Tutoriel	**AmigaCracking : MFM : TURRICAN II**
Protection	**MFM (multi file)**
Auteur Original	**aLpHa oNe**
Soumit par (sur flashtro.com)	**aLpHa oNe [2004-09-12]**
Version	**17/03/2018 Gi@nts**
Vérification/Correction	**V2, Testé et fonctionnel de A à Z**

* TURRICAN II CRACK TUTORIEL *

Table des matières

Matériels nécessaires 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 Ripage des fichiers #1 20
Part 3 Ripage des fichiers #2 22
Part 4 Comprendre la routine Trackloading 23
Part 5 Décompression fichier MAIN 25
Part 6 Création de notre nouvelle filetable 26
Part 7 Création de notre propre trackloader 28
Part 8 Patch des fichiers LOADER et MAIN 32
Part 9 Créer Les images Disque 34

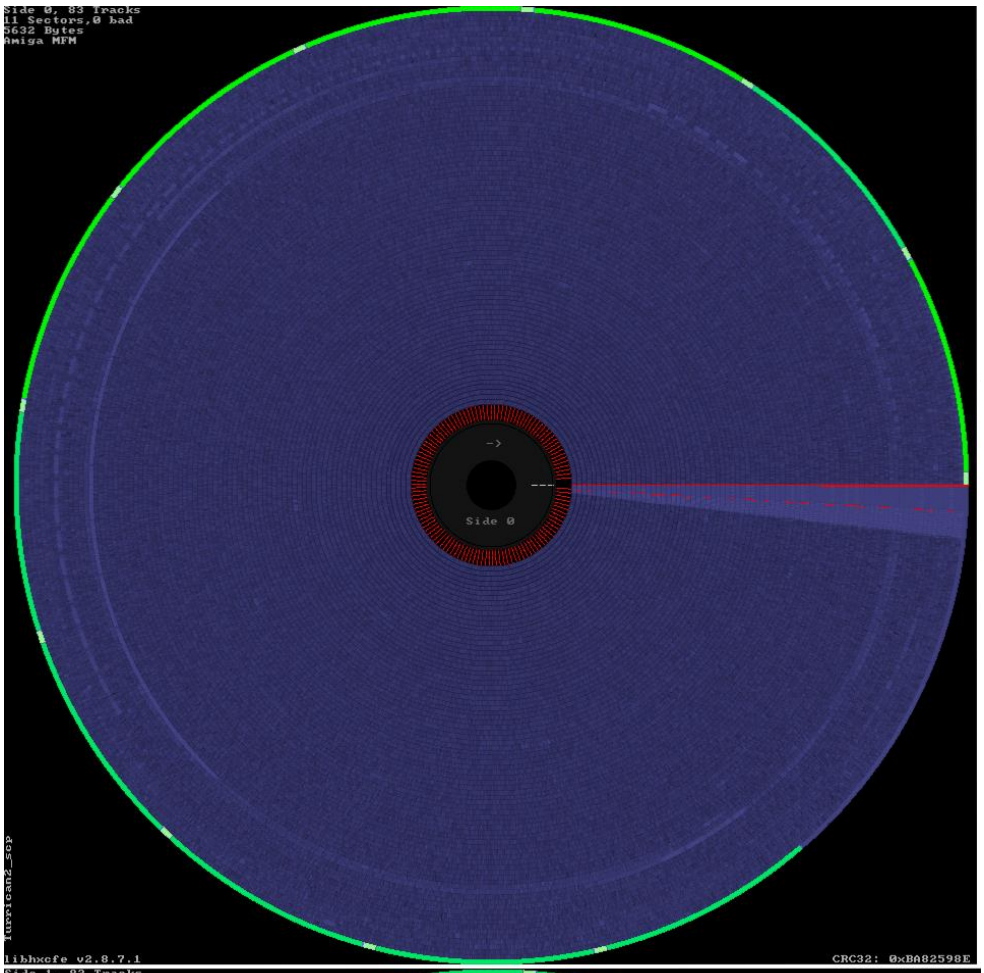
Matériels nécessaires

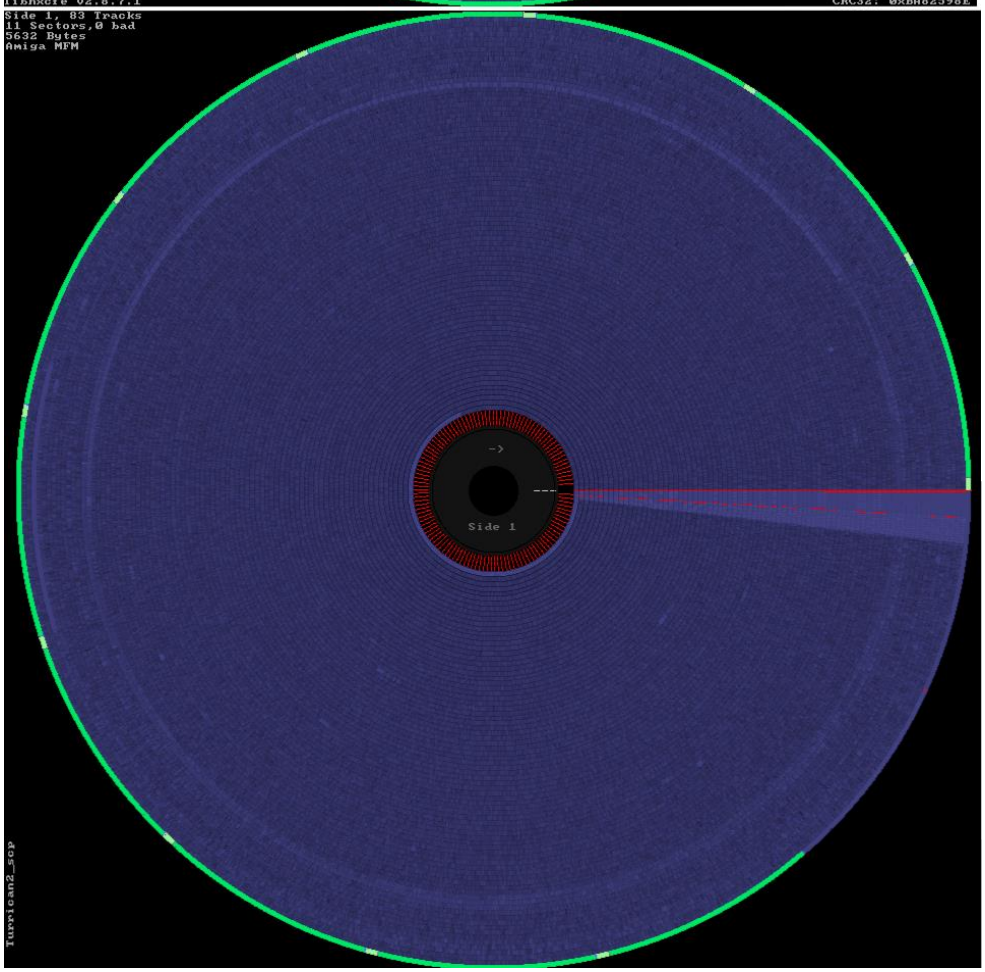
1. Un AMIGA avec 1Mega Chip + 512 Slow ou l'émulateur WinUAE.
2. Une carte ACTION REPLAY MKIII si l'on passe par un Amiga réel ou ça ROM Image si on passe par WinUAE.
3. Il est fortement conseillé d'avoir un second lecteur de disquette ou un disque dur.
4. Le jeu Original turrican II ou son image CAPS (SPS 0029)
5. Le logiciel Powerpacker ou XFD
6. Un assembler (ASM-One / Trash-M One / Seka ou similaire)
7. Les connaissances pour utiliser un des assembleurs ci-dessus !

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.
Il est tout à fait possible de réaliser ce crack avec une configuration H/W différente mais il faudra dans ce cas adapter par vous-même les étapes intermédiaires.

Bon tuto.
Gi@nts


Side 0, 83 Tracks
11 Sectors,0 bad
5632 Bytes
Amiga MFM
->
Side 0
Turrican2_scp
libhxcfe v2.8.7.1
CRC32: 0xBA82598E

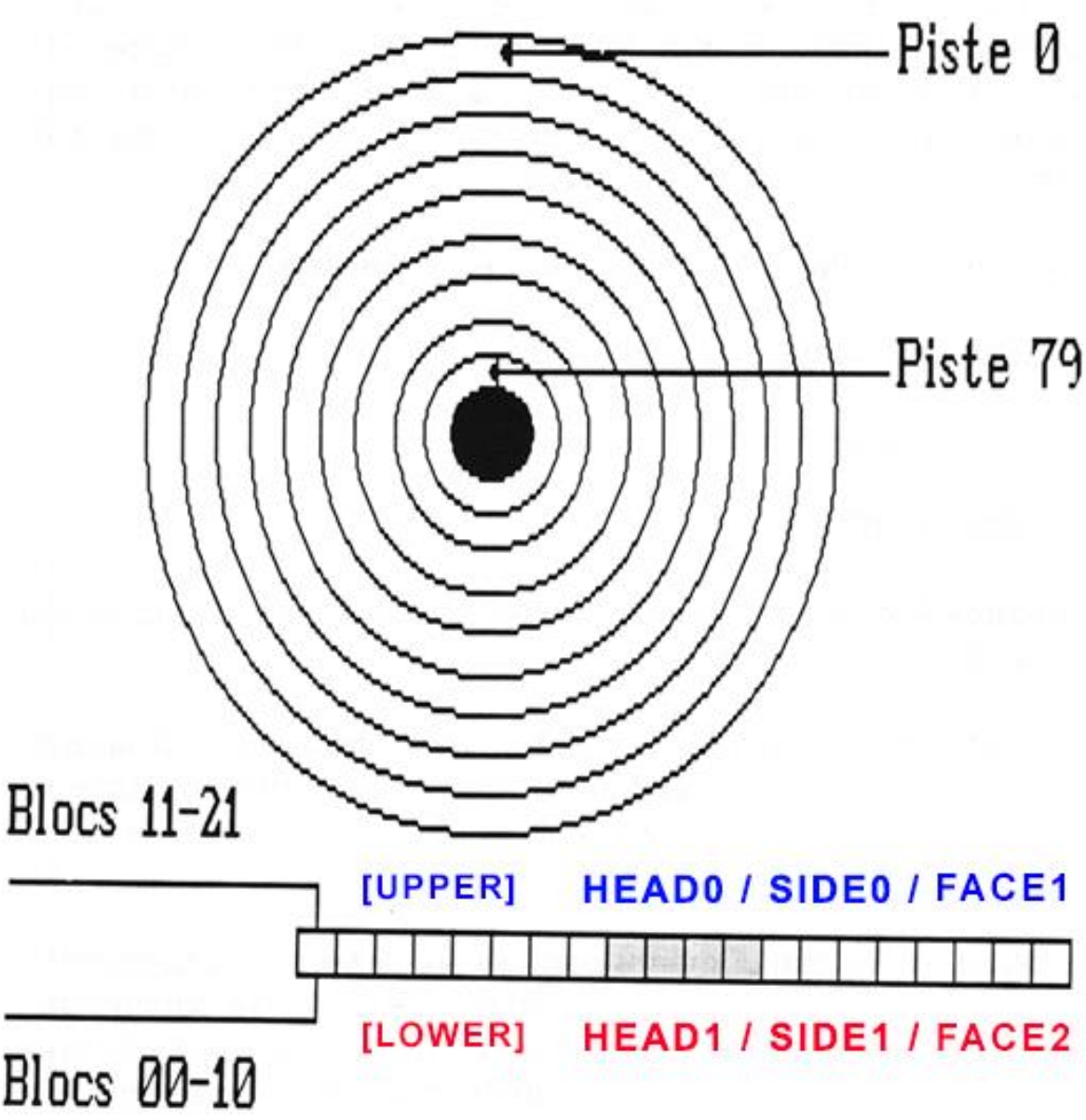

Side 1, 83 Tracks
11 Sectors,0 bad
5632 Bytes
Amiga MFM
->
Side 1
Turrican2_scp

Agencement des disquettes Amiga v1.1

En France :

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

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



Dans d'autre pays :

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

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

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

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

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

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

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

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

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

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

On notera que :

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

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

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

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

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

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

Mise en application sous l'AR :

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

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

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

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

Exemples :

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

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

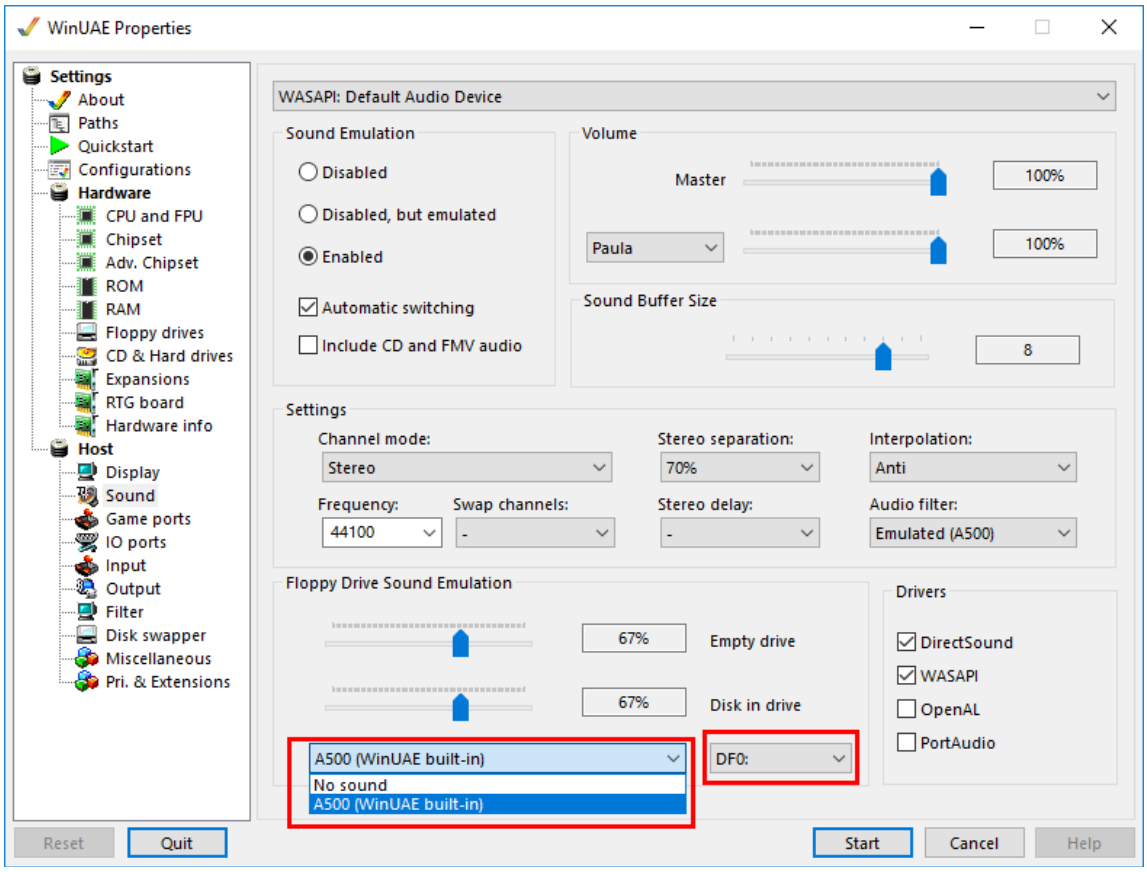
$20/2=$10 = !16 (en décimal donc) et comme il n'y a pas de retenu on est sur la face0.

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

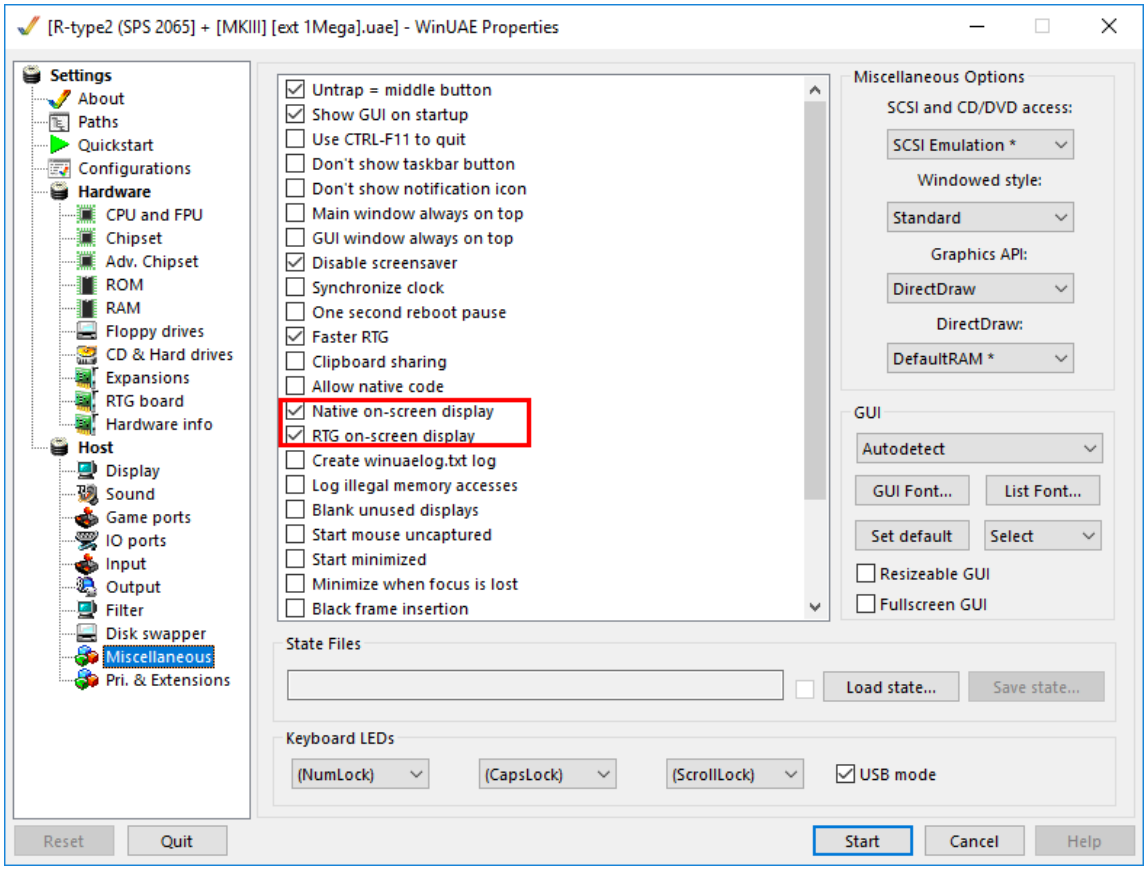
21 est donné en hexa **donc $21 = !33** en décimal.
33/2 = 16.5 ,donc piste 16 side 1 et comme on continue à lire/copier les donnes (**taille à lire =2**), on continue la copie.
On change donc de track car on est déjà sur la face 1 (il existe que 2 faces sur une disquette)
On arrive donc sur la prochaine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

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

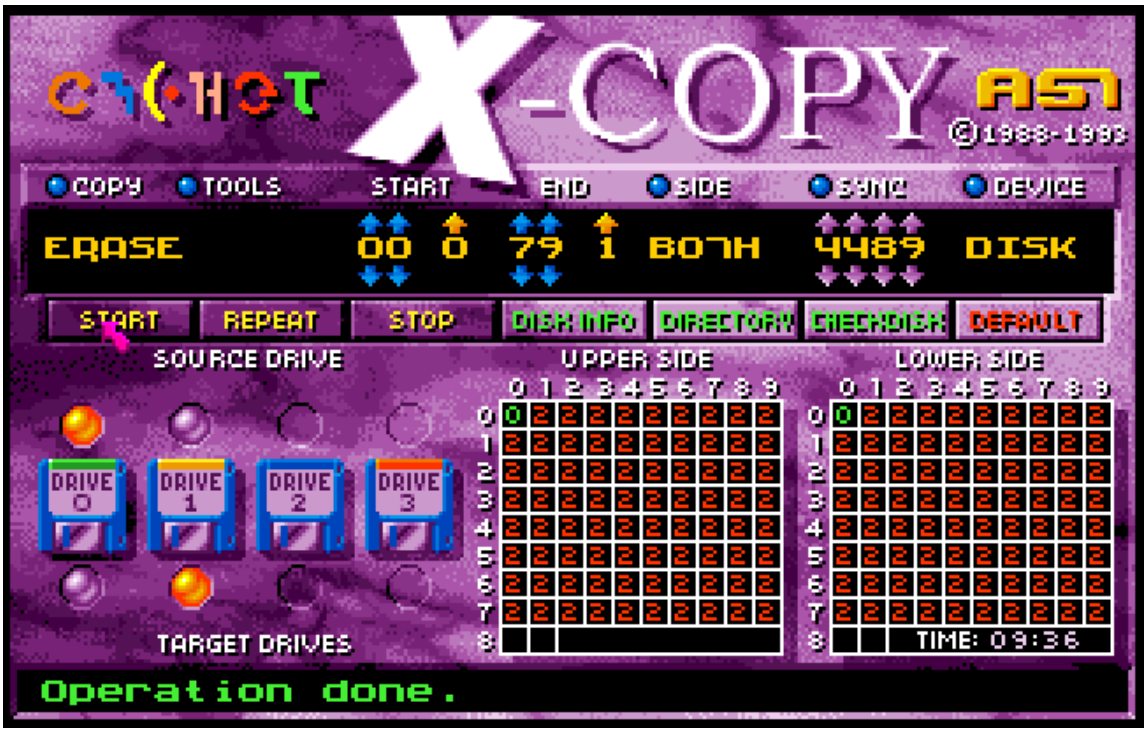


Voir même, pour plus d'information. Par exemple afficher sur qu'elle face l'on se trouve, d'activer :
Host -> Miscellaneous -> Native on-screen display AND RTG on-screen display



Part 1 X-Copy

Comme dans d'autres tutoriel, nous allons essayer dans un premier temps d'effectuer une copie du disk du jeux original **Turrican II**



Si on regarde de plus près l'écran de '*X-Copy*', nous pouvons voir que ce disque n'est pas vraiment copiable.
On a là un format bien spécifique.

Pas besoin d'être clairvoyant pour deviner que la copie ne fonctionnera pas !

PS : *Lors de la copie, on passe de l'erreur 5 à l'erreur 2 sur la plupart des pistes.*

Petit 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	0029_Turrican2.ipf
TYPE	Floppy_Disk
ENCODER	CAPS(V1)
FILE	29(V1)
DISK	0
TRACK	00-83
SIDE	0-1
PLATFORM	Amiga
REVOLUTION	4

Concernant la piste 00.0 et 00.1 rien de bien spécial : 11 secteurs de 512 bytes, standard AmigaDOS

TRACK		Data Length (bytes)		Data (bits)				CRC32 of the complete Extra Data Block			Adress
Data block Description	Sector ID	Data		bytes/sector	GAP		Codage	GapDef	DataOff		Adresse
		MFM bits	bytes		MFM bits	bytes			MFM bits	bytes	
[T00.0]		6446		51568				6EC760D5			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	15188	950	MFM	5892	5892	369	13896-13927
[T00.1]		6446		51568				B9BE315B			20050-26495
#0	0	8704	545	512	0	1	MFM	0352	0352	15	20050-20081
#1	1	8704	545	512	0	1	MFM	0906	0906	57	20082-20113
#2	2	8704	545	512	0	1	MFM	1460	1460	92	20114-20145
#3	3	8704	545	512	0	1	MFM	2014	2014	126	20146-20177
#4	4	8704	545	512	0	1	MFM	2568	2568	161	20178-20209
#5	5	8704	545	512	0	1	MFM	3122	3122	196	20210-20241
#6	6	8704	545	512	0	1	MFM	3676	3676	230	20242-20273
#7	7	8704	545	512	0	1	MFM	4230	4230	265	20274-20305
#8	8	8704	545	512	0	1	MFM	4784	4784	300	20306-20337
#9	9	8704	545	512	0	1	MFM	5338	5338	334	20338-20369
#10	10	8704	545	512	15190	950	MFM	5892	5892	369	20370-20401

Ensuite, **toutes les pistes** se ressemblent. A savoir pas de secteur standard, un codage MFM et une taille de $6849 au niveau Data Track
Et ce, jusqu'à la piste 79.1

TRACK		Data Length (bytes)		Data (bits)				CRC32 of the complete Extra Data Block			Adress
[T01.0]		6849		54792				08E9E8E3			26524-33372
#0	N/A	108928	6809	N/A	2008	126	MFM	0032	0032	2	26524-26555

Ce qui, au passage, valide ce que l'on a déjà vue sous Xcopy :

Part 3 Let's Rock'n Roll

Commençons donc par charger la premier piste lisible **dos** dans la mémoire pour voir ce qui se passe ...
#RT alias Read Track, permet le chargement de donnée à partir de la track 0 et d'une taille de 1
#On lie donc ici uniquement la 1er face de la piste 0 et on copie le tout à l'adresse mémoire 50000
Le disque original de **Turrican 2** est inséré dans le lecteur **DF0**
Entrer directement dans votre AR et **taper** le texte suivant : `RT 0 1 50000`

Désassemblons le *bootcode*
#D, alias Désassemble
Taper `D 5000`

```
d 50000
~050000 NEG.W   A7
~050002 SUBQ.B  #1,D0
~050004 SUB.B   D0,-(A5)
~050006 MOVE.L  CCR,CCR
~050008 ORI.B   #70,D0
~05000C MOVE.W  #1A0,00DFF096
~050014 CLR.W   00DFF180
~05001A MOVE.L  A1,-(A7)
~05001C CLR.L   D5
~05001E CLR.L   D7
~050020 MOVEQ   #4,D6
~050022 BSR     0005008A
~050026 MOVEQ   #2,D6
~050028 BSR     0005008A
~05002C MOVE.L  D7,-(A7)
~05002E MOVEA.L 4(A7),A1
~050032 MOVE.W  #2,1C(A1)
~050038 MOVE.L  #60000,28(A1)
~050040 MOVE.L  #800,24(A1)
~050048 MOVE.L  #400,2C(A1)
~050050 JSR     -1C8(A6)
~050054 MOVE.W  #9,1C(A1)
~05005A CLR.L   24(A1)
```

Défiler vers le bas, comme dans l'image ci-dessous.

La première chose intéressante que nous pouvons voir ici est typique.

A partir de l'adresse **$50032 - $50050** le '*trackdisk-device*' est utilisé pour lire **$800** octets
de la position du disque **$400** vers la destination mémoire **$60000**.

Dans les 2 dernières instructions à **$50080 / $50088**, une exception est causée et notre petit programme se poursuit dans la mémoire à l'adresse **$60000**.

Donc, ce sera probablement l'un des '*gameloaders*' !

Enregistrons dans un premier temps ces données sur une disquette.
Insérer une disquette vierge dans **DF0** et sauvons tout ceci.
#format, Permet de formater une disquette au format AmigaDOS
#SM, alias SaveMemory permet donc de sauver une zone mémoire vers un fichier.

Taper :
```
FORMAT Turrican2_1
```
Ready to format disk in drive DF0: (y/n)? y
Disk ok

```
SM BOOTBLOCK, 50000 50400
SM LOADER, 50400 50400+800
```

Poursuivons nos travaux avec le désassemblage du '*loadercode*'...
Nous pourrions déjà désassembler à partir de l'adresse **$50400** mais, dû au fait que nous savons que le chargeur est situé à l'adresse mémoire **$60000**, nous allons dans un premier temps le charger à cette adresse avant de continuer...

#LM alias loadMemory. Permet de charger un fichier vers une zone mémoire spécifique.
Taper : `LM LOADER, 60000`
Désassemblons tout ça avec la commande `D 60000`
Vous allez voir exactement ce code :

```
d 60000
~060000 MOVE.L  D0,0006015E
~060006 LEA     00DFF000,A6
~06000C MOVE.W  #7FFF,9A(A6)
~060012 MOVE.W  #7FFF,96(A6)
~060018 LEA     0007FFF0,A7
~06001E MOVE.L  #600CA,80(A6)
~060026 CLR.W   88(A6)
~06002A MOVE.W  #8380,96(A6)
~060030 MOVE.L  #52412D49,D0
~060036 MOVE.L  #30000,D1
~06003C MOVEQ   #1,D2
~06003E BSR     00060164
~060042 JSR     0003010A
~060048 MOVE.L  #4D41494E,D0
~06004E MOVE.L  #C0,D1
~060054 MOVEQ   #1,D2
~060056 BSR     00060164
~06005A MOVE.W  #7FFF,00DFF09A
~060062 MOVE.W  #F,00DFF096
~06006A CLR.L   00DFF0A8
~060070 CLR.L   00DFF0B8
~060076 CLR.L   00DFF0C8
~06007C CLR.L   00DFF0D8
~060082 LEA     000383C8,A0
```

Le code commence avec des instructions standards comme le *kill* de toutes les interruptions, le pointage de la pile de registre vers une zone quelconque de la mémoire, la définition d'une '*copperlist*' et ainsi de suite...
Mais après ça, on trouve quelques lignes vraiment intéressantes.

Jeter un coup d'œil à l'instruction située en **$60030**

La valeur du 'longword' **$52412D49** est déplacée dans **d1** (peut être une adresse mémoire ?)
et d'autres paramètres sont déplacés dans **d2($1)** avant d'appeler la routine située en **$60164**
Apres le retour de cette sous-routine, le programme saute en **$3010A**

Hummm **$3010A** ?!
Cela semble familier avec la valeur déplacée en **d1 ($30000)** avant le branchement en **$60164**
Allons jeter un coup d'œil au 4 prochaines instructions.

Une valeur est encore déplacée dans **d0 ($4D41494E.l)**
Une autre dans **d1 ($c0.l)** même si **d2** est encore défini à la valeur **$1**

Et bien, si vous continuez à désassembler quelques lignes (pas d'image inclus),
vous pouvez voir que le programme continue sont exécution à l'adresse **$C0** !!!

$c0 ? Ce n'étais pas la valeur dans **d1** avant que nous nous branchions en **$60164** la seconde fois ?

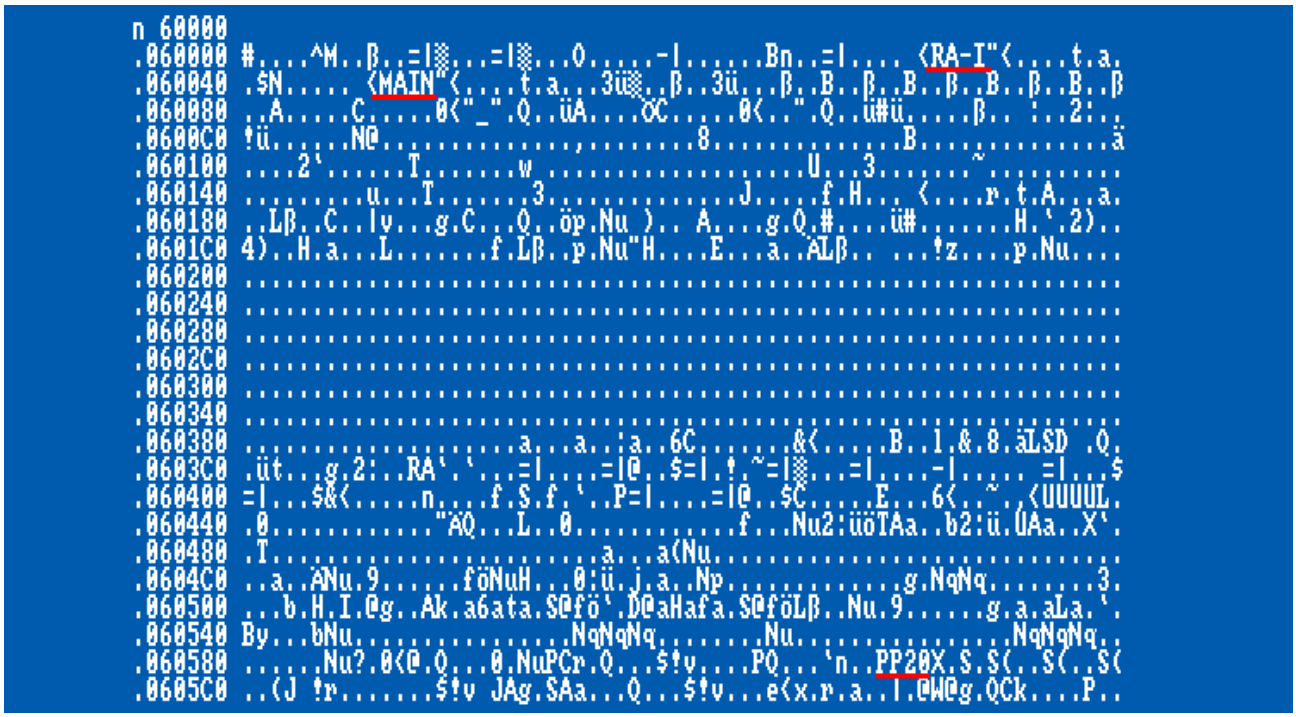
Après avoir pris quelques autres bouteilles de bière, je suis arrivé au point que :

$60164 semble être une sorte de '*loaderroutine*' qui est appelée avec 3 paramètres.

D1 pourrait être l'adresse à laquelle le 'loader' charge les pistes.
Mais que signifie la valeur dans **d0** ?

Allons voir à l'adresse **$60000** en ASCII (dans le loader donc)
#N, alias memory read. Permet de voir les données en mémoire au format uniquement ASCII.
Taper : `n 60000`

et regardons tout ça plus en détail !



Dans l'image ci-dessus, est souligné en rouge les choses intéressantes.

Nous pouvons voir ici les 3 valeurs *longwords* : **RA-I, MAIN** et **PP20** (Une entête *Powerpacker* ?)
Ce qui est vraiment intéressant, c'est la centaine d'octets entre le code qui sont complètement vide.

Après vérification de la manière dont les deux valeurs **RA-I** et **MAIN** sont inclus dans le code,
je me suis aperçu que c'est la chose qui est déplacé vers **d0** avant que notre '*loader*' soit appelé.
Ainsi, le '*loadercode*' se présente comme suit :

```
move.l #"RA-I",d0
move.l #$30000,d1
moveq  #1,d2
bsr    $60164
jsr    $3010a
move.l #"MAIN",d0
move.l #$c0,d1
moveq  #1,d2
bsr    $60164
```

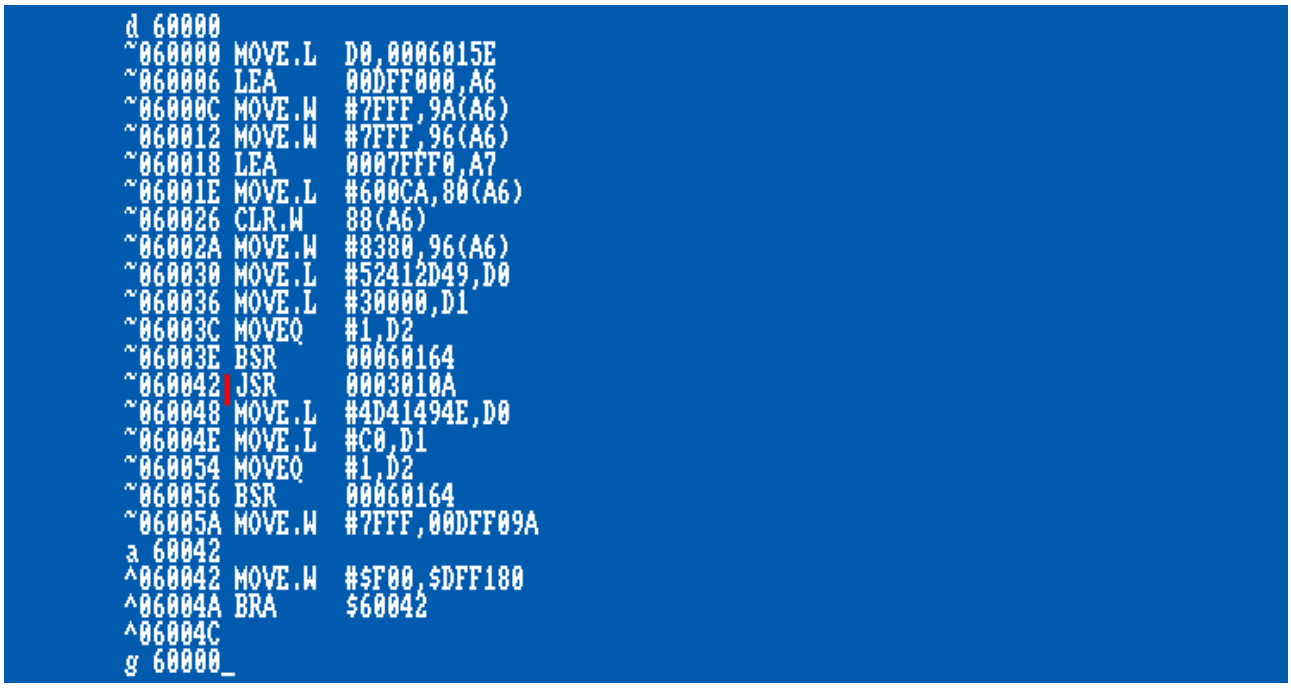
d0 semble contenir quelques chose comme le **nom**.

Mais si c'est le cas, le '*gameloader*' doit probablement avoir quelque part quelque chose comme une 'lookup-table/filetable'
qui dit au '*trackloader*' où ces fichiers sont stockés sur le disk.

Mais, impossible de trouver en mémoire quelque chose de ce type. (**de $60000 - $60800**)

De toute façon...
Essayons d'abord de ripper ces deux fichiers.
Nous trouverons probablement la signification du paramètre **d2**
Peut-être allons-nous découvrir une '*lookup-table*' dans la mémoire lorsque le '*loader*' sera appelé la première fois.

Insérer maintenant votre disque Original de Turrican II et allons faire quelques modifications dans le '*loadercode*'



Nous allons insérer 2 lignes de code à l'adresse **$60042** juste après le retour de la probable routine de '*trackloading*'
Cela introduira une boucle infinie avec un écran rouge !

Taper :
#A, alias Assemble, Instruction qui va permettre de taper du code assembleur.
```
a 60042
^60042 move.w #$f00,$dff180
^6004a bra $60042
```

Sautons ensuite dans le '*loadercode*', **taper** :
#G comme GO, démarre le code à l'adresse indiquée.
```
g 60000
```

Après quelques chargements de piste l'écran tourne au rouge et plus rien de se passe.
Le '*trackloader*' a donc fini et le processeur semble avoir accroché notre boucle.

Entrer donc dans l'AR pour voir ce qui se passe.

Si le paramètre dans **d1 ($30000)** est vraiment une adresse mémoire, nous allons peut-être voir quelque bout code intéressant. Allons voir tout ça !

Taper : `n 30000`

```
n 30000
.030000 `.#..h...ä#..l....`..ö,x..C...N..h#....ÖgFC...N..h#.....g,3..ß..
.030040 ....3ü▒..ß..#.......!ü...r..N@,x..N..b"y...ÖN..bNuFü'.#.....Q...
.030080 #..h...ä#..l....N.....Fü'.!..6..M..ß..=|▒....z.20:.,.@..=@.."z..
.0300C0 -i.&..Bn..=|.0..Ns........................graphics.library.dos
.030100 .library..a^Nu....f..9......g..9......g.S.....fü!ü.....la...Jy..
.030140 ..g.=|▒...=|▒...=|....=|....!....h!....lNuM..ß..=|....=|▒...=|▒.
.030180 ..=|....=|....B...A.x.0<..B.Q..üa..pA...C..FE...G...I.xr6<.Pp.r.
.0301C0 $.......(.Q..........ö.Q...!ü...Z.lNqNqNqNq=|....-|...f..=|....Fü
.030200  ..ü................NuH...C......)....g,.)........r6Q.........F...
.030240 ....g.J.k.3ü......=|....`PH...K...Jm..f.a...a...a....m.5.8f.!ü..
.030280 ...la..~`.Jm..f.a...a...Jm..f.ara...=|....Lß▒.NsH...a..~.m....m.
.0302C0 a...`.H...C...Si...j.3|....A...p.r.t.6.8..q .g.RA.q ..q ..D0.UBjä
.030300 XHQ...JAf.3|....`.R...0-..t..m..f.t.j.t.D@6@2-...m..j.DA8A6.SC m
.030340 .H0-...H2..I.A..(.).0-..2..I.@...@..x.JDg.SD`..p...ü.(.Fk.f.JBj.
.030380 S@j.p.RA`.R@.@..f.SA.LQ...Nu m..Jm..kRJm..f.;h....0..m..0(...m..
.0303C0 Sm..f2Sm..j&;|....A...p.1@..1@..=@..=@..=@..=@..Nu\H+H..NuJm..gJ
.030400 I..ß..a..XBl.`9|.".f9|...@Bl.B m..C.zd|.)H.T)I.H9|...Xa..(A."`C.
.030440 ..Q..äBm..Nup. m.H0-...@...ü.(..0-...@..)..G...H.@....+H..0-..R@
.030480 .@..f.p.;@..2..H.@.H.@.IE.u....IC.....G.y.I..ß..a...Bl.f9|.".`9|
.0304C0 ...D9|...@Bl.B|.)H.H)K.T9|...Xa...A."`G...Q..ä m..9|...d)I.P)K.T
.030500 Bl.F9|...@9|...Xa..N"KBl.b9|.".f0..H.H9@.B.@..9@.@|.)H.H)H.T)I.L
.030540 )J.P9|...Xa...A."`C...Q..üP...Nu.9...ß...,....f.Nup.Su..k.U@jöNu
.030580 25.,.A.5g.;.....Ru.,4..A.A.B.BA....."m.H..E..T...;....2(..4..A..
.0305C0 .B..SA).a.2..ü"`a.2..ü"`a.2..ü"`a.`.v.x(JBg.3.0.ÖDQ...Nu..0.ÖDQ.
```

C'est ça, vous êtes en mesures de voir que le '*loader*' a vraiment chargé des données non décompressées dans la mémoire.

Au lieu de déplacer **$1** dans **d2**, allons voir ce que le ‘loader’ fait si nous effaçons celui-ci.

Désassemblons le code encore une fois pour trouver ou l'instruction **moveq #1,d2** est exécutée,
et remplaçons-la par un **moveq #0,d2**

Entrer le texte suivant :

```
a 6003C
^6003c moveq #0,d2
^6003e
```

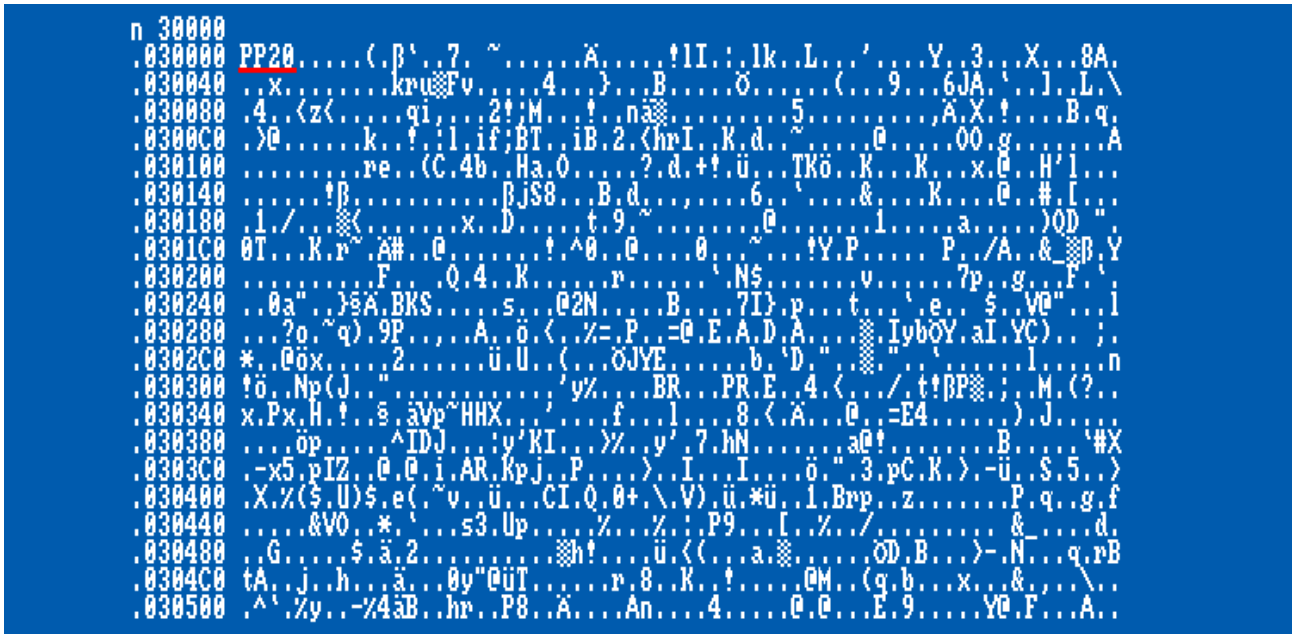
```
d 60006
~060006 LEA     00DFF000,A6
~06000C MOVE.W  #7FFF,9A(A6)
~060012 MOVE.W  #7FFF,96(A6)
~060018 LEA     0007FFF0,A7
~06001E MOVE.L  #600CA,80(A6)
~060026 CLR.W   88(A6)
~06002A MOVE.W  #8380,96(A6)
~060030 MOVE.L  #52412D49,D0
~060036 MOVE.L  #30000,D1
~06003C MOVEQ   #1,D2
~06003E BSR     00060164
~060042 MOVE.W  #F00,00DFF180
~06004A BRA     00060042
;=======================================
^06004C LINEF
~06004E MOVE.L  #C0,D1
~060054 MOVEQ   #1,D2
~060056 BSR     00060164

a 6003c
^06003C MOVEQ #0,D2
^06003E
g 60000_
```

Sauter vers le '*loader*' encore une fois en **tapant** : `g 60000`

Après quelques chargements de pistes, nous pouvons voir notre écran passer en rouge.
Entrer dans l'AR et jeter un coup d'œil à l'adresse $30000

Taper : `n 30000` et regardons ce qui s'est passé.



Yess... Les données commencent par *PP20* (qui est normalement une entête de donnée compressé par *PowerPacker*)
Ce qui signifie que le loader vient juste de lire les pistes du fichier **sans** les décompresser.

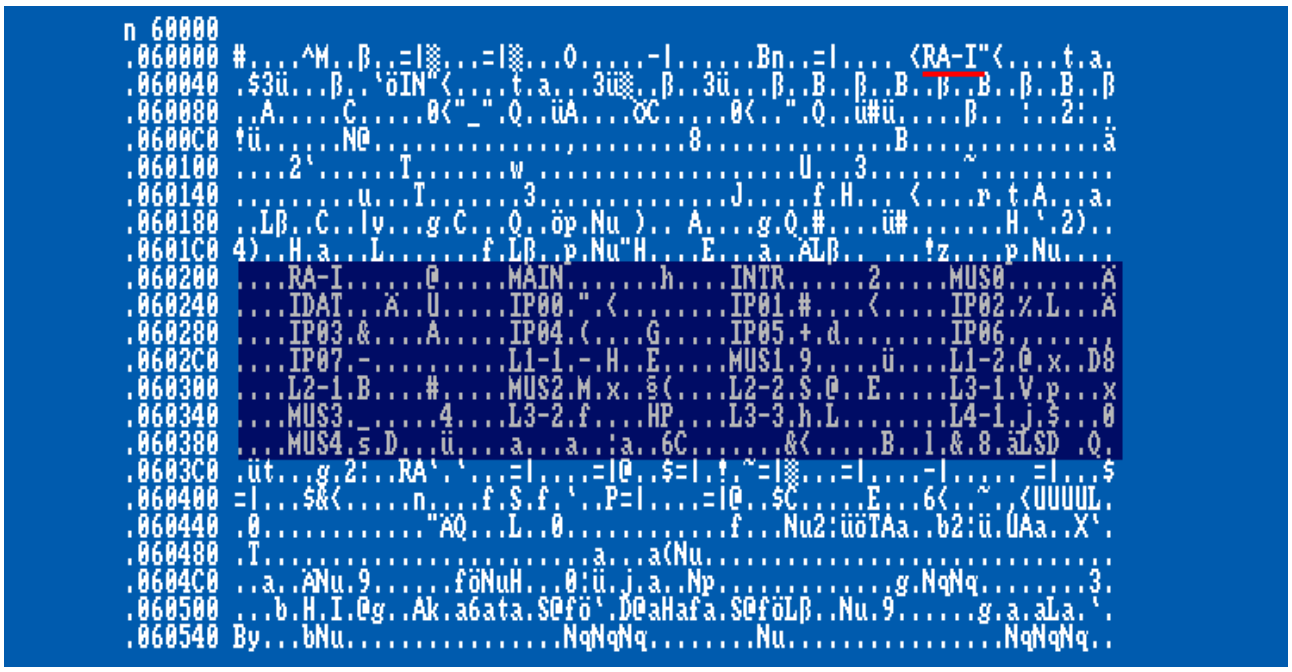
Donc le paramètre **d2** semble indiquer au '*loader*' si le fichier est compressé avec PowerPacker
ou s'il n'est pas compressé (ou compressé par autre chose !).

Maintenant nous savons (Espérons-le) comment ce '*loader*' fonctionne.

Regardons donc ce qui se passe dans notre code du loader à l'adresse **$60000**.

Nous avions parlé d'une certaine '*lookup-table*' quelques lignes plus haut.
Allons voir si quelque chose a changé.

Taper : `n 60000`



Maintenant, la région qui était vide avant que l'on est appelé le '*trackloader*' est remplie de données.
Regardons d'un peu plus près ces bytes.

Taper :
#M alias memory read. Permet de voir les données en mémoire au format HEXA/ASCII.
`m 60204` et regarder.

```
M 60204_
:060204 52 41 2D 49 00 02 01 E0 00 00 40 F0 00 00 01 09 RA-I......@.....
:060214 4D 41 49 4E 00 04 0D B0 00 00 E0 68 00 00 01 09 MAIN.......h....
:060224 49 4E 54 52 00 0C 19 98 00 00 32 88 00 00 01 01 INTR......2.....
:060234 4D 55 53 30 00 0E 17 00 00 01 A8 C4 00 00 00 01 MUS0.......Ä....
:060244 49 44 41 54 00 1E 16 C4 00 00 55 B8 00 00 01 01 IDAT...Ä..U.....
:060254 49 50 30 30 00 22 02 3C 00 00 1C F8 00 00 01 04 IP00."<........
:060264 49 50 30 31 00 23 04 A4 00 00 3C C8 00 00 01 04 IP01.#....<.....
:060274 49 50 30 32 00 25 0C 4C 00 00 18 C4 00 00 01 04 IP02.%.L...Ä....
:060284 49 50 30 33 00 26 0A 80 00 00 41 B0 00 00 01 04 IP03.&....A.....
:060294 49 50 30 34 00 28 17 10 00 00 47 04 00 00 01 04 IP04.(....G.....
:0602A4 49 50 30 35 00 2B 0E 64 00 00 0D 1C 00 00 01 04 IP05.+.d........
:0602B4 49 50 30 36 00 2C 00 F0 00 00 1A 2C 00 00 01 04 IP06.,.....,....
:0602C4 49 50 30 37 00 2D 00 8C 00 00 0F BC 00 00 01 04 IP07.-.........
:0602D4 4C 31 2D 31 00 2D 10 48 00 01 45 04 00 00 01 01 L1-1.-.H..E.....
:0602E4 4D 55 53 31 00 39 16 8C 00 00 A6 DC 00 00 01 01 MUS1.9.....ü....
:0602F4 4C 31 2D 32 00 40 03 78 00 00 44 38 00 00 01 01 L1-2.@.x..D8....
:060304 4C 32 2D 31 00 42 12 90 00 01 23 18 00 00 01 02 L2-1.B....#.....
:060314 4D 55 53 32 00 4D 11 78 00 00 A7 28 00 00 01 02 MUS2.M.x..§(....
:060324 4C 32 2D 32 00 53 19 40 00 00 45 E0 00 00 01 02 L2-2.S.@..E.....
:060334 4C 33 2D 31 00 56 0F 70 00 00 E9 78 00 00 01 02 L3-1.V.p...x....
:060344 4D 55 53 33 00 5F 09 D8 00 00 B7 34 00 00 01 02 MUS3._.....4....
:060354 4C 33 2D 32 00 66 07 1C 00 00 48 50 00 00 01 02 L3-2.f....HP....
:060364 4C 33 2D 33 00 68 1A 4C 00 00 1F F8 00 00 01 02 L3-3.h.L........
:060374 4C 34 2D 31 00 6A 05 24 00 00 FB 30 00 00 01 02 L4-1.j.$...0....
:060384 4D 55 53 34 00 73 11 44 00 00 EE FC 00 00 01 02 MUS4.s.D...ü....
```

Yesss, ça ressemble à une '*lookup-table*' pour moi ça, hein ?

Elle commence par le fichier **RA-I** et fini par le fichier **L4-1** et **MUS4** !
(Ce qui nous donnes une longueur de **400 Bytes** pour cette table !)

Et bien Ok, faisons une petite pause et mettons à plat ce que nous connaissons maintenant.

Nous sommes en état de pouvoir ripper les fichiers présents dans '*lookup-table*' en passant le nom du fichier dans **d0**
et en appelant la routine du *loader*.

Mais comment connaître la taille de chaque fichier ?! Est-ce que cette valeur est aussi inclus dans la '*lookup-table*,
et si oui, qu'elle valeur est-ce ?

Il y a beaucoup de valeurs entre chaque nom de fichier…
Commençons donc et trouvons ces bonnes valeurs grâce à quelques astuces simples.

Nous allons appeler le '*loader*' encore une fois pour qu'il charge les pistes du fichier **RA-I** dans la mémoire à l'adresse **$30000**
Mais cette fois, avants, nous allons avant remplir la zone-mémoire avec quelques bytes **A**.
Comme ça, quand le chargement sera fini, nous pourrons voir combien de '*bytes*' ont changés.

Taper :
#o, alias Fill mémory. Permet de remplir une zone mémoire avec une valeur hexa.
`o "A", 30000 50000`

Cela remplie la mémoire de **$30000** à **50000** avec la valeur donnée, **A**

Ensuite, faisons un autre saut vers notre routine de chargement, **taper** : `G 60000`

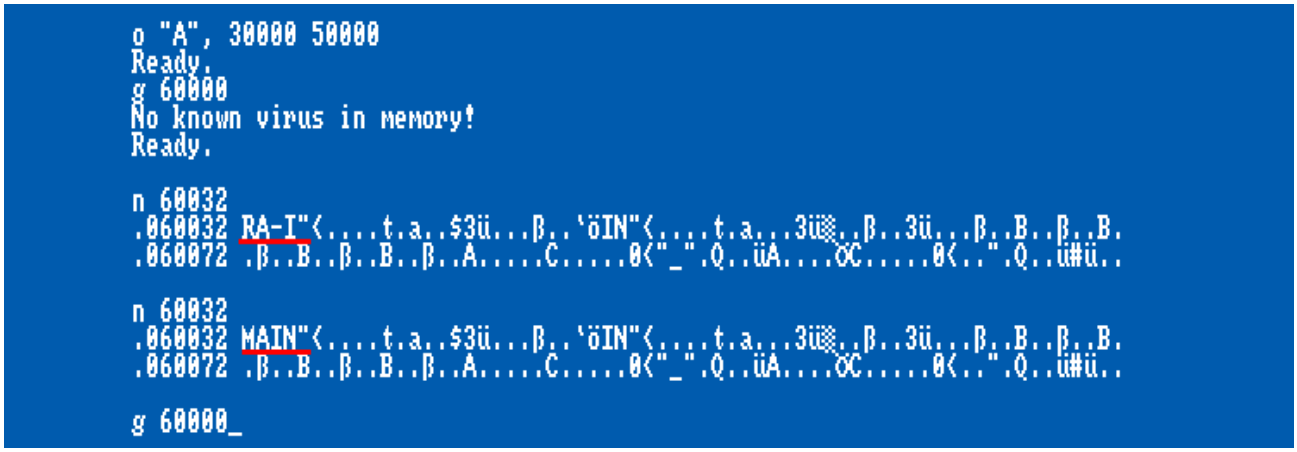
Après que notre écran au rouge apparaisse**. Entrer** dans l'AR encore une fois et descender
de quelques lignes depuis l'adresse **$30000** jusqu'à ce que nous voyions des centaines de bytes **A**.
`m 340A0`

Après avoir vérifié ou les premiers **A** apparaissent, nous notons une longueur de **$40F0 bytes** !!!
Noter cette valeur et regardons encore une fois dans la table des fichiers stockés en **$60204**

STRIKE ! Regarder la '*longword-value*' à l'offset **$8** depuis le début de la table. Ça doit être la taille.

```
M 60204_
:060204 52 41 2D 49 00 02 01 E0 00 00 40 F0 00 00 01 09 RA-I......@.....
```

Essayons maintenant de charger le fichier **MAIN** dans la mémoire pour nous assurer de ceci.
Nous remplissons donc encore une fois la mémoire à l'adresse **$30000** avec des **A** comme précédemment, puis faite ce qui suit :



Changeons maintenant la valeur qui a été mise dans **d0** de **RA-I** à **MAIN**

Taper : `n 60032`

(Nous prenons l'adresse **$60032** car les 2 premiers '*bytes*' concerne l'instruction : **move.l** !!!)

Remplacer les 4 *bytes* **RA-I** en **MAIN** et **appuyer** sur entrée pour que l'AR reconnaisse notre petit changement !
Ecriver bien **MAIN** et non main ou Main car le '*loader*' ne trouvera pas le fichier dans la '*lookup-table*' !

Continuons donc en sautant vers notre loader encore une fois en **tapant** :
`g 60000`

Après quelques chargements de pistes, l'écran tourne encore au rouge, nous savons donc que le chargement est fini.
Descendons dans la mémoire en **$30000** jusqu'à que nous atteignons notre zone remplie de **A**.
(pas de photo car même action que précédemment)

Pour info, **MAIN** semble aussi '*powerpacked*'

PS : Rappeler-vous la longueur des données de ce fichier (cella devrait être de **$E068 bytes**)
Comparons la valeur trouvée avec la valeur dans la '*lookup-table*' !!!

Nous avons raison, c'est la même.

Et bien, nous pourrions maintenant commencer à ripper tous les fichiers du disque original, mais nous ne le ferons pas encore !! ;)
Maintenant, nous connaissons beaucoup de chose sur la façon dont ce '*loader*' fonctionne.

Nous allons en premier temps démarrer le jeu pour vérifier ce qui se passe après le chargement du fichier **MAIN e**n mémoire **$C0** !

Du au faite que nous allons mettre ces fichiers sur un disque '*alterne*', nous allons devoir écrire notre propre '*trackloader*'.
Il pourrait être intéressant de connaitre en combien de partie le '*loader*' va charger le jeu, car nous allons devoir toutes les patcher !!

Ok, **effectuer un reset** de votre machine, **insérer le disk original** et lancer le.

Quand l'intro de Rainbox Arts est fini et que le jeu continu à charger quelques données du disque,
Entrer dans votre AR et allons **désassembler** le code à l'adresse **$C0** où le fichier **MAIN** devrait être situé.

La première instruction est un branchement vers une autre adresse mémoire **$123E**

```
d C0
~0000C0 BRA     0000123E
_
```

Continuons donc notre désassemblage à l'adresse **$123E**
Taper : `D $123E`

```
~00125E MOVEQ   #0,D1
~001260 MOVEA.L D0,A0
~001262 ADD.L   4(A0),D1
~001266 MOVE.L  0(A0),D0
~00126A BNE     00001260
~00126C SWAP    D1
~00126E LSR.W   #3,D1
~001270 MOVE.W  D1,0000155E.S
~001274 LEA     00008428,A7
~00127A LEA     00DFF000,A6
~001280 BSR     000070AE
~001284 BSR     000070D4
~001288 BSR     00003730
~00128C BSR     000014B8
~001290 BSR     00000BE0
~001294 MOVE.L  #494E5452,D0
~00129A MOVE.L  #20700,D1
~0012A0 BSR     000006F8
~0012A4 MOVE.L  #4D555330,D0
~0012AA MOVE.L  0002070C,D1
~0012B0 BSR     000006F8
~0012B4 JSR     00020700
~0012BA LEA     00DFF000,A6
~0012C0 BSR     000013CC
~0012C4 BRA     000051B4
```

Dans ces lignes de code, on peut apercevoir quelque chose de familier à l'adresse **$1294.**

Une autre routine est appelée deux fois avec un branchement en **$6F8** qui semble prendre les mêmes paramètres que nous avons déjà vue dans notre '*loader*' !

C'est bien car nous avons besoin de coder un nouveau '*trackloader*' pour notre '*loader*' en **$60000**

Avant de se lancer plus profondément dans notre '*trackloader*' et ce, juste pour voir s'il y a une autre '*lookup-table*' ou si l'ancienne de **$60204** est utilisé, allons jeter un petit coup d'œil en ASCII dans la mémoire à l'adresse **$C0**

Taper : `N C0`

Et, quelques lignes plus bas…

```
.000540 Jx.,f.`.Y.L.....n...n.0H.....J@k6JAk2.|.P1,.|..l&P...0-.2A....p
.000580 ....j.Bn.2.P..08.$.n.2NuQ...Nu.....................d.......d.2....
.0005C0 .d.d.....2.d.......d.......d.......d.......2.....................
.000600 ..........................2.......d.......d......A....@C...z.L...
.000640 H...C.. 0(..".Q..ü (..z."(..v.N....$N.....P..(P.l.p.N.....48\..J
.000680 A..F 0 ./0 ."(....t.a..b ."9....t.a..T (....N....8N....(J.fö0(..
.0006C0 N.....NuLl-1MUS1L2-1MUS2L3-1MUS3L4-1MUS4L5-1MUS5INTRMUS0J...f.H.
.000700 .. (....r.t.A...a...Lß..C...v...g.C...Q..öp.NuH.@@a...Lß..g.k."A
.000740 E....PXHa...p.Nu$).. ).. AH...2)..4)..H.a..xL.......g.$H.... ".!"
.000780 Y.f.!(PAK+X.H...6..|...x.^n&....g C...XH .......H...E....Pa...L.
.0007C0 ..6..|..UC.x.^n.$. ."..a...Lß......g.C...XH..E....Pa...p.Nu.....
.000800 _.RA-I.....@.....MAIN.....h....INTR.....2....MUS0.....Ä..
.000840 ..IDAT...Ä..U.....IP00."(.......IP01.#....(.....IP02.%.L...Ä..
.000880 ..IP03.&....A.....IP04.(....G.....IP05.+.d.......IP06..........
.0008C0 ..IP07.-.........L1-1.-.H..E.....MUS1.9.....ü....L1-2.@.x..D8..
.000900 ..L2-1.B....#.....MUS2.M.x..§(....L2-2.S.@..E.....L3-1.V.p...x..
.000940 ..MUS3._.....4....L3-2.f....HP....L3-3.h.L........L4-1.j.$...0..
.000980 ..MUS4.s.D...ü....L4-2.|.0..BÄ....L5-1.▒.D..Ud....MUS5...X......
.0009C0 ..L5-2...$..Q.....EDAT.............a...a...a..6C...@...&(.....B..
.000A00 1.&.8.ãLSD .Q..üt...g.28..RA`.`...=|....=|@..$=|.!.~=|▒...=|....
.000A40 -|..@.. =|...$=|...$&(.....n....f.S.f.`..P=|....=|@..$C...@.E...
.000A80 6(..~..(UUUUL..0.............."ÄQ...L..0.............f...Nu28..TAa.
.000AC0 .b28..ÜAa..X`..T.........................a...a(Nu................
.000B00 ................a...Nu.9.....föNuH...08..j.a..Lp.............g.
.000B40 NqNq........1....H.I.@g..Ak.a4ara.S@fö`.D@aFada.S@föLß..Nu.9....
```

Comme nous pouvons le voir, il y a quelque chose d'intéressant qui commence à l'adresse $802.
C'est notre '*lookup-table*' que nous avons déjà vue.

Mais, si vous descendez de quelques lignes, vous pouvez voir que cette nouvelle table semble complète car elle ne finit pas par **L4-1** et **MUS4** mais par **L5-2** et **EDAT** qui correspondent probablement la fin des données !

Sauvons donc sur notre disquette cette nouvelle '*lookup-table*' qui semble complète pour une utilisation plus tard.

Insérer la disquette préalablement crée : **Turrican2_1**
Entrer dans l'**AR** et **Taper** : `SM FILETABLE, 802 9E2`

La taille de cette nouvelle '*table complet*' est de **!480 Bytes !** ($9E2-$802=$1E0= !480)

Après quelque test, je suis arrivé à la conclusion qu'il n'y a qu'un seul '*loader*', situé en **$6F8**

Nous connaissons donc tous les noms des fichiers, nous pouvons donc continuer notre petit crack en ripant ces fichiers sur un notre disk.
Nous allons utiliser le 1er 'loader'

Part 3 Ripage des fichiers #1

Rebooter votre machine toujours avec notre disquette **Turrican_1** dans le lecteur et **charger** notre propre *loader*
que nous avons préalablement sauvé au début de notre tutoriel en **$60000** !
Entrer dans l'AR et **Taper** :
`LM LOADER, 60000`

Maintenant que nous avons chargé le '*loader*' original à l'adresse **$60000**, à nouveau les choses deviennent intéressantes !

IMPORTANT : Les fichiers sur le disque sont stockés dans l'ordre qu'il apparaisse dans la '*lookup-table*' !
/! Il n'est pas possible de ripper dans le sens inverse, ce chargeur n'aime pas revenir sur le fichier précèdent ! ;)

Exemple, impossible de ripper les donnes **L4-1** et ensuite **RA-I** qui elles, sont au début de la 'lookup-table'.

Il est plus facile de noter chaque nom de fichier à ripper ainsi que sa longueur le tout dans le bon ordre.

En l'occurrence :

```
:050000 52 41 2D 49 00 02 01 E0 00 00 40 F0 00 00 01 09 RA-I......@.....
:050010 4D 41 49 4E 00 04 0D B0 00 00 E0 68 00 00 01 09 MAIN.......h....
:050020 49 4E 54 52 00 0C 19 98 00 00 32 88 00 00 01 01 INTR......2.....
:050030 4D 55 53 30 00 0E 17 00 00 01 A8 C4 00 00 00 01 MUS0........Ä....
:050040 49 44 41 54 00 1E 16 C4 00 00 55 B8 00 00 01 01 IDAT...Ä..U.....
:050050 49 50 30 30 00 22 02 3C 00 00 1C F8 00 00 01 04 IP00."..<........
:050060 49 50 30 31 00 23 04 A4 00 00 3C C8 00 00 01 04 IP01.#....<.....
:050070 49 50 30 32 00 25 0C 4C 00 00 18 C4 00 00 01 04 IP02.%.L...Ä....
:050080 49 50 30 33 00 26 0A 80 00 00 41 B0 00 00 01 04 IP03.&....A.....
:050090 49 50 30 34 00 28 17 10 00 00 47 04 00 00 01 04 IP04.(....G.....
:0500A0 49 50 30 35 00 2B 0E 64 00 00 0D 1C 00 00 01 04 IP05.+.d........
:0500B0 49 50 30 36 00 2C 00 F0 00 00 1A 2C 00 00 01 04 IP06.,.......,....
:0500C0 49 50 30 37 00 2D 00 8C 00 00 0F BC 00 00 01 04 IP07.-..........
:0500D0 4C 31 2D 31 00 2D 10 48 00 01 45 04 00 00 01 01 L1-1.-.H..E.....
:0500E0 4D 55 53 31 00 39 16 8C 00 00 A6 DC 00 00 01 01 MUS1.9.....ü....
:0500F0 4C 31 2D 32 00 40 03 78 00 00 44 38 00 00 01 01 L1-2.@.x..D8....
:050100 4C 32 2D 31 00 42 12 90 00 01 23 18 00 00 01 02 L2-1.B....#......
:050110 4D 55 53 32 00 4D 11 78 00 00 A7 28 00 00 01 02 MUS2.M.x..§(....
:050120 4C 32 2D 32 00 53 19 40 00 00 45 E0 00 00 01 02 L2-2.S.@..E.....
:050130 4C 33 2D 31 00 56 0F 70 00 00 E9 78 00 00 01 02 L3-1.V.p...x....
:050140 4D 55 53 33 00 5F 09 D8 00 00 B7 34 00 00 01 02 MUS3._.....4....
:050150 4C 33 2D 32 00 66 07 1C 00 00 48 50 00 00 01 02 L3-2.f....HP....
:050160 4C 33 2D 33 00 68 1A 4C 00 00 1F F8 00 00 01 02 L3-3.h.L.........
:050170 4C 34 2D 31 00 6A 05 24 00 00 FB 30 00 00 01 02 L4-1.j.$...0....
:050180 4D 55 53 34 00 73 11 44 00 00 EE FC 00 00 01 02 MUS4.s.D...ü....
:050190 4C 34 2D 32 00 7C 11 30 00 00 42 C4 00 00 01 02 L4-2.|.0..BÄ....
:0501A0 4C 35 2D 31 00 7F 04 44 00 01 55 64 00 00 01 02 L5-1.▒.D..Ud....
:0501B0 4D 55 53 35 00 8C 00 58 00 00 B8 2C 00 00 01 02 MUS5...X...,....
:0501C0 4C 35 2D 32 00 92 19 24 00 00 51 A0 00 00 01 02 L5-2....$..Q.....
:0501D0 45 44 41 54 00 96 00 84 00 00 F7 04 00 00 01 06 EDAT............
```

Ce qui nous donnes : **Figure 1 : Filetable**

DISK 1	RA-I	MAIN	INTR	MUS0	IDAT	IP00	IP01	IP02	IP03	IP04	IP05	IP06	IP07	L1-1	MUS1
	$0040F0	$00E068	$003288	$01A8C4	$0055B8	$001CF8	$003CC8	$0018C4	$0041B0	$004704	$000D1C	$001A2C	$000FBC	$014504	$00A6DC
	16624	132568	12936	108740	21944	7416	15560	6340	18816	18180	3356	6700	4028	83204	42716

DISK 1	L1-2	L2-1	MUS2	L2-2	L3-1	MUS3	L3-2	L3-3
	$004438	$012318	$00A728	$0045E0	$00E978	$00B734	$004850	$001FF8
	17464	74250	42792	17888	59768	46900	18512	8184

DISK 2	L4-1	MUS4	L4-2	L5-1	MUS5	L5-2	EDAT
	$00FB30	$00EEFC	$0042C4	$015564	$00B82C	$0051A0	$00F704
	64304	61180	17092	87396	47148	20896	63236

Recommençons donc par patcher notre '*loader*' une seconde fois, **taper** :
```
A 60042
^60042 move.w #$f00,$dff180
^6004a bra $60042
```

Cela introduira une boucle infinie avec un écran rouge après chaque chargement des pistes de données de nos fichiers.

En premier, **écrasons** le **moveq #1,d2** avant que le '*loader*' appel dans **moveq #0,d2**
car nous voulons obtenir les données dans l'état où elles sont stockées sur le disque et non décompressée ou autre chose.
```
A 6003c
^6003c moveq #0,d2
^6003e
```

On va sauver ce '*loader patché*' en tant que *LOADER_Patched_For_Rip* sur notre disque, **Taper :**
```
SM LOADER_Patched_For_Rip, 60000 60000+800
```

Du au faite que le fichier **RA-I** est le premier à être chargé, on ne va rien changer dans un premier temps.
/!\ Changer la disquette de sauvegarde Turrican2_1 dans le lecteur par la disquette original de Turrican2
Taper : `G 60000`

Attendons que l'écran rouge apparaisse. (La tête se déplace d'une piste), **Entrer dans votre AR** encore une fois.
/!\ Changer la disquette original de Turrican2 dans le lecteur par notre disquette de sauvegarde Turrican2_1
Sauvons le tout, **taper :** `SM RA-I,30000 30000+40F0`

/!\ Si vous avez un second lecteur, ce qui est vivement conseillé, voir Note2 quelques lignes plus bas.

340F0 car nous connaissons la longueur de chaque fichier maintenant (**offset +$8** dans la 'lookup-table' pour chaque entrée)
Pour le fichier RA-I : 40E8 + 8 = 40F0 et **30000 + 40F0 = 340F0**

Début boucle

Il est temps maintenant de passer au fichier suivant.
/!\ Changer la disquette de sauvegarde Turrican2_1 dans le lecteur par la disquette original de Turrican2

Changer le nom qui est déplacé dans **d0** à l'adresse **$60030** avant que le '*loader*' ne soit appelé, **taper** : `N $60032`

Rappeler vous, les **2 bytes** en **$60030** correspondent à l'instruction **move.l**
Ecraser le '*longword*' **RA-I** avec le prochain nom de fichier de notre '*lookup-table*' (voir figure 1 : Filetable)
Début de la boucle = fichier **MAIN**

```
n $60032
.060032 MAIN"(....t.a..$3ü...ß..'öIN"(....t.a...3ü▒..ß..3ü...ß..B..ß..B.
.060072 .ß..B..ß..B..ß..A.....C.....0("_".Q..üA....ÖC.....0(.." .Q..ü#ü..
```

Taper : `G 60000`
Attendons que l'écran rouge apparaisse. (La tête se déplace de 4 pistes) et **Entrer dans votre AR** encore une fois.
/!\ Changer la disquette original de Turrican2 dans le lecteur par notre disquette de sauvegarde Turrican2_1
Sauvons le tout, **taper :** `SM MAIN,30000 30000+E068`

Répéter ces actions (retour Début boucle) afin de sauver tous les fichiers **jusqu'au fichier** MUS4 (dernier fichier dans le *filetable* en mémoire)
Une fois le fichier **L3-3 rippé**, **Insérer** une nouvelle disquette vierge et formater la sous le nom de **Turrican2_2**
Sauver les 2 fichiers restant de notre table en mémoire (L4-1 et MUS4) sur cette disquette, le disque **Turrican2_2**

Fin boucle

Note1 :
Si votre ordinateur crash, par exemple sur le fichier **L1-1,** vous n'avez pas besoin de recommencer le rip depuis le début.
Charger juste le '*loader patché*' en *$60000* et continuer avec le dernier fichier que vous étiez en train de ripper, ainsi de suite.
comme indiqué plus haut.

Note2 :
Si vous réalisé ce tuto avec un second lecteur, à savoir **DF1**
Après avoir chargé notre loader patché en mémoire, il est plus facile de mettre la disquette originale dans **DF0**
et d'insérer notre disquette de sauvegarde, Turrican2_1 ou Turrican2_2 dans **DF1** afin de faciliter les taches de sauvegardes.
Ex : Pour le second fichier :

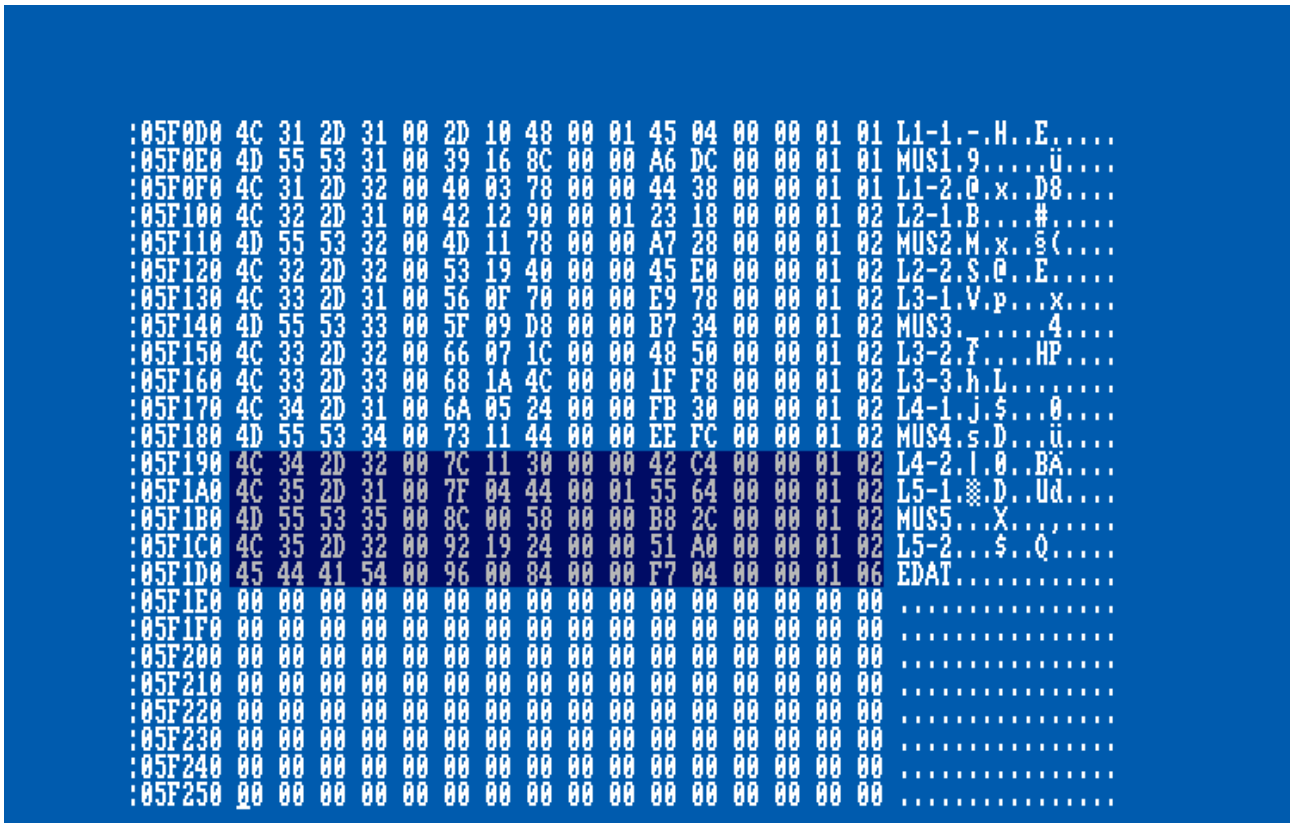
```
n 60032
.060032 MAIN"(....t.a..$3ü...ß..'öIN"(....t.a...3ü▒..ß..3ü...ß..B..ß..B.
.060072 .ß..B..ß..B..ß..A.....C.....0("_".Q..üA....ÖC.....0(.." .Q..ü#ü..
g 60000
No known virus in memory!
Ready.
sm 1:MAIN,30000 30000+E068
Disk ok
```

Part 3 Ripage des fichiers #2

Pour finir avec les derniers fichiers (**L4-2**, **L5-1**, **MUS5**, **L5-2** et **EADT**) non présent dans la *filetable* en mémoire.
Nous allons copier ces 5 fichiers depuis la 'table complète' en utilisant l'adresse **$60204**.
Nous pouvons **écraser** ces données sans problèmes car nous avons déjà rippé ces fichiers.

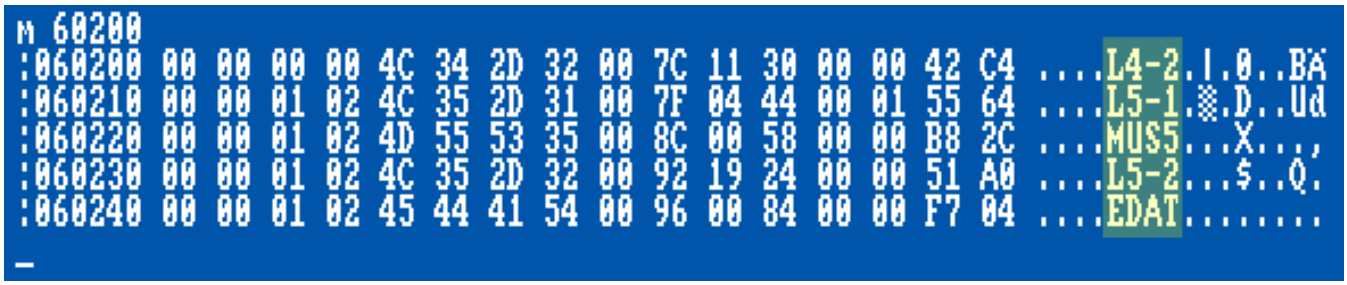
Insérer votre disque de sauvegarde Turrican_1 et **taper** : **LM FILETABLE, 5F000**

Descendre à la fin en utilisant la commande : **M 5F000**



Copions donc le block mémoire avec les 5 derniers fichiers dans notre '*lookup-table*' à l'adresse **$60204**
#TRANS, alias transfert. Permet le transfert d'une zone mémoire vers une autre.
Taper : **TRANS 5F190 5F1E0 60204**

Une petite vérification : **Taper** : **M 60200**



Maintenant, nous pouvons continuer le rip des fichier **L4-2** à **EADT** avec notre système de boucle.
/! N'oublier pas de bien insérer la disquette original dans DF0 avant de recommencer notre système de boucle
pour le rip de ces 5 derniers fichiers et d'attendre que l'Amiga est bien compris ce changement de disque.

On va finir la sauvegarde de ces 5 fichiers avec notre nouvelle bout de *filetable*, en utilisant donc notre système précédent de boucle.
Swapper avec les disquettes Turrican2_2 et le disquette original de **turrican 2**.

Félicitation !!!
Vous pouvez ranger votre disque original dans sa boite, nous n'en auront plus besoin.

Note :
Afin de rendre les choses plus faciles, ça serait une bonne idée de créer un répertoire sur votre disque dur (si vous en avez un) et de copier tous les fichiers dedans !

Part 4 Comprendre la routine Trackloading

Maintenant nous devons coder notre propre '*trackloader*',
Pour cela, il est nécessaire de comprendre comment le '*trackloader*' de Turrican II fonctionne…

Comme nous l'avons vues au début de ce tutoriel il y a 3 paramètres à définir avant d'appeler le '*loader*' : **d0, d1** et **d2**
Comme nous le savons le 3eme paramètre contenue dans **d2** sert à quelque chose comme une décompression.
Donc la routine à l'adresse **$60164** n'est pas uniquement un pure '*trackloader*'…

Il semble même être composé d'une autre routine qui est en mesure de décompresser les pistes de données.
Nous avons rippé les fichiers dans leur format compressé, donc nous avons encore besoin de cette routine de décompression.
Nous allons donc chercher ou le "vrais" '*trackloader*' commence, et quels types de paramètres a-t-il besoin pour fonctionner.

Désassemblons notre '*loader*' à partir de l'adresse **$60164**

Taper : **D 60164** et faites **défiler** quelques lignes ...

```
d 60164
~060164 TST.L   00060204
~06016A BNE     00060186
~06016C MOVEM.L D0-D2,-(A7)
~060170 MOVE.L  #190,D0
~060176 MOVEQ   #2,D1
~060178 MOVEQ   #0,D2
~06017A LEA     60204(PC),A0
~06017E BSR     00060394
~060182 MOVEM.L (A7)+,D0-D2
~060186 LEA     60204(PC),A1
~06018A MOVEQ   #18,D3
~06018C CMP.L   (A1),D0
~06018E BEQ     0006019C
~060190 LEA     10(A1),A1
~060194 DBF     D3,0006018C
~060198 MOVEQ   #FFFFFFFF,D0
~06019A RTS
'---------------------------------------
~06019C MOVE.L  8(A1),D0
~0601A0 MOVEA.L D1,A0
~0601A2 BTST    #0,D2
~0601A6 BEQ     000601B8
~0601A8 SUBQ.L  #8,A0
```

```
$60164  TST.L $60204          Esc ce que la lookup-table est déja chargé ?
$6016A  BNE $60186            Oui ... on continue.
$6016C  ...
$60170  ...
                              Pas neccessaire pour nous...
                              Un branchement vers une routine
$60176  ...                   Qui semble definir la 'lookup-table' à l'adresse $60204

$60178  ...
$6017A  ...
$6017E  ...
$60182  ...
$60186  LEA $60204(PC), A1    Stock l'adresse dans notre table de fichier en A1
$6018A  MOVEQ #$18, D3        Combien de fichiers sont stocké dans la 'lookup-table'
$6018C  CMP.L (A1), D0        Compare le nom de fichier donnée en D0 avec l"entrée dans lookup-table

$6018E  BEQ $6019C            Trouve le fichier... Continue le code en $6019C
                              (A1 pointe maintenant vers la correcte entrée dans table des fichiers)

$60190  LEA $10(A1), A1       Laisse la table des fichiers sur la prochaine entrée!
$60194  DBF D3, $6018C        Continue de chercher tant que d3 ne donne pas -1 !!!
$60198  MOVEQ #$-1, D0        Errorcode dans D0 -> File not found! ;)

$6019A  RTS                   Retourne de la ou l'on viens.

                              Rappeler vous l'Offset $8 ?
$6019C  MOVE.L 8(A1), D0      -> Deplacement de la longueur du fichier dans D0.

                              Rappeler vous D1 ?
$601A0  MOVE.L D1, A0         -> Mais notre ou charger´ à l'adresse dans A0.

                              Rappeler vous notre parametre de compression D2 ?
$601A2  BTST #$0, D2          -> Test Bit 0 ?

                              Zero... pas besoin de decompresser...
$601A6  BEQ.B $601B8          Continue en $601B8

                              Et bien... notre 'ou charger' est soustraie de 8! ;)
                              Sauve les 8 Bytes avant de charger la vrais adresse  de chargement
$601A8  SUBQ.L #$8, A0        ... Elle semble definit en 8 Bytes

$601B0  MOVE.L 4(a0), $60200  Pour le header "PP20" (4 Byte) inclus la longueur de decompression (4 Byte)

$601B8  MOVEM.L D1-D2/A0-A1, -(A7)  Sauve les choses necessaires dans la pile.

                              Maintenant ouvrer vos yeux... Les 4 instructions interressantes arrivent
                              (Rappeler vous le point A1 dans la 'look-table' !!!)
$601BC  MOVE.W 4(A1), D1      La valeur word située à l'offset+4 de la table de fichiers en D1
$601C0  MOVE.W 6(A1), D2      La valeur word située à l'offset+6 de la table de fichiers en D2

$601C4  EXT.L D2              Elargie D2 vers un longword !
$601C6  BSR.W $60394          Devinez quoi !!! C'est notre 'trackloader' !
$601CA  ...
```

Et bien... maintenant nous avons toutes les informations pour notre '*trackloader*'. Il est situé à l'adresse **$60394**
et il prend les paramètres suivants :

D0.L ➔ LONGUEUR DU FICHIER
D1.W ➔ Valeur WORD' stocké dans table des fichiers à **l'offset+$4**
D2.W ➔ Valeur WORD' stocké dans table des fichiers à **l'offset+$6** (élargi en longword)
A0.l ➔ ADRESSE MEMOIRE

Les registres dans **D1** et **D2** sont surement utilisés pour appeler le '*loader*' et lui indiquer où est situé le fichier sur le disk.

D1 = NUMERO DE LA PISTE
D2 = OFFSET DEPUIS LE DEBUT DE LA PISTE !!!

Le même code que ci-dessus est utilisé dans le fichier **MAIN**.
Bien sur le *trackloader* commence en mémoire à l'adresse **$9E2** et bien sûr, il prend les mêmes paramètres !

Pas besoin d'image pour celui-ci ! On ne va pas sans préoccuper plus que ça.

La prochaine partie sera de sauver les fichiers enregistrés sur un disque normal dos en utilisant le '*trackdisk-device*'
et de faire quelques modifications dans le fichier 'filetable'.

Comme vous l'avez peut-être deviné, nous ne pouvons placer les fichiers au même endroit que sur le disk original,
car nous n'aurons pas la même quantité d'octets de données écrites sur un piste dos que sur le disk de jeu Turrican.*
***Format de Track spécifique, voir début du tuto.**

Part 5 Décompression fichier MAIN

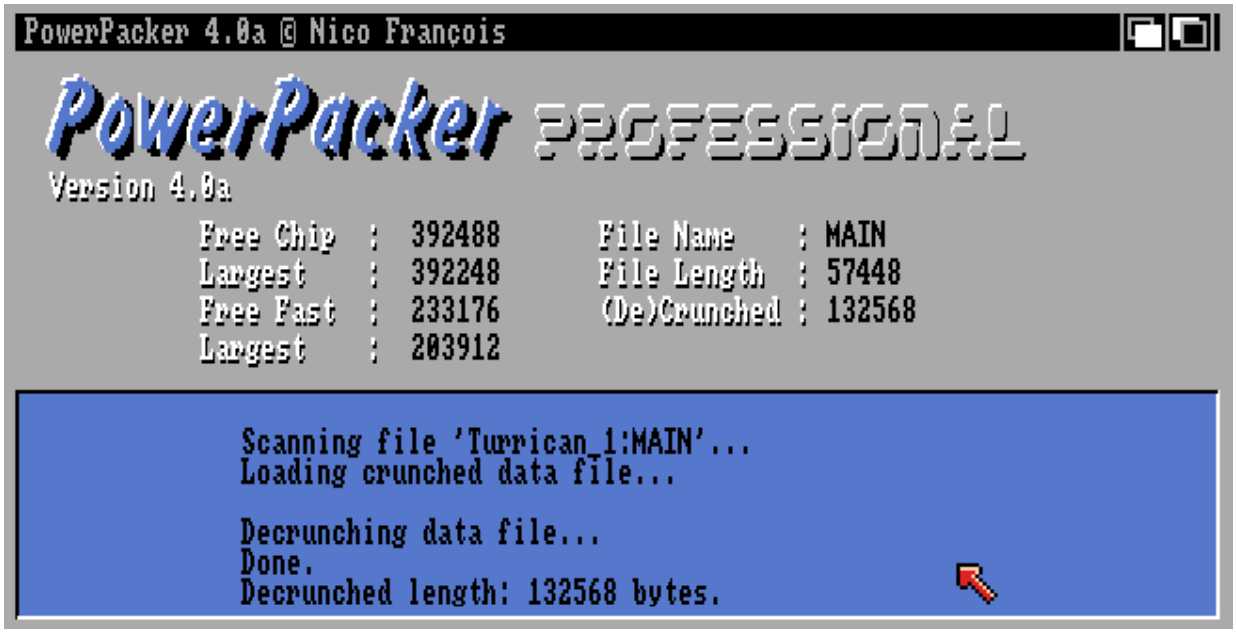
Avant de créer ces *disk-image* nous avons encore oublié de de faire une chose....
Nous savons qu'il y a un *'loader'* distinct dans le fichier **MAIN** que nous devons aussi modifier.

Comme nous avons sauvé le fichier **MAIN** dans un format *powerpacké* nous devons dans un premier temps le décompresser et l'enregistrer décompressé sur notre disk, cela facilitera notre '*patchage*' de celui-ci.

Commençons donc par décompresser le fichier en utilisant *Powerpacker* ou *XFD* et sauvons notre fichier décompressé (**!132568 bytes**) sur (un de vos) disk de sauvegarde(s).

Exécuter votre logiciel de compression/décompression, dans mon cas, PowerPacker 4.0a, **Charger** le fichier **MAIN**

L'entête *Powerpacker* est détectée et le fichier décompressé.
Sauver le tout.
Vous pouvez écraser l'ancien fichier **MAIN**, nous n'en auront plus besoin dans sa forme compressé !



Part 6 Création de notre nouvelle filetable

Si vous additionner toutes les longueurs de nos fichiers, vous réaliserez alors que cela ne rentre pas sur un disque standard.
Nous allons donc utiliser 2 disques.

Sur le **1er disk** nous allons stocker les fichiers de **A-I** à **L3-3** et sur **le second** le reste des fichiers.
Vous aurez alors à swapper de disk avant le niveau 4 !

Pour dire au *'loader'* à quel disque le fichier actuel appartiens, nous allons utiliser comme astuce le numéro des '*tracks'*.
de 0-159 indiquera que les données sont sur le **disque 1**
et de 160-xxx indiquera que les données sont sur le **disque 2**
Nous allons créer un bout de code qui va remplir automatiquement notre nouvelle filetable avec les valeurs correctes.

Exécuter *ASM-One*, **réserver 100 Ko de ram**, (exemple CHIP/100), cela suffira pour commencer.

Passer en mode Edition en utilisant la touche **ESC**
Taper le code suivant et **sauver** la source (au cas où).

patchfiletable.s ➔

```
        LEA     FILETABLE,A0
        MOVE.L  #132568,16+8(A0)
        MOVE.L  #30-1,D0
        MOVE.L  #$4*$1600,D1
LOOP:   MOVE.L  D1,D2
        DIVS.W  #$1600,D2
        MOVE.W  D2,$4(A0)
        SWAP    D2
        MOVE.W  D2,$6(A0)
        ADD.L   $8(A0),D1
        ADD.L   #16,A0
        CMP.L   #"L4-1",(A0)
        BNE.B   NOT
        MOVE.L  #160*$1600,D1
NOT:    DBF     D0,LOOP
        RTS

FILETABLE:

        INCBIN  "Turrican2_1:FILETABLE"
```

Quelques explications s'imposent :

La seconde instruction est vraiment importante.
Du au faite que nous avons décompressé le fichier *main* nous devons écrire une nouvelle longueur de taille de fichier dans la *filetable*.
Taille qui va être de **!132568 ($205d8) bytes** (***voir image PowerPacker page précédente***)

Chaque entrée est de **$16 bytes**, donc l'offset de la longueur de fichier **MAIN**, qui est le second fichier de la table, est à l'offset !16 + !8 !!!

Dans le code source un compteur de '*byte*' est utilisé via **d1** pour stocker la position actuelle du fichier du jeux sur le disque,
le tout stocké en valeur de 'byte'.

Nous commençons donc avec **RA-I** sur la piste 4 qui semble être à la position-disk **4*$1600**
Rappeler vous, une piste a une taille de **$1600 bytes**.

Le compteur de '*byte*' est divisé par **$1600** pour avoir le numéro de la piste courant du fichier et le stock dans la table des fichiers.

Puis, nous échangerons la valeur pour prendre le reste de notre division qui est en fait l'*offset-byte* du fichier sur la piste.

Maintenant la longueur de l'actuel fichier (**$40F0** dans la premier boucle) est ajoutée à la valeur de notre compteur de *byte* dans **d1**,
le changeant alors en **4*$1600+$40F0**

Finalement nous pointons **a0** sur la prochaine entrée dans notre *filetable* et recommençons la même manipulation jusqu'à la fin de la table des fichiers.

SI le fichier **L4-1** est atteint, alors le compteur de *byte* est positionné à **160*$1600** car à partir de là, les fichiers appartiennent au disk 2 et commence encore en piste 0 !

Notre propre loader va vérifier la position de la piste et si elle est plus large que **!159,** effectuera un flash d'écran pour indiquer à 'utilisateur que le disque doit être changé.

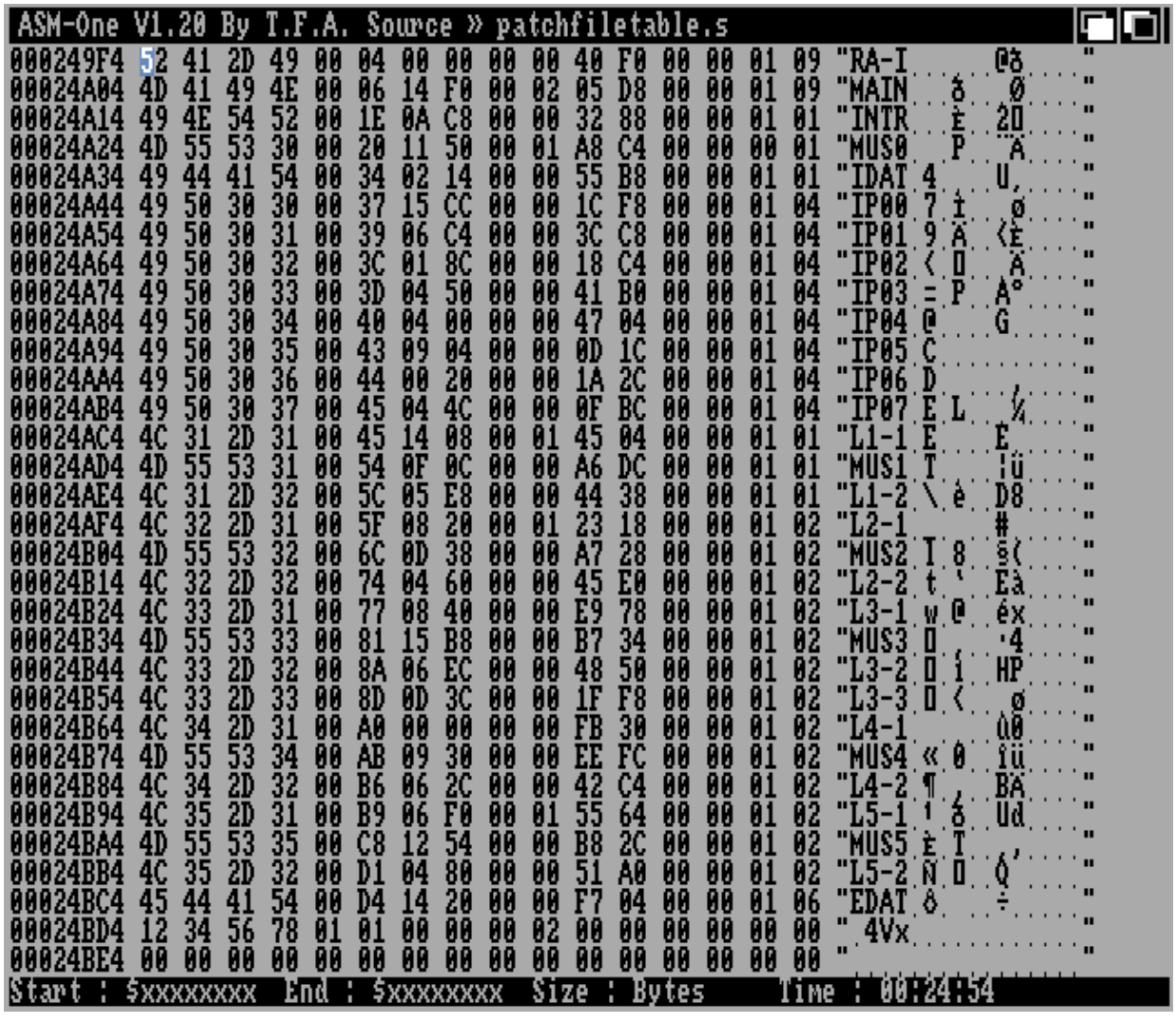
Évidement cela fonctionne dans l'autre sens également, si le disque 2 est placé dans le lecteur alors que le disque 1 est nécessaire !

Il est temps de compile et d'exécuter le tout.
Passer en ligne de commande en utilisant la touche **ESC** et **assembler** le tout avec la commande **A**
Exécuter le tout avec la commande **J** (Modifier avant ci besoin les chemin pour accéder au fichier **FILETABLE** si nécessaire**)**

Si vous n'avez pas d'erreur vous devriez avoir quelque chose comme ça :

On va maintenant vérifier qu'on a bien patché notre filetable
En mode ligne de commande, **taper : H FILETABLE**

Vous deviez voir ceci :



Sauvons tout sous le nom de **FILETABLE_PATCHED**

Retourner en ligne de commande avec la touche **ESC**
Taper : WB, puis comme début **'BEG'**, l'étiquette **FILETABLE** et on sait que la taille de *Filetable* est de !480 *(voir début tuto)*
donc, **END = FILETABLE+$1E0**

Part 7 Création de notre propre trackloader

PS : Dans ce tutoriel, je ne vous explique pas comment fonctionne le *hardware-trackloading* ou comment les données sont stockées, encodé dans le format **MFM**.
Car c'est un tutoriel Comment cracker **Turrican 2** et non pas un tutoriel Comment coder un hardware-trackloader.

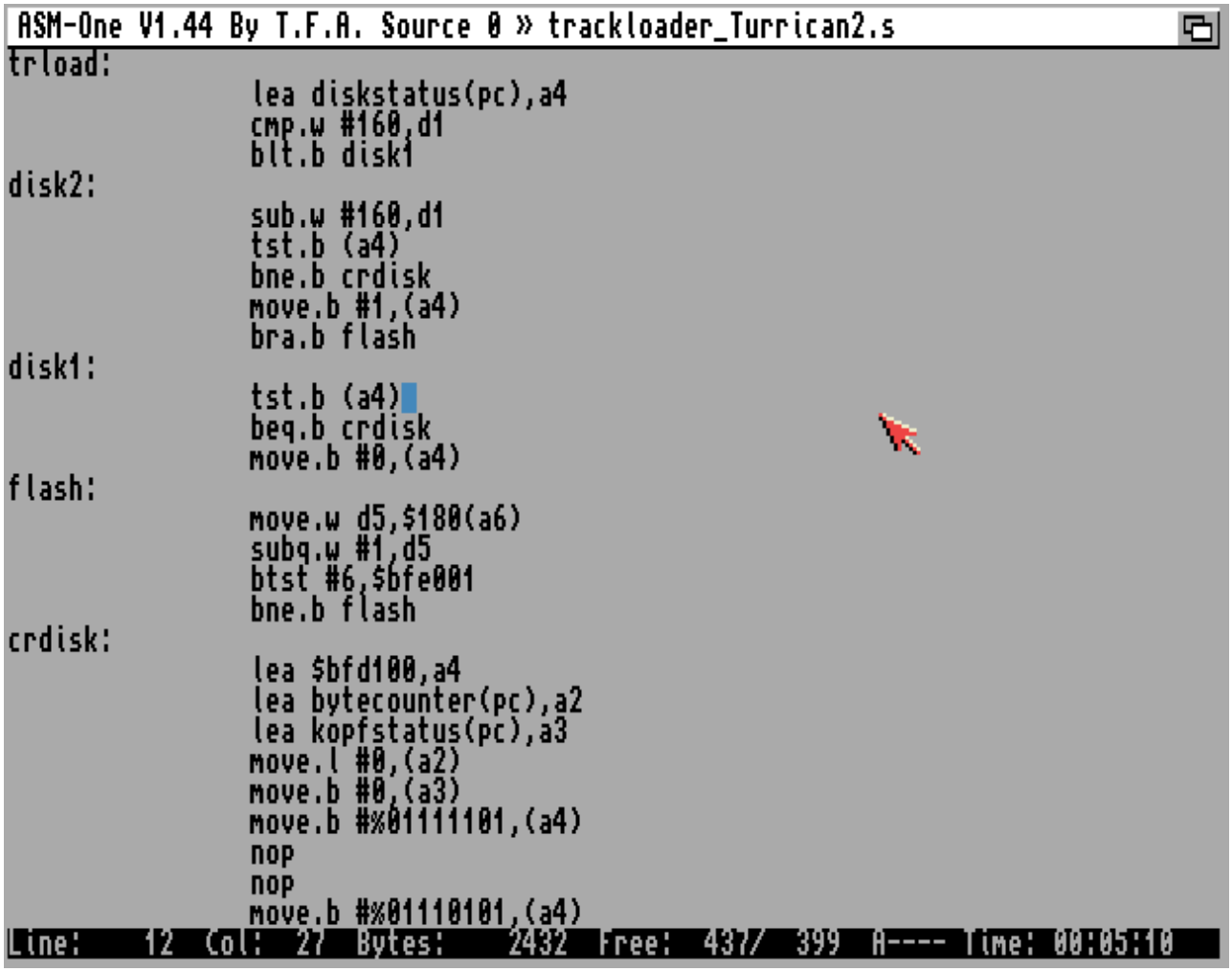
Notre propre *loader* devra travailler avec les mêmes paramètres donnés dans les mêmes registres que le loader original, donc notre routine de *trackloading* fonctionnera aussi avec les paramètres suivants :

```
D0.l = Bytes à lire
D1.w = Tracknumber (où se situe le début des données actuel du fichier courant!)
D2.l = Byteoffset (l'offset-byte, ou notre fichier commence sur la piste)
A0.l = Memory Loadadress (adresse mémoire de chargement)
```

Vous ne pourrez comprendre les prochaines lignes qui vont suivre que si vous avez déjà programmé un *H/W trackloader*

Ce loader est un *trackgrinding*, car il est toujours un pas derrière la piste 0, puis il avance jusqu'à la piste où le fichier courant est !

Passer en mode édition avec la touche **ESC** si vous ne l'êtes pas déjà et **taper ZS** puis entrer le nouveau code suivant :



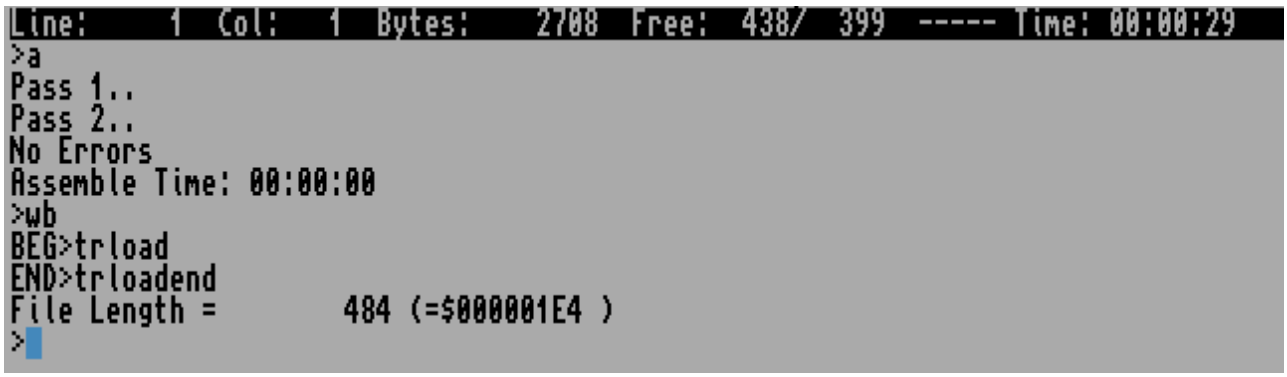
trackloader.s →

```
; ================================
; Trackloader for Turrican 2.
; D0.l = Bytes to read
; D1.w = Tracknumber
; D2.l = Trackoffset
; A0.l = Buffer to read
; A6.l = $dff000
;================================
trload:
                lea diskstatus(pc),a4
                cmp.w #160,d1
                blt.b disk1
disk2:
                sub.w #160,d1
                tst.b (a4)
                bne.b crdisk
                move.b #1,(a4)
                bra.b flash
disk1:
                tst.b (a4)
                beq.b crdisk
                move.b #0,(a4)
flash:
                move.w d5,$180(a6)
                subq.w #1,d5
                btst #6,$bfe001
                bne.b flash
crdisk:
                lea $bfd100,a4
                lea bytecounter(pc),a2
                lea kopfstatus(pc),a3
                move.l #0,(a2)
                move.b #0,(a3)
                move.b #%01111101,(a4)
                nop
                nop
                move.b #%01110101,(a4)
                bsr.w diskok
                add.l d2,d0
mvzero:
                btst #4,$bfe001
                beq.b trzero
                bset #1,(a4)
                bsr.w movehd
                bsr.w delay
                bra.w mvzero
trzero:
                bsr.w diskok
                moveq #0,d3
                move.b d1,d3
                divs.w #2,d3
                move.l d3,d1
                swap d1
                cmp.w #0,d1
                beq.b headd
                bclr #2,(a4)
                bsr.w delay
                move.b #1,(a3)
                bra.b mvzyl
headd:
                bset #2,(a4)
                bsr.w delay
                move.b #0,(a3)
mvzyl:
                cmp.b #0,d3
                beq.b zylok
moveit:
                bsr.w delay
                bclr #1,(a4)
                bsr.w movehd
                dbf d3,mvzyl
zylok:
                bsr.w diskok
                bsr.w readtr
                bra.w decode
trdone:
                cmp.b #1,(a3)
                bne.b headup
                bset #2,(a4)
                move.b #0,(a3)
                moveq #1,d3
                bra.b moveit
                bclr #1,(a4)
```

```
                bsr.w movehd
                bsr.w delay
                bra.b zylok
headup:
                bclr #2,(a4)
                move.b #1,(a3)
                bra.w zylok
ready:
                bsr.w delay
                move.b #%11111101,(a4)
                nop
                nop
                move.b #%11100111,(a4)
                rts
diskok:
                btst #5,$bfe001
                bne.b diskok
                rts
readtr:
                move.w #$2,$9c(a6)
                move.w #$8210,$96(a6)
                clr.w $24(a6)
                move.l #$74000,$20(a6)
                move.w #$4489,$7e(a6)
                move.w #%0111111100000000,$9e(a6)
                move.w #%1001010100000000,$9e(a6)
                move.w #$9900,$24(a6)
                move.w #$9900,$24(a6)
dwait:
                btst #1,$1f(a6)
                beq.b dwait
                clr.w $24(a6)
                rts
movehd:
                bclr #0,(a4)
                nop
                nop
                nop
                bset #0,(a4)
                rts
delay:
                move.w #$9999,d4
wait:
                dbra d4,wait
                rts
decode:
                move.l #$55555555,d4
                moveq #0,d5
dec:
                lea $74000,a1
ssync:
                cmp.w #$4489,(a1)+
                bne.b ssync
                cmp.w #$4489,(a1)
                beq.b ssync
                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 bfound
                add.l #1086,a1
                bra.b ssync
bfound:
                add.l #56,a1
                move.l #(512/4)-1,d6
decbl:
                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 (a2),d2
                bgt.b ofschk
                move.l d3,(a0)+
ofschk:
                addq.l #4,(a2)
                cmp.l (a2),d0
                ble.w ready
                dbra d6,decbl
```

```
                cmp.b #10,d5
                beq.w trdone
                addq.b #1,d5
                bra.w dec
bytecounter:    dc.l 0
kopfstatus:     dc.b 0
diskstatus:     dc.b 0
trloadend:
```

Sauver les sources au cas où puis **Passer** en ligne de commande en utilisant la touche **ESC** et assembler le tout avec la commande **A**
Si vous n'avez aucun message d'erreur, sauver le tout sous le nom de fichier trackloader
Taper : WB, puis comme début **'BEG'**, l'étiquette **trload et pour END, trloadend**



Vous devriez avoir une taille de fichier de **!484** soit **($1E4) bytes** au total.

Part 8 Patch des fichiers LOADER et MAIN

Maintenant, il est temps de patcher nos deux fichiers **loader** et **MAIN** avec la nouvelle table des fichiers
et la nouvelle routine de *trackloading*...

Nous allons encore utiliser l'**AR** pour notre action de patchage
Rebooter votre amiga et **entrer dans l'AR MAINTENANT !**
Insérer notre disquette de sauvegarde Turrican2_1 dans le lecteur.

Nous allons commencer par le *LOADER*

Taper : LM LOADER, $50000

Nous savons que la *filetable* est stocké à l'adresse **$60204** dans la mémoire, donc l'offset du début de notre *routine loader* sera **$204**.

Chargeons notre table de fichier patché vers l'adresse **$45000** et copions la dans notre code de chargement !
Taper : LM FILETABLE_PATCHED, $45000

Transférons maintenant nos premier **400 bytes** de celui-ci vers la position correcte.
(Nous avons uniquement pris les **400 premiers bytes** car comme nous savons que la *lookuptable* dans ce loader n'est pas complète et copier la totalité de nos **480 bytes** pourrait écraser un peut notre trackloader qui est situé à la fin de la table des fichiers !)

Taper : TRANS $45000 $45000+!400 $50204

Nous devons maintenant faire la même chose avec la routine de *trackloading*

Comme vue dans la partie précédente, le *trackloader* commence en **$60394,** ce qui veux dire que l'*offset* de notre loader commence en **$394**.

Chargeons donc le *loader*, nous allons utiliser l'adresse **$46000** pour celui-ci.

Taper : LM trackloader, $46000

Copions le dans le *loader* en utilisant la commande : **TRANS $46000 $46000+!484 $50394**

Avant d'enregistrer notre *loader_patché*, il est nécessaire de faire encore une modification.

Il faut changer le paramètre dans **d2** avant que le *loader* essaye de récupérer le fichier **MAIN**
Comme nous avons sauvé ce fichier dans un format non compressé sur notre disk il est nécessaire qu'il n'essaye pas de de le décompresser.

Si nous laissions dans le code le **moveq #1,d2**, le *loader* voudra essayer de décompresser le fichier ce qui causera un crash du jeu !
Il est donc nécessaire de changer cette instruction maintenant...

Taper : A 50054 et **insérer** un **moveq #0,d2** suivit par la touche entrée !

Maintenant nous sommes prêts à écrire notre *loader patché* sur notre disk.

Taper : SM LOADER_PATCHED, $50000 $50000+!2048

Maintenant le fichier MAIN

Taper : LM MAIN, $50000

Nous savons que la *lookuptable* est stocké à l'adresse mémoire **$802**, donc l'offset du début du fichier main est **$742 ($802-$C0)**.

Copions donc la *lookuptable complète*, **taper** : **TRANS $45000 $4500+!480 $50742**

Maintenant, effectuons encore la même chose pour le *trackloader* qui est en mémoire à l'adresse **$9E2**,
l'offset de l'adresse de départ est donc : **$922 ($9E2-$C0)**

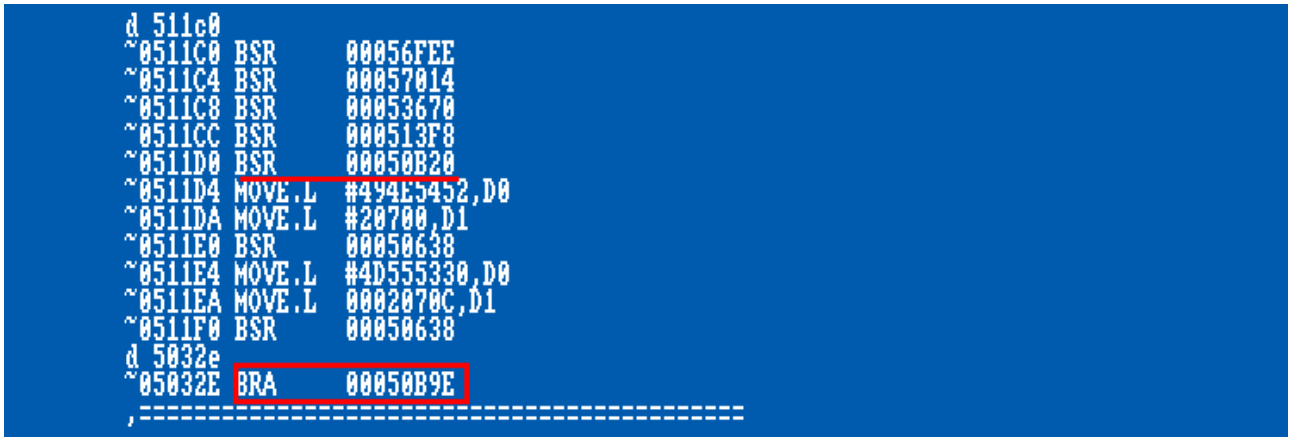
Taper : TRANS $46000 $46000+!484 $50922

Nous avons maintenant finalement notre fichier MAIN patché.

Mais avant de le sauver nous devons effectuer encore 2 patches, que je vais d'abord vous expliquer ! ;)

Le jeu charge et sauve les highscores ce qui provoquera un crach du jeu s'il ne trouve pas l'original du *trackformat* sur notre crack.
Il y a uniquement 2 branchements que nous devons enlever pour que cela ne se produise pas.
Il n'y a pas d'explication détaillé de 'comment' j'ai trouvé ces branchements car il n'existe pas de technique spécial pour les trouver.

Quelque fois, vous pouvez passer des heures à chercher des satanés sauts !
Vous trouverez ces deux bâtards ici :



Remplaçons donc le premier par un **nop** et le second par un **rts** comme ceci.

```
A $511D0
^511d0          nop
^511d2          nop
^511d4          <ESC>
```

... et finalement

```
A $5032E
^5032e          rts
^50330          nop
^50332          <ESC>
```

Sauvons donc le fichier final sur notre disk en utilisant la commande : **SM MAIN_PATCHED, $50000 $50000+!132568**

Part 9 Créer Les images Disque

Rebooter votre Amiga et relancer *ASM-One*, **réserver au moins 840 Ko de ram**, (exemple CHIP/840).

```
ALLOCATE Fast/Chip/Publ/Abs>chip
WORKSPACE (Max.861) KB>840
>
```

Insérer ensuite votre disque de sauvegarde Turrican2_1 dans le lecteur.

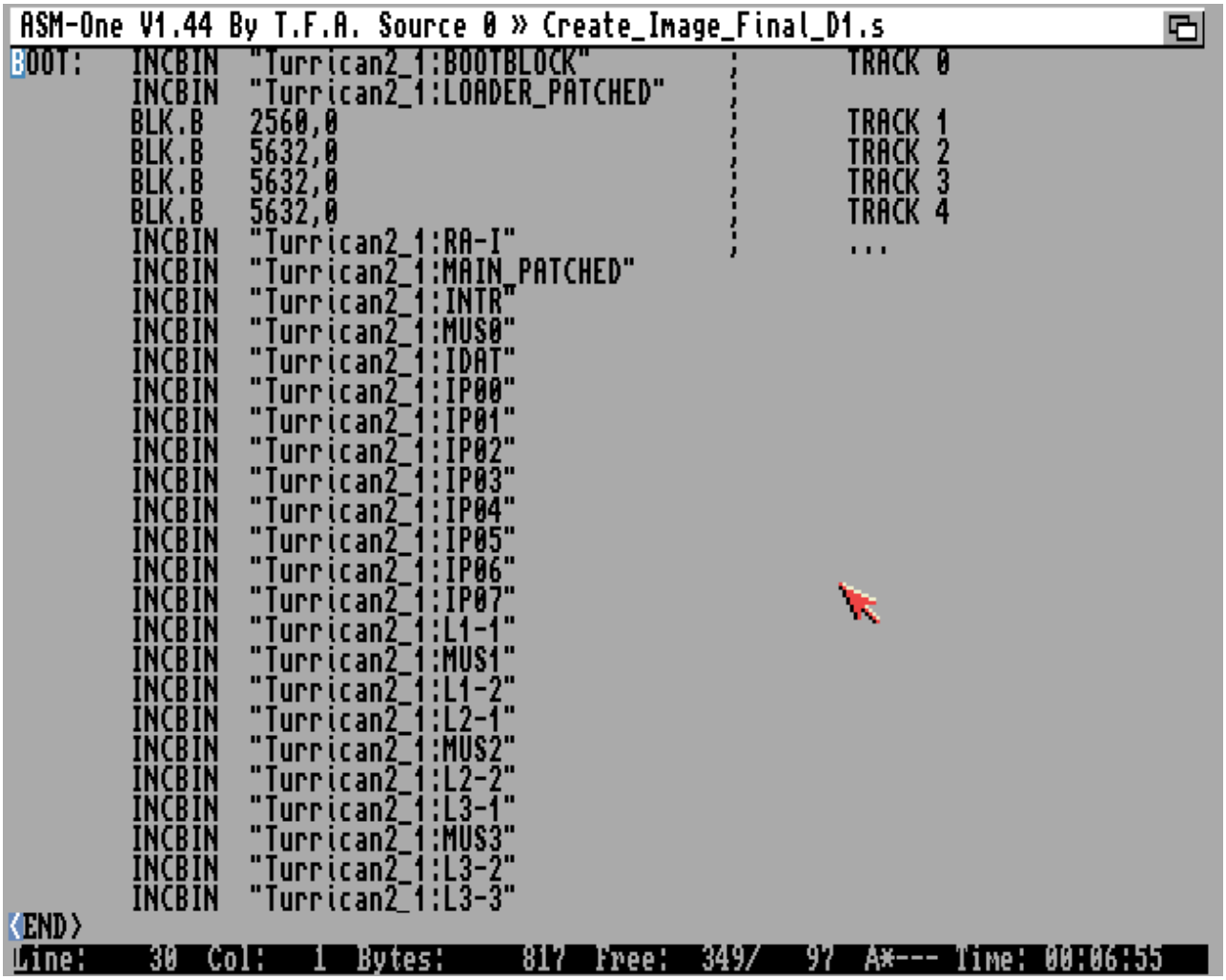
PS : Si vous n'avez pas au moins 1Mega de RAM Chip, vous allez devoir écrire les images-disk en plusieurs étapes…
Si vous n'avez pas de disque dur et que vous avez les fichiers stockés sur 2 disquettes, vous allez devoir jongler avec vos disquettes.
Vous pouvez éviter tous ces soucis avec un peu de RAM et en utilisant un second lecteur de disquette.

Passer en mode Edition en utilisant la touche **ESC**
Taper le code suivant et **sauver** la source (au cas où).
Bien sur, adapter les chemins si nécessaire.

Create_Image_Final_D1.s ➔

```
;Final Disk1 Piste/Secteur, 80 pistes [0-79]

BOOT:  INCBIN  "Turrican2 1:BOOTBLOCK"       ;P00.0/S00 -> P00.0/S01   1024 oct
       INCBIN  "Turrican2 1:LOADER PATCHED"  ;P00.0/S02 -> P00.0/S05   2048 oct
       BLK.B   2560,0                        ;P00.0/S05 -> P01.0/S00   2560 oct, Fin de piste
       BLK.B   5632,0                        ;P01.0/S00 -> P02.0/S00   5632 oct de vide, 1 piste
       BLK.B   5632,0                        ;P02.0/S00 -> P03.0/S00   5632 oct de vide, 1 piste
       BLK.B   5632,0                        ;P03.0/S00 -> P04.0/S00   5632 oct de vide, 1 piste
       INCBIN  "Turrican2 1:RA-I"            ;P04.0/S00 -> P06.0/S10  16624 oct
       INCBIN  "Turrican2 1:MAIN PATCHED"    ;P06.0/S10 -> P30.0/S05 132568 oct
       INCBIN  "Turrican2 1:INTR"            ;P30.0/S05 -> P32.0/S08  12936 oct
       INCBIN  "Turrican2 1:MUS0"            ;P32.0/S08 -> P52.0/S01 108740 oct
       INCBIN  "Turrican2 1:IDAT"            ;P32.1/S01 -> P55.0/S10  21944 oct
       INCBIN  "Turrican2 1:IP00"            ;P55.0/S10 -> P57.0/S03   7416 oct
       INCBIN  "Turrican2 1:IP01"            ;P55.0/S03 -> P60.0/S00  15560 oct
       INCBIN  "Turrican2 1:IP02"            ;P60.0/S00 -> P61.0/S02   6340 oct
       INCBIN  "Turrican2 1:IP03"            ;P61.0/S02 -> P64.0/S01  16816 oct
       INCBIN  "Turrican2 1:IP04"            ;P64.0/S02 -> P67.0/S04  18180 oct
       INCBIN  "Turrican2 1:IP05"            ;P67.0/S04 -> P68.0/S00   3356 oct
       INCBIN  "Turrican2 1:IP06"            ;P68.0/S00 -> P69.0/S02   6700 oct
       INCBIN  "Turrican2 1:IP07"            ;P69.0/S02 -> P69.0/S10   4028 oct
       INCBIN  "Turrican2 1:L1-1"            ;P69.0/S10 -> P04.1/S07  83204 oct
       INCBIN  "Turrican2 1:MUS1"            ;P04.1/S07 -> P12.1/S02  72716 oct
       INCBIN  "Turrican2 1:L1-2"            ;P12.1/S02 -> P15.1/S04  17464 oct
       INCBIN  "Turrican2 1:L2-1"            ;P15.1/S04 -> P28.1/S06  74520 oct
       INCBIN  "Turrican2 1:MUS2"            ;P28.1/S06 -> P36.1/S02  42792 oct
       INCBIN  "Turrican2 1:L2-2"            ;P36.1/S02 -> P39.1/S04  17888 oct
       INCBIN  "Turrican2 1:L3-1"            ;P39.1/S04 -> P49.1/S10  59768 oct
       INCBIN  "Turrican2 1:MUS3"            ;P49.1/S10 -> P58.1/S03  46900 oct
       INCBIN  "Turrican2 1:L3-2"            ;P58.1/S03 -> P61.1/S06  18512 oct
       INCBIN  "Turrican2 1:L3-3"            ;P61.1/S06 -> P63.1/S00   8184 oct
```

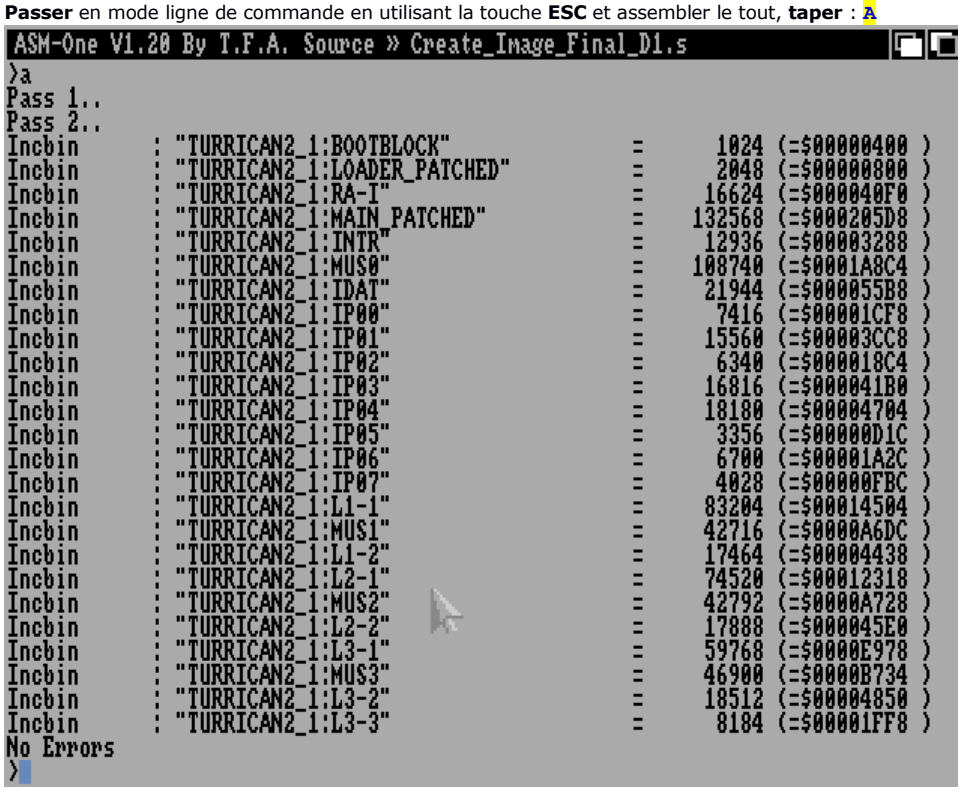


Quelques explications s'imposent :

En premier, nous intégrons le 'bootblock' (**!1024 bytes**) suivit par le premier 'loader' (**!2048 bytes**).
Nous remplissons ensuite les **!2560 bytes** suivant avec des zero (Une piste a une taille de **!5632 -$1600- bytes**)

Nous insérons ensuite 3 pistes vides pour une jolie cracktro
Suivit par les fichiers du jeu qui apparaissent dans la 'filetable'.

Passer en mode ligne de commande en utilisant la touche **ESC** et assembler le tout, **taper** : **A**



Puis, **insérer** une disquette vierge dans ***DF0*** et **Taper** : WT, avec comme début '**PTR'**, l'étiquette **BOOT** et *DISK PTR*=**0** et *LENGTH*=**144**

```
No Errors
>wt
RAM PTR>BOOT
DISK PTR>0
LENGTH>144
>
```

➔72 pistes sont écrites (de 0 à 71)

Vous Venez de créer la 1ère disquette de notre crack, vous pouvez l'enlever du lecteur.

Taper ZS en ligne de commande puis **passer** en mode Edition en utilisant la touche **ESC**
Taper le code suivant et **sauver** la source (au cas où).
Bien sur, adapter les chemins si nécessaire.

Create_Image_Final_D2.s ➔

```
;Final Disk2 Piste/Secteur, 80 pistes [0-79]

L4 1 :INCBIN  "Turrican2 2:L4-1"          ;P00.0/S00 -> P05.1/S04 64304 oct
      INCBIN  "Turrican2 2:MUS4"          ;P05.1/S04 -> P11.0/S03 61180 oct
      INCBIN  "Turrican2 2:L4-2"          ;P11.0/S03 -> P12.1/S03 17092 oct
      INCBIN  "Turrican2 2:L5-1"          ;P12.1/S03 -> P20.0/S09 87396 oct
      INCBIN  "Turrican2 2:MUS5"          ;P20.0/S09 -> P24.1/S02 47148 oct
      INCBIN  "Turrican2 2:L5-2"          ;P24.1/S02 -> P26.0/S10 20896 oct
      INCBIN  "Turrican2 2:EDAT"          ;P26.0/S10 -> P32.0/S01 63236 oct
```

```
ASM-One V1.20 By T.F.A. Source » Create_Image_Final_D2.s
L4_1:   INCBIN  "Turrican2_2:L4-1"              ;       Begin Of New Disk 2!
        INCBIN  "Turrican2_2:MUS4"
        INCBIN  "Turrican2_2:L4-2"
        INCBIN  "Turrican2_2:L5-1"
        INCBIN  "Turrican2_2:MUS5"
        INCBIN  "Turrican2_2:L5-2"
        INCBIN  "Turrican2_2:EDAT"
(END)
```

Réinsérer notre disquette de sauvegarde Turrican2_1 dans le lecteur **DF0** et si vous avez un second lecteur de disquette, **insérer** notre disque de sauvegarde Turrican2_2 dedans, sinon, jongler avec ces deux disques lors de la commande d'assemblage à venir.
Retourner en mode ligne de commande en utilisant la touche **ESC** et **assembler** le tout, **taper** : A

```
Line:     1  Col:    1  Bytes:     218  Free:  305/   59  -----  Time: 06:53:03
>a
Pass 1..
Pass 2..
Incbin      : "TURRICAN2_2:L4-1"                =     64304 (=$0000FB30 )
Incbin      : "TURRICAN2_2:MUS4"                =     61180 (=$0000EEFC )
Incbin      : "TURRICAN2_2:L4-2"                =     17092 (=$000042C4 )
Incbin      : "TURRICAN2_2:L5-1"                =     87396 (=$00015564 )
Incbin      : "TURRICAN2_2:MUS5"                =     47148 (=$0000B82C )
Incbin      : "TURRICAN2_2:L5-2"                =     20896 (=$000051A0 )
Incbin      : "TURRICAN2_2:EDAT"                =     63236 (=$0000F704 )
No Errors
>
```

Puis, **Retirer** toutes les disquettes de l'Amiga et **insérer** une nouvelle disquette vierge dans ***DF0***
Taper : WT, avec comme début '**PTR'**, l'étiquette **L4_1** puis, *DISK PTR*=**0** et *LENGTH*=**67**

```
>wt
RAM PTR>L4_1
DISK PTR>0
LENGTH>67
>
```

➔34 pistes sont écrites (de 0 à 33)

Voilà, vous venez de créer la seconde disquette de notre crack, vous pouvez l'enlever du lecteur.

Au moins quelques greetings au meilleur créateur de trainer sur terre : SIRIaX !
et au dernier des Demogroup sur Amiga, *cREATIVE MINDS* ! :-)