00 7F 22 29 02 00 26 19 C6 84
C2 84 E3 83 86 81 47 FA 00 A4 26 83 76 03 B0 87 67 46 B4 87 6E 02 10 D3
52 8B 52 87 51 CB FF F0 51 CE FF D6 52 05 0C 05 00 0B 66 00 FF 90 B0 87
67 26 08 14 00 02 67 0C 08 94 00 02 61 00 00 32 60 00 FF 1A 08 D4 00 02
61 00 00 26 08 94 00 01 61 00 00 46 60 00 FF 06 18 BC 00 FD 4E 71 4E 71
18 BC 00 E7 47 FA 00 44 10 2B 00 01 D1 13 4E 75 42 2C 03 00 19 7C 00 19
04 00 19 7C 00 01 0D 00 08 2C 00 00 0C 00 67 F8 08 AC 00 00 0D 00 4E 75
08 15 00 05 66 FA 4E 75 08 94 00 00 61 00 FF D2 08 D4 00 00 61 00 FF CA
4E 75 FF 00 00 00 00 00
```

Part 21 Hack du fichier Last_Load-Disk1 part #01

Comme nous l'avons expliqué à travers divers chapitres et organigrammes de ce tuto, le dernier chargement effectué sur l'original N°1 va charger en mémoire du code puis le recopier à l'adresse **$256** et l'exécuter.
Le fameux 'Last_Load-Disk1' qui est… coriace.

Pour une meilleure compréhension globale nous allons charger nos données ripées à cette adresse.
Il est nécessaire de revoir le chapitre '*Part 9 Analyse du dernier code chargé : Last_Load*' afin de bien comprendre le code que nous allons modifier.

Le but étant de préserver au maximum le code d'origine et de patcher ce qui n'est plus nécessaire.

Insérez maintenant le disk contant le fichier Last_Load dans le lecteur interne.
Tapez :
LM Last_Load, 256 **// Chargé en mémoire $256 -> $9DCE**

Insérez maintenant le disk contenant le fichier TRACKLOADER_AlphaOne_P2 et DELTA dans le lecteur interne.
Et comme prévu on insère notre *trackloader* en **$19C4**
LM TRACKLOADER_AlphaOne_P2, 19F8
LM DELTA, 9B40

Ci-dessous :

- **En bleu** le code d'origine que l'on garde.
- **En rouge** le code d'origine que l'on ne garde pas.
- **En orange** le code d'origine que l'on garde car il est désactivé via une autre routine.
- **En violet** notre **nouveau code** de remplacement à l'aide de la commande **A** de l'Action Replay.
- Et toujours en **fond Jaune** les **commandes** à taper dans l'AR.

```
256     BRA     1DE0            ; Code d'origine qui saute en 1DE0
;==============================
1DE0    MOVE.B  C0.S,D0         ;
...
1E48    JSR     19C4.S          ; GoSub ➔ #Trackloader_2_Init
1E4C    JSR     1A96.S          ; GoSub ➔ #Retour_T00
1E50    JSR     8C30            ; GoSub ➔ #BASE_INSERT_DISK_ASKED
...
1E48    BSR     9D3E            ; On Préféra supprimer les routines ci-dessus, cela nous permet de gagner
                                ; quelques octets et on décale le reste.
1E4C    MOVE.W  #1A0,DFF096     ;
1E54    BSR     1C5C            ; Saute en 1C5C vers #Phase_De_Chargement_#1
```

S'en suit des phases de chargement que l'on ne touche pas comme prévu, notre routine **DELTA**, s'occupera de calculer le *Raw.Pos* réel avant le *trackload* proprement dit.

```
1C5C    LEA     46FB4,A0        ; Load_Menu_01
1C62    LEA     17D6.S,A1       ;
1C66    BSR     1B2E            ; Signature Hexa = 0003A91C46244304A924783188E0649088F1EB8400482075E0B6AA411301800
1C6A    BSR     1824            ;

1C6E    LEA     70000,A0        ; Load_Menu_02
1C74    LEA     17DE.S,A1       ;
1C78    BSR     1B2E            ; Signature Hexa = 00000250C1C0303200A06617C0080081880703380100 3A3ADBF0040020CF803F
1C7C    BSR     1824            ;

1C80    LEA     45F34,A0        ; Load_Menu_03
1C86    LEA     17E6.S,A1       ;
1C8A    BSR     1B2E            ; Signature Hexa = 0000043980D7046478DA0F679AE33C6E0448C02100B33BDCC79A03DDDB9EBE7D
1C8E    BSR     1824            ;

1C92    LEA     50000,A0        ; Load_Menu_04
1C98    LEA     17EE.S,A1       ;
1C9C    BSR     1B2E            ; Signature Hexa = 0000012421809 04B8A0A2188008C4042100048C0022183430283888A8181CD83
1CA0    BSR     1824            ;

1CA4    LEA     515DC,A0        ; Load_Menu_05
1CAA    LEA     $17F6.S,A1      ;
1CAE    BSR     1B2E            ; Signature Hexa = 00000665E080500020003808C0003E024005BD05C203FE030107FE8A81077F99
1CB2    LEA     42000,A1        ;
1CB8    BSR     1826            ;

1CBC    LEA     50398,A0        ; Load_Menu_06
1CC2    LEA     17FE.S,A1       :
1CC6    BSR     1B2E            ; Signature Hexa = 00000269AC054004C00BDEA00BC9A8A01B90782D2C023041CA05B443CC737227
1CCA    BSR     1824            ;

1CCE    LEA     457A4,A0        ; Load_Menu_07
1CD4    LEA     1806.S,A1       ;
1CD8    BSR     1B2E            ; Signature Hexa = 000003FDE5FF07F9078B40A078967827ED28B08F40700463F81FB0904C6C0482
1CDC    JMP     1824            ; GoTo #Decomp/Decrypt puis retournera après le BSR $1E54 effectué plus haut
```

Notre prochain chargement si on croit le processus original devrait concerner les données '**Load_Menu_08**'
Sauf que dans notre cas, ces données se trouvent sur **notre disk2**.

On va devoir donc se servir et modifier la **routine** de demande de **changement de disk** de Shadow Of The Beast II

Donc, de retour en **$1E58** on change le code.

```
1E58    JSR     8C30            ; GoSub #BASE_INSERT_DISK_ASKED car les fichiers 8 et 9 sont sur notre disk2
```

Et on regarde dans ce 'processus de changement de disk' si rien ne va poser problème avec notre Hack.

#BASE_INSERT_DISK_ASKED

```
8C30    MOVEQ   #1,D0                   ; D0=01                         // Choix du disk demandé (00=Disk1, 01=Disk2)

#BASE_INSERT_DISK_ASKED_WITHOUT_FLOPPY_DISK_MARKER
8C32    MOVE.W  D0,A0.S                 ;
8C36    BSR     9D3E                    ; GoSub ➔ #Wait_BB6
8C3A    LEA     DFF000,A6               ;
8C40    MOVE.W  #1A0,96(A6)             ; Rien à
8C46    MOVE.L  #8D1E,DFF080            ; changer ici.
8C50    MOVE.W  #1200,100(A6)           ;
8C56    CLR.L   102(A6)                 ; On
8C5A    CLR.L   108(A6)                 ; Ne
8C5E    MOVE.L  #2C81F4C1,8E(A6)        ; Touche
8C66    MOVE.L  #3800D0,92(A6)          ; à

8C6E    LEA     30000, A0               ; rien.
8C74    MOVE.W  #7CF,D1                 ;
8C78    CLR.L   (A0)+                   ; etc
8C7A    DBF     D1,8C78                 ; etc

8C7E    TST.W   A0.S                    ;
8C82    BEQ     8C8C                    ; GoTo ➔ #CHECK_DISK
8C84    LEA     8D2A,A0                 ;
8C8A    BRA     8C92                    ; GoTo ➔ #CHECK_DISK_A0
```

On continue toujours dans le process de 'demande changement de disque'.

#CHECK_DISK

```
8C8C    LEA     8DEE,A0                 ;

#CHECK_DISK_A0
8C92    LEA     30E16,A1                ; Idem !
8C98    MOVEQ   #6,D1                   ; On ne
8C9A    MOVEQ   #0D,D2                  ; change rien
8C9C    MOVE.W  (A0)+,(A1)+             ; ici
8C9E    DBF     D2,8C9C                 ;
8CA2    ADDA.W  #C,A1                   ; etc
8CA6    DBF     D1,8C9A                 ;

8CAA    LEA     31021,A1                ;
8CB0    LEA     8EB2,A0                 ; etc
8CB6    MOVEQ   #6,D1                   ;
8CB8    MOVEQ   #15,D2                  ; et
8CBA    MOVE.B  (A0)+,(A1)+             ;
8CBC    DBF     D2,8CBA                 ; etc
8CC0    ADDA.W  #12,A1                  ;
8CC4    DBF     D1,8CB8                 ;

8CC8    MOVE.W  #F00,DFF182             ; Affiche du message 'PLEASE INSER BEAST DISK'
8CD0    BSR     9D3E                    ; GoSub ➔ #Wait_BB6
8CD4    MOVE.W  #8180,DFF096            ;

#DISK2_OR_DISK1_INSERTED?
8CDC    BSR     1ACA                    ; GoSub ➔ #DF0__SIDE_DOWN__MOTOR_OFF__DIR_EXT       // inutile mais déjà patché
8CE0    TST.B   BB4.S                   ;
8CE4    BEQ     8CE0                    ;
8CE6    BSR     19C4                    ; GoSub ➔ #Trackloader_2_Init
8CEA    BSR     1A96                    ; GoSub ➔ #Retour_T00                               // inutile mais déjà patché
...
8CE6    NOP                             ; Anciennement GoSub ➔ #Trackloader_2_Init
8CE8    NOP                             ;
8CEA    BSR     1A96                    ; On le garde, de toute façon il est désactiver dans notre TrackLoader
...

8CEE    LEA     B0.S,A0                 ;
8CF2    LEA     8D16,A1                 ; Pos. Tableau    // Check_Signature DISK, 1 WORD
                                        ; :008D16 00 00 18 9C      //adr. Signature Disk qu'on ne change pas.
                                        ; Car elle est géré directement dans notre routine DELTA (voir plus bas)


8CF8    BSR     1B2E                    ;
8CFC    TST.W   A0.S                    ;
8D00    BEQ     8D0C                    ; Si bits à zéro alors GoTo ➔ #Check_Signature_DISK_01
```

#CHECK_DISK (suite)

Alors, il est évident que la '**signature disque'** n'est plus la même que celle d'origine car nous avons changé l'organisation des données sur les disques.

Plusieurs solutions sont possibles pour connaitre ces 2 words de chaque disque.

- En live en posant un *BreakPoint* dans l'exécution de notre code.
- Utiliser un éditeur hexa (par exemple *Deksid*) et aller regarder ce qu'il y a en position *$189C* sur nos deux disques.
- Utiliser l'action Replay pour lire la Track en question et aller regarder à l'offset concernée, etc.

BREF… Quoi qu'il en soit je vous donne directement ces deux valeurs.

Disk1 ➔ 64 A6 **Disk2 ➔ 44 CC**

#Check_Signature_DISK_02

```
8D02    CMPI.W   #D0D2,B0.S          ; Check Signature Disk2
8D02    CMPI.W   #44CC,B0.S          ; Notre nouvelle signature

8D08    BNE      8CDC                ; Si signature Disk2 pas trouvée, on boucle sur #DISK2_OR_DISK1_INSERTED?
8D0A    RTS                          ; Signature trouvé, On retourne au code
```

#Check_Signature_DISK_01

```
8D0C    CMPI.W   #4,B0.S             ; Check Signature Disk1
8D0C    CMPI.W   #64A6,B0.S          ; Notre nouvelle signature

8D12    BNE.B    8CDC                ; Si différent, on repart pour un tour
8D14    RTS                          ; E.T Retour Maison
```

Le processus de demande de changement de disk étant terminé, la routine retourne en $1E6C où l'on continue notre chargement, à savoir les données `Load_Menu_08` et `Load_Menu_09` Présent sur notre disque2.

On va devoir tout décaler à cause de notre précédent **JSR 8C30** en $1E58

```
1E5E    LEA      4324C,A0            ; Load_Menu_08
1E64    LEA      180E.S,A1           ;
1E68    BSR      1B2E                ; Signature Hexa = 4AB8041E67162078041E20B8042620780420220B8042A700021C0041E41F80B1A
1E6C    MOVE.L   #7C000,C1A.S        ;
1E74    BSR      1D48                ; On saute sur la seconde phase attendu (on ne change rien)
```

Nous voilà maintenant dans la seconde phase de chargement.

LoadPhase_#1_End_Part1/2

```
1D48    MOVE.W   #775,D0
...
1D56    LEA      2A8.S,A0            ; Load_Menu_09
1D5A    LEA      17C6.S,A1           ;
1D5E    BSR      1B2E                ; Signature Hexa = 0000006037906060 40FF807848B5D9E050211706142C24000609274211C42570
1D62    MOVE.W   #0,dff180
1D6A    BSR      1824
...
1DDE    RTS                          ; Retour en $1E86 vu que l'on a tout décalé.
```

D'origine, si on avait pas tout décalé, on serait revenu en $1E80 et l'on effectuera le branchement vers la routine **#Retour_T00**
Saut bien sûr que l'on ne remettra pas car il n'a aucun intérêt dans notre Crack et cela nous permet encore de gagner quelques précieux octets.

Pour rappel :

```
1E80    BSR      1A96                ; GoSub ➔ #Retour_T00
```

Donc, nous voilà de retour en **$1E78** et l'on doit charger les prochaines data '**Load_Menu_10'** qui se trouvent sur…. **Le disk1**
L'on doit donc encore faire appel à la routine de changement de disque mais en demandant le disk1 cette fois ci.

```
1E78    MOVEQ    #0,D0               ; D0=00                          // Choix du disk demandé (00=Disk1, 01=Disk2)
1E7A    JSR      8C32                ; GoSub #BASE_INSERT_DISK_ASKED car le 'fichier' 10 est sur notre disk1
```

De retour sur cet ensemble de **routine de changement de disk**, il faut maintenant charger les data '**Load_Menu_10'**

```
1E80    LEA      641DC,A0            ; Load_Menu_10
1E86    LEA      1676.S,A1           ;
1E8A    BSR      1B2E                ; Signature Hexa = 00004C19441E01FB52EEBFB9E1A93F80945B24B05FC0EB206DBC0074CF981007
1E8E    BSR      1824                ;
```

Et tout le reste des chargements va se faire maintenant sur le **disk2**, donc encore une fois, un **JSR 8C30** à effectuer.

```
1E92    JSR     8C30            ;
1E98    LEA     70400,A0        ; Load_Menu_11
1E9E    LEA     167E.S,A1       ;
1EA2    BSR     1B2E            ; Signature Hexa = 0000271DF050378932041AB57E0081AF0503A82E68250321D001030407C05839
1EA6    BSR     1824            ;

1EAA    BSR     1CE0            ; GoSub ➔ #Donnée_déja_chargée     // Voir quelques lignes plus bas
1EAE    NOP                     ; Pour retomber sur nos pieds et supprimer le GoSub #DF0__SIDE_DOWN__MOTOR_OFF__DIR_EXT

1EB0    LEA     DFF180,A0       ; A0=DFF180
1EB6    MOVEQ   #F,D0           ; D0=$F, compteur
1EB8    CLR.L   (A0)+           ; ➔ Efface (A0)
1EBA    DBF     D0,1EB8         ; ← D0=D0-1 et tant que D0 est différent de -1, on boucle (donc 16 fois)
1EBE    BSR     5710            ; GoSub ➔ #Decrypt_RAW
...
```

On se retrouve ensuite sur la dernière partie du chargement, en '**Phase_De_Chargement_#2_2/2**'
C'est-à-dire en *1CE0*

```
1CE0    TST.B   BE0.S           ; Test des Bits de $BE0, marqueur donnée déjà chargé
1CE4    BEQ     1D46            ; Si égale à 0, alors GoTo ➔ 1D46 (qui est un RTS)

#Phase_De_Chargement_#2_2/2
1CE6    LEA     EB7E,A0         ; Load_Menu_12
1CEC    LEA     1686.S,A1       ;
1CF0    BSR     1B2E            ; Signature Hexa = 000007FD55536181F6C1031007350001FA0F01054BEA7C121795BC00D821E107
1CF4    BSR     1824            ;

1CF8    LEA     F920,A0         ; Load_Menu_13
1CFE    LEA     168E.S,A1       ;
1D02    BSR     1B2E            ; Signature Hexa = 0000E9D9B686A6042EEEF60446D60E20E0EA16717274B3B734631FA61C8AA478
1D06    BSR     1824            ;

1D0A    LEA     26488,A0        ; Load_Menu_14
1D10    LEA     1696.S,A1       ;
1D14    BSR     1B2E            ; Signature Hexa = 00001408EFF1915FC81C0C34284FF8280FFF84107F800070797962B70F2F9004
1D18    BSR     1824            ;

1D1C    LEA     27BC2,A0        ; Load_Menu_15
1D22    LEA     169E.S,A1       ;
1D26    BSR     1B2E            ; Signature Hexa = 000E0D18191A150C02FEFDFE9A010200FDFAF9F7F8FD000304030202000010408
1D2A    BSR     1958            ;

1D2E    LEA     28AB2,A0        ; Load_Menu_16
1D34    LEA     16A6.S,A1       ;
1D38    BSR     1B2E            ; Signature Hexa = 000B48F6F7F7FC00030A110D17161A2120212016120E0702FEF6F2EEE8E8E6E4
1D3C    BSR     1958            ;

1D40    MOVE.B  #1,BE1.S        ;
1D46    RTS                     ; ← Retour en 1EAE
```

Bon ! Arrivé à ce stade nous avons patché l'ensemble des chargements jusqu'au Menu.

Nous voilà de retour en *1EAE* ou l'ensemble de la palette est mis à Zero.
S'en suit tout une phase de travail sur le code en mémoire qui ne nous intéresse pas.
Revoir si nécessaire le code en mémoire *1EC2* → … '*Part 9 Analyse du dernier code chargé : Last_Load*'

Part 21b Hack du fichier Last_Load-Disk1 part #02

Avec au final l'affichage du Menu principal du jeu et l'attente d'appui sur FIRE pour commencer le chargement du Level1
A savoir la routine : **#CHARGEMENT_LEVEL1**

On regarde de plus près le code et on change ce qui doit l'être.

#CHARGEMENT_LEVEL1

```
24CC    MOVE.L   #70400,C1A.S            ;
24D4    LEA      515DC.A0                ; Chargement Level#1_01/06
24DA    LEA      16AE.S,A1               ;
24DE    BSR      1B2E                    ;
24E2    BSR      1824                    ;
24E6    LEA      4D800,A0                ;
24EC    LEA      16B6.S,A1               ; Chargement Level#1_02/06
24F0    BSR      1B2E                    ;
24F4    BSR      1824                    ;
24F8    TST.B    C0.S                    ; Marqueur donnée déjà chargé ?    // Plus utile
24FC    BNE      24CC                    ; GoTo ➔ #CHARGEMENT_LEVEL1
24F8    NOP                              ; Anciennement le TST
24FA    NOP                              ;
24FC    NOP                              ; Anciennement le Branchement vers le rechargement du Level1 depuis le début

24FE    LEA      2AFBE,A0                ;
2504    LEA      16BE.S,A1               ; Chargement Level#1_03/06
2508    BSR      1B2E                    ;
250C    BSR      1824                    ;

2510    LEA      AC00,A0                 ;
2516    LEA      16C6.S,A1               ; Chargement Level#1_04/06
251A    BSR      1B2E                    ;
251E    MOVE.B   #1,F3BE                 ;

2526    LEA      29654,A0                ;
252C    LEA      16D6.S,A1               ; Chargement Level#1_05/06
2530    BSR      1B2E                    ;
2534    BSR      1824                    ;

2538    LEA      6DE1C,A0                ;
253E    LEA      16CE.S,A1               ; Chargement Level#1_06/06
2542    BSR      1B2E                    ;
2546    BSR      1824                    ;

254A    MOVEA.L  C1A.S,A0                ;
254E    BSR      9B24                    ; GoSub ➔ #RAZ_Trackloader                 // Voir quelques lignes plus bas
2552    BNE      2526                    ; Boucle rechargement Level1
2552    NOP                              ; Anciennement le Branchement vers le rechargement du Level1 05/06
...
```

#RAZ_TRACKLOADER

```
9B24    MOVE.L   A0,1818.S               ; Le seul 'truc' à garder.
                                         ; Et encore, il ne sert que dans les sous-routines qu'on va supprimer...
                                         ; Bref… aller, on le garde...

9B28    BSR      9B40                    ; GoSub ➔ #Pre_Base_TrackLoadX             // On sup
9B2C    BSR      9B8A                    ; GoSub ➔ #Motor_ON                        // On sup
9B30    BSR      9BBA                    ; GoSub ➔ #Retour_T00                      // On sup
9B34    BSR      9C24                    ; GoSub ➔ #Pre_Base_TRK_X2                 // On sup
9B38    BSR      9C94                    ; GoSub ➔ #Base_Conf_Interupt              // On sup
9B3C    BRA      9CDC                    ; GoTo ➔ #Traitement_Trait_01              // On sup

9B28    NOP                              ;
9B2A    NOP                              ;
9B2C    NOP                              ;
9B2E    CLR.W    C20                     ; RAZ du compteur de piste à Zero
9B34    NOP                              ; RAZ de D0 et Set du flag Z=0 au passage
9B36    NOP                              ;
9B38    NOP                              ;
9B3A    NOP                              ;
9B3C    MOVEQ    #0,D0                   ;
9B3E    RTS                              ; et on rentre à la maison
```

Par contre, il ne faut pas oublier que nous avons déplacer des fichiers du Disk Original 1 vers notre Disk2

```
07052    CMPI.W #$40B0, $02C0.W              ; Check si on descend vers le niveau 'CRYSTAL_CAVERNS'
07058    BGE.W $0000724A                     ; Si c'est le cas on branche en $724A
➔
0724A    MOVE.L $02CA.W, $0C1A.W
07250    BSR.W $00008C2C                     ; GoTo #Positionne_Marqueur_Insert_Disk1
07254    BSR.W $00009D3E                     ; Gosub #Erase_BB6
07258    MOVE.W #$01A0, $00DFF096            ; Conf DMACON, DSKEN enable, ALL DMA enable
07260    LEA $000515DC, A0                   ; A0=515DC=DSKPTH
07266    BSR.W $00009B24                     ; GoTo #Update_DSKPTH      // dejà patché
0726A    BNE.B $000072B0                     ; Check flag Z de CCR si 0 alors
                                             ; on branche sur le 3eme Trackload du niveau Crystal...
                                             ; Pas Bon ça, à désactiver patcher par un NOP
0726A    NOP
0726C    MOVE.B   #1,F3BE
07274    MOVEQ    #2,D0
07276    MOVEQ    #1,D1
07278    LEA      1626.S,A1                  ; Pour le prochain Trackload de Crystal Level 01/11
...
```

Il nous reste à désactiver les des routines de **CRC** vue en '**Part 14B'** de ce tuto.

```
0722C    CMP.W #$3D69, D0                    ;
07230    BNE.B $00007226                     ;

0722C    MOVE.W #$3D69,D0                    ; On patch directement D0 avec la bonne valeur attendue.
07230    NOP                                 ; et on désactive le BNE
```

Et le second.

```
07D52    CMP.W #$3D69, D0                    ;
07D56    BNE.B $00007D4C                     ;

07D52    MOVE.W #$3D69,D0                    ; On patch directement D0 avec la bonne valeur attendue.
07D56    NOP                                 ; et on désactive le BNE.
```

Il n'est pas nécessaire de désactiver les appels que l'on a écrasé avec notre *TrackLoader*, revoir tableau en 'part 20a' de ce tuto si nécessaire.
à savoir : $4ED0 et $8CEA pour la routine **#Retour_T00**
Car on a déjà patché en interne, dans notre trackloader, cette routine ☺

Par contre, il nous reste encore un électron libre en 4ECC pour un appel vers **#Trackloader_2_Init** qui elle n'est pas patchée.
Il faut désactiver cet appel.

```
04ECC    BSR      19C4                       ; Anciennement #Trackloader_2_Init

04ECC    NOP                                 ; Branchement
04ED0    NOP                                 ; désactivé.
```

Un peu plus haut dans notre hack de ce chapitre, nous avons décalé/modifier le code depuis 1E48 jusqu'à 1EAE
Ce qui concerne les routines suivantes :
1ED0 #Base_LastLoad
1E66 #Phase_De_Chargement_#2_1/2
1E74 #Phase_De_Chargement_#2_1/2_bis

La question que l'on doit se poser c'est : Existe-t-il dans le code original des appels vers des sous-routines ?
Si oui, il faut bien sûr les patcher car le code attendu n'est plus au même endroit.
Cela se fait avec les commandes :
FA 1ED0 ➔ **256 BRA 1DE0** // Ok, ça ne change pas, c'est le début de la routine et elle est tjs au même endroit
FA 1E66 ➔ Rien trouvé, impeccable.
FA 1E74 ➔ **5086 BRA 1E74** // Celle-là est problématique car le code attendu à cette adresse a été déplacé.

#Phase_De_Chargement_#2_1/2_bis

```
1E74     MOVE.L   #7C000,C1A.S       ; Copie $7C000 à l'adresse $C1A, MAJ du DSKPTH
1E7C     BSR      1D48               ; GoSub ➔ #LoadPhase_#1_End_Part1/2        // TrackLoad de Load_Menu_09
1E80     BSR      1A96               ; GoSub ➔ #Retour_T00
```

Maintenant cette portion de code se trouve en $1E6C
Du coup, tout ce que l'on a à faire c'est de remplacer l'adresse du **BRA**
5086 BRA 1E74
Par un
5086 BRA 1E6C

On sauve notre nouveau fichier patché : **SM Last_Load-MOD, 256 9DCE**
On écrit le tout sur notre **disque1 de crack fraichement inséré dans DF0**

```
O 00, 20000 60000
RT !152 1 20000
LM 1:Last_Load_mod, 20000+200           ; PsygnosisMod    Size=$9B78$200=$9D78      $20F7E/$1600= !7,16 donc !8
WT !152 !8 20000
```

Prochaine section : test de notre crack ☺

Part 22 Test de notre crack

Insérez notre disk1 fraichement réalisé dans le lecteur et rebooter votre amiga.
Faites 2 tests.

Le premier est de laisser les animations se jouer jusqu'au bout et poursuivre le chargement jusqu'au Menu.
(en changeant comme demandé de disk bien sûr)

Le second test est de bybasser les animations en appuyant sur Fire et poursuivre aussi le chargement jusqu'au Menu.
Ceci validera notre 1er partie du crack, tout fonctionne ☺

Une fois sur le menu du jeu, appuyer de nouveau sur FIRE pour démarrer une partie.

On est parti pour quelques tests, déjà une série de trackload en partant à gauche.
Puis on revient sur nos pas et on se dirige vers Crystal cavern puis, une fois le Crystal Level atteint, (qui valide donc le trackload), remonter les marches pour revenir au début.

Bon… jusqu'ici vous ne devriez avoir aucun problème.

On test aussi la fin d'une partie, appuyer sur la touche **F10** pendant le jeu, cela permet d'abandonner la partie en cours.
L'intro de fin se charge et se lance (le disk2 est bien demandé si au préalable le dernier chargement était sur le Disk1)
Il revient ensuite bien au niveau du menu après les changements attendus de disquette.
On relance une partie… c'est bon, tout fonctionne. ☺

Il nous reste donc plus qu'à jouer au jeu pour le finir ou trouver d'autres problèmes et les patcher bien sûr.
(ce qui ne sera pas le cas pour info)

Le mieux est donc de tester complétement le jeu jusqu'à sa fin.
Il existe quelques vidéos sur internet sur le sujet ainsi que des 'soluces' complètes de où aller et quand, etc.
Un conseil…, activer avant le cheat mode (voir section au début de ce tuto).