



Tutoriel

Protection

Auteur Original

Soumit par (sur flashtro.com)

Version

Vérification/Correction

AmigaCracking : R-TYPE II

MFM (multi file)

AlphaOne

Musashi9 [2013-06-05]

02/01/2018 [Gi@nts](#)

V2, Testé et fonctionnel de A à Z

**\* R-TYPE 2 CRACK TUTORIEL \***

## Table des matières

Matériels nécessaire .....	3
Général Info .....	3
Agencement des disquettes Amiga v1.1 .....	5
WinUAE.....	7
Part 1 X-Copy .....	8
Part 2 Analyse de l'image IPF .....	9
Part 3 Début d'analyse du chargement du jeu .....	10
Part 4 Analyse bootblock .....	11
Part 5 Analyse et compréhension de la routine de Trackloading .....	14
Part 6 Tableau des registres lors de l'appel de la routine de <i>trackloader</i> .....	17
Part 7 Extra - Taille d'une piste .....	21
Part 8 Extra - Taille d'un secteur .....	22
Part 9 Calcul avant Rip des pistes .....	23
Part 10 Ripage des pistes .....	24
Part 11 Création de notre propre loader .....	25
Part 12 Binaire du SectorLoader 'rncloader.bin' de Rob Northen.....	28

## Matériels nécessaire

- Un Amiga 500 avec 1 Mega de Chip et 512Ko de slow ou l'émulateur WINUAE.  
Un Amiga 1200 est quand même fortement conseillé, ces 2méga de chip ne sont pas négligeable et de plus il fait fonctionner HRTmon sans problème. (Il est aussi possible de passer par WinUAE bien sûr)
- Une carte ACTION REPLAY MKIII si l'on passe par un Amiga réel ou ça ROM Image si on passe par WinUAE.
- Le jeu Original R-TYPE II ou son image CAPS (SPS 2065)
- Le logiciel Xcopy-Pro en disquette ou image disk selon configuration H/W.
- Le logiciel ASM-ONE v1.20 ou supérieur.
- Le binaire *rmcloader.bin* du SectorLoader de Rob Northen. (Disponible dans ce tuto sous forme Hexa)
- Optionnel mais plus que conseillé : L'utilisation de HRTmon soit en passant par un amiga Réel (ex Amiga1200 ou Amiga 500 avec 1 Mega de chip (et slow désactivable) ou bien sûr, passer par ça ROM Image si on passe par WinUAE

## Général Info

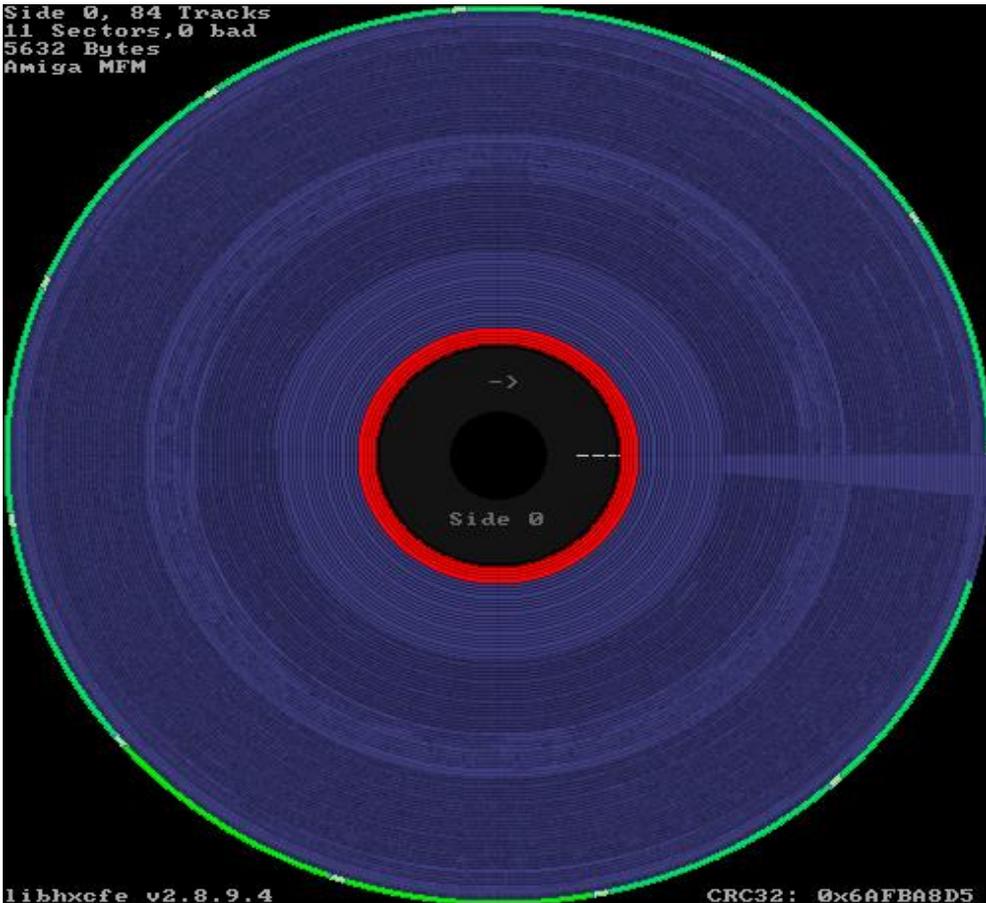
Ce tutoriel Français est basé sur le tutoriel original de Alpha One.

Néanmoins, ce document n'est PAS une traduction mot par mot de celui-ci mais plus une nouvelle version.

Basé sur l'original mais avec des nouvelles informations et parfois une autre approche de la problématique.

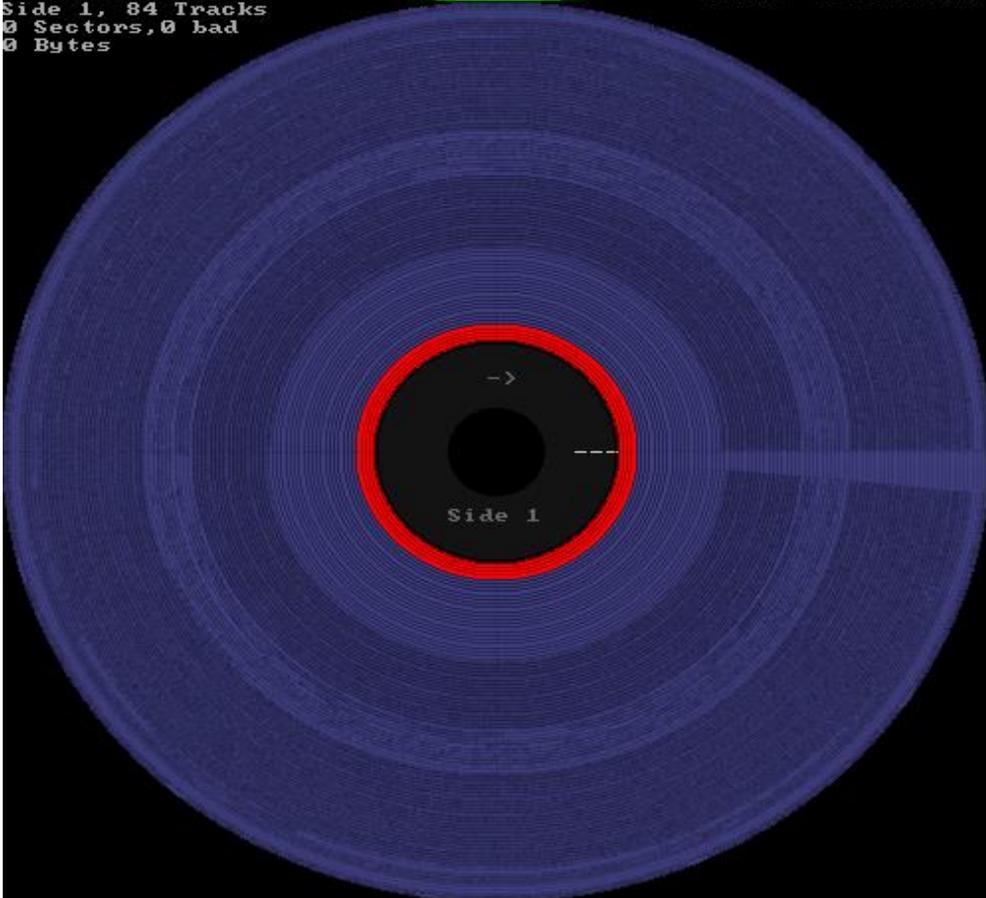
Bon tuto.  
Gi@nts

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



libxcfe v2.8.9.4  
Side 1, 84 Tracks  
0 Sectors, 0 bad  
0 Bytes

CRC32: 0x6AFBA8D5



## Agencement des disquettes Amiga v.1.1

### En France :

On utilise des termes comme : *piste, bloc, secteur, face...*

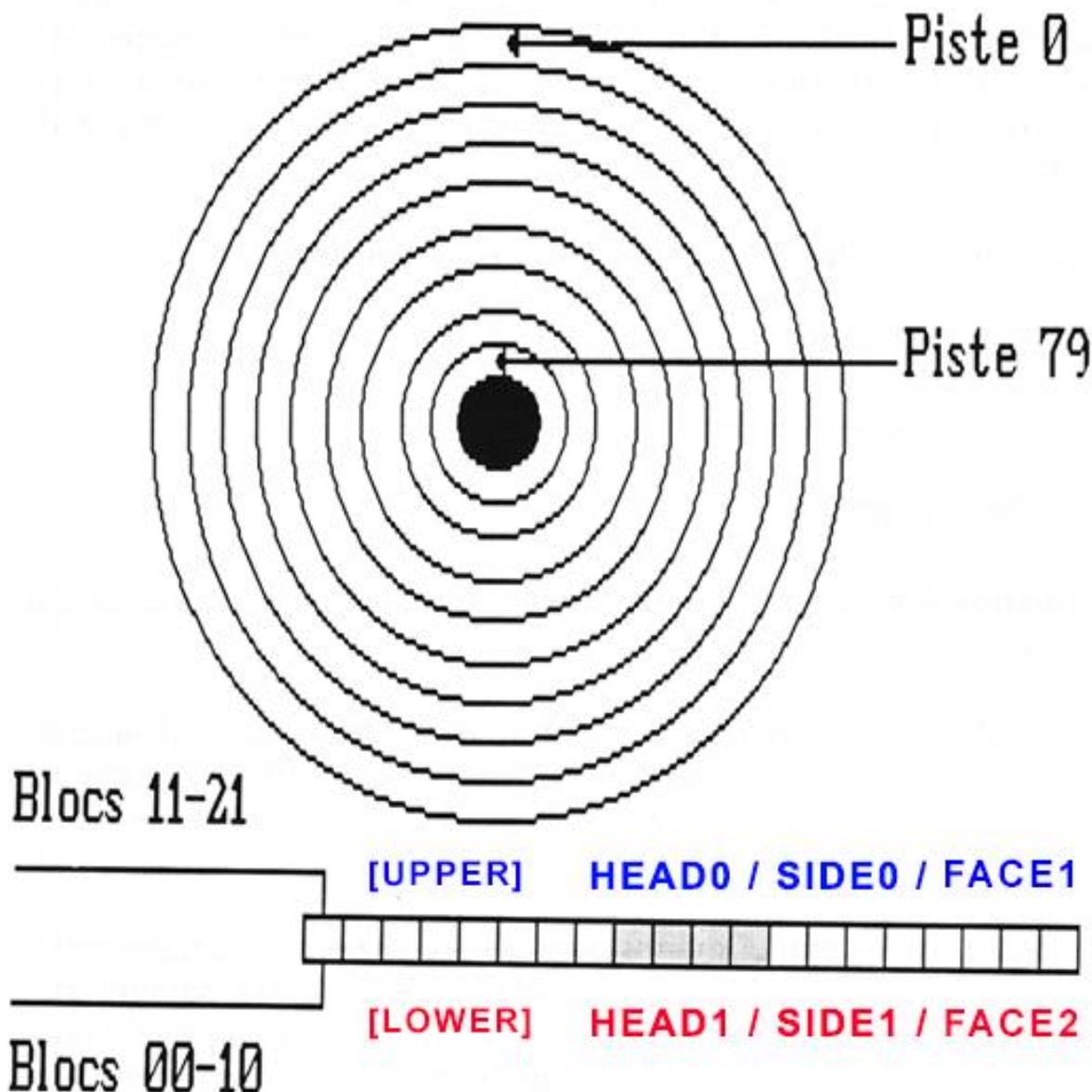
**Piste : 0 à 79** Certaines disquette pousse jusqu'à 81 voire 82 pistes mais le standard reste quand même 80 pistes (de 0 à 79)

**Face : 0/1 ou 1/2 ou A/B**, Dessus ou dessous tout simplement. Sur Amiga nous avons deux faces utilisées sur 99% des jeux.

**Chaque piste**, pour un format standard 'AmigaDOS' est composée de plusieurs *bloc* ou *secteur*, en général 11 **par face**.

Le terme piste peut désigner l'ensemble d'une piste (les deux 'side' du disque), ou uniquement une 'side' d'une piste.

Une piste standard *amigados* est découpée en plusieurs partie appelé **bloc, secteur, sector**.



### Dans d'autre pays :

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

Le terme **track** par exemple que l'on aurait vite fait de traduire 'piste' ne colle pas forcément à notre description française.

En général, le terme **tracks** désigne toujours une position sur la disquette mais **elle va de 0 à 159** (soit 160 **tracks**)

Le maximum étant 160 et non 80 car on a deux faces bien sûres, en fait, elle correspond à une piste sur une face.

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

C'est le terme **cylindre** qui 'colle' plus à notre définition française de **piste**.

En effet, il est courant d'utiliser le terme **cylindre** pour désigner une position sur la disquette de 0 à 79.

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

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

La taille d'une piste ayant une valeur physiquement maximum.

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

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

Il existe aussi un autre type d'appellation utilisé par exemple par le logiciel **MFM-Warp** de Ferox\*

*\*C'est un programme qui 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 1<sup>er</sup> 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 1<sup>er</sup> secteur de la seconde piste mais le 1<sup>er</sup> secteur *de la face suivante*. (voir image ci-dessus)

En format **Amigados**, la taille d'un secteur est de **512 octets**

Ce qui nous donne comme taille disponible : 512\*11 secteurs\*80 pistes\*2 faces = 901 120 octets soit 880Ko

Une 'track' AmigaDos a une taille de 512 \* 11 = **5632** en décimal soit **\$1600 octets**

### **Mise en application sous l'AR :**

Il existe deux commandes sous l'AR qui permettent de charger 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 donne !32 en décimal **mais** il indique une track (de 0 à 159) **PAS en piste**. **Pour avoir l'équivalent en piste** on divisera donc par 2 (car deux faces).

\$20/2=\$10 = !16 (en décimal donc) et comme il n'y a pas de 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 donnees (**taille à lire =2**), on continue la copie.

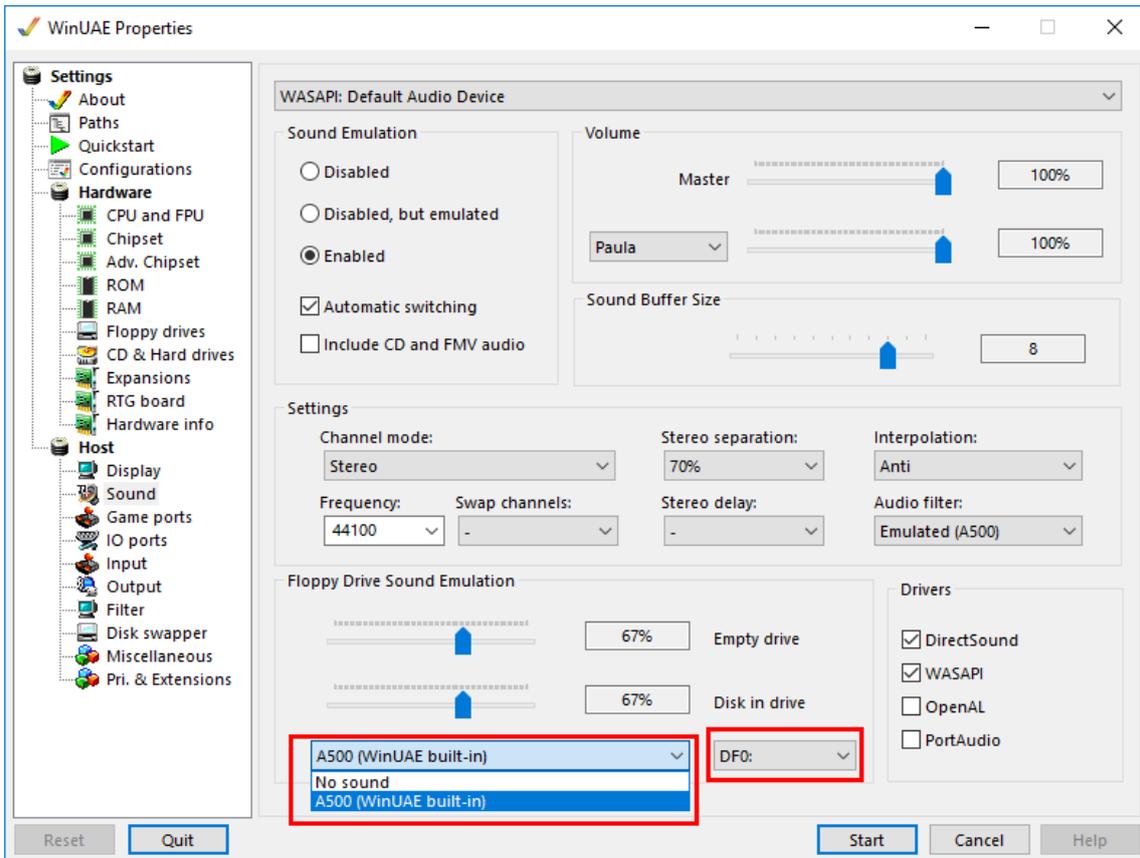
On change donc de *track* car on est déjà sur la *face* 1 (il existe que 2 faces sur une disquette)

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.

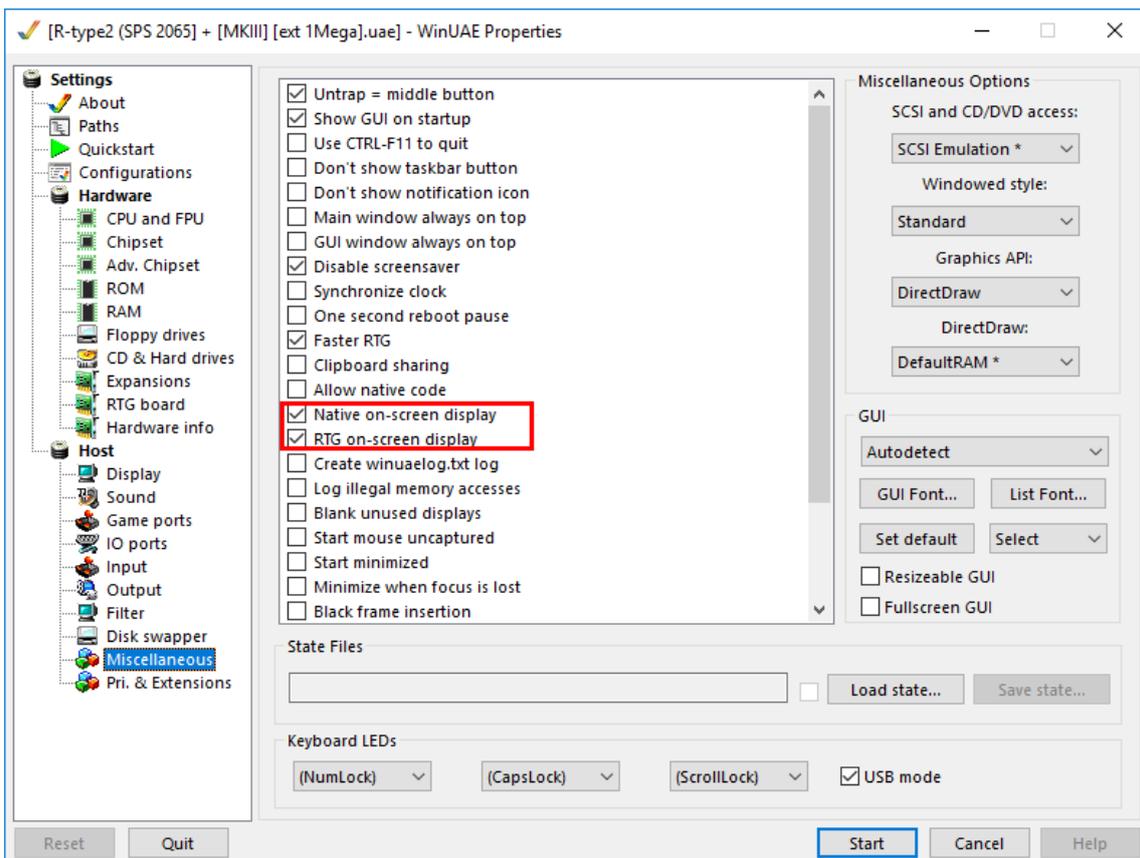
## WinUAE

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 - > DFO 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

Comme dans Tous bon hack qui se respecte, on va commencer par essayer de copier le disk original. Démarrer votre logiciel de copie préféré, à savoir X-copy Pro

**Prendre** le mode **DOSCOPY+**, insérer une **disquette vierge** en **DF1** et la **disquette du jeu original** en **DF0**  
**Lancer** la copie en mode **DOSCOPY+**



Cela semble assez clair, la copie ne fonctionnera pas et il semble que l'on est là un format spécifique. On notera tout de même que l'ensemble des erreurs est de l'ordre de :

### 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	2065_RType2_AMIGA.ipf
TYPE	Floppy_Disk
ENCODER	CAPS(V1)
FILE	2065(V1)
DISK	0
TRACK	00-83
SIDE	0-1
PLATFORM	Amiga
REVOLUTION	2
PROTECTION	Copylock [T00.1]

Concernant la piste 00.0 rien de bien spécial : 11 secteurs de 512 bytes, standard AmigaDOS  
Par contre, la piste 00.1 elle a bien 11 zones mais pas défini en reconnu en tant que secteurs.

TRACK		Data Length (bytes)		Data (bits)			CRC32 of the complete Extra Data Block			Adress	
Data block Description	Sector ID	Data		bytes/sector	GAP		Codage	CapDef	DataOff		Adresse
		MFM bits	bytes		MFM bits	bytes			MFM bits	bytes	
[T00.0]		6446		51568			C96E3DF6			13576-20021	
#0	4	8704	545	512	0	1	MFM	0352	0352	15	13576-13607
#1	5	8704	545	512	0	1	MFM	0906	0906	57	13608-13639
#2	6	8704	545	512	0	1	MFM	1460	1460	92	13640-13671
#3	7	8704	545	512	0	1	MFM	2014	2014	126	13672-13703
#4	8	8704	545	512	0	1	MFM	2568	2568	161	13704-13735
#5	9	8704	545	512	0	1	MFM	3122	3122	196	13736-13767
#6	10	8704	545	512	0	1	MFM	3676	3676	230	13768-13799
#7	0	8704	545	512	0	1	MFM	4230	4230	265	13800-13831
#8	1	8704	545	512	0	1	MFM	4784	4784	300	13832-13863
#9	2	8704	545	512	0	1	MFM	5338	5338	334	13864-13895
#10	3	8704	545	512	4552	285	MFM	5892	5892	369	13896-13927
[T00.1]		6138		49104			5214A82B			20050-26187	
#0	N/A	8272	518	N/A	720	46	MFM	0352	0352	15	20050-20081
#1	N/A	8272	518	N/A	720	46	MFM	0878	0878	55	20082-20113
#2	N/A	8272	518	N/A	720	46	MFM	1404	1404	88	20114-20145
#3	N/A	8272	518	N/A	720	46	MFM	1930	1930	121	20146-20177
#4	N/A	8272	518	N/A	720	46	MFM	2456	2456	154	20178-20209
#5	N/A	8272	518	N/A	720	46	MFM	2982	2982	187	20210-20241
#6	N/A	8272	518	N/A	720	46	MFM	3508	3508	220	20242-20273
#7	N/A	8272	518	N/A	720	46	MFM	4034	4034	253	20274-20305
#8	N/A	8272	518	N/A	720	46	MFM	4560	4560	286	20306-20337
#9	N/A	8272	518	N/A	720	46	MFM	5086	5086	318	20338-20369
#10	N/A	8272	518	N/A	1808	114	MFM	5612	5612	351	20370-20401

Ensuite, **toutes les pistes** se ressemblent, à savoir pas de secteur standard, un codage MFM et une taille de **\$6214** au niveau Data Track.  
Et ce, de la piste 01.0 jusqu'à la piste 79.1

[T01.0]		6214		49712			9E99CC1B			26216-32429	
#0	N/A	98768	6174	N/A	1544	97	MFM	0032	0032	2	26216-26247
[T01.1]		6214		49712			6268D520			32458-38671	
#0	N/A	98768	6174	N/A	1544	97	MFM	0032	0032	2	32458-32489
[T02.0]		6214		49712			40F151C9			38700-44913	
#0	N/A	98768	6174	N/A	1544	97	MFM	0032	0032	2	38700-38731

On notera au passage la détection de track taguée 'copylock' par notre logiciel d'analyse dans l'image IPF.

On notera aussi qu'il est fort probable, au vue de l'image de la copy via X-COPY que toute les pistes n'aient pas la même tailles de data.

### Part 3 Début d'analyse du chargement du jeu

**Insérer** la disquette originale du jeu dans le lecteur et **allumer** votre Amiga.

Si vous passez par un émulateur comme *WinUAE*, il est assez facile de connaître la piste en court.

Sur un amiga réel, à moins d'avoir un lecteur ou un montage spécifique affichant les pistes...il est toujours possible d'utiliser le bruit de chargement des pistes du lecteur pour les compter mais bon.

Quoi qu'il en soit, voilà ce que l'on peut noter sur les chargements du jeu.

<b>Pistes</b>	<b>Information</b>
00 -> 1	Micro Chargement, RAS, écran noir
2 -> 13	Ras, écran noir
13 -> 16	Image jeu r-type
57 -> 59	Image copyright #1
59 -> 60	Image copyright #2
16 -> 18	Ras
18 -> 20	Image écran de départ du jeu
21 -> 26	Démo du jeu L1 qui commence après semble-t-il une phase de décompression
27 -> 32	Démo du jeu L2 qui commence après semble-t-il une phase de décompression
33 -> 43	Démo du jeu L3 qui commence après semble-t-il une phase de décompression

## Part 4 Analyse bootblock

Toujours avec la disquette originale du jeu dans le lecteur de l'Amiga, nous allons regarder ce qui se passe dans le *bootblock* Entrer dans votre **AR**

#RT alias Read Track, permet le chargement de track

Taper : **RT 0 1 50000** (puis touche ESC à la seconde erreur si présente) puis **D 50000**

```
rt 0 1 50000
Disk ok
-
d 50000
~050000 NEG.W A7
~050002 SUBQ.B #1,D0
~050004 CMP.B -(A7),D0
~050006 BCLR #4445,21(A4,D5.L)
~05000C LEA 50016(PC),A0
~050010 MOVE.L A0,00000000.S
~050014 TRAP #0
~050016 MOVE.W #2700,SR
~05001A LEA 00000000,A7
~050020 MOVE.W #7FFF,D0
~050024 MOVE.W D0,00DFF096
~05002A MOVE.W D0,00DFF09A
~050030 CLR.W 00DFF100
~050036 LEA 5004E(PC),A0
~05003A LEA 00078000,A1
~050040 LEA 5043E(PC),A2
~050044 MOVE.L A1,-(A7)
~050046 MOVE.B (A0)+,(A1)+
~050048 CMPA.L A0,A2
~05004A BNE 00050046
~05004C RTS
=====
^05004E MOVEQ #FFFFFFF,D0
~050050 LEA 0007C000,A0
~050056 BSR 00050072
~05005A MOVEQ #0,D0
~05005C LEA 00000500.S,A0
~050060 MOVE.L A0,-(A7)
~050062 BSR 00050076
~050066 LEA 00000E2C.S,A0
~05006A MOVE.L #AE3B9CE3,(A0)
~050070 RTS
=====
```

On peut voir une boucle de copie de donnée de l'adresse **\$5004E** vers **\$78000** et pour finir un **RTS**

La valeur **\$7C000** est mise dans **A0** et on effectue un **BSR 50072**  
Puis on met la valeur **\$500** dans **A0** et on effectue un **BSR 50076**  
suivi d'une modification de **A7** et finalement le **RTS exécute** le code en **\$500**

Il est donc fort probable que le code en **\$50072 et/ou \$50076** charge des données en mémoire **\$500**  
On notera aussi la valeur **AE3B9CE3** mis dans **A0** à l'adresse **\$5006A**  
C'est une valeur... peu commune et juste avant, le chargement de donnée de **\$E2C** dans **A0**...

Effectuer un **redémarrage** de votre Amiga et on attend la fin de la 1er phase de chargement\* avant d'entrer dans notre **AR**  
\*piste 00->13, voir tableau plus haut

Taper : **D**  
#D, alias Désassemble permet de désassembler du code.

```
d
~078174 BEQ 0007816C
~078176 MOVE.W D4,9C(A1)
~07817A MOVE.W #4000,24(A1)
~078180 RTS
```

Comme prévue, le code en court d'exécution est *dans la zone* **\$78000**

Taper : **R** puis **D 500**

#R permet d'afficher tous les registres du 68000

```
R
D0=FC8603FE 00000000 00000000 00000000 00000001 FFFF001A 00000000 00000003
A0=00000AFE 00DF000 0007FEE2 00000CA4 2F3B7F57 D04698F6 000008C8 0007FED0
PC = 0007FEEC USP = 00C014B2 SR = 2704 T=0 S=1 I=111 X=0 N=0 Z=1 V=0 C=0

d 500
~000500 MOVEQ #0,D0
~000502 MOVEQ #0,D1
~000504 MOVEQ #0,D3
~000506 PEA 512(PC)
~00050A MOVE.L (A7)+,00000010
~000510 ILLEGAL
~000512 MOVEM.L D0-D7/A0-A7,-(A7)
~000516 PEA 532(PC)
~00051A MOVE.L (A7)+,00000010
~000520 MOVEA.L A7,A0
~000522 LINEF
~000524 ORI.B #40,D2
```

Cela ressemble à une petite routine de 'copylock' ce qui valide la tag copylock trouvé dans notre analyse de l'image IPF. Peut-être pouvons-nous gagner du temps sur la compréhension de celle-ci en allant directement regarder à l'adresse notée préalablement : **\$E2C** (voir code plus haut sur image du *bootblock* en **\$50066**)

Taper : **D E2A**

```
d e2a
~000E2A CMP.L #AE3B9CE3,D0
~000E30 BNE 000021EA
~000E34 LEA E3C(PC),A0
~000E38 MOVE.L A0,00000020.S
~000E3C MOVE.W #2000,SR
~000E40 LEA 00DF000,A0
~000E46 MOVE.W #7FFF,D0
```

Il est fort probable que ce soit le code qui soit exécuté après la routine de 'copylock'. Le test en **\$E2A** est intéressant car l'on compare la valeur **AE3B9CE3** avec **D0**. On notera au passage que c'est la même valeur que dans notre *bootblock* préalablement chargé en **\$50000** voir **\$5006A**

**Redémarrer** votre Amiga et dès la fin de la première phase de chargement (jusqu'à piste 13) **entrer** dans votre **AR**

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

Taper : **A E2A** puis

```
^000E2A move.w #$f00,$dff180
^000E32 bra $e2a
^000E34
X
```

```
a e2a
^000E2A move.w #$f00,$dff180
^000E32 bra $e2a
^000E34
X
```

Le code continue son petit chemin puis, et assez rapidement, l'écran passe au rouge. Il est temps pour nous d'**entrer** dans notre **AR**, taper : **R**

```
R
D0=AE3B9CE3 00000000 0000FFFF 00000000 55555555 00000000 00001AF9 00001051
A0=00000E2C 00DF000 00BFD000 0007CC10 00001558 00C014B6 00C00276 00000000
PC = 00000E32 USP = 00C014B2 SR = 2700 T=0 S=1 I=111 X=0 N=0 Z=0 V=0 C=0
```

On peut donc connaître la valeur de **D0**, à savoir : **AE3B9CE3**. C'est la valeur qu'il s'attend à avoir avant lors du test en **\$E2A** et si cela n'est pas le cas, il effectue un **BNE 21EA**. Regardons d'ailleurs ce qui se passe en **\$21EA**

Taper : **D 21EA**

```
d 21ea
~0021EA MOVE.W #F0,00DF180
~0021F2 BRA 000021EA
;=====
```

Code assez simple à comprendre, on met le fond d'écran en **VERT** et on boucle sur ce code. Cela ressemble à 100% à une procédure de fin de parcours :)

Check du disk = bon → on continue  
pas bon → On boucle sur ce code d'écran vert.

Passer ce système de protection est assez facile, il suffit d'écraser la valeur de **D0** suivi d'un JUMP vers la fin de la routine de *'copylock'*

Pour être sûr qu'il n'existe pas d'autre vérification sur cette valeur, le mieux est recherché en mémoire si elle est présente à une autre adresse.

*#F alias Find. Permet de chercher des données hexa ou ascii dans la mémoire.*

Taper : **f AE 3B 9C E3**



```
f AE 3B 9C E3
Search from: 000000 to: C00000
000060 07801E 07FF0C 07FF24 07FFBA
Ready.
```

**\$7801E** étant très proche de **\$78000** est à coup sur la 1er copie que nous avons vue juste au-dessus.  
Il semblerait donc qu'il existe pour l'instant 4 autres vérifications au zone mémoire soulignées en rouge dans l'image ci-dessous.

**\$07801E**  
**\$7FF0C**  
**\$7FF24**  
**\$7FFBA**

Nous reviendrons plus tard sur cette problématique.

## Part 5 Analyse et compréhension de la routine de Trackloading

Continuons l'exécution de notre code

#G comme GO, démarre le code en \$e34 comme demandé ici.

Taper : **G E34**

Laisser le jeu se charger un peu, **et pendant un accès disque entrer à nouveau dans l'AR, Taper : D**

On se trouve quelque part en mémoire vers \$FA00.

Un *trackloader* standard fonctionne avec le contrôleur de disquette, on va donc chercher si les adresses des I/O disk sont utilisées.

Petite pique de rappel :

Address	Chip	Name	Description / data bits							
BFD000	ciab	pra	/DTR	/RTS	/CD	/CTS	/DSR	SEL	POUT	BUSY
BFD100	ciab	prb	/MTR	/SEL3	/SEL2	/SEL1	/SELO	/SIDE	DIR	/STEP
BFE001	ciab	pra	/FIR1	/FIRO	/RDY	/TKO	/WPRO	/CHNG	/LED	OVL

#FA, alias Find addressing opcode, permet de chercher une adresse de code modifiant une zone mémoire.

Taper : **FA BFD000** puis **FA BFE001**

```
fa bfd000
Search from: 000000 to: C80000
00F9FC LEA    00BFD000,A2
Searched up to adr: C80000
Ready.

fa bfe001
Search from: 000000 to: C80000
001356 BTST   #6,00BFE001
00EC6C BTST   #6,00BFE001
00EC86 BTST   #7,00BFE001
00EE58 LEA    00BFE001,A0
00F998 BTST   #4,00BFE001
00FA38 BTST   #5,00BFE001
0159A6 BSET   #1,00BFE001
015CF2 BSET   #1,00BFE001
Searched up to adr: C80000
Ready.
```

```
$bfe001      bits 6  /Fire0      Aucun intérêt pour nous dans le cadre du trackloader
$bfe001      bits 7  /Fire1      Aucun intérêt pour nous dans le cadre du trackloader
$bfe001     bits 4  /TKO        Intéressant ($F998)
$bfe001     bits 5  /RDY       Intéressant ($FA38)
$bfe001      bits 1  /LED       Aucun intérêt pour nous dans le cadre du trackloader
```

On devrait donc trouver notre *trackloader* au-dessus de l'adresse \$F998 (avant l'instruction sur le bit /RDY), voir image ci-dessus.

D'ailleurs, en \$00F998, on peut voir quelque chose d'intéressant, un accès 'track'

Jetons un coup d'œil aux codes avant l'adresse \$00F998

Taper : **D F998** et 'remonter le code' comme le montre l'image ci-dessous et regarder un peu le code.

On trouve quelque chose d'étrange de \$F956 à \$F95E. Cela ne ressemble pas à du code normal, d'ailleurs Le LINEF en \$F946 valide ceci.

```
~00F940 SUBI.B #0,0000002.S
~00F946 LINEF
~00F948 SUBI.B #0,2(A7,D0.W)
~00F94E ROXL.B #6,D0
~00F950 SUBI.B #1,0(A3,D6.W)
~00F956 ORI.B #0,-(A2)
~00F95A ORI.B #0,D6
~00F95E ORI.W #E740,(A2)
~00F962 LEA FB20(PC),A1
~00F966 ADDA.W D0,A1
~00F968 MOVE.B (A1)+,D0
~00F96A MOVE.B (A1)+,D1
~00F96C EXT.W D1
~00F96E MOVE.W (A1)+,D2
~00F970 MOVE.L (A1)+,-(A7)
~00F972 BSR 0000F9B0
~00F974 MOVE.L (A7)+,D0
~00F976 RTS
=====
^00F978 LEA FB1A(PC),A2
~00F97C TST.W D0
~00F97E BNE 0000F982
~00F980 MOVE.W (A2),D0
~00F982 MOVE.W D0,(A2)+
~00F984 MOVE.L A0,(A2)+
~00F986 TST.W D0
~00F988 BMI 0000F98C
~00F98A RTS
=====
^00F98C BSR 0000F9F6
~00F98E ORI.B #2,100(A2)
```



C'est plus visible sous le logiciel **HRTmon\*** pour info. (Disponible en ROM pour WinUAE et en disquette pour un Amiga Réel)  
HRTmon rappelle le, se comporte un peu comme l'**AR**, il permet de désassembler du code mémoire et de faire tout plein d'autre chose ;)
\*Bon le hic, c'est qu'utiliser **HRTmon** sous WinUAE n'est pas un problème (buildin in rom), mais avec un Amiga réel... avec un A500+, OK
Un Amiga 1200, OK, un Amiga 500... à moins d'avoir 1Mega de CHIP (et que, pas de slow), le soft ne vas pas fonctionner...

Quoi qu'il en soit, voila ce que ca donne avec celui-ci.

```

d f946
D $0000F946 PFLUSH      (A0)                ;$00000AFE
D $0000F948 SUBI.B     #???,(2,A7,D0.W) ;$000802CA
D $0000F94E ROXL.B     #6,D0
D $0000F950 SUBI.B     #???,(0,A3,D6.W) ;$00000CA4
D $0000F956 ORI.B      #???,-(A2)      ;$0007FEE2
D $0000F95A ORI.B      #???,D6
D $0000F95E ORI.W      #E740,(A2)      ;$0007FEE2
D $0000F962 LEA        ($FB20,PC),A1
D $0000F966 ADDA.W     D0,A1
D $0000F968 MOVE.B     (A1)+,D0        ;$000FF000
D $0000F96A MOVE.B     (A1)+,D1        ;$000FF000
D $0000F96C EXT.W      D1
D $0000F96E MOVE.W     (A1)+,D2        ;$000FF000
D $0000F970 MOVE.L     (A1)+,-(A7)     ;$000FF000 $0007FECA
D $0000F972 BSR.B      $F9B0
D $0000F974 MOVE.L     (A7)+,D0        ;$0007FECA

```

---

HRTmon V2.36 by Alain Malek Page 1  
Track:[00] Drive:[0] Address:[\$00000000] Hrt:[\$00A10000] Fcpu:[65802] Ins:[off]

Il y a forte chance que le code commence en **\$F962** et même... si on regarde bien, **\$F95E** semble autant suspect qu'en **\$F956**.  
**Par contre, le code en \$F962** lui, semble normal... Hummmm, on va même quand même vérifier quelque chose.

On va **désassembler** mais à partir de cette **adresse-1**, donc : **\$F962-1** et **\$F95E-1**

Taper : **D F962-1** puis **D F95E-1**

```

d F962-1
~00F960 ASL.W #3,D0
~00F962 LEA FB20(PC),A1
~00F966 ADDA.W D0,A1

d F95E-1
~00F95C BRA 0000F9B0
|=====
^00F960 ASL.W #3,D0
~00F962 LEA FB20(PC),A1
~00F966 ADDA.W D0,A1

```

**! YES.** On retombe bien sur notre code 'normal' en **\$F962** puis **\$F966** mais on peu voir clairement qu'une instruction nous était 'cachée'  
à savoir : **\$F960 ASL.W #3,D0**

```

d f95d
~00F95C BRA 0000F9B0
|=====
^00F960 ASL.W #3,D0
~00F962 LEA FB20(PC),A1
~00F966 ADDA.W A4,A1
~00F968 MOVE.B (A1)+,D0
~00F96A MOVE.B (A1)+,D1
~00F96C EXT.W D1
~00F96E MOVE.W (A1)+,D2

```

On retombe bien sur notre code (ligne verte image ci-dessus) **mais avec une instruction en plus**.  
Et on voie assez clairement que le début du code a de grande chance d'être en **\$F960**  
Donc notre **trackloader**, commencerait à l'adresse **\$F960**

Si on cherche en mémoire un branchement vers cette adresse, on trouve bien quelque chose, **Taper : D F960**

```
fa f960
Search from: 000000 to: C80000
00F958 BRA      0000F960
-
```

Puis, bien sûr, par rapport au résultat, **taper : FA F958**

```
fa f958
Search from: 000000 to: C80000
0013F4 JSR      0000F958
001D82 JSR      0000F958
001EDE JSR      0000F958
001FD0 JSR      0000F958
002250 JSR      0000F958
00AA42 JSR      0000F958
00AAE8 JSR      0000F958
Searched up to adr: C80000
Ready.
```

Il y a visiblement du monde qui saute en **\$F958**, ce qui est assez logique car **R-type2** possède plusieurs niveaux. D'ailleurs, rien qu'au début de ce tuto, on a compté pas mal de chargements.

On en profite donc pour noter les adresses concernées : **\$13F4, \$1D82, \$1EDE, \$1FD0, \$2250, \$AA42, \$AAE8**

Bon, voyons ce trackloader en action dans son fonctionnement normal !

**Redémarrer** votre Amiga, laisser finir la 1er phase de chargement (jusqu'à piste 13) puis **entrer** dans votre **AR**

**Taper : D F958** histoire d'être sur du code que l'on va remplacer, puis, **appuyer sur ESC** et **taper :**

*#a, alias Assemble. Permet d'assembler du code assembleur.*

**A F958**

```
^00F958 BRA      F958
^00F95A
```

Cela sera notre *'boucle de pause'* avant l'exécution du *Trackloader* original.

**Taper X** pour retourner à l'exécution normal du code.

```
d f958
^00F958 BRA      0000F960
;=====

a f958
^00F958 bra f958
^00F95A

x
```

Assez vite, le jeu semble ne plus rien faire... **Entrer** dans votre **AR**, **taper : D**

On est bien dans notre *'boucle de pause'* créée juste au-dessus.

**Appuyer** sur la touche **ESC** puis **R**, **Noter** les registres puis continuer l'exécution du code vers le *trackloader* en **tapant : G F960**

```
d
^00F958 BRA      0000F958

r
D0=00000001 00000001 0000FFFF 0000FFFF 55555555 00000000 AAAAF2ED 00005045
A0=0004CE58 00DFF000 00BFD000 00030188 00001558 00C014B6 000148F6 000004F4
PC = 0000F958 USP = 00C014B2 SR = 2000 T=0 S=1 I=000 X=0 N=0 Z=0 V=0 C=0

g f960_
```

Assez rapidement, le jeu va à nouveau entrer dans notre boucle en **\$F958** et ne plus rien faire, **Entrez** de nouveau dans votre **AR**

**Noter les registres** puis continuer l'exécution du code vers le *trackloader* en tapant : **G F960**

**!\ Continuer ce mode opératoire jusqu'à la fin du jeu et noter les valeurs des registres dans un tableau.**

**Mode Invulnérable** = Commencer une partie, metre le jeu en pause puis appuyer sur le bouton gauche de la souris et la touche F1, Relâcher la touche et le bouton, un écran vert confirme la manipulation. Appuyez sur P pour retourner au jeu.

**Petit rappel :** Pour connaître l'adresse de retour après un branchement à une sous routine. Elle est stockée dans **A7** qui **contient l'adresse mémoire** ou est **écrite l'adresse de retour**.

Exemple :

```
r
D0=00000001 00000001 0000FFFF 0000FFFF 55555555 00000000 AAAAF2ED 00005045
A0=0004CE58 00DFF000 00BFD000 00030188 00001558 00C014B6 000148F6 000004F4
PC = 0000F958 USP = 00C014B2 SR = 2000 T=0 S=1 I=000 X=0 N=0 Z=0 V=0 C=0
n 4F4
:0004F4 00 00 1E E4 00 00 1E B0 00 00 0E 92 70 00 72 00 ...ä.....p.r.
```

## Part 6 Tableau des registres lors de l'appel de la routine de trackloader

Tableau1

BOOT	N/A	N/A	Screen RTYPE	Screen (c) #1	Screen (c) #2	N/A	Screen Start
TRK	0 --> 13	0-->1	13 --> 16	57 --> 59	59 --> 60	16 --> 18	18 --> 20
D0	N/A	00000001	0000000F	00000010	00000002	00000004	00000003
D1	N/A	00000001	0000001F	0000001F	0000001F	0000001F	00000078
D2	N/A	0000FFFF	0000FFFF	0000FFFF	0000FFFF	0000FFFF	0000000E
D3	N/A	0000FFFF	0000FFFF	0000FFFF	0000FFFF	0000FFFF	0000FFFF
D4	N/A	55555555	55555555	55555555	55555555	55555555	55555555
D5	N/A	00000000	00000000	00000000	00000000	00000000	00000000
D6	N/A	AAAAF2ED	0000FFFF	9C71FFFF	3A6EFFFF	399C093D	0000FFFF
D7	N/A	00005045	0000FFFF	1451FFFF	1044FFFF	114FFFFF	0000FFFF
A0	N/A	0004CE58	0004CE58	0004CE58	0004CE58	00053000	00053000
A1	N/A	00DFF000	000102F6	000102F6	000102F6	0005CF4C	00010378
A2	N/A	00BFD000	00054658	00054658	00054658	0002FD80	0000B4F6
A3	N/A	00030188	00056E58	00056E58	00056E58	00037A80	000151F0
A4	N/A	00001558	00059658	00059658	00059658	0003F780	0003F780
A5	N/A	00C014B6	0007BA00	0007BA00	0007BA00	00047480	000150B8
A6	N/A	000148F6	000148F6	000148F6	000148F6	000148F6	000148F6
A7	N/A	000004F4	000004F4	000004F8	000004F8	000004F8	000004F8
(SP)	N/A	00001EE4	00001EE4	00001EE4	0000AA48	000013FA	00001D88

	N/A	Anim Intro	Niveau 1	Niveau 2	Niveau 3	Niveau 4	Niveau 5	Niveau 6
TRK	N/A	18 --> 19	19 --> 26	26 --> 32	32 --> 43	43 --> 48	48 --> 52	52 --> 57
D0	N/A	00000000	00000005	00000006	00000007	00000008	00000009	0000000A
D1	N/A	00000000	0000000E	00063918	00063918	00063918	00063918	00063918
D2	N/A	00000001	00063000	00063000	00063000	00063000	00063000	00063000
D3	N/A	00000001	00007D00	00000000	00000000	00000000	00000000	00000000
D4	N/A	00000082	00000028	00000028	00000028	00000028	00000028	00000028
D5	N/A	0000001F	00008FCA	00000001	00000001	00000001	00000001	00000001
D6	N/A	82E10020	82E10080	00000030	00000030	00000030	00000030	00000030
D7	N/A	0000FFFF	0000FFFF	0000FFFF	0000FFFF	0000FFFF	0000FFFF	0000FFFF
A0	N/A	00DFF000	0004F180	0004F180	0004F180	0004F180	0004F180	0004F180
A1	N/A	0000EF36	0001039C	0001039C	0001039C	0001039C	0001039C	0001039C
A2	N/A	000276C8	0005058C	00007f89A	00007f89A	00007f89A	00007f89A	00007f89A
A3	N/A	00073602	000796E8	00007F93A	00007F93A	00007F93A	00007F93A	00007F93A
A4	N/A	0000F1A4	0000F1A4	0007F9DA	0007F9DA	0007F9DA	0007F9DA	0007F9DA
A5	N/A	00009348	00009348	000150C0	000150C0	000150C0	000150C0	000150C0
A6	N/A	000148F6	000148F6	000148F6	000148F6	000148F6	000148F6	000148F6
A7	N/A	000004F8	000004F8	00004F8	00004F8	00004F8	00004F8	00004F8
(SP)	N/A	00001E80	0000AAEE	0000AAEE	0000AAEE	0000AAEE	0000AAEE	0000AAEE

Avec ce tableau, on va comprendre le fonctionnement du *trackloader*

Si on regarde de nouveau le code du *trackloader* on peut voir qu'une lecture mémoire en **\$FB20** est effectuée.

Taper : **D F960** puis **M F920**

#M, alias Visualisation mémoire en HEXA/ASCII

```

d f960
~00F960 ASL.W #3,D0
~00F962 LEA FB20(PC),A1
~00F966 ADDA.W D0,A1
~00F968 MOVE.B (A1)+,D0
~00F96A MOVE.B (A1)+,D1
~00F96C EXT.W D1
~00F96E MOVE.W (A1)+,D2
~00F970 MOVE.L (A1)+,-(A7)
~00F972 BSR 0000F9B0
~00F974 MOVE.L (A7)+,D0
~00F976 RTS
;=====
M fb20
:00FB20 02 01 01 21 00 02 43 CC 1A 03 00 45 00 00 8B C5 ...!..C....E...
:00FB30 20 01 00 39 00 00 73 8F 24 0B 00 1C 00 00 38 AF ..9..s.$.....8.
:00FB40 27 04 00 08 00 00 10 70 28 01 00 9F 00 01 3E CB ',.....p(.....>.
:00FB50 35 05 00 7F 00 00 FE 22 40 01 01 0D 00 02 1A 96 5.⌘.....0.....
:00FB60 56 07 00 7C 00 00 F8 5A 61 00 00 56 00 00 AD 35 V..l...Za..V...5
:00FB70 68 03 00 76 00 00 ED 9B 72 02 00 02 00 00 04 00 h..v....r.....
:00FB80 72 04 00 02 00 00 04 00 72 06 00 02 00 00 04 00 r.....r.....
:00FB90 72 08 00 02 00 00 05 75 72 0B 00 26 00 00 4D 5C r.....ur..&..M\
:00FBA0 76 02 00 27 00 00 4F EF AA AA AA AA AA AA AA AA v..'..0.....
:00FBB0 AA .....

```

Cela ressemble à une *filetable*

Si on regarde le début du code du *trackloader*, ça nous donne :

```

00F960 ASL.W #3,D0 #Décalage de 3 bit vers la gauche de D0 ce qui nous crée au final D0=D0*8
00F962 LEA FB20(PC),A1 #On met FB20 dans A1, c'est notre Adr. De départ de notre filetable
00F966 ADDA.W D0,A1 #Ajoute D0 à A1, ce qui va donc nous positionner dans la filetable
00F968 MOVE.B (A1)+,D0 #On récupère la première valeur du tableau 'TA1' dans D0, ensuite A1=A1+1
00F96A MOVE.B (A1)+,D1 #On récupère la seconde valeur du tableau 'TA2' dans D1, ensuite A1=A1+1
00F96C EXT.W D1 #On efface les 2 premiers bytes de D1
00F96E MOVE.W (A1)+,D2 #On récupère les deux premières valeurs du tableau 'TA3' dans D2, ensuite A1=A1+2
00F970 MOVE.L (A1)+,-(A7) #On récupère les quatre premières valeurs du tableau 'TA4' dans SP
00F972 BSR 0000F9B0 #On se branche sur le sous-programme en $F9B0
00F974 MOVE.L (A7)+,D0 #On restitue le contenu préalablement sauvé 2 lignes au-dessus vers D0 (check?)
00F976 RTS #On revient

```

Ça nous 'découpe' la *filetable* comme ceci :

```

M FB20
:00FB20 02 01 01 21 00 02 43 CC 1A 03 00 45 00 00 8B C5 ...!..C....E...
:00FB30 20 01 00 39 00 00 73 8F 24 0B 00 1C 00 00 38 AF ..9..s.$.....8.
:00FB40 27 04 00 08 00 00 10 70 28 01 00 9F 00 01 3E CB ',.....p(.....>.
:00FB50 35 05 00 7F 00 00 FE 22 40 01 01 0D 00 02 1A 96 5.⌘.....0.....
:00FB60 56 07 00 7C 00 00 F8 5A 61 00 00 56 00 00 AD 35 V..l...Za..V...5
:00FB70 68 03 00 76 00 00 ED 9B 72 02 00 02 00 00 04 00 h..v....r.....
:00FB80 72 04 00 02 00 00 04 00 72 06 00 02 00 00 04 00 r.....r.....
:00FB90 72 08 00 02 00 00 05 75 72 0B 00 26 00 00 4D 5C r.....ur..&..M\
:00FBA0 76 02 00 27 00 00 4F EF AA AA AA AA AA AA AA AA v..'..0.....
:00FBB0 AA .....
:00FBC0 AA .....

```

Tableau2

Adr. Filetable	INFO	D0	D1	D2		A7			
		TA1	TA2	TA3		TA4			
\$00FB20	N/A	02	01	01	21	00	02	43	CC
\$00FB28	SCREEN R-TYPE2	1A	03	00	45	00	00	8B	C5
\$00FB30	N/A	20	01	00	39	00	00	73	8F
\$00FB38	Anim Intro	24	0B	00	1C	00	00	38	AF
\$00FB40	Screen Start	27	04	00	08	00	00	10	70
\$00FB48	Niveau 1	28	01	00	9F	00	01	3E	CB
\$00FB50	Niveau 2	35	05	00	7F	00	00	FE	22
\$00FB58	Niveau 3	40	01	01	0D	00	02	1A	96
\$00FB60	Niveau 4	56	07	00	7C	00	00	F8	5A
\$00FB68	Niveau 5	61	00	00	56	00	00	AD	35
\$00FB70	Niveau 6	68	03	00	76	00	00	ED	9B
\$00FB78	N/A	72	02	00	02	00	00	04	00
\$00FB80	N/A	72	04	00	02	00	00	04	00
\$00FB88	N/A	72	06	00	02	00	00	04	00
\$00FB90	N/A	72	08	00	02	00	00	05	75
\$00FB98	Screen © #1	72	0B	00	26	00	00	4D	5C
\$00FBA0	Screen © #2	76	02	00	27	00	00	4F	EF

**TA1** varie de \$02 à \$76, (max !118 en décimal).  
Possible qu'il indique 'track' de départ. (de !0 à !159)

**TA2** varie de \$01 à \$0B, (max !11 en décimal)  
Vue que le format de ces pistes n'est pas standard **AMIGADOS** et que l'on a des **ERROR 1** 'Less or more than 11 sectors' sous X-Copy, **TA2** pourrait du coup indiquer un nbr de secteurs custom ou quelque chose dans le genre, un nbr de bloc custom par exemple.

**TA3** varie de \$0002 à \$0121, (max !289 en décimal)  
Il indique sûrement d'une taille de donnée.  
Peut-être en nbr de secteur car il est trop petit pour indiquer directement une taille et trop grosse indiquer autre chose.

**TA4** au vue de sa taille, on imagine aussi une taille de donnée, mais en quoi...

Ce qui serait intéressante serait dans un premier temps serait de connaître l'adresse de destination mémoire...  
Il est impossible que ce soit la même tout le temps donc elle doit être passé aussi via les registres vus que dans la *filetable*, on a rien.

Au vue du tableau des registres (tableau 1), il est impossible qu'elle se trouve en **D0-D7** car, **D0** à **D2** sont déjà utilisé (voir code original au dessus) et au vue des valeurs contenu dans **D3** à **D7** cela semble très improbable.  
On va donc commencer nos tests avec **A0**

**Redémarrer** votre Amiga et dès la fin de la première phase de chargement (jusqu'à piste 13) **entrer** dans votre **AR**  
**Taper** : **A f958** puis  
^00F958 BRA F958  
^00F95A

#X permet le retour au code Amiga en court.  
X

Très vite l'Amiga semble ne plus rien faire.

On va entrer dans l'AR, afficher les registres et remplir la zone mémoire que l'on pense de destination avec la valeur **AA**  
puis regarder dans la pile l'adresse de retour et poser une boucle à celle-ci.

Comme prévu (déjà noter dans notre relevé des registres précédent lors des chargements, **tableau 1**), on tombe sur **\$4CE58**

Entrer dans l'AR et Taper :

#o alias Fill Memory. Permetts-le remplisse avec une valeur précise d'une zone mémoire.

R puis O AA, 4CE58 60000

M 4F4 (pour afficher l'adresse de retour stockée dans la pile) puis

A 1EE4

^001EE4 bra lee4

^001EE6

G f960

```

d f958
^00F958 BRA    0000F960
a f958
^00F958 bra f958
^00F95A
x

p
D0=00000001 00000001 0000FFFF 0000FFFF 55555555 00000000 AAAAF2ED 00005045
A0=0004CE58 00DF000 00BFD000 00030188 00001558 00C014B6 000148F6 000004F4
PC = 0000F958 USP = 00C014B2 SR = 2000 T=0 S=1 I=000 X=0 N=0 Z=0 V=0 C=0

o AA, 4ce58 60000
Ready,

M 4f4
:0004F4 00 00 1E E4 00 00 1E B0 00 00 0E 92 70 00 72 00 ...ä.....p.p.

a lee4
^001EE4 bra lee4
^001EE6

g f960_

```

L'Amiga va charger quelques pistes et assez vite, ne plus rien faire. Entrer dans l'AR et Taper :

F AA AA AA AA AA AA AA AA AA, 4CE58 (et appuyer sur ESC dès le 1er affichage)

La fin des données est située en \$55A58, ce qui nous donne comme taille : \$55A58-\$4CE58 = \$8C00

Humm, ça ne ressemble pas à ce que l'on a dans notre filetable...

Si TA4 est la taille de nos donnée chargée par le trackloader, la fin des données devrait se trouver en \$4CE58+\$8BC5 = \$55A1D

Mais, si on regarde ça de plus près...

Taper :

M 55A0D

On peut voir qu'en fait, \$55A1D est tout à fait probable comme fin de donnée.

```

f AA AA AA AA AA AA AA AA AA, 4ce58
Search from: 04CE58 to: C00000
055A58 055A59 055A5A 055A5B 055A5C 055A5D 055A5E 055A5F 055A60 055A61

? 55a58-4ce58
%00000000000000001000110000000000 = $00008C00 = !0000035840

? 4ce58+8bc5
%0000000000000000101010101000011101 = $00055A1D = !0000350749

M 55a0d
:055A0D 64 7F F0 08 FF 83 80 00 00 00 C8 40 49 63 65 21 d@.....@Ice!
:055A1D 00 00 00 00 00 00 00 00 04 9E E8 00 00 00 00 00 .....
:055A2D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

```

TA4 serait donc plutôt la taille des données sans extra-Info

D'ailleurs, si on prend pour référence la taille de ce qui a été lue, à savoir : \$8C00

Que l'on divise par TA3 de 'screen RTYPE2' de notre filetable, à savoir \$45

\$8C00 = !35840

\$45 = !69 mais en fait, si on part de Zero comme 1ère valeur, cela nous donne un total de !70 valeurs possibles.

Donc : !35840 / !70 = !512

Tiens donc, exactement la taille d'un secteur AMIGADOS.

Cela reste à vérifier mais il semblerait que TA3 indique la taille des données lue en nombre de block de 512 bytes

## Part 7 Extra - Taille d'une piste

Pour mieux ripper l'ensemble des données il est préférable (mais pas obligatoire) de connaître la taille d'une piste. Sur une disquette au format AmigaDOS, elle est de \$1600 mais quand est t'il sur notre jeux R-Type II ?

Au vue des erreurs affichées lors de la phase de copie avec le logiciel X-copy pro. L'ensemble des pistes semblent non homogène... On va quand même vérifier quelques pistes. La 1er piste étant *AmigaDOS*, on va faire nos tests sur la *Seconde* et *Troisième TRACK*

**Redémarrer** votre Amiga et dès la fin de la première phase de chargement (jusqu'à piste 13), ou plus exactement, dès le début de la seconde phase, **entrer** dans votre **AR**

```
Taper : A F958 puis
^00F958 BRA F958
^00F95A
x
```

Très vite l'amiga semble ne plus rien faire.

On va **entrer** dans l'**AR**, afficher les registres, regarder dans la pile l'adresse de retour et poser une boucle à celle-ci. puis modifier table *filetable* pour lire le début de la 3eme **TRACK**

**Entrer** dans l'**AR** et **Taper** :

**R** (on note **A0** et **A7**)

**M 4F4** (On affiche le contenu à l'adresse de **A7**, à savoir l'adresse de retour stockée dans la pile, ici **1EE4**) puis, on crée notre boucle.

```
A 1EE4
^001EE4 bra lee4
^001EE6
```

**M FB28** (Comme vue plus haut, c'est l'adresse FB28 qui est adressé en 1er donc on modifie celle-ci)  
:00FB28 03 00 00 01 00 00 02 00 (3eme track, sector 0, 1 secteur, taille 200)

**G F960**

Très vite l'amiga semble ne plus rien faire. On re **entrer** dans l'**AR**, puis

**Taper** : **M 4CE58** (4CE58 étant l'adresse de destination du trackloader, contenue dans A0 lors de l'appel de la routine)

```
a f958
^00F958 bra f958
^00F95A
x
No known virus in memory!
Ready.

r
D0=00000001 00000001 0000FFFF 0000FFFF 55555555 00000000 AAAAF2ED 00005045
A0=0004CE58 00DFF000 00BFD000 00030188 00001558 00C014B6 000148F6 000004F4
PC = 0000F958 USP = 00C014B2 SR = 2000 T=0 S=1 I=000 X=0 N=0 Z=0 V=0 C=0
n 4f4
:0004F4 00 00 1E E4 00 00 1E B0 00 00 0E 92 70 00 72 00 ...ä.....p.r.
a lee4
^001EE4 bra lee4
^001EE6
n fb28
:00FB28 03 00 00 01 00 00 02 00 20 01 00 39 00 00 73 8F ...E....9..s.
:00FB38 24 0B 00 1C 00 00 38 AF 27 04 00 08 00 00 10 70 $......8.'.....p

g f960
No known virus in memory!
Ready.
n 4ce58
:04CE58 00 00 1C 8C 30 39 00 00 1C 12 1A B2 00 00 48 E7 ....09.....H.
```

On note, le début de la TRACK03 est : 00 00 1C 8C 30 39 00 00 1C 12 1A B2 00 00 48 E7

On refait la même manipulation mais pour lire le début de la 4eme track donc modification de \$fb28 par  
:00FB28 04 00 00 01 00 00 02 00 (Début 4eme track, sector 0 avec une taille de 1 secteur, taille \$200 bytes)

On note, le début de la TRACK04 est : 00 40 3D 40 00 CA 3D 41 00 CC 41 F9 00 02 76 00

Maintenant, on va refaire cette manipulation mais en commençant 1 TRACK avant (donc 2 au lieu de 3 et 3 au lieu de 4) Mais en lisant le plus de donnée possible d'une piste. Sur une disquette AmigaDOS il y a 11 secteurs, on va donc prendre un peu plus pour 'déborder' sur la prochaine TRACK

Refaite donc la manipulation mais avec  
:00FB28 02 00 00 0F 00 00 1E 00 (Début 2eme track, sector 0 avec une taille de 15 secteurs, taille \$1E00 bytes)

Puis on recherche en mémoire le début que l'on a noté de la **TRACK03** en **Tapant** :  
**F 00 00 1C 8C 30 39 00 00 1C 12 1A B2 00 00 48 E7, 4CE58** (Début le Premier affichage, appuyer sur **ESC**)  
Cela nous donne **\$4E616** donc **\$4E616 - \$4CE58 = \$17BE**  
La taille de la **TRACK02** est donc de **\$17B0** au lieu des **\$1600** Standard AmigaDOS

Refaite la manipulation mais avec  
:00FB28 03 00 00 0F 00 00 1E 00 (Début 3eme track, sector 0 avec une taille de 15 secteurs, taille \$1E00 bytes)  
**F 00 40 3D 40 00 CA 3D 41 00 CC 41 F9 00 02 76 00, 4CE58** (Dès le début du Premier affichage, appuyer sur **ESC**)  
Cela nous donne **\$4E658** donc **\$4E658 - \$4CE58 = \$1800**

Bingo, pas la même taille. La taille de la **TRACK03** est donc de **\$1800** au lieu des **\$1600** Standard AmigaDOS

## Part 8 Extra - Taille d'un secteur

On ne sait jamais... Même si le 'standard' nous appelle vers une taille de 512 bytes, il est préalablement de tester.

On va se passer de copie d'écran vue que ce sont les mêmes manipulations que précédemment mais en travaillant sur les secteurs.

Refaites donc les manipulations mais avec

:00FB28 02 01 00 01 00 00 20 00

(Début 2eme track, sector 1 avec une taille de 1 secteur, taille \$200 bytes)

On note, le début du secteur 1 en TRACK02 est : 70 00 72 00 76 00 48 7A 00 0A 23 DF 00 00 00 10

Refaites donc la manipulation mais avec

:00FB28 02 00 00 02 00 00 40 00

(Début 2eme track, sector 0 avec une taille de 2 secteurs, taille \$400 bytes)

Puis on recherche en mémoire le début que l'on a noté du secteur 1 en TRACK02

F 70 00 72 00 76 00 48 7A 00 0A 23 DF 00 00 00 10, 4CE58 (Début le Premier affichage, appuyer sur ESC)

Cela nous donne \$4D058 donc  $\$4D058 - \$4CE58 = \$200$

La taille d'un secteur en track2 est donc de \$200 bytes à savoir 512 bytes en décimal



## Part 10 Ripage des pistes

Bon, il n'y a plus qu'a comme on dit.

On a vu que la taille d'une **TRACK** n'était pas constante et que celle-ci pouvait atteindre une taille de \$1800 ce qui, à ma connaissance est le maximum que l'on puisse avoir sur Amiga. On va donc partir de cette base pour nos calculs et ajuster si besoin à la fin.

Au vue de ce que l'on sait et de notre *filetable*, notre début de rip va commencer en **TRACK2** et aller jusqu'à la TRACK 76 + 27 secteurs

\$75 - \$2 = \$73

\$73 \* \$1800 = \$AC800

\$AC800 + \$4E4F = \$B164F

Le *trackloader* du jeu fonctionne en taille de secteur donc : **\$B164F / \$200 = \$58B**

Comme je suis d'une nature méfiante, on va ajouter encore \$F à celui-ci ce qui nous donne : **\$58B + \$F = \$59A**

De toute façon, on a noté la fin des données en hexa donc, on pourra ajuster si nécessaire lors de la sauvegarde.

N'oublions pas que le *trackloader* a aussi besoin d'une taille de donnée donc : **\$59A \* \$200 = \$B3400**

**C'est partie !** Je passe la section commentaire qui reste exactement la même que précédemment.

**Redémarrer** votre Amiga et dès la fin de la première phase de chargement (jusqu'à piste 13), **entrer** dans l'**AR**

**Taper** : **A F958** puis

*#BRA, Instruction du 68000 qui permet de se brancher à l'adresse indiqué, ici une boucle sur nous-même.*

**^00F958 BRA F958**

**^00F95A**

**X**

Très vite l'amiga semble ne plus rien faire. **Entrer** dans l'**AR** et **Taper** :

**A 1EE4**

**^001EE4 bra lee4**

**^001EE6**

**R A0 40000**

**M FB28**

**:00FB28 02 00 05 9A 00 0B 34 00**

**O BB, 40000 f3400**

**G f960**

Bon, pour une raison que j'ignore, la fonction de recherche de chaîne dans la MKIII ne fonctionne pas bien en mémoire haute... -\_-'  
On va donc aller voir à la main.

**Taper** : **M F3400** (et remonter jusqu'à apercevoir la vraie fin des données, en l'occurrence, en **\$F33E0** on a quelque chose)

```
m fb28
:00FB28 02 00 05 9a 00 0b 34 00 20 01 00 39 00 00 73 8f ...E....,9..s.
:00FB38 24 08 00 1c 00 00 38 af 27 04 00 08 00 00 10 70 $.....8.'.....p

? 40000+b3400
%00000000000011110011010000000000 = $000F3400 = !0000996352

o BB, 40000 f3400
Ready.

g f960
No known virus in memory!
Ready.

m f33e0
:0F33E0 1f 00 06 ff 05 00 00 00 00 c8 40 49 63 65 21 DE .....@Ice!,
:0F33F0 00 00 00 00 00 04 da c0 9e c5 8a b3 3a 6e 28 6b .....:n(k
:0F3400 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
:0F3410 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
:0F3420 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
:0F3430 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
:0F3440 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
:0F3450 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

On a donc une des données (qui colle au passage EXACTEMENT à ce que l'on a noté de la fin des DATA du Screen #2) en **\$F33FF** (plus exactement en **\$F33EF** mais bon, soyons prudent, on préfère en garder un peu plus que pas assez)

Ce qui nous donne une taille finale de : **\$F33FF - \$40000 = \$B33FF**, Fichtre, on n'était pas loin !)

Et **\$B33FF** en décimal nous donne : 734 207 Bytes soit environ **717 Ko**, on devrait pouvoir stocker ça sur une disquette standard Amiga ☺

**Insérer** dans votre Amiga une disquette vierge et **Taper**

*#format, Permet de formater une disquette au format AmigaDOS*

*#SM, alias SaveMemory permet donc de sauver la zone mémoire de 40000 à F33FF*

**Format R-type2\_1**

Ready to format disk in drive DF0: (y/n)? y

Disk ok

**SM dump, 40000 F33FF**

## Part 11 Création de notre propre loader

On va reprendre exactement ce qui existe dans le tuto original d'AlphaOne à savoir, le *sectorLoader* de Rob Northen

**Insérer** votre disk **ASM One 1.20** ou son image adf dans votre **Amiga**.

Une fois notre assembleur préféré chargé, **remplacer** la Disquette d'**ASM one** par notre disquette de **Sauvegarde** en **DF0**  
Celle contenant le fichier dump de sauvegarde.

```
ASM-One V1.20 By T.F.A. Source »
----- ASM-One V1.20 MC680x0/MC6888x Macro Assembler (KS 1.2/1.3) -----
Original coding by Rune Gram-Madsen      (1990-1991)
Additional coding by T.F.A.              (1991-1992)
Additional 680x0/6888x coding by T.F.A.  (1992-1993)
Release date 19-09-1993 by T.F.A.

Changed standard directory to » SOURCES:
ALLOCATE Fast/Chip/Publ/Abs>Chip
WORKSPACE (Max.2047) KB>800
>
```

Après avoir alloué environ **800 Ko** de **Chipmen**, **Taper** le code suivant  
Il est très bien commenté et ne nécessite à mon sens aucun addon d'explication.

```
; DISKIMAGE_RNC.s
; #####
; ## R-TYPE II (C) ACTIVISON ##
; ## DISKIMAGE (133 TRACKS) FOR FLASHTRO TUTORIAL. ##
; ## !! USE CHIPMEM FOR THIS FOR WRITEBACK !! ##
; ## ----- ##
; ## USAGE: ##
; ## 1. <A> ASSEMBLE SOURCECODE ##
; ## 2. <J> EXECUTE SOURCE TO MAKE NECESSARY PATCHES IN DISKIMAGE ##
; ## 3. <WT> WRITE BACK 133 TRACKS FROM LABEL #DISK_START ##
; ## 4. <CC> CALCULATE BOOTBLOCK-CHECKSUM ##
; ## ----- ##
; #####

DISKOFFSET_COPYLOCK_CODE = 2*$1800+$200
DISKOFFSET_ORIGINAL_LOADER = 2*$1800+$200+$F960-$500
DISKOFFSET_FILETABLE = 2*$1800+$200+$FB20-$500

; OVERWRITE OLD COPYLOCK CODE WITH OUR NEW LOADER
; -----

LEA NEW_COPYLOCK_CODE(PC),A0
LEA DISK_START+DISKOFFSET_COPYLOCK_CODE,A1
MOVE.L # (COPYLOCK_CODE_END-NEW_COPYLOCK_CODE)-1,D0
COPY_CL: MOVE.B (A0)+,(A1)+
         DBF D0,COPY_CL

; OVERWRITE FIRST INSTRUCTION OF ORIGINAL LOADER WITH
; A "JMP $500" -> SO OUR LOADER IS EXECUTED!
; -----

LEA DISK_START+DISKOFFSET_ORIGINAL_LOADER,A0
MOVE.W #$4EF9,(A0)+
MOVE.L #$500,(A0)+

; THE FILETABLE IS LOCATED AT TRACK #2 SECTOR #0 ON DISK
; AND IS LOADED SEPERATELY BY THE ORIGINAL CODE. WE SKIP THIS PART
; AND COPY IT INTO THE RIGHT PLACE HERE ($FB20 IN MEMORY).
; -----

LEA DISK_START+2*$1800,A0 ; TR #2 SEC #0
LEA DISK_START+DISKOFFSET_FILETABLE,A1
MOVE.L #$90,D0 ; SIZE OF FILETABLE
COPY_FT: MOVE.B (A0)+,(A1)+
         DBF D0,COPY_FT

MOVEQ #0,D0 ; END OF PATCHING!!
RTS
; -----
```



Avant toute exécution, sauvegarder le code sur votre disquette de sauvegarde.

Celle-ci doit maintenant contenir le fichier **dump**, le fichier **DISKIMAGE\_RNC.s** et bien sur le binaire **rnloader.bin** du **sectorloader**

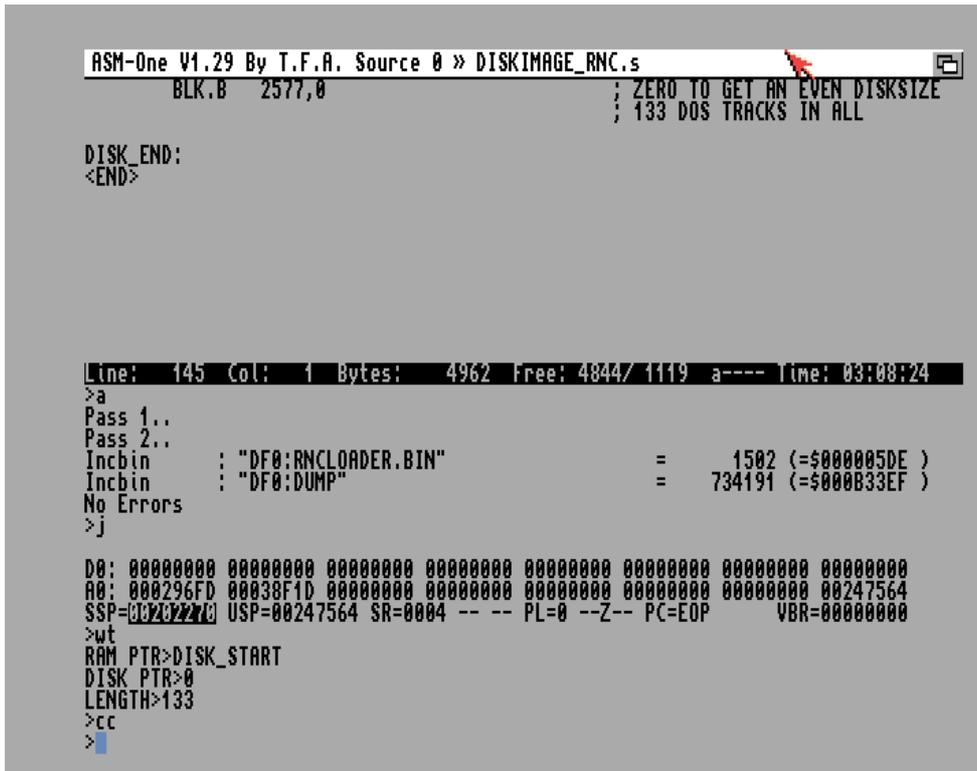
**Basculer** en mode **'command line'** en appuyant sur **ESC** puis compiler le tout à l'aide de la commande **A**, puis exécuter le code avec la commande **J**

Insérer ensuite une nouvelle disquette vierge et **Taper** :

```
WT  
RAM PTR>DISK_START  
DISK PTR>0  
LENGTH>133
```

Sans oublier la commande pour calculer le nouveau checksum du bootblock (sinon cela ne bootera pas)

**CC**



```
ASM-One V1.29 By T.F.A. Source 0 » DISKIMAGE_RNC.s  
BLK.B 2577,0 ; ZERO TO GET AN EVEN DISKSIZE  
; 133 DOS TRACKS IN ALL  
  
DISK_END:  
<END>  
  
Line: 145 Col: 1 Bytes: 4962 Free: 4844/1119 a---- Time: 03:08:24  
>a  
Pass 1..  
Pass 2..  
Incbin : "DF0:RNLOADER.BIN" = 1502 (=000005DE )  
Incbin : "DF0:DUMP" = 734191 (=000B33EF )  
No Errors  
>j  
D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  
A0: 000296FD 00038F1D 00000000 00000000 00000000 00000000 00000000 00247564  
SSP=00202270 USP=00247564 SR=0004 -- -- PL=0 --Z-- PC=EOP VBR=00000000  
>wt  
RAM PTR>DISK_START  
DISK PTR>0  
LENGTH>133  
>cc  
>
```

Une fois le disk crée, redémarrer votre Amiga avec cette même disquette dans le lecteur et... apprécier le jeu 😊





48 E7 7F FC 4E 56 FF DC B6 7C 00 03 66 08 61 00 00 DC 60 00 00 CE 38 00 02 44 00 03 3D 44 FF DC 3D 41 FF DE 3D 42 FF E0  
3D 43 FF E2 2D 48 FF E4 2D 49 FF E8 E4 58 02 40 00 01 52 40 3D 40 FF EC 70 00 36 02 67 72 70 1E D6 41 B6 7C 06 E0 6E 00  
00 92 02 81 00 00 FF FF 82 FC 00 0B 0C 6E 00 01 FF EC 67 02 D2 41 3D 41 FF EE 48 41 3D 41 FF F0 61 00 04 32 30 2E FF F0  
72 0B 92 40 B2 6E FF E0 6F 04 32 2E FF E0 3D 41 FF F2 61 00 00 BC 66 28 30 2E FF E0 90 6E FF F2 67 1E 3D 40 FF E0 30 2E  
FF F2 E1 88 D0 80 D1 AE FF E4 42 6E FF F0 30 2E FF EC D1 6E FF EE 60 BC 2F 00 61 00 03 BA 20 1F 67 20 72 00 32 2E FF EE  
0C 6E 00 01 FF EC 67 02 E2 49 C2 FC 00 0B D2 6E FF F0 D2 6E FF FA 2F 41 00 28 4E 5E 4A 80 4C DF 3F FE 4E 75 3D 7C 00 03  
FF DC 7A 01 61 00 03 AE 61 00 03 8A 38 2E FF DC 56 44 61 16 B0 BC FF FF FF FF 66 04 57 44 09 C5 53 6E FF DC 66 DE 20 05  
4E 75 76 1F 70 00 61 0A D5 02 D1 80 51 CB FF F8 4E 75 72 FF 09 81 13 C1 00 BF D1 00 08 39 00 05 00 BF E0 01 57 C2 09 C1  
13 C1 00 BF D1 00 4E 75 78 02 42 6E FF FC 42 6E FF FA 42 6E FF F8 34 2E FF EE 61 00 03 60 66 00 00 9E 70 1D 08 39 00 02  
00 BF E0 01 67 00 00 90 2A 6E FF E8 4B ED 04 00 2A BC AA AA AA AA 3B 7C 44 89 00 04 61 00 01 D0 61 00 00 94 66 70 30 2E  
FF F4 67 40 C0 FC 04 40 41 ED 00 06 61 00 02 62 49 F9 00 DF F0 1E 61 00 00 CE 66 6A 30 2E FF FA 90 6E FF F2 67 64 2A 6E  
FF E8 4B ED 04 00 30 2E FF F4 C0 FC 04 40 DB C0 2A BC AA AA AA AA 3B 7C 44 89 00 04 30 2E FF F6 67 18 C0 FC 04 40 41 ED  
00 06 61 00 02 1C 49 EE FF FE 42 54 61 00 00 88 66 24 30 2E FF FA 90 6E FF F2 67 1E 70 1A 2F 00 74 02 61 00 02 B8 61 00  
03 02 20 1F 08 39 00 02 00 BF E0 01 67 04 51 CC FF 32 60 00 02 46 74 0A 41 ED 00 06 30 3C 00 40 61 00 01 D6 61 00 02 10  
66 36 61 00 01 6A 67 06 51 CA FF E6 60 2C 61 00 01 28 66 2A B2 6E FF EE 66 24 B4 3C 00 0B 6C 1E B6 3C 00 0B 6E 18 53 03  
3D 43 FF F4 3D 7C 00 0B FF F6 97 6E FF F6 70 00 4E 75 70 18 4E 75 70 1B 4E 75 70 19 4E 75 2A 6E FF E8 4B ED 04 00 30 2E  
FF F8 C0 FC 04 40 DB C0 20 3C 00 00 17 70 61 00 03 2C 08 2C 00 01 00 01 66 00 00 94 61 00 03 10 67 00 00 90 4A AD 04 40  
67 E8 61 00 00 F2 66 BA 61 00 00 B6 66 B8 B2 6E FF EE 66 B2 36 02 B6 6E FF F0 6D 5C 30 2E FF F2 D0 6E FF F0 B6 40 6C 50  
08 2C 00 01 00 01 66 56 30 2E FF FC 07 00 66 40 41 ED 00 40 32 3C 04 00 61 00 00 C8 2F 00 41 ED 00 38 61 00 00 94 B0 9F  
66 00 FF 78 08 2C 00 01 00 01 66 2A 61 30 41 ED 00 40 22 6E FF E4 D3 C1 61 00 00 C6 61 30 30 2E FF FA B0 6E FF F2 67 0E  
52 6E FF F8 0C 6E 00 0B FF F8 66 00 FF 4A 70 00 4E 75 70 FF 4E 75 22 03 92 6E FF F0 20 3C 00 00 02 00 C2 C0 4E 75 30 2E  
FF FC 07 C0 3D 40 FF FC 52 6E FF FA 4E 75 20 4D 72 0A 70 00 41 E8 04 40 20 80 51 C9 FF F8 4E 75 41 ED 00 08 61 1A 36 00  
02 43 00 FF 34 00 E0 4A 48 40 32 00 02 41 00 FF E0 48 B0 3C 00 FF 4E 75 20 18 22 18 02 80 55 55 55 55 02 81 55 55 55 55  
D0 80 80 81 4E 75 61 0C 2F 00 41 ED 00 30 61 E0 B0 9F 4E 75 41 ED 00 08 72 28 2F 02 E4 49 53 41 70 00 24 18 B5 80 51 C9  
FF FA 24 1F 02 80 55 55 55 55 4E 75 08 39 00 06 00 DF F0 02 66 F6 4E 75 48 E7 F0 E0 70 7F 45 E8 02 00 26 3C 55 55 55 55  
22 18 24 1A C2 83 C4 83 D2 81 82 82 22 C1 51 C8 FF F0 4C DF 07 0F 4E 75 43 F9 00 DF F0 00 33 7C 40 00 00 24 33 7C 82 10  
00 96 33 7C 66 00 00 9E 33 7C 95 00 00 9E 33 7C 44 89 00 7E 23 48 00 20 33 7C 00 02 00 9C E2 48 00 40 80 00 33 40 00 24  
33 40 00 24 4E 75 43 F9 00 DF F0 00 20 3C 00 00 09 C4 61 00 01 70 08 29 00 01 00 1F 66 0A 61 00 01 56 66 F2 70 FF 60 02  
70 00 33 FC 00 02 00 DF F0 9C 33 FC 40 00 00 DF F0 24 4A 80 4E 75 33 FC 04 00 00 DF F0 9E 4A 6E FF E2 6A 1E 72 FF 13 C1  
00 BF D1 00 30 2E FF DC 56 80 01 81 13 C1 00 BF D1 00 01 C1 13 C1 00 BF D1 00 4E 75 72 FF 13 C1 00 BF D1 00 08 81 00 07  
61 D4 20 3C 00 00 00 C8 60 00 00 E2 48 E7 30 00 26 02 61 00 00 9E 30 2E FF DC D0 40 41 FA 01 08 30 30 00 00 6A 04 61 32  
66 2A E2 48 E2 4A 72 01 94 40 67 0E 6A 04 72 FF 44 42 70 03 61 50 53 42 66 F8 30 2E FF DC D0 40 41 FA 00 DC 31 83 00 00  
61 60 70 00 4C DF 00 0C 4E 75 48 E7 20 00 74 55 08 39 00 04 00 BF E0 01 67 0E 70 03 72 FF 61 1E 51 CA FF EE 70 1E 60 10  
30 2E FF DC D0 40 41 FA 00 A6 42 70 00 00 70 00 4C DF 00 04 4E 75 2F 00 61 2A 4A 01 6B 04 08 80 00 01 08 80 00 00 13 C0  
00 BF D1 00 08 C0 00 00 13 C0 00 BF D1 00 20 1F 60 3A 61 08 13 C0 00 BF D1 00 4E 75 48 A7 60 00 30 2E FF DC 14 39 00 BF  
D1 00 00 02 00 7F 56 00 01 82 57 00 D0 40 32 3B 00 4E 08 01 00 00 67 04 08 82 00 02 10 02 4C 9F 00 06 4E 75 61 1E 08 39  
00 00 00 BF DE 00 66 F6 53 80 66 F0 4E 75 08 39 00 00 00 BF DE 00 66 1C 53 80 67 18 13 FC 00 08 00 BF DE 00 13 FC 00 CC  
00 BF D4 00 13 FC 00 02 00 BF D5 00 4E 75 FF FF FF FF FF FF FF FF