Tutoriel	**AmigaCracking : BLOOD MONEY**
Protection	**MFM + Checksum**
Auteur Original	**Gi@nts**
Version	**28/08/2018 Gi@nts**
Vérification/Correction	**V2, Testé et fonctionnel de A à Z**

* BLOOD MONEY CRACK TUTORIEL *

Table des matières

Matériels nécessaires 3
Général Info 3
Agencement des disquettes Amiga v1.3 6
Le format MFM v1.1 9
WinUAE 12
Part 1 X-copy 13
Part 2 Analyse de l'image IPF 14
Part 3 Analyse et modification du bootblock 16
Part 4 Taille d'une Track 19
Part 5 Analyse du Trackloader(1) 20
Part 6 Ready ? Go 23
Part 7 Rip - Disk 1 26
Part 8 Disk 2 – Nouveau Trackloader(2) 31
Part 9 Cheats 34
Part 10 Rip - Disk 2 36
Part 11 Les HighScores 53
Part 12 Topo analyse du RIP Disk 2 55
Part 13 Piste/code des HighScores 56
Part 14 Protection Track 00 sur le Disque2, nouveau trackloader(3) 60
Part 15 Protection Track 00 : Création Dump FULL 65
Part 16 Protection Track 00 sur le Disque1, nouveau trackloader(4) 71
Part 17 Récapitulatif : Patch, Checksum, Cheat 77
Part 18 Création crack Disque 2 78
Part 19 Création crack Disque 1 81
Part 20 Création de la Table d'appel pour nos TrackLoaders 98
Part 21 Debug 99
Part 22 Modification du trackloader d'AlphaOne v1 106
Part 23 Modification du trackloader d'AlphaOne v2 109

Matériels nécessaires

1) Un AMIGA 500 de base sans extension mémoire ou l'émulateur WINUAE
2) Une Carte ACTION REPLAY (ou ça ROM Image) selon configuration utilisée.
3) Un second lecteur de disquette est conseillé mais pas obligatoire. (Dans ce cas, tuto a adapter avec un lecteur)
4) Le jeu Original Blood Money ou son image CAPS (SPS 0302)
5) De bonne base sur l'assembleur 68000 en général.
6) Le logiciel Xcopy-pro en disquette ou son image disk.
7) L'assembleur ASM-One (pas obligatoire)
8) Source et/ou binaire du TrackLoader d'AlphaOne v2004 (celui qui fait 404 Bytes) (inclus dans ce tuto)

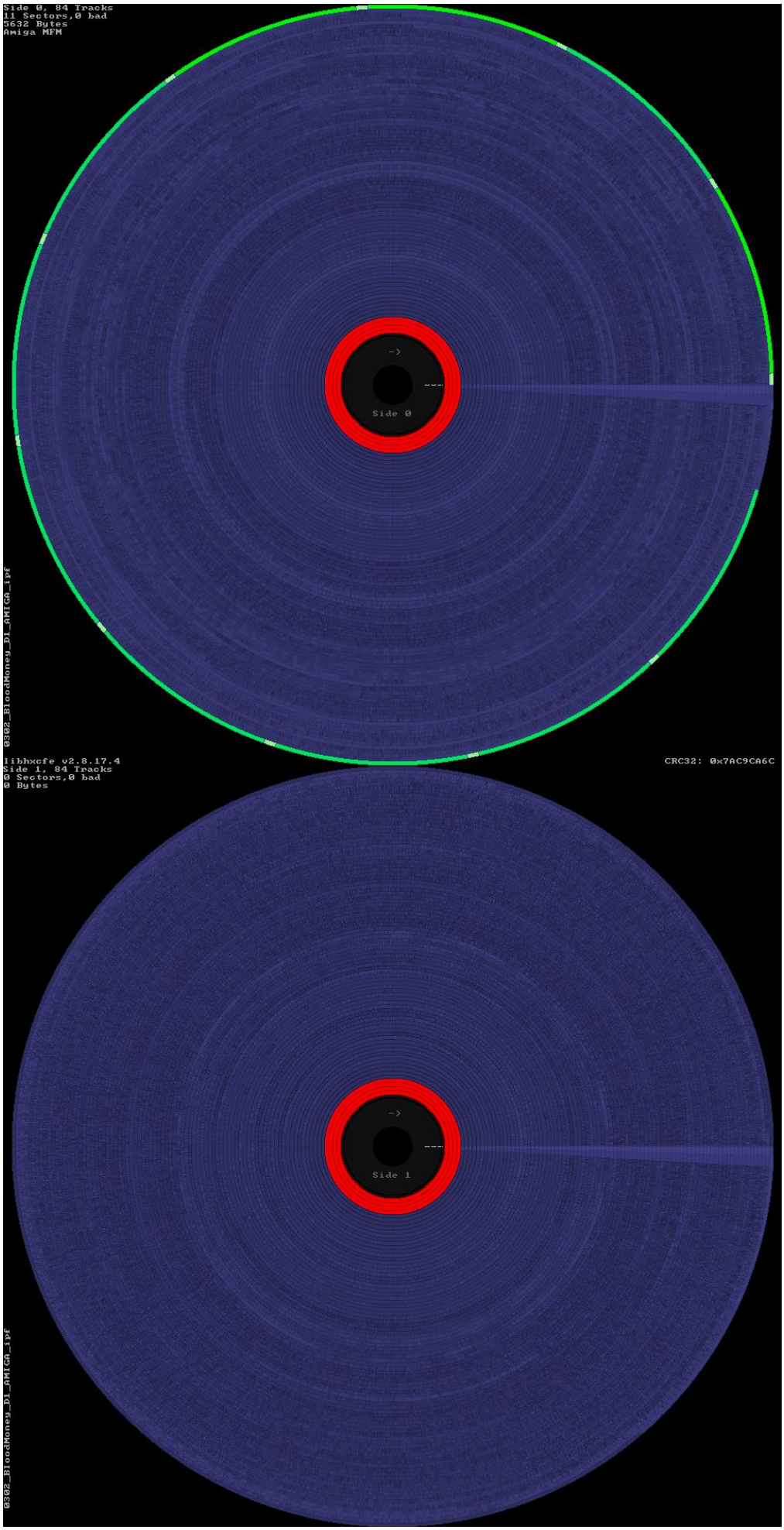
Général Info

Gros Tutorial en approche, sortez vos pelles, ça va être long et détaillé.
A noter que ce tuto est basé uniquement sur mon travail et propose une version différente de ce que j'ai pu trouver sur le web. Qu'il est à mon sens très complet et complexe. Il est fortement conseillé de suivre chronologiquement le tuto sinon vous risquer d'être perdu dans la compréhension.

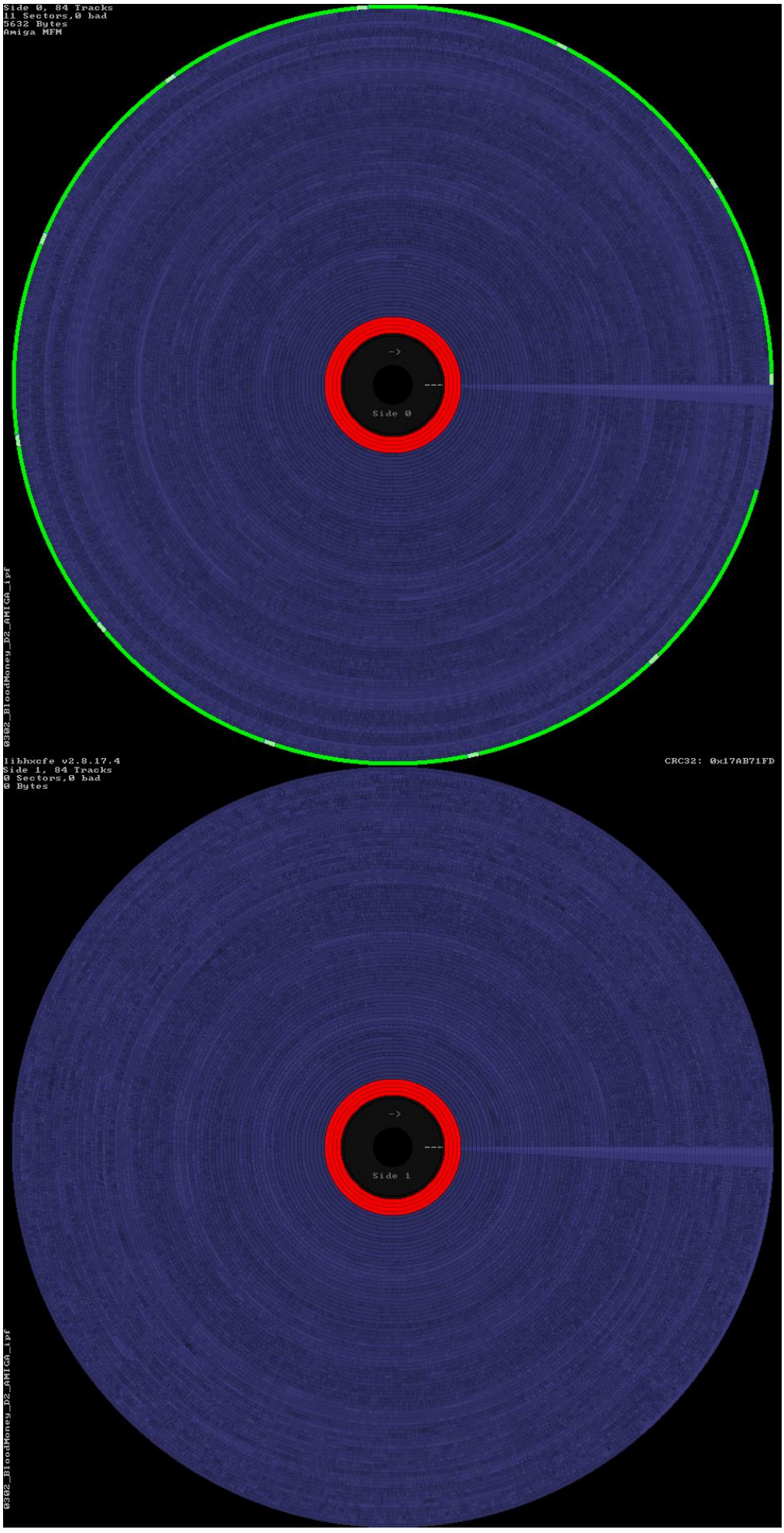
Bon Tuto.

Gi@nts

Disque 1


Side 0, 84 Tracks
11 Sectors,0 bad
5632 Bytes
Amiga MFM
Side 0
0302_BloodMoney_D1_AMIGA_ipf
libhxcfe v2.8.17.4
Side 1, 84 Tracks
0 Sectors,0 bad
0 Bytes
CRC32: 0x7AC9CA6C
Side 1
0302_BloodMoney_D1_AMIGA_ipf

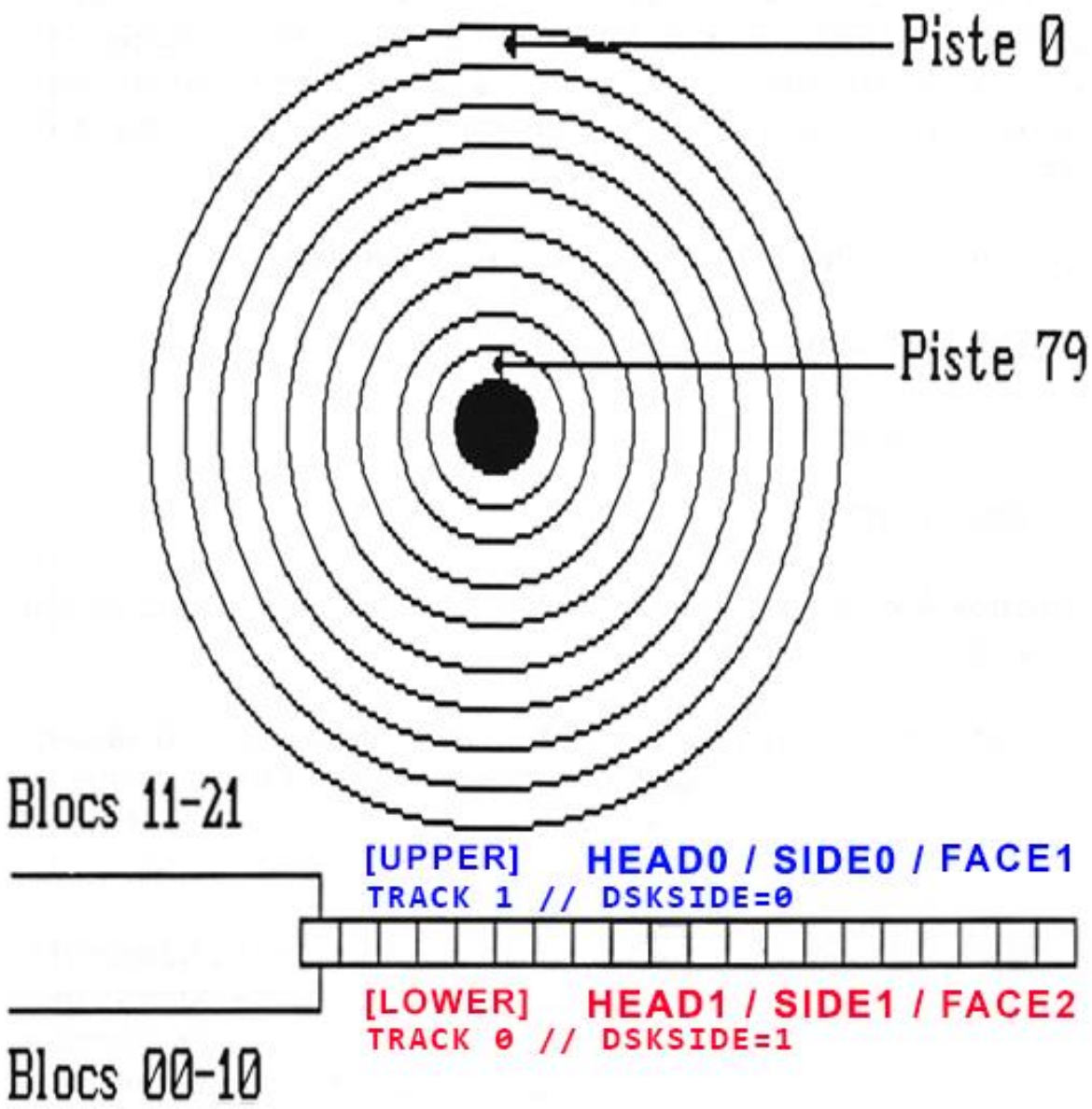
Disque 2


Side 0, 84 Tracks
11 Sectors,0 bad
5632 Bytes
Amiga MFM
Side 0
0302_BloodMoney_D2_AMIGA.ipf
libhxcfe v2.8.17.4
Side 1, 84 Tracks
0 Sectors,0 bad
0 Bytes
CRC32: 0x17AB71FD
Side 1
0302_BloodMoney_D2_AMIGA.ipf

Agencement des disquettes Amiga v1.3

En France :

On utilise des termes comme : *piste, bloc, secteur, face*…
Piste : 0 à 79 Certaine disquette pousse jusqu'à 81 voire 82 pistes mais officiellement c'est 80 pistes (de 0 à 79)
Face : 0/1 ou **1/2** ou **A/B**, Dessus ou dessous tout simplement. Sur Amiga nous avons deux faces utilisées sur 99% des jeux.
Chaque piste, pour un format standard 'AmigaDOS' est composée de plusieurs *bloc ou secteur*, en général 11 **par face**.
Le terme piste peut désigner l'ensemble d'une piste (les deux 'side' du disque), ou uniquement une 'side' d'une piste.
Une piste standard *AmigaDos* est découpée en plusieurs partie appelé **bloc**, **secteur**, **sector**.



Dans d'autre pays :

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

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

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

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

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

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

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

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

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

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

On notera que :

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

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

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

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

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

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

Les secteurs

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

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

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

Mise en application sous l'AR :

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

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

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

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

Exemples :

```
RT 20 1 50000
```

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

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

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

```
RT 21 2
```

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

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

Le format MFM v1.1

Original source : *Codetapper/Action!*
Original Date : *updated: 11/2/2001* *©1997-2001 Codetapper/Action! All rights reserved.*

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

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

...	GAP	...	TRACK	...	GAP	...

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

- S E C T O R -

00	00	Sync	Sync	SectorHeader	Fill	Headerchecksum	Datachecksum	Data

00

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

Sync

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

SectorHeader

1 longWord **[8 octets MFM]**
Header du secteur

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

Fill

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

HeaderChecksum

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

DataChecksum

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

Data

512 octets **[1024 octets MFM]**
Block de DATA

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

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

Un Exemple :

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

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

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

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

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

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

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

La plupart des *TrackLoader* n'ont pas de compteur de track pour savoir qu'elle track la tête de lecture à 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)

Le registre **$BFEC01 SDR**

On va juste noter que celui permet de tester l'appui de touche du clavier
Exemple :

```
cmp.b #$37, $BFEC01              ; Test l'appuie sur la touche ALT gauche
```

Quelques exemples des valeurs de touches valides pour le registre **$BFEC01**

$3F	$3D	$39	$37	$33	$31
Shift Left	Shift Right	Control	Alternate	Amiga Left	Amiga Right

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

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

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

Après le déplacement de la tête de lecture il est important d'attendre 3ms avant de faire une nouvelle action sur le lecteur de disquette. Comme les boucles de synchro logiciel ne sont pas précise (dépends de la vitesse d'horloge de l'ordinateur qui varie d'un ordinateur à l'autre), il est recommandé d'utiliser un des timers des chipset **CIA** pour attendre les 3ms nécessaire et il est de 18 ms entre un changement de direction.

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

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

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

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

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

La valeur de ce bit affecte seulement le(s) lecteur(s) sélectionné(s).
SIDE doit être maintenant pendant 100 microSecondes avant une opération d'écriture et pendant 1,3 MilliSeconde entre un changement de face.

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

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

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

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

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

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

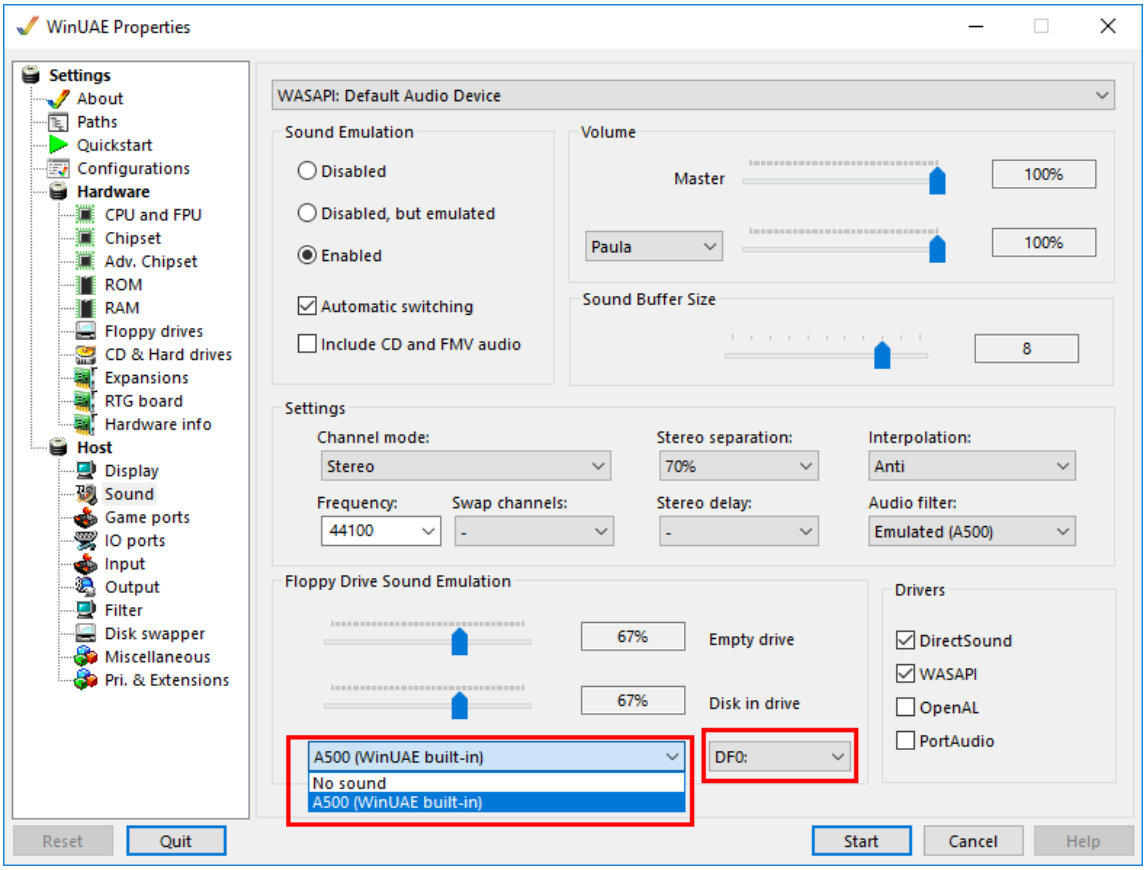
Quand le moteur d'un lecteur est passé à **ON**, vous devez attendre que celui-ci atteigne sa pleine vitesse de rotation avant d'effectuer d'autre opération. Vous pouvez vérifier ceci en lisant le bit 5 du **CIA-A** située à l'adresse : **$BFE001.** Il passe à 0 quand le lecteur a atteint sa pleine vitesse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

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

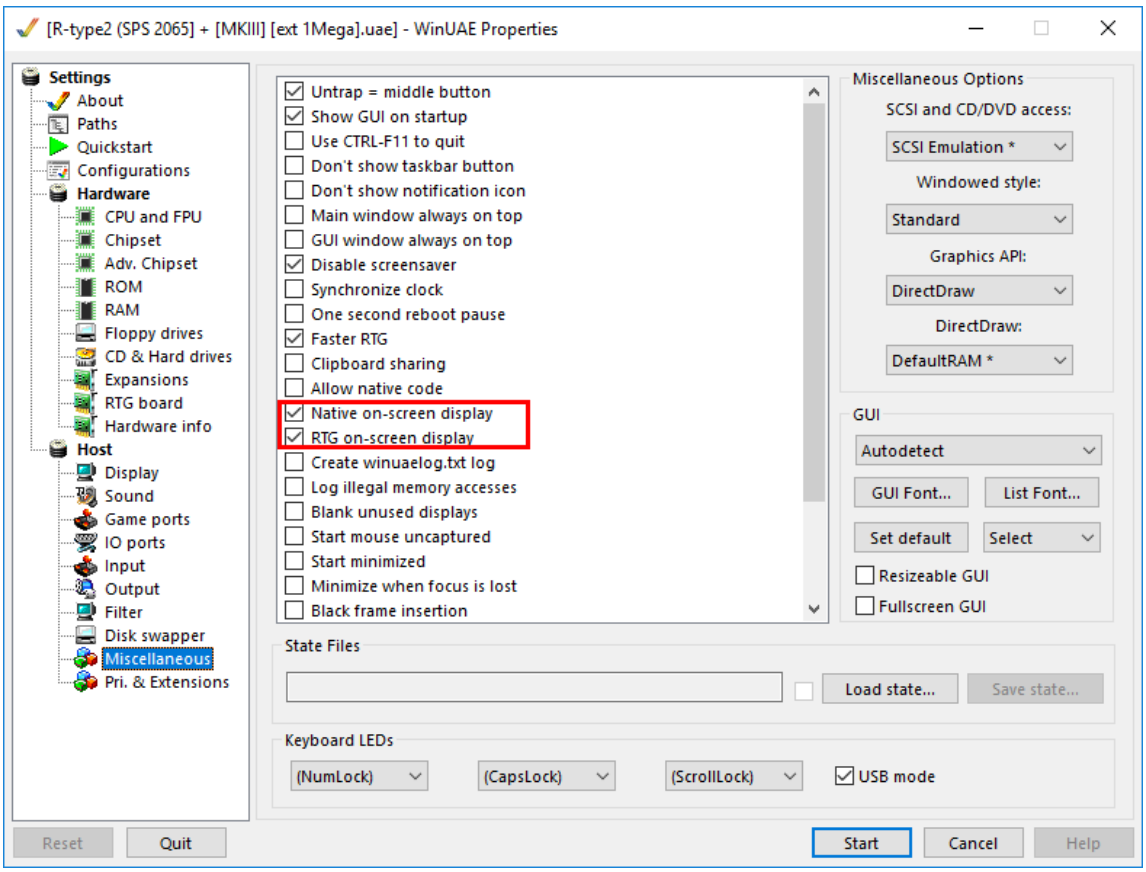
```
CIAAPRA         equ     $BFE001
CIABPRB         equ     $BFD100

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

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

WinUAE

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

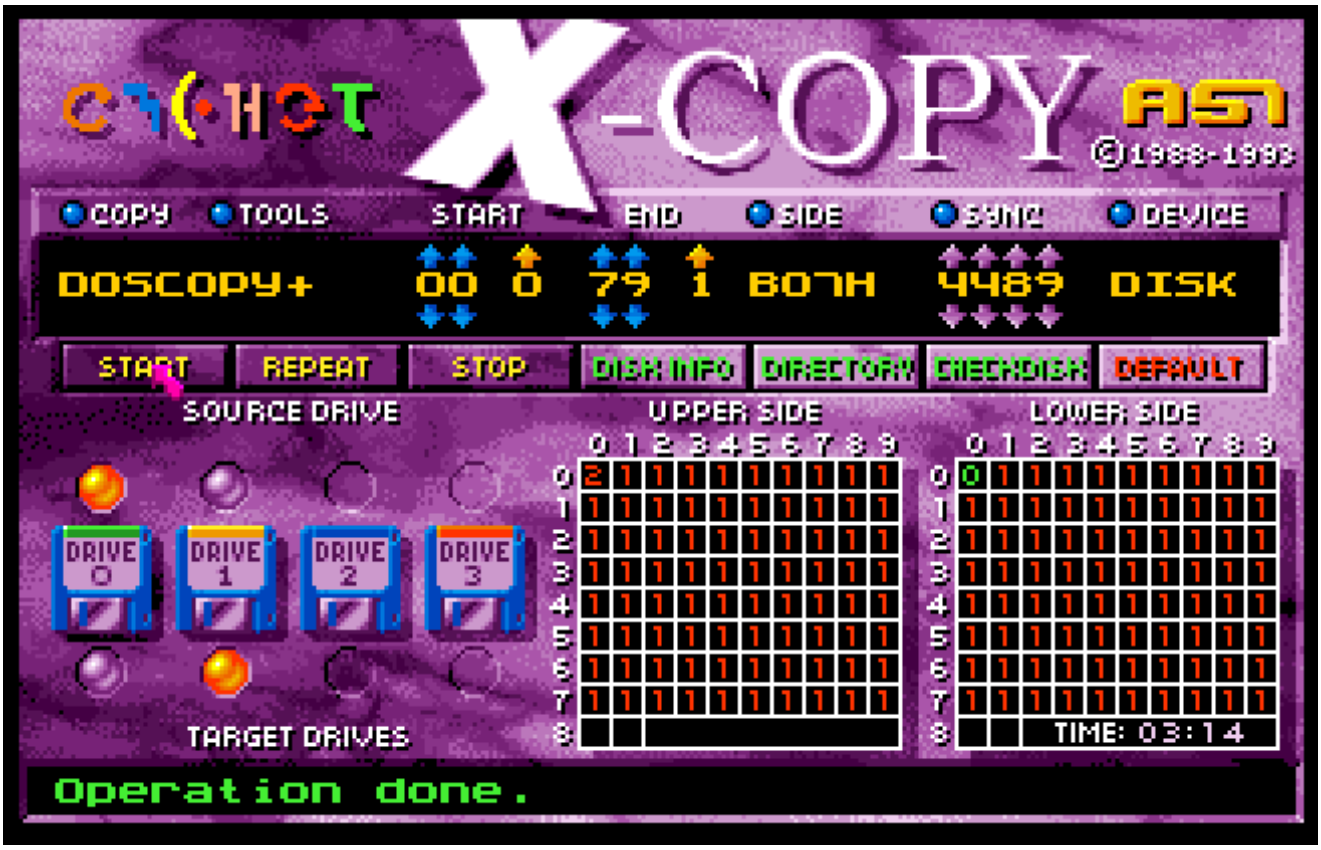


Voir même, pour plus d'information. Par exemple afficher sur qu'elle face l'on se trouve, d'activer :
Host -> Miscellaneous -> Native on-screen display AND RTG on-screen display



Part 1 X-copy

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



Même constat sur le disque 1 et disque 2, hormis la 1er track, tout le reste est dans un format bien propriétaire.
à noter que sur la seconde track, l'on passe sous *Xcopy* de l'erreur 5 à 2

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

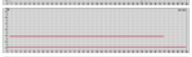
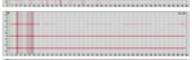
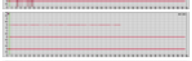
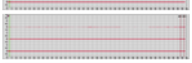
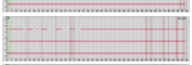
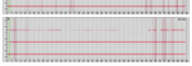
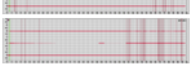
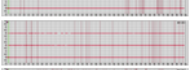
FILENAME	0302_Blood_Money_D1_AMIGA.ipf
TYPE	Floppy_Disk
ENCODER	CAPS(V1)
FILE	0302(V1)
DISK	2
TRACK	00-83
SIDE	0-1
PLATFORM	Amiga
REVOLUTION	5

Comme prévue, rien de spécial sur la premier track, on est sur un format Standard *AmigaDOS*.

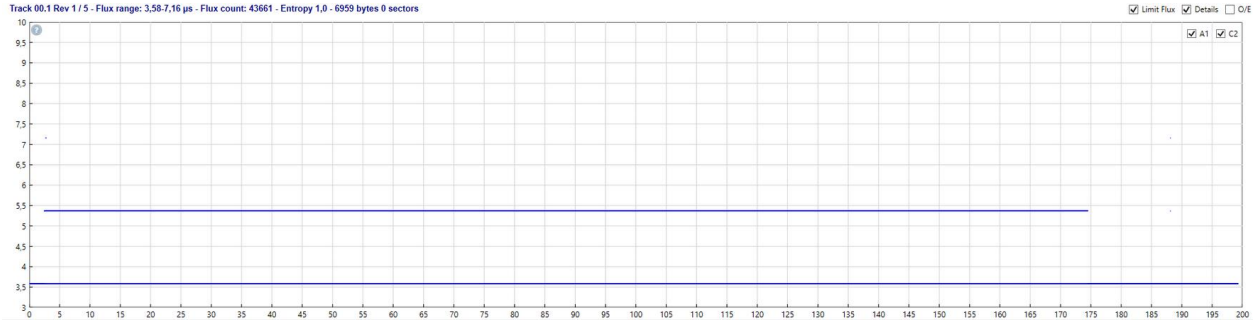
TRACK		Data Length (bytes)		Data (bits)				CRC32 of the complete Extra Data Block			Adress
Data block Description	Sector ID	Data		bytes/sector	GAP		Codage	GapDef	DataOff		Adresse
		MFM bits	bytes		MFM bits	bytes			MFM bits	bytes	
[T00.0]		6446		51568				0D55D4CB			13576-20021
#0	0	8704	545	512	0	1	MFM	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	92	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	4480	281	MFM	5892	5892	369	13896-13927

Cependant, le reste des tracks sont dans un format bien spécifique.
Noter une variante sur la **T00.1**, qui n'a pas la même *Data Length, Size de Gap* que les autres pistes.

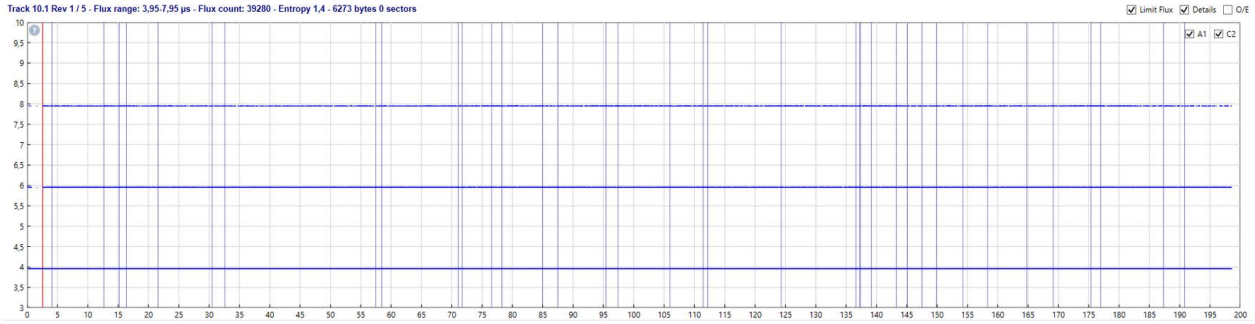
TRACK		Data Length (bytes)		Data (bits)				CRC32 of the complete Extra Data Block			Adress
Data block Description	Sector ID	Data		bytes/sector	GAP		Codage	GapDef	DataOff		Adresse
		MFM bits	bytes		MFM bits	bytes			MFM bits	bytes	
[T00.1]		6048		48384				5C0251F2			20050-26097
#0	N/A	96112	6008	N/A	15240	953	MFM	0032	0032	2	20050-20081
[T01.0]		6253		50024				8D8D7123			26126-32378
#0	N/A	99312	6208	N/A	904	57	MFM	0032	0032	2	26126-26157
[T01.1]		6253		50024				12571789			32407-38659
#0	N/A	99312	6208	N/A	904	57	MFM	0032	0032	2	32407-32438
[T02.0]		6253		50024				AF609E1F			38688-44940
#0	N/A	99312	6208	N/A	912	58	MFM	0032	0032	2	38688-38719
[T02.1]		6253		50024				592D7739			44969-51221
#0	N/A	99312	6208	N/A	912	58	MFM	0032	0032	2	44969-45000
[T03.0]		6253		50024				61540BA6			51250-57502
#0	N/A	99312	6208	N/A	912	58	MFM	0032	0032	2	51250-51281
[T03.1]		6253		50024				B33BD798			57531-63783
#0	N/A	99312	6208	N/A	904	57	MFM	0032	0032	2	57531-57562
[T04.0]		6253		50024				DAC16DDB			63812-70064
#0	N/A	99312	6208	N/A	912	58	MFM	0032	0032	2	63812-63843

TrackNumber	Size Record (bytes)	Crc	Status	Track Size	Detail Tr. Size	Start Byte	Bit	DataKey	Block	Density	Signal	Encoder	Flag	Aufit Track View
T00.0	80	23CCDD45	Good	12528	100224 bits = Data=95744 + Gap=4480	300	2401	001	11	Auto	cell_2us	0	None	
T00.1	80	01EBBCA9	Good	13919	111352 bits = Data=96112 + Gap=15240	992	7941	002	1	Auto	cell_2us	0	None	
T01.0	80	A5B8EE48	Good	12527	100216 bits = Data=99312 + Gap=904	60	484	003	1	Auto	cell_2us	0	None	
T01.1	80	A25E7F8A	Good	12527	100216 bits = Data=99312 + Gap=904	55	447	004	1	Auto	cell_2us	0	None	
T02.0	80	AC292598	Good	12528	100224 bits = Data=99312 + Gap=912	60	484	005	1	Auto	cell_2us	0	None	
T02.1	80	83614E6D	Good	12528	100224 bits = Data=99312 + Gap=912	55	446	006	1	Auto	cell_2us	0	None	
T03.0	80	B1BD74D5	Good	12528	100224 bits = Data=99312 + Gap=912	59	479	007	1	Auto	cell_2us	0	None	
T03.1	80	4D9EBA3D	Good	12527	100216 bits = Data=99312 + Gap=904	56	449	008	1	Auto	cell_2us	0	None	
T04.0	80	6A1BEE14	Good	12528	100224 bits = Data=99312 + Gap=912	60	485	009	1	Auto	cell_2us	0	None	
T04.1	80	FC351426	Good	12528	100224 bits = Data=99312 + Gap=912	56	453	010	1	Auto	cell_2us	0	None	

En détail sur la T00.1 que ce soit sur le disque original 1 ou 2



Alors que sur les autres (quand elles ne sont pas vides), on a plus quelque chose qui ressemble à ça :



Peut-être une piste de protection ? On verra ça plus tard.
Que ce soit sur le Disque1 ou le Disque2, les données semblent présentes jusqu'en T79.1
Il n'y a plus aucune donnée à partir de la piste80

T79.0	80	AD53B664	Good	12525	100200 bits = Data=99312 + Gap=888	59	473	159	1	Auto	cell_2us	0	None
T79.1	80	9C1931AC	Good	12527	100216 bits = Data=99312 + Gap=904	53	427	160	1	Auto	cell_2us	0	None
T80.0	80	E4FD048E	Good	0	0 bits = Data=0 + Gap=0	0	0	161	0	Noise	cell_2us	0	None
T80.1	80	24FA257F	Good	0	0 bits = Data=0 + Gap=0	0	0	162	0	Noise	cell_2us	0	None
T81.0	80	044BE323	Good	0	0 bits = Data=0 + Gap=0	0	0	163	0	Noise	cell_2us	0	None
T81.1	80	04F5A844	Good	0	0 bits = Data=0 + Gap=0	0	0	164	0	Noise	cell_2us	0	None
T82.0	80	FEE1CD95	Good	0	0 bits = Data=0 + Gap=0	0	0	165	0	Noise	cell_2us	0	None
T82.1	80	3EE6EC64	Good	0	0 bits = Data=0 + Gap=0	0	0	166	0	Noise	cell_2us	0	None
T83.0	80	1E572A38	Good	0	0 bits = Data=0 + Gap=0	0	0	167	0	Noise	cell_2us	0	None
T83.1	80	44EAB232	Good	0	0 bits = Data=0 + Gap=0	0	0	168	0	Noise	cell_2us	0	None

Part 3 Analyse et modification du bootblock

La seule piste lisible est la piste du *bootblock* au format *AmigaDos*, nous allons donc la charger et regarder de plus près tout ça.
Insérer notre **disquette 1 de backup** préalablement crée avec *Xcopy* dans le lecteur.

#RT alias Read Track, permet le chargement de la track 0 à 1 (1ère piste de la face 0)
#D, alias Désassemble
Taper : `RT 0 1 50000` *puis* `D 50000+c`

```
rt 0 1 50000
Disk ok

d 50000+c
~05000C MOVEA.L 00000004,A6
~050012 MOVEA.L #7DBB8,A5
~050018 MOVE.W  #2,1C(A1)
~05001E MOVE.L  A5,28(A1)
~050022 MOVE.L  #400,2C(A1)
~05002A MOVE.L  #1200,24(A1)
~050032 JSR     -1C8(A6)
~050036 JMP     0007DBB8
;---------------------------------------
~05003C ORI.B   #0,D0
~050040 ORI.B   #0,D0
~050044 ORI.B   #0,D0
~050048 ORI.B   #0,D0
~05004C ORI.B   #0,D0
~050050 ORI.B   #0,D0
~050054 ORI.B   #0,D0
~050058 ORI.B   #0,D0
~05005C ORI.B   #0,D0
~050060 ORI.B   #0,D0
~050064 ORI.B   #0,D0
~050068 ORI.B   #0,D0
```

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

Il semblerait que l'on ait un simple appel au *trackdisk.device* avec un chargement des données de l'adr. physique du disque $400
d'une longueur de $1200 vers l'adresse mémoire $7DBB8
Puis nous avons un jolie jmp à cette adresse.

```
05000C MOVEA.L      4,A6                ; Pas vraiment nécessaire, au boot A6 est déjà sur l'ExecBase mais bon
050012 MOVEA.L      #7DBB8,A5           ; On met $7DBB8 dans A5
050018 MOVE.W       #2,1C(A1)           ; Mode lecture (ak, 2 dans $1C de (A1))
05001E MOVE.W       A5,28(A1)           ; $28(a1) = IO_DATA, adresse de destination en chip          $7DBB8
050022 MOVE.L       #400,2C(A1)         ; $2c(a1) = IO_OFFSET, secteur de départ en octet            $400
05002A MOVE.L       #1200,24(A1)        ; $24(a1) = IO_Length, longueur de l'I/O en octet            $1200
050032 JSR          -1C8                ; Et on exécute le trackdisk.device, le chargement s'effectue.
050036 JMP          0007DBB8            ; Une fois le code chargée en mémoire, ou saute vers celui-ci
;=================================
```

On regarde ce qu'il y a en $400
Taper : `D 50000+400` (et on descend jusqu'à trouver du code)

```
rt 0 1 50000
Disk ok

d 50000+400
~050400 ORI.B   #0.D0
  ↓     ↓     ↓
~05043C LEA     50448(PC),A0
~050440 MOVE.L  A0,00000084
~050446 TRAP    #1
~050448 MOVE.W  #2700,SR
~05044C MOVEA.L #80000,A7
~050452 MOVEA.L #DFF000,A6
~050458 MOVE.W  #7FFF,9A(A6)
~05045E MOVE.W  #7FFF,96(A6)
~050464 MOVE.W  #7FFF,9E(A6)
~05046A MOVE.W  #9500,9E(A6)
~050470 MOVE.W  #4489,7E(A6)
~050476 BSR     000506IE
~05047A BSR     000505D4
~05047E BSET    #2,00BFD100
~050486 MOVEA.L #7A120,A0
~05048C MOVEQ   #1C,D0
~05048E MOVEQ   #2,D1
~050490 BSR     0005049A
~050494 JMP     0007A120
;---------------------------------------
```

On a ici plusieurs choses intéressantes.
Après un passage en SuperUser, nous avons une mini configuration système disque qui est effectuée puis s'ensuit deux branchements
Aux sous routines en *$5061E* et *$505D4* (adr. relative bien sûr).

Puis on se positionne sur une face du disque, ont défini quelques variables : **A0, D0 et D1** et on branche en *$5049A*
Fort probable qu'au vue de ces quelques lignes qu'il s'agisse de l'adresse d'un *trackloader* custom.

Avant d'aller plus loin, je pense qu'il est mieux de travailler avec les adresses réelles utilisées par le code original.

On va modifier un peut le code afin d'ajouter une pause **après** l'appel du trackdisk.device et **avant** le jmp.
Cela nous permettra de travailler plus tranquillement.
#A, alias Assemble, Instruction qui va permettre de taper du code assembleur.
Taper : `A 50036`

```
^050036 MOVE.W          #F00,$DFF180    ; Fond d'écran en rouge
^05003E BTST            #6,$BFE001      ; ➔ On test l'appuie sur le bouton de la souris
^050046 BNE             50036           ; ← tant que l'on n'a pas appuyé dessus, on boucle sur notre écran rouge
^050048 MOVE.W          #000,$DFF180    ; Une fois appuyé, on met le fond d'écran en noir
^050050 JMP             7DBB8           ; et on saute comme le code original en $7DBB8
^050056 <RETURN>
```

```
a 50036
^050036 MOVE.W  #F00,$DFF180
^05003E BTST    #6,$BFE001
^050046 BNE     50036
^050048 MOVE.W  #000,$DFF180
^050050 JMP     7DBB8
^050056
```

On calcule le nouveau checksum du bootblock et on sauve le tout (toujours sur notre disque 1 de backup)
#BOOTCHK Alias Boot Check. Permet de calculer un nouveau checksum pour un bootblock
#WT, alias Write Track. Permet d'écrire une zone mémoire sur la disquette à l'adresse indiqué en cylindre.
Taper : `BOOTCHK 50000` puis `WT 0 1 50000`

```
bootchk 50000
Old checksum was 5CDAA9B4, now is set to 1CCE3E1A

wt 0 1 50000
Disk ok
```

On **reboot ensuite notre Amiga**, l'on devrait arriver sur un écran rouge.
On retire notre disquette 1 de backup pour la remplacer par la disquette 1 original

Sur **l'écran rouge** donc, **on entre dans l'AR** et on regarde le code en *$7DBB8*
Taper : `D 7DBB8` (et on descend jusqu'à retrouver notre code)

```
~07DBF4 LEA     7DC00(PC),A0
~07DBF8 MOVE.L  A0,00000084
~07DBFE TRAP    #1
~07DC00 MOVE.W  #2700,SR
~07DC04 MOVEA.L #80000,A7
~07DC0A MOVEA.L #DFF000,A6
~07DC10 MOVE.W  #7FFF,9A(A6)
~07DC16 MOVE.W  #7FFF,96(A6)
~07DC1C MOVE.W  #7FFF,9E(A6)
~07DC22 MOVE.W  #9500,9E(A6)
~07DC28 MOVE.W  #4489,7E(A6)
~07DC2E BSR     0007DDD6
~07DC32 BSR     0007DD8C
~07DC36 BSET    #2,00BFD100
~07DC3E MOVEA.L #7A120,A0
~07DC44 MOVEQ   #1C,D0
~07DC46 MOVEQ   #2,D1
~07DC48 BSR     0007DC52
~07DC4C JMP     0007A120
;=====================================
```

On va vérifier que l'appel en *$7DC48* vers *$7DC52* est bien une routine de *trackload*
Taper : `A 7DC4C`

```
^07DC4C BRA             $7DC4C          ; Boucle sur nous même
^07DC4E NOP                             ; Afin d'avoir un bout de code propre
^07DC50 NOP                             ; Idem
^07DC52 <RETURN>
```

Petite analyse de ce que l'on peut déjà voir :

```
50486   MOVEQ.L #7A120,A0       ; ➔ A0 = 7A120
                                ; Trop grand pour être une position disque, on penchera donc vers la dest. Mémoire

5048C   MOVEQ   #1C,D0          ; ➔ D0 = 1C
                                ; Trop petit pour être une position disque raw, peut-être une position 'piste'

5048E   MOVEQ   #2,D1           ; ➔ D1 = 2
                                ; Trop petit pour être une position disque raw, et trop petit par rapport à D0
                                ; pour être la 'fin', donc on pensera plus à une 'longueur'

50490   BSR     5049A           ; ➔ Appel du code potentiel du trackloader
50494   JMP     7A120           ; ➔ Saute vers $7A120 qui, est la même valeur que A0 défini plus haut
```

Logiquement, on pensera donc que **A0** est l'adresse de destination mémoire.
On va vérifier tout ça.

Jetons un coup d'œil en *$7A120*
Taper : `M 7A120`

```
M 7A120-10
:07A110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:07A120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:07A130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:07A140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

On remplit cette zone mémoire de destination potentiel, à savoir **A0**, en l'occurrence donc *$7A120*
#O, alias Fill mémory. Permet de remplir une zone mémoire avec une valeur hexa.
Taper : `O "AA", 7A120-10 7A120+10` puis `X`

Une fois de retour à notre jolie écran rouge, l'on **appuie sur le bouton de la souris**.
Après un cours chargement d'une piste (*piste 28->29 si vous avez WinUae*), on est dans notre boucle de la mort en **$7DC4C**
Entrer dans l'AR et regarder la zone mémoire en question.

#M alias memory read. Permet de voir les données en mémoire au format HEXA/ASCII.
Taper : `M 7A120`

```
M 7A120-10
:07A110 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
:07A120 41 FA 02 78 21 C8 00 84 4E 41 57 65 6C 6C 20 68 A..x!...NAWell h
:07A130 65 6C 6C 6F 20 74 68 65 72 65 20 68 61 63 6B 65 ello there hacke
:07A140 72 73 2E 20 49 27 6C 6C 20 77 61 72 6E 20 79 6F rs. I'll warn yo
:07A150 75 20 6E 6F 77 20 74 68 61 74 20 74 68 69 73 20 u now that this
:07A160 67 61 6D 65 20 68 61 73 20 61 20 4C 4F 54 20 6F game has a LOT o
:07A170 66 20 70 72 6F 74 65 63 74 69 6F 6E 2C 20 73 6F f protection, so
```

Bon clairement, **A0** indique bien la zone mémoire de destination et aux vues des donnée chargées, nous sommes sur la bonne piste :=)

Il nous reste dans l'équation **D0** et **D1** qui était avant l'appel à :
D0 : ***$1C***
D1 : ***$2***

Vue que l'on a chargé qu'*une piste*, on peut déjà écarter l'idée que **D0** serait une quantité de donnée à charger.
De plus, si vous avez *WinUae*, vous avez pu remarquer qu'il y a eu un accès disque de la piste 28 à 29 (ce qui fait 2 'pistes', 28 ET 29)
et **$1C** en décimal nous donnes **28**

A0 Correspond donc à l'adresse mémoire de destination.
D0 Correspond visiblement donc à la piste de départ.
D1 La longueur à lire en Track ou piste ? (Pour l'instant, on ne sait pas)*

Part 4 Taille d'une Track

On va refaire la même manipulation mais en changeant la destination mémoire.
Rebooter votre Amiga en réinsérant notre disquette 1 de backup dans le lecteur.
Sur l'écran rouge, retirer la disquette 1 de backup dans le lecteur pour la remplacer par la disquette 1 original.

Entrer dans l'AR et supprimer comme précédemment le jmp en *$7DC4C*

Taper : `A 7DC4C`

```
^07DC4C BRA             $7DC4C                  ; Boucle sur nous même
^07DC4E NOP                                     ; Afin d'avoir un bout de code propre
^07DC50 NOP                                     ; Idem
^07DC52 <RETURN>
```

On change la destination mémoire, **taper :** `A 7DC3E`

```
^7DC3E  MOVEA.L         #20000,A0
<RETURN>
```

On remplit la zone mémoire, **taper :** `O "AA", 20000 30000` puis `X`
Une fois de retour à notre écran rouge, l'on **appuie sur le bouton de la souris**.

Après un cours chargement d'une piste (*piste 28->29 si vous avez WinUae*), on est dans notre boucle de la mort en **$7DC4C**
On **entre dans l'AR** et on cherche la fin des données chargée en cherchant notre pattern à partir des $20000.

#F, alias FIND. Permet de rechercher une suite ascii ou hexa en mémoire.
Taper : `F "AA", 20000`

Appuyer rapidement sur la touche **<ESC>** dès que l'on a plein de résultat qui s'affichent à l'écran.

On trouve notre pattern en **$23072, $23073, $23074** ...

Donc nous avons chargé la zone mémoire de *$20000* à *$23070*
Ce qui nous donnes **$3070** soit 12400 octets en décimal

Et l'on a chargé **2** *Tracks* (dixit **D1)**
En effet **D1** ne peut être qu'une longueur en ***Track*** et non en *piste* sinon, cela nous donnerait une taille de *Track* ...vraiment trop petite 😉
Donc 23070 / 2 = 6200 soit ***$1838 Bytes***

Nous avons donc ici une taille custom de *Track* de ***$1838*** au lieu des **$1600** standard *AmigaDos*.
Soit **$238** octets en plus (568 octets de donnée en plus) pas mal.

Gardons sous le coude cette valeur de **$1838**

Noter aussi qu'avant l'appel à la routine de *trackload*, la side est définie.

```
05047E BSET             #2,$BFD100              ; CIA-B / PRB    Bit2 à 1, Side=lower
```

Le trackloader fonctionnerait t'il uniquement par Face sélectionnée ?

Petit détail non négligeable car si vous avez *WinUae*, vous avez pu remarquer que ce 1er Appel disque n'a en fait travailler qu'avec la **side lower**.

On est donc avec un *trackloader* qui semble travailler uniquement sur une side qui doit être définie avant l'appel au *trackload*
Si l'on n'est pas sur *WinUae*, on ne peut que supposer cela mais le calcul ci-dessus de la longueur d'une *Track* va dans ce sens ainsi que l'instruction BSET.

On en profite pour formater une nouvelle disquette et sauver ce 1er Appel.

#format, Permet de formater une disquette au format AmigaDOS
Taper : `FORMAT bloodmoneyD1`
Ready to format disk in drive DF0: (y/n)? y
Disk ok

Et on sauve le tout.
#SM, alias SaveMemory. Permet donc de sauver une zone mémoire vers un fichier.
Taper : `SM 28_29, 20000 20000+3070`

Part 5 Analyse du Trackloader(1)

On va revenir sur le code du *trackloader*, celui présent en adresse physique *$400* du disque et chargé lors du boot.
En clair, le code que l'on a commencé à regarder plus haut 😉

Taper : `RT 0 1 50000` puis `D 50000+400` (et on descend jusqu'à retrouver notre code)

On trouve du code à partir de l'adresse *$5043C*, ce qui nous donnes 5043C-(50000+400)=*$3C*
Code que l'on retrouvera donc en réel en *$7DBB8+3C=**$7DBF4***

Je vous mets l'analyse que l'on peut en déduire **après investigation**.

```
================================                ; #Init
7DBF4   5043C   LEA     50448(PC),A0            ; A0 pointe vers 50448 ou 7DC00
7DBF8   50440   MOVE.l  A0,84                   ; Copie A0 en $84 (84 car trap #1)
7DBFE   50446   TRAP #1                         ; On exécute le code en Supervisor
7DC00   50448   MOVE.W  #2700,SR                ; Pile Supervisor = 2700
7DC04   5044C   MOVEA.L #80000,A7               ; BLTDDAT = 80000
7DC0A   50452   MOVEA.L #DFF000,A6              ; A6=$DFF000
7DC10   50458   MOVE.W  #7FFF,96(A6)            ; Conf DMACON, clear all DMA channels and BBUSY, BZERO, BLTPRI
7DC16   5045E   MOVE.W  #7FFF,9E(A6)            ; Conf ADKCON, clear all bits of ADKCON
7DC1C   50464   MOVE.W  #7FFF,9E(A6)            ; Encore une fois pour démarrer le transfert
7DC22   5046A   MOVE.W  #9500,9E(A6)            ; Conf ADKCON, enable WORDSYNC, Disk data clock rate=2us, MFM precomp
7DC28   50470   MOVE.W  #4489,7E(A6)            ; DSKSYNC pattern=$4489
7DC2E   50476   BSR     5061E / 7DDD6           ; #MOTOR ON
7DC32   5047A   BSR     505D4 / 7DD8C           ; Retour T00
7DC36   5047E   BSET    #2,BFD100               ; CIA-B / PRB   Bit2 à 1, Side=lower
7DC3E   50486   MOVEQ.L #7A120,A0               ; --> A0 = 7A120
7DC44   5048C   MOVEQ   #1C,D0                  ; --> D0 = 1C
7DC46   5048E   MOVEQ   #2,D1                   ; --> D1 = 2
7DC48   50490   BSR     5049A / 7DC52           ; GoTo #TRACKLOADER base
7DC4C   50494   JMP     7A120                   ; --> saute vers le code copié
================================
```

```
================================                ; #TRACKLOADER base
7DC52   5049A   MOVEM.L D0-D1/A0,-(A7)          ; Sauvegarde des registres dans la pile
7DC56   5049E   BSR     50502 / 7DCBA           ; GoTo #READ TRACK
7DC5A   504A2   MOVEM.L (A7),D0-D1/A0           ; On restaure les registres de la pile
7DC5E   504A6   BSR     504D4 / 7DC8C           ; GoTo #Traitement MFM
7DC62   504AA   MOVEM.L (A7)+,D0-D1/A0          ; On restaure les registres de la pile
7DC66   504AE   CLR.W   D3                      ; On nettoie D3
7DC68   504B0   MOVEA.L A0,A1                   ; Copie l'adr. Destination mémoire présente en A0 Dans A1
7DC6A   504B2   MOVE.W  #C1B,D4                 ; D4 = $C1B
7DC6E   504B6   ADD.W   (A1)+,D3                ; ← Boucle qui va lire les 2 bytes de l'adr. A1
                                                ; et les ajouter à D3 puis A1=A1+2

7DC70   504B8   DBF     D4,504B6 / 7DC6E        ; → Boucle avec décrémentation D4,
                                                ; donc il va faire cette manip 3100
                                                ; oui 3100 car le dbf va jusqu'à -1 donc $C1C+1
                                                ; $C1B+1=$C1C= !3100 en décimal
                                                ;
                                                ; Comme on parle de word (2 bytes, donc XX XX),
                                                ; cela fait !6200  ($C1C*2) de donnée lue et traitée

7DC74   504BC   MOVE.W  (A1),D4                 ; On lit des deux dernier bytes(WORD contenue à l'adresse A1)
                                                ; donc A0+6198 et on met ça dans D4

7DC76   504BE   CMP.W   D3,D4                   ; On compare cette valeur lue, en D4,
                                                ; avec ce que l'on calculé en D3

7DC78   504C0   BNE     5049A / 7DC52           ; Non, c'est que les données lues ne sont pas 'correctes',
                                                ; on repart pour un tour, GoTo #TRACKLOADER base
                                                ;
                                                ; On a donc ici une routine de check, de checksum

7DC7C   504C4   ADDA.L  #1838,A0                ; A0 = A0+$1838 // Taille d'une track ?
7DC82   504CA   ADDQ.W  #1,D0                   ; D0 = D0+1
7DC84   504CC   SUBQ.W  #1,D1                   ; D1 = D1-1
7DC86   504CE   BNE     5049A / 7DC52           ; On Reboucle sur le début de cette routine
7DC8A   504DE   RTS                             ; E.T retour maison
================================
```

```
==============================          ; #Traitement MFM
7DC8C   504D4   MOVEM.L D0/A0,-(A7)      ; Sauvegarde D0 et A0 dans la pile
7DC90   504D8   MOVE.Q  D0,D5             ; D5 = D0, Numéro de piste
7DC92   504DA   MOVEA.L #76E5C,A1         ; A1 = 76E5C = adr buffer MFM + 4
7DC98   504E0   MOVE.W  #C1C,D7           ; Compteur D7=$C1C à savoir !3100, la moitié de !6200
                                          ; ($1838 en hexa)
                                          ;
                                          ; ➔ Début boucle, Traitement donc ici d'une 'track custom'
                                          ; d'une taille de $1838

7DC9C   504E4   MOVE.W  (A1)+,D0          ; Copie le word de l'adresse pointé par A1 (Word impair)
                                          ; vers D0 et A1=A1+2

7DC9E   504E6   MOVE.W  (A1)+,D1          ; Copie le word de l'adresse pointé par A1 (word pair)
                                          ; vers D1 et A1=A1+2

7DCA0   504E8   ASL.W   #1,D0             ; Décalage de 1 bit vers la gauche
7DCA2   504EA   ANDI.W  #AAAA,D0          ; Décodage MFM bits pair de D0
7DCA6   504EE   ANDI.W  #5555,D1          ; Décodage MFM bits impair de D1
7DCAA   504F2   OR.W    D1,D0             ; Traitement des bits paire et impair #1
7DCAC   504F4   EOR.W   D5,D0             ; Traitement des bits paire et impair #2
7DCAE   504F6   MOVE.W  D0,(A0)+          ; Finalement, on copie le word MFM décodé vers l'adresse
                                          ; mémoire de destination puis A0=A0+2

7DCB0   504F8   DBF     D7,504E4 / 7DC9C  ; ➔ Boucle de copie avec décrémentation D7,
                                          ; on traite l'intégralité d'une 'track custom'

7DCB4   504FC   MOVEM.L (A7)+,D0/A0       ; On restaure ce que l'on a sauvé dans la pile
7DCB8   50500   RTS                       ; E.T retour maison
==============================
```

```
==============================          ; #READ TRACK
7DCBA   50502   BSR     50574 / 7DD2C     ; Goto #Position Reach ?
7DCBE   50506   MOVEQ.L #76E58,A1         ; A1 = 76E58, adresse buffer MFM
7DCC4   5050C   MOVE.W  #8210,96(A6)      ; Conf DMACON, DSKEN enable, ALL DMA enable
7DCCA   50512   MOVE.L  A1,20(A6)         ; ➔ Boucle, lecture/remplissage buffer MFM
7DCCE   50516   CLR.W   24(A6)            ; conf DSKLEN,  DMA data Length clear
7DCD2   5051A   BSR     50608 / 7DDC0     ; Goto #CIA READY ?
7DCD6   5051E   MOVE.W  #4000,24(A6)      ; DMA disk off
7DCDC   50524   MOVE.W  #9842,24(A6)      ; DMA disk enable, Length DMA data=$1842
7DCE2   5052A   MOVE.W  #9842,24(A6)      ; Deux fois pour démarrer le transfert DMA

7DCE8   50530   BSR     5054A / 7DD02     ; Goto #DiskBlockFinished ?
7DCEC   50534   CMPI.L  #552A2A55,(A1)     ; Recherche la sequence 552A2A55, SyncWord dans le buffer MFM

7DCF2   5053A   BNE     50512 / 7DDCA     ; ➔ Tant que ce n'est pas bon, on boucle,
                                          ; on continue la lecture DMA dsk

7DCF6   5053E   CLR.W   24(A6)            ; conf DSKLEN,  DMA data Length=$0000
7DCFA   50542   MOVE.W  #10,96(A6)        ; Clear Disk DMA
7DD00   50548   RTS                       ; E.T retour maison
==============================
```

```
==============================          ; #DiskBlockFinished ?
7DD02   5054A   MOVE.W  1E(A6),D0         ; ➔ A6=DFF000, DFF01E=INTREQR
7DD06   5054E   BTST    #1,D0             ; On test le bit 1 de de INTREQR, à savoir :
                                          ; Level 1 Disk Block Finished Interupt

7DD0A   50552   BEQ     5054A / 7DD02     ; Test ? ➔ On boucle
7DD0E   50556   MOVE.W  #2,9C(A6)         ; clear DSKBLK(Disk Block Finished Interupt)

7DD14   5055C   RTS                       ; E.T retour maison
==============================
```

```
==============================          ; #FLOPPY READY ?
7DD16   5055E   MOVE.W  #BB8,D6           ; Boucle d'attente D6 = $BB8, $BB8=!3000
7DD1A   50562   DBF     D6,50562 / 7DD1A  ; ➔ ➔ Boucle d'attente
7DD1E   50566   BTST    #5,BFE001         ; CIA-A / PRA, On test le bit5, lecteur prêt ?
7DD26   5056E   BNE     50566 / 7DD1E     ; Non ? on boucle jusqu'a que ce soit le cas.
7DD2A   50572   RTS                       ; E.T retour maison
==============================
```

```
================================        ; #Position Reach ?
7DD2C   50574   CMP.B   7DBB7,D0        ; On compare le pointeur en 7DBB7
                                        ; (pointeur, Position Track) avec D0
7DD32   5057A   BNE     50580 / 7DD38   ; Si ce n'est pas identique, GoTo #Move Exter.
7DD36   5057E   RTS                     ; E.T retour maison
================================
```

```
================================        ; #Move Exter.
7DD38   50580   BGT     505AC / 7DD64   ; si N=0, (plus grand que), alors on va #Move Inter.
7DD3C   50584   BSET    #1,BFD100       ; CIA-B / PRB, bit1 mis à 1, DIR=vers l'extérieur
7DD44   5058C   BCLR    #0,BFD100       ; CIA-B / PRB, bit0 mis à 0, STEP, pré/déplacement tête
7DD4C   50594   BSET    #0,BFD100       ; CIA-B / PRB, bit0 mis à 1, STEP, déplacement tête
7DD54   5059C   BSR     5055E / 7DD16   ; GoTo FLOPPY READY ?
7DD58   505A0   SUBI.B  #1,7DBB7        ; On enlève 1 au pointeur de position de la tête
                                        ; situé en 7DBB7

7DD60   505A8   BRA     50574 / 7DD2C   ; GoTo #Position Reach ?
================================
```

```
================================        ; #Move Inter.
7DD64   505AC   BCLR    #1,BFD100       ; CIA-B / PRB, bit1 mis à 0, DIR=vers l'intérieur
7DD6C   505B4   BCLR    #0,BFD100       ; CIA-B / PRB, bit0 mis à 0, STEP, pré/déplacement tête
7DD74   505BC   BSET    #0,BFD100       ; CIA-B / PRB, bit0 mis à 1, STEP, déplacement tête
7DD7C   505C4   BSR     5055E / 7DD16   ; GoTo #FLOPPY READY ?
7DD80   505C8   ADDI.B  #1,7DBB7        ; On ajoute 1 au pointeur de position de la tête situé en
7DBB7
7DD88   505D0   BRA     50574 / 7DD2C   ; GoTo #Position Reach ?
================================
```

```
================================        ; #Retour T00
7DD8C   505D4   BTST    #4,BFE001       ; CIA-A / PRA, On test le bit4, TK0, Tête sur T00 ?
7DD94   505DC   BEQ     50600 / 7DDB8   ; Oui ? GoTo #Update Position Track
7DD98   505E0   BSET    #1,BFD100       ; CIA-B / PRB, bit1 mis à 1, DIR=vers l'extérieur
7DDA0   505E8   BCLR    #0,BFD100       ; CIA-B / PRB, bit0 mis à 0, STEP, pré/déplacement tête
7DDA8   505F0   BSET    #0,BFD100       ; CIA-B / PRB, bit0 mis à 1, STEP, déplacement tête
7DDB0   505F8   BSR     5055E / 7DD16   ; GoTo #FLOPPY READY ?
7DDB4   505FC   BRA     505D4 / 7DD8C   ; Boucle sur cette routine tant que l'on n'est pas sur la T00
================================
```

```
================================        ; #Update Position Track
7DDB8   50600   CLR.B   7DBB7           ; On nettoie en Byte le pointeur en 7DBB7
                                        ; (pointeur, Position Track)

7DDBE   50606   RTS                     ; E.T retour maison
================================
```

```
================================        ; #CIA READY ?
7DDC0   50608   MOVE.B  BFDD00,D6       ; D6 = BFDD00
7DDC6   5060E   MOVE.B  BFDD00,D6       ; D6 = icr register de CIA-B
7DDCC   50614   BTST    #4,D6           ; Test du bit4 (FLAG) de icr
7DDD0   50618   BEQ     5060E / 7DDC6   ; Interruption générée ? On boucle
7DDD4   5061C   RTS                     ; E.T retour maison
================================
```

```
================================        ; #MOTOR ON
7DDD6   5061E   BCLR    #7,BFD100       ; CIA-B / PRB, bit7 mis à 0, motor On
7DDDE   50626   MOVEQ   #64,D0          ; D0 = 64
7DDE0   50628   BCLR    #3,BFD100       ; ← Boucle avec le CIA-B, bit3 à 0, DF0 sélectionné
7DDE8   50630   DBF     D0,50628 / 7DDE0 ; → Boucle avec décrémentation de D0
7DDEC   50634   BSR     5055E / 7DD16   ; GoTo #FLOPPY READY ?
7DDF0   50638   RTS                     ; E.T retour maison
================================
```

```
================================        ; #STOP FLOPPY
7DDF2   5063A   BSET    #7,BFD100       ; CIA-B / PRB, bit7 mis à 1, motor OFF
7DDFA   50642   BSET    #3,BFD100       ; CIA-B / PRB, bit3 mis a 1, SEL0, DF0 désélectionné
7DE02   5064A   MOVEQ   #64,D0          ; D0 = 64
7DE04   5064C   BCLR    #3,BFD100       ; ← Boucle avec le CIA-B, bit3 à 0, DF0 sélectionné
7DE0C   50654   DBF     D0,5064C / 7DE04 ; → Boucle avec décrémentation de D0
7DE10   50658   BSET    #3,BFD100       ; CIA-B / PRB, bit3 mis a 1, SEL0, DF0 désélectionné
7DE18   50660   RTS                     ; E.T retour maison
================================
```

Le code confirme bien que c'est un *Trackloader* et qu'il **fonctionne par Side**, comprendre, il ne switch pas d'une side à l'autre en fin de *Track.*
Du coup, le terme piste ou track dans ce tuto pour **D1** signifiera la même chose dans le sens où quand l'on avance, on avance sur la même face préalablement sélectionnée.

Part 6 Ready ? Go

On revient sur notre code en *$7DC2E*

```
7DC2E   50476   BSR     5061E / 7DDD6           ; #MOTOR ON
7DC32   5047A   BSR     505D4 / 7DD8C           ; Retour T00
7DC36   5047E   BSET    #2,BFD100               ; CIA-B / PRB   Bit2 à 1, Side=lower
7DC3E   50486   MOVEQ.L #7A120,A0              ; A0 = 7A120
7DC44   5048C   MOVEQ   #1C,D0                  ; D0 = 1C
7DC46   5048E   MOVEQ   #2,D1                   ; D1 = 2
7DC48   50490   BSR     5049A / 7DC52           ; GoTo #TRACKLOADER base
7DC4C   50494   JMP     7A120                   ; ➔ saute vers le code copié
```

On va donc assez logiquement aller jeter un coup d'œil au code en **$7A120**

Rebooter votre Amiga en réinsérant notre disquette 1 de backup dans le lecteur
Sur l'écran rouge, retirer la disquette 1 de backup dans le lecteur pour la remplacer par la disquette 1 original.

Entrer dans l'AR et supprimons comme précédemment le jmp en *$7DC4C*

Taper : `A 7DC4C`

```
^07DC4C BRA     7DC4C
^07DC4E NOP
^07DC50 NOP
^07DC52
```

Et on retourne au code, t**aper :** `X`
Une fois de retour à notre écran rouge, l'on **appuie sur le bouton de la souris**.

Après un cours chargement d'une piste (*piste 28->29 si vous avez WinUae*), on est dans notre boucle de la mort en **$7DC4C**
On **entre dans l'AR** et on va regarder ça de plus prêt.

#N, alias memory read. Permet de voir/écrire les données en mémoire au format uniquement ASCII.
Taper : `N 7A120`

```
n 7A120
.07A120 A..x!...NAWell hello there hackers, I'll warn you now that this
.07A160 game has a LOT of protection, so it will be a few late nights fo
.07A1A0 r you lot. It's a mugs game anyway, you should be writing games
.07A1E0 and making loads of money like me (you too could afford a 16V As
.07A220 tra GTE), but you are obviously lacking in a few brain cells & w
.07A260 ill put a weeks work into cracking this for what? SOD ALL, thats
.07A2A0  what. Well if you do manage to crack it send me a copy: DAVE JO
.07A2E0 NES, DMA DESIGN, MINTLAW PARK, WEST FERRY, DUNDEE DD5 1RD. In re
.07A320 turn you will get sod all as usual. In the meantime I'll be thin
.07A360 king of you when I'm in Florida, spending some of my dosh.Fü'..|
.07A3A0 ....|.β..=|▒...=|▒...=|▒...=|....=|...=|D..~ |.... (...HB.S.f.
.07A3E0 Bn..3ü.....F3ü.L...H#.....PB9...A0(..".............|.,|...Q...P.
.07A420 ...A3ü.....JA...!..1=|. ..Fü".a.!Fa. ü........ |...Dp.r.a...A.#H
.07A460  (...Dr.1@..H@1@..H@....P.PHQ...A.#$-H..Bn..=|....A.%*C.#bp.3X..
.07A4A0 XIQ...A....Ö0(.?a..FU(..T(..Q...=|.E.~........ |...Hp.r.a..B....
```

On peut voir clairement un petit bout de code au début de cette adresse suivi d'un message en ascii destiné visiblement aux hackers.
On regarde ce petit bout de code .

Taper : `D 7A120`

```
7A120   LEA             7A39A(PC),A0    ; A0=7A39A
7A124   MOVE.L          A0,84.S ;
71128   TRAP            #1              ; passe en mode superviseur et continue le code en A0, à savoir 7A39A
7A12A   ########################        ; ➔ Message ascii voir au-dessus, qui va de jusqu'à 7A399 inclus
```

Le code continue en $7A39A ➔

```
7A39A   MOVE.W          #2700,SR
7A3A4   MOVEA.L #80000,A7
7A3AA   MOVEA.L #DFF000,A6
7A3AA   MOVE.W          #7FFF,9A(A6)
```

… …
…
Si on continue de regarder le code, on peut voir que qu'une adresse est vraiment appelée plusieurs fois : *$7C41C*
Et que tout le temps, juste avant son appel, ont défini **A0**, **D0** et **D1**

On va regarder si on n'a pas chargé un nouveau trackloader.
Nous avions trouvé une taille de track de *$1838*, le code de ce nouveau trackloader devrait contenir au moins cette valeur.

Taper : `F 18 38, 7A120`

```
f 18 38, 7A120
Search from: 07A120 to: C80000
07C448  07DC80
Ready.
```

On jette un coup d'œil à ces deux adresses.
Taper : `D 7C488-6` puis `D 7DC80-6`

```
d 7C448-6
~07C442 BNE     0007C41C
~07C444 ADDA.L  #1838,A0
~07C44A ADDQ.W  #1,D0
~07C44C SUBQ.W  #1,D1
~07C44E BNE     0007C41C
~07C450 RTS

d 7DC80-6
~07DC7A LINEF
~07DC7C ADDA.L  #1838,A0
~07DC82 ADDQ.W  #1,D0
~07DC84 SUBQ.W  #1,D1
~07DC86 BNE     0007DC52
~07DC8A RTS
```

```
================================
7C442   BNE     7C41C           ; un branchement en $7C41C
7C444   ADDA.L  #1838,A0        ; ajoute 1 longueur de track a A0 qui est le pointeur mémoire
7C44A   ADDQ.W  #1,D0           ; ajoute 1 a D0 qui est la piste lue
                                ; pour l'instant, c'est logique, on a avancé d'une track

7C44C   SUBQ.W  #1,D1           ; On enlève 1 à D1 qui est la longueur
                                ; Tjs logique, on a avancé de 1 donc la longueur à diminuer de 1

7C44E   BNE     7C41C           ; Encore un branchement en $7C41C
7C450   RTS                     ; retour à la maison
================================
```

```
================================
7DC7A   LINEF
7DC7C   ADDA.L  #1838,A0        ; ajoute 1 longueur de track a A0 qui est le pointeur mémoire
7DC82   ADDQ.W  #1,D0           ; ajoute 1 a D0 qui est la piste lue
                                ; pour l'instant, c'est logique, on a avancé d'une track

7DC84   SUBQ.W  #1,D1           ; On enlève 1 à D1 qui est la longueur
                                ; Toujours logique, on a avancé de 1 donc la longueur à diminuer de 1

7DC86   BNE     7DC52           ; Branchement en $7DC52 qui est le #TRACKLOADER base
7DC8A   RTS                     ; retour à la maison
================================
```

On regarde ce qu'il y a en *$7C41C*
Taper : `D 7C41C`

```
d 7C41C
~07C41C MOVEM.L D0-D1/A0,-(A7)
~07C420 BSR     0007C480
~07C424 MOVEM.L (A7),D0-D1/A0
~07C428 BSR     0007C452
~07C42C MOVEM.L (A7)+,D0-D1/A0
~07C430 CLR.W   D3
~07C432 MOVEA.L A0,A1
~07C434 MOVE.W  #C1B,D4
~07C438 ADD.W   (A1)+,D3
~07C43A DBF     D4,0007C438
~07C43E MOVE.W  (A1),D4
~07C440 CMP.W   D3,D4
~07C442 BNE     0007C41C
~07C444 ADDA.L  #1838,A0
~07C44A ADDQ.W  #1,D0
~07C44C SUBQ.W  #1,D1
~07C44E BNE     0007C41C
~07C450 RTS
;====================================
```

C'est exactement le même code que l'on à déjà vue en *$7DC52* qui est le `#TRACKLOADER base`

On cherche en mémoire tous les appels vers cette adresse.
#FA, alias Find for Addressing Upcode, On va chercher une instruction qui fait appel à une adresse mémoire.
Taper : `FA 7C41C`

```
fa 7C41C
Search from: 000000 to: C80000
07A458 BSR      0007C41C
07A4D8 BSR      0007C41C
07A4EE BSR      0007C41C
07A4FC BSR      0007C41C
07A600 BSR      0007C41C
07A60E BSR      0007C41C
07A70C BSR      0007C41C
07A71A BSR      0007C41C
07C442 BNE      0007C41C     >En plein dans la routine du trackloader
07C44E BNE      0007C41C     >En plein dans la routine du trackloader
Searched up to adr: C80000
Ready.
```

Bon, il y'a du monde...on peut donc penser logiquement que l'adresse **$7C41C** est la routine de *trackloader* principale.
On vérifie :

Taper : `D 7A458-12`

```
d 7A458-12
~07A446 BSET    #2,00BFD100
~07A44E MOVEA.L #58E44,A0
~07A454 MOVEQ   #1E,D0
~07A456 MOVEQ   #14,D1
~07A458 BSR     0007C41C
```

```
7A446   BSET    #2,BFD100                ; CIA-B / PRB  Bit2 à 1, Side=lower
7A44E   MOVEA.L #58E44,A0                ; A0 = 58E44
7A454   MOVEQ   #1E,D0                   ; $1E = !30, piste de départ D0=30
7A456   MOVEQ   #14,D1                   ; $14, $1E+$14=$32 = !50, D1=50
7A458   BSR     7C41C                    ; Et on appelle la routine de trackloader
...
```

Vous pouvez contrôler les autres appels de **$7C41C**. Tous positionnent les valeur **D0**,**D1** et **A0** avant l'appel en **$7C41C**
Noter aussi que selon l'appel, **la side est sélectionnée aussi avant.**

Et si vous avez bien suivi l'analyse du *trackloader* au début, cela confirme qu'il 'trackload' en Side.
Comprendre, il ne 'saute' pas d'une track lower a upper comme c'est le cas normalement avec le *trackdisk.device* standard *AmigaDOS.*

On regarde la fin du code de ce *trackloader* en mémoire.
Taper : `N 7C41C` et faite défiler jusqu'à trouver une zone qui vous semble vide.

Le code de ce *trackloader* semble se finir vers **$7CA60**
Vous pouvez valider le tout avec la commande **:** `M 7CA60-12` puis `D 7CA60-12`

```
m 7CA60-12
:07CA4E 00 00 00 00 00 00 00 02 FF 38 00 C8 FE D4 01 AE .........8......
:07CA5E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:07CA6E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:07CA7E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:07CA8E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:07CA9E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:07CAAE 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:07CABE 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................


d 7CA60-12
~07CA4E ORI.B   #0,D0
~07CA52 ORI.B   #2,D0
~07CA56 LINEF
~07CA58 ORI.?   #FED401AE,A0
~07CA5E ORI.B   #0,D0
~07CA62 ORI.B   #0,D0
~07CA66 ORI.B   #0,D0
~07CA6A ORI.B   #0,D0
~07CA6E ORI.B   #0,D0
~07CA72 ORI.B   #0,D0
~07CA76 ORI.B   #0,D0
~07CA7A ORI.B   #0,D0
~07CA7E ORI.B   #0,D0
```

Part 7 Rip - Disk 1

On va taper un bout de code pour aider au rip.

Taper : `A 7CA60`

```
d 7CA60
~07CA60 MOVE.W  #F00,00DFF180     ; Fond d'écran en rouge, on note A0, D0 et D1
~07CA68 BTST    #6,00BFE001       ; On test l'appuie sur la souris
~07CA70 BNE     0007CA60          ; Tant que ce n'est pas bon, on boucle
~07CA72 BSR     0007C41C          ; C'est appuyé, GoTo #Trackloader
~07CA76 BSR     0007CA8E          ; GoTo #Petite pause
~07CA78 MOVE.W  #F0,00DFF180      ; Fond d'écran en Vert, on peut sauver
~07CA80 BTST    #6,00BFE001       ; On test l'appuie sur la souris
~07CA88 BNE     0007CA78          ; Tant que ce n'est pas bon, on boucle
~07CA8A BSR     0007CA8E          ; GoTo #PetitePause
~07CA8C RTS                       ; Retourne au code original
;=====================================
~07CA8E MOVE.W  #FFF,0007CAAA     ; Notre compteur de $FFF en 7CAAA
~07CA96 SUBI.W  #1,0007CAAA       ; On enleve 1 à ce compteur
~07CA9E CMPI.W  #0,0007CAAA       ; On compare notre compteur avec 0
~07CAA6 BNE     0007CA96          ; Tant que ce n'est pas le cas, on boucle
~07CAA8 RTS                       ; Routine PetitePause Fini, retour au code
;=====================================
~07CAAA ORI.B   #0,D0
```

* *Bit 6 de BFE001 = Bouton Fire port1 = Bouton gauche de la souris*

On en profite pour sauver ce bout de code sur notre disquette préalablement crée.
Insérer notre disquette *bloodmoneyD1* dans le lecteur de disquette.

Taper : `SM code_rip, 7CA60 7CAAA` **et réinsérer la disquette 1 original du jeu original dans le lecteur.**

Et maintenant, il faut changer tous les appels vers *$7C41C* en *$7CA60*
7A458, 7A4D8, 7A4EE, 7A4FC, 7A600, 7A60E, 7A70C, 7A71A

Taper : `A 7A458`

```
^7A458  bsr 7CA60               ; etc...pour tous les appels
^7A45C
```

```
a 7A458
^07A458 BSR     7CA60
^07A45C
a 7A4D8
^07A4D8 BSR     7CA60
^07A4DC
a 7A4EE
^07A4EE BSR     7CA60
^07A4F2
a 7A4FC
^07A4FC BSR     7CA60
^07A500
a 7A600
^07A600 BSR     7CA60
^07A604
a 7A60E
^07A60E BSR     7CA60
^07A612
a 7A70C
^07A70C BSR     7CA60
^07A710
a 7A71A
^07A71A BSR     7CA60
^07A71E
```

Je vous conseille aussi de créer un tableau avec les différentes valeurs lors des appels.
Exemple pour l'appel en *$7A458*

```
d 7A458-12
~07A446 BSET    #2,00BFD100
~07A44E MOVEA.L #58E44,A0
~07A454 MOVEQ   #1E,D0
~07A456 MOVEQ   #14,D1
```

Est Appelé par	Head	Move	A0	D0	D1	Chargement de	Extra Info
TrkLoad(1) $7A446	DOWN	30->49	58E44	1E	14	Screenshot	Affichage image du jeu

On n'oublie pas de rétablir le code original en *$7DC4C*
Taper : `A 7DC4C`
^07DC4C JMP 7A120
^07DC52

Et on retourne au code avec la commande **X**
Après un rapide accès disque (retour en T00), notre joli écran rouge apparait. On **rentre dans l'AR** et l'on **tape** : **R** (noter **A0**, **D0** et **D1**)

```
r
D0=0000001E 00000014 00000001 0000527F  0000527F 0000001D FFFFFFFF 0000FFFF
A0=00058E44 0007D190 FFFFFFFF 00FE86EE  00001558 0007DBB8 00DFF000 0007FFFC
PC = 0007CA68 USP = 00C014B2 SR = 2200 T=0 S=1 I=010 X=0 N=0 Z=0 V=0 C=0
```

On retourne au code, taper : X et on appuie sur le bouton de la souris
Après un chargement de plusieurs *tracks*, notre écran passe au **vert**, il est temps de **retourner dans l'AR** et de **sauver** les DATA
D0 = $1E = **!30**
D1 = $14 = **!20**

Nous avons vue qu'une *track* custom fait ici *$1838* et nous avons chargé 20 tracks
Donc !20*$1838=**$1E460** de donnée chargée

Insérer notre disquette *bloodmoneyD1* dans le lecteur de disquette et sauvons tout ça.
Taper : `SM 30_49, 58E44 58E44+1E460`

(*) Piste de fin 49 car nous avons une longueur de D1 à !20 et que l'on part de la Piste 30 (inclus)

On remet notre disquette 1 original dans le lecteur de disquette et l'on revient au code avec la commande : X
On appuie sur le bouton de la souris et ainsi de suite.

Noter que si vous avez un second lecteur de disquette (ce qui est fortement recommandé), vous pouvez insérer la disquette ***bloodmoneyD1*** dans celui-ci et sauver directement dessus depuis l'**AR** cela vous évitera le swap de disquette.
Exemple : `SM 1:30_49, 58E44 58E44+1E460`

Continuer cette manipulation jusqu'au dernier appel, à savoir le message **Insert Disk 2**
Cela devrait vous donner le tableau suivant :

Est Appelé par	Head	Move	A0	D0	D1	Chargement de	Extra Info
Trackdisk.device	**DOWN**	0	N/A	N/A	N/A	Bootsecteur	Copie de 1200 byte (trackdisk.device) depuis l'adresse disque $400 vers $7DBB8
$7DC36 + TrkLoad(1)	**DOWN**	28 ➔ 29	$7A120	$1C	$02	Trackloader(1)	Copie du trackloader principal en mémoire. Contiens tous les appels au TrkLoad du D1
$7A446 + TrkLoad(1)	**DOWN**	30 ➔ 49	$58E44	$1E	$14	Screenshot	Affichage image du jeu
$7A4C6 + TrkLoad(1)	**DOWN**	01 ➔ 13	$8A48	$01	$0D	N/A	Retour en piste 00 à la fin de l'anim. Check T00 ?
$7A4DC + TrkLoad(1)	**UP**	01 ➔ 28	$2E7C0	$01	$1C	N/A	
$7A4F2 + TrkLoad(1)	**UP**	29 ➔ 40	$1C520	$1D	$0C	Psysnosis Animation	
$7A5F6 + TrkLoad(1)	**UP**	41 ➔ 61	$400	$29	$15	N/A	
$7A604 + TrkLoad(1)	**UP**	62 ➔ 69	$1FFD0	$3E	$08	Chip Animation	
$7A6FA + TrkLoad(1)	**DOWN**	14 ➔ 22	$400	$0E	$09	N/A	
$7A710 + TrkLoad(1)	**DOWN**	23 ➔ 27	$14E72	$17	$05	Insert Disk2 Message	Retour en piste 00,Check T00 ?

Extra info :

TrkLoad(1) = *$7C41C*
BSET #2, BFD100 = **Side lower**
BCLR #2, BFD100 = **Side upper**
TrackSize Custom = *$1838* *= !6200*

Move	A0	D1	LENGTH	COMMAND
28 ➔ 29	$7A120	$02	$02 * $1838 = $3070	`SM 28_29, 7A120 7A120+3070`
30 ➔ 49	$58E44	$14	$14 * $1838 = $1E460	`SM 30_49, 58E44 58E44+1E460`
01 ➔ 13	$8A48	$0D	$0D * $1838 = $13AD8	`SM 01_13, 8A48 8A48+13AD8`
01 ➔ 28	$2E7C0	$1C	$1C * $1838 = $2A620	`SM 01_28, 2E7C0 2E7C0+2A620`
29 ➔ 40	$1C520	$0C	$0C * $1838 = $122A0	`SM 29_40, 1C520 1C520+122A0`
41 ➔ 61	$400	$15	$15 * $1838 = $1FC98	`SM 41_61, 400 400+1FC98`
62 ➔ 69	$1FFD0	$08	$08 * $1838 = $C1C0	`SM 62_69, 1FFD0 1FFD0+C1C0`
14 ➔ 22	$400	$09	$09 * $1838 = $D9F8	`SM 14_22, 400 400+D9F8`
23 ➔ 27	$14E72	$05	$05 * $1838 = $7918	`SM 23_27, 14E72 14E72+7918`

Je vous invite lors d'un rip à tester par vous-mêmes la 'zone mémoire de destination'.
Il suffit pour cela, avant l'appel à la routine de *trackload* (donc sur le fond d'écran rouge), de remplir sous l'AR la zone mémoire de destination.
De changer la destination mémoire (**A0**) et d'utiliser la commande `O` associer à un pattern spécifique (peu importe) pour remplir cette zone

Exemple sur le second appel disque, chargement **30** à **49**, Zone mémoire de destination normal en **$58E44**
Entrer dans l'AR sur **notre écran rouge, AVANT** cet l'appel au *trackload* puis,
taper : `O "PaTtErN", 20000 20000+1E460+100` sans oublier `R A0 20000`
Retourner au code avec la commande `X`, appuyer sur la souris **et sur l'écran vert entrer dans l'AR**

Vous verrez que, pour ce test :

- Il ne faut pas trop déborder lors du remplissage de la zone mémoire, sinon le *trackloader* va crasher.
- Qu'une fois les datas chargées, est ajouté en fin de donnée une sorte de CRC de celui-ci, ce 'CRC' varie selon plusieurs facteurs comme la position mémoire. (Donc **A0**)
  Qu'il est 'ajouté' à la fin des données chargées, comprendre qu'il est calculé, il ne fait pas partie des données lue sur le disque.
  Que ça taille varie (1 word, 1 long word, des fois un peu plus, un peu moins)
- Que finalement, on modifiant la zone mémoire de destination, vous verrez qu'il lie exactement le nombre de data prévue (D1*$1838)

Bref, qu'en clair, on peut utiliser directement **D1** comme base de calcul associer à la taille d'une track (*$1838*) pour calculer exactement la longueur de donnée lue/chargée depuis le disque. (et donc à sauver).

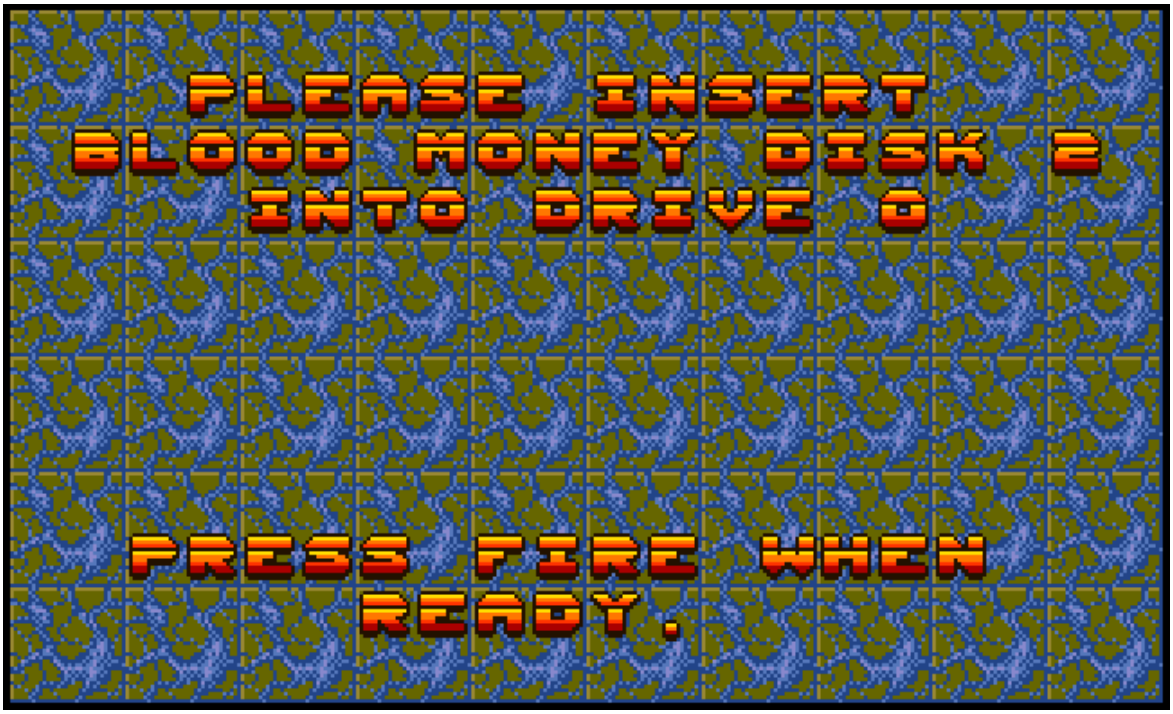
Chose que vous venez normalement de réaliser ci-dessus (voir tableau et commandes juste au-dessus)

Qu'assez logiquement, au vue de ces résultats, les données ne sont pas compressées.
La taille de la zone mémoire remplie après lecture et la même que la taille attendue de lecture.
Aucune routine de décompression après la routine de *trackload*, on a directement un **jmp** en général.
Donc pas de routine de compression dans le trackloader ni d'appel vers une routine de décompression ensuite.

Finalement, vous devriez avoir ripé les fichiers suivants :

```
dir
Directory of (bloodmoneyD1)
      074400  29_40
      012400  28_29
      130200  41_61
      049600  62_69
      055800  14_22
      124000  30_49
      080600  01_13
      000074  code_rip
      173600  01_28
      031000  23_27
0227 blocks free, 87.1 % of disk used
Disk ok
```

Une fois donc tous ces fichiers ripés, vous deviez donc vous retrouver logiquement sur l'écran où **il vous demande d'insérez le Disk2**



Assez logiquement, on peut déjà prévoir que le prochain appel au *trackloader* ne fera PAS appel à celui en **7C41C**
Pourquoi ? Tout simplement car nous avons ripé TOUS les appels que l'on a trouvés vers celui-ci.

D'ailleurs vous pouvez le vérifier par vous-même.
Insérez le disk **BLOOD MONEY DISK 2** dans le lecteur comme demandé et appuyé sur **FIRE**
Vous voyez ? Il a fait quelques chargements et nous sommes maintenant sur l'écran du menu principal.

Et aucun fond d'écran rouge ou vert, il n'est pas passé par le code du *trackloader(1)* que l'on à vue.
Il est donc assez logique de penser qu'il a utilisé un autre *trackloader* quelque part en mémoire.

Nous voilà donc bon pour le disque1. Regardons les positions de ce que l'on a ripé.

LOWER	UPPER
01 ➔ 13 14 ➔ 22 23 ➔ 27 28 ➔ 29 30 ➔ 49	01 ➔ 28 29 ➔ 40 41 ➔ 61 62 ➔ 69

Il semblerait donc que les données finissent en **fin de piste 49** en **Lower** et en **fin de piste 69** en **Upper**
On peut vérifier s'il y a des données sur ces pistes.

Rebooter votre Amiga en réinsérant notre disquette 1 de backup dans le lecteur
Sur l'écran rouge, retirer la disquette 1 de backup dans le lecteur pour la remplacer par la disquette 1 original.
Entrer dans l'AR et taper : A 7DC36

```
^07DC36         BSET    #2,BFD100               ; Side Lower
^07DC3E         MOVEA.L #20000,A0               ; Destination mémoire = $20000
^07DC44         MOVEQ   #32,D0                  ; Piste !50
^07DC46         MOVEQ   #1E,D1                  ; Longueur !30
^07DC48         <RETURN>
```

Taper : A 7DC4C

```
^07DC4C         JMP     7DC4C                   ; On boucle sur nous-même (dead loop)
```

Puis, retourner au code, **taper X**
Une fois le chargement terminé et que l'on a donc plus aucun d'accès sur le lecteur de disquette, **entrer dans l'AR**
Taper N 20000 et faite défilé jusqu'à **$4E000** ($1838*$1E=$2D690, $2D690+$20000=$4D690)
Vous pouvez voir qu'il n'y a aucune donnée. ($00 partout dans la zone mémoire **A0**)

ON vérifie l'autre côté de la disquette, le coté *UPPER*
Rebooter votre Amiga en réinsérant notre disquette 1 de backup dans le lecteur
Sur l'écran rouge, retirer la disquette 1 de backup dans le lecteur pour la remplacer par la disquette 1 original.
Entrer dans l'AR et taper : A 7DC36

```
^07DC36         BCLR    #2,BFD100               ; Side Upper
^07DC3E         MOVEA.L #20000,A0               ; Destination mémoire = $20000
^07DC44         MOVEQ   #46,D0                  ; Piste !70
^07DC46         MOVEQ   #A,D1                   ; Longueur !10
^07DC48         <RETURN>
```

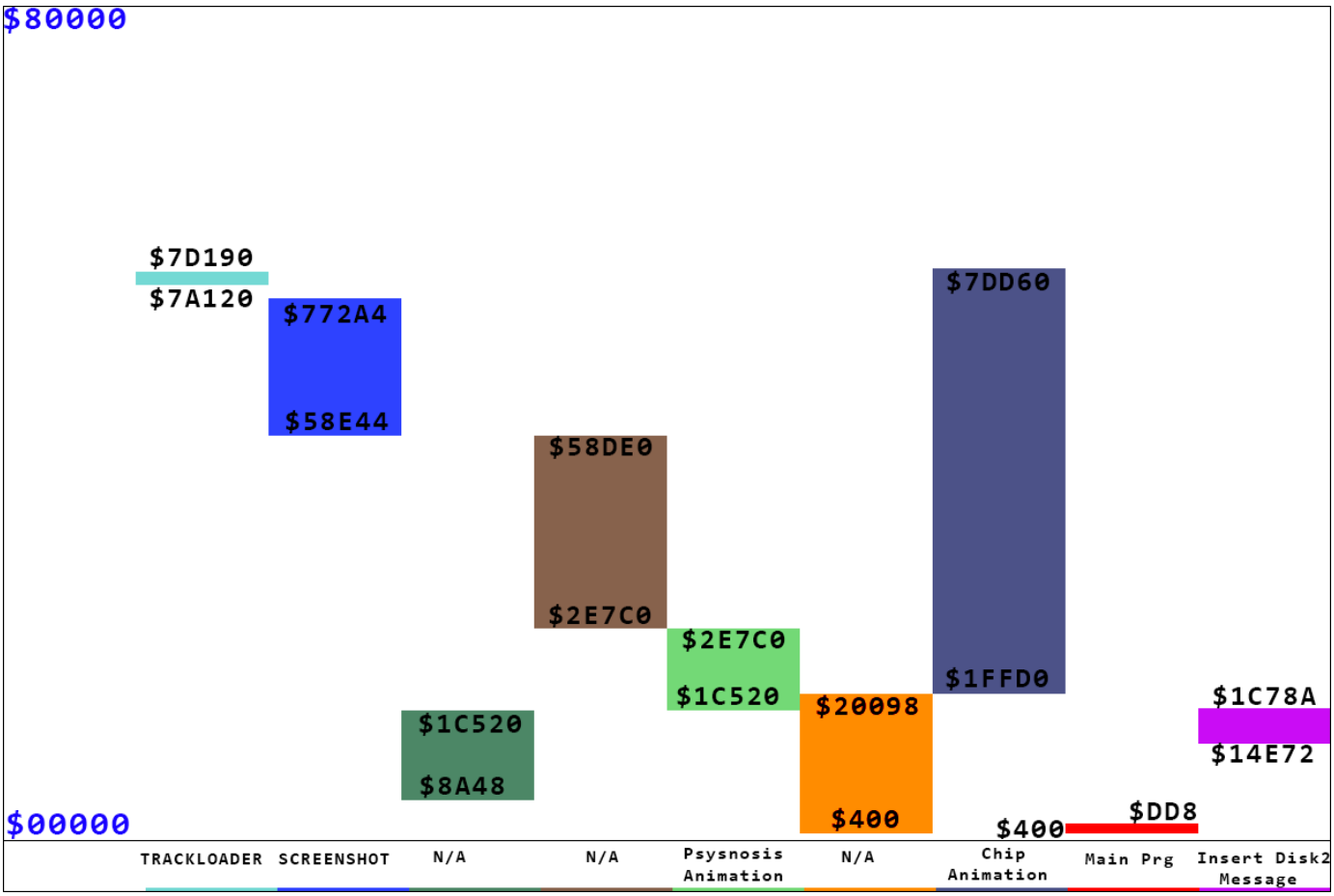
Taper : A 7DC4C

```
^07DC4C         JMP     7DC4C                   ; On boucle sur nous-même (dead loop)
```

On retourne au code, **taper X**
Une fois le chargement terminé et que l'on a donc plus aucun d'accès sur le lecteur de disquette, **entrer dans l'AR**
Taper N 20000 et faite défilé jusqu'à **$2FF00** ($1838*$A=$F230, $20000+$F230=$2F230)
Vous pouvez voir qu'il n'y a aucune donnée. ($00 partout dans la zone mémoire **A0**)

Le disque 1 est donc complétement ripé (hormis le *bootsecteur* mais qui ne nous servira pas dans pour notre version hacké)

Finalement, cela nous donne le tableau suivant pour l'utilisation des plages mémoire, dans l'ordre de chargement.



Part 8 Disk 2 – Nouveau Trackloader(2)

Rebooter donc votre Amiga avec le Disque1 de l'original dans le lecteur.
Aller jusqu'à l'écran **Insérez** le disk **BLOOD MONEY DISK 2** puis **entrer dans l'AR**

Taper : `D`

```
d
~002D64 MOVE.B  00BFE001,D0



;======================================
~002D64 MOVE.B  00BFE001,D0
~002D6A NOT.B   D0
~002D6C ANDI.B  #C0,D0
~002D70 BEQ     00002D64
~002D72 RTS
;======================================
```

On se trouve en **$2D64** qui attends tout simplement que l'on appuie sur la souris pour faire un **RTS**
On regarde où pointe **A7** (la pile), **taper** `R`
On voie la valeur **$3F8** dans **A7**, **taper** `M 3F8`
Ce qui nous donnes l'adresse mémoire de retour à savoir : **$4326**
Taper `D 4326-4` (et descendez un peu dans le code)

```
r
D0=00072500 0000FFFF 00000000 0000BE01  00000000 0000FFFE 00000000 0000FFFF
A0=000076BA 00008452 0006B74E 00004650  0006354E 00008C98 00DFF000 000003F8
PC = 00002D64 USP = 000018B2 SR = 2104 T=0 S=1 I=001 X=0 N=0 Z=1 V=0 C=0

m 3F8
:0003F8 00 00 43 26 00 00 04 C8 41 F9 00 00 04 0C 21 C8 ..C&....A.....!.

d 4326-4
~004322 BSR     00002D64
~004326 MOVEA.L #5AE3E,A0
~00432C MOVEQ   #E,D0
~00432E MOVEQ   #1,D1
~004330 BSET    #2,00BFD100
~004338 BSR     00004066
~00433C CMPI.L  #32323232,0005AE3E
~004346 BNE     00004322
~004348 RTS
;======================================
~00434A LEA     0003D090,A0
```

```
4322    BSR     2D64            ; On se retrouve avec le code ci-dessous qui ressemble à ce que l'on
                                ; a déjà vue à savoir, positionnement de la side avec bset ou bclr sur le CIA
                                ; positionnement de A0, D0 et D1 suivi d'un BSR
                                ;
                                ; Regarder plutôt ➔

4326    MOVEA.L #5AE3E,A0       ; Adresse mémoire de destination = 5AE3E
432C    MOVEQ   #E,D0           ; Piste de départ = $E = !14
432E    MOVEQ   #1,D1           ; longueur, 1 track
4330    BSET    #2,BFD100       ; On se positionne sur la face 'lower'
4338    BSR     4066            ; Adresse du nouveau Trackloader utilisé sur le second disque.

En 4066 and co ➔                ; Moteur On, Sélection Drive DF0, Timeout de 64ms + timeout plus long de $BB8
                                ; Met $4489 à l'adresse (A6)+7E
                                ; On enlève 1 à D1
                                ; On sauve la pile, bla bla, semble bien une routine de trakloader

433C    CMPI.L  #32323232,5AE3E ; On compare ce qu'il y a en 5AE3E avec la sequence 32323232
                                ; En ascii cela nous donnes : 2222
                                ; Bon clairement, il semblerait que l'on vient de faire appel à une routine
                                ; de trackload qui a vérifié une signature

4346    BNE     4322            ; Si ce n'est pas bon,
                                ; on boucle jusqu'à la lecture et validation de la signature

4348    RTS                     ; Retourne à la suite du code
```

On va poser un *breakpoint* en **$4066** et continuer le code original
#BS, alias BreakPoint. Permet, dès que l'adresse mémoire indiquée est atteinte, d'effectuer un arrêt du code.
Taper : `BS 4066` puis `X` et appuyer bien sûr, sur FIRE.

Notre *breakpoint* en **$4066** est de suite atteint, on entre automatiquement dans l'AR
Taper : `R` et regarder les valeurs de **D0**, **D1** et **A0**

```
Breakpoint raised at address: 00004066

r
D0=0000000E 00000001 00000000 0000BE01  00000000 0000FFFE 00000000 0000FFFF
A0=0005AE3E 00008452 0006B74E 00004650  0006354E 00008C98 00DFF000 000003F8
PC = 00004066 USP = 000018B2 SR = 2104 T=0 S=1 I=001 X=0 N=0 Z=1 V=0 C=0
```

Tout ça ressemble fortement à ce que l'on a déjà vue lors du fonctionnement du 1er *trackloader*

On regarde où pointe **A7** (la pile), **A7** a la valeur **$3F8**
Taper `M 3F8`

```
M 3F8
:0003F8 00 00 43 3C 00 00 04 C8 41 F9 00 00 04 0C 21 C8 ..C<....A.....!.
```

Ce qui nous donnes l'adresse mémoire de retour à savoir : **$433C**
Taper `D 433C-12`

```
d 433C-12
~00432A LINEA
~00432C MOVEQ   #E,D0
~00432E MOVEQ   #1,D1
~004330 BSET    #2,00BFD100
~004338 BSR     00004066
```

On retrouve le même type d'appel que sur le 1er *trackloader*
Positionnement de **D0**, **D1**, **A0** et la **side** puis appel du *trackloader*

On va chercher en mémoire tous les appels vers cette nouvelle adresse.
#FA, alias Find for Addressing Upcode, On va chercher une instruction qui fait appel à une adresse mémoire.

Taper : `FA 4066`

```
fa 4066
Search from: 000000 to: 080000
0004EE BSR      00004066
000CBA BSR      00004066
000CF4 BSR      00004066
000D48 BSR      00004066
000D88 BSR      00004066
004338 BSR      00004066
004BB4 BSR      00004066
```

On a du monde…Regardons ça de plus prêt.
Taper : `D 4EE-22`

```
d 4EE-22
~0004CC BSR     00004292
~0004D0 BSR     00003F70
~0004D4 BSET    #2,00BFD100
~0004DC MOVE.L  #732,0000006C.S
~0004E4 MOVEA.L #D55A,A0
~0004EA MOVEQ   #8,D0
~0004EC MOVEQ   #5,D1
~0004EE BSR     00004066
```

Et le suivant, **taper :** `D CBA-22`

```
d CBA-22
~000C98 CMPI.W  #2,00032604
~000CA0 BEQ     00002E42
~000CA4 MOVEQ   #14,D0
~000CA6 MOVEQ   #1E,D1
~000CA8 BSET    #2,00BFD100
~000CB0 BRA     00000CB4
;=======================================
~000CB4 MOVEA.L #3D090,A0
~000CBA BSR     00004066
```

Etrange ce **CMP** juste au-dessus non ?
Remonter encore un peu dans le code, on voie ceci :

On regarde si d'autre portion de code en mémoire ont cette tête et fond donc appel à l'adresse mémoire **$CB4**
Taper : `FA CB4`

Non, on trouve uniquement nos adresses en question.
Nous avons donc ici une **4 mini routines** qui branchent à chaque fois, après avoir défini **D0** et **D1** ainsi que la Side, en **$CB4**
Qui défini **TOUT LE TEMPS A0** à *$3D090* et appel le *trackloader(2)*

Combien de niveau peut-on trouver dans ce jeu déjà ? ah oui, 4. Il est fort probable que ces mini-routines appel le Niveau choisi.

Utiliser donc la commande `D` sur chacune des adresses trouvées (en prenant un petit delta juste avant pour être sûr de ne rien louper)
Et construisez un petit tableau de tous ces appels.

Extra info :

TrkLoad(2) = *$4066*

Est Appelé par	Head	Move	A0	D0	D1
$4D4 + TrkLoad(2)	DOWN	08 → 12	D55A	08	05
$C50 + $CB4 + TrkLoad(2)	UP	01 → 30	3D090	01	1E
$C6C + $CB4 + TrkLoad(2)	UP	36 → 60	3D090	1F	1E
$C88 + $CB4 + TrkLoad(2)	DOWN	50 → 79	3D090	32	1E
$CA4 + $CB4 + TrkLoad(2)	DOWN	20 → 49	3D090	14	1E
$CDA + TrkLoad(2)	DOWN	08 → 12	D55A	08	05
$D36 + TrkLoad(2)	DOWN	05 → 08	D55A	05	03
$D76 + TrkLoad(2)	DOWN	01 → 04	7541E	01	04
$4326 + TrkLoad(2)	DOWN	14 → 15	5AE3E	0E	01
$4BA2 + TrkLoad(2)	UP	61 → 73	1B312	3D	D1

Noter (fond en marron dans le tableau), que 2 appels sont identiques (**A0**, **D0** et **D1** identique) mais situé à deux adresses mémoire différentes.

Part 9 Cheats

Démarrer une partie, on va commencer par la recherche de **vie infinie**

Comme vous pouvez le remarquer, on démarre une partie avec 3 Vies
Taper : `TS 3` puis `X`

De retour au jeu, on perd une vie et on **entre dans l'AR**
Taper : `T 2`

L'AR nous retourne une seule adresse possible : *8C29*
#TFD, alias TrainerFindDecrement. Cherche une commande de décrémentation en mémoire par rapport à une adresse donnée
La commande `TFD 8C29` ou `TFD 8C28` ne rapporte rien…

On va procéder autrement, **taper :** `MS 8C29` puis `X`
De retour au code, perdez une vie.
Notre *MemWatch Point* est atteint (modification de la zone mémoire *$8C29*), on entre automatiquement dans l'AR

Taper : `D` (on se retrouve en 682)
Taper : `D 682-4`

```
00067E SUBQ.W #1,2(A4)                  ; Un SUB bien caché
```

```
A 67E
^0067E  TST.W 2(A4)
^00682  <RETURN>
```

On continue avec la recherche **d'invulnérabilité**

On commence donc avec 3 vies et pour chaque vie nous avons 6 barres d'Energie (**visible à l'écran en bas orange et rouge**)

On va donc utiliser la fonction TS et T de l'AR
#TS, alias StartTrainer. On commencer une recherche de trainer.
#T, alias Trainer. Permet de continuer la recherche d'un trainer
Entrer dans l'AR et taper : `TS 6`

On retourne ensuite au jeu et on perd une barre d'Energie (ou plus, vous adapter la recherche en conséquence bien sûr)
Entrer dans l'AR et taper : `TS 5`

On tombe sur une seule adresse possible : **$8C47**

Utiliser la commande `M 8C47` et faite quelques tests (retour au jeu, perte d'Energie, retour AR, utilisation commande **M 8C47**)
Effectivement, cela correspond bien au nombre de barre d'Energie.

#MS, alias Set MemWatch Point. Permet dès que l'adresse mémoire est modifié, d'effectuer un arrêt du code.
On va poser un *MatchWatch*, **Entrer dans l'AR et taper :** `MS 8C47` et **retourner** au jeu avec la commande `X`
Effectuer un crash du vaisseau sur le décor, **on entre automatiquement dans l'AR**
On regarde où l'on se trouve, **taper :** `D` nous sommes en *$2FBC*
On regarde le code juste au-dessus, **taper :** `D 2FBC-4`

```
2FB8 SUBQ.W #1,20(A4)                   ; Bingo, même type de code que précédemment
```

Si vous taper `R` vous pouvez voir que **A4** à la valeur **8C26**
On prend le **SUB** au-dessus et on calcul $8C26+$20=**$8C46**
Et comme le **SUB** est ici en **WORD**, cela nous donne bien *$8C46* et *$8C47* comme adresse mémoire

On va changer ça en un simple **TST.W**
Taper : `A 2FB8`

```
^002FB8 TST.W 20(A4)
^002FBC <RETURN>
```

Et on retourne au code avec la commande `X`
OK, cela fonctionne bien, on est invulnérable au contact des décors **mais** on perd toujours une barre d'Energie si une ennemie nous touche…

Le **MS** étant toujours d'actu, dès que l'on touche un ennemi, on entre toujours automatiquement dans l'AR.
On fait donc la même chose qu'avant : **taper :** `D` nous sommes en *$14D0*
On regarde juste au-dessus, **taper :** `D 14D0C-4`

```
14CC SUBQ.W #2,20(A4)                   ; Bingo, again...
```

Comme au préalable, on change ce SUBQ.W en TST.W, **Taper :** `A 14CC`

```
^0014CC TST.W 20(A4)
^0014D0 <RETURN>
```

Mais avant de repartir au jeu, on va chercher si on n'a pas encore d'autre **SUB** caché.
En **OPCODE** le **SUBQ.W #2,20(A4)** nous donnes : `55 6C 00 20`
`00 20` étant la valeur à ajouter en l'occurrence le **20** du 20(A4)
et le `6C` étant l'**OPCODE** du registre **A4,** le `55` étant l'**OPCODE** de la commande `SUBQ.W`

Sur le 1er SUB, on enlevait 1 et sur celui d'au-dessus 2, on ne peut donc pas chercher directement l'**UPCODE** complet `55 6C 00 20`
On va donc chercher uniquement ce qui fait appel à **20(A4)**, à savoir en **OPCODE** `6C 00 20`
Taper : `F 6C 00 20`

On trouve 3 adresses : ***14CD*** *152D et* ***2FB9*** ***(****14CD* et *2FB9* sont les deux que l'on a déjà fait.)
On va donc regarde le dernier, **taper :** `D 152D-1` (-1 car il manque volontairement dans notre recherche le 1er opcode)
`^00152C SUBQ.W #2,20(A4)` *; et encore un*

Même punition, on le remplace comme au préalable, **taper :**
`A 152C`
`^0152C TST.W 20(A4)`
`^01530` **<RETURN>**

On retourne au jeu, sans oublier d'enlever le **MS**
#MDA, alias MemWatch Point Delete. Supprime tous les MemWatch point.
Taper : `MDA` puis `X`

Nous sommes devenues complétement invulnérable, que ce soit au tir ennemi, au contact ennemi et au décors.

C'est maintenant au tour du **Nombre de dollar**

Toujours en pleine partie, on s'arrange pour avoir (par exemple) 100 $ de money, **on entre dans l'AR**
(Bien sûr, adapter le tuto par rapport à la valeur que vous avez de dollar)

100 en décimal nous donnes **$64** en hexa, on va donc commencer notre recherche avec cette valeur
Taper : `TS 64` puis `X`

Détruisez un ennemi et prenez une pièce , dans notre exemple nous sommes maintenant à **125 $** ce qui donnes en **hexa $7D**
Taper : `T 7D`
L'AR nous donne un seul résultat : **$8C2F**

Visuellement, le compteur de monnaie semble pouvoir aller jusqu'à 999 en décimal donc **03E7** en Hexa
C'est la longueur d'un **WORD**, il faut donc reculer de 1 sur l'adresse mémoire, on se trouve donc en **$8C2E**
Taper : `M 8C2E` et changer la valeur par **03 E7**, valider le tout
On retourne au jeu, taper : `X`

Le jeu affiche une valeur de **999 $** de monnaie, Cool ☺

Et pour finir, nous cherchons le **Nombre de point**

Encore et toujours en partie, nous **noter le score** que vous avez et vous **entrer dans l'AR** (pour notre exemple, on part sur un score de 500)

500 en décimal cela nous donne **01F4** en Hexa
Taper : `TS 01F4` puis `X`

Augmenter le score et chercher cette nouvelle valeur (qui est toujours en décimal) avec la commande `T`
Exemple : `T 20D` si votre score est à 525 à l'écran.*
* 525 en décimal nous donnes **$20D** en hexa

L'AR nous donne deux résultats : **$8C2D** et **$8C43**

Taper : `M 8C43` et changer la valeur par **$FF** (par exemple) puis retourner au jeu avec la commande `X`
Aucun changement visible à l'écran, mauvaise pioche, on continue.

Taper : `M 8C2D` et changer la valeur par **$FF** (par exemple) puis retourner au jeu avec la commande `X`
Le score à bien changer, nous avons trouvé la bonne adresse mémoire.

Le score max (visuellement) peut monter est jusqu'à **999999**, ce qui nous donnes en Hexa : **0F423F**

On recule de 2 sur notre adresse mémoire trouvée, ce qui nous donnes **$8C2B**
Retourner dans l'AR et taper : `M 8C2B` et changer la valeur par **0F 42 3F**, valider le tout
On retourne au jeu, taper : `X`

Le score est maintenant à 999999 ☺

Part 10 Rip - Disk 2

On reprend notre tableau vu en **part 8** et on en profite pour ajouter quelques colonnes et on trie le tout.

TrkLoad(2) = *$4066*

Est Appelé par	Head	Move	A0	D0	D1	LENGTH	**ZONE MEMOIRE UTLISEE**
$D36 + TrkLoad(2)	DOWN	05 ➔ 08	D55A	05	03	$48A8	$D55A ➔ $11E02
$4D4 + TrkLoad(2)	DOWN	08 ➔ 12	D55A	08	05	$7918	$D55A ➔ $14E72
$CDA + TrkLoad(2)	DOWN	08 ➔ 12	D55A	08	05	$7918	$D55A ➔ $14E72
$4BA2 + TrkLoad(2)	UP	61 ➔ 73	1B312	3D	0D	$13AD8	$1B312 ➔ $2EDEA
$4326 + TrkLoad(2)	DOWN	14 ➔ 15	5AE3E	0E	01	$1838	$5AE3E ➔ $5C676
$C50 + $CB4 + TrkLoad(2)	UP	01 ➔ 30	3D090	01	1E	$2D690	$3D090 ➔ $6A720
$C6C + $CB4 + TrkLoad(2)	UP	31 ➔ 60	3D090	1F	1E	$2D690	$3D090 ➔ $6A720
$C88 + $CB4 + TrkLoad(2)	DOWN	50 ➔ 79	3D090	32	1E	$2D690	$3D090 ➔ $6A720
$CA4 + $CB4 + TrkLoad(2)	DOWN	20 ➔ 49	3D090	14	1E	$2D690	$3D090 ➔ $6A720
$D76 + TrkLoad(2)	DOWN	01 ➔ 04	7541E	01	04	$60E0	$7541E ➔ $7B4FE

Concrètement, nous avons plusieurs options pour riper le disque2.
Mètre un **BS** sur l'adresse du **trackloader(2)** et attendre que le code du jeu fasse appel à lui.
Cela risque de poser plusieurs dont le principal est que suivant le code du jeu, des **BS** peuvent être tout simplement ignorer par l'**AR.**
On peut à l'aide du tableau que l'on a relevé modifier un peu le code (ajout d'une Dead Loop en fin du *trackloader*) et appelé à la main en modifiant si nécessaire les registres le *trackloader.*

On peut aussi ajouter notre ancien bout de code : **code_rip**, le modifier en conséquence, patcher aussi tous les appels et jouer au jeu et le riper 'petit à petit'.

Histoire de changer un peu la méthode utilisée et de aussi de faire les choses une par une on va partir sur la 1er solution, à savoir passer par des *BreakPoints*.

On va faire les choses proprement.

Avant de sauver nos DATA du Disque2, il nous faut formater une nouvelle disquette.

- **ENLEVER LE DISK 2 ORIGINAL BLOODMONEY DU LECTEUR DE DISQUETTE.**
- **INSERER UNE NOUVELLE DISQUETTE, nous allons l'appeler *bloodmoneyD2***

Entrer dans l'AR et **Taper :** `FORMAT bloodmoneyD2`

Une fois le formatage terminé, **ENLEVER LE DISK *bloodmoneyD2* ET REINSERER LE DISK 1 ORIGINAL BLOODMONEY DANS DF0**

Selon votre configuration (lecteur externe ou pas), garder le disk ***bloodmoneyD2*** à coté de vous ou insérez le dans le lecteur **DF1**
Durant notre phase de RIP, je partirai du principe que vous avez un lecteur externe et que le disk ***bloodmoneyD2*** y est inséré.

Rebooter votre Amiga avec donc le disque original dans le lecteur puis aller jusqu'à l'écran **Insérez** le disk **BLOOD MONEY DISK 2**

Entrer dans l'AR et **taper :** `BS 4066` puis `X`

De retour sur l'écran du jeu, **insérez le second disque original de BloodMoney dans le lecteur DF0,**
Appuyer sur Fire comme demandé à l'écran.

Notre *BreakPoint $4066* est atteint très rapidement et on entre automatiquement dans l'AR
Taper : `R` et regarder **D0**, **D1** et **A0**

```
Breakpoint raised at address: 00004066
r
D0=0000000E 00000001 00000003 0000BE01  00000000 0000FFFE 00000000 0000FFFF
A0=0005AE3E 00008452 0006C662 00004650  00064462 00008C98 00DFF000 000003F8
PC = 00004066 USP = 000018B2 SR = 2104 T=0 S=1 I=001 X=0 N=0 Z=1 V=0 C=0
```

Profiter en pour mettre à jour notre tableau (ici avec un nouveau look)

Est Appelé par	Head	Move	A0	D0	D1	Length	Chargement de	Extra Info
$4326 + TrkLoad(2)	**DOWN**	14 ➔ 14	5AE3E	0E	01	$1838	N/A	Chargement Disque 2

A7, (la pile) est à **$3F8**, on regarde qu'elle adresse elle contient, ça sera l'adresse de retour après l'appel au *trackloader(2)*
Taper : `M 3F8`

```
M 3F8
:0003F8 00 00 43 3C 00 00 04 C8 41 F9 00 00 04 0C 21 C8 ..C<....A.....!.
```

L'adresse de retour sera donc **$433C**, on pose un *breakPoint* à cette adresse, **taper :** `BS 433C` puis `X`
Après un rapide accès au lecteur, notre *breakPoint $433C* est atteint, on entre automatiquement dans l'AR
On en profite pour regarder ce qu'il a été chargé en **$5AE3E** d'une taille d'une track... (semble vraiment très court pour du code)
Taper : `M 5AE3E` et descendre jusqu'à 5AE3E+1838 (à savoir : **$5C676**)

Alors c'est simple, C'est vide cde chez vide (que des Zero), SAUF, au début de cette adresse, un **LongWord** d'une valeur : **32323232**
Ce qui en ascii nous donnes : **2222**
Ce qui valide le code que l'on a vue plus haut :

```
433C    CMPI.L  #32323232,5AE3E         ; On recherche donc juste la signature du disque2
```

On sauve bien sur le tout, **taper :** `SM 1:14_14, 5AE3E 5AE3E+1838`

Il peut arriver, selon l'interaction **carte ACTION-REPLAY/Code du jeu** qu'une fois un *BreakPoint* atteint, il devienne 'transparent'
En clair, si le code du jeu fait encore appel au **trackloader(2)** en *$4066* que l'**AR** ne le voie pas.

Pour cela il existe une solution simple et efficace à 100%, on efface tous les *BreakPoints* et on les repose.
#BDA, alias AllBreakPoint Delete. Permet de supprimer tous les breakpoints
Taper : `BDA` puis `BS 4066` et bien sûr `X`
Assez rapidement, notre *breakPoint* est atteint et on entre automatiquement dans l'AR.

Taper : `R` et regarder **D0**, **D1** et **A0** sans oublier **A7** et l'adresse de retour comme effectué précédemment.

```
Breakpoint raised at address: 00004066
r
D0=00000008 00000005 00000003 0000C3DE  0000C3DE 0000000D 0000FFFF 0000FFFF
A0=0000D55A 00008290 0006C662 00004650  00064462 00008C98 00DFF000 000003FC
PC = 00004066 USP = 000018B2 SR = 2100 T=0 S=1 I=001 X=0 N=0 Z=0 V=0 C=0

M 3FC
:0003FC 00 00 04 F2 41 F9 00 00 04 0C 21 C8 00 84 4E 41 ....A.....!...NA
```

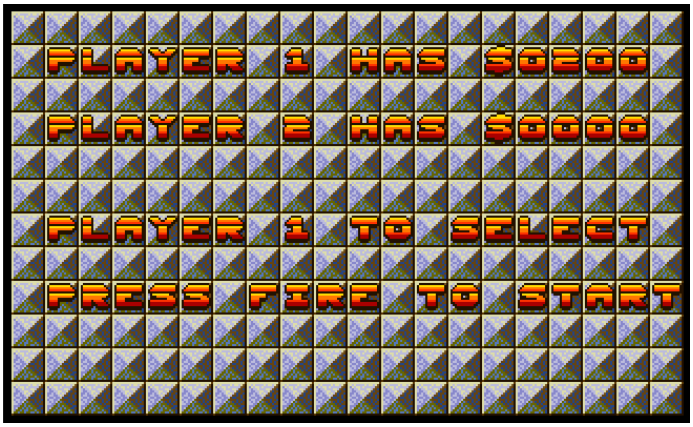
On pose donc un *breakPoint* à cette adresse, **taper :** `BS 4F2` puis `X`
Après un rapide accès au lecteur, notre *breakPoint $433C* est atteint, on entre automatiquement dans l'AR.
Taper : `SM 1:08_12, D55A D55A+7918`

Et bien sûr, **taper maintenant :** `BDA` puis `BS 4066` et bien sûr `X`

Sans oublier pour mettre à jour notre tableau.

Est Appelé par	Head	Move	A0	D0	D1	Length	Chargement de	Extra Info
$4326 + TrkLoad(2)	**DOWN**	14 ➔ 14	5AE3E	0E	01	$1838	Signature Disque 2	Check Disk2
$4D4 + TrkLoad(2)	**DOWN**	08 ➔ 12	D55A	08	05	$7918	Menu du jeu	

Je pense que vous avez compris la méthode. On va continuer tranquillement avec celle-ci.
De retour dans le code du jeu, nous sommes sur le *MENU principal* de celui-ci, **appuyer sur F1** pour commencer une partie d'1 joueur.
Puis sur Fire comme affiché à l'écran.



Assez rapidement, notre *breakPoint $4066* est atteint et on entre automatiquement dans l'AR.

Taper : `R` et regarder **D0**, **D1** et **A0** ainsi que **A7**
Comme précédemment, on pose un *breakPoint* sur **(A7)**, en l'occurrence *$D8C*, **taper :** `BS D8C` puis `X`

Après un plus long chargement, notre *breakPoint $D8C* est atteint, on entre automatiquement dans l'AR.
Taper : `SM 1:08_12, D55A D55A+7918`

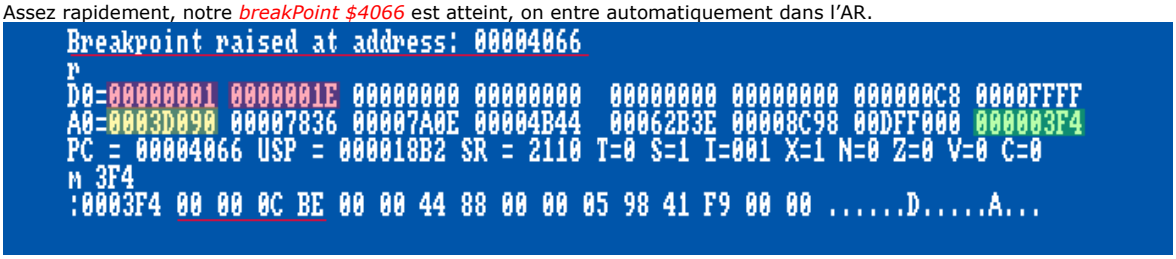
Et on n'oublie pas de mettre à jour notre tableau.

Est Appelé par	Head	Move	A0	D0	D1	Length	Chargement de	Extra Info
$4326 + TrkLoad(2)	**DOWN**	14 → 14	5AE3E	0E	01	$1838	Signature Disque 2	Check Disk2
$4D4 + TrkLoad(2)	**DOWN**	08 → 12	D55A	08	05	$7918	Menu du jeu	
$D76 + TrkLoad(2)	**DOWN**	01 → 04	7541E	01	04	$60E0	Menu choix Level	

Et c'est reparti pour un tour.
Taper : `BDA` puis `BS 4066` et bien sûr `X`

De retour sur le menu choix level on sélectionne le 1er level

Assez rapidement, notre *breakPoint $4066* est atteint, on entre automatiquement dans l'AR.



Comme précédemment, on pose un *breakPoint* sur **(A7)**, en l'occurrence *$CBE*, **taper :** `BS CBE` puis `X`

Après un plus long chargement, notre *breakPoint $CBE* est atteint, on entre automatiquement dans l'AR.
Taper : `SM 1:01_30, 3D090 3D090+2D690`

On met à jour notre tableau.

Est Appelé par	Head	Move	A0	D0	D1	Length	Chargement de	Extra Info
$4326 + TrkLoad(2)	**DOWN**	14 ➔ 14	5AE3E	0E	01	$1838	Signature Disque 2	Check Disk2
$4D4 + TrkLoad(2)	**DOWN**	08 ➔ 12	D55A	08	05	$7918	Menu du jeu	
$D76 + TrkLoad(2)	**DOWN**	01 ➔ 04	7541E	01	04	$60E0	Menu choix Level	
$C50+$CB4+TrkLoad(2)	**UP**	01 ➔ 30	3D090	01	1E	$2D690	Level1	Check T00 en fin de chargement

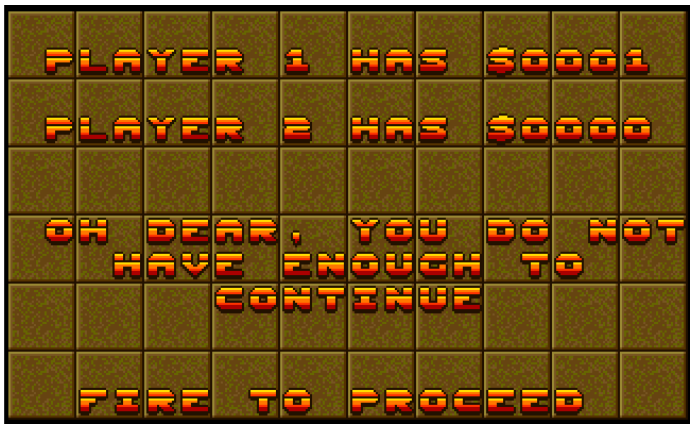
Taper maintenant : `BDA` puis `X`

Nous sommes maintenant dans le LEVEL1.
Le jeu étant particulièrement dur, nous allons utiliser l'invulnérabilité trouvé préalablement.
Entrer dans l'AR et taper : `BDA`

```
a 2FB8
^002FB8 TST.W  20(A4)
^002FBC
a 14CC
^0014CC TST.W 20(A4)
^0014D0
a 152C
^00152C TST.W  20(A4)
^001530

bs 4066
Breakpoint inserted
Ready.
```

```
A 2FB8
^02FB8  TST.W 20(A4)            ; Invulnérabilité #1
^02FBC  <RETURN>

A 14CC
^014CC  TST.W 20(A4)            ; Invulnérabilité #2
^014D0  <RETURN>

A 152C
^0152C  TST.W 20(A4)            ; Invulnérabilité #3
^01530  <RETURN>

BS 4066                         ; BreakPoint sur l'adresse du TrackLoader(2)
```

Chaque Level dure environ 15Minutes.
Vue que l'on est invulnérable, laisser le jeu tel quel pendant 15minute, ne jouer pas, n'entrer pas dans les SHOPS.
Une fois la fin de Level atteint, on est devant un boss de Fin de Niveau.



Un premier écran nous indique notre score obtenu et un second écran nous indique que nous n'avons pas assez d'argent pour continuer.
Forcément…vue que l'on a laissé le jeu tourner sans jouer… On est repartie pour faire un tour, pas cool **>_<**

Le breakpoint en *$4066* étant toujours d'actualité, on entre dans l'AR assez rapidement.
Si vous regarder les valeurs de **D0, D1 et A0** vous verrez qu'on a déjà ripé cette partie (en fait, le code revient au menu)

Resélectionner F1 pour 1 Player et choisissez le Level1 mais au finalement… on a le droit à un plantage.
(si ce n'est pas le cas chez vous, tant mieux, continuer mais il est fort probable que vous aillez aussi un crash)

Protection ? Bug ?
Quoi qu'il en soit, on notera que le jeu n'aime pas trop être bousculé on va donc faire attention dans nos commandes
et que l'on doit avoir un minimum de monnaie pour accéder au Level2 et assez logiquement aux autres Level.

Hélas, on est parti pour rebooter le jeu sur le disque original 1 puis repasser par toute les phases pour arriver jusqu'au Level1
Une fois donc de nouveau dans le level1 avec le cheat mode d'invulnérabilité d'activé et le **BS $4066**
On va donc changer la valeur de Dollar du player 1. Cheat déjà trouvé, rappelé vous.
Entrer dans l'AR et taper :

`M 8C2E` et changer les deux premières bytes par **02 FD**

```
M 8C2E
:008C2E 02 FD 00 0C 00 00 00 00 00 00 00 5E 00 50 00 52 ...........^.P.R
:008C3E 00 00 00 00 00 00 00 00 00 00 06 05 00 00 00 02 ................
```

Et pour peu qu'il y ait aussi un test sur le score, on va aussi augmenter le score. (Dans le doute…)

`M 8C2B` et changer le première bytes par **0F**

```
M 8C2B
:008C2B 0F 00 00 02 FD 00 0C 00 00 00 00 00 00 00 5E 00 ..............^.
:008C3B 50 00 52 00 00 00 00 00 00 00 00 00 00 06 05 00 P.R.............
```

Et on retourne au code, **Taper** `X`
Laisser le jeu tourner et une fois la fin de Level atteint, tuer le Boss de fin de Niveau.
Une fois le Boss tué on arrive sur l'écran des scores, on appuie sur FIRE pour continuer (2 fois, comme demandé à l'écran)

Notre *breakPoint $4066* est atteint et on entre automatiquement dans l'AR.

Taper : `R` et regarder **D0**, **D1** et **A0**
Nous sommes encore avec **D0** et **D1** a **01** et **04**, autrement dit, on va retourner au menu choix level.
Il n'est pas nécessaire de sauver les données vues qu'on l'a déjà fait pour ces valeurs **D0** et **D1.**
Taper : `X`

Bingo, comme prévu, nous sommes de retour sur le choix des niveaux avec un petit détail en plus, La planète du Level1
à maintenant un logo *PLUNDERED* sur elle.

Entrer dans l'AR et taper : `BDA` puis `BS 4066` et `X`
On choisit assez logiquement le Level2 à **$200** et on **appuie sur FIRE.**
Notre *breakPoint $4066* est alors atteint et on entre automatiquement dans l'AR.

Taper : `R` et regarder **D0**, **D1** et **A0** ainsi que **A7**

```
r
D0=0000001F 0000001E 00000000 00000000  100000FC 00000001 000002FE 0000FFFF
A0=0003D090 00007836 00007A0E 00004B44  00064462 00008C98 00DFF000 000003F4
PC = 00004066 USP = 000018B2 SR = 2110 T=0 S=1 I=001 X=1 N=0 Z=0 V=0 C=0

m 3F4
:0003F4 00 00 0C BE 00 00 44 98 00 00 05 98 41 F9 00 00 ......D.....A...
```

On pose un *breakPoint* sur **(A7)**, en l'occurrence *$CBE*, **taper :** `BS CBE` puis on retourne au code du jeu avec `X`

Après un plus long chargement, notre *breakPoint $CBE* est atteint, on entre automatiquement dans l'AR.
Taper : `SM 1:36_60, 3D090 3D090+2D690`

Comme on a vue que la configuration était assez sensible, on va enlever le Cheat sur l'invulnérabilité, on ne sait jamais.

```
a 2FB8
^002FB8 SUBQ.W #1,20(A4)
^002FBC
a 14CC
^0014CC SUBQ.W #2,20(A4)
^0014D0
a 152C
^00152C SUBQ.W #2,20(A4)
^001530
```

Taper maintenant : `BDA` puis `X`
Le Level2 apparait.

On met à jour notre tableau.

Est Appelé par	Head	Move	A0	D0	D1	Length	Chargement de	Extra Info
$4326 + TrkLoad(2)	**DOWN**	14 → 14	5AE3E	0E	01	$1838	Signature Disque2	Check Disk2
$4D4 + TrkLoad(2)	**DOWN**	08 → 12	D55A	08	05	$7918	Menu du jeu	
$D76 + TrkLoad(2)	**DOWN**	01 → 04	7541E	01	04	$60E0	Menu choix Level	
$C50+$CB4+TrkLoad(2)	**UP**	01 → 30	3D090	01	1E	$2D690	Level1	Check T00 en fin de chargement
$C50+$CB4+TrkLoad(2)	**UP**	36 → 60	3D090	1F	1E	$2D690	Level2	Check T00 en fin de chargement

De nouveau on entre dans l'AR pour remettre l'invulnérabilité et sans oublier aussi notre breakpoint sur le trackloader.

```
a 2FB8
^002FB8 TST.W  20(A4)
^002FBC
a 14CC
^0014CC TST.W 20(A4)
^0014D0
a 152C
^00152C TST.W  20(A4)
^001530

bs 4066
Breakpoint inserted
Ready.
```

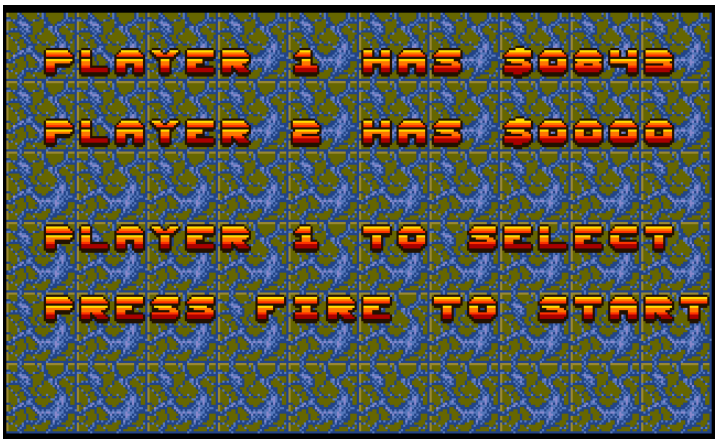
On n'oublie la monnaie.

`M 8C2E` et changer les deux premières bytes par **02 FD**

```
M 8C2E
:008C2E 02 FD 00 0C 00 00 00 00 00 00 5E 00 50 00 52 ..........^.P.R
:008C3E 00 00 00 00 00 00 00 00 00 06 05 00 00 00 02 ................
```

Et on retourne au Level2, taper : `X`

Laisser le jeu tel quel pendant 15minute, ne jouer pas, n'entrer pas dans les SHOPS.
Une fois la fin de Level atteint, **tuer le Boss de fin de Niveau** comme précédemment puis **appuyer sur Fire**.



Une fois le Boss tué on arrive sur l'écran des scores, on appuie sur FIRE pour continuer (2 fois, comme demandé à l'écran)

Notre *breakPoint $4066* est atteint et on entre automatiquement dans l'AR.

Taper : `R` et regarder **D0**, **D1** et **A0**
Nous sommes encore avec **D0** et **D1** a **01** et **04**, autrement dit, on va retourner au menu choix level.
Il n'est pas nécessaire de sauver les données vues qu'on l'a déjà fait pour ces valeurs **D0** et **D1.**
Taper : `X`

Comme prévu, nous sommes de retour sur le choix des niveaux avec maintenant le logo *PLUNDERED* sur les deux premiers level.

Entrer dans l'AR et taper : `BDA` puis **`BS 4066`** et **`X`**
On choisit assez logiquement le Level3 à **$300** et on **appuie sur FIRE.**
Notre *breakPoint $4066* est alors atteint et on entre automatiquement dans l'AR.

Taper : `R` et regarder **D0**, **D1** et **A0** ainsi que **A7**

```
Breakpoint raised at address: 00004066

r
D0=00000032 0000001E 00000000 00000000  1000F4FC 00000002 0000034B 0000FFFF
A0=0003D090 00007836 00007A0E 00004B44  0006443E 00008C98 00DFF000 000003F4
PC = 00004066 USP = 000018B2 SR = 2110 T=0 S=1 I=001 X=1 N=0 Z=0 V=0 C=0

m 3F4
:0003F4 00 00 0C BE 00 00 44 A8 00 00 05 98 41 F9 00 00 ......D.....A...
```

On pose un *breakPoint* sur **(A7)**, en l'occurrence *$CBE* (encore et toujours).
Taper : `BS CBE` puis on retourne au code du jeu avec `X`

Après un plus long chargement, notre *breakPoint $CBE* est atteint, on entre automatiquement dans l'AR.
Taper : `SM 1:50_79, 3D090 3D090+2D690`

Comme on a vu que la configuration était assez sensible, on va enlever le Cheat sur l'invulnérabilité.

```
a 2FB8
^002FB8 SUBQ.W #1,20(A4)
^002FBC
a 14CC
^0014CC SUBQ.W #2,20(A4)
^0014D0
a 152C
^00152C SUBQ.W #2,20(A4)
^001530
```

Taper maintenant : `BDA` puis `X`
Le Level3 apparait.

On met à jour notre tableau.

Est Appelé par	Head	Move	A0	D0	D1	Length	Chargement de	Extra Info
$4326 + TrkLoad(2)	**DOWN**	14 ➔ 14	5AE3E	0E	01	$1838	Signature Disque2	Check Disk2
$4D4 + TrkLoad(2)	**DOWN**	08 ➔ 12	D55A	08	05	$7918	Menu du jeu	
$D76 + TrkLoad(2)	**DOWN**	01 ➔ 04	7541E	01	04	$60E0	Menu choix Level	
$C50+$CB4+TrkLoad(2)	**UP**	01 ➔ 30	3D090	01	1E	$2D690	Level1	Check T00 en fin de chargement
$C6C+$CB4+TrkLoad(2)	**UP**	31 ➔ 60	3D090	1F	1E	$2D690	Level2	Check T00 en fin de chargement
$C88+$CB4+TrkLoad(2)	**DOWN**	50 ➔ 79	3D090	32	1E	$2D690	Level3	Check T00 en fin de chargement

De nouveau on entre dans l'AR pour remettre l'invulnérabilité et sans oublier aussi notre breakpoint sur le trackloader.

```
a 2FB8
^002FB8 TST.W  20(A4)
^002FBC
a 14CC
^0014CC TST.W 20(A4)
^0014D0
a 152C
^00152C TST.W  20(A4)
^001530

bs 4066
Breakpoint inserted
Ready.
```

On n'oublie pas de vérifier la monnaie.

`M 8C2E` et changer les deux premières bytes par **02 FD**

```
M 8C2E
:008C2E 02 FD 00 0C 00 00 00 00 00 00 5E 00 50 00 52 ..........^.P.R
:008C3E 00 00 00 00 00 00 00 00 00 06 05 00 00 00 02 ................
```

Et on retourne au Level3, taper : `X`

Laisser le jeu tel quel pendant 15minute, ne jouer pas, n'entrer pas dans les SHOPS.
Une fois la fin de Level atteint, **tuer le Boss de fin de Niveau** comme précédemment puis **appuyer sur Fire**.

Notre *breakPoint $4066* est atteint et on entre automatiquement dans l'AR.

Taper : `R` et regarder **D0**, **D1** et **A0**
Nous sommes encore avec **D0** et **D1** a **01** et **04**, autrement dit, on va retourner au menu choix level.
Il n'est toujours pas nécessaire de sauver les données vues qu'on l'a déjà fait pour ces valeurs **D0** et **D1.**
Taper : `X`

Comme prévu, nous sommes de retour sur le choix des niveaux avec maintenant le logo *PLUNDERED* sur les trois premiers level.

Entrer dans l'AR et taper : `BDA` puis **`BS 4066`** et **`X`**
On choisit assez logiquement le Level4 à **$400** et on **appuie sur FIRE.**
Notre *breakPoint $4066* est alors atteint et on entre automatiquement dans l'AR.

Taper : `R` et regarder **D0**, **D1** et **A0** ainsi que **A7**

```
Breakpoint raised at address: 00004066

r
D0=00000014 0000001E 00000000 00000000  1000F4FC 00000003 00000230 0000FFFF
A0=0003D090 00007836 00007A0E 00004B44  00062B3E 00008C98 00DFF000 000003F4
PC = 00004066 USP = 000018B2 SR = 2110 T=0 S=1 I=001 X=1 N=0 Z=0 V=0 C=0

M 3F4
:0003F4 00 00 0C BE 00 00 44 B0 00 00 05 98 41 F9 00 00 ......D.....A...
```

On pose un *breakPoint* sur **(A7)**, en l'occurrence *$CBE* (encore et toujours).
Taper : `BS CBE` puis on retourne au code du jeu avec `X`

Après un long chargement, notre *breakPoint $CBE* est atteint, on entre automatiquement dans l'AR.
Taper : `SM 1:20_49, 3D090 3D090+2D690`

On enlève le Cheat d'invulnérabilité

```
a 2FB8
^002FB8 SUBQ.W #1,20(A4)
^002FBC
a 14CC
^0014CC SUBQ.W #2,20(A4)
^0014D0
a 152C
^00152C SUBQ.W #2,20(A4)
^001530
```

Taper : `BDA` puis `X`
Le Level4 apparait.

On met à jour notre tableau.

Est Appelé par	Head	Move	A0	D0	D1	Length	Chargement de	Extra Info
$4326 + TrkLoad(2)	**DOWN**	14 ➔ 14	5AE3E	0E	01	$1838	Signature Disque2	Check Disk2
$4D4 + TrkLoad(2)	**DOWN**	08 ➔ 12	D55A	08	05	$7918	Menu du jeu	
$D76 + TrkLoad(2)	**DOWN**	01 ➔ 04	7541E	01	04	$60E0	Menu choix Level	
$C50+$CB4+TrkLoad(2)	**UP**	01 ➔ 30	3D090	01	1E	$2D690	Level1	Check T00 en fin de chargement
$C6C+$CB4+TrkLoad(2)	**UP**	31 ➔ 60	3D090	1F	1E	$2D690	Level2	Check T00 en fin de chargement
$C88+$CB4+TrkLoad(2)	**DOWN**	50 ➔ 79	3D090	32	1E	$2D690	Level3	Check T00 en fin de chargement
$CA4+$CB4+TrkLoad(2)	**DOWN**	20 ➔ 49	3D090	14	1E	$2D690	Level4	Check T00 en fin de chargement

De nouveau on entre dans l'AR pour remettre l'invulnérabilité et sans oublier notre breakpoint sur le trackloader.

```
a 2FB8
^002FB8 TST.W  20(A4)
^002FBC
a 14CC
^0014CC TST.W 20(A4)
^0014D0
a 152C
^00152C TST.W  20(A4)
^001530

bs 4066
Breakpoint inserted
Ready.
```

On n'oublie pas de vérifier la monnaie.

Taper : `M 8C2E` on change/vérifie les deux premières bytes : **02 FD**

```
M 8C2E
:008C2E 02 FD 00 0C 00 00 00 00 00 00 5E 00 50 00 52 ..........^.P.R
:008C3E 00 00 00 00 00 00 00 00 00 06 05 00 00 00 02 ................
```

Et on retourne au Level4, **taper : `X`**
Ne toucher plus à rien, laisser le jeu tel quel pendant 15minute, ne jouer pas, n'entrer pas dans les autres SHOPS.
Sauf que… si vous faite ça, la position de départ du vaisseau fait que vous allez entrer dans le 1er Shop
Et la… c'est le drame, écran noir et plus rien ne se passe.

Entrer dans l'AR et taper : `D` (On est bloqué en vers **$3164**)
Taper : `D 3161` (et remonter un peu dans le code)

```
0000314E        MOVEA.L #0,A0           ; → A0=0
00003154        ADDA.W  #200,A0         ;   A0=A0+200
00003158        ADDA.W  #200,A0         ;   Encore une fois A0=A0+200
0000315C        MOVE.W  #2237,D0        ;   D0=2237
00003160        MOVEQ   #0,D1           ;   D1=0
00003162        ADD.W   (A0)+,D1        ;   → Copie le Word de l'adresse A0 dans D1 et ensuite A0=A0+1 Word
00003164        DBF     D0,3162         ;   ← Décrémente D0 et tant qu'il n'est pas à -1, boucle
                                        ;   Sinon on arrive juste en dessous avec une valeur calculée en D1

00003168        LEA     CCC0,A0         ;   A0=CCC0
0000316E        CMP.W   898(A0),D1      ;   Compare le contenu de A0+898 avec notre valeur préalablement calculée en D1
00003172        BNE     314E            ; ← Si pas égale alors on branche en repart pour un tour.
```

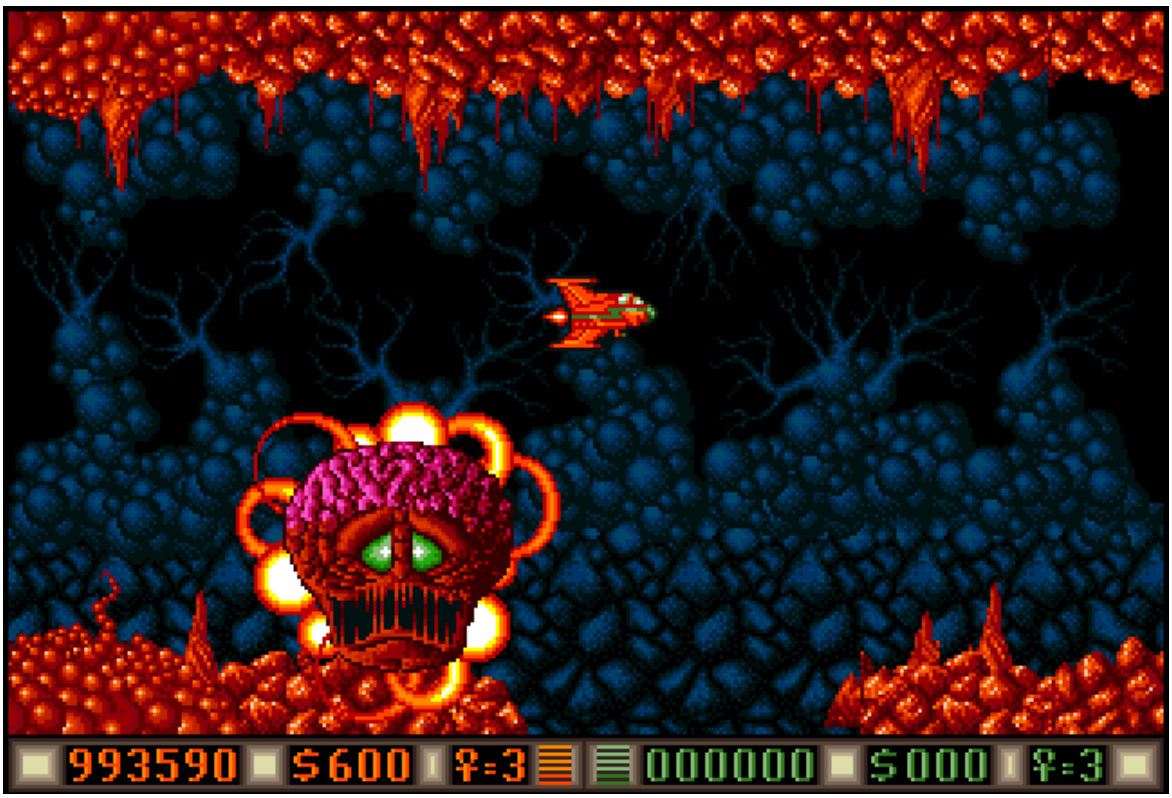
Une belle routine checksum tout en un. Semble Vérifier que le code en **$400** n'a pas été modifié (notre *trackloader(1)*)
On va tout simplement désactiver ce **BNE**
Taper : `A 3172`

```
^003172 NOP                     ; On désactive le BNE du CheckSum
^003174 <RETURN>
```

Taper : `X`
Une fois de retour au code du jeu, l'écran noir devrait maintenant faire place à une jolie image de boutique !

Sortez de la boutique et laissez le jeu continuer tranquillement.

Une fois la fin de Level atteint, **tuer le Boss de fin de Niveau** (et du jeu)



Un nouvel écran apparait vous indiquant votre score, **appuyer sur FIRE pour continuer**

Suivi d'un joli écran nous indiquant que l'on a fini le jeu.

Quelques instants plus tard, un appel est effectué vers notre *trackloader(2*) et on entre automatiquement dans l'AR

Taper : `R` et regarder **D0**, **D1** et **A0** ainsi que **A7**

```
r
D0=0000003D 0000000D 0000FFFF 0000BE01  00000000 0000FFFE 00000000 0000FFFF
A0=0001B312 00008452 0006AD3E 0000748A  00062B3E 00008C98 00DFF000 000003F4
PC = 00004066 USP = 000018B2 SR = 2100 T=0 S=1 I=001 X=0 N=0 Z=0 V=0 C=0
m 3F4
:0003F4 00 00 4B B8 00 00 08 08 00 00 06 2E 41 F9 00 00 ..K.........A...

bs 4BB8
Breakpoint inserted
Ready.
x
```

On pose un *breakPoint* sur **(A7)**, en l'occurrence *$4BB8*.
Taper : `BS 4BB8` puis on retourne au code du jeu avec `X`

Après un chargement, notre *breakPoint $4BB8* est atteint, on entre automatiquement dans l'AR.

On s'apprête à sauver **$13AD8** de DATA, 80600 octets en décimal soit un peu moins de 79Ko
Si on regarde notre disque ***bloodmoneyD2,*** nous avons 803 Ko utilisé, il nous reste plus que 34Ko

Cependant sur le disque ***bloodmoneyD1,*** nous avons 731 Ko utilisé, il nous reste environ 106Ko de libre
On va donc utiliser le disque ***bloodmoneyD1*** pour la sauvegarde de l'animation de fin.

Swaper donc votre disque ***bloodmoneyD2*** par ***bloodmoneyD1***
Taper : `SM 1:61_73, 1B312 1B312+13AD8`

On enlève le Cheat d'invulnérabilité

```
a 2FB8
^002FB8 SUBQ.W #1,20(A4)
^002FBC
a 14CC
^0014CC SUBQ.W #2,20(A4)
^0014D0
a 152C
^00152C SUBQ.W #2,20(A4)
^001530
```

Taper : `BDA` puis `X`

L’intro de fin apparait.

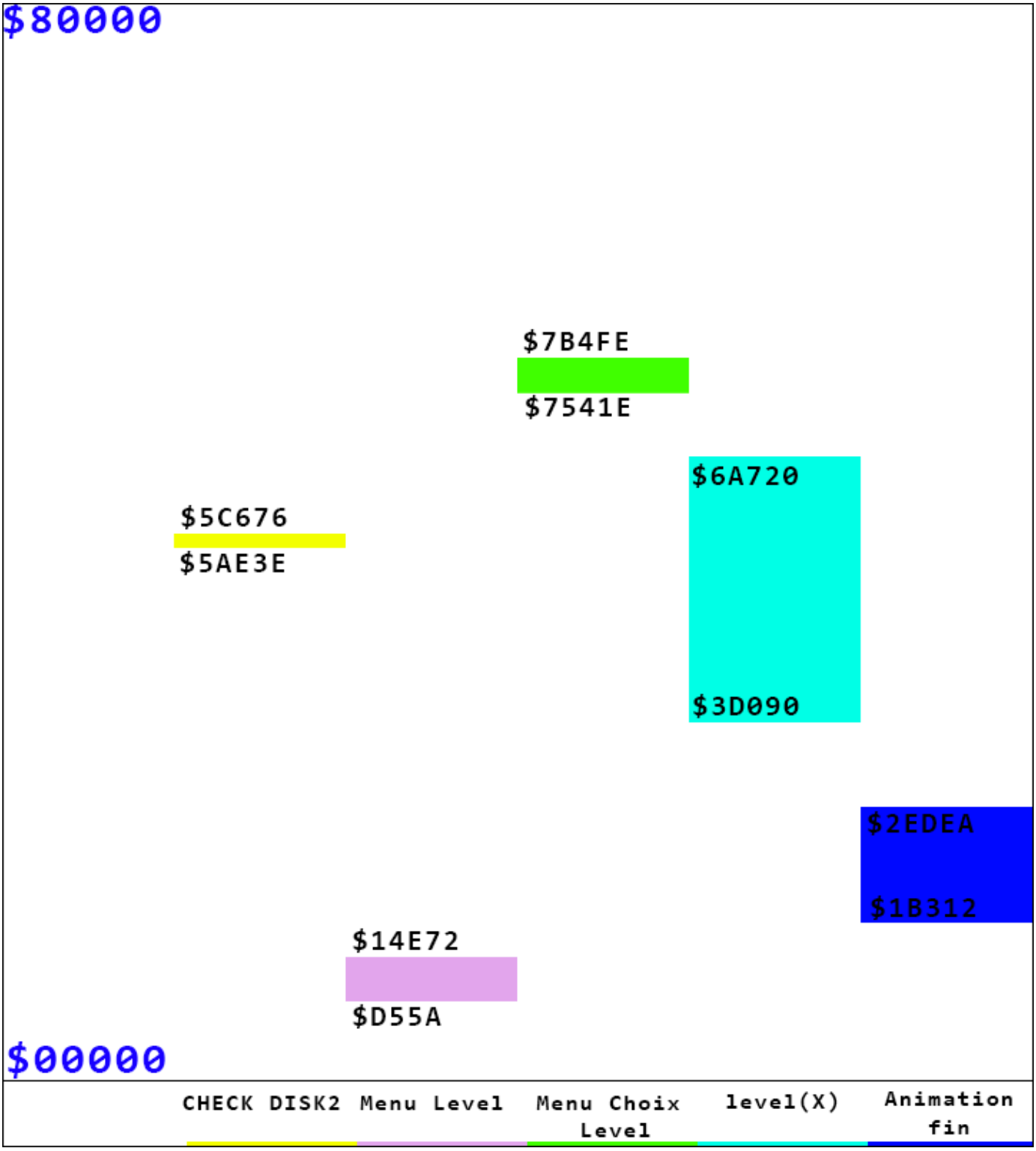
On met à jour notre tableau.

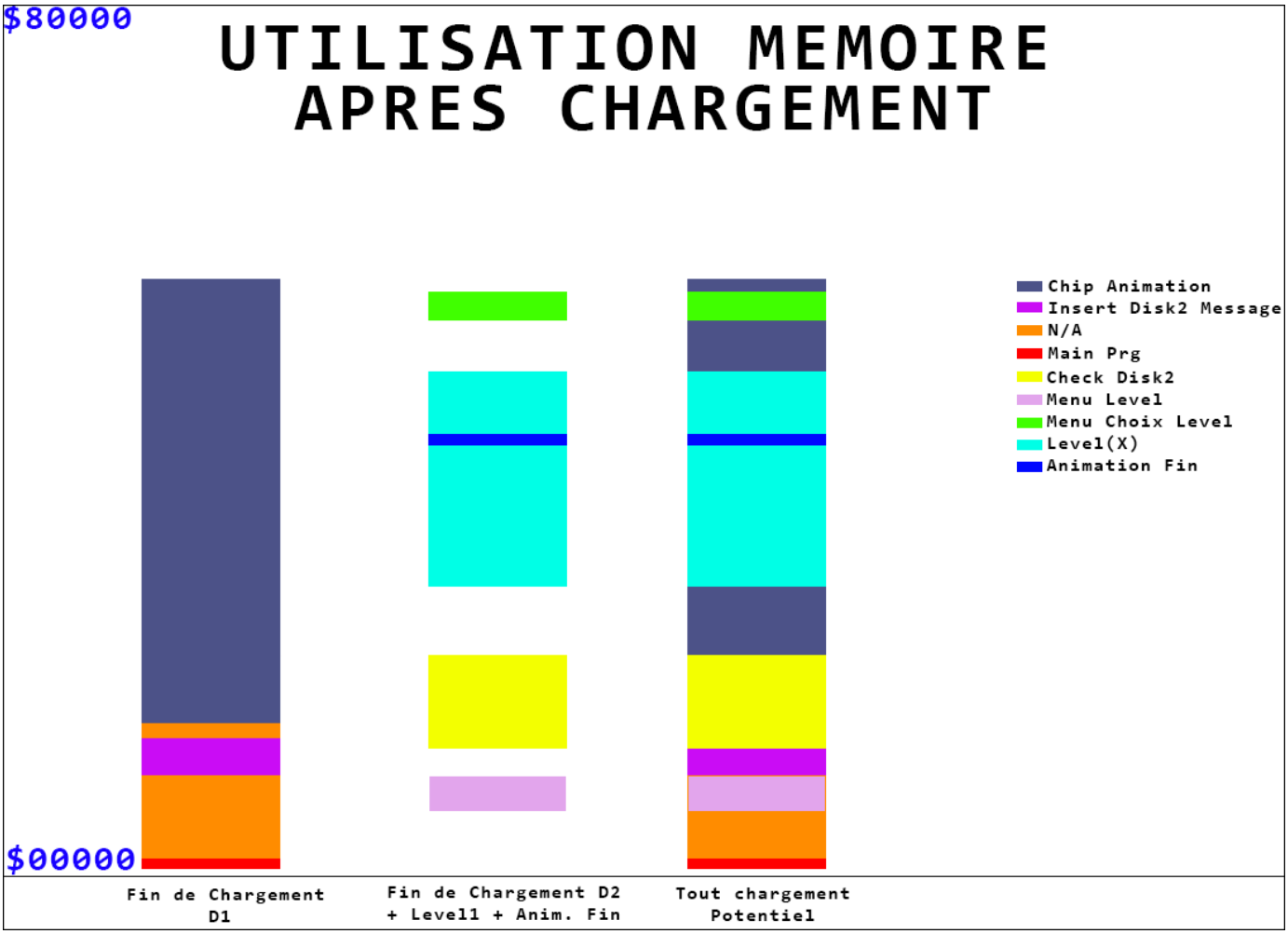
Est Appelé par	Head	Move	A0	D0	D1	Length	Chargement de	Extra Info
$4326 + TrkLoad(2)	**DOWN**	14 ➔ 14	5AE3E	0E	01	$1838	Signature Disk2	Check Disk2
$4D4 + TrkLoad(2)	**DOWN**	08 ➔ 12	D55A	08	05	$7918	Menu du jeu	
$D76 + TrkLoad(2)	**DOWN**	01 ➔ 04	7541E	01	04	$60E0	Menu choix Level	
$C50+$CB4+TrkLoad(2)	**UP**	01 ➔ 30	3D090	01	1E	$2D690	Level1	Check T00 en fin de chargement
$C6C+$CB4+TrkLoad(2)	**UP**	31 ➔ 60	3D090	1F	1E	$2D690	Level2	Check T00 en fin de chargement
$C88+$CB4+TrkLoad(2)	**DOWN**	50 ➔ 79	3D090	32	1E	$2D690	Level3	Check T00 en fin de chargement
$CA4+$CB4+TrkLoad(2)	**DOWN**	20 ➔ 49	3D090	14	1E	$2D690	Level4	Check T00 en fin de chargement
$4BA2 + TrkLoad(2)	**UP**	61 ➔ 73	1B312	3D	0D	$13AD8	Animation de Fin	

On en profite pour la comparer avec ce que l’on avait noté au début de la partie 10 de ce tuto, ok, tout y est.
SAUF l’appel en doublon en $D36

Est Appelé par	Head	Move	A0	D0	D1	LENGTH	**ZONE MEMOIRE UTLISEE**
$D36 + TrkLoad(2)	DOWN	05 ➔ 08	D55A	05	03	$48A8	$D55A ➔ $11E02
$4D4 + TrkLoad(2)	DOWN	08 ➔ 12	D55A	08	05	$7918	$D55A ➔ $14E72
$CDA + TrkLoad(2)	DOWN	08 ➔ 12	D55A	08	05	$7918	$D55A ➔ $14E72
$4BA2 + TrkLoad(2)	UP	61 ➔ 73	1B312	3D	0D	$13AD8	$1B312 ➔ $2EDEA
$4326 + TrkLoad(2)	DOWN	14 ➔ 15	5AE3E	0E	01	$1838	$5AE3E ➔ $5C676
$C50 + $CB4 + TrkLoad(2)	UP	01 ➔ 30	3D090	01	1E	$2D690	$3D090 ➔ $6A720
$C6C + $CB4 + TrkLoad(2)	UP	31 ➔ 60	3D090	1F	1E	$2D690	$3D090 ➔ $6A720
$C88 + $CB4 + TrkLoad(2)	DOWN	50 ➔ 79	3D090	32	1E	$2D690	$3D090 ➔ $6A720
$CA4 + $CB4 + TrkLoad(2)	DOWN	20 ➔ 49	3D090	14	1E	$2D690	$3D090 ➔ $6A720
$D76 + TrkLoad(2)	DOWN	01 ➔ 04	7541E	01	04	$60E0	$7541E ➔ $7B4FE

Finalement, cela nous donne le tableau suivant pour l'utilisation de la plage mémoire dans l'ordre de chargement.

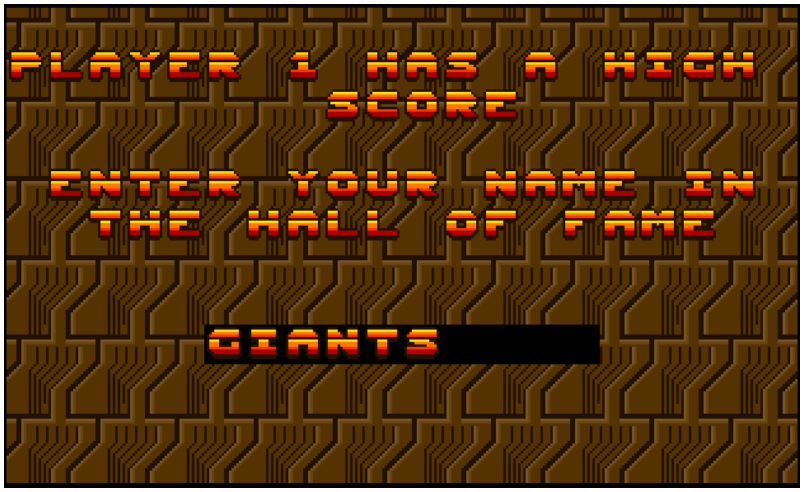



$80000
UTILISATION MEMOIRE
APRES CHARGEMENT
Chip Animation
Insert Disk2 Message
N/A
Main Prg
Check Disk2
Menu Level
Menu Choix Level
Level(X)
Animation Fin
$00000
Fin de Chargement
D1
Fin de Chargement D2
+ Level1 + Anim. Fin
Tout chargement
Potentiel

Part 11 Les HighScores

Sur cette intro de fin, on entre dans l'AR et on repose notre breakpoint du *trackloader(2)*
Taper : `BS 4066` puis on retourne au code du jeu avec `X`

Après une autre page nous invitant à recommencer, nous arrivons finalement sur la page des Highscores (sans aucun accès disque effectuée)
Entrer le nom du gros tricheur que vous êtes et valider le tout.



Une fois le tout validé, rien ne se passe, on semble bloqué dans une *DeadLoop*
Entrer dans l'AR et taper : `D`

Vous devriez vous trouver quelque part par-là :

```
~004994 SUBA.L  A0,A0
~004996 ADDA.W  #200,A0
~00499A ADDA.L  A0,A0
~00499C MOVE.W  #2327,D0
~0049A0 MOVEQ   #0,D1
~0049A2 ADD.W   (A0)+,D1
~0049A4 DBF     D0,000049A2
~0049A8 SUB.W   0000D558,D1
~0049AE BNE     00004994
~0049B0 RTS
;=====================================
```

Taper : `BS 49B0` puis `X`
… Rien… il ne se passe rien, le *breakpoint* **$49B0** n'est jamais atteint. Le code semble bloqué dans cette boucle de décrémentation de **D1**

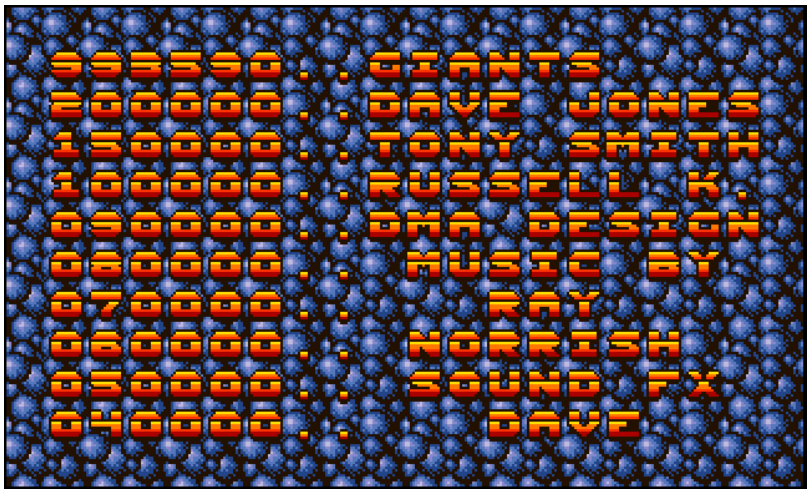
On va faire sauter ce **BNE** en *$49AE*, **Entrer dans l'AR et taper :** `A 49AE`

```
^049AE  NOP
^049B0  <RETURN>
```

La tête de lecture c'est déplacé et semble avoir effectué un I/O
(si vous avez WinUae vous pouvez vois qu'elle est sur T13) mais nous ne sommes pas passé par notre *trackloader(2).*

Ce qui au passage ne devrais pas nous surprendre car les pistes de HighScores sont souvent dans un format différent de celui du reste de la disquette et utilise aussi leur propre routine de Lecture/écriture.

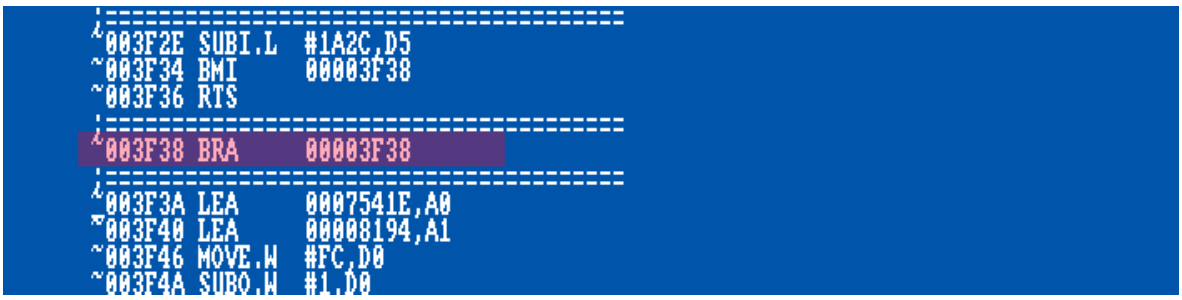
Nous voilà maintenant sur la page des HighScores.



Appuyer sur Fire
Nous voilà de retour sur la page de Menu du jeu, aucun accès disque n'a été effectuée.

Pendant qu'on est sur le sujet **DeadLoop**, on va regarder si on trouve ce genre de code flagrant.
Un **BRA** sur lui-même en **OPCODE** nous donnes : **60 FE**
On va donc chercher cet **OPCODE** en mémoire, **sur l'écran menu du jeu taper** : **F 60 FE**

Selon la position où l'on se trouve dans le jeu, il est possible d'avoir un ou plusieurs résultats.
Vous pouvez utiliser la commande **D** sur chaque adresse possible pour regarder le code au-dessus et en dessous de l'adresse en question.
Une seul ressortira du lot à savoir : *$3F38*



Il faut bien sur la désactiver, **taper** : **A 3F38**

```
^03F38  RTS
^03F3A  <RETURN>
```

Part 12 Topo analyse du RIP Disk 2

Nous voilà donc ok pour le disque2, regardons les positions de ce que l'on a ripé.

LOWER	UPPER
01 ➔ 04 05 ➔ 08 08 ➔ 12 **13 ➔ 13** 14 ➔ 14 20 ➔ 49 50 ➔ 79	01 ➔ 30 31 ➔ 60 61 ➔ 73 **74 ➔ 79**

LOWER

Nous avons une piste non utilisée par les **trackloader** en T13 (voir chapitre HighScore)
Ainsi qu'un ensemble pistes vide de 15 à 19
Noter que si vous êtes curieux concernant la T14 (au vue de sa taille d'une Track on peut être curieux)
Vous pouvoir voir qu'elle ne contient pas grand-chose à part une signature Disk pour indiquer Disquette N°2 (codé 2222 au début du fichier/track)
A voir avec un **LM 14_14, 20000** et **M 20000**

T12.0	80	E0D37ED1	Good	12529	100232 bits = Data=99312 + Gap=920	51	411	025	1	Auto	cell_2us	0	None
T13.0	80	4D04F214	Good	12528	100224 bits = Data=99312 + Gap=912	51	411	027	1	Auto	cell_2us	0	None
T14.0	80	81D4FA2E	Good	12529	100232 bits = Data=99312 + Gap=920	50	404	029	1	Auto	cell_2us	0	None
T15.0	80	079B3962	Good	12529	100232 bits = Data=99312 + Gap=920	51	413	031	1	Auto	cell_2us	0	None
T16.0	80	DC3C5DDB	Good	12528	100224 bits = Data=99312 + Gap=912	52	416	033	1	Auto	cell_2us	0	None
T17.0	80	8CC967FE	Good	12529	100232 bits = Data=99312 + Gap=920	51	411	035	1	Auto	cell_2us	0	None
T18.0	80	8B41FFA8	Good	12529	100232 bits = Data=99312 + Gap=920	52	416	037	1	Auto	cell_2us	0	None
T19.0	80	05D973A0	Good	12529	100232 bits = Data=99312 + Gap=920	50	405	039	1	Auto	cell_2us	0	None
T20.0	80	2D20C96B	Good	12528	100224 bits = Data=99312 + Gap=912	49	398	041	1	Auto	cell_2us	0	None
T21.0	80	FBF19FF2	Good	12528	100224 bits = Data=99312 + Gap=912	52	419	043	1	Auto	cell_2us	0	None

UPPER

Les vont jusqu'à la piste 73

T71.1	80	E725D179	Good	12529	100232 bits = Data=99312 + Gap=920	50	403	144	1	Auto	cell_2us	0	None
T72.1	80	2C64AB36	Good	12529	100232 bits = Data=99312 + Gap=920	49	393	146	1	Auto	cell_2us	0	None
T73.1	80	5295B28A	Good	12529	100232 bits = Data=99312 + Gap=920	50	405	148	1	Auto	cell_2us	0	None
T74.1	80	9D730F0E	Good	12529	100232 bits = Data=99312 + Gap=920	50	402	150	1	Auto	cell_2us	0	None
T75.1	80	A1401D33	Good	12528	100224 bits = Data=99312 + Gap=912	50	400	152	1	Auto	cell_2us	0	None
T76.1	80	B8087320	Good	12529	100232 bits = Data=99312 + Gap=920	50	400	154	1	Auto	cell_2us	0	None
T77.1	80	7B4E49B9	Good	12529	100232 bits = Data=99312 + Gap=920	50	403	156	1	Auto	cell_2us	0	None
T78.1	80	27A42978	Good	12529	100232 bits = Data=99312 + Gap=920	51	410	158	1	Auto	cell_2us	0	None
T79.1	80	A7F32CE1	Good	12529	100232 bits = Data=99312 + Gap=920	52	417	160	1	Auto	cell_2us	0	None
T80.1	80	24FA257F	Good	0	0 bits = Data=0 + Gap=0	0	0	162	0	Noise	cell_2us	0	None
T81.1	80	04F5A844	Good	0	0 bits = Data=0 + Gap=0	0	0	164	0	Noise	cell_2us	0	None
T82.1	80	3EE6EC64	Good	0	0 bits = Data=0 + Gap=0	0	0	166	0	Noise	cell_2us	0	None
T83.1	80	44EAB232	Good	0	0 bits = Data=0 + Gap=0	0	0	168	0	Noise	cell_2us	0	None

Bref, on a ici des disques finalement assez remplis.

Part 13 Piste/code des HighScores

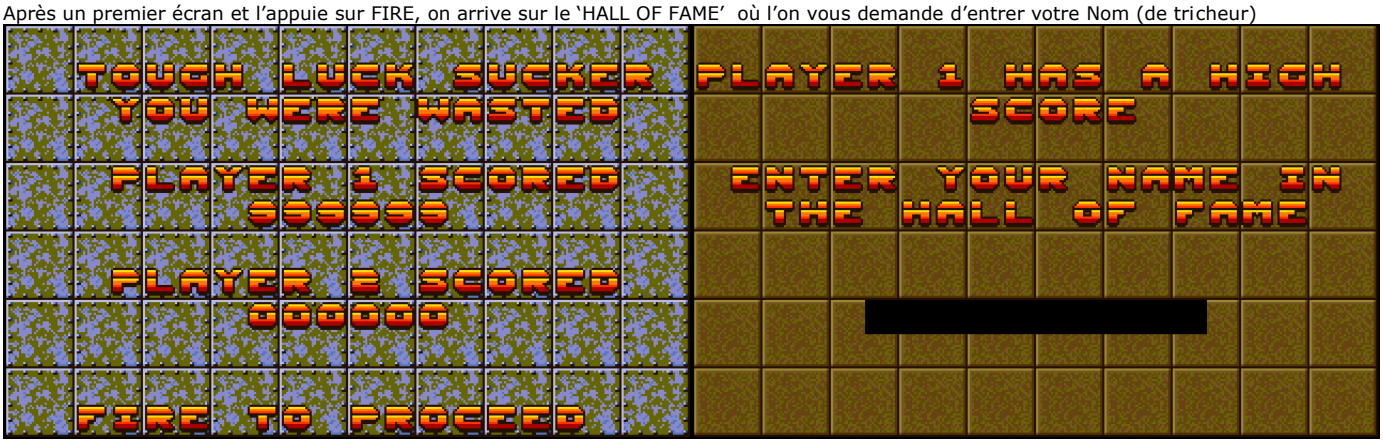
On revient sur cette histoire de HighScore.

Rebooter votre amiga avec la disquette originale : D1 BloodMoney
Changer de disque comme demandé pour arriver au menu du jeu et **lancer une partie dans le Level1**
Entrer dans l'AR, taper : `M 8C2B` et changer la valeur par **0F 42 3F**

```
M 8C2B
:008C2B 0F 42 3F 00 64 00 0C 00 00 00 00 00 00 64 00 ....d.........d.
:008C3B 64 00 52 00 00 00 00 00 00 00 00 00 06 06 00 00 d.R.............

X
```

Valider le tout et taper : `X`
Crashez votre vaisseau, perdez toute vos vies.

Après un premier écran et l'appuie sur FIRE, on arrive sur le 'HALL OF FAME' où l'on vous demande d'entrer votre Nom (de tricheur)



Entrer dans l'AR, taper : `D`

On se trouve quelque part dans la sous routine 'Keyboard Input for Highscore' entre **49FE** et **4A48**

```
;=======================================
~0049FE LEA     000082E0,A1
~004A04 LEA     00008136,A0
~004A0A MOVEQ   #0,D0
~004A0C CLR.W   D1
~004A0E MOVE.B  5C(A5),D1
~004A12 BEQ     00004A0C
~004A14 CLR.B   5C(A5)
~004A18 CMP.B   #44,D1
~004A1C BEQ     00004A6A
~004A20 CMP.B   #41,D1
~004A24 BEQ     00004A4A
~004A28 MOVE.B  0(A1,D1.W),2(A0,D0.W)
~004A2E MOVEM.L D0/A0-A1,-(A7)
~004A32 LEA     0006A83E,A1
~004A38 BSR     00003596
~004A3C MOVEM.L (A7)+,D0/A0-A1
~004A40 ADDQ.W  #1,D0
~004A42 CMP.W   #A,D0
~004A46 BNE     00004A0C
~004A48 RTS
;=======================================
```

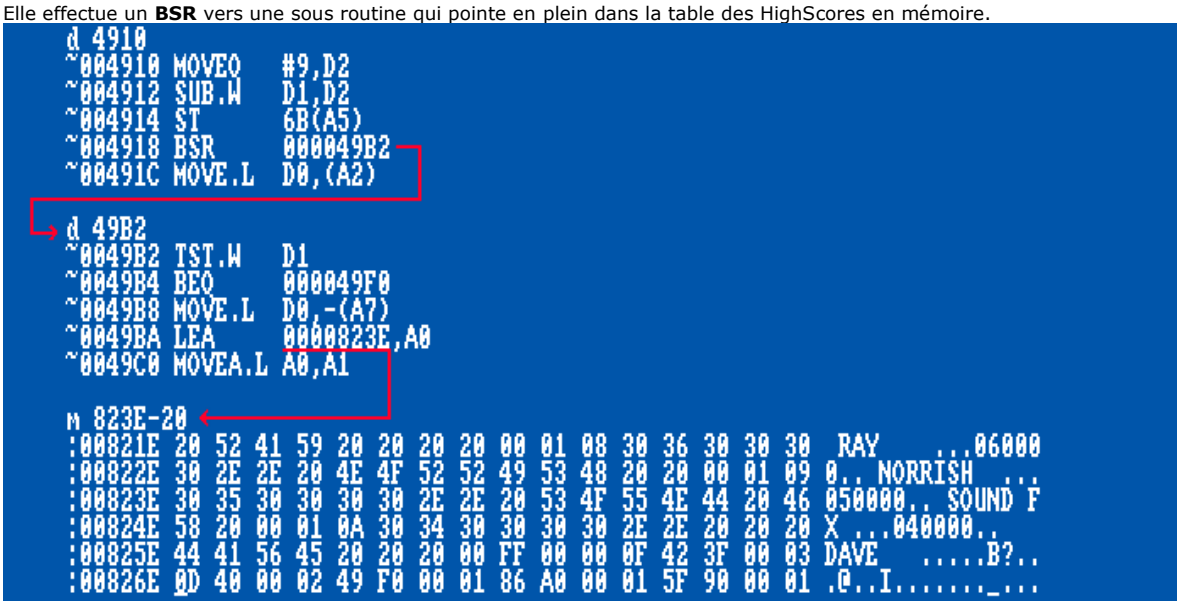
Taper : `R` et regarder ou pointe **(A7)**

```
R
D0=00000003 00000000 00000000 0000BE01  00000000 0000FFFE 00000000 0000FFFF
A0=00008136 000082E0 0006AD62 0000748A  00062B62 00008C98 00DFF000 000003EC
PC = 00004A0C USP = 000018B2 SR = 2104 T=0 S=1 I=001 X=0 N=0 Z=1 V=0 C=0

M 3EC
:0003EC 00 00 49 7E 00 00 81 96 00 00 48 DA 00 00 07 08 ..I~......H.....
```

On regarde le code en **$497E**, à savoir là où le code retournera donc.
Taper : `D 497E` et chercher le début et la fin de cette sous routine.
La sous routine en question commence en $4910 et fini en $49B0 avec en **497E** notre **BSR 49FE**
Elle est assez longue et travaille beaucoup avec **A0**

Néanmoins, si on regarde au début de cette sous routine on peut voir qu'il y a de grande chance que ce soit la routine principale qui gère les **HighScore** :

Elle effectue un **BSR** vers une sous routine qui pointe en plein dans la table des HighScores en mémoire.



Néanmoins, rien de probant au début de ces deux sous routine comme test sur le score.
On a bien en **49B2** un **TST** sur **D1** suivi d'un **BEQ** mais… **BEQ** qui retombe en plein dans notre sous routine précédente.
On continue donc de remonter un peu le code.

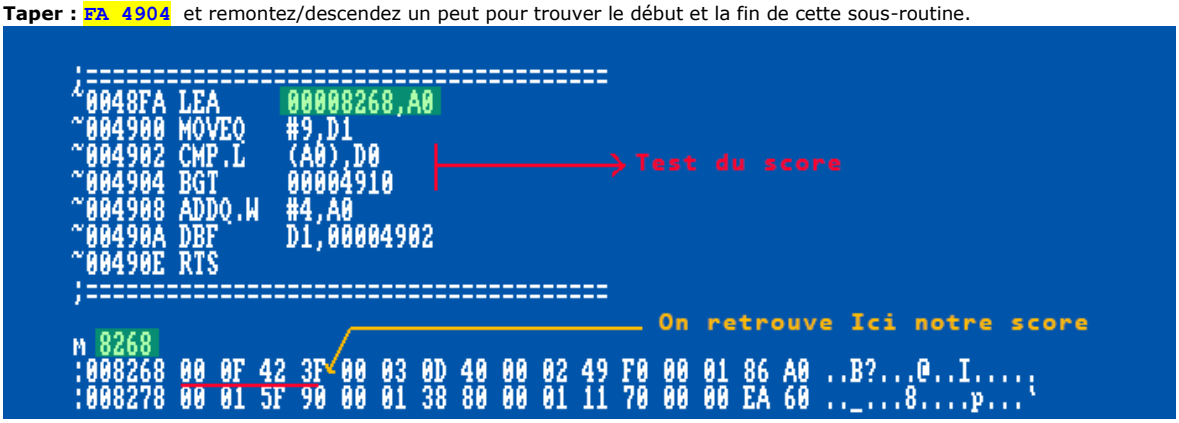
Comme vu précédemment, notre sou routine commence en 4910 on va donc chercher où son branchement est effectuée.
Taper : `FA 4910`

L'AR nous donne une seule réponse possible :

Ce qui est très intéressant pour deux raisons :

- Il n'y a qu'un seul branchement
- Ce branchement est effectué par un **BGT**

Taper : `FA 4904` et remontez/descendez un peut pour trouver le début et la fin de cette sous-routine.

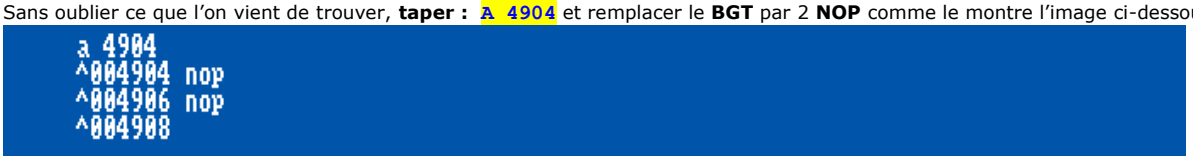


Là, je pense qu'on est bon.

```
48FA    LEA 00008268,A0         ; On met 2868 dans A0. 8268 qui, comme le montre l'image ci-dessus
                                ; contient le score que l'on a mis via l'AR au début de chapitre
4902    CMP.L   (A0),D0         ; Un test est effectué entre ce contenu de A0 et D0
4904    BGT     4910            ; Un branchement est effectué si 'plus grand que'
```

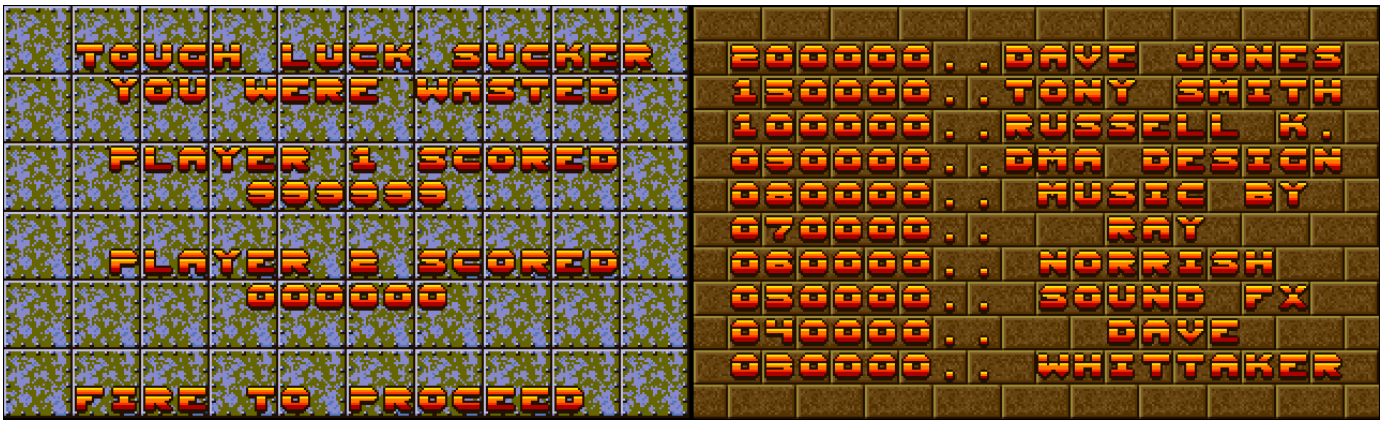
On va vérifier tout ça !
Rebooter votre amiga et effectuer les mêmes manipulations qu'au début de ce chapitre pour arriver sur le Level1
Entrer dans l'AR et patcher le score en **$8C2B** comme précédemment, **taper :** `M 8C2B` et changer la valeur par **0F 42 3F**
On n'oublie pas notre **NOP en** `49AE` pour ne pas être bloqué sur la section **HighScore (au cas où)**

Sans oublier ce que l'on vient de trouver, **taper :** `A 4904` et remplacer le **BGT** par 2 **NOP** comme le montre l'image ci-dessous



Retourner au code du jeu, **taper : X**
Crashez votre vaisseau, perdez toute vos vies.

Après un premier écran identique à ce que l'on à déjà vue, **appuyer sur FIRE** et…est affiché la table des HighScores.
Nous ne sommes pas entrés dans la routine ou l'on nous demande d'entrer notre nom, impeccable ☺



Après un appuie sur Fire, nous voilà de nouveau sur la page du Menu du jeu.
Cependant, si on attend un peu et que le Level1 se lance en démo ou qu'on démarre une partie, nous avons le droit à un jolie écran noir (Écran noir ou garbage à l'écran). Nous sommes Ici en présence d'une jolie protection checksum qu'il va falloir trouver.

Rebooter votre amiga et effectuer les mêmes manipulations qu'au début de ce chapitre pour arriver sur le Level1
Entrer dans l'AR et patcher le score en *$8C2B* comme précédemment, **taper : M 8C2B** et changer la valeur par **0F 42 3F**
On désactive le **BGT** par deux **NOP en** *$4904* pour ne pas entrer dans la sous-routine **HighScore**
On n'oublie pas notre **NOP en** *$49AE* pour ne pas être bloqué sur la section **HighScore (au cas où)**

Retourner au code du jeu, **taper : X**
Crashez votre vaisseau, perdez toute vos vies.
Appuyer une fois sur FIRE afin d'arriver sur le Menu principal du jeu, **ne toucher plus à rien**
Au bout de quelques secondes, le level 1 devrait se lancer en démo mais nous avons à la place un joli écran noir.

Entrer dans l'AR, taper : D
Nous sommes bloqués quelque part vers **$2C04**

Taper : D 2C04 et remonter un peut dans le code, vous pouvez voir que cette sous-routine commence en **$2BEC**

On va chercher qui appel cette adresse, **taper : FA 2BEC** *(l'AR nous retourne 2 résultats)*

```
0598    BSR     2BEC            ; pas suspect puisque c'est un branchement sans condition.
2C16    BNE     2BEC            ; suspect car c'est un branchement sur condition.
```

On regarde donc en $2C16, **taper : D 2C16** (et remonter dans le code)

Pas besoin de remonter très loin, nous avons un superbe **CMP.W** juste au-dessus qui travaille avec l'adresse **$D558**

```
002C10  CMP.W   D558,D0         ; Test sur $D558 et selon le résultat branche en 2BEC.
                                ; Cela ressemble plus que fortement à l'adresse d'un checksum calculé
002C16  BNE     2BEC            ; Notre branchement ci-dessus
```

On cherche logiquement **$D558** en mémoire, t**aper : F D5 58**
L'AR nous donnes 4 possibilités.

00002C14 <-> 000044C6 <-> 000049AC <-> 00044E0E

00002C14 **=** C'est celui vue juste au-dessus, le **CMP.W D558,D0**

000044C6 **=** Très intéressant celui-ci, voyez plutôt, **taper : D44C6-4**

```
44C2    MOVE.W  D558,D0         ; D0=D558           adresse du checksum en court de création
44C8    LEA     4A50.S,A0       ; A0=4A50.S         adresse source ou on va lire le code
44CC    MOVE.W  #2328,D6        ; D6=2328           compteur
44D0    MOVEQ   #0,D1           ; D1=0
44D2    ADD.W   -(A0),D1        ; ➔ Ajoute la valeur de l'adr Source avec D1 et le met en D1 et on remonte ensuite d'1 word
44D4    SUBQ.W  #1,D6           ; On enleve 1 à notre compteur (qui va donc se faire sur 2328 word)
44D6    BNE     44D2            ; ← tant que le compteur n'est pas a Zero, on recommence
44D8    SUB.W   D1,D0           ; D0=D0-D1, influe sur le résultat de Z
44DA    BNE     44D2            ; ← On boucle encore  tant que la condition n'est pas bonne.
44DC    RTS                     ; On retourne à la maison avec au passage un jolie checksum calculé en D558
```

000049AC **=** Ressemble à un checksum mais noyé dans une sous-routine beaucoup plus grosse et complexe, on le met de côté quand même.
00044E0E **=** Ne ressemble pas à du code

Bon, malheureusement, nous allons encore devoir rebooter notre amiga pour tester tout ça.
Bien sûr, si vous utilisé **WinUAE** vous pouvez utiliser l'option Load/Save State qui permettra d'aller plus vite.

Sinon :
Rebooter votre amiga et effectuer les mêmes manipulations qu'au début de ce chapitre pour arriver sur le Level1
Entrer dans l'AR et patcher le score en *$8C2B* comme précédemment, **taper : M 8C2B** et changer la valeur par **0F 42 3F**
On désactive le **BGT** par deux **NOP en** *$4904* pour ne pas entrer dans la sous-routine **HighScore**
On n'oublie pas notre **NOP en** *$49AE* pour ne pas être bloqué sur la section **HighScore (au cas où)**

On va maintenant désactiver le **BNE** en *$2C16,* **taper : A 2C16**

```
^02C16  NOP                     ; GoTo #checksum enlevé
^02C18  <RETURN>
```

Taper : X
Et faites le même test que précédemment.
Perdez toutes vos vies, passer les divers écrans du jeu et arriver sur le MENU.
De toucher a rien, attendez que la démo du *LEVEL1* se lance et… BINGO, ça fonctionne.

De retour au menu, **appuyer sur F1** pour commencer une partie sur le Level1
mais…le level1 ne se lance pas et l'on a droit à un joli crash écran.
Entrer dans l'AR, taper : D
Nous sommes quelque part vers **$44D6** (le même code donc vue juste au-dessus)
Plus exactement, nous sommes bloqués par une boucle effectuée par le **BNE 44D2** en $44DA

```
44D2    ADD.W   -(A0),D1        ; → Ajoute la valeur de l'adr Source avec D1 et le met en D1 et on remonte ensuite d'1 word
44D4    SUBQ.W  #1,D6           ; On enleve 1 à notre compteur (qui va donc se faire sur 2328 word)
44D6    BNE     44D2            ; ← tant que le compteur n'est pas a Zero, on recommence
44D8    SUB.W   D1,D0           ; D0=D0-D1, influe sur le résultat de Z
44DA    BNE     44D2            ; ← On boucle encore  tant que la condition n'est pas bonne.
44DC    RTS                     ; On retourne à la maison avec au passage un
                                ; joli checksump calculé en D558
```

On va supprimer ce **BNE** en $44DA

Rebooter votre amiga et effectuer les mêmes manipulations qu'au début de ce chapitre pour arriver sur le Level1
Entrer dans l'AR et patcher le score en $8C2B comme précédemment, **taper : M 8C2B** et changer la valeur par **0F 42 3F**
On désactive le **BGT** par deux **NOP en** $4904 pour ne pas entrer dans la sous-routine **HighScore**
On n'oublie pas notre **NOP en** $49AE pour ne pas être bloqué sur la section **HighScore (au cas où)**
Ainsi que le **NOP** en $2C16 pour le 1er checksum vue juste au-dessus

Et on ajoute notre **NOP** en **44DA**, **taper :**
A 44DA

```
^044DA  NOP                     ; Sup. de la boucle dans la sous-routine du #checksum vue au-dessus
^024DC  <RETURN>
```

Taper : X
Et faites **exactement** le même test que précédemment.
Perdez toutes vos vies, passer les divers écrans du jeu et arriver sur le MENU.
De toucher a rien, attendez que la démo du *LEVEL1* se lance et fonctionne.
Appuyer sur Fire pour revenir au menu puis **appuyer sur F1** et choisissez le 1er Level
Impeccable ☺, cela fonctionne, nous sommes sous le Level1.

Part 14 Protection Track 00 sur le Disque2, nouveau trackloader(3)

On revient quelques minutes sur le code que l'on a vue en chapitre **Disk 2 – Nouveau Trackloader**
Routine en *$CB4* ➔ *$CCA*

```
;================================
000CB4  MOVEA.L #3D090,A0       ; A0=3D090, l'adresse mémoire de destination, unique pour les levelS
000CBA  BSR     4066            ; ➔ GoTo #TrackLoader
000CBE  BSR     434A            ; ➔ à définir
000CC2  BSR     462E            ; ➔ à définir
000CC6  CLR.W   1E(A5)          ; A5 = 8C98 sur tous les appels, $8C98+$1E=$8CB6, trouvable avec un BS CC6
                                ; Il met donc 00 00 (1 word) en $8CB6
000CCA  RTS                     ; E.T retour maison
;================================
```

On regarde ce qu'il y a en *$434A*, **entrer dans l'AR** si ce n'est pas déjà fait et **taper :** `D 434A`

```
;================================ #POST_LOAD_LEVEL
00434A  LEA     3D090,A0        ; A0=3D090, adresse unique utilisée pour tous les Level
004350  MOVEQ.L A0,A2           ; A2=A0 donc A2=3D090
004352  ADDA.L  #29810,A2       ; Ajoute $29810 à A2 qui passe donc à $668A0
004358  LEA     1B312,A1        ; A1=1B312
00435E  MOVE.W  #1BF1,D7        ; D7=1BF1
004362  MOVEQ   #F,D5           ; ➔ D5=F
004364  MOVE.W  (A2)+,D6        ; Copie le word de l'adresse A2 dans D6 puis D6=D6+1 Word
004366  BTST    D5,D6           ;       ➔ Compare le Bit D5 avec D6
004368  BEQ     4372            ;       Si identique (donc D6=$F), alors on branche en 4372
00436C  MOVE.W  (A0)+,(A1)+     ;       sinon on copie le word à l'adresse A0 vers l'adresse A1
                                ;       puis A0=A0+2 et A1=A1+1 (on avance d'1 Word)
                                ;       copiera donc
00436E  BRA     4374            ;       On branche en 4374
;================================
004372  CLR.W   (A1)+           ;       On efface le Word de l'adresse A1 puis A1=A1+2 (avance d'1 Word)
004374  DBF     D5,4366         ;       ➔ Décrémente D5 et tant qu'il n'est pas égal à -1, on boucle en 4366
004378  DBF     D7,4362         ; ➔ Décrémente D7 et tant qu'il n'est pas égal à -1, on boucle en 4362
00437C  JMP     (A2)            ; On saute à l'adresse contenue par A2
```

Cette sous-routine va calculer et créer l'adresse de saut par rapport à un algo utilisant les données du level chargée en **$3D090**
Taper : `BS 437C` puis `X`
Une fois le *BreakPoint* atteint on peut afficher **A2** et connaitre l'adresse calculé du **JMP** (qui sera la même pour les 4 Levels pour info)
Il va aussi copier le code du level préalablement chargé en **$3D090+**, le traité et l'envoyé vers **$1B312+** grâce au `MOVE.W (A0)+,(A1)+`

A2 : ***$6A084*** *(valeur identique pour les 4 level)*
A1 : ***$53152*** *(valeur identique pour les 4 level)*

Donc une partie du code préalablement chargé des levels est écrasé : de **$3D090** jusqu'à **$53152** par le code/traitement ci-dessus.

On jette un coup d'œil à l'adresse du saut, **A2**, **taper :** `D 6A084`

```
;================================
06A084  BSR     6A22C           ;
06A088  BSR     6A1DE           ;
06A08C  BCLR    #2,BFD100       ; Sélection face Upper
06A094  BSR     6A0A2           ;
06A098  BSR     6A0F2           ;
06A08C  BSR     6A248           ;
06A0A0  RTS                     ; E.T retour maison
```

Et on regarde tout ça, point par point.

>>6A22C

```
;================================ #CONF_DISQUE
06A22C  BCLR    #7,BFD100       ; Motor On
06A234  MOVEQ   #64,D6          ; D6=$64
06A236  BCLR    #3,BFD100       ; ➔ Select DF0
06A23E  DBF     D6,6A236        ; ➔ Petite boucle d'attente utilisant D6
06A242  BSR     6A166           ; GoTo #TRACK_ZERO
06A246  RTS
;================================

;================================ #DISK_READY?
06A166  MOVE.W  #BB8,D6         ; ➔ D6=$BB8
06A16A  DBF     D6,6A16A        ; ➔ Petite boucle d'attente utilisant D6
06A16E  BTST    #5,BFE001       ; ➔ Test Bit 5 du registre CIA-A, à savoir, RDY ?
06A176  BNE     6A16E           ; ➔ on boucle tant que le floppy n'est pas READY
06A178  RTS                     ;
```

>>6A1DE

```
;================================ #MOVE_TO_TRACK_ZERO
06A1DE  BTST    #4,BFE001       ; ➔    Test si on est sur TK0
06A1E6  BEQ     6A210           ;      Yep, on va en 6A210
06A1EA  BSET    #1,BFD100       ;      Nop, On déplace la tête pour y aller
06A1F2  NOP                     ;
06A1F4  NOP                     ;
06A1F6  BCLR    #0,BFD100       ;      Nop, On déplace la tête pour y aller
06A1FE  NOP                     ;
06A200  NOP                     ;
06A202  BSET    #0,BFD100       ;      Nop, On déplace la tête pour y aller
06A20A  BSR     6A166           ;      GoTo #DISK_READY?
06A20E  BSR     6A1DE           ; ➔ On boucle si tjs pas sur TK0
;================================
06A210  CLR.W   6A270           ; On efface le Word en 6A270
06A216  RTS                     ; E.T retour maison
```

>>6A0A2

```
;================================ #TRACKLOADER(3)
06A0A2  MOVE.W  #400,9E(A6)     ; ➔    ADKCON, disable WORDSYNC
06A0A8  MOVE.W  #8210,96(A6)    ;      DMACON (Disk DMA enable, enable All DMA)
06A0AE  MOVE.W  #5313E,20(A6)   ;      DSKPTH // adr. MFM buffer=$5313E
06A0B6  CLR.W   24(A6)          ;      DSKLEN,  DMA data Length clear
06A0BA  BSR     6A218           ;      GoTo #CONF_DiskIndexPulse
06A0BE  MOVE.W  #4000,24(A6)    ;      DSKLEN, Disk DMA=OFF
06A0C4  MOVE.W  #B778,24(A6)    ;      DSKLEN, Disk DMA=enable, Length DMA data=$3778
06A0CA  MOVE.W  #B778,24(A6)    ;      Encore une fois pour démarrer le transfert DMA
06A0D0  MOVE.W  #30D40,D1       ;      D1=30D40
06A0D6  MOVE.W  1E(A6),D0       ; ➔    D0=DFF01E (INTREQR)
06A0DA  SUBQ.L  #1,D1           ;      D1=D1-1
06A0DC  BEQ     6A0A2           ; ➔ Si flag Z=1, pas bon, on repart pour un tour #1
06A0DE  BTST    #1,D0           ; Test Bit 1 de INTREQR (Disk Block Finished Interupt)
06A0E2  BEQ     6A0D6           ; ➔ si flag Z=1 on repart pour un tour #2
06A0E4  MOVE.W  #2,9C(A6)       ; INTREQ, Disk Block Finished Interuppt
06A0EA  MOVE.W  #8400,9E(A6)    ; ADKCON, Enable SyncWord
06A0F0  RTS                     ; E.T retour maison
;================================ #CONF_DiskIndexPulse
06A218  MOVE.B  BFDD00,D6       ; D6=BFDD00=Conf ICR   CIA-B
06A21E  MOVE.B  BFDD00,D6       ; ➔D6=BFDD00=Conf ICR  CIA-B
06A224  BTST    #4,D6           ;   Test Bit 4 de BFDD00 (Disk Index Pulse)
06A228  BEQ     6A21E           ; ➔ Si flag Z=1 on repart pour un tour
06A22A  RTS                     ; Sinon, retour maison
```

>>6A0F2

```
;================================ #START_TRAITEMENT_MFM
06A0F2  MOVEA.L #5313E,A0       ; A0=5313E
06A0F8  MOVE.L  #3778,D3        ; D3=3778
06A0FE  MOVE.L  (A0),D3         ; ➔ Copie le LongWord de l'adresse A0 vers D3
06A100  ADDQ.L  #2,A0           ;    A0=A0+2
06A102  MOVE.L  #F,D2           ;    D2=F
06A108  MOVE.L  D0,D1           ;    ➔ D1=D0
06A10A  SWAP    D1              ;       Swap D1, D1=xxxxYYYY devient YYYYxxxx
06A10C  CMPI.W  #4454,D1        ;       Compare $4454 avec D1 (syncword ?)
06A110  BEQ     6A120           ;       Si C'est égal alors, GoTo #TRAITEMENT_MFM
06A112  LSL.L   #1,D0           ;       Décalage d'1bit vers la gauche de D0, ex sur 1 word : 0110 devient 1100
06A114  DBF     D2,6A108        ;    ➔ D2=D2-1 et tant que D2 n'est pas égale à -1, on boucle #1
06A118  DBF     D3,6A0FE        ; ➔ D3=D3-1 et tant que D3 n'est pas égale à -1, on boucle #2
06A1C1  BRA     6A164           ; Fin de traitement, RTS
;================================ #TRAITEMENT_MFM
06A120  MOVE.L  #0,D5           ; D5=0
06A126  MOVE.L  (A0),D0         ; ➔ Copie le LongWord de l'adresse A0 vers D0
06A128  ADDQ.L  #2,A0           ;    A0=A0+2
06A12A  MOVE.L  #F,D2           ;    D2=F
06A130  MOVE.L  D0,D1           ;    ➔D 1=D0
06A132  SWAP    D1              ;      Swap D1, D1=xxxxYYYY devient YYYYxxxx
06A134  CMPI.W  #4454,D1        ;      Compare $4454 avec D1 (syncword ?)
06A138  BEQ     6A148           ;      Si C'est égal alors on va en #COPIE_TRAITEMENT_MFM_EN_MEMOIRE
06A13A  LSL.L   #1,D0           ;      Décalage d'1bit vers la gauche de D0
06A13C  DBF     D2,6A130        ;    ➔ D2=D2-1 et tant que D2 n'est pas égale à -1, on boucle #3
06A140  ADDQ.L  #1,D5           ;    D5=D5+1, D5=longueur data traitée en LongWord
06A142  DBF     D3,6A126        ; ➔ D3=D3-1 et tant que D3 n'est pas égale à -1, on boucle #4
06A146  BRA     6A164           ; GoTo RTS
;================================ #COPIE_TRAITEMENT_MFM_EN_MEMOIRE
06A148  SUBI.L  #1A2C,D5        ; D5=D5-$1A2C
06A14E  BMI     6A164           ; Si flag N=1 (résultat inférieur) alors traitement Fini, RTS
06A150  LEA     325DE,A0        ; A0=$325DE (Adresse de destination après traitement)
06A156  MOVE.W  #FFF,D1         ; D1=FFF (le mask)
06A15A  MOVEQ   #1,D0           ; D0=1
06A15C  EOR.W   D0,(A0)+        ; ➔ Traitement des bits vers A0
06A15E  DBF     D1,6A15C        ; ➔ D1=D1-1 et tant que D1 n'est pas égale à -1, on boucle
06A162  RTS                     ; E.T retour maison
06A164  RTS                     ; E.T retour maison
```

>>6A248

```
;================================ #STOP_DF0
06A248  BSET    #7,BFD100       ; Motor Off
06A250  BSET    #3,BFD100       ; Un-select DF0
06A258  MOVEQ   #64,D6          ; D6=64
06A25A  BCLR    #3,BFD100       ; -> Select DF0
06A262  DBF     D6,6A25A        ; <- Petite boucle d'attente utilisant D6
06A266  BSET    #3,BFD100       ; Un-select DF0
06A26E  RTS                     ; E.T Retour Maison
;================================
```

>>462E

```
;================================ #POST_TRAITEMENT_MFM
00462E  MOVEA.L 325E6,A0        ; A0=Long word contenu à l'adresse $325E6
                                ; 1 LongWord plus loin que l'adresse utilisé en COPIE_TRAITEMENT_MFM_EN_MEMOIRE
                                ; La sous-routine #COPIE_TRAITEMENT_MFM_EN_MEMOIRE est donc nécessaire
                                ; avant l'appel de cette routine.

004634  CLR.W   D6              ; -> On efface le Word en D6
004636  MOVEA.L A0,A1           ;   A1=A0
004638  CMPI.W  #190,(A1)       ;   Compare le word à l'adresse A1 avec la valeur $190
00463C  BEQ     4648            ;   Si C'est égal alors on saute en 4648
004640  ADDQ.W  #1,D6           ;   D6=D6+1
004642  ADDA.W  #A,A1           ;   A1=A1+$A
004646  BRA     4638            ;   On repart pour un tour, retour en 4638
;================================
004648  SUB.W   #2,D6           ;   D6=D6-2
00464A  BMI     4696            ;   Si flag N=1 (résultat inférieur) alors
00464E  MOVEM.L D6/A0,-(A7)     ;   -> Sauvegarde des registres dans la Pile
004652  CLR.W   D5              ;      On efface le word de D5
004654  MOVE.W  (A0),D0         ;      -> copie le Word de l'adresse A0 en D0
004656  CMP.W   A(A0),D0        ;        Compare le word en D0 avec le word à l'adresse A0+A
00465A  BLE     4686            ;        Si plus petit ou égal alors on continue en 4686
00465E  MOVE.L  (A0),D0         ;        On stocke le LongWord 4(A0) dans D1
004660  MOVE.L  4(A0),D1        ;        On stocke le LongWord 8(A0) dans D2
004664  MOVE.W  8(A0),D2        ;        et on commence le traitement vers A0
004668  MOVE.L  A(A0),(A0)      ;        Post Traitement vers A0
00466C  MOVE.W  E(A0),4(A0)     ;        Post Traitement vers A0
004672  MOVE.L  12(A0),8(A0)    ;        Post Traitement vers A0
004678  MOVE.L  D0,12(A0)       ;        Post Traitement vers A0
004680  MOVE.W  D2,12(A0)       ;        Post Traitement vers A0
004684  ST      D5              ;        Positionne les 8 premier bits de D5 à 1
004686  ADDA.W  #A,A0           ;         A0=A0+$A
00468A  DBF     D6,4654         ;      <- D6=D6-1 et tant que D6 est diffèrent de -1 on repart en 4654
00468E  MOVEM.L (A7)+,D6/A0     ;    Sinon on restaure la pile
004692  TST.W   D5              ;    Compare le Bit D5 (en word)
004694  BNE     464E            ;   <- Si flag Z=0 (résultat différent de zero), alors on repart en 464E
004696  ADDQ.W  #3,D6           ;     D6=D6+3 en Word
004698  MULU.W  #A,D6           ;     D6=D6*$A en Word
00469C  ADDA.W  D6,A0           ;     A0=A0+D6
00469E  TST.W   (A0)            ;   Test du Word en A0
0046A0  BMI     2E42            ;   Si flag N=1 (Strictement inferieur) alors on part en 2E42 pour un RTS
0046A4  BRA     4634            ; <- Repart pour un tour complet de la sous-routine en 4634
```

```
;================================
002E42  RTS
;================================
```

Ce qui nous donnes finalement :

```
;================================
000CB4  MOVEA.L #3D090,A0       ; A0=3D090, l'adresse mémoire de destination, unique pour les levelS
000CBA  BSR     4066            ; ➔ GoTo #TrackLoader(2)
000CBE  BSR     434A            ; ➔ GoTo #POST_LOAD_LEVEL
000CC2  BSR     462E            ; ➔ GoTo #POST_TRAITEMENT_MFM
000CC6  CLR.W   1E(A5)          ; A5 = 8C98 sur tous les appels, $8C98+$1E=$8CB6, trouvable avec un BS CC6
                                ; Il met donc 00 00 (1 word) en $8CB6
000CCA  RTS                     ; E.T retour maison
;================================
```

Et

```
;================================
06A084  BSR     6A22C           ; GoTo #CONF_DISQUE
06A088  BSR     6A1DE           ; GoTo #MOVE_TO_TRACK_ZERO
06A08C  BCLR    #2,BFD100       ; Sélection face Upper
06A094  BSR     6A0A2           ; GoTo #CONF_DMA_AND_CO
06A098  BSR     6A0F2           ; GoTo #START_TRAITEMENT_MFM
06A08C  BSR     6A248           ; GoTo #STOP_DF0
06A0A0  RTS                     ; E.T retour maison
;================================
```

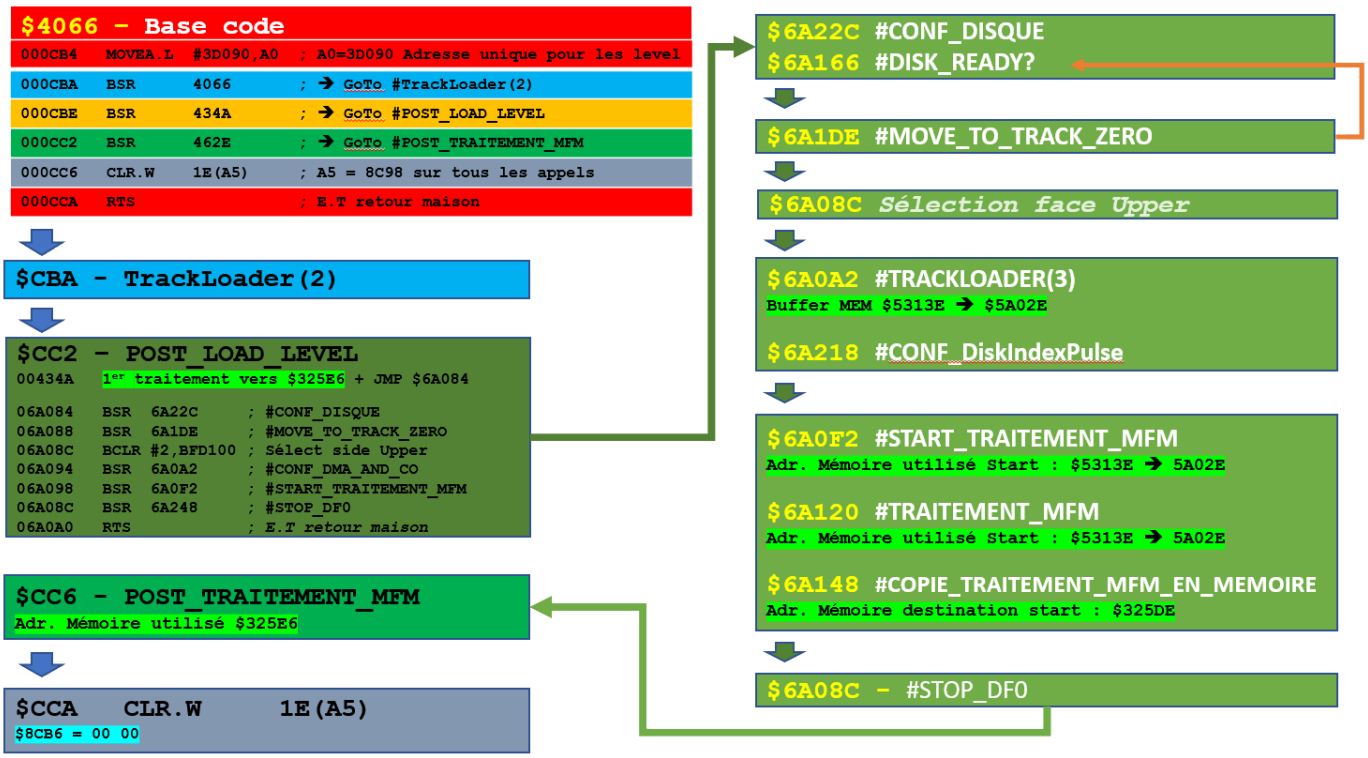
On sait que chaque level est chargé par le `trackloader(2)` à l'adresse *$3D090* et à une longueur de **$1E*$1838**
Ce qui nous donnes la plage mémoire suivante : *$3D090➔$6A720*

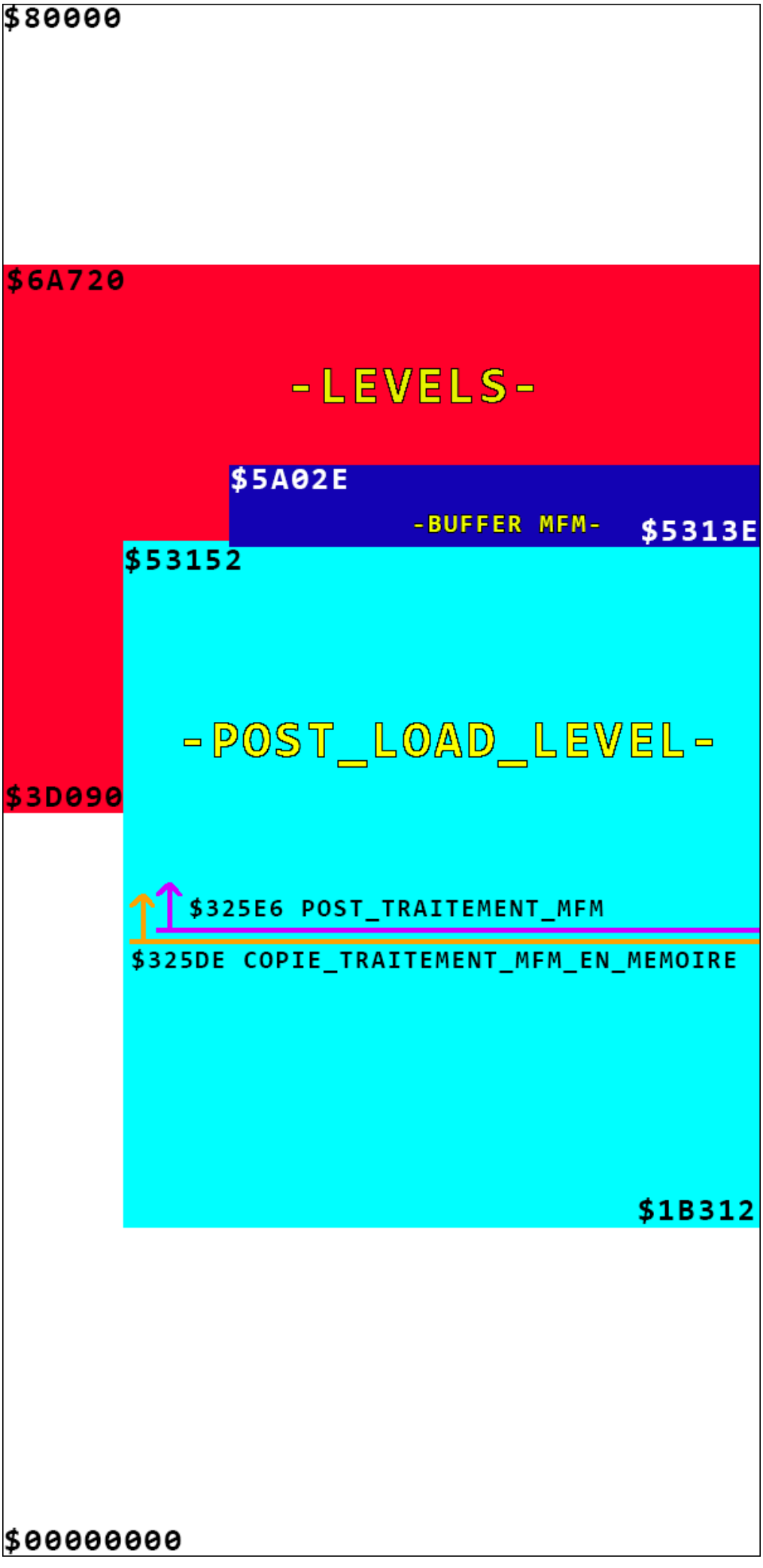
Que le `trackloader(3)` travaille uniquement avec la **T00**
Que la routine `#POST_LOAD_LEVEL` qui calcul l'adresse du **JMP** fonctionne avec l'adresse *$1B312➔$53152*

Que son buffer **MFM** fonctionne avec son buffer en *$5313E* que *DSKLEN* a une valeur **DMA** de *$3778*
Ce qui nous donnes comme plage potentiel pour le Buffer *$3778*2=$6EF0*
Donc un buffer **MFM** dispo de *$5313E à $5313E+$6EF0*, donc *$5313E➔$5A02E*

Que le **traitement MFM** qui s'ensuit utilise l'adresse mémoire *$325DE* comme source et destination.
Que le **post-traitement** qui s'ensuit utilise l'adresse mémoire *$325E6* comme source et destination.

Il n'y a pas de 'check' proprement dis du contenu de la **T00** mais une lecture et décodage qui va influencer le contenue en mémoire **$5313E**
Est-ce que c'est une clé de décodage ? Possible, regardons à l'esprit le fonctionnement global de ce 'Check T00'




$80000
$6A720
-LEVELS-
$5A02E
-BUFFER MFM-
$5313E
$53152
-POST_LOAD_LEVEL-
$3D090
$325E6 POST_TRAITEMENT_MFM
$325DE COPIE_TRAITEMENT_MFM_EN_MEMOIRE
$1B312
$00000000

Part 15 Protection Track 00 : Création Dump FULL

Afin de travailler plus rapidement nous allons utiliser l'option **SA** de l'AR (sous *WinUae* utiliser l'option SafeState dans le menu Miscellaneous)
Insérez-le Disk Original 1 de Blood Money dans le lecteur **et rebooter votre Amiga**
Passer les diverses **étapes** pour **arriver au menu du jeu.**
Entrer dans l'AR, insérer une nouvelle disquette vierge dans le lecteur.
Taper :
#format, Permet de formater une disquette au format AmigaDOS
`FORMAT WORK`
Ready to format disk in drive DF0: (y/n)? y
Disk ok

Il est temps de sauver tout ça.
#SA, alias SaveAllMemory. Permet de sauver toute la zone mémoire du programme en cours.
Taper : `SA FULL`

```
format WORK
Ready to format disk in drive DF0: (y/n)?
y
Disk ok
sa FULL
WARNING: No load without cartridge possible
Try to fill $900 bytes in chipmem with zero (be careful)
Disk ok
```

Comme précédemment, je pars du principe que vous avez un lecteur externe.
Insérez notre disquette `WORK` dans le lecteur **DF1** et le disk **Original N°2 de BloodMoney** dans **DF0**
Effectuer un reboot de votre Amiga et entrer dans l'AR.
Taper : `LA 1:FULL`

```
la 1:FULL
Memoryconfiguration: 0.5 MByte ChipMem,  no FastMem,  no ExternalMem
Active drives: DF0: DF1:
Disk ok
```

Après une phase de chargement et de décompression, **taper** `X`
Vous devriez vous retrouver au menu du jeu **BloodMoney**.

On continue notre phase de compréhension abordé dans le **chapitre 14, la protection Track Zero.**
Intéressons-nous donc à la phase `#POST_LOAD_LEVEL`, étape qui suit juste après la phase de chargement du level avec le **trackloader(2)**
Poser un *breakpoint* à la fin de cette routine, **taper :** `BS 437C` et on retourne au code avec la commande `X`
Appuyer sur F1 pour commencer une partie d'un joueur, sélectionner le 1er niveau puis **appuyer sur Fire**

Le chargement du 1er Level s'effectue et notre *BreakPoint* **$437C** est atteints, **on entre automatiquement dans l'AR.**
Taper : `R`

```
Breakpoint raised at address: 0000437C

r
D0=0000001F 0000FFFF 00000000 000083AF  100083AF 0000FFFF 0000FCFE 0000FFFF
A0=000633F2 00053152 0006A084 00004B44  0006354E 00008C98 00DFF000 000003F4
PC = 0000437C USP = 000018B2 SR = 2104 T=0 S=1 I=001 X=0 N=0 Z=1 V=0 C=0
```

Comme prévue, on retrouve nos deux valeurs en **A1** et **A2** déjà vue en chapitre 14
Jetons un coup d'œil au traitement effectué par cette routine, **tapez :** `M 1B312` et `D 1B312` puis descendez un peu.
Bon…ça ne ressemble pas à du code 68000.

```
m 1B312
:01B312 FF AB 9F 1F FF AB 9F 1F FF AB 9F 1F FF AB 9F 1F ................
:01B322 00 00 1F 00 00 00 1F 00 00 00 1F 00 F1 FF FF FF ................
:01B332 F1 FF FF FF F1 F7 77 73 F1 F7 77 73 F1 F7 77 73 ......ws..ws..ws
:01B342 F1 F7 77 73 F1 F7 77 73 F1 F7 77 73 E1 F7 77 73 ..ws..ws..ws..ws

d 1B312
~01B312 LINEF
~01B314 SUB.B   D7,(A7)+
~01B316 LINEF
~01B318 SUB.B   D7,(A7)+
~01B31A LINEF
```

Et quand est-il des data de la routine **#COPIE_TRAITEMENT_MFM_EN_MEMOIRE** qui utilise l'adresse mémoire ***$325DE***
Tapez : `M 325DE` et `D 325DE`

```
m 325DE
:0325DE 00 02 26 2F 00 02 27 97 00 02 2A ED 00 02 46 35 ..&/..'...*...F5
:0325EE 00 02 2A 07 00 02 2A CF 00 02 29 C7 00 02 4D A5 ..*...*...)...M.

d 325DE
~0325DE ORI.B   #2F,D2
~0325E2 ORI.B   #97,D2
~0325E6 ORI.B   #ED,D2
~0325EA ORI.B   #35,D2
~0325EE ORI.B   #7,D2
~0325F2 ORI.B   #CF,D2
```

Tapez : `BS 6A162` et `X`
Rapidement notre *breakpoint* ***$6A162*** est atteint et on entre automatiquement dans l'AR, **tapez :** `M 325DE` et `D 325DE`

```
Breakpoint raised at address: 0006A162

m 325DE
:0325DE 00 03 26 2E 00 03 27 96 00 03 2A EC 00 03 46 34 ..&...'...*...F4
:0325EE 00 03 2A 06 00 03 2A CE 00 03 29 C6 00 03 4D A4 ..*...*...)...M.

d 325DE
~0325DE ORI.B   #2E,D3
~0325E2 ORI.B   #96,D3
~0325E6 ORI.B   #EC,D3
~0325EA ORI.B   #34,D3
```

Les données ne sont plus exactement les mêmes et c'est normal, notre routine est passé par là.
Mais ce n'est toujours pas du code 68000

```
Avant  0325DE 00 02 26 2F 00 02 27 97 00 02 2A ED 00 02 46 35 ..&/..'...*...F5
Après  0325DE 00 03 26 2E 00 03 27 96 00 03 2A EC 00 03 46 34 ..&...'...*...F4
```

Vous pouvez **tapez :** `X` pour lancer le level puis re entrer dans l'AR et vérifier encore les adresses ***$325DE*** et ***$1B312***
Vous verrez que les DATA n'ont pas changé.

Le 1er traitement : #POST_LOAD_LEVEL
Utilise les données du LEVEL en *$3D090* pour en sortir quelque chose qu'il écrit en *$1B312*

Le 2em traitement qui englobe les routines : #START_TRAITEMENT_MFM, #TRAITEMENT_MFM, COPIE_TRAITEMENT_MFM_EN_MEMOIRE
Utilise les données chargées par la routine *trackloader(3)*, celle de la T00 en ***$531E*** pour en sortir quelque chose qu'il écrit en ***$325DE***

Et les deux 'output' ne sont pas du code 68000, du moins, pas en l'état.

On jette un coup d'œil en mémoire si une autre partie du code utilise ces adresses.
Tapez : `FA 325DE`

```
0021FC  MOVEA.L         325DE,A2        ; Code
06A150  LEA             325DE,A0        ; #COPIE_TRAITEMENT_MFM_EN_MEMOIRE
078A78  CMPI.W          #7,-2000(A0)    ; Etrange...
```

Puis **Tapez :** `FA 1B312`

```
004358  LEA             1B312,A1        ; #POST_LOAD_LEVEL
004BAE  LEA             1B312,A0        ; Code
```

On va poser deux *BreakPoints* afin de voir si on passe par ces adresses avant le lancement du LEVEL.
Mais avant, on va désactiver les deux routines de **Checksum** que l'on a vu en **Part 13 Piste/code des HighScores**
Taper :

```
A 2C16
^02C16  NOP                     ; GoTo #checksum enlevé
^02C18  <RETURN>
```

Puis

```
A 44DA
^044DA  NOP                     ; Sup. de la boucle dans la sous-routine du #checksum
^024DC  <RETURN>
```

Et on pose nos deux *BreakPoints,* **taper :** `BS 4BAE` et `BS 21FC` et on retourne au code, **taper :** `X`

On lance bien sur une partie sur le Level1
Le **LEVEL1** se lance mais très vite le *BreakPoints* **$21FC** est atteint.
On jette coup d'œil au code en cours, **taper :** `D` et descendez dans le code.

```
Breakpoint raised at address: 000021FC
d
~0021FC MOVEA.L 000325DE,A2
~002202 MOVE.L  4(A2,D0.W),D1
~002206 MOVE.L  0(A2,D0.W),D0
~00220A EXT.L   D4
~00220C ADD.L   D4,D0
~00220E ADD.L   D4,D1
~002210 RTS
;=======================================
~002212 MOVE.W  1C(A5),D0
```

Après un traitement qui positionne **D0** et **D1** avec l'utilisation du contenu en **$325DE**, un **RTS** est effectué
On regarde vers où l'on retournera, stocker dans la pile.
Taper : `R` puis `M 3F4` puisqu'**A7** pointe vers **3F4**
on regarde le code qui sera exécuté au retour avec la commande : `D 2052`

```
r
D0=00000030 00000B00 00000054 00000542  00000000 00000000 00008000 0000000F
A0=00075F46 00006BA0 00008D06 0000748A  00008C4C 00008C98 00DFF000 000003F4
PC = 000021FC USP = 000018B2 SR = 2100 T=0 S=1 I=001 X=0 N=0 Z=0 V=0 C=0
m 3F4
:0003F4 00 00 20 52 00 00 1F 4E 00 00 06 16 41 F9 00 00 .. R...N....A...
d 2052
~002052 ADD.L   D5,D0
~002054 ADD.L   D5,D1
~002056 MOVE.L  1A(A1),D4
~00205A BTST    #F,D4
~00205E BEQ     0000206A
~002062 BTST    #10,D4
~002066 BNE     00002092
~00206A MOVE.W  42(A5),D4
~00206E MOVE.L  D1,50(A6)
~002072 MOVE.L  D0,4C(A6)
~002076 MOVE.L  A0,48(A6)
~00207A MOVE.L  A0,54(A6)
~00207E MOVE.W  D3,58(A6)
~002082 BSR     00002E3A
~002086 ADDA.W  #2260,A0
~00208A ADD.L   D2,D0
~00208C DBF     D4,0000206E
~002090 RTS
```

Et on suit le tout.

```
d 2092
~002092 BCLR    #10,D4
~002096 MOVE.L  D4,1A(A1)
~00209A MOVE.W  42(A5),D4
~00209E SUBQ.W  #1,D4
~0020A0 MOVE.L  D1,50(A6)
~0020A4 MOVE.L  D1,4C(A6)
~0020A8 MOVE.L  A0,48(A6)
~0020AC MOVE.L  A0,54(A6)
~0020B0 MOVE.W  D3,58(A6)
~0020B4 BSR     00002E3A
~0020B8 ADDA.W  #2260,A0
~0020BC MOVE.W  4A(A5),D2
~0020C0 ORI.W   #B0A,D2
~0020C4 MOVE.W  D2,40(A6)
~0020C8 MOVE.L  D1,50(A6)
~0020CC MOVE.L  A0,48(A6)
~0020D0 MOVE.L  A0,54(A6)
~0020D4 MOVE.W  D3,58(A6)
~0020D8 BSR     00002E3A
~0020DC ADDA.W  #2260,A0
~0020E0 DBF     D4,000020C8
~0020E4 RTS
;=======================================
```

```
d 00002E3A
^002E3A BTST    #E,2(A6)

^002E40 BNE     00002E3A
^002E42 RTS
;=====================================
```

Le *BreakPoint* Un ***$21FC*** a donc été atteint, on peut en déduire assez facilement que la routine `#COPIE_TRAITEMENT_MFM_EN_MEMOIRE` est donc nécessaire.

On va la désactiver pour voir comment se comporte le jeu.
Rebooter votre amiga et recharger comme précédemment notre Full Dump en **tapant :** `LA 1:FULL`

Taper : `BS 437C` puis `X`
On lance bien sur une partie sur le Level1
De retour sur l'AR après notre *BreakPoint* atteint, on pose nos **NOP** à l'adresse ***$2C16*** et ***$44DA***
Et on désactive toute la procédure de **T00** en posant un **RTS** en **$6A084** et on retourne au code avec la commande `X`

```
la 1:FULL
Memoryconfiguration: 0.5 MByte ChipMem,  no FastMem,  no ExternalMem
Active drives: DF0: DF1:
Disk ok

bs 437C
Breakpoint inserted
Ready.
x
No known virus in memory!
Ready.
Breakpoint raised at address: 0000437C

a 6A084
^06A084 RTS
^06A086
a 2C16
^002C16 nop
^002C18
a 44DA
^0044DA nop
^0044DC
x
```

Ce qui nous un joli crash system assez rapidement.
Voyons voir… on sait que la routine `#POST_LOAD_LEVEL` effectue un premier traitement de donnée vers **$325E6** avant de faire son **JUMP (A2)**
On sait aussi que ce traitement 'continue' (car même adresse utilisée) en **$6A148** avec la routine `#COPIE_TRAITEMENT_MFM_EN_MEMOIRE`
Par contre, les 2 premiers opérations ne nous intéressent pas car utilisation de **D5** qui est lui utilisé dans la routine juste au-dessus qui, ne nous intéresse pas au vue de la zone mémoire avec laquelle elle travaille.

```
06A148  SUBI.L  #1A2C,D5        ; D5=D5-$1A2C
06A14E  BMI     6A164           ; Si flag N=1 (résultat inférieur) alors traitement Fini, RTS
```

Donc au lieu de faire un **RTS** directement en **$6A084**, on va faire un **branchement** faire cette routine: `COPIE_TRAITEMENT_MFM_EN_MEMOIRE`
Routine qui va continuer son 'job' avec notre zone mémoire concernée, à savoir : ***$325DE***

Et comme cette routine finie par un **RTS**, on n'a même pas besoin d'en ajouter un, il nous suffit juste de faire un **BRA 6A150** en ***$6A084*** comme ça, on effectue tout le traitement de la zone ***$325DE*** et on bypass toutes les routines concernant la **T00**

Rebooter votre amiga et recharger comme précédemment notre Full Dump en **tapant :** `LA 1:FULL`

Taper : `BS 437C` puis `X`
On lance bien sur une partie sur le Level1
De retour sur l'AR après notre *BreakPoint* atteint, on pose nos **NOP** à l'adresse ***$2C16*** et ***$44DA***
Et on remplace le `BSR 6A22C` en **$6A084** par un `BRA 6A150`
Taper : `A 6A084`

```
^06A084 BRA     6A150           ; On branche directement sur la suite du traitement des données en $325DE
^06A088 <RETURN>
```

Sans oublier de retourner au code avec la commande `X` (et on regarde/écoute notre lecteur de disquette)
Aucun accès disque (ce qui est normal vue que l'on a désactiver toutes les routines de **T00**) et le Level c'est lancé et semble fonctionner.
Néanmoins, une fois de retour au MENU, (après avoir perdu toutes ses vies ou avoir appuyer sur la touche **ESC)**
Impossible de lancer de nouveau une partie. Après un court accès disque, nous sommes bloqués sur l'écran **PRESS FIRE TO START**
Entrer dans l'AR et Taper : `D`

Etrange d'ailleurs ce petit accès disque, normalement on devrait avoir un accès de 4 pistes pour arriver au **Menu choix Level**

Nous sommes bloqués dans cette routine

```
d
~0042DA BEQ     000042D0

d 42DA-12
~0042C8 RTS
;-------------------------------------
~0042CA MOVE.B  00BFDD00,D6
~0042D0 MOVE.B  00BFDD00,D6
~0042D6 BTST    #4,D6
~0042DA BEQ     000042D0
~0042DC RTS
;-------------------------------------
```

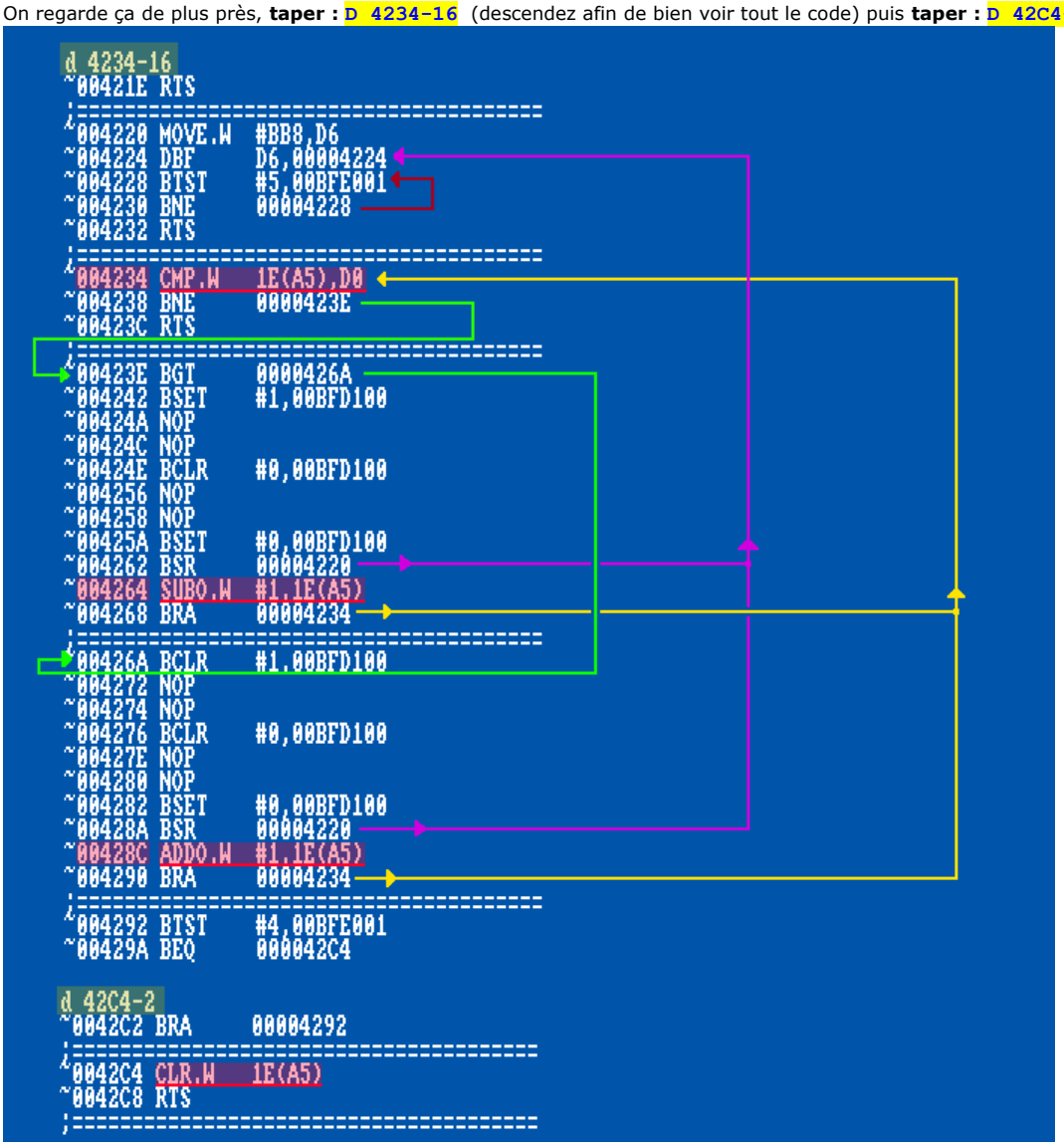
Hummm, revenons sur notre bout de code spécifique au chargement des **LEVEL**
Il y a une seule inconnue dans toutes les opérations que l'on a vue. A quoi sert l'adresse ***$8CB6*** et donc le **CLR.W 1E(A5)** en ***$CC6***

```
;---------------------------------
000CB4  MOVEA.L #3D090,A0       ; A0=3D090, l'adresse mémoire de destination, unique pour les levelS
000CBA  BSR     4066            ; ➔ GoTo #TrackLoader(2)
000CBE  BSR     434A            ; ➔ GoTo #POST_LOAD_LEVEL
000CC2  BSR     462E            ; ➔ GoTo #POST_TRAITEMENT_MFM
000CC6  CLR.W   1E(A5)          ; A5 = 8C98 sur tous les appels, $8C98+$1E=$8CB6, trouvable avec un BS CC6
                                ; Il met donc 00 00 (1 word) en $8CB6
000CCA  RTS                     ; E.T retour maison
;---------------------------------
```

On va chercher en mémoire qui se sert de cette adresse, **taper : `FA 8C98+1E`**
Nous avons logiquement plusieurs résultats à cette requête.

```
000CC6  CLR.W   1E(A5)          ; Notre CLR présent dans la routine de chargement des LEVEL
004234  CMP.W   1E(A5)          ; à définir
004264  SUBQ.W  #1,1E(A5)       ; à définir
00428C  ADDQ.W  #1,1E(A5)       ; à définir
0042C4  CLR.W   1E(A5)          ; à définir
```

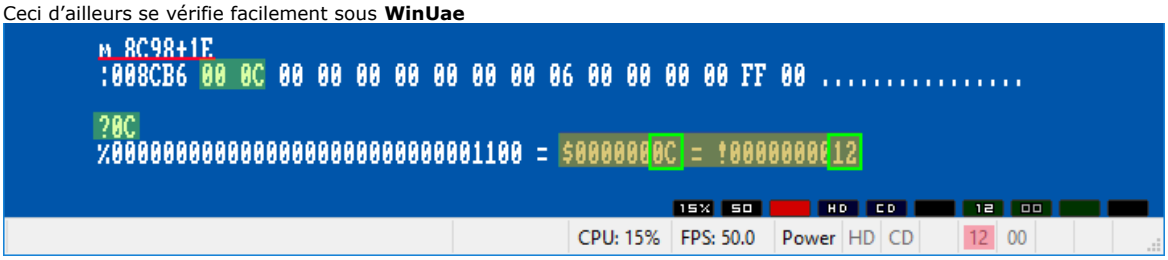
On regarde ça de plus près, **taper : `D 4234-16`** (descendez afin de bien voir tout le code) puis **taper : `D 42C4-2`**



On peut retrouver notre commande **CLR.W 1E(A5)** tout le temps juste après une opération sur le **Bit 0** du registre **BFD100**
A savoir le **PR-B, bit0=STEP** (déplace la tête d'une track dans une direction), voir chapitre **LE FORMAT MFM** au début du tuto.
CLR.W 1E(A5) suivant la portion de code utilisé :

- Soit il ajoute 1 à notre adresse **1E(A5)** comme par exemple en ***$428C*** **: ADDQ.W 1E(A5)**
- Soit il retire 1 à notre adresse **1E(A5)**, exemple en ***$4264*** : **SUBQ.W 1E(A5)**
- Soit il la remet à Zero comme en **$CC6** ou en **$42C4** : **CLR.W 1E(A5)**

Ceci d'ailleurs se vérifie facilement sous **WinUae**



Donc l'adresse $8C98+$1E = Position de la Tête du lecteur en Piste.
Alors effectivement, vue que l'on a supprimer les routines en **T00**, la tête du floppy n'a pas bougé.
Le **CLR.W 1E(A5)** en **$CC6** n'a plus lieu d'être.

On va tester ça.
Rebooter votre amiga et recharger comme précédemment notre Full Dump en **tapant : LA 1:FULL**

Taper : BS 437C puis **X**
On lance bien sur une partie sur le Level1
De retour sur l'AR après notre *BreakPoint* atteint, on pose nos **NOP** à l'adresse ***$2C16*** et ***$44DA***
Et on remplace le **BSR 6A22C** en **$6A084** par un **BRA 6A150**
Sans oublier de supprimer notre **CLR.W** en **$CC6**

Taper : A CC6

```
^000CC6 NOP                    ; Désactivation de la MAJ du compteur de piste vue que l'on n'a pas bougé.
^000CC8 NOP                    ; Deux fois un NOP pour retomber sur le code original.
^000CCA <RETURN>
```

Et bien sûr **X** pour retourner au jeu, perdre toutes ses vies puis retourner au MENU et **relancer une partie avec F1**
et…BINGO, c'était bien ça, le LEVEL 1 c'est bien relancer.

Humm, il serait intéressant de regarde si un tel code existe aussi sur la 1ère phase, la 1er disquette original
Et vue que la **T00.1** est indique sur le disque1 et le disque2 (voir **Part 2 Analyse de l'image IPF**), il est fort probable qu'il y ait une routine de check.

Part 16 Protection Track 00 sur le Disque1, nouveau trackloader(4)

Nous avons vu en **Part 06 Let's Rock'n roll** que tous les appels disque sur la 1er phase, le disque 1, passait par le trackloader en **$7C41C**
Nous avions listé tous les appels vers celui-ci.

```
fa 7C41C
Search from: 000000 to: C80000
07A458 BSR      0007C41C
07A4D8 BSR      0007C41C
07A4EE BSR      0007C41C
07A4FC BSR      0007C41C
07A600 BSR      0007C41C
07A60E BSR      0007C41C
07A70C BSR      0007C41C
07A71A BSR      0007C41C
07C442 BNE      0007C41C     >En plein dans la routine du trackloader
07C44E BNE      0007C41C     >En plein dans la routine du trackloader
Searched up to adr: C80000
Ready.
```

On va jeter un coup d'œil au code qui suivant chaque appel pour cela, on va utilise notre petite routine préalablement : **code_rip**
Si une telle routine existe en mémoire, elle doit forcément tester si la tête du lecteur est sur **T00,** donc la commande : `BTST #4,BFE001`
En ***opcode*** : `08 39 00 04 00 BF E0 01`

Insérer dans un premier temps notre **disquette 1 de backup** préalablement crée et modifié dans le lecteur.
Rebooter votre Amiga, l'amiga boot sur cette disquette et notre **écran rouge apparait.**
Entrer dans l'AR, taper : `BS 7DC4C`
Retirez la disquette de backup et insérez la disquette original 1 dans le lecteur, **taper :** `X`
Après un rapide accès disque ,notre *BreakPoint* est atteint

Taper : `BDA` puis `F 08 39 00 04 00 BF E0 01`
L'AR nous donnes comme résultat **$7C540** et **$7DD8C**

On va poser un *BreakPoint* après chaque appel que l'on a noté, **taper :** `BS 7A458+4` puis `X`
Ps : +4 car c'est la longueur d'un **BSR 7C41C** et on veut un arrêt APRES pas avant l'appel au *trackloader* (que les données soient chargées)
Taper : `BDA` puis `F 08 39 00 04 00 BF E0 01`
L'AR nous donnes toujours le même résultat, **$7C540** et **$7DD8C**

On continue, **taper :** `BS 7A4D8+4` puis `X`
et on recommence avec le **BS** une fois le *BreakPoint* atteint, etc, etc

On obtient le résultat suivant :
Pour les *BreakPoints* après : **$7A458, $7A4D8, $7A4EE, $7A4FC, $7A600, $7A60E** ➔ **7C540** & **7DD8C**
Pour les *BreakPoints* après : **$7A70C** & **$7A71A** ➔ **7C540** & **7DD8C** & **$4292**

On va regarder ces 3 adresses point par point mais on peut déjà dire qu'une routine spécifique est chargé après l'appel en **$7A70C**
chargement qui travaille avec les pistes 14 ➔ 22

Insérer notre **disquette 1 de backup** préalablement crée et modifiée dans le lecteur.
Rebooter votre Amiga, l'amiga boot sur cette disquette et notre **écran rouge apparait, entrer dans l'AR**
Taper : `BS 7DC4C`
Retirez la disquette de backup et insérez la disquette original 1 dans le lecteur, **taper :** `X`

Après un rapide accès disque ,notre *BreakPoint* est atteint, on entre automatiquement dans l'AR.
Taper : `BDA` puis `BS 7C540` et `BS 7DD8C` et `BS 4292` puis `X`
Assez rapidement notre *BreakPoints* **$7C540** est atteint, on entre automatiquement dans l'AR

Taper : `D 7C540-2` et on désassemble tous les appels `D 7C4D8` `D 7C540-2`

```
d 7C540-2
~07C53E BRA     0007C4EC
;-------------------------------------
;-------------------------------------
~07C540 BTST    #4,00BFE001 ; Disque sur Track Zero ?
~07C548 BEQ     0007C56A    ; Oui, on branche en 7C56A
~07C54C BSET    #1,00BFD100 ; Sinon on positionne le déplacement vers l'ext.
~07C554 BCLR    #0,00BFD100 ; Et on déplace la tête
~07C55C BSET    #0,00BFD100 ; Doit changer d'état pour activer le déplacement
~07C564 BSR     0007C4D8    ; On branche en 7C4D8
~07C568 BRA     0007C540    ; C'est repartie pour un tour
;-------------------------------------

~07C56A CLR.B   0007CA3E    ; On efface le byte en 7C13E (compteur Piste ?)
~07C570 RTS                 ;
;-------------------------------------


d 7C4D8
~07C4D8 MOVE.W  #BB8,D6     ; D6=BB8, (petite boucle de pause)
~07C4DC DBF     D6,0007C4DC ; D6=D6-1 et tant que D6  n'est pas égale à -1, on boucle
~07C4E0 BTST    #5,00BFE001 ; Le disque est prêt pour une opération ?
~07C4E8 BNE     0007C4E0    ; Nop, on boucle pour attendre
~07C4EA RTS                 ; Retour
;-------------------------------------
```

Voilà une jolie routine qui va déplacer la tête du lecteur jusqu'à la **TK0** avec semble t'il un compteur de piste en **$7CA3E**

On va poser un *BreakPoint* en **$7C570** (une fois que la **T00** est atteinte) pour voir ce qu'il va faire ensuite :
Taper : `BS 7C570` puis `X`
La tête du lecteur se déplace rapidement et on atteint la piste0, notre *BreakPoint* est alors atteint, on entre automatiquement dans l'AR.

#ST, alias TRACE. Permet de tracer le code avec les sous-routine, il est possible de lui indiquer le nombre de 'pas'
Taper : `ST`
Nous sommes en *$7A446*
Tapez : `D 7A446` (et descendez de quelques lignes)

```
d 7A446
~07A446 BSET    #2,00BFD100
~07A44E MOVEA.L #58E44,A0
~07A454 MOVEQ   #1E,D0
~07A456 MOVEQ   #14,D1
~07A458 BSR     0007C41C
~07A45C LEA     7C7A6(PC),A0
~07A460 MOVE.L  #58E44,D0
```

Le code se poursuivra avec la sélection d'une side et le positionnement de **A0, D0 et D1** avant de faire appel au *Trackloader(1)*
Donc on à clairement trouver une sous-routine de **#GoTo_TrackZero** mais elle n'est pas utilisée pour une protection mais pour un départ de chargement avant l'appel au *trackloader(1).*

En clair, ce n'est pas ce que l'on recherche MAIS cela nous a permis de connaitre l'adresse mémoire du compteur de piste et qui sait ? peut être cette portion de code sera utilisé plus tard dans un autre but !

Comme déjà expliqué, il peut arriver qu'une fois un *BreakPoint* atteint, il ne soit 'plus vue' par l'AR
Dans le doute, on efface tous nos *BreakPoints* et on les remet de nouveau.
Taper : `BDA` puis `BS 7C540` et `BS 7DD8C` et `BS 4292` puis `X`

Après plusieurs chargements où l'on passe le ScreenShot et l'animation Psysnosis, notre *BreakPoint* ***$7c540*** est de nouveau atteint, on Entre automatiquement dans l'AR.

Même combat que tout à l'heure, on pose un *BreakPoint* en sortie de cette sous-routine.

Taper : `BS 7C570` puis `X`
La tête du lecteur se déplace rapidement et on atteint la piste0, notre *BreakPoint* est alors atteint, on entre automatiquement dans l'AR.

Taper : `ST`
Nous sommes en *$7C348*
Tapez : `D 7C348-9` (et descendez de quelques lignes), on suit le code, **taper :** `D 7C584-2` puis `7C4D8-2`

```
d 7C348-9
~07C33E RTS
;-------------------------------------
~07C340 BSR     0007C586    ;
~07C344 BSR     0007C540    ; Notre BreakPoint
~07C348 BCLR    #2,00BFD100 ; CIA-B bit2 = Select Side upper
~07C350 BSR     0007C35A    ; à Définir
~07C354 BSR     0007C3AA    ; à définir
~07C358 RTS
;-------------------------------------

d 7C586-2
~07C584 RTS
;-------------------------------------
~07C586 BCLR    #7,00BFD100 ; CIA-B bit 7 = Motor On
~07C58E MOVEQ   #64,D0      ; D0=64, pour petite boucle d'attente
~07C590 BCLR    #3,00BFD100 ; CIA-B bit 3= Sel DF0
~07C598 DBF     D0,0007C590 ; D0=D0-1, tant que D0 n'est pas égale à -1 on boucle
~07C59C BSR     0007C4D8    ; GoTo #DisqueReady?
~07C5A0 RTS                 ; E.T Retour maison
;-------------------------------------


d   0007C4D8-2
~07C4D6 RTS
;-------------------------------------
~07C4D8 MOVE.W  #BB8,D6     ; D6=BB8
~07C4DC DBF     D6,0007C4DC ; D6=D6-1, tant que D6 n'est pas égale à -1, on boucle
~07C4E0 BTST    #5,00BFE001 ; CIA-A, Bit5 = Disque Ready ?
~07C4E8 BNE     0007C4E0    ; Non ? On boucle pour attendre
~07C4EA RTS                 ; E.T retour maison
;-------------------------------------
```

Choses très intéressantes ici car différente de ce que l'on a pu voir avant.
Deux **BSR** sont effectués juste après notre *BreakPoint* (juste après la sélection de Side)
On jette un coup d'œil à ces deux adresses et on suit le code.

(Petit rappel)

```
;================================
07C340  BSR     7C586           ; ➔ GoTo #Conf_Disque
07C344  BSR     7C540           ; ➔ GoTo #TrackZero
07C348  BCLR    #2,BFD100       ; Sélection Side Upper
07C350  BSR     7C35A           ; ➔ à définir
07C354  BSR     7C3AA           ; ➔ à définir
07C358  RTS                     ; E.T retour maison
;================================
```

Tapez : `D 7C35A` et on suit le code

>>7C35A

```
;================================ #TRACKLOADER(0)
07C35A  MOVE.W  #400,9E(A6)     ; ➔    ADKCON, disable WORDSYNC
07C360  MOVE.W  #8210,96(A6)    ;      DMACON, clear DSKEN (disk DMA), DMAEN (disable all DMA)
07C366  MOVE.W  #400,20(A6)     ;      DSKPTH // adr. MFM buffer=$400
07C36E  CLR.W   24(A6)          ;      DSKLEN,  DMA data Length clear
07C372  BSR     7C572           ;      GoTo #CONF_DiskIndexPulse
07C376  MOVE.W  #4000,24(A6)    ;      DSKLEN, Disk DMA=OFF
07C37C  MOVE.W  #B778,24(A6)    ;      DSKLEN, Disk DMA=enable, Length DMA data=$3778
07C382  MOVE.W  #B778,24(A6)    ;      Encore une fois pour démarrer le transfert DMA
07C388  MOVE.W  #30D40,D1       ;      D1=30D40
07C38E  MOVE.W  1E(A6),D0       ; ➔    D0=DFF01E (INTREQR)
007C92  SUBQ.L  #1,D1           ;      D1=D1-1
007C94  BEQ     7C35A           ; ➔ Si flag Z=1, pas bon, on repart pour un tour #1
007C96  BTST    #1,D0           ; Test Bit 1 de INTREQR (Disk Block Finished Interupt)
007C9A  BEQ     7C38E           ; ➔ si flag Z=1 on repart pour un tour #2
007C9C  MOVE.W  #2,9C(A6)       ; INTREQ, Disk Block Finished Interuppt
07C3A2  MOVE.W  #8400,9E(A6)    ; ADKCON, Enable SyncWord
07C3A8  RTS                     ; E.T retour maison
```

Bon… Nous sommes dans le même type de code vu en **Part 14 Protection Track Zero sur le Disque2, nouveau trackloader(3)**
On continue notre investigation en suivant ce code.

>>7C572

```
;================================ #CONF_DiskIndexPulse
07C572  MOVE.B  BFDD00,D6       ; D6=BFDD00=Conf ICR   CIA-B
07C578  MOVE.B  BFDD00,D6       ; ➔D6=BFDD00=Conf ICR  CIA-B
07C57E  BTST    #4,D6           ;   Test Bit 4 de BFDD00 (Disk Index Pulse)
07C582  BEQ     7C578           ; ➔ Si flag Z=1 on repart pour un tour
07C584  RTS                     ; Sinon, retour maison
```

>>7C3AA

```
;================================ #START_TRAITEMENT_MFM
07C3AA  MOVEA.L #400,A0         ; A0=400
07C3B0  MOVE.L  #3778,D3        ; D3=3778
07C3B6  MOVE.L  (A0),D3         ; ➔ Copie le LongWord de l'adresse A0 vers D3
07C3B8  ADDQ.L  #2,A0           ;    A0=A0+2
07C3BA  MOVE.L  #F,D2           ;    D2=F
07C3C0  MOVE.L  D0,D1           ;    ➔ D1=D0
07C3C2  SWAP    D1              ;       Swap D1, D1=xxxxYYYY devient YYYYxxxx
07C3C4  CMPI.W  #4454,D1        ;       Compare $4454 avec D1 (syncword ?)
07C3C8  BEQ     7C3D8           ;       Si C'est égal alors, GoTo #TRAITEMENT_MFM
07C3CA  LSL.L   #1,D0           ;       Décalage d'1bit vers la gauche de D0, ex sur 1 word : 0110 devient 1100
07C3CC  DBF     D2,7C3C0        ;    ➔ D2=D2-1 et tant que D2 n'est pas égale à -1, on boucle #1
07C3D0  DBF     D3,7C3B6        ; ➔ D3=D3-1 et tant que D3 n'est pas égale à -1, on boucle #2
07C3D4  BRA     7C414           ; Fin de traitement suivi d'un RTS
;================================ #TRAITEMENT_MFM
07C3D8  MOVE.L  #0,D5           ; D5=0
07C3DE  MOVE.L  (A0),D0         ; ➔ Copie le LongWord de l'adresse A0 vers D0
07C3E0  ADDQ.L  #2,A0           ;    A0=A0+2
07C3E2  MOVE.L  #F,D2           ;    D2=F
07C3E8  MOVE.L  D0,D1           ;    ➔D 1=D0
07C3EA  SWAP    D1              ;      Swap D1, D1=xxxxYYYY devient YYYYxxxx
07C3EC  CMPI.W  #4454,D1        ;      Compare $4454 avec D1 (syncword ?)
07C3F0  BEQ     7C400           ;      Si C'est égal alors on va en #NEXT_MFM
07C3F2  LSL.L   #1,D0           ;      Décalage d'1bit vers la gauche de D0
07C3F4  DBF     D2,7C3E8        ;    ➔ D2=D2-1 et tant que D2 n'est pas égale à -1, on boucle #3
07C3F8  ADDQ.L  #1,D5           ;    D5=D5+1, D5=longueur data traitée en LongWord
07C3FA  DBF     D3,7C3DE        ; ➔ D3=D3-1 et tant que D3 n'est pas égale à -1, on boucle #4
07C3FE  BRA     7C414           ; GoTo RTS
;================================ #NEXT_MFM
07C400  SUBI.L  #1A2C,D5        ; D5=D5-$1A2C
07C406  BMI     7C414           ; Si flag N=1 (résultat inférieur) alors traitement Fini, RTS
07C408  MOVE.L  #400,7A72C      ; Met $400 à l'adresse $7A72C
07C412  RTS                     ; E.T retour maison
07C414  CLR.L   7A72C           ; LongWord à l'adresse $7A72C mis à Zero
07C41A  RTS                     ; E.T retour maison
```

On regarde où va continuer le code ensuite, **taper :** `BS 7C358` puis `X`

Assez vite notre *BreakPoint* ***$7C358*** est alors atteint, on entre automatiquement dans l'AR.

Taper : `ST` Nous sommes en *$7A5F0*
Tapez : `D 7A5F0` (et descendez de quelques lignes)

```
d 7A5F0
~07A5F0 MOVE.W  #AA25,7E(A6)
~07A5F6 MOVEA.L #400,A0
~07A5FC MOVEQ   #29,D0
~07A5FE MOVEQ   #15,D1
~07A600 BSR     0007C41C
~07A604 MOVEA.L #1FFD0,A0
~07A60A MOVEQ   #3E,D0
~07A60C MOVEQ   #8,D1
~07A60E BSR     0007C41C
```

Nous sommes de retour dans un traitement 'normal' de notre *Trackloader(1)* avec l'utilisation de **A0**, **D0** et **D1**
Ce qui nous intéresse est donc forcément dans le code que l'on à examiner juste au-dessus.

Au vue de notre analyse, cela nous donnes :

```
;================================
07C340  BSR     7C586           ; ➔ GoTo #Conf_Disque
07C344  BSR     7C540           ; ➔ Notre BreakPoint pour GoTo #TrackZero
07C348  BCLR    #2,BFD100       ; Sélection Side Upper
07C350  BSR     7C35A           ; ➔ GoTo #TrackLoader(0)
07C354  BSR     7C3AA           ; ➔ GoTo #START_TRAITEMENT_MFM
07C358  RTS                     ; E.T retour maison
;================================
```

La routine diffère un petit peut de celle utilisé sur le Disque2, elle est plus simple ici mais elle reste la même.
Noter qu'un **CLR.L** est effectué à la fin de cette routine, il est intégré à la routine **#START_TRAITEMENT_MFM** et pas en dehors de ce code comme pour c'est le cas pour le Disque 2.

Du coup, c'est beaucoup plus simple, il suffit juste de désactiver le **BSR** en *$7C344* vers la routine `#TrackZero`
Pour cela et désactiver tout le traitement MFM qu'il en suit, on remplace le **BSR** par un **RTS**

On essaye. **Rebooter votre Amiga avec la disquette original N°1 de Blood Money dans le lecteur**
Sur le Title Screen, entrer dans l'AR

Tapez : `BS 7C340` puis `X`
Une fois notre *BreakPoints* ***$7C340*** atteint, on entre automatiquement dans l'AR.
Taper : `A 7C344`

```
^07C344 RTS                     ; GoTo #TrackZero désactivé.
^07C346 <RETURN>
```

Et on retourne au code, **Taper :** `X`

Le chargement continue et on arrive dans la partie : **Chip Anim**
Animation qui se poursuit complétement et le jeu continue son chargement mais… étrangement il revient à son point de départ et recharge le TitleScreen.

Je pense qu'il n'est pas utile de s'attarder plus sur les routines `#TrackZero` que l'on vient de voir, nous connaissons assez bien leurs fonctionnement grâce à son homologue sur le disque 2.

Nous sommes ici soit en présence d'une protection checksum ou d'un pointeur.
Revenons au code qui s'exécute après les routines `#TrackZero`, peut être sommes nous passer trop vite au niveau de leurs analyse.

```
d 7A5F0
~07A5F0 MOVE.W  #AA25,7E(A6)
~07A5F6 MOVEA.L #400,A0
~07A5FC MOVEQ   #29,D0
~07A5FE MOVEQ   #15,D1
~07A600 BSR     0007C41C
~07A604 MOVEA.L #1FFD0,A0
~07A60A MOVEQ   #3E,D0
~07A60C MOVEQ   #8,D1
~07A60E BSR     0007C41C
```

Comme précédemment dis, rien de bien spécial **SAUF** sur la 1er instruction, à savoir : **MOVE.W #AA25,7E(A6)**
Ok, **rebooter votre Amiga**, sur le Title Screen, **entrer dans l'AR**, poser un *BreakPoint* en sortie des routines de check **T00 :**
Taper : `BS 7C358`

Une fois notre *BreakPoints* ***$7C358*** atteint, on entre automatiquement dans l'AR, **taper :** `ST` puis `R`

```
Breakpoint raised at address: 0007C358

st
D0=4454A520 A5204454 0000000A 00001A52  00008189 00000103 FFFFFF10 0000FFFF
A0=00003E50 0007C6E2 0001863C 00FE86EE  00005C40 0007DBB8 00DFF000 00080000
PC = 0007A5F0 USP = 000018B2 SR = 2200 T=0 S=1 I=010 X=0 N=0 Z=0 V=0 C=0
~07A5F0 MOVE.W  #AA25,7E(A6)
```

Ce qui nous donnes comme adresse : *$DFF000+$7E = DFF07E = DSKSYNC (Disk Sync Pattern)*
Le **DSKSYNC** passe donc à **$AA25**

Hum, aucun rapport avec notre problème vu que les données se chargent bien ensuite.
Cependant, adresse à garder de coté car quand l'on créera nos disques de crack, celles-ci seront au format AmigaDos et donc *SyncWord* standard.

Cependant, cela ne répond toujours pas à notre question.
Voyons voir… Si on compare les routines de check **T00** actuel avec celle que l'on a vue sur le Disque2, la grande différence vient du traitement de fin.

Dans la routine présente sur le Disque 2, on copie des données vers une zone mémoire alors qu'ici, on pose une valeur en **$7A72C**

```
07C408 MOVE.L   #400,7A72C
```

Rebooter votre Amiga, sur le Title Screen, **entrer dans l'AR**, poser un *BreakPoint* en sortie des routines de check **T00 :**
Taper : `BS 7C358`

Une fois notre *BreakPoint* ***$7C358*** atteint, on entre automatiquement dans l'AR, on regarde ce qu'il y a en **$7A72C**
Taper : `M 7A72C`

```
Breakpoint raised at address: 0007C358
m 7A72C
:07A72C 00 00 04 00 49 FA 15 94 47 FA 14 78 42 79 00 07 ....I...G..xBy..
```

Hummm, c'est sûr qu'avec le **RTS** que l'on a posé, il n'y a pas eu le longword *$400* de posé à cette adresse.

On recommence, **rebooter votre Amiga**, sur le Title Screen, **entrer dans l'AR**, poser un *BreakPoints* en **$7C340**, avant l'appel vers les routines de **check T00**

Tapez : `BS 7C340` puis `X`
Une fois notre *BreakPoints* ***$7C340*** atteint, on entre automatiquement dans l'AR.
Taper : `A 7C344`

```
^07C344 RTS                     ; GoTo #TrackZero désactivé.
^07C346 <RETURN>
```

Mais avant de changer la valeur en **$7A72C**,on va jeter un coup d'œil à ce qu'il y a à cette adresse :
Taper : `D 7A72C` suivi de `M 7A72C`

```
d 7A72C-2
~07A72A JMP     0007A120
;=======================================

m 7A72C
:07A72C 00 07 A1 20 49 FA 15 94 47 FA 14 78 42 79 00 07 ... I...G..xBy..
```

Effectivement…cela ne risquait pas de fonctionner, sans le *LongWord* **$400** de poser, il faisait un **JMP** à une adresse non désirée et repart pour un tour. Je ne sais pas quel nom donner à ce genre de protection…
Quoi qu'il en soit, il faut à cette adresse la valeur de $400 en *LongWord*

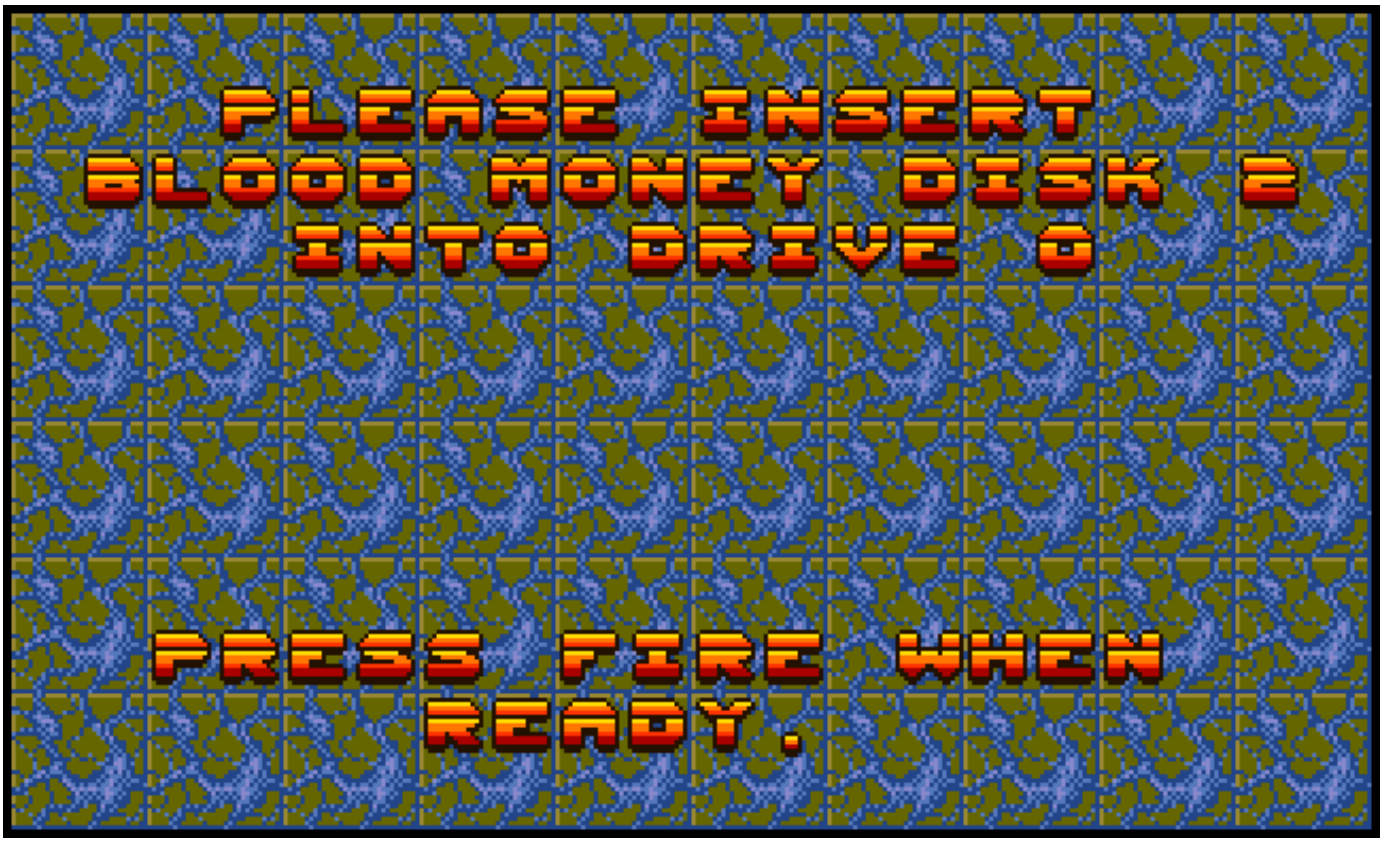
Taper : M 7A72C et mettons la valeur **$0400**

```
m 7A72C
:07A72C 00 07 A1 20 49 FA 15 94 47 FA 14 78 42 79 00 07 ... I...G..xBy..    AVANT

m 7A72C
:07A72C 00 00 04 00 49 FA 15 94 47 FA 14 78 42 79 00 07 ... I...G..xBy..    Apres
:07A73C CA 42 30 2C 00 00 B0 7C FF FF 67 00 02 6E 4A 40 .B0,...|..g..nJ@

d 7A72C-2
~07A72A JMP     00000400              Ce qui change tout, un joli JMP en $400
;======================================
```

(revenons 2 secondes au code) : Ceci est validé par le chargement qu'il effectue par la suite.

```
d 7A5F0
~07A5F0 MOVE.W  #AA25,7E(A6)
~07A5F6 MOVEA.L #400,A0        Un appel au Trackloader(1)
~07A5FC MOVEQ   #29,D0         avec comme Adr. de destination $400
~07A5FE MOVEQ   #15,D1
~07A600 BSR     0007C41C

~07A604 MOVEA.L #1FFD0,A0      et de suite après; un autre appel
~07A60A MOVEQ   #3E,D0         au trackloader(1) sans que l'on
~07A60C MOVEQ   #8,D1          est executé le code préalablement
~07A60E BSR     0007C41C       chargé en $400
                               Il est donc logique qu'il soit
                               executé/utilisé plus loin.
```

On retourne maintenant au code du jeu, **taper :** M
Le chargement continue et on arrive dans la partie : **Chip Anim** qui se termine toujours correctement
Les accès disques s'enchainent et l'on arrive sur la page **PLEASE INSERT BLOOD MONEY DISK 2**



Tout semble bon dans le meilleur des mondes.

Part 17 Récapitulatif : Patch, Checksum, Cheat

ADRESSE MEMOIRE	FICHIER	COMMANDE ORIGINALE	COMMANDE PATCHEE	TYPE	INFO
$02C16	14_22	*BNE 2BEC*	**NOP**	CHECKSUM	Bug Start Level
$03172	14_22	*BNE 314E*	**NOP**	CHECKSUM	Bug entrer dans une Shop
$044DA	14_22	*BNE 44D2*	**NOP**	CHECKSUM	Sup. compteur Track=00
$03F38	14_22	*BRA 3F38*	**RTS**	PATCH	DeadLoop
$04904	14_22	*BGT 4910*	**NOP, NOP**	PATCH	Empêche GoTo #Highscore
$049AE	14_22	*BNE 4994*	**NOP**	PATCH	Bloqué dans #Highscore
$00CC6	14_22	*CLR.W $1E(A5)*	**NOP, NOP**	PATCH	Bug Start Level
$6A084	01_30 20_49 31_60 50_79	*BSR 6A22C*	**BRA 6A150**	PATCH	Empêche routineS T00 du Disk2 Patch nécessaire dans tous les fichiers LEVEL
$7C344	28_29	*BSR 7C540*	**RTS**	PATCH	Empêche routineS T00 du Disk1
$7A72C	28_29	*00 07 A1 20*	**00 00 04 00**	PATCH	Change adr. du BRA vers $400
$0067E	14_22	*SUBQ.W #1,2(A4)*	**TST.W 20(A4)**	CHEAT	Vie infinie
$02FB8 $014CC $0152C	14_22	*SUBQ.W #1,20(A4)* *SUBQ.W #2,20(A4)* *SUBQ.W #2,20(A4)*	**TST.W 20(A4)**	CHEAT	Energie infinie
$08C2B	N/A	*XX XX XX*	**0F 42 3F**	CHEAT	Score Player1
$08C2E	N/A	*XX XX*	**03 E7**	CHEAT	Nbr de dollar

Part 18 Création crack Disque 2

Il est préférable de commencer par la création du disque 2 pour la simple et bonne raison qu'il n'y a rien à 'patcher' sur celui-ci.
En effet, si on regarde le chapitre : **Récapitulatif : Patch, Checksum, Cheat**, on peut s'apercevoir que la seule chose que l'on doit faire c'est un 'patch' sur les Levels Après leurs chargement.

Comme ce patche est identique pour tous les LEVELs et que, comme nous l'avons vue plus haut le code original des chargements des LEVELS Est spécifique (par rapport aux autres appels j'entends), il est très facile de prévoir un simple petit bout de code pour patcher après leur chargement.

Revenons sur le tableau final que nous avons déduis à la fin du **chapitre 10** sur **le rip des données du disque 2**.
Tous les appels sont dans la zone mémoire **($4D4 ← → $4BA2)**

Est Appelé par	Head	Move	A0	D0	D1	Length	Chargement de	Extra Info
$4326 + TrkLoad(2)	**DOWN**	14 → 14	5AE3E	0E	01	$1838	Signature disque 2	Check Disk2
$4D4 + TrkLoad(2)	**DOWN**	08 → 12	D55A	08	05	$7918	Menu du jeu	
$D76 + TrkLoad(2)	**DOWN**	01 → 04	7541E	01	04	$60E0	Menu choix Level	
$C50+$CB4+TrkLoad(2)	**UP**	01 → 30	3D090	01	1E	$2D690	Level1	Check T00 en fin de chargement
$C6C+$CB4+TrkLoad(2)	**UP**	31 → 60	3D090	1F	1E	$2D690	Level2	Check T00 en fin de chargement
$C88+$CB4+TrkLoad(2)	**DOWN**	50 → 79	3D090	32	1E	$2D690	Level3	Check T00 en fin de chargement
$CA4+$CB4+TrkLoad(2)	**DOWN**	20 → 49	3D090	14	1E	$2D690	Level4	Check T00 en fin de chargement
$4BA2 + TrkLoad(2)	**UP**	61 → 73	1B312	3D	0D	$13AD8	Animation de Fin	

On sait que le disque original utilise un format spécifique avec une longueur de Track de **$1838** au lieu de **$1600**
La réelle question est donc pour nous, est ce que les données que l'on a ripées peuvent tenir sur un disque standard ***AmigaDos***...

Si on calcul la sommes des fichiers que l'on a ripé sur le disque 2, cela entre sur une disquette standard ***AmigaDos*** de 80 pistes
Ps : Petit rappel d'une capacité d'une disquette ***AmigaDos*** : 512 octets*11 secteurs*80 pistes*2 faces=**901 120 Octets**

1838+7918+60E0+2D690+2D690+2D690+2D690+13AD8 = **$D8748 = 886 600 Octets**

Sauf que les données ne se suivent pas bout à bout sur le disque.
Nous avons choisie de passer par l'**AR** pour écrire les fichiers car c'est beaucoup plus simple et rapide, de plus nous avons aussi choisie de ne pas utiliser la fonction 'offset dans la track' ak, **D2** de notre *TrackLoader* custom, du coup chaque chargement se fera obligatoirement depuis le début d'une TRACK.

De plus, même si nous voulions mettre nos données ripées bout à bout cela deviendrait beaucoup plus complexe car.
Si l'on veut toujours utiliser que l'**AR,** il faudrait passer par plusieurs phases de calcul/écriture des tracks assez lourdes.
Et surtout modifier chaque appel avec l'offset dans la track...

En effet, d'origine pour chaque chargement nous utilisons **A0** en **longWord** pour **l'adresse de destination** et **D0** et **D1** en **Byte**
Si nous devions indiquer pour chaque chargement un offset il faudrait que ce soit en **LongWord** et du coup, passer par un **MOVE.L** et non un **MOVEQ.**

C'est problématique car **l'OPCODE** d'un **MOVE.L** est plus grande qu'un simple **MOVEQ** et nous n'avons pas spécialement la place pour cette modification sur **CHAQUE appel**.

Bien sûr, il serait possible de passer par une table mais si on regarde bien, il y a plus simple et rapide.

Déjà, on regarde d'origine, si on ne touche à rien ce que ça nous donnerait :
Pour chaque chargement, on prend la longueur, on la divise par **$1600** qui est la taille standard d'une **TRACK** ***AmigaDos*** et l'on arrondi à la valeur supérieure bien sûr, cela nous donne le tableau suivant

Chargement de	Length en Hexa	Length en Décimal	Track Amigados Nécessaire
Signature Disque 2	$1838	6200	2
Menu du jeu	$7918	31000	6
Menu choix Level	$60E0	24800	5
Level1	$2D690	186000	34
Level2	$2D690	186000	34
Level3	$2D690	186000	34
Level4	$2D690	186000	34
Animation de Fin	$13AD8	80600	15

2+6+5+34+34+34+34+15=**164**

Pas bon... on dépasse le maximum de 160 **TRACK** sur une disquette **AmigaDos**
Revoir chapitre Agencement des Disquettes si nécessaire.

Mais en y regardant de plus près, il est tout à fait possible de gagner de la place sur certain fichier **SANS passer par une phase de compression**.

Déjà, sur le 1er appel **Signature Disque 2** ,si vous vous souvenez bien, cette TRACK est vide de chez vide à part le **LongWord 2222** au début de celle-ci. On peut donc la réduite à 1 TRACK au lieu de 2

Insérez la disquette bloodmoneyD2 dans le lecteur interne de l'Amiga.
On va charger et regarder le contenu de chaque **LEVEL**, pourquoi les LEVELS ? Car ce sont les plus gros fichiers.
Entrer dans l'AR et taper : `LM 01_30, 20000`

```
lm 01_30, 20000
Loading from 020000 to 04D690
Disk ok
```

Fin des données en **$4D690**, on jette un coup d'œil à la fin de ces DATA chargés, **taper :** `D 4D690-80` et descendez un peu dans la mémoire.

```
m 4D690-80
:04D610 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:04D620 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:04D630 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:04D640 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:04D650 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:04D660 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:04D670 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:04D680 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:04D690 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Vide, ça c'est une bonne nouvelle mais vide jusqu'à où ? Remonter dans la mémoire toujours et chercher le 'début' de cette chaine de Zero.

```
:04D140 00 BF D1 00 61 90 52 79 00 06 A2 70 60 9C 08 39 ....a.Ry...p'..9
:04D150 00 04 00 BF E0 01 67 00 00 28 08 F9 00 01 00 BF ......g..(......
:04D160 D1 00 4E 71 4E 71 08 B9 00 00 00 BF D1 00 4E 71 ..NqNq........Nq
:04D170 4E 71 08 F9 00 00 00 BF D1 00 61 00 FF 5A 60 CE Nq........a..Z'.
:04D180 42 79 00 06 A2 70 4E 75 1C 39 00 BF DD 00 1C 39 By...pNu.9.....9
:04D190 00 BF DD 00 08 06 00 04 67 F4 4E 75 08 B9 00 07 ........g.Nu....
:04D1A0 00 BF D1 00 7C 64 08 B9 00 03 00 BF D1 00 51 CE ....|d........Q.
:04D1B0 FF F6 61 00 FF 22 4E 75 08 F9 00 07 00 BF D1 00 .öa.."Nu........
:04D1C0 08 F9 00 03 00 BF D1 00 7C 64 08 B9 00 03 00 BF ........|d......
:04D1D0 D1 00 51 CE FF F6 08 F9 00 03 00 BF D1 00 4E 75 ..Q..ö........Nu
:04D1E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:04D1F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:04D200 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:04D210 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:04D220 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Sur le fichier 01_30 (qui correspond au **LEVEL1**), on a donc une zone mémoire vide de $4D690-$4D1E0=**$4B0** de longueur

En clair, les DATA vont de **$20000** à **$4D1DF**, soit **185 800 Octets** au lieu des 186 000 Attendu.

Charger les autres fichiers (31_60, 50_79, 20_49) et jetez un coup d'œil pour voir si c'est aussi le cas.
Exemple : `LM 31_60, 20000` puis `M 4D1B0` et on descend un peu dans la mémoire
Vous verrez que c'est le cas sur TOUS les fichiers LEVELs ripés.

On peut donc revoir la taille à sauver des 4 Levels à **185 800 octets**
185800/$1600= 32,9 **donc 33 TRACKs**
Ce qui nous fait gagner donc 1 TRACKS par LEVEL.
4 LEVELS = 4 TRACKS de gagner + 1 TRACK de GAGNER sur la signature disque ce qui nous fait un total de 5 TRACKS en moins

Et maintenant, (je vous laisse refaire le calcul), on est bon 😊

Il suffira de créer un petit bout de code **AVANT** le chargement des **LEVELS** pour mettre à zéro la zone mémoire **$6A270 à $6A720**
3D090+$2D690-$4B0=$6A270***

On met à jour notre joli tableau

Chargement de	Nom du fichier ripé	Length en Hexa	Length en Décimal	Track Amigados Nécessaire	New Tracks cible
Signature Disque 2	14_14	$1838	6200	!01	00 → 01
Menu du jeu	08_12	$7918	31000	!06	02 → 07
Menu choix Level	01_04	$60E0	24800	!05	08 → 12
Level1	01_30	$2D690	186000	!33	13 → 45
Level2	31_60	$2D690	186000	!33	46 → 78
Level3	50_79	$2D690	186000	!33	79 → 111
Level4	20_49	$2D690	186000	!33	112 → 144
Animation de Fin	61_73	$13AD8	80600	!15	145 → 159

Bon… y'a plus qu'a comme on dit 😊

Retirer toutes les disquettes des lecteurs Amiga (**DF0 et DF1 donc**)
Insérez notre disquette `bloodmoneyD2` dans le lecteur externe **DF1**
Insérez maintenant une nouvelle disquette dans le lecteur DF0, tapez : `FORMAT CR_BLOOD_MONEY_D2` et **validez le tout**.

Comme vous l'avez compris, cela sera notre disquette Final de Crack Disque2
Entrer dans l'AR si ce n'est pas déjà fait.

Nombre de track *AmigaDos* nécessaire pour le fichier **14_14** = 1, donc 1*$1600= *$1600*
Taper : `O 00, 20000 20000+1600`
Puis : `LM 1:14_14, 20000`
Et : `WT 0 1 20000`

Nombre de track *AmigaDos* nécessaire pour le fichier **08_12** = 6, donc 6*$1600=*$8400*
Taper : `O 00, 20000 20000+8400`
Puis : `LM 1:08_12, 20000`
Et : `WT 2 6 20000`

Nombre de track *AmigaDos* nécessaire pour le fichier **01_04** = 5, donc 5*$1600=*$6E00*
Taper : `O 00, 20000 20000+6E00`
Puis : `LM 1:01_04, 20000`
Et : `WT 8 5 20000`

Nombre de track *AmigaDos* nécessaire pour le fichier **01_3**0 = 33, donc !33*$1600=*$2D600*
Taper : `O 00, 20000 20000+2D600`
Puis : `LM 1:01_30_comp, 20000`
Et : `WT !13 !33 20000`

Nombre de track *AmigaDos* nécessaire pour le fichier **31_60** = 33, donc !33*$1600=*$2D600*
Taper : `O 00, 20000 20000+2D600`
Puis : `LM 1:31_60_comp, 20000`
Et : `WT !46 !33 20000`

Nombre de track *AmigaDos* nécessaire pour le fichier **50_79** = 33, donc !33*$1600=*$2D600*
Taper : `O 00, 20000 20000+2D600`
Puis : `LM 1:50_79_comp, 20000`
Et : `WT !79 !33 20000`

Nombre de track *AmigaDos* nécessaire pour le fichier **20_49** = 33, donc !33*$1600=*$2D600*
Taper : `O 00, 20000 20000+2D600`
Puis : `LM 1:20_49_comp, 20000`
Et : `WT !112 !33 20000`

Nombre de track *AmigaDos* nécessaire pour le fichier **61_73** = 15, donc !15*$1600= *$14A00*
Taper : `O 00, 20000 20000+14A00`
Puis : `LM 1:61_73, 20000`
Et : `WT !145 !15 20000`

Part 19 Création crack Disque 1

Maintenant on peut s'attaquer à la création du disque 1

- On va devoir, cibler tous les fichiers **ET** leurs tailles à écrire sur notre Disque 1 cracké.
- Calculer où placer ses fichiers sur notre disquette physiquement.
- Modifier TOUS les appels vers le trackloader original en conséquence (**D0** et **D1)**
- Modifier le code original pour utiliser un chargement 'standard', un format AmigaDos
- Ecrire le tout sur notre disquette de CRACK

Aux vues de ce que l'on a vu dans le chapitre : **Récapitulatif : Patch, Checksum, Cheat** on va devoir modifier les 'fichiers' : **trackloader(1)** et **main Prg**

Petit rappel : **TrackAmigaDos** standard = $1600 = 5632 octets

Adr. de chargement Mémoire réel	Est Appelé par	Nom du fichier ripé	Chargement de	Longueur en décimal	Nbr de TRACK AmigaDos Nécessaire	New Tracks cible
$7A120	Trackdisk.device	28_29	Trackloader(1)	12400	!03	02 → 04
$58E44	$7A446 + TrkLoad(1)	30_49	Screenshot	124000	!23	05 → 27
$8A48	$7A4C6 + TrkLoad(1)	01_13	N/A	80600	!15	28 → 42
$2E7C0	$7A4DC + TrkLoad(1)	01_28	N/A	173600	!31	43 → 73
$1C520	$7A4F2 + TrkLoad(1)	29_40	Psysnosis Animation	74400	!14	74 → 87
$400	$7A5F6 + TrkLoad(1)	41_61	N/A	130200	!24	88 → 111
$1FFD0	$7A604 + TrkLoad(1)	62_69	Chip Animation	49600	!09	112 → 120
$400	$7A6FA + TrkLoad(1)	14_22	Main Prg	55800	!10	121 → 130
$14E72	$7A710 + TrkLoad(1)	23_27	Insert Disk2 Message	31000	!06	131 → 136

On commencera à poser nos fichiers en Piste 01. (Track2 donc)
La piste 00 sera gardé pour notre bootsecteur et pourquoi pas, un petit crackTro.

1er fichier à charger est le rip ***28_29,*** nous allons utiliser le trackdisk.device disponible au boot pour faire le travail.
Le fichier fait 12400 octets de longueur et nous avons dis que l'on gardait la piste 0 complète pour le boot et une cracktro.
Donc nous devrons placer ce fichier à la fin de la 1er track (on commence de 0, donc Track0 + Track1 = garder pour cracktro et boot), donc **Track2**
Une track AmigaDos = $1600 = !5632 multiplier par deux nous donnes **$2C00** comme adresse physique de destination.
Le fichier ***28_29*** fait donc 12400 octets ce qui nous donnes **$3070**
$3070 / longueur d'une track AmigaDos nous donnes : $3070/$1600 = 2,2
Le fichier ***28_29*** prendra donc 3 tracks.

On récapitule :
Track 00 : BootSecteur (contenant notre code de chargement via le *trackdisk.device*)
Track 01 : Vide, réservé pour une cracktro (par exemple)
Track 02 : Fichier ***28_29***
Track 03 : Fichier ***28_29***
Track 04 : Fichier ***28_29***

Insérez maintenant une nouvelle disquette dans le lecteur, tapez : `FORMAT CR_BLOOD_MONEY_D1` et validez le tout.
On charge le bootsecteur en mémoire, **taper :** `RT 0 1 20000`

On vérifie qu'on a bien la signature DOS en offset 0, **taper :** `M 20000`

```
:02000  44 4f 53 00 00 00 00 00 00 00 00 00 00 00 00 00 DOS.............
```

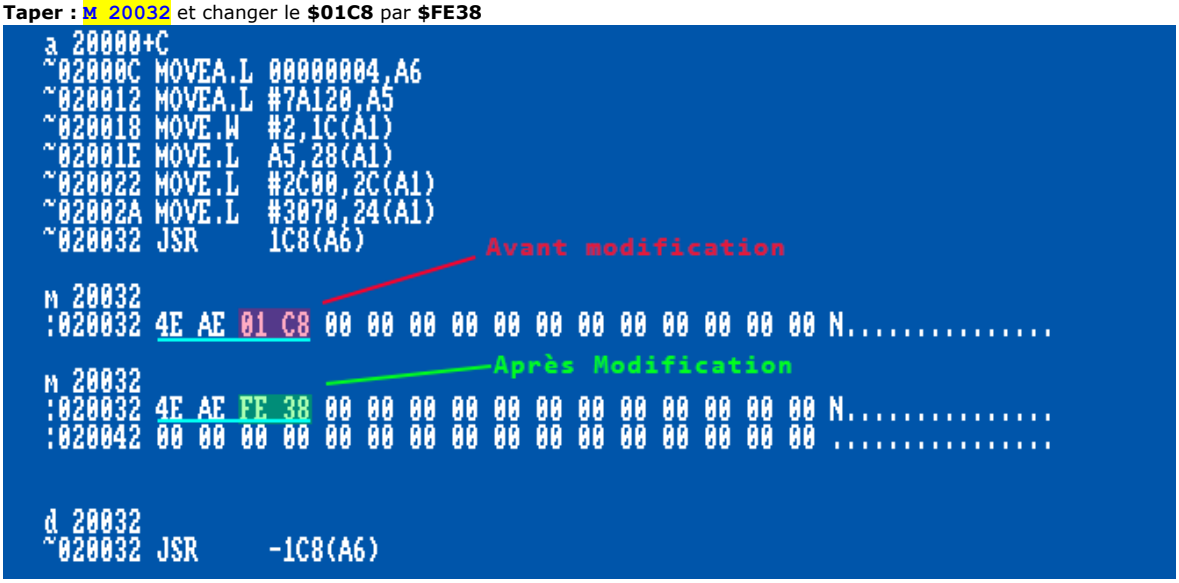
0n nettoie un peut le tout, **taper :** `O 00, 20000 20000+1600`
Et maintenant, on s'attaque maintenant à notre code propre code du bootsecteur, **taper :** `A 20000+c`

```
^02000C MOVEA.L $4,A6                  ; A6=execbase
^020012 MOVE.L  #$7A120,A5             ; A5=$7A120, notre adresse où l'on va copier notre code
^020018 MOVE.W  #$2,$1C(A1)            ; Utilisation du trackdisk.device en mode lecture ($2)
^02001E MOVE.L  A5,$28(A1)             ; $28(A1)=Trackdisk.device destination mémoire, à savoir A5 ou $7A120
^020022 MOVE.L  #$2C00,$2C(A1)         ; $2C(A1)=Trackdisk.device adr. Physique source sur le disk
^02002A MOVE.L  #$4200,$24(A1)         ; $24(A1)=Trackdisk.device Long. à copier (/!\ Multiple d'une TRACK)
^020032 JSR     1C8(A6)                ; Appel du trackdisk.device
```

/! Bon alors ici on a un souci, **l'AR** n'est pas vraiment prévue pour être utilisé en tant qu'assembleur.
L'appel du trackdisk.device se fait normalement via la commande **JSR -1C8(A6)**
Sauf que… impossible à taper ça sous l'AR, il n'en veut pas (en fait c'est le '-' qui lui pose problème).
Mais chaque problème a une solution. On va calculer l'opcode de -$01C8.

Je ne vais pas entrer dans des cours de binaires and co, ce sont normalement des bases nécessaires.
Pour calculer le négatif on procède comme suis, on prend la valeur max nécessaire, ici un Word donc $FFFF
On lui ajoute +1 ce qui nous donnes $10000 et on lui soustraie la valeur négative désiré, ici $1C8
$10000-$1C8=$FE38
L'opcode du **JSR** ne change pas lui **$4EAE** en l'occurrence
Mais le **word** qui le suit doit avoir la valeur calculé, à savoir **$FE38** (et non $01C8)

Taper : `M 20032` et changer le **$01C8** par **$FE38**



Nous n'allons plus utiliser le TrackLoader d'origine mais une version modifiée de celui d'AlphaOne. (voir les derniers chapitres de ce tuto)
Petit rappel, d'origine le code fonctionnait avec les variables :

A0	LongWord	**LEA**	Adresse de Destination
D0	Byte	**MOVEQ**	Piste de départ
D1	Byte	**MOVEQ**	Nbr de Track à lire

Et notre Trackloader d'AlphaOne lui fonctionne avec

A0	LongWord	**LEA**	Adresse de Destination
D0	LongWord	**MOVE.L**	Longueur en Byte à lire
D1	LongWord	**MOVE.L**	Track de départ pour lecture
D2	LongWord	**MOVE.L**	Offset dans la Track (dans notre cas 0 puisque nos données sont toujours au début de la
A6	LongWord	**LEA/MOVE.L**	DFF000, pas utile car A6 est déjà positionné à cette valeur dans le code original
A2	LongWord	**LEA/MOVE.L**	Adresse du Buffer MFM, dépends pour nous de quelle version de TrackLoader on parle (1)/(2)

Sur **A0**, aucun souci puisque que c'est identique entre le *TrackLoader* d'origine et nos *TrackLoader* que l'on a modifié.
Pour **D0** et **D1** c'est problématique car d'un coté on parle en PISTE et de l'autre il s'attends à recevoir des TRACKS.
Ou plus exactement, d'origine on utilise un **MOVEQ** (Longueur OPCODE = 2 Bytes)
et on devrait lui envoyer, dans le cas de **D0**, une longueur à lire donc un **LongWord** en **MOVE.L** (Longueur OPCODE = 6 Bytes)
Nous n'avons pas tout le temps cette place libre nécessaire.

La meilleure solution est de créer et d'utiliser une table.
Nous allons créer un code qui fonctionnera avec **D0** et récupèrera les valeurs de notre Table en mémoire.
Positionnera ensuite **D0** et **D1** correctement avant d'appeler notre version de TrackLoader

Si vous avez bien suivi toutes les phases d'occupation mémoire des DATA du jeu, vous avez pu remarquer qu'une petite zone mémoire était disponible. Nous allons donc créer notre table en **$200**

Mais avant cela, il faut créer un petit bout de code qui copiera notre table mis en BootSecteur vers la zone mémoire **$200**
Taper : `M 20032+4`

```
^020036 LEA     20050(PC),A0          ; A0=Adresse source où se trouve nos datas à copier
^02003A LEA     200,A1                ; A1=$200, adresse mémoire de destination.
^020040 MOVE.W  #$01,D0               ; D0=Longueur à copier en Byte, pour l'instant, aucune idée, on met 1
^020044 MOVE.B  (A0)+,(A1)+           ; ➔ On copie notre table en mémoire $200
^020046 DBF     D0,20044              ; ← On décrémente D0 et tant qu'il n'est pas égal à -1, on boucle
^02004A JMP     $7A720                ; On saute vers le code chargé du fichier 28_29
<RETURN>
```

Pour l'instant, nous n'avons que les informations/positionnements des données sur le disque 2, il nous manque les informations du disque 1 avant de pouvoir créer cette table, aussi, nous mettons de coté la création de celle-ci et continuons-la suite de la création de notre Disque1

On écrit quand même le code que l'on vient de créer sur notre disquette de crack N°1
On calcule le nouveau checksum du bootblock et on sauve le tout sur **DF0**
Taper : `BOOTCHK 20000` puis `WT 0 1 20000`

Avant de partir dans la section 'Patch' lui-même qui, finalement ne concerne que 2 fichiers (**28_29** et **14_22**)
On va écrire tous nos fichiers ripés sur notre nouveau disque.

Comme d'habitude, je pars du principe que vous avez un second lecteur de disquette, à savoir **DF1:**

- **Dans DF0,** laisser le **nouveau disk** que l'on est en train de créer.
- **Dans DF1,** insérez notre disquette `bloodmoneyD1` contenant tous nos fichiers ripés du disque Original 1.

Dans l'ordre :

- On nettoie une zone mémoire réservée avant de charger le fichier.
- On charge le fichier ripé depuis **DF1**
- On l'écrie à la *TRACK* désiré (voir tableau ci-dessous/dessus)

Petit rappel du tableau :

Adr. de chargement Mémoire réel	Est Appelé par	Nom du fichier ripé	Chargement de	Longueur en décimal	Nbr de TRACK AmigaDos Nécessaire	New Tracks cible
$7A120	Trackdisk.device	28_29	Trackloader(1)	12400	!03	02 ➔ 04
$58E44	$7A446 + TrkLoad(1)	30_49	Screenshot	124000	!23	05 ➔ 27
$8A48	$7A4C6 + TrkLoad(1)	01_13	N/A	80600	!15	28 ➔ 42
$2E7C0	$7A4DC + TrkLoad(1)	01_28	N/A	173600	!31	43 ➔ 73
$1C520	$7A4F2 + TrkLoad(1)	29_40	Psysnosis Animation	74400	!14	74 ➔ 87
$400	$7A5F6 + TrkLoad(1)	41_61	N/A	130200	!24	88 ➔ 111
$1FFD0	$7A604 + TrkLoad(1)	62_69	Chip Animation	49600	!09	112 ➔ 120
$400	$7A6FA + TrkLoad(1)	14_22	Main Prg	55800	!10	121 ➔ 130
$14E72	$7A710 + TrkLoad(1)	23_27	Insert Disk2 Message	31000	!06	131 ➔ 136

Nombre de track *AmigaDos* nécessaire pour le fichier **28_29** = 3, donc 3*$1600= *$4200*
Taper : `O 00, 7A120 7A120+4200`
Puis : `LM 1:28_29, 7A120`
Et : `WT 2 3 7A120`

Nombre de track *AmigaDos* nécessaire pour le fichier **30_49** = 23, donc !23*$1600= *$1FA00*
Taper : `O 00, 58E44 58E44+1FA00`
Puis : `LM 1:30_49, 58E44`
Et : `WT 5 !23 58E44`

Nombre de track *AmigaDos* nécessaire pour le fichier **01_13** = 15, donc !15*$1600= *$14A00*
Taper : `O 00, 8A48 8A48+14A00`
Puis : `LM 1:01_13, 8A48`
Et : `WT !28 !15 8A48`

Nombre de track *AmigaDos* nécessaire pour le fichier **01_28** = 31, donc !31*$1600= *$2AA00*
Taper : `O 00, 2E7C0 2E7C0+2AA00`
Puis : `LM 1:01_28, 2E7C0`
Et : `WT !43 !31 2E7C0`

Nombre de track *AmigaDos* nécessaire pour le fichier **29_40** = 14, donc !14*$1600= *$13400*
Taper : `O 00, 1C520 1C520+13400`
Puis : `LM 1:29_40, 1C520`
Et : `WT !74 !14 1C520`

Nombre de track *AmigaDos* nécessaire pour le fichier **41_61** = 24, donc !24*$1600= *$21000*
Taper : `O 00, 20000 20000+21000`
Puis : `LM 1:41_61, 20000`
Et : `WT !88 !24 20000`

Nombre de track *AmigaDos* nécessaire pour le fichier **62_69** = 9, donc !9*$1600= *$C600*
Taper : `O 00, 1FFD0 1FFD0+C600`
Puis : `LM 1:62_69, 1FFD0`
Et : `WT !112 9 1FFD0`

Nombre de track *AmigaDos* nécessaire pour le fichier **14_22** = 10, donc !10*$1600= *$DC00*
Taper : `O 00, 20000 20000+DC00`
Puis : `LM 1:14_22, 20000`
Et : `WT !121 !10 20000`

Nombre de track *AmigaDos* nécessaire pour le fichier **23_27** = 6, donc !6*$1600= *$8400*
Taper : `O 00, 14E72 14E72+8400`
Puis : `LM 1:23_27, 14E72`
Et : `WT !131 6 14E72`

Bon… comme nous l'avons vue, nous n'avons que 2 fichiers à modifier, commençons donc par le 1er à savoir : **28_29**
C'est lui qui contient le *trackloader* principale utilisé pour tous les chargements du Disque1, c'est aussi lui qui contient tous les appels vers ce *trackloader*.

Petit rappel, le trackloader se trouve en **$7C41C**

```
;================================          ; #TRACKLOADER base
7C41C    MOVEM.L  D0-D1/A0,-(A7)           ; Sauvegarde des registres dans la pile
7C420    BSR      7C480                    ; GoTo #READ TRACK
7C424    MOVEM.L  (A7),D0-D1/A0            ; On restaure les registres de la pile
7C428    BSR      7C452                    ; GoTo #Traitement MFM
7C42C    MOVEM.L  (A7)+,D0-D1/A0           ; On restaure les registres de la pile
7C430    CLR.W    D3                       ; On nettoie D3
7C432    MOVEA.L  A0,A1                    ; Copie l'adr. Destination mémoire présente en A0 Dans A1
7C434    MOVE.W   #C1B,D4                  ; D4 = $C1B
7C438    ADD.W    (A1)+,D3                 ; ➔ Boucle qui va lire les 2 bytes de l'adr. A1
                                           ; et les ajouter à D3 puis A1=A1+2

7C43A    DBF      D4,7C438                 ; ➔ Boucle avec décrémentation D4,
                                           ; donc il va faire cette manip 3100
                                           ; oui 3100 car le dbf va jusqu'à -1 donc $C1C+1
                                           ; $C1B+1=$C1C= !3100 en décimal

                                           ; Comme on parle de word (2 bytes, donc XX XX),
                                           ; cela fait !6200  ($C1C*2) de donnée lue et traitée

7C43E    MOVE.W   (A1),D4                  ; On lit des deux dernier bytes(WORD contenue à l'adresse A1)
                                           ; donc A0+6198 et on met ça dans D4

7C440    CMP.W    D3,D4                    ; On compare cette valeur lue, en D4,
                                           ; avec ce que l'on calculé en D3

7C442    BNE      7C41C                    ; Non, c'est que les données lues ne sont pas 'correctes',
                                           ; on repart pour un tour, GoTo #TRACKLOADER base
                                           ;
                                           ; On a donc ici une routine de check, de checksum

7C444    ADDA.L   #1838,A0                 ; A0 = A0+$1838   // Taille d'une track ?
7C44A    ADDQ.L   #1,D0                    ; D0 = D0+1
7C44C    SUBQ.W   #1,D0                    ; D1 = D1-1
7C44E    BNE      7C41C                    ; On Reboucle sur le début de cette routine
7C450    RTS                               ; E.T retour maison
;================================

;================================          ; #Traitement MFM
7C452    MOVEM.L  D0/A0,-(A7)              ; Sauvegarde D0 et A0 dans la pile
7C456    MOVE.W   D0,D5                    ; D5 = D0, Numéro de piste
7C458    MOVEA.L  #76EC0,A1                ; A1 = 76EC0 = adr buffer MFM + 4
7C45E    MOVE.W   #C1C,D7                  ; Compteur D7=$C1C à savoir !3100, la moitié de !6200
                                           ; ($1838 en hexa)
                                           ;
                                           ; ➔ Début boucle, Traitement donc ici d'une 'track custom'
                                           ; d'une taille de $1838

7C462    MOVE.W   (A1)+,D0                 ; Copie le word de l'adresse pointé par A1 (Word impair)
                                           ; vers D0 et A1=A1+2

7C464    MOVE.W   (A1)+,D0                 ; Copie le word de l'adresse pointé par A1 (word pair
                                           ; vers D1 et A1=A1+2

7C466    ASL.W    #1,D0                    ; Décalage de 1 bit vers la gauche
7C468    ANDI.W   #AAAA,D0                 ; Décodage MFM bits pair de D0
7C46C    ANDI.W   #5555,D1                 ; Décodage MFM bits impair de D1
7C470    OR.W     D1,D0                    ; Traitement des bits paire et impair #1
7C472    EOR.W    D5,D0                    ; Traitement des bits paire et impair #2
7C474    MOVE.W   D0,(A0)+                 ; Finalement, on copie le word MFM décodé vers l'adresse
                                           ; mémoire de destination puis A0=A0+2

7C476    DBF      D7,7C462                 ; ➔ Boucle de copie avec décrémentation D7
                                           ; on traite l'intégralité d'une 'track custom'

7C47A    MOVEM.L  (A7)+,D0/A0              ; On restaure ce que l'on a sauvé dans la pile
7C47E    RTS                               ; E.T retour maison
;================================

;================================          ; #READ TRACK
7C480    BSR      7C4EC                    ; Goto #Position Reach ?
7C484    MOVEA.L  #76EBC,A1                ; A1 = 76EBC, adresse buffer MFM
7C48A    MOVE.W   #8210,96(A6)             ; Conf DMACON, DSKEN enable, ALL DMA enable
7C490    MOVE.L   A1,20(A6)                ; ➔ Boucle, lecture/remplissage buffer MFM
7C494    CLR.W    24(A6)                   ; conf DSKLEN,  DMA data Length clear
7C498    BSR      7C572                    ; Goto #CIA READY ?
7C49C    MOVE.W   #4000,24(A6)             ; DMA disk off
7C4A2    MOVE.W   #9842,24(A6)             ; DMA disk enable, Length DMA data=$1842
7C4A8    MOVE.W   #9842,24(A6)             ; Deux fois pour démarrer le transfert DMA

7C4AE    BSR      7C4C6                    ; Goto #DiskBlockFinished ?
7C4B2    CMPI.L   #552A2A55,(A1)           ; Recherche la sequence 552A2A55, SyncWord dans le buffer MFM

7C4B8    BNE      7C490                    ; ➔ Tant que ce n'est pas bon, on boucle,
                                           ; on continue la lecture DMA dsk

7C4BA    CLR.W    24(A6)                   ; conf DSKLEN,  DMA data Length=$0000
7C4BE    MOVE.W   #10,96(A6)               ; Clear Disk DMA
7C4C4    RTS                               ; E.T retour maison
;================================
```

```
;================================          ; #DiskBlockFinished ?
7C4C6    MOVE.W   1E(A6),D0                ; A6=DFF000, DFF01E=INTREQR
7C4CA    BTST     #1,D0                    ; On test le bit 1 de de INTREQR, à savoir :
                                           ; Level 1 Disk Block Finished Interupt

7C4CE    BEQ      7C4C6                    ; Test ? ← On boucle
7C4D0    MOVE.W   #2,9C(A6)                ; clear DSKBLK(Disk Block Finished Interuppt)
7C4D6    RTS                               ; E.T retour maison
;================================

;================================          ; #FLOPPY READY ?
7C4D8    MOVE.W   #BB8,D6                  ; Boucle d'attente D6 = $BB8, $BB8=!3000
7C4DC    DBF      D6,7C4DC                 ; <== =>> boucle d'attente
7C4E0    BTST     #5,BFE001                ; ==> CIA-A / PRA, On test le bit5, lecteur prêt ?
7C4E8    BNE      7C4E0                    ; <== Non ? on boucle jusqu'a que ce soit le cas.
7C4EA    RTS                               ; E.T retour maison

;================================          ; #Position Reach ?
7C4EC    CMP.B    7CA3E,D0                 ; On compare le pointeur en 7CA3E
                                           ; (pointeur, Position Track) avec D0

7C4F2    BNE      7C4F8                    ; Si ce n'est pas identique, GoTo #Move Exter.
7C4F6    RTS                               ; E.T retour maison
;================================

;================================
7C4F8    BGT      7C51E                    ; #Move Exter.
7C4FC    BSET     #1,BFD100                ; CIA-B / PRB, bit1 mis à 1, DIR=vers l'extérieur
7C504    BCLR     #0,BFD100                ; CIA-B / PRB, bit0 mis à 0, STEP, pré/déplacement tête
7C50C    BSET     #0,BFD100                ; CIA-B / PRB, bit0 mis à 1, STEP, déplacement tête
7C514    BSR      7C4D8                    ; GoTo #FLOPPY READY ?
7C516    SUBQ.B   #1,7CA3E                 ; Mise à jour du pointeur de track -1
7C51C    BRA      7C4EC                    ; GoTo #Position Reach ?
;================================

;================================          ; #Move Inter.
7C51E    BCLR     #1,BFD100                ; CIA-B / PRB, bit1 mis à 0, DIR=vers l'intérieur
7C526    BCLR     #0,BFD100                ; CIA-B / PRB, bit0 mis à 0, STEP, pré/déplacement tête
7C52E    BSET     #0,BFD100                ; CIA-B / PRB, bit0 mis à 0, STEP, déplacement tête
7C536    BSR      7C4D8                    ; GoTo #FLOPPY READY ?
7C538    ADDQ.B   #1,7CA3E                 ; Mise à jour du pointeur de track +1
7C53E    BRA      7C4EC                    ; GoTo #Position Reach ?
;================================

;================================          ; #Retour T00
7C540    BTST     #4,BFE001                ; CIA-A / PRA, On test le bit4, TK0, Tête sur T00 ?
7C548    BEQ      7C56A                    ; Oui ? GoTo #Update Position Track
7C54C    BSET     #1,BFD100                ; CIA-B / PRB, bit1 mis à 1, DIR=vers l'extérieur
7C554    BCLR     #0,BFD100                ; CIA-B / PRB, bit0 mis à 0, STEP, pré/déplacement tête
7C55C    BSET     #0,BFD100                ; CIA-B / PRB, bit0 mis à 1, STEP, déplacement tête
7C564    BSR      7C4D8                    ; GoTo #FLOPPY READY ?
7C568    BRA      7C540                    ; E.T retour maison
;================================

;================================          ; #Update Position Track
7C56A    CLR.B    7CA3E                    ; On nettoie en Byte le pointeur en 7CA3E
7C570    RTS                               ; E.T retour maison
;================================

;================================          ; #CIA READY ?
7C572    MOVE.B   BFDD00,D6                ; D6 = BFDD00
7C578    MOVE.B   BFDD00,D6                ; D6 = icr register de CIA-B
7C57E    BTST     #4,D6                    ; Test du bit4 (FLAG) de icr
7C582    BEQ      7C578                    ; Interruption générée ? On boucle
7C584    RTS                               ; E.T retour maison
;================================

;================================          ; #MOTOR ON
7C586    BCLR     #7,BFD100                ; CIA-B / PRB, bit7 mis à 0, motor On
7C58E    MOVEQ    #64,D0                   ; D0 = 64
7C590    BCLR     #3,BFD100                ; ← Boucle avec le CIA-B, bit3 à 0, DF0 sélectionné
7C598    DBF      D0,7C590                 ; → Boucle avec décrémentation de D0
7C59C    BSR      7C4D8                    ; GoTo #FLOPPY READY ?
7C5A0    RTS                               ; E.T retour maison
;================================

;================================          ; #STOP FLOPPY
7C5A2    BSET     #7,BFD100                ; CIA-B / PRB, bit7 mis à 1, motor OFF
7C5AA    BSET     #3,BFD100                ; CIA-B / PRB, bit3 mis a 1, SEL0, DF0 désélectionné
7C5B2    MOVEQ    #64,D0                   ; D0 = 64
7C5B4    BCLR     #3,BFD100                ; → Boucle avec le CIA-B, bit3 à 0, DF0 sélectionné
7C5BC    DBF      D0,7C5B4                 ; ← Boucle avec décrémentation de D0
7C5C0    BSET     #3,BFD100                ; CIA-B / PRB, bit3 mis a 1, SEL0, DF0 désélectionné
7C5C8    RTS                               ; E.T retour maison
;================================
```

- **Autre analyse sur ce code**, on peut s'apercevoir que les registres sont sauvés tout au début du code.
- Que le code utilise l'ensemble des registres **D0 à D7** et qu'il utilise aussi **A0, A1, A5, A6**
- **A2** et **A3** ne sont pas utilisée et **A5** et **A6** sont utilisé uniquement en LECTURE (pour leur adresse pointé) et ne sont pas modifié.

Par ailleurs, si on recherche tous les appels vers toutes les sous-routines de ce **TrackLoader**
grâce à la commande **FA** (Exemple ; **FA 7C5A2**)

Cela nous donne le tableau suivant.

Adresse mémoire	Routine	Zone d'appel	Routine d'appel
7C41C	#TRACKLOADER base	7A458, 7A4D8, 7A4EE, 7A4FC, 7A600, 7A60E, 7A70C, 7A71A, 7C442, 7C44E	#TRACKLOADER base + 8 appels Hors code du TrackLoader via D0/D1 et A0
7C452	#Traitement MFM	7C428	#TRACKLOADER base
7C480	#READ TRACK	7C420	#TRACKLOADER base
7C4C6	#DiskBlockFinished ?	7C4AE, 7C4CE	#READ TRACK, #DiskBlockFinished ?
7C4D8	#FLOPPY READY ?	7C514, 7C536, 7C564, 7C59C	#Move Exter., #Move Inter., #Retour T00, #MOTOR ON
7C4EC	#Position Reach ?	7C480, 7C51C, 7C53E	#READ TRACK, #Move Exter., #Move Inter.
7C4F8	#Move Exter.	7C4F2	#Position Reach ?
7C51E	#Move Inter.	7C4F8	#Move Exter.
7C540	#Retour T00	7A442, 7C344, 7C568	#Retour T00 et 2 pré-code (7A442 et 7C344)
7C56A	#Update Position Track	7C548	#Retour T00
7C572	#CIA READY ?	7C372, 7C498	#Traitement MFM, #READ TRACK
7C586	#MOTOR ON	7A43E, 7A6F6, 7C340	Juste avant le chargement du Screenshot, il lance le moteur. Juste avant le chargement du dernier N/A, il lance le moteur. Juste avant la routine de check T00, il lance le moteur.
7C5A2	#STOP FLOPPY	7A500, 7A612, 7A71E	Juste après le chargement de la psysnosis animation, il coupe le moteur. Juste après le chargement de la Chip Animation, il coupe le moteur. Juste après le chargement de 'Insert Disk2', il coupe le moteur.

Tous les appels se font depuis le code du trackloader lui-même sauf ce qui est souligné en jaune dans le tableau ci-dessus.

+ 8 appels pré-code des autres 'données' via A0/D0/D1
Normal, ce sont les appels proprement dis vers le *trackloader*, nous allons devoir les modifier un par un.

2 pré-code (7A442 et 7C344)
Logiquement il doit en avoir un de base, de départ et un autre pour la vérification de la protection en **T00**
Le 1er est inutile car nous allons intégrer un trackloader qui n'a pas besoin de partir de T00 AVANT sont code, il le gère lui-même
Le second est inutile aussi car nous allons 'patcher' comme déjà vue la routine de protection en **T00**

Juste avant le chargement du Screenshot, il lance le moteur.
Juste avant le chargement du dernier N/A, il lance le moteur.
Juste avant la routine de check T00, il lance le moteur.

Juste après le chargement de la psysnosis animation, il coupe le moteur.
Juste après le chargement de la Chip Animation, il coupe le moteur.
Juste après le chargement de 'Insert Disk2', il coupe le moteur.

M'ouai…normalement plus utile non plus car le Start/Stop du moteur est intégré à notre nouveau *trackloader*

Nous avons au minimum une zone mémoire libre de **7C41C à 7C5C8+2 = $1AE** soit **!430 octets**

Concernant notre TrackLoader *d'AlphaOne*, nous l'avons modifié pour qu'il prenne en entré :
D0=Position dans la table pour récupérer la Longueur des data à lire ainsi que la piste de départ.
A0=Adr. Mémoire de destination reste inchangée.

On recharge le fichier en question, taper : `O 00, 7A120 7A120+4200` **puis** `LM 1:28_29, 7A120`
On efface la zone du *trackloader* original **taper :** `O 00, 7C41C 7C5C8`
On charge notre tracker d'AlphaOne modifié, préalablement crée (voir les derniers chapitres), **taper :** `LM 1:AlphaOne_v1.exe, 7C41C`

Dans le doute, on recode le dernier **BSR** présent juste après le code du trackloader (afin d'être sur)
Taper : `A 7C5CA`

```
^07C5CA BSR     7C2F8
```

<RETURN>

On applique les Patches que l'on a vue dans la section **'Récapitulatif : Patch, Checksum, Cheat'**
Taper : `A 7C344`

```
^07C344 RTS                     ; BSR 7C540 (GoTo #TrackZero)
```

<RETURN>

Puis avec la commande `M 7A72C` changer le `00 07 A1 20` par `00 00 40 00`

```
d 7C344
~07C344 BSR     0007C540 ── Avant
a 7C344
^07C344 RTS ──────────── Après
^07C346

M 7A72C Avant
:07A72C 00 07 A1 20 49 FA 15 94 47 FA 14 78 42 79 00 07 ... I...G..xBy..
M 7A72C Après
:07A72C 00 00 04 00 49 FA 15 94 47 FA 14 78 42 79 00 07 ... I...G..xBy..
:07A73C CA 42 30 2C 00 00 B0 7C FF FF 67 00 02 6E 4A 40 .B0,...|..g..nJ@
```

Avant de s'attaquer au 8 Appels, on va d'abord supprimer ceux qui ne sont plus utiles.

Taper : `A 7A442`
```
^07A442 NOP             ; BSR 7C540
^07A444 NOP             ; GoTo #TrackZero
^07A446 <RETURN>
```

Taper : `A 7A500`
```
^07A500 NOP             ; BSR 7C5A2
^07A502 NOP             ; GoTo #STOP FLOPPY
^07A504 <RETURN>
```

Taper : `A 7C340`
```
^07C340 NOP             ; BSR 7C586
^07C342 NOP             ; GoTo #Conf_Disque
^07C346 <RETURN>
```

Taper : `A 7A612`
```
^07A612 NOP             ; BSR 7C5A2
^07A614 NOP             ; GoTo #STOP FLOPPY
^07A616 <RETURN>
```

Taper : `A 7A43E`
```
^07A43E NOP             ; BSR 7C586
^07A440 NOP             ; GoTo #Conf_Disque
^07A442 <RETURN>
```

Taper : `A 7A71E`
```
^07A71E NOP             ; BSR 7C5A2
^07A720 NOP             ; GoTo #STOP FLOPPY
^07A722 <RETURN>
```

Taper : `A 7A6F6`
```
^07A6F6 NOP             ; BSR 7C586
^07A6F8 NOP             ; GoTo #Conf_Disque
^07A6FA <RETURN>
```

On en profite pour supprimer aussi tous les commandes de position de side.

`BSET #2, BFD100` = **Side lower** = **08 F9 00 02 00 BF D1 00**

`BCLR #2, BFD100` = **Side uppe** = **08 B9 00 02 00 BF D1 00**

Taper : `F 08 F9 00 02 00 BF D1 00, 7A120`

07A4C6 07A6FA

Puis, taper : `F 08 B9 00 02 00 BF D1 00, 7A120`

07A446 07A4DC 7C348

Taper : `A 7A4C6`
```
^07A4C6 NOP             ; BSET #2, BFD100
^07A4C8 NOP             ; Side Lower
^07A4CA NOP
^07A4CC NOP
^07A4CE <RETURN>
```

Taper : `A 7A4DC`
```
^07A4DC NOP             ; BCLR #2, BFD100
^07A4DE NOP             ; Side Upper
^07A4E0 NOP
^07A4E2 NOP
^07A4E4 <RETURN>
```

Taper : `A 7A6FA`
```
^07A6FA NOP             ; BSET #2, BFD100
^07A6FC NOP             ; Side Lower
^07A6FE NOP
^07A700 NOP
^07A442 <RETURN>
```

Taper : `A 7C348`
```
^07C348 NOP             ; BCLR #2, BFD100
^07C34A NOP             ; Side Upper
^07C34C NOP
^07C34E NOP
^07C350 <RETURN>
```

Taper : `A 7A446`
```
^07A446 NOP             ; BSET #2, BFD100
^07A448 NOP             ; Side Lower
^07A44A NOP
^07A44C NOP
^07A44E <RETURN>
```

Dans la même suite d'idée dans la '**partie 5 Analyse du trackloader'**, nous avons vu une configuration de DSKSYNC

```
7DC28   50470   MOVE.W  #4489,7E(A6)          ; DSKSYNC pattern=$4489
```

On va vérifier que quelque part dans le code chargé, il n'y a pas d'autre modification de faite
Comme on n'est pas sur **L'OPCODE** exact (car la valeur positionnée peut ne pas être 4489), on va chercher la chaine hexa **00 7E**
Taper : `F 00 7E, 7A120`

07A3CC 07A4C4 07A5F4 07C161 07C189 07C4D2 07C6DE 07C7F6

On regarde chaque appel, **taper :** `D 7A3CC-4` et ainsi de suite

```
f 00 7E, 7A120
Search from: 07A120 to: 080000
07A3CC  07A4C4  07A5F4  07C161  07C189  07C4D2  07C6DE  07C7F6
Ready.
d 7A3CC-4
^07A3C8 MOVE.W  #4489,7E(A6)  ; Aucun interet mais bon, il ne gene pas
d 7A4C4-4
^07A4C0 MOVE.W  #9245,7E(A6)  ; Conf Custom, a supprimer
d 7A5F4-4
^07A5F0 MOVE.W  #AA25,7E(A6)  ; Conf Custom, a supprimer
d 7C161-4
^07C15C ORI.B   #88,D0        ; Ne concerne pas DSKSYNC
d 7C189-4
^07C184 ORI.B   #42,D0        ; Ne concerne pas DSKSYNC
d 7C4D2-4
^07C4CE MOVE.W  #4489,7E(A6)  ; Notre nouveau Trackloader, on garde
d 7C6DE-4
^07C6DA ORI.L   #D0007E,(A4)  ; Ne concerne pas DSKSYNC
d 7C7F6-4
^07C7F2 ORI.L   #D0007E,(A4)  ; Ne concerne pas DSKSYNC
```

On supprime donc les 2 configurations custom en **7A4C0** et **7A5F0 :**

Taper : `A 7A4C0`

```
^07A4C0 NOP             ; Conf custom DSKSYNC=9245
^07A4C2 NOP
^07A4C4 NOP
^07A4C6 <RETURN>
```

Taper : `A 7A5f0`

```
^07A5F0 NOP             ; Conf custom DSKSYNC=AA25
^07A5F2 NOP
^07A5F4 NOP
^07A5F6 <RETURN>
```

On passe à la modification des Appels au *TrackLoader.*
On reprend notre tableau préalablement crée et on lui ajoute 3 colonnes, **D0, D1** et **A0** et une petite dernière pour le numéro d'appel.

D0 D'appel	A0	Est Appelé par	Nom du fichier ripé	Chargement de	Size Réel	D0 Track à lire	D1 Track de Départ
0	$58E44	$7A446 + TrkLoad(1)	30_49	Screenshot	$1E460	$17	$05
1	$8A48	$7A4C6 + TrkLoad(1)	01_13	N/A	$13AD8	$0F	$1C
2	$2E7C0	$7A4DC + TrkLoad(1)	01_28	N/A	$2A620	$1F	$2B
3	$1C520	$7A4F2 + TrkLoad(1)	29_40	Psysnosis Animation	$122A0	$0E	$4A
4	$400	$7A5F6 + TrkLoad(1)	41_61	N/A	$1FC98	$18	$58
5	$1FFD0	$7A604 + TrkLoad(1)	62_69	Chip Animation	$C1C1	$09	$70
6	$400	$7A6FA + TrkLoad(1)	14_22	Main Prg	$D9F8	$0A	$79
7	$14E72	$7A710 + TrkLoad(1)	23_27	Insert Disk2 Message	$7918	$06	$83

Comme expliquer un peu plus haut, le *trackloader* **AlphaOne** v2004 prends normalement en entrée la longueur des données à lire alors que le *trackloader* d'origine prends lui un nombre de track à lire.

La solution retenue est d'utiliser une table contenant les informations nécessaire (à savoir la **longueur à lire** en Byte et la **Track de départ**)
Il est assez facile de créer un bout de code pour faire le job mais où la placer en mémoire le tout le plus près possible du trackloader...
La réponse est simple, nous connaissons déjà une zone mémoire libre. Nous l'avons déjà utilisé cette zone mémoire sans problème pour coder notre routine de RIP des données. A savoir **7CA60**

Taper : `A 7CA60`

```
^07CA60 ASL.W   1#2,D0          ; Notre tableau fonctionnera en multiple de 4,
                                ; On utilise donc la commande ASL multiplié par 4
                                ; D0=D0*4

^07CA62 LEA     $200,A2         ; A2=$200, c'est l'adresse mémoire où l'on va stocker notre table

^07CA68 ADD.L   D0,A2           ; On ajoute le produit de D0 à cette adresse, ce qui nous donnes
                                ; Une position dans la table

^07CA6A MOVEQ   #0,D1           ; On nettoie D1
^07CA6C MOVE.B  (A2),D1         ; Et on lui copie dedans le 1er Byte de (A2) (Track de départ)

^07CA6E MOVE.L  (A2),D0         ; On récupère cette fois si le LongWord de (A2)
^07CA70 LSL.L   #8,D0           ; Petit traitement Pour supprimer
^07CA72 LSR.L   #8,D0           ; le Byte tout à gauche
^07CA74 LEA     76EBC,A2        ; L'adresse du Buffer MFM (identique à celle utilisé dans le code original)
^07CA7A MOVEQ   #0,D2           ; On nettoie D2 qui doit être a Zero pour le TrackLoader d'AlphaOne
^07CA7C JSR     7C41C           ; GoTo #TRACKLOADER base
^07CA80 RTS                     ; E.T Retour maison
^07CA82 <RETURN>
```

On s'attaque maintenant à la modification des appels au *TrackLoader(1)*

Appel N°1 : $7A446

On a déjà commencé à travailler avec celui-là, on peut commencer à partir de **$7A43E** (où nous avions mis des NOP)

Taper : A 7A44E

```
^07A44E MOVEA.L         #58E44,A0       ; Screenshot
^07A454 MOVEQ           #0,D0           ; 1er section dans notre tableau
^07A456 MOVEQ           #0,D1
^07A458 BSR             7CA60
^07A45C <RETURN>
```

Appel N°2, 3 et 4 : $7A4C6 $7A4DC $7A4F2

Taper : A 7A4CE

```
^07A4CE MOVEA.L         #8A48,A0        ; N/A
^07A4D4 MOVEQ           #1,D0           ; 2eme section dans notre tableau
^07A4D6 MOVEQ           #0,D1
^07A4D8 BSR             7CA60
^07A4DC <RETURN>
```

Taper : A 7A4E4

```
^07A4E4 MOVEA.L         #2E7C0,A0       ; Psysnosis Animation
^07A4EA MOVEQ           #2,D0           ; 3eme section dans notre tableau
^07A4EC MOVEQ           #0,D1
^07A4EE BSR             7CA60
^07A4F2 <RETURN>
```

Taper : A 7A4F2

```
^07A4F2 MOVEA.L         #1C520,A0       ; N/A
^07A4F8 MOVEQ           #3,D0           ; 4eme section dans notre tableau
^07A4FA MOVEQ           #0,D1
^07A5FC BSR             7CA60
^07A500 <RETURN>
```

Appel N°5 et 06 : $7A5F6 $7A604

Taper : A 7A5F6

```
^07A5F6 MOVEA.L         #400,A0         ; N/A
^07A5FC MOVEQ           #4,D0           ; 5eme section dans notre tableau
^07A5FE MOVEQ           #0,D1
^07A600 BSR             7CA60
^07A604 <RETURN>
```

Taper : A 7A604

```
^07A604 MOVEA.L         #1FFD0,A0       ; Chip Animation
^07A60A MOVEQ           #5,D0           ; 6eme section dans notre tableau
^07A60C MOVEQ           #0,D1
^07A60E BSR             7CA60
^07A612 <RETURN>
```

Appel N°7 et 08 : $7A6FA $7A710

Taper : A 7A702

```
^07A702 MOVEA.L         #400,A0         ; Main Prg
^07A708 MOVEQ           #6,D0           ; 7eme section dans notre tableau
^07A70A MOVEQ           #0,D1
^07A70C BSR             7CA60
^07A710 <RETURN>
```

Taper : A 7A710

```
^07A710 MOVEA.L         #14E72,A0       ; Insert Disk2 Message
^07A716 MOVEQ           #7,D0           ; 8eme section dans notre tableau
^07A718 MOVEQ           #0,D1
^07A71A BSR             7CA60
^07A71E <RETURN>
```

On sauve le tout, **taper : SM 1:28_29_Patched, 7A120 7A120+3070**
Et on-écrie ces data sur le disque, **taper : WT 2 3 7A120**

Il nous reste maintenant que le fichier **Main Prg** de notre tableau, à savoir le fichier ripé sous le nom de : **14_22** à patcher.
En effet si vous vous souvenez bien, il contient lui aussi un nouveau *trackloader* que l'on a appelé *trackloader(2)* **(voir chapitre 8)**
De plus il y a aussi des patches à effectuer dans cette zone mémoire **(voir chapitre 16)**

On reprend pour information le tableau avec uniquement les informations concernant ce fichier :

A0	Est Appelé par	Nom du fichier ripé	Chargement de	Size Réel	D0 Track à lire	D1 Track de Départ
$400	$7A6FA + TrkLoad(1)	14_22	Main Prg	$D9F8	$0A	$79

Taper : `O 00, 400 400+DC00` puis `LM 1:14_22, 400`

Analyse complète du *trackloader(2)* qui commence en $4066 **(revoir chapitre 8 si nécessaire)**

```
;================================        ; #TRACKLOADER Init
4066    BSR     42DE                    ; GoTo #MOTOR ON
406A    MOVE.W  #4489,7E(A6)            ; DSKSYNC pattern=$4489
4070    SUBQ.W  #1,D1                   ; D1=D1-1

                                        ; #TRACKLOADER base
4072    MOVEM.L D0-D1/A0,-(A7)          ; Sauvegarde des registres dans la pile
4076    BSR     41C8                    ; GoTo #READ TRACK DMA=186A
407A    MOVEM.L (A7)+,D0-D1/A0          ; On restaure les registres de la pile
407E    BSR     419A                    ; GoTo #Traitement MFM
4082    MOVEM.L (A7)+,D0-D1/A0          ; On restaure les registres de la pile
4086    CLR.W   D3                      ; On nettoie D3
4088    MOVEA.L A0,A1                   ; Copie l'adr. Destination mémoire présente en A0 Dans A1
408A    MOVE.W  #C1B,D4                 ; D4 = $C1B
408E    ADD.W   (A1)+,D3                ; ← Boucle qui va lire les 2 bytes de l'adr. A1
                                        ; et les ajouter à D3 puis A1=A1+2

4090    DBF     D4,408E                 ; → Boucle avec décrémentation D4,
                                        ; donc il va faire cette manip 3100
                                        ; oui 3100 car le dbf va jusqu'à -1 donc $C1C+1
                                        ; $C1B+1=$C1C= !3100 en décimal
                                        ;
                                        ; Comme on parle de word (2 bytes, donc XX XX),
                                        ; cela fait !6200  ($C1C*2) de donnée lue et traitée

4094    MOVE.W  (A1),D4                 ; On lit des deux dernier bytes(WORD contenue à l'adresse A1
                                        ; donc A0+6198 et on met ça dans D4

4096    CMP.W   D3,D4                   ; On compare cette valeur lue, en D4,
                                        ; avec ce que l'on calculé en D3
                                        ;
                                        ; On a donc ici une routine de check, de checksum

4098    BNE     4072                    ; Non, c'est que les données lues ne sont pas 'correctes',
                                        ; on repart pour un tour, GoTo #TRACKLOADER base
                                        ;
                                        ; On a donc ici une routine de check, de checksum 409A      ADDA.L  #1838,A0
                                        ; A0 = A0+$1838   // Taille d'une track ?
40A0    ADDQ.W  #1,D0                   ; D0 = D0+
40A2    DBF     D1,4072                 ; → Boucle avec décrémentation D1, c'est repartie pour un tour.
40A6    BSR     42FA                    ; E.T retour maison
40AA    RTS
;================================


;================================        ; #Init pattern spécifique
40AC    MOVEM.L D0-D1/A0,-(A7)          ; Sauvegarde D0, D1 et A0 dans la pile
40B0    MOVE.W  #7F00,9E(A6)            ; Pré-Conf. ADKON
40B6    MOVE.W  #B100,9E(A6)            ; Conf ADKON
40BC    MOVEA.L A0,A1                   ; A1=A0
40BE    CLR.W   D3                      ; On nettoie D3
40C0    MOVE.W  #C1B,D4                 ; D4 = $C1B
40C4    ADD.W   (A1)+,D3                ; ← Boucle qui va lire les 2 bytes de l'adr. A1
                                        ; et les ajouter à D3 puis A1=A1+2

40C6    DBF     D4,40C4                 ; → Boucle avec décrémentation D4,comme précédemment
40CA    MOVE.W  (A1),52(A5)             ; On lit des deux dernier bytes(WORD contenue à l'adresse A1
                                        ; Et on copie le tout dans 52(A5)

40CE    MOVE.W  D3,(A1)                 ; Copie du Word D3 à l'adresse pointé par A1
40D0    MOVE.L  A1,-(A7)                ; Sauvegarde de A1 dans la pile
40D2    BSR     4136                    ; GoTo #Traitement MFM sur pattern spécifique
40D6    BSR     40F6                    ; GoTo #READ TRACK DMA=1888
40DA    MOVEA.L (A7)+,A1                ; Restaure A1 de la pile
40DC    MOVE.W  52(A5),(A1)             ; Copie du word 52(A5) à l'adresse pointée par A1
40E0    MOVEM.L (A7)+,D0-D1/A0          ; Restaure D0, D1 et A0 de la Pile
40E4    BSR     42FA                    ; GoTo #
40E8    MOVE.W  #7F00,9E(A6)            ; Pré-Conf. ADKON
40EE    MOVE.W  #9500,9E(A6)            ; Conf ADKON
40F4    RTS                             ; E.T retour maison
;================================
```

```
;================================        ; #READ TRACK DMA=1888
40F6    BSR       4234                  ; GoTo #Position Reach ?
40FA    MOVE.W    #8210,96(A6)          ; DMACON (Disk DMA enable, enable All DMA)
4100    MOVE.W    #2,9C(A6)             ; INTREQ, Disk Block Finished Interupt
4106    BSR       42CA                  ; GoTo #CIA READY ?
410A    MOVE.L    #7CD38,20(A6)         ; DSKPTH // adr. MFM buffer=$7CD38
4112    MOVE.W    #4000,24(A6)          ; DMA disk off
4118    MOVE.W    #D888,24(A6)          ; DMA disk enable, Length DMA data=$1888
411E    MOVE.W    #D888,24(A6)          ; Deux fois pour démarrer le transfert DMA
4124    BSR       420E                  ; GoTo #DiskBlockFinished ?
4128    MOVE.W    #4000,24(A6)          ; DMA disk off
412E    MOVE.W    #10,96(A6)            ; Clear Disk DMA
4134    RTS                             ; E.T retour maison
;================================


;================================        ; #Traitement MFM sur pattern spécifique
4136    MOVE.W    D0,D5                 ; D5 = D0
4138    MOVEM.L   D0/A0,-(A7)           ; Sauvegarde D0 et A0 dans la pile
413C    MOVEQ     #7CD38,A1             ; A1 = Adresse du buffer MFM
4142    MOVEQ     #F,D0                 ; D0 = $F
4144    MOVE.W    #AAAA,(A1)+           ; ➔ Copie du word de masquage des bits pair à l'adresse pointé par A1 puis A1=A1+2
4148    DBF       D0,4144               ; ← Boucle de copie avec décrémentation D0, donc copie 16 fois
414C    MOVE.W    #4489,(A1)+           ; Copie du word $4489 à l'adresse pointée par A1, puis A1=A1+2
4150    MOVE.L    #552A2A55,(A1)+       ; Copie le LongWord 552A2A55 à l'adresse pointée par A1, puis A1=A1+4
4156    MOVE.W    #C1C,D7               ; D7 = $C1C
415A    MOVE.B    -1(A1),D6             ; Copie du Byte contenu à l'adresse -1(A1) dans D6
415E    MOVE.W    (A0)+,D0              ; ➔ Copie le word pointé par A0 dans D0 puis A0=A0+2
4160    EOR.W     D5,D0                 ;    Traitement des bits pair
4162    MOVE.W    D0,D1                 ;    D1 = D0
4164    ANDI.W    #AAAA,D0              ;    Décodage MFM bits pair de D0
4168    LSR.W     #1,D2                 ;    Décalage d'1 bit vers la droite sur D2
416A    ANDI.W    #5555,D1              ;    Décodage MFM bits impair de D1
416E    SWAP      D0                    ;    Swap D0, D0=xxxxYYYY devient YYYYxxxx
4170    MOVE.W    D1,D0                 ;    Copie du word D0 dans D1
4172    MOVE.L    D0,D1                 ;    Copie du LongWord D0 dans D1
4174    MOVE.L    D0,D2                 ;    Copie du LongWord D0 dans D2
4176    NOT.L     D1                    ;    Inverse tous les bits de D1
4178    ANDI.L    #55555555,D1          ;    Décodage MFM bits impair de D0
417E    ASL.L     #1,D1                 ;    Décalage de 1 bit vers la gauche sur le LongWord D1
4180    MOVE.L    D1,D3                 ;    D3 = D1
4182    ROXR.B    #1,D6                 ;    Rotation d'1 bit vers la droite du Byte D6
4184    ROXR.L    #1,D0                 ;    Rotation d'1 bit vers la gauche du LongWord D0
4186    EOR.L     D0,D1                 ;    Traitement des bits paire et impair #2
4188    AND.L     D3,D1                 ;    Traitement MFM
418A    OR.L      D1,D2                 ;    Traitement MFM
418C    MOVE.L    D2,(A1)+              ;    Copie du LongWord D2 à l'adresse pointée par A1 puis A1=A1+4
418E    MOVE.B    D2,D6                 ;    Copie du Byte D2 dans D6
4190    DBF       D7,415E               ; ← Boucle de copie avec décrémentation D7
                                        ;    il va faire cette manip 3100
                                        ;    3100 car le dbf va jusqu'à -1 donc $C1C+1
                                        ;    $C1B+1=$C1C= !3100 en décimal

4194    MOVEM.L   (A7)+,D0/A0           ; On restaure D0 et A0 de la pile et A7 = A7 + 4
4198    RTS                             ; E.T retour maison
;================================


;================================        ; #Traitement MFM
419A    MOVE.W    D0,D5                 ; D5 = D0, Numéro de piste
419C    MOVEM.L   D0/A0,(-A7)           ; Sauvegarde D0 et A0 dans la pile
41A0    MOVEA.L   #7CD3C,A1             ; A1 = 7CD3C = adr buffer MFM + 4
41A6    MOVE.W    #C1C,D7               ; Compteur D7=$C1C à savoir !3100, la moitié de !6200
                                        ; ($1838 en hexa)
                                        ;
                                        ; ➔ Début boucle, Traitement donc ici d'une 'track custom'
                                        ; d'une taille de $1838

41AA    MOVE.W    (A1)+,D0              ; Copie le word de l'adresse pointé par A1 (Word impair)
                                        ; vers D0 et A1=A1+2

41AC    MOVE.W    (A1)+,D1              ; Copie le word de l'adresse pointé par A1 (word pair)
                                        ; vers D1 et A1=A1+2

41AE    ASL.W     #1,D0                 ; Décalage de 1 bit vers la gauche
41B0    ANDI.W    #AAAA,D0              ; Décodage MFM bits pair de D0
41B4    ANDI.W    #5555,D1              ; Décodage MFM bits impair de D1
41B8    OR.W      D1,D0                 ; Traitement des bits pair et impair #1
41BA    EOR.W     D5,D0                 ; Traitement des bits pair et impair #2
41BC    MOVE.W    D0,(A0)+              ; Finalement, on copie le word MFM décodé vers l'adresse
                                        ; mémoire de destination puis A0=A0+2

41BE    DBF       D7,41AA               ; ← Boucle de copie avec décrémentation D7,
                                        ; on traite l'intégralité d'une 'track custom'

41C2    MOVEM.L   (A7)+,D0/A0           ; On restaure ce que l'on a sauvé dans la pile
41C6    RTS                             ; E.T retour maison
;================================
```

```
;================================        ; #READ TRACK DMA=186A
41C8    BSR     4234                    ; GoTo #Position Reach ?
41CC    MOVEA.L #7CD38,A1               ; A1 = Adresse du buffer MFM
41D2    MOVE.W  #8210,96(A6)            ; Conf DMACON (Disk DMA enable, enable All DMA)
41D8    MOVE.L  A1,20(A6)               ; ➔DSKPTH // adr. MFM buffer=A1=7CD38
41DC    CLR.W   24(A6)                  ;   DSKLEN,  DMA data Length clear
41E0    BSR     42CA                    ;   GoTo #CIA READY ?
41E4    MOVE.W  #4000,24(A6)            ;   DMA disk off
41EA    MOVE.W  #986A,24(A6)            ;   DMA disk enable, Length DMA data=$186A
41F0    MOVE.W  #986A,24(A6)            ;   Deux fois pour démarrer le transfert DMA
41F6    BSR     420E                    ;   GoTo #DiskBlockFinished ?
41FA    CMPI.L  #552A2A55,(A1)          ;   Compare le LongWord 552A2A55 avec la valeur à l'adresse de A1
4200    BNE     41D8                    ; ➔ Pas égale ? on repart pour un tour
4202    CLR.W   24(A6)                  ; conf DSKLEN,  DMA data Length clear
4206    MOVE.W  #10,96(A6)              ; Conf DMACON, Disk DMA clear
420C    RTS                             ; E.T retour maison
;================================

;================================        ; #DiskBlockFinished ?
420E    MOVE.W  1E(A6),D0               ; ➔A6=DFF000, DFF01E=INTREQR
4212    BTST    #1,D0                   ;   On test le bit 1 de de INTREQR, à savoir :
                                        ;   Level 1 Disk Block Finished Interupt

4216    BEQ     420E                    ; Test ? ➔ On boucle
4218    MOVE.W  #2,9C(A6)               ; clear DSKBLK(Disk Block Finished Interupt)
421E    RTS                             ; E.T retour maison
;================================

;================================        ; #FLOPPY READY ?
4220    MOVE.W  #BB8,D6                 ; Boucle d'attente D6 = $BB8, $BB8=!3000
4224    DBF     D6,4224                 ; ➔ ➔ Boucle d'attente
4228    BTST    #5,BFE001               ; CIA-A / PRA, On test le bit5, lecteur prêt ?
4230    BNE     4228                    ; Non ? on boucle jusqu'a que ce soit le cas.
4232    RTS                             ; E.T retour maison
;================================

;================================        ; #Position Reach ?
4234    CMP.W   1E(A5),D0               ; Compare le word à l'adresse de 1E(A5) avec D0
4238    BNE     423E                    ; Pas égale ? Alors on GoTo #Move Exter.
423C    RTS                             ; E.T retour maison
;================================

;================================        ; #Move Exter.
423E    BGT     426A                    ; GoTo #Move Inter.
4242    BSET    #1,BFD100               ; CIA-B / PRB, bit1 mis à 1, DIR=vers l'extérieur
424A    NOP                             ;
424C    NOP                             ;
424E    BCLR    #0,BFD100               ; CIA-B / PRB, bit0 mis à 0, STEP, pré/déplacement tête
4256    NOP                             ;
4258    NOP                             ;
425A    BSET    #0,BFD100               ; CIA-B / PRB, bit0 mis à 1, STEP, déplacement tête
4262    BSR     4220                    ; GoTo #FLOPPY READY ?
4264    SUBQ.W  #1,1E(A5)               ; On enlève 1 au pointeur de position de la tête situé en
                                        ; 1E(A5)    8C98+1E = $8CB6

4268    BRA     4234                    ; GoTo #Position Reach ?
;================================

;================================        ; #Move Inter.
426A    BCLR    #1,BFD100               ; CIA-B / PRB, bit1 mis à 0, DIR=vers l'intérieur
4272    NOP                             ;
4274    NOP                             ;
4276    BCLR    #0,BFD100               ; CIA-B / PRB, bit0 mis à 0, STEP, pré/déplacement tête
427E    NOP                             ;
4280    NOP                             ;
4282    BSET    #0,BFD100               ; CIA-B / PRB, bit0 mis à 0, STEP, déplacement tête
428A    BSR     4220                    ; GoTo #FLOPPY READY ?
428C    ADDQ.W  #1,1E(A5)               ; On ajoute 1 au pointeur de position de la tête situé en
                                        ; 1E(A5)    8C98+1E = $8CB6

4290    BRA     4234                    ; GoTo #Position Reach ?
;================================

;================================        ; #Retour T00
4292    BTST    #4,BFE001               ; CIA-A / PRA, On test le bit4, TK0, Tête sur T00 ?
429A    BEQ     42C4                    ; OUI ? GoTo #Update Position Track
429E    BSET    #1,BFD100               ; CIA-B / PRB, bit1 mis à 1, DIR=vers l'extérieur
42A6    NOP                             ;
42A8    NOP                             ;
42AA    BCLR    #0,BFD100               ; CIA-B / PRB, bit0 mis à 0, STEP, pré/déplacement tête
42B2    NOP                             ;
42B4    NOP                             ;
42B6    BSET    #0,BFD100               ; CIA-B / PRB, bit0 mis à 0, STEP, déplacement tête
42BE    BSR     4220                    ; GoTo #FLOPPY READY ?
42C2    BRA     4292                    ; GoTo #Retour T00
;================================
```

```
;================================        ; #Update Position Track
42C4    CLR.W   1E(A5)                  ; On nettoie en Byte le pointeur en 7DBB7
                                        ; (pointeur, Position Track)

42C8    RTS                             ; E.T retour maison
;================================

;================================        ; #CIA READY ?
42CA    MOVE.B  BFDD00,D6               ; D6 = BFDD00
42D0    MOVE.B  BFDD00,D6               ; ➔ D6 = icr register de CIA-B
42D6    BTST    #4,D6                   ;    Test du bit4 (FLAG) de icr
42DA    BEQ     42D0                    ; ← Interruption générée ? On boucle
42DC    RTS                             ; E.T retour maison
;================================

;================================        ; #MOTOR ON
42DE    BCLR    #7,BFD100               ; CIA-B / PRB, bit7 mis à 0, motor On
42E6    MOVEQ   #64,D6                  ; D6 = 64
42E8    BCLR    #3,BFD100               ; ← Boucle avec le CIA-B, bit3 à 0, DF0 sélectionné
42F0    DBF     D6,42E8                 ; ➔ Boucle avec décrémentation de D6
42F4    BSR     4220                    ; GoTo #FLOPPY READY ?
42F8    RTS                             ; E.T retour maison
;================================


;================================        ; #STOP FLOPPY
42FA    BSET    #7,BFD100               ; CIA-B / PRB, bit7 mis à 1, motor OFF
4302    BSET    #3,BFD100               ; CIA-B / PRB, bit3 mis a 1, SEL0, DF0 désélectionné
430A    MOVEQ   #64,D6                  ; D6 = 64
430C    BCLR    #3,BFD100               ; ← Boucle avec le CIA-B, bit3 à 0, DF0 sélectionné
4314    DBF     D6,430C                 ; ➔ Boucle avec décrémentation de D6
4318    BSET    #3,BFD100               ; CIA-B / PRB, bit3 mis a 1, SEL0, DF0 désélectionné
4320    RTS                             ; E.T retour maison
;================================
```

Il commence donc en **$4066** et fini en **$4322**, ce qui nous donnes : **$4322-$4066=$2BC = !700 Bytes**
On a donc ici largement la place pour le remplacer par le *trackloader* d'AlphaOne que l'on a modifié.

Tableau d'appel des sous routines du TrackLoader d'Origine

Adresse mémoire	Routine du TrackLoader d'origine concernée
4066	#TRACKLOADER Init
4072	#TRACKLOADER base
40AC	#Init pattern spécifique
40F6	#READ TRACK DMA=1888
4136	#Traitement MFM sur pattern spécifique
419A	#Traitement MFM
41C8	#READ TRACK DMA=186A
420E	#DiskBlockFinished ?
4234	#Position Reach ?
423E	#Move Exter.
426A	#Move Inter.
4292	#Retour T00
42C4	#Update Position Track
42CA	#CIA READY ?
42DE	#MOTOR ON
42FA	#STOP FLOPPY

On charge notre tracker *d'AlphaOne* modifié **version2** en lieu et place du **TrackLoader** d'origine identifié juste au-dessus.
Taper : `LM 1:AlphaOne_v2.exe, 4066`

Le **TrackLoader** d'origine utilise la plage d'adresse suivante : **$4066 ➔ $4320**
Le **TrackLoader** d'AlphaOne modifié **V2** utilise la plage d'adresse suivante : **$4066 ➔ $4220**

On applique les patchs vus en section **'Récapitulatif : Patch, Checksum, Cheat'**

$02C16	**14_22**	***BNE 2BEC***	**NOP**	CHECKSUM	Bug Start Level
$03172	**14_22**	***BNE 314E***	**NOP**	CHECKSUM	Bug entrer dans une Shop
$044DA	**14_22**	***BNE 44D2***	**NOP**	CHECKSUM	Sup. compteur Track=00
$03F38	**14_22**	***BRA 3F38***	**RTS**	PATCH	DeadLoop
$04904	**14_22**	***BGT 4910***	**NOP, NOP**	PATCH	Empêche GoTo #Highscore
$049AE	**14_22**	***BNE 4994***	**NOP**	PATCH	Bloqué dans #Highscore
$00CC6	**14_22**	***CLR.W $1E(A5)***	**NOP, NOP**	PATCH	Bug Start Level

Taper : A 2C16
```
^002C16 NOP             ; BNE 2BEC
^002C18
<RETURN>
```

Taper : A 3F38
```
^003F38 RTS             ; BRA 3F38
^003F3A <RETURN>
```

Taper : A 3172
```
^003172 NOP             ; BNE 314E
^003174
<RETURN>
```

Taper : A 4904
```
^004904 NOP             ; BGT 4910
^004906 NOP
^004908
<RETURN>
```

Taper : A 44DA
```
^0044DA NOP             ; BNE 44D2
^0044DC
<RETURN>
```

Taper : A CC6
```
^000CC6 NOP             ; CLR.W $1E(A5)
^000CC8 NOP
^000CCA
<RETURN>
```

Sans oublier la suppression des commandes de position de side.

`BSET #2, BFD100` = **Side lower** = **08 F9 00 02 00 BF D1 00**

`BCLR #2, BFD100` = **Side uppe** = **08 B9 00 02 00 BF D1 00**

Taper : F 08 F9 00 02 00 BF D1 00, 400

0004D4 000C8C 000CA8 000CDA 000D36 000D80 003F5E 003F74 004330

Puis, taper : F 08 B9 00 02 00 BF D1 00, 400

000C54 000C70 003E72 004BA2

Taper : A 4D4
```
^0004D4 NOP             ; BSET #2, BFD100
^0004D6 NOP             ; Side Lower
^0004D8 NOP
^0004DA NOP
^0004DC
<RETURN>
```

Taper : A 3F74
```
^003F74 NOP             ; BSET #2, BFD100
^003F76 NOP             ; Side Lower
^003F78 NOP
^003F7A NOP
^003F7C
<RETURN>
```

Taper : A C8C
```
^000C8C NOP             ; BSET #2, BFD100
^000C8E NOP             ; Side Lower
^000C90 NOP
^000C92 NOP
^000C94
<RETURN>
```

Taper : A 4330
```
^004330 NOP             ; BSET #2, BFD100
^004332 NOP             ; Side Lower
^004334 NOP
^004336 NOP
^004338
<RETURN>
```

Taper : A CA8
```
^000CA8 NOP             ; BSET #2, BFD100
^000CAA NOP             ; Side Lower
^000CAC NOP
^000CAE NOP
^000CB0
<RETURN>
```

Taper : A C54
```
^000C54 NOP             ; BCLR #2, BFD100
^000C56 NOP             ; Side Upper
^000C58 NOP
^000C5A NOP
^000C5C
<RETURN>
```

Taper : A CDA
```
^000CDA NOP             ; BSET #2, BFD100
^000CDC NOP             ; Side Lower
^000CDE NOP
^000CE0 NOP
^000CE2
<RETURN>
```

Taper : A C70
```
^000C70 NOP             ; BCLR #2, BFD100
^000C72 NOP             ; Side Upper
^000C74 NOP
^000C76 NOP
^000C78
<RETURN>
```

Taper : A D36
```
^000D36 NOP             ; BSET #2, BFD100
^000D38 NOP             ; Side Lower
^000D3A NOP
^000D3C NOP
^000D3E
<RETURN>
```

Taper : A 3E72
```
^003E72 NOP             ; BCLR #2, BFD100
^003E74 NOP             ; Side Upper
^003E76 NOP
^003E78 NOP
^003E7A
<RETURN>
```

Taper : A D80
```
^000D80 NOP             ; BSET #2, BFD100
^000D82 NOP             ; Side Lower
^000D84 NOP
^000D86 NOP
^000D88
<RETURN>
```

Taper : A 4BA2
```
^004BA2 NOP             ; BCLR #2, BFD100
^004BA4 NOP             ; Side Upper
^004BA6 NOP
^004BA8 NOP
^004BAA
<RETURN>
```

Taper : A 3F5E
```
^003F5E NOP             ; BSET #2, BFD100
^003F60 NOP             ; Side Lower
^003F62 NOP
^003F64 NOP
^003F66 <RETURN>
```

On n'oublie pas non plus la cherche de code sur DSKSYNC

Taper : `F 00 7E, 400`
On regarde chaque appel, **taper :** `D 53F-4` et ainsi de suite

Si vous avez nettoyer complètement la mémoire avant le chargement du fichier 14_22, vous devriez avoir ce résultat.
Comment dire…il y 'a du monde.

```
f 00 7E, 400
Search from: 000400 to: 080000
00053F  000555  00055F  00057D  000777  00078B  0007AB  0007D5  000D17  000D63
000D97  00411E  00454D  00456D  005555  005815  00946E  009470  009472  009492
0094B2  0094CE  0094D0  0094D2  009AEF  009AF1  009B11  009B31  009B4F  009B51
009CE0  009CFE  009D00  009D1E  009D20  009D42  009DDF  009DE3  009DE7  009DE9
009DFD  009DFF  009E03  009E05  009E07  009E09  009E1D  009E1F  009E23  009E25
009E27  009E29  009E41  009E43  009FEF  009FF1  00A011  00A031  00A04F  00A051
Ready.
```

Noter au passage, que même si configuration il y a, elle sera écrasée par celle de notre *Trackloader* mais bon...
Ne vous inquiétez pas, sur toute cette liste il n'y a que 2 adresses à retenir 😉

`$3FB8` et `$411E`

```
3FB8-4  MOVE.W  #4489,7E(A6)    ; à supprimer (pas obligatoire non plus de le supprimer)
411E-4  MOVE.W  #4489,7E(A6)    ; Configuration de notre Trackloader, on ne touche surtout pas.
```

Donc finalement, une seul, celle de **$3FB8** et encore... elle utilise un pattern standard **AmigaDos** , LOL

Histoire de ne pas avoir perdu notre temps pour rien, on va quand même la supprimer.

Taper : `A 3FB8-4`

```
^003FB4 NOP
^003FB6 NOP
^003FB8 NOP
^003FBA <RETURN>
```

On peut maintenant passer à la modification des Appels au *TrackLoader(2)*
Toujours avec le même de fonctionnement que précédemment (Utilisation de notre tableau d'appel)

On reprend les informations des Appels disque sur le Disque2

Est Appelé par	Adr. de chargement Mémoire réel	Nom du fichier ripé	Chargement de	Longueur en hexa	Nbr de TRACK AmigaDos Nécessaire	New Tracks cible
$4326 + TrkLoad(2)	$5AE3E	14_14	Signature Disque 2	$1838	!01	02 → 01
$4D4 + TrkLoad(2)	$D55A	08_12	Menu du jeu	$7918	!06	02 → 07
$D76 + TrkLoad(2)	$7541E	01_04	Menu choix Level	$60E0	!05	08 → 12
$C50+$CB4+TrkLoad(2)	$3D090	01_30	Level1	$18940	!33	13 → 45
$C6C+$CB4+TrkLoad(2)	$3D090	31_60	Level2	$1D518	!33	46 → 78
$C88+$CB4+TrkLoad(2)	$3D090	50_79	Level3	$1F248	!33	79 → 111
$CA4+$CB4+TrkLoad(2)	$3D090	20_49	Level4	$1E908	!33	112 → 144
$4BA2 + TrkLoad(2)	$1B312	61_73	Animation de Fin	$13AD8	!15	145 → 159

Sans oublier le doublon de $4D4 en $CDA
Même s'il n'a pas été appelé lors de nos tests, cela reste une bonne idée de le patcher quand même.

$CDA + TrkLoad(2)	$D55A	08_12	Menu du jeu	$7918	!06	02 → 07

On Passe à la modification de chaque appel :

Appel N°1 : $4326

Taper : A 4326

```
^004326 MOVEA.L         #5AE3E,A0       ; 9eme section dans notre tableau
^00432C MOVEQ           #8,D0           ; Signature Disque2
^00432E MOVEQ           #0,D1
^004330 <RETURN>
```

Appel N°2 : $4D4

Taper : A 4E4

```
^0004E4 MOVEA.L         #D55A,A0        ; 10eme section dans notre tableau
^0004EA MOVEQ           #9,D0           ; Menu du jeu
^0004EC MOVEQ           #0,D1
^0004EE BSR             4066
^0004F2 <RETURN>
```

Appel N°3 : $D76

Taper : A D76

```
^000D76 MOVEA.L         #D55A,A0        ; 11eme section dans notre tableau
^000D7C MOVEQ           #$A,D0          ; Menu Choix Level
^000D7E MOVEQ           #0,D1
^000D80 <RETURN>
```

Appel N°8 : $4BA2

Taper : A 4BAA

```
^004BAA MOVEQ           #$F,D0          ; 16eme section dans notre tableau
^004BAC MOVEQ           #0,D1           ; Animation de fin
^004BAE <RETURN>
```

Appel N°2 bis :

On n'oublie pas non plus notre 'doublon' de **$4D4** présent en **$CDA**

Taper : A CDA

```
^000CEA MOVEA.L         #D55A,A0        ; 10eme section dans notre tableau
^0004F0 MOVEQ           #9,D0           ; Menu du jeu
^0004F2 MOVEQ           #0,D1           ; En tout point identique à l'appel N°02 en $4D4
^0004F4 <RETURN>
```

Appel N°4, 5, 6 et 7 : $C50 $C6C $C88 $CA4

Taper : A C50

```
^000C50 BSR 4220                        ; Vers notre petit code
^000C54 MOVEQ           #$B,D0          ; 12eme section dans notre tableau
^000C56 MOVEQ           #0,D1           ; Level 1
^000C58 <RETURN>
```

Taper : A C6C

```
^000C6C BSR 4220                        ; Vers notre petit code
^000C70 MOVEQ           #$C,D0          ; 13eme section dans notre tableau
^000C72 MOVEQ           #0,D1           ; Level2
^000C74 <RETURN>
```

Taper : A C88

```
^000C88 BSR 4220                        ; Vers notre petit code
^000C8C MOVEQ           #$D,D0          ; 14eme section dans notre tableau
^000C8E MOVEQ           #0,D1           ; Level3
^000C90 <RETURN>
```

Taper : A CA4

```
^000CA4 BSR 4220                        ; Vers notre petit code
^000CA8 MOVEQ           #$E,D0          ; 15eme section dans notre tableau
^000CAA MOVEQ           #0,D1           ; Level4
^000CAC <RETURN>
```

Petit arrêt sur cette section car spécifique comme l'on à déjà vue dans ce tuto pour le chargement des LEVELs.
Tous les chargements de LEVEL vont se font en plusieurs étapes :

1. Configuration de **D0** et **D1**, puis **BRA** sur **$CB4**
2. Sauvegarde de **D0** et **A0** dans la pile
3. Remplissage/nettoyage de la zone mémoire **$6A200->$6A720**
4. On remet les valeurs de **A0** et **D0**
5. Appel du Trackloader et chargement du LEVEL
6. Patch du LEVEL chargé
7. Retour au code d'origine

Nous avons chargé et remplacé le *trackloader* original de l'adresse **$4066** jusqu'à **$4218**
Aussi notre code commencera logiquement en **$4220**

C'est ici que l'on va devoir modifier le code car nous avons supprimer les *ZERO-ADD* présent en fin de donnée de chaque LEVELs.
Et Appliquer aussi le Patch sur le Level Chargé.

Rappel du Patch en question

$6A084	**01_30** **20_49** **31_60** **50_79**	***BSR 6A22C***	**BRA 6A150**	PATCH	Empêche routineS T00 du Disk2 Patch nécessaire dans tous les fichiers LEVEL

OPCODE d'un **BRA 6A150** en position mémoire **$6A084** = **60 00 00 CA**

On va donc code quelque ligne à partir de **$4220** et changer les branchements en **$CB4**

```
d CB4
~000CB4 MOVEA.L #3D090,A0
~000CBA BSR     00004066
~000CBE BSR     0000434A
~000CC2 BSR     0000462E
~000CC6 NOP

a CB4
~000CB4 MOVEA.L #3D090,A0
~000CBA BSR     00004220
~000CBE NOP
~000CC0 NOP
~000CC2
```

→ Code original

→ On saute vers notre Patch

```
a 4220
~004220 MOVEM.L D0/A0,-(A7)
~004224 MOVEA.L #6A200,A0
~00422A MOVE.W  #51F,D0
~00422E MOVE.B  #0,(A0)+
~004232 DBF     D0,0000422E
~004236 MOVEM.L (A7)+,D0/A0
~00423A BSR     00004066
~00423E MOVE.L  #600000CA,0006A084
~004248 JMP     0000434A
```

- Notre Petit bout de code -
On Sauve D0 et A0 dan la pile
Et on remplie de 00 notre zone mémoire
$6A200 -> 6A720
On restaure D0 et A0 de la pile

→ GoTo #TrackLoader
→ Patch le Level chargé
→ On retourne au code original

Déjà, on change le code pour sauter vers notre petit bout de code.
Taper : A CB4

```
^000CB4 MOVEA.L #3D090,A0              ; On ne change rien
^000CBA BSR     4220                   ; On change par un BSR vers notre PATCH
^000CBE NOP
^000CC0 NOP
^000CC2 <RETURN>
```

Et ensuite le code en question :
Taper : A 4220

```
^004220 MOVEM.L         D0/A0,-(A7)          ; #Notre_Petit_Patch
^004224 MOVEA.L         #6A200,A0            ; On Sauve D0 et A0 dans la pile
^00422A MOVE.W          #51F,D0              ; A0=Adresse de départ pour le remplissage de Zero
^00422E MOVE.B          #0,(A0)+             ; D0=Longueur de Byte à copier -1
^004232 DBF             D0,422E              ; → On copie $00 à l'adresse de destination
^004236 MOVEM.L         (A7)+,D0/A0          ; ← Tant que compteur en D0 n'est pas égal à -1, on boucle
^00423A BSR             4066                 ; On restaure D0 et A0 de la pile
^00423E MOVE.L          #600000CA,$6A084     ; GoTo #TrackLoader pour le chargement du LEVEL
^004248 JMP 434A                             ; On patch le Level Chargé
^00424E <RETURN>                             ; On retourne au code d'origine
```

Il nous reste plus qu'a sauver tout ça sur le disque sauf que, il est impossible d'utiliser la commande **WT** de l'**AR** avec une zone mémoire **<$8400**

Ce n'est pas bien grave, on va d'abord sauver le tout sur une nouvelle disquette.
Enlever la disquette **bloodmoneyD1** du lecteur **DF1** et **insérer une nouvelle disquette, formater la si nécessaire**.

Taper : `SM 1:14_22_mod, 400 400+D9F8`

$D9F8 = !55800
$1600 = !5632
55800/5632=9,9
10 = $A
$1600 x $A = $DC00

On nettoie donc une zone mémoire de 10 Tracks, **taper :** `O 00, 20000 20000+DC00`
On recharge notre fichier, **taper :** `LM 1:14_22_mod, 20000`
Et on écrit le tout sur la bonne Track, **taper :** `WT !121 !10 20000`

Part 20 Création de la Table d'appel pour nos TrackLoaders

Il nous faut maintenant créer et écrire cette table d'appel sur notre disque cracké.

Voila un nouveau tableau qui regroupe les dernières informations collectées le long du tuto

D0 D'appel	A0	Nom du fichier ripé	Chargement de	Length en Hexa	Length en Décimal	D0 Track à lire	D1 Track de Départ
0	$58E44	30_49	Screenshot	$1E460	124000	$17	$05
1	$8A48	01_13	N/A	$13AD8	80600	$0F	$1C
2	$2E7C0	01_28	N/A	$2A620	173600	$1F	$2B
3	$1C520	29_40	Psysnosis Animation	$122A0	74400	$0E	$4A
4	$400	41_61	N/A	$1FC98	130200	$18	$58
5	$1FFD0	62_69	Chip Animation	$C1C1	49601	$09	$70
6	$400	14_22	Main Prg	$D9F8	55800	$0A	$79
7	$14E72	23_27	Insert Disk2 Message	$7918	31000	$06	$83
8	$D55A	14_14	Signature Disk2	$1600	5632	$01	$00
9	$1B312	08_12	Menu du jeu	$7918	31000	$06	$02
A	$5AE3E	01_04	Menu choix Level	$60E0	24800	$05	$08
B	$3D090	01_30	Level1	$2D1E0	184800	$33	$0D
C	$3D090	31_60	Level2	$2D1E0	184800	$33	$2E
D	$3D090	50_79	Level3	$2D1E0	184800	$33	$4F
E	$3D090	20_49	Level4	$2D1E0	184800	$33	$70
F	$7541E	61_73	Animation de Fin	$13AD8	80600	$15	$91

Comme prévu nous allons utiliser la zone libre en **$200** (pendant le jeu)

Taper : `RT 0 1 20000`

Comme nous l'avons lors de la création du *bootsecteur*, notre code se fini en **$2004A**

Logiquement, notre tableau commencera donc en code suivant le **$2004A**, à savoir **$20050**
Taper : `M 20050` puis taper la chaine hexa suivante

```
M 20050
:020050 05 01 E4 60 1C 01 3A D8 2B 02 A6 20 4A 01 22 A0 ..ä`..:.+.. J."
:020060 58 01 FC 98 70 00 C1 C1 79 00 D9 F8 83 00 79 18 X.ü.p...y.....y.
:020070 00 00 16 00 02 00 79 18 08 00 60 E0 0D 02 D1 E0 ......y...`.....
:020080 2E 02 D1 E0 4F 02 D1 E0 70 02 D1 E0 91 01 3A D8 .-..Ð...p.....:.
:020090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Début des données en **$20050** et fin des données **$2008F** et, ce qui nous donnes comme longueur **$3F** *($2008F-20050)*
On change notre compteur en **D0** par ***$3F*** (mis initialement à 01 au début de la création du *BootSecteur*)

Petit Rappel :

```
^020040 MOVE.W  #$1,D0                  ; D0=Longueur à copier en Byte
```

On change donc ceci, **taper :** `M 20050`

```
^020040 MOVE.W  #$3F,D0                 ; D0=Longueur total de notre table
^020044 <RETURN>
```

Et bien sûr on re-calcul le nouveau checksum du *bootblock* et à sauver le tout sur **DF0**
Taper : `BOOTCHK 20000` puis `WT 0 1 20000`

Part 21 Debug

On reboot l'amiga avec notre disquette `CR_BLOOD_MONEY_D1` dans le lecteur.
Tout semble se passer correctement (screenshot, anim, etc), jusqu'au dernier chargement où il ne se passe plus rien (écran noir)
Entrer dans l'**AR, taper :** `D`

On semble bloqué quelque part en **$3E16**

Si on regarde notre tableau crée lors du *chapitre 7 Rip Disk1*, le dernier appel au *TrackLoader* se fait en **$7A710**
Rebooter votre amiga, une fois sur la **'chip animation'**, **entrer dans l'AR et taper :** `BS 7A710` puis `X`

Assez rapidement notre *BreakPoint* est atteint. C'est une bonne chose car cela veut dire que tout fonctionne jusqu'ici.
On regarde ce que va faire le code après avoir effectué ce dernier chargement, **tapez** : `D 7A710` et **descendez dans le code**

```
Breakpoint raised at address: 0007A710  <-------- Notre BreakPoint Atteint
d 7A710
~07A710 MOVEA.L #14E72,A0  ]
~07A716 MOVEQ   #7,D0      ]------- Le dernier chargement du Disque1
~07A718 MOVEQ   #0,D1      ]
~07A71A BSR     0007C41C   ]
~07A71E NOP
~07A720 NOP
~07A722 LEA     7BAEA(PC),A0
~07A726 BSR     0007C640              Suivi d'un BSR en 7C640
~07A72A JMP     00000400              et pour finir un JMP en $400
;=======================================
```

On supprime tous nos *BreakPoints* et on va poser un nouveau en $400, **taper :** `BDA` puis `BS 400` sans oublier de retourner au code `X`

Après donc ce dernier chargement, notre *BreakPoint* en *$400* est atteint ce qui veux dire 3 choses :

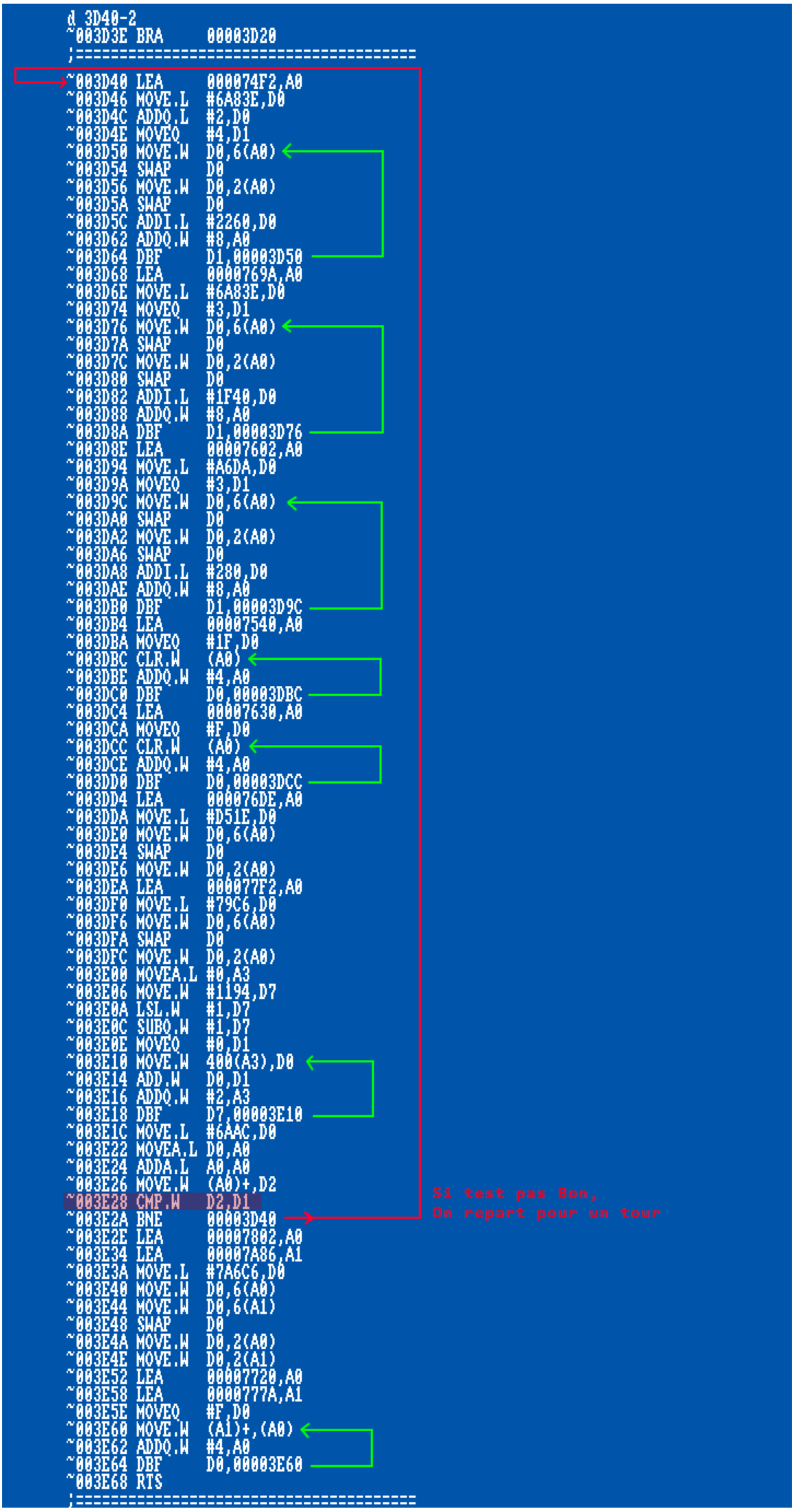
- Que le problème n'est pas dans ce dernier chargement.
- Que le problème n'est pas dans la/les routines en **7C640**
- Que le problème se situe après le **JMP en $400**

Logiquement, on regarde le code en 400, **tapez :** `D 400` et **descendez un peu dans le code**

```
d 400
~000400 LEA     0000040C,A0
~000406 MOVE.L  A0,00000084.S
~00040A TRAP    #1
~00040C MOVE.W  #2100,SR
~000410 LEA     00008C98,A5
~000416 MOVEA.L #400,A7
~00041C BSR     00003D40  ----------> Kesako ?
~000420 MOVEA.L #DFF000,A6
~000426 MOVE.L  4(A6),14(A5)
~00042C MOVE.W  #7DF0,96(A6)
```

La question est de savoir si le problème se situe dans cette sous-routine ou après.
Pour cela, c'est simple, il suffit de poser un *BreakPoints* en sortie de cette sous-routine et de voir s'il est atteint.
Taper : `BDA` puis `BS 420` puis `X`

On retombe sur notre blocage sur un écran noir, il y a donc quelque chose qui cloche pour nous dans cette sous-routine.
Entrer dans l'**AR** et **taper** : `D 3D40-2` puis **descendez jusqu'à trouver la fin de cette sous-routine.**


d 3D40-2
~003D3E BRA 00003D20
;=====================================
~003D40 LEA 000074F2,A0
~003D46 MOVE.L #6A83E,D0
~003D4C ADDQ.L #2,D0
~003D4E MOVEQ #4,D1
~003D50 MOVE.W D0,6(A0)
~003D54 SWAP D0
~003D56 MOVE.W D0,2(A0)
~003D5A SWAP D0
~003D5C ADDI.L #2260,D0
~003D62 ADDQ.W #8,A0
~003D64 DBF D1,00003D50
~003D68 LEA 0000769A,A0
~003D6E MOVE.L #6A83E,D0
~003D74 MOVEQ #3,D1
~003D76 MOVE.W D0,6(A0)
~003D7A SWAP D0
~003D7C MOVE.W D0,2(A0)
~003D80 SWAP D0
~003D82 ADDI.L #1F40,D0
~003D88 ADDQ.W #8,A0
~003D8A DBF D1,00003D76
~003D8E LEA 00007602,A0
~003D94 MOVE.L #A6DA,D0
~003D9A MOVEQ #3,D1
~003D9C MOVE.W D0,6(A0)
~003DA0 SWAP D0
~003DA2 MOVE.W D0,2(A0)
~003DA6 SWAP D0
~003DA8 ADDI.L #280,D0
~003DAE ADDQ.W #8,A0
~003DB0 DBF D1,00003D9C
~003DB4 LEA 00007540,A0
~003DBA MOVEQ #1F,D0
~003DBC CLR.W (A0)
~003DBE ADDQ.W #4,A0
~003DC0 DBF D0,00003DBC
~003DC4 LEA 00007630,A0
~003DCA MOVEQ #F,D0
~003DCC CLR.W (A0)
~003DCE ADDQ.W #4,A0
~003DD0 DBF D0,00003DCC
~003DD4 LEA 000076DE,A0
~003DDA MOVE.L #D51E,D0
~003DE0 MOVE.W D0,6(A0)
~003DE4 SWAP D0
~003DE6 MOVE.W D0,2(A0)
~003DEA LEA 000077F2,A0
~003DF0 MOVE.L #79C6,D0
~003DF6 MOVE.W D0,6(A0)
~003DFA SWAP D0
~003DFC MOVE.W D0,2(A0)
~003E00 MOVEA.L #0,A3
~003E06 MOVE.W #1194,D7
~003E0A LSL.W #1,D7
~003E0C SUBQ.W #1,D7
~003E0E MOVEQ #0,D1
~003E10 MOVE.W 400(A3),D0
~003E14 ADD.W D0,D1
~003E16 ADDQ.W #2,A3
~003E18 DBF D7,00003E10
~003E1C MOVE.L #6AAC,D0
~003E22 MOVEA.L D0,A0
~003E24 ADDA.L A0,A0
~003E26 MOVE.W (A0)+,D2
~003E28 CMP.W D2,D1
~003E2A BNE 00003D40
~003E2E LEA 00007802,A0
~003E34 LEA 00007A86,A1
~003E3A MOVE.L #7A6C6,D0
~003E40 MOVE.W D0,6(A0)
~003E44 MOVE.W D0,6(A1)
~003E48 SWAP D0
~003E4A MOVE.W D0,2(A0)
~003E4E MOVE.W D0,2(A1)
~003E52 LEA 00007720,A0
~003E58 LEA 0000777A,A1
~003E5E MOVEQ #F,D0
~003E60 MOVE.W (A1)+,(A0)
~003E62 ADDQ.W #4,A0
~003E64 DBF D0,00003E60
~003E68 RTS
;=====================================
Si test pas Bon,
On repart pour un tour

Ce code semble effectuer tout une suite d'opérations pour finir par un **CMP.W D2,D1** qui, finalement s'il n'est pas concluant, on repart pour un tour.

Lorsque nous sommes entrés dans l'AR sur l'écran noir, nous étions bloqués en **$3E16**, cela veut dire que les traitements précédemment ont bien fonctionnés.

Vue que l'on n'arrive pas jusqu'au **RTS** de fin, on est soit bloqué dans le **DBF** en **3E60➔ 3E64**, soit l'on n'arrive même pas jusqu'à ce code et on est bloqué par le **CMP D2,D1** qui nous fait reboucler en Dead-loop sur cette sous-routine

On va poser un *Breakpoint* en **$3E2A** afin d'être fixé sur le sujet, **taper :** `BDA` puis `BS 3E2A` puis `X`
Bimm, instantanément notre *Breakpoint* en **$3E2A** est atteint, c'est donc bien le **CMP D2,D1** qui pose un problème ou plus exactement, le **BNE** juste après. On va le désactiver.

Taper : `BDA` puis `A 3E2A`
```
^003E2A NOP
^003E2C NOP
^003E2E <RETURN>
```

Et pour être sûr que l'on sort bien de cette sous-routine, on ajoute un nouveau *Breakpoint* en **420** (adresse de retour après le **RTS**)
Taper : `BS 420` puis `X`

Assez rapidement notre *Breakpoint* en **$420** est atteint, on entre automatiquement dans l'AR.
Taper : `X`

Arg… Cette fois si nous avons droit à un gros plantage.
Rebooter votre Amiga et laisser le jeu effectuer tous ses chargements, nous voilà de nouveau bloqué sur notre écran noir.

Entrer dans l'AR, on va désactiver le **BNE** vue juste au-dessus
Taper : `A 3E2A`
```
^003E2A NOP
^003E2C NOP
^003E2E <RETURN>
```

Mais cette fois si on ne retourne pas au code, on va regarder ce qu'il fait ensuite, **tapez :** `D 420` descendez un peu dans le code.

```
d 420
~000420 MOVEA.L #DFF000,A6
~000426 MOVE.L  4(A6),14(A5)
~00042C MOVE.W  #7DF0,96(A6)
~000432 MOVE.W  #7FDF,9A(A6)
~000438 MOVE.W  #7FDF,9C(A6)
~00043E MOVE.W  #7F00,9E(A6)
~000444 MOVE.W  #9500,9E(A6)
~00044A MOVE.B  #7F,00BFED01
~000452 CLR.B   00BFEC01
~000458 BCLR    #6,00BFEE01
~000460 MOVE.B  #98,00BFED01
~000468 MOVE.B  00BFED01,D0
~00046E LEA     2DC6(PC),A1
~000472 MOVE.L  A1,00000068.S
~000476 BSR     00004B18
~00047A MOVE.L  #4ADE,0000006C.S
~000482 MOVE.W  #2100,SR
~000486 BSR     00002E56
~00048A MOVE.W  #FFFF,44(A6)
~000490 MOVE.W  #C028,9A(A6)
~000496 MOVE.W  #FFFF,00032604
~00049E BSR     0000364E
~0004A2 MOVE.W  #87E0,96(A6)
~0004A8 MOVE.W  #3,1C(A5)
~0004AE BSR     00002E56
~0004B2 BSR     00003E6A
~0004B6 LEA     00008144,A0
~0004BC BSR     00003B46
~0004C0 BSR     00003AB2
~0004C4 BSR     00004322
~0004C8 BSR     000042DE
~0004CC BSR     00004292
~0004D0 BSR     00003F70
~0004D4 NOP
~0004D6 NOP
```

Ce que l'on doit regarder, c'est bien sur les sauts. Ici, une floppé de **JSR.** La question c'est lequel ou lesquels pose problème...
On va couper la poire en deux en posant un *BreakPoint* en *$4B6* pour commencer et voir si l'on l'atteint.
Taper : `BDA` puis `BS 4B6` et `X`
Bon... mauvaise pioche, nous avons droit à un jolie crach system. Mais bon, cela nous fait avancer dans la recherche du problème.
On sait maintenant que le problème est avant l'adresse **$4B6,** ce qui nous laisse finalement que 5 **BSR**

476	**BSR**	**4B18**
486	**BSR**	**2E56**
49E	**BSR**	**364E**
4AE	**BSR**	**2E56**
4B2	**BSR**	**3E6A**

Rebooter votre Amiga et laisser le jeu effectuer tous ses chargements, nous voilà de nouveau bloqué sur notre écran noir.

Entrer dans l'AR, on va désactiver le **BNE** vue juste au-dessus
Taper : A 3E2A

```
^003E2A NOP
^003E2C NOP
^003E2E <RETURN>
```

Taper : BS 476 puis **BS 486** puis **BS 49E** puis **BS 4AE** puis **BS 4B2**
Sans oublier de retourner au code, **taper : X**

Dans l'ordre tous les *BreakPoint* sont atteints, donc arrive bien jusqu'en **$4B2** mais jamais en **$4B6**, c'est donc assez logiquement que nous allons regarder le **BSR 3E6A** en *$4B2*
Taper : D 3E6A

```
d 3E6A
~003E6A BSR     000042DE
~003E6E BSR     00004292
~003E72 NOP
~003E74 NOP
~003E76 NOP
~003E78 NOP
~003E7A BSR     00003E88
~003E7E BSR     00003ED8
~003E82 BSR     000042FA
~003E86 RTS
;=====================================
```

En **42DE** on sait déjà ce qu'il y a (voir chapitre **Création version crack Disque 1)**
C'est la sous-routine de **Motor-On**, le problème ne vient pas d'elle c'est sûr.

```
;================================        ; #MOTOR ON
42DE    BCLR    #7,BFD100               ; CIA-B / PRB, bit7 mis à 0, motor On
42E6    MOVEQ   #64,D6                  ; D6 = 64
42E8    BCLR    #3,BFD100               ; ← Boucle avec le CIA-B, bit3 à 0, DF0 sélectionné
42F0    DBF     D6,42E8                 ; → Boucle avec décrémentation de D6
42F4    BSR     4220                    ; GoTo #FLOPPY READY ?
42F8    RTS                             ; E.T retour maison
;================================
```

Suivant : **4292**
Celle-ci par contre est plus problématique car, d'origine c'est la sous-routine : **Retour-T00**

```
;================================        ; #Retour T00
4292    BTST    #4,BFE001               ; CIA-A / PRA, On test le bit4, TK0, Tête sur T00 ?
429A    BEQ     42C4                    ; OUI ? GoTo #Update Position Track
429E    BSET    #1,BFD100               ; CIA-B / PRB, bit1 mis à 1, DIR=vers l'extérieur
42A6    NOP                             ;
42A8    NOP                             ;
42AA    BCLR    #0,BFD100               ; CIA-B / PRB, bit0 mis à 0, STEP, pré/déplacement tête
42B2    NOP                             ;
42B4    NOP                             ;
42B6    BSET    #0,BFD100               ; CIA-B / PRB, bit0 mis à 0, STEP, déplacement tête
42BE    BSR     4220                    ; GoTo #FLOPPY READY ? ← Problème, cette sous-routine n'existe plus
42C2    BRA     4292                    ; GoTo #Retour T00
;================================
```

Plus problématique car elle fait appel à une routine qui n'existe plus...
En effet notre *TrackLoader* modifié remplit la zone mémoire **$4066 → $4214** ce qui en soit n'est pas un problème mais...
Rappelez-vous, nous avons coder un petit bout de code juste après la fin de notre *TrackLoader*, en **$4214** *(Patch chargement Level)*
Et nous avons écrasé donc le code en **$4220**. Forcément, cela explique le crash system.

On va désactiver à la source tous ces **JSR**, à savoir la suppression du **JSR 3E6A**
Taper : BDA puis **A 4B2**

```
^0004B2 NOP
^0004B4 NOP
^0004B6 <RETURN>
```

On retourne à la suite du code, **taper : G 4B2**

Bingo, nous voila maintenant sur l'écran **'PLEASE INSERT BLOOD MONEY DISK2'**

On va déjà modifier ça en dur sur la disquette, **taper :** `O 00, 20000 20000+DC00`
Puis on recharge nos données `RT !121 !10 20000`
Précédemment on a changé deux branchements par des **NOP**s en **$3E2A** et **$4B2** avec données chargées à partir de **$400**
Donc : $3E2A-$400= **$3A2A** et $4B2-$400=**$B2**

Maintenant, nous avons chargées nos données en **$20000,** ce qui nous donnes comme adresses à modifier :
$20000+$3A2A=**$23A2A**
$20000+$B2=**$200B2**

Taper : `A 23A2A`
```
^023A2A NOP
^023A2C NOP
^023A2E <RETURN>
```

Taper : `A 200B2`
```
^0200B2 NOP
^0200B4 NOP
^0200B6 <RETURN>
```

Et on re-écrie le tout sur le disque, **taper :** `WT !121 !10 20000`

Rebooter l'Amiga, tout se passe bien jusqu'à l'affichage **INSERT DISK2**
On ne touche à rien, on laisse le Disque Cracké N°1 dans le lecteur, **on appuie sur FIRE**
Le code part sur la lecture de la signature présente sur le disque et puisque ce n'est pas celle du Disque 2, reste sur ce menu.
Ok, jusqu'ici, tout est bon.

Changer le disque dans le lecteur DF0 par notre second disque de crack crée précédemment et appuyez sur fire
Le code part encore une fois sur la vérification de la signature, trouve cette fois-ci la bonne et…Arg…crash

Ok, on est repartie pour un **reboot.**
Re-inserez le disk 1 de notre version crackée dans DF0, rebooter votre amiga
Une fois sur l'affichage 'Insert Disk2', changer le disque en DF0 par le disk 2 de notre version crackée
Entrer dans l'AR, taper : `D` nous sommes quelque part en **$2D6A**
Continuer le désassemblage pour isoler la sous-routine où nous sommes :

```
;================================
¨002D64 MOVE.B  BFE001,D0             ; → CIA-A / PRA dans D0
¨002D6A NOT.B   D0                    ; Test d'appuie
¨002D6C ANDI.B  #C0,D0                ; sur le bouton FIRE
¨002D70 BEQ     2D64                  ; ← Tant que l'on n'appuie pas, on boucle
¨002D72 RTS                           ; Bouton Fire appuyer, on retourne au code
;================================
```

Logiquement, on regarde l'adresse de retour stocké dans la pile, **taper :** `R`

```
r
D0=000725D8 0000FFFF 000014AD 0000BE01  00000000 0000FFFE 00000000 0000FFFF
A0=000076BA 00008452 0006C662 00004650  00064462 00008C98 00DFF000 000003F8
PC = 00002D6A USP = 000018B2 SR = 2108 T=0 S=1 I=001 X=0 N=1 Z=0 V=0 C=0
m 3F8
:0003F8 00 00 43 26 00 00 04 C8 41 F9 00 00 04 0C 21 C8 ..C&....A.....!.
```

On jette un coup d'œil au code en **$4326**, code qui sera exécuté après l'appuie sur Fire, **taper :** `D 4326`

```
d 4326
~004326 MOVEA.L #5AE3E,A0     ; Adresse de chargement pour le TrackLoader
~00432C MOVEQ   #8,D0         ; D0 pour la lecture de notre Table
~00432E MOVEQ   #0,D1         ; qui nous donnes comme valeur ensuite
~004330 NOP                   ; D0=$1838
~004332 NOP                   ; D1=2
~004334 NOP
~004336 NOP
~004338 BSR     00004066              ; Appel du Trackloader
~00433C CMPI.L  #32323232,0005AE3E    ; Test si signature Disk2 présente
~004346 BNE     00004322              ; Pas le cas, on revient au test
~004348 RTS                           ; d'appuie sur Fire
;=======================================

                                      Hexa      : Ascii
                                      $32323232 = 2222
```

Ce qui nous intéresse est le code exécuté après la vérification de la signature, après le **RTS** en **$4348** Fire
Pour cela nous allons utiliser un *BreakPoint*, **taper :** `BS 4348` puis `X`
Et bien sûr **appuyez sur fire**

Notre *BreakPoint* **$4348** est rapidement atteint, on entre automatiquement dans l'**AR, taper :** `ST`
Nous sommes **$4C8**, **taper :** `D 4C8` **et descendez un peu dans le code**

```
d 4C8
~0004C8 BSR     000042DE
~0004CC BSR     00004292
~0004D0 BSR     00003F70
~0004D4 NOP
~0004D6 NOP
~0004D8 NOP
~0004DA NOP
~0004DC MOVE.L  #732,0000006C.S
~0004E4 MOVEA.L #D55A,A0
~0004EA MOVEQ   #9,D0
~0004EC MOVEQ   #0,D1
~0004EE BSR     00004066
```

Prochain Appel au TrackLoader
Avec A0=D55A
Donc en l'occurence, le menu du jeu

Bon, Il est facile de comprendre que le problème vient d'un ou des **BSR** avant l'appel au *TrackLoader* vue que, cet appel n'est pas effectué.
Si on reprend notre tableau listant les Appel au code du *TrackLoader* original

Adresse mémoire	Routine du TrackLoader d'origine concernée		Adresse mémoire	Routine du TrackLoader d'origine concernée
4066	#TRACKLOADER Init		**4234**	#Position Reach ?
4072	#TRACKLOADER base		**423E**	#Move Exter.
40AC	#Init pattern spécifique		**426A**	#Move Inter.
40F6	#READ TRACK DMA=1888		**4292**	**#Retour T00**
4136	#Traitement MFM sur pattern spécifique		**42C4**	#Update Position Track
419A	#Traitement MFM		**42CA**	#CIA READY ?
41C8	#READ TRACK DMA=186A		**42DE**	**#MOTOR ON**
420E	#DiskBlockFinished ?		**42FA**	#STOP FLOPPY

Le **BSR 4292** en **$4C8** ne sert plus à rien. En effet notre *TrackLoader* a déjà une sous-routine de Motor On/Off
Le **BSR 42DE** en **$4CC est problématique** et non utile, il est aussi à supprimer

Quand au dernier appel en **$4D0**, **taper :** `D 4D0`

```
d 4D0
~0004D0 BSR     00003F70
~0004D4 NOP
```

A **désactiver aussi** finalement. Ce qui nous donnes finalement 3 **BSR** à désactiver, **taper :** `A 4C8`

```
^0004C8 NOP             ; Désactivation du BSR 42DE
^0004CA NOP
^0004CC NOP             ; Désactivation du BSR 4292
^0004CE NOP
^0004D0 NOP             ; Désactivation du BSR 3F70
^0004D2 NOP
^0004D4 <RETURN>
```

Et on retourne au code original, **taper :** `X`
Après un court chargement, nous sommes maintenant sur le Menu Principal du Jeu.
Appuyer sur **F1**, lancer une partie.
Tout semble fonctionner, le Level1 c'est chargé et c'est lancé.

Il nous reste plus que, encore une fois, modifier cela en dur sur notre Disquette.
Re-inserez le disk 1 de notre version crackée dans DF0

Taper : `O 00, 20000 20000+DC00`
On recharge nos données `RT !121 !10 20000`
On a donc mis **6 NOP** à partir de l'adresse **$4C8** avec données chargées à partir de **$400**
Donc : $4C8-$400= **$C8**

Maintenant, nous avons chargées nos données en **$20000, c**e qui nous donnes comme adresses à modifier :
$20000+$C8=**$200C8**

Taper : `A 200C8`

```
^0200C8 NOP
^0200CA NOP
^0200CC NOP
^0200CE NOP
^0200D0 NOP
^0200D2 NOP
^0200D4 <RETURN>
```

Et on re-écrie le tout sur le disque, **taper :** `WT !121 !10 20000`

On reboot et on test le jeu.
Tout semble bien fonctionner, aucun problème pour le chargement des Levels.
En utilisant les CheatMode il est facile de tester le jeu en entier. Aucun souci apparent, on termine le jeu sans aucun problème.
Plus de Check de la **T00**, plus d'entrée dans les HighScores, l'animation de fin fonctionne bien.
Tout semble bon mais...de retour au menu, on peut s'apercevoir qu'au bout d'un certain temps, le jeu passe en mode 'demo' (Fonctionnement normal du jeu) et lance le Level1. Sauf que dans notre cas...nous avons droit à un jolie plantage.

Alors plusieurs solutions :
1 – Chercher pourquoi ça plante !
2 – Désactiver le lancement du mode 'démo'

Comme ce tuto est déjà assez complet on va plus s'orienter vers la facilité à savoir, désactiver le lancement du mode démo.

Rebooter votre Amiga, passer les diverses animations, insérez le disk2, Bref...**arrivez jusqu'au Menu général du jeu.**

Il y a forcément un compteur. Ce compteur peu utiliser une zone mémoire comme un registre.
On va commencer par le plus simple, la vérification visuelle des registres. **Entrer dans l'AR, taper :** `R` puis `X`
Attendez 2 secondes, **entrer de nouveau dans l'AR, taper :** `R` puis `X`
Encore une fois, Attendez 2 secondes, **entrer de nouveau dans l'AR, taper :** `R` puis `X`
Attendez 1 secondes, **entrer de nouveau dans l'AR**

Voilà ce que l'on peut observer :

```
r
D0=00072500 0000FFFF 00014A5A 0000BE01  00000000 0000FFFE 00000000 00000176
A0=000076BA 00008452 0006BC46 00004650  00063A46 00008C98 00DFF000 000003F8
PC = 00002E88 USP = 000018B2 SR = 2104 T=0 S=1 I=001 X=0 N=0 Z=1 V=0 C=0
x
No known virus in memory!
Ready.
r
D0=00072500 0000FFFF 00014A5A 0000BE01  00000000 0000FFFE 00000000 0000013F
A0=000076BA 00008452 0006BC46 00004650  00063A46 00008C98 00DFF000 000003F8
PC = 00002E88 USP = 000018B2 SR = 2104 T=0 S=1 I=001 X=0 N=0 Z=1 V=0 C=0
x
No known virus in memory!
Ready.
r
D0=00072500 0000FFFF 00014A5A 0000BE01  00000000 0000FFFE 00000000 000000E4
A0=000076BA 00008452 0006BC46 00004650  00063A46 00008C98 00DFF000 000003F8
PC = 00002E84 USP = 000018B2 SR = 2104 T=0 S=1 I=001 X=0 N=0 Z=1 V=0 C=0
```

On peut clairement voir qu'uniquement **D7** est décrémenté, est-ce notre compteur ?
Il existe plusieurs façons de décrémenter un registre, les deux plus simples étant un **SUB** ou un **DBF**
On commence par recherche **l'OPCODE** d'un **SUBQ.W** sur **D7,** en l'occurrence **: 53 47**

Taper : `F 53 47`

```
f 53 47
Search from: 000000 to: 080000
000BE0  000ED6  0030FC  00312A  003E0C
```

On pose tout simplement des **BS** sur chaque adresse, **taper :**
`BS BE0` puis `BS ED6` et `BS 30FC` sans oublier `BS 312A` et finalement `BS 3E0C`
Il est temps pour nous de retourner au code du jeu, **taper :** `X`
Notre *BreakPoint* en **$BE0** est de suite atteint, on retourne au code, **taper :** `X`
Aucun autre *BreakPoint* est atteint et le mode 'démo' se lance et plante notre code.
Rebooter l'Amiga, retourner sur le Menu principal du jeu, entrer dans l'AR
Taper : `A BE0`

```
000BE0 NOP
000BE2 <RETURN>
```

Sans oublier de retourner au code, **taper :** `X`
On attend...encore...encore… Rien ne se passe.
Notre compteur est donc bien sur **D7** et il se décrémente par un **SUBQ** en **$BEO**
$BE0 c'est dans la zone de chargement de notre fichier ripé **14_22** (Track disk2=!121, longueur !10, voir précédemment)

Bon, on va patcher ceci 😉
Taper : `O 00, 20000 20000+DC00`
On charge nos données, **Insérez notre Disque1 cracké**, **tapez :** `RT !121 !10 20000`
Et on écrit le tout sur la bonne Track, **taper :** `WT !121 !10 20000`
L'adresse original à modifier est **$BE0** pour des données chargées en **$400**, ce qui nous donnes comme delta : $BE0-$400= **$7E0**
Taper : `D 20000+7E0`

```
¨0207E0 SUBQ.W #1,D7
```

Ok, c'est la bonne adresse
Taper : `A 20000+7E0`

```
¨0207E0 NOP
¨0207E2 <RETURN>
```

On re-écrie le tout sur le disque à l'emplacement en question, **taper :** `WT !121 !10 20000`
Rebooter votre Amiga, retourner au Menu principal du jeu et…impeccable, plus de lancement du mode 'démo'

Part 22 Modification du trackloader d'AlphaOne v1

Le *trackloader* d'origine fonctionne uniquement sur une face (sélectionné avant son appel).
Nous utilisons dans ce tuto une méthode standard pour la re-écriture des disques crackés.
Il nous faut donc un *trackloader* au fonctionnement au standard **AmigaDos** et de petite taille.
Celui d'AlphaOne v2004 (404 Bytes) fera très bien l'affaire.

Quelques modification ont été faites, celles-ci sont commentées dans le code ci-dessous.

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

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

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

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

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

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

TRACKLOADER:
                MOVEM.L  D0-D7/A0-A6,-(A7) ; ADD Save All registry

;               LEA $DFF000,A6                  ; Plus nécessaire car déjà et en A6 par le code Original
                                                ; du jeu Blood Money

                LEA      $BFD100,A4             ; DRIVESELECT REGISTER
;               LEA      $BFE001,A5             ; DRIVESTATUS REGISTER
                                                ; Mauvaise idée de modifié A5 car il est utilisé uniquement en lecture
                                                ; par le code du trackloader Original.
                                                ; Il indique une position mémoire utilisée par diverses commandes

                MOVEQ    #0,D7                  ; D7 = BYTECOUNTER



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

;STEPHEADZERO:  BTST     #4,(A5)                ; On change le code pour une utilisation en accès direct
                                                ; donc pour ne plus utiliser A5

STEPHEADZERO:   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 ; ADD restore registry
                RTS

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

;DISKREADY:     BTST     #5,(A5)                  ; On change le code pour une utilisation en accès direct
                                                  ; donc pour ne plus utiliser A5

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

TRACKLOADERENDE:

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

On compile le tout et on sauve de l'étiquette **TRACKLOADER** a l'étiquette **TRACKLOADEREND**
***** Revoir les autres tutos concernant l'utilisation ASM sous WinUAE

Fichier sauvé sous le nom : **AlphaOne_v1.exe**

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

```
48 E7 FF FE 49 F9 00 BF D1 00 7E 00 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 23 Modification du trackloader d'AlphaOne v2

Code identique à la v1 que l'on a modifié sauf que celui intègre directement en son code la récupération des variables dans notre Table crée pour l'occasion.

Pourquoi ? Car tout simplement nous avons plus de place mémoire dans la seconde partie.
Le **trackloader(2)** original occupe une place mémoire plus importante que le **trackloader(1)** original

De plus, la v2 utilise une autre adresse mémoire pour le Buffer MFM
Sinon, c'est exactement le même code.

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

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

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

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

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

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

TRACKLOADER:
                MOVEM.L  D0-D7/A0-A6,-(A7) ; ADD Save All registry

                ASL.W    #2,D0                  ; Récupération des informations dans notre TABLE
                LEA      $200,A2                ; Présente en 200
                ADD.L    D0,A2
                MOVEQ    #0,D1
                MOVE.B   (A2),D1
                MOVE.L   (A2),D0
                LSL.L    #8,D0
                LSR.L    #8,D0
                MOVEQ    #0,D2
                LEA      $7CD38,A2

;               LEA $DFF000,A6                  ; Plus nécessaire car déjà et en A6 par le code Original
                                                ; du jeu Blood Money

                LEA      $BFD100,A4             ; DRIVESELECT REGISTER
;               LEA      $BFE001,A5             ; DRIVESTATUS REGISTER
                                                ; Mauvaise idée de modifié A5 car il est utilisé uniquement en lecture
                                                ; par le code du trackloader Original.
                                                ; Il indique une position mémoire utilisée par diverses commandes

                MOVEQ    #0,D7                  ; D7 = BYTECOUNTER



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

;STEPHEADZERO:  BTST     #4,(A5)                ; On change le code pour une utilisation en accès direct
                                                ; donc pour ne plus utiliser A5

STEPHEADZERO:   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 ; ADD restore registry
                  RTS

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

;DISKREADY:       BTST     #5,(A5)                  ; WAIT FOR DISK-READY
DISKREADY:        BTST     #5,$BFE001               ; Direct acces, don't use A5
                  BNE.B    DISKREADY                ; ====================
                  RTS

TRACKLOADERENDE:

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

On compile le tout et on sauve de l'étiquette **TRACKLOADER** a l'étiquette **TRACKLOADEREND**
***** Revoir les autres tutos concernant l'utilisation ASM sous WinUAE

Fichier sauvé sous le nom : **AlphaOne_v2.exe**

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

```
48 E7 FF FE E5 40 45 F9 00 00 02 00 D5 C0 72 00 12 12 20 12 E1 88 E0 88 74 00 45 F9 00 07 CD 38
74 00 49 F9 00 BF D1 00 7E 00 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
```