Tutorial	AmigaCracking : Shadow Of The Beast I I
Protection	MFM + CRC + Others
Original Author	Gi@nts
original Source	Giants
Version	Started in 28 / 10 / 2019 Gi@nts Done French Version @ 10 / 08 / 2020 (yes, one year) And English version (22 / 09 / 2023) : DONE !
Check / Correction	v1.2

* CRACK TUTORIAL - SHADOW OF THE BEAST II *

Table of Contents

Necessary materials

1) AMIGA 500 or WINUAE emulator.
2) Lot of floppy disk.
3) External Floppy reader is strongly recommended.
4) ACTION REPLAY Cartridge (or Image ROM) depending on the used configuration.
5) Original disks game or CAPS files (SPS 1359)
6) Source Code of trackloader : AlphaOne 2004 (v. 404 Bytes)
7) Source Code of trackloader : AlphaOne 2005 pro v2 (v. 460 Bytes)
8) Archive with source code of cruncher/uncruncher RNC propack (Aminet : util/pack/RNC_ProPack.lha)
9) A loooooot of time.

General Information

Important : This is a traduction* of the original French Tuto done by.. me ☺
*and some 'little' corrections.

We will try through this tutorial to be as complete as possible. (too much ?)
This tutorial is based on my analysis work done directly on the original.
In my opinion, it's more than complete.
It is strongly advised to follow the steps in chronological order otherwise you may be lost in its apprehension.
Do not hesitate also to return to chapters for a better understanding.

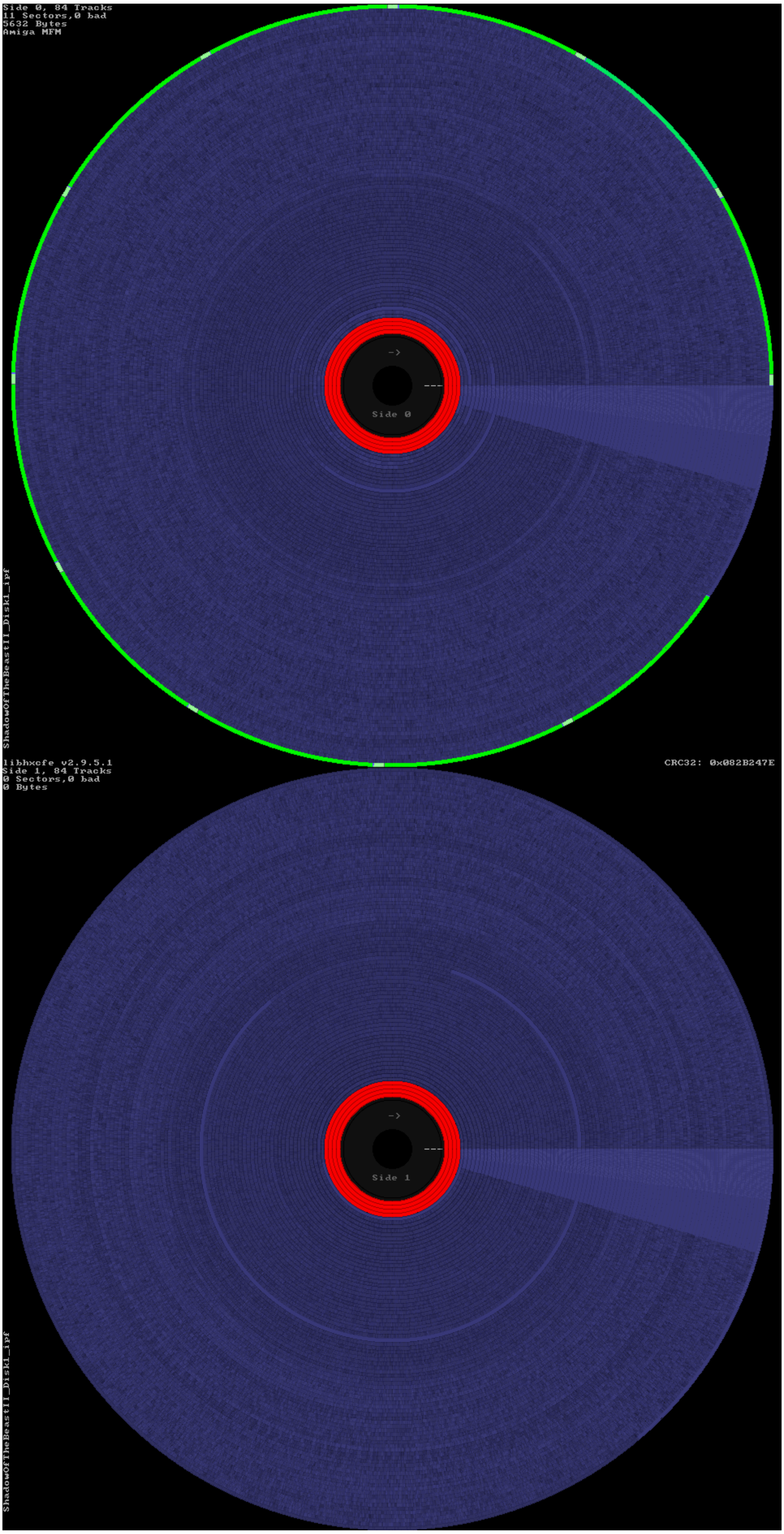
Note : This is a crack tutorial and not 'how to learn assembler on Amiga' document.
The goal is to understand the instructions without detailing them all, to be able to understand the overall functioning of the code.
in order to be able to isolate the parts related to the Hack (Trackloader, CheckSum routine, CopyLock...).
Considering the size of the document, there are surely some errors, don't hesitate to give me a feedback if necessary.
Beware, this tutorial is heavy ☺

By the way, a big thank you to Amigars from the AmigaImpact.org Forum for proofreading and correcting the Tuto in the French Version.

'Bon Tuto.'

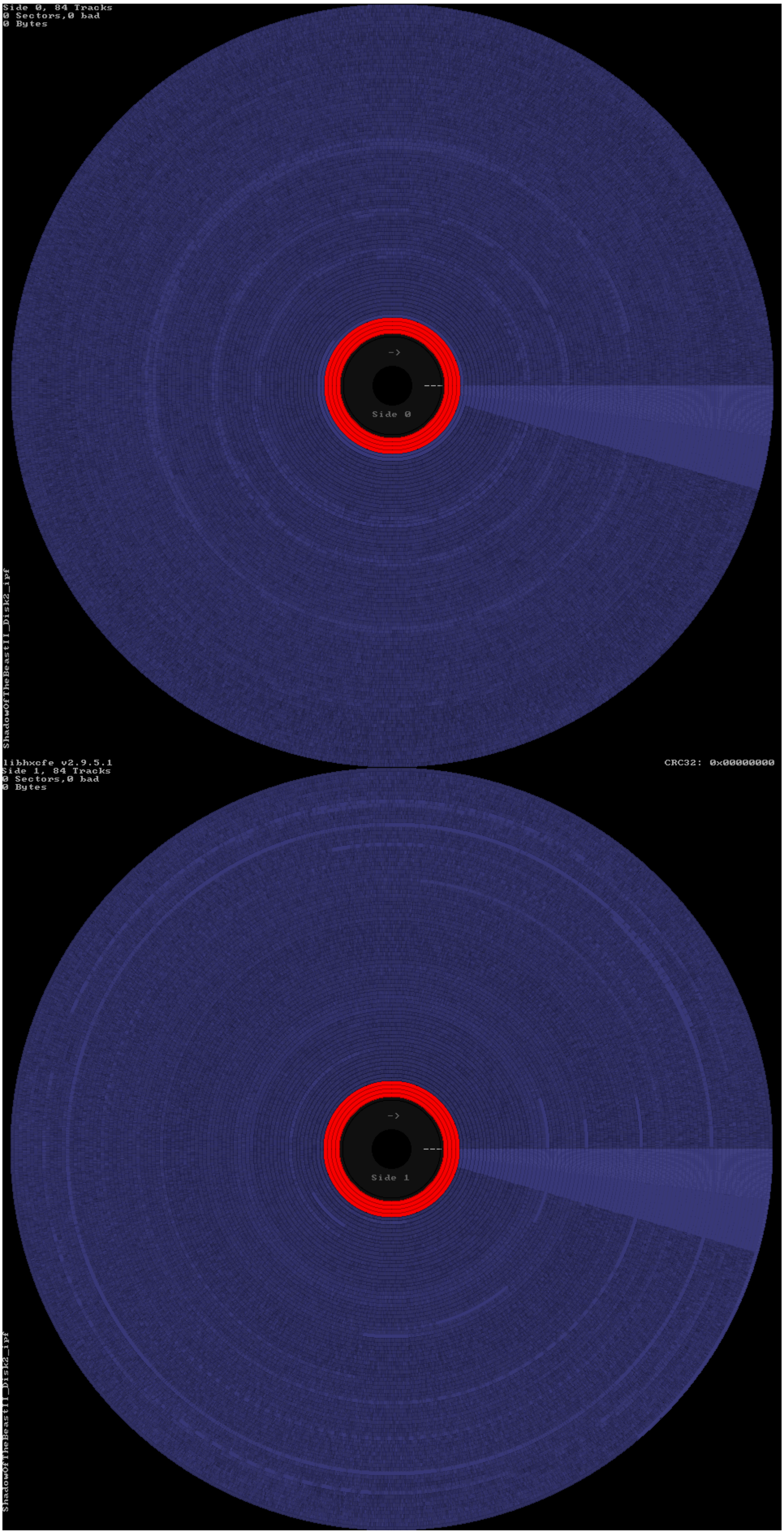
Gi@nts

Disk 1



Side 0, 84 Tracks
11 Sectors,0 bad
5632 Bytes
Amiga MFM
Side 0
ShadowOfTheBeastII_Disk1_ipf
libhxcfe v2.9.5.1
CRC32: 0x082B247E
Side 1, 84 Tracks
0 Sectors,0 bad
0 Bytes
Side 1
ShadowOfTheBeastII_Disk1_ipf

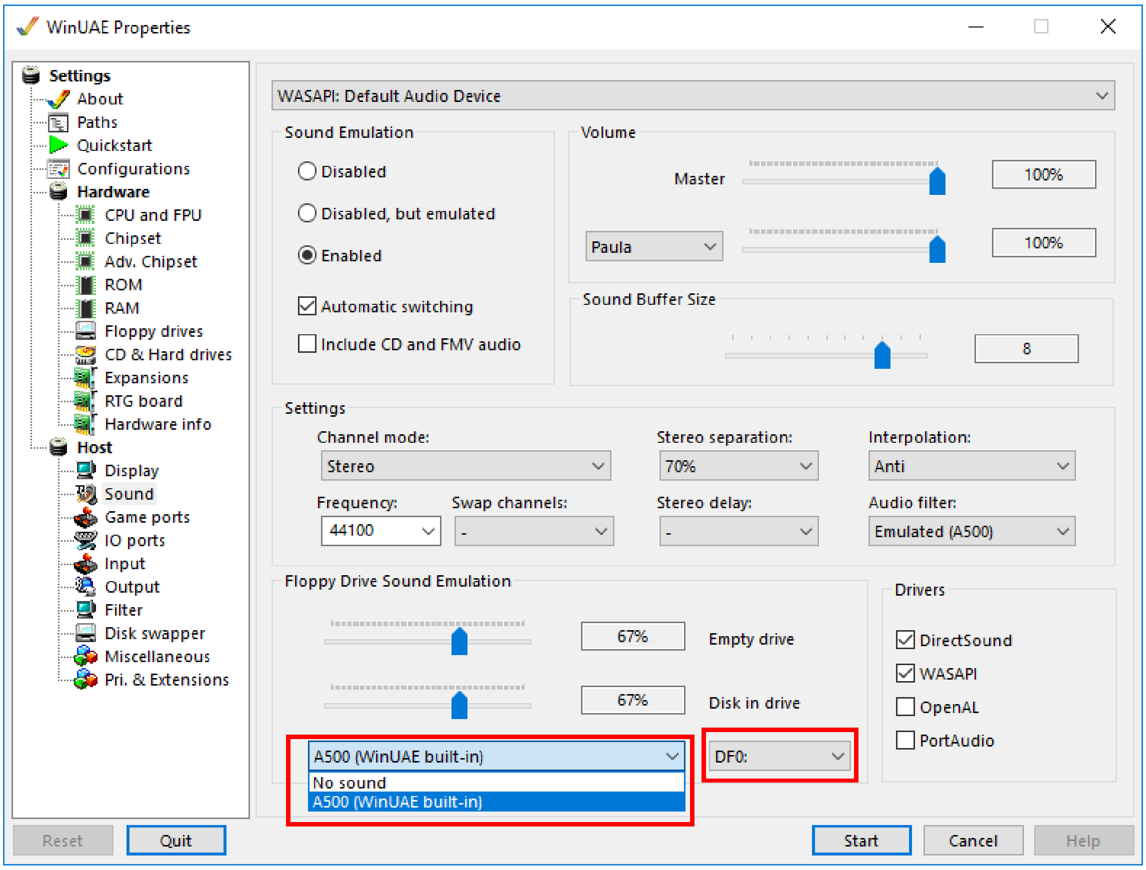
Disk 2



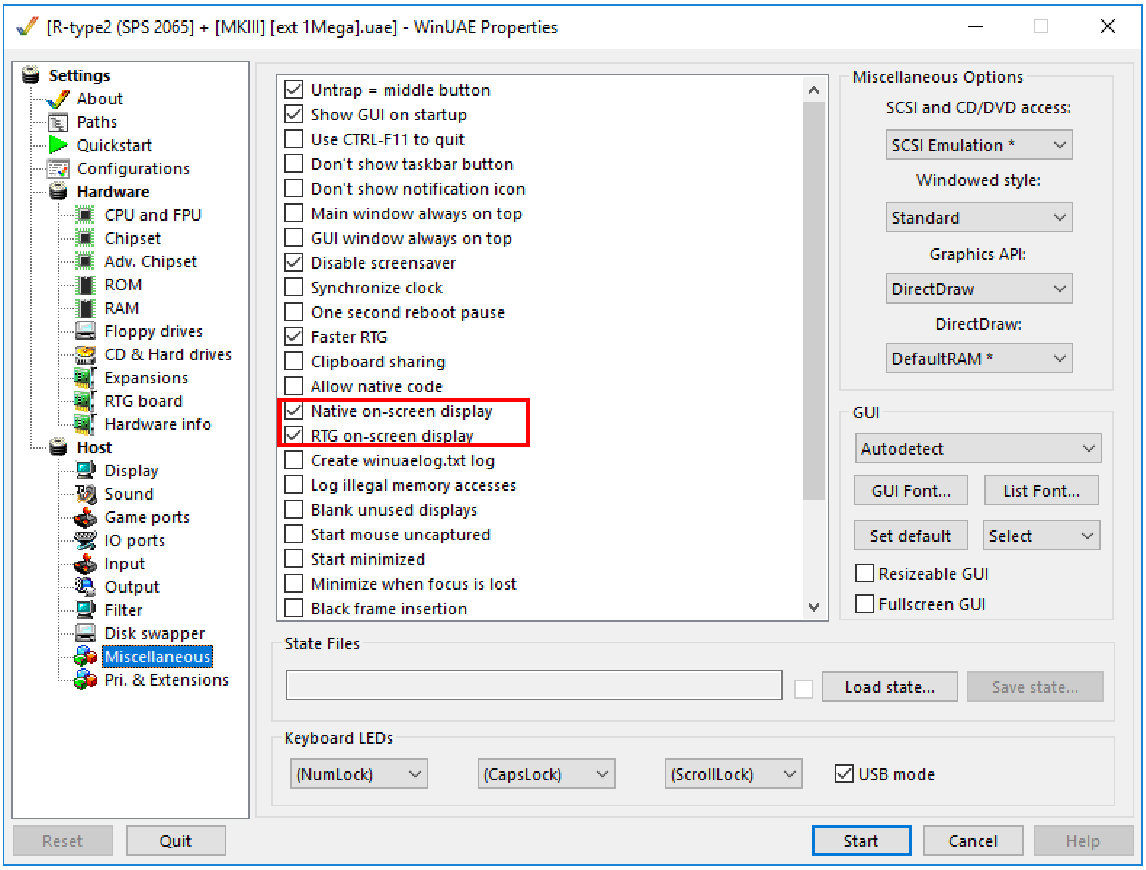
Side 0, 84 Tracks
0 Sectors,0 bad
0 Bytes
Side 0
ShadowOfTheBeastII_Disk2_ipf
libhxcfe v2.9.5.1
Side 1, 84 Tracks
0 Sectors,0 bad
0 Bytes
CRC32: 0x00000000
Side 1
ShadowOfTheBeastII_Disk2_ipf

WinUAE

For those who use winUAE for these tutorials (I guess most people), I strongly advise you to turn on the sound of the floppy drives so you can hear what the drive is doing as access.
HOST -> SOUND -> FLOPPY DRIVE SOUND EMULATION - > DF0 Built-In



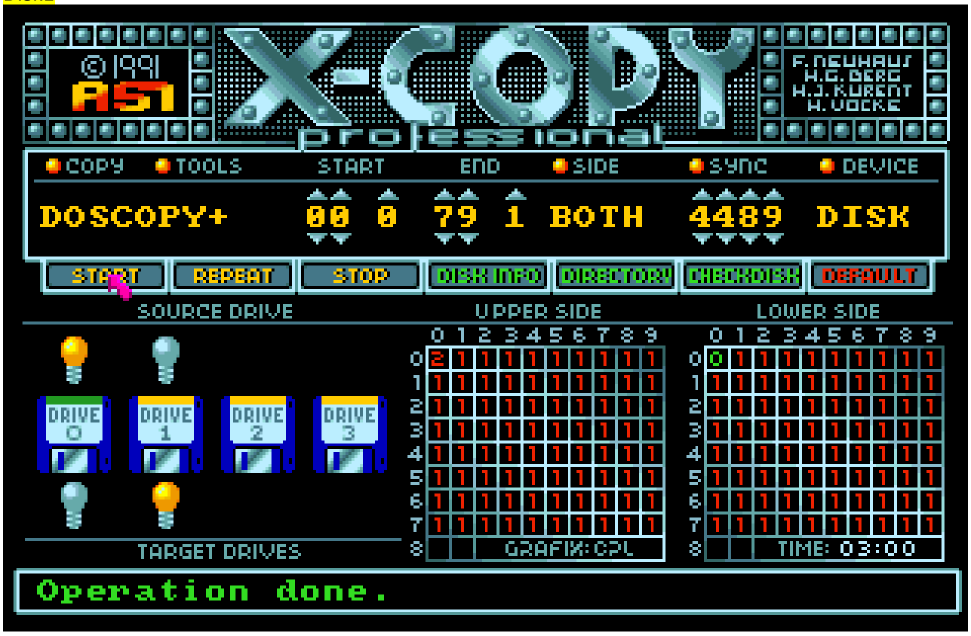
For more information such as which side we are :
Host -> Miscellaneous -> Native on-screen display AND RTG on-screen display



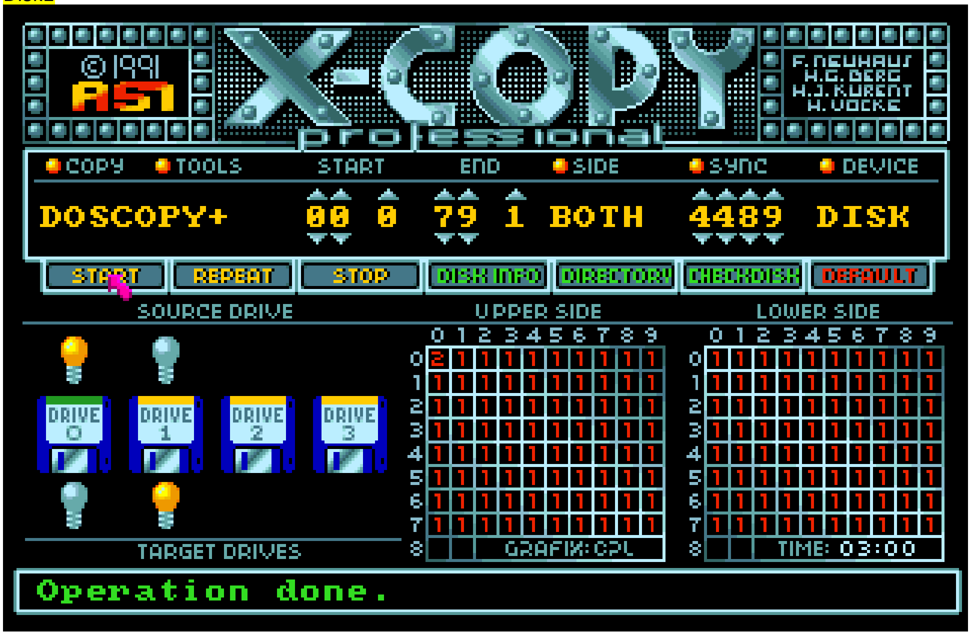
Part 1 X-copy

The first thing to do is to try to make backup of our floppy disks.
For this, we will use X-Copy.

Disk1



Disk2



Except the track 00 of the 1st floppy disk, the whole seems to be impossible to copy.
Note : 1 track that seems different on the two floppy disks in the T00 Upper position, probably a track dedicated to the disk signature.
It seems obvious that our backups will not work. Nevertheless we will keep the copy of the 1st floppy disk in order to work on its *bootSector*.

Review of Xcopy error codes :

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 of IPF files

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

About Track 00.0 of the 1st disk, nothing special: 11 sectors of 512 bytes, AmigaDOS standard.

TRACK		Data Length (bytes)		Data (bits)				CRC32 of the complete Extra Data Block			Adress
Data block Description	Sector ID	Data MFM bits	Data bytes	bytes/sector	GAP MFM bits	GAP bytes	Codage	GapDef	DataOff MFM bits	DataOff bytes	Adresse
[T00.0]		6446		51568				4C164D7B			13576-20021
#0	0	8704	545	512	0	1	MFM	0352	0352	15	13576-13607
#1	1	8704	545	512	0	1	MFM	0906	0906	57	13608-13639
#2	2	8704	545	512	0	1	MFM	1460	1460	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	9872	618	MFM	5892	5892	369	13896-13927

Then and on both floppy disks we find a unique and proprietary format until the end of the disk.
(Highlighted in green below)
Size of **Data : 6352**

Disk 1

TRACK		Data Length (bytes)		Data (bits)				CRC32 of the complete Extra Data Block			Adress
Data block Description	Sector ID	Data MFM bits	Data bytes	bytes/sector	GAP MFM bits	GAP bytes	Codage	GapDef	DataOff MFM bits	DataOff bytes	Adresse
[T00.1]		6048		48384				39DE4C18			20050-26097
#0	N/A	96112	6008	N/A	15376	962	MFM	0032	0032	2	20050-20081
[T01.0]		6352		50816				A1D37604			26126-32477
#0	N/A	100976	6312	N/A	4640	291	MFM	0032	0032	2	26126-26157
[T01.1]		6352		50816				477D023B			32506-38857
#0	N/A	100976	6312	N/A	4640	291	MFM	0032	0032	2	32506-32537
[T02.0]		6352		50816				1292FFE8			38886-45237
#0	N/A	100976	6312	N/A	4640	291	MFM	0032	0032	2	38886-38917
[T02.1]		6352		50816				2EC87572			45266-51617

Only the Track T00.1 of disk 1 and 2 come out of the batch, we can see it under X-copy with an **Error Type 2.**
As under my IPF image analyzer at Data size: **6048**
(highlighted) in red in the above image of Disk1 and below of Disk2)

Disk 2

TRACK		Data Length (bytes)		Data (bits)				CRC32 of the complete Extra Data Block			Adress
Data block Description	Sector ID	Data MFM bits	Data bytes	bytes/sector	GAP MFM bits	GAP bytes	Codage	GapDef	DataOff MFM bits	DataOff bytes	Adresse
[T00.0]		6352		50816				616D61B5			13576-19927
#0	N/A	100976	6312	N/A	4616	289	MFM	0032	0032	2	13576-13607
[T00.1]		6048		48384				6ACDE18D			19956-26003
#0	N/A	96112	6008	N/A	15344	960	MFM	0032	0032	2	19956-19987
[T01.0]		6352		50816				89FBE992			26032-32383
#0	N/A	100976	6312	N/A	4608	289	MFM	0032	0032	2	26032-26063
[T01.1]		6352		50816				F5E7E137			32412-38763
#0	N/A	100976	6312	N/A	4616	289	MFM	0032	0032	2	32412-32443
[T02.0]		6352		50816				F8D0C550			38792-45143

This would confirm the idea that one finds the signature of the floppy disks in **T00.1** and not the Data

In any case, both floppy disks seem to be well filled.

Part 3 Cheats

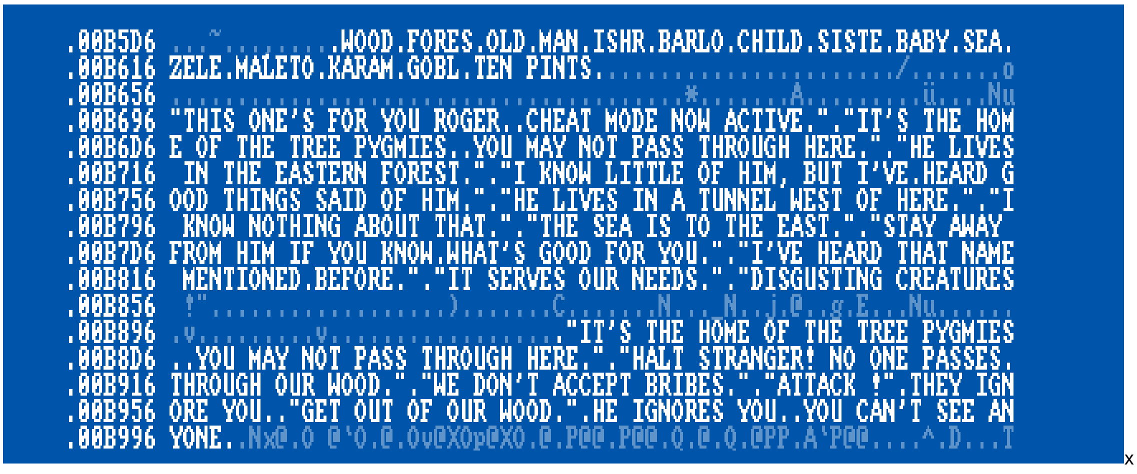
[S] Key during game allows to display the **SCORE** and **MONEY**.

```
$2B1 → $2B3          : MONEY, decimal coded, maximum = 99 99 99
$2B5 → $2B7          : SCORE, decimal coded, maximum = 99 99 99

$2AF                 : Energy. Quite vicious to find, the counter is reversed.
                       $00=Full Energy
```

MS 2AF allows you to find $006FCE **DBF D5,6FC6** we go back in the code (1 line above) and we find
$006FCA **ADDQ.W #4,2AE.S** To have the infinite Energy, we replace by
$006FCA **TST.W 2AE.S**

Keys [Q] or [A] (depending on the amiga keyboard) during game allows to 'start' a dialogue.
Only possible when one encounters intelligent life.

F "CHEAT" give you only 1 result ➔ *$B6B1*
We go up a little higher and we see a 'text area'. **Words** and **phrases** used in game.



```
.00B5D6 ...~.........WOOD.FORES.OLD.MAN.ISHR.BARLO.CHILD.SISTE.BABY.SEA.
.00B616 ZELE.MALETO.KARAM.GOBL.TEN PINTS.......................:/.......o
.00B656 ...................................*.......A..........ü....Nu
.00B696 "THIS ONE'S FOR YOU ROGER..CHEAT MODE NOW ACTIVE."."IT'S THE HOM
.00B6D6 E OF THE TREE PYGMIES..YOU MAY NOT PASS THROUGH HERE."."HE LIVES
.00B716  IN THE EASTERN FOREST."."I KNOW LITTLE OF HIM, BUT I'VE.HEARD G
.00B756 OOD THINGS SAID OF HIM."."HE LIVES IN A TUNNEL WEST OF HERE."."I
.00B796  KNOW NOTHING ABOUT THAT."."THE SEA IS TO THE EAST."."STAY AWAY
.00B7D6 FROM HIM IF YOU KNOW.WHAT'S GOOD FOR YOU."."I'VE HEARD THAT NAME
.00B816  MENTIONED.BEFORE."."IT SERVES OUR NEEDS."."DISGUSTING CREATURES
.00B856  !"..................)........C........N...N..j.@..g.E...Nu......
.00B896 .v.........v.........................."IT'S THE HOME OF THE TREE PYGMIES
.00B8D6 ..YOU MAY NOT PASS THROUGH HERE."."HALT STRANGER! NO ONE PASSES.
.00B916 THROUGH OUR WOOD."."WE DON'T ACCEPT BRIBES."."ATTACK !".THEY IGN
.00B956 ORE YOU.."GET OUT OF OUR WOOD.".HE IGNORES YOU..YOU CAN'T SEE AN
.00B996 YONE..Nx@.O @'O.@.Ov@XOp@XO.@.P@@.P@@.Q.@.Q.@PP.A'P@@....^.D...T
```

x

Directly under the game it's possible to open 'a dialogue' during an encounter with intelligent life.
Example: At the beginning of the game, on the right you meet a indigenous man with a spear.
By trying the words above (WOOD, FORES, OLD…) it's possible to create the following 'table'.

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

Example : During a meeting requiring a password and by using the same method we also come across a **ASCII** chain.
F "WHISPER" gives us a single result ➔ *$E03E* *$E46C* *$E49C* *$E4CE* *$E500*

E03E : End of the ASCII chain, nothing afterwards.

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

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

Tips : During Animation, enter into AR and enter : **G 638** to skip all the animation. (see end of Part8 for more information)

Part 4 Game loading behavior and testing of our Backup

Before getting to the heart of the matter, we will linger 5 minutes to see how the game behaves in terms of 'loading'.
Insert the original floppy disk into the drive and boot it.

This is what can be deduced from the loading noise of the floppy drive tracks.

'Track' Read	Display After loading or info	'Track' Read under WinUAE
00➔	**BOOT**	00
➔47	**N/A**	N/A
24	***N/A***	47-70 **Lower**
🡐 00	***PSYGNOSIS ANIMATION***	N/A
47	***ANIMATION PART #1***	01-47 **Lower**
🡐 00	***N/A***	N/A
48	***ANIMATION PART #2***	01-48 **Upper**
➔ 70	***N/A***	N/A
7	***N/A***	70-76 **Lower**
🡐 00	**INSERT DISK2**	N/A
INSERT DISK 2		
1	**N/A**	1
🡐 67	**N/A**	N/A
3	**N/A**	67-69
2	**N/A**	70-71
➔ 64	**N/A**	N/A
🡐 00	**N/A**	N/A
1	**N/A**	01-03
1	**N/A**	04-04
10	**N/A**	05-14
1	**N/A**	15-15
2	**N/A**	15-16
MAIN MENU		
PRESS FIRE		
9	**N/A**	16-24
2	**N/A**	24-25
9	**N/A**	26-34
2	**N/A**	35-36
2	**N/A**	36-37
🡐 00	**N/A**	N/A
LEVEL1		

Note : It's not necessarily 'right' at 'one unit' because it's not obvious to the ear to be formal about the loading or not of a track. But it gives an idea of the disk accesses.

Now it's time to test our copy made with **X-copy.**
Insert our backup into the drive and boot it.

'Track' Read	Display After loading or info	'Track' Read under
00➔	**BOOT**	00
➔47	**N/A**	N/A
- RED SCREEN -		

Part 5 Analyse and modification of the bootblock

Let's see and analyze our bootblock.
Always with the 'backup' in the drive.

#RT alias Read Track, allow loading of tracks <Track Start> <Count> <Memory Dest.>
#D, alias Desassemble
Type in : RT 0 1 20000 then D 20000+c
+c because the code of bootblock start in this adress, before it's the AmigaDOS disk signature.

Let's look this in detail:

```
2000C    MOVEA.L  A1,A5                    ; Save adr. of trackdisk.device in A1 on boot into A5
2000E    MOVE.L   #1200,D0                 ; D0     = 1200
20014    MOVE.L   D0,24(A1)                ; $24(A1)= 1200 Nmbr of Bytes to reserve
20018    MOVE.L   #2,D1                    ; D1     = 2 (req)
2001E    MOVEA.L  4.S, A6                  ; ExecBase in A6
20022    JSR      -C6(A6)                  ; Call AllocMem (memory reservation)

20026    MOVE.L   D0,28(A5)                ; 28(A5) = Adr of AllocMem in D0   Memory Destination Adr. for trackdisk.device
2002A    BEQ      200D8                    ; Problem ? GoTo → #Crash_red_screen
2002E    MOVEA.L  D0,A4                    ; A4=D0  = $59E8  We retrieve the address of the reserved area
                                           ; Amiga500 + 512K Memory Ext. = Memory Area $59E8
                                           ;
20030    MOVE.L   #400,2C(A5)              ; 2C(A1) = $400   Raw. Disk. Position 'start'
20038    MOVE.W   #2,1C(A5)                ; 1C(A5)= Read Mode
2003E    MOVE.L   A5,A1                    ; A1 is restored
20040    JSR      -1C8(A6)                 ; Call of Trackdisk.device for the loading of $1200 bytes
20044    TST.L    D0                       ; Test D0
20046    BNE      200D8                    ; Loading Problem ? GoTo → #Crash_red_screen
                                           ;
                                           ; #Base_Conf
2004A    MOVE.W   #7FFF,D6                 ; D6=$7FFF
2004E    LEA      DFF000,A6                ; A6=DFF000
20054    MOVE.W   D6,9A(A6)                ; DFF09A=Conf. INTENA // Disable all interrupts
20058    MOVE.W   D6,96(A6)                ; DFF096=Conf. DMACON // Disable all DMA
2005C    LEA      20068(PC),A0             ; A0=20068(PC)
20060    MOVE.L   A0,20.S                  ; A0=20
20064    MOVE.W   #2700,SR                 ; Conf Status Register
20068    MOVE.W   #2700,SR                 ; Conf Status Register
2006C    LEA      BFDA00,A1                ; CIAB, A1=todhi (Horizontal sync event bit 23-16)
20072    BSET     #7,400(A1)               ; CIAB, CRA (Control register A), set bit à 1
20078    MOVEQ    #0,D2                    ; D2=00
2007A    MOVEQ    #0,D3                    ; D3=00

                                           ; #Conf Sync #1
2007C    MOVE.W   D6,9C(A6)                ; Conf INTREQ à $7FFF // Clear all pending interrupts.
20080    BTST     #5,1F(A6)                ; → Test bit 5 Intreq register (vbl)  (Vertical Blank Line)
20086    BEQ      20080                    ; ← as long as VLB is not reach, we loop
20088    MOVE.B   (A1),D3                  ; D3=(A1), CIAB, A1=todhi (Horizontal sync event)
2008A    SWAP     D3                       ; Inverse in longword D3 00001111 becomes 11110000 for example
2008C    MOVE.B   -100(A1),D3              ; Address pointed by (BFDA00) -100 = BFD900 = CIAB, todmid put into D3
20090    LSL.W    #8,D3                    ; Shift of 8bit to the left of D3
20092    MOVE.B   -200(A1),D3              ; Address pointed by A1 (BFDA00) -200 = BFD800 = CIAB, todlo put into D3

                                           ; #Conf Sync #2
20096    MOVE.W   D6,9C(A6)                ; Conf INTREQ à $7FFF // Clear all pending interrupts.
2009A    BTST     #5,1F(A6)                ; → Test bit 5 Intreq register (vbl)  (Vertical Blank Line)
200A0    BEQ      2009A                    ; ← as long as VLB is not reach, we loop
200A2    MOVE.B   (A1),D2                  ; D2=(A1), CIAB, A1=todhi (Horizontal sync event)
200A4    SWAP     D2                       ; Inverse in longword D2 00001111 becomes 11110000 for example
200A6    MOVE.B    -100(A1),D2             ; Address pointed by A1 (BFDA00) -100 = BFD900 = CIAB, todmid
200AA    LSL.W    #8,D2                    ; Shift of 8bit to the left of D2
200AC    MOVE.B   -200(A1),D2              ; Address pointed by A1 (BFDA00) -200 = BFD800 = CIAB, todlo put into D3
200B0    SUB.L    D3,D2                    ; D2=D2-D3
200B2    CMP.W    #11F, D2                 ; Compare 011F and D2. Flag N is set if result is 'smaller than'
200B6    SLT      C0.S                     ; If the result is 'smaller than' then connect to C0

                                           ; #Copy_code_to_70000
200BA    LEA      70000,A7                 ; A7=70000
200C0    MOVEA.L  A7,A2                    ; A2=A7=70000
200C2    MOVE.W   #47F,D0                  ; D0=47F (counter)
200C6    MOVE.L   (A4)+,(A2)+              ; → copy in LongWord in (A4) to (A2), then A4=A4+4 and A2=A2+4
                                           ;    So copy the data already 'read' in (A4) to $70000
                                           ;    A498 without 512 Ext and 59E8 on A500 + Memory Ext
200C8    DBF      D0,200C6                 ; ← as long as D0 is different from -1, we loop
                                           ;    ($47F * 4) + 4 = $1200  Bytes copy to $70000
                                           ;
200CC    MOVE.L   A7,10.S                  ; Address of the Stack copied to the adress $10
200D0    MOVEQ    #8,D0                    ; D0=8
         ------------------------------------ Well, The AR don't understand that, in truth it gives us this →
         200D2    LINEF                    ; 4E 7B        → 200D2    MOVEC D0,CACR  ;
         200D4    ORI.B    #D7,D2          ; 00 02 4E D7  → 200D6    JMP (A7)        ; We jump to $70000
         ------------------------------------------------------------------------
200D8    MOVE.W   #F00,180(A6)             ; → #Crash_red_screen
200DE    BRA      200D8                    ; ← Dead Loop
================================
```

As it is preferable to work with the real addresses used in the game and for a better ease of analysis,
We are going to make some modifications on this code.

#A, alias Assemble, Instruction that will allow to type assembly code.
#BOOTCHK Alias Boot Check. Allows to calculate a new checksum for a bootblock
#WT, alias Write Track. Allows you to write a memory area on the disk at the address indicated in 'Track', ak $1600

After a call from *Trackdisk.device*, it copy and execute the code in *$70000*, so..we will change it at this address

Type in :

```
A 20000+400
$020400  MOVE.W   #0F0,DFF180          ; → Green Screen
$020408  BRA      20400                ; ← Our Dead-Loop
$02040A  <RETURN>
```

For information →

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

We re-calculate the boot-checksum
BOOTCHK 20000

and **always** with the **Floppy backup** in DF0, we write this new *bootblock*
WT 0 1 20000

Now, just **Reboot** your Amiga, very quicly we arrive on our green screen.
We **Enter into AR** and the original code is returned.

Type in :

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

Part 6 Analyse of the TrackLoader #1

Let's take a look of the code in *$70000*, it's a **TrackLoader** for sure.
After studying the code, here is what we can retain.
Type in : D 70000

#Pre_Conf_TrackLoader

```
70000    MOVE.W   #2700,SR          ; Conf Status Register
70004    MOVE.W   #7FFF,DFF096      ; Conf. DMACON // Clear all pending interrupts.
7000C    MOVE.W   #8210,DFF096      ; Conf. DMACON
70014    MOVE.W   #7FFF,DFF09A      ; Conf INTENA // Disable all interrupts.
7001C    MOVE.W   #7FFF,DFF09C      ; Conf INTENA // Clear all pending interrupts.
70024    CLR.W    DFF180            ; black screen
7002A    BSET     #1,BFE001         ; CIA-A / PRA, test of the bit5, drive ready ?
70032    LEA      256.S,A7          ; A7=256, change address of the stack
70036    BSR      70078             ; GoSub ➔ #Trackloader_Init
                                    ;
7003A    LEA      45610,A0          ; A0=45610
                                    ; A0 is used in the routine of #Decrypt_decomp_Init
                                    ; so A0=Destination Address for raw read data (i.e Not decrypted)
                                    ;
                                    ; $45610->$69894  Data not decrypted/decompressed = $24284 = !148100
                                    ;
70040    LEA      70060(PC),A1      ; A1=70060(PC)                A1=70060 (position in the table)
70044    BSR      701E2             ; GoSub ➔ #Trackloader_Start
                                    ; A0=Adr_memory_dest          A1=Table Pointer Adress_Start_raw
                                    ;
                                    ; The data is now loaded, it needs to be decompressed/decrypted.
70048    LEA      2B0.S,A1          ; A1=$2B0                     A1=Final Position in memory (after decrypt/decomp)
7004C    BSR      70310             ; GoSub ➔ #Decrypt_decomp_Init
                                    ;
                                    ; UPDATE of the table for future calls.
70050    LEA      70068(PC),A0      ; A0=70068(PC)                A0=position in the table
70054    MOVE.L   (A0)+,C2.S        ; Copy of (A0) into $C2, so $C2= 0006CA94        /!\ important     And A0=A0+4
70058    MOVE.L   (A0)+,C6.S        ; Copy of (A0) into $C6, so $C6= 00009B78        /!\ important     And A0=A0+4
7005C    JMP      2B0.S             ; We jump to $2B0                ➔ // Start of the psygnosis Animation
```

#Table_#01

```
70060    00 04 88 10                ← Address_Start_raw, (Minimum $189C, size of a custom Track)
                                      (so, Require to minimum start in Track 1 (Because track 0 is specific)
70064    00 02 42 84                ← Length_To_Read
70068    00 06 CA 94                ← Put in $C2, no idea why.
7006C    00 00 9B 78                ← Put in $C6, no idea why.
70070    00 00                      ← Pointer Track in progress.
70072    00 00                      ← Marker for Trackloading or not.
70074    00 06 A0 00                ← Conf DSKPTH
```

#Trackloader_Init

```
70078    MOVE.W   #10,DFF096        ; Conf DMACON
70080    MOVE.W   #2,DFF09C         ; Conf INTENA
70088    MOVE.W   #7FFF,DFF09E      ; Conf ADKCON
70090    MOVE.W   #8100,DFF09E      ; Conf ADKCON
70098    ORI.B    #78,BFD100        ; 'OR' binary between 0111 1000 and  $BFD100, so DF0 to DF3 select and Motor ON
700A0    BCLR     #7,BFD100         ; CIA-B / PRB, bit7 set to 0, motor On
700A8    BCLR     #3,BFD100         ; CIA-B, bit3 set to 0, DF0 selected
700B0    MOVE.W   #BB8,D6           ; ➔ D6=BB8
700B4    DBF      D6,700B4          ; ← D6=D6-1, loop as long D6 is different from -1. Her ewe have a 'pause'
700B8    BTST     #5,BFE001         ; ➔ CIA-A / PRA, test of the bit5, Drive Ready ?
700C0    BNE      700B8             ; ← As long is not ready, we loop.
700C2    LEA      70072(PC),A3      ; A3=70072(PC) Adress in the table.
700C6    MOVE.W   #1,(A3)           ; Copy the word 1 in (A3) (so put 00 01 in table at $70072)
700CA    RTS                        ; E.T Back Home
```

#Move_Inside.

```
700CC    BCLR     #1,BFD100         ; CIA-B / PRB, bit1 set to 0, DIR=toward the inside.
700D4    BCLR     #0,BFD100         ; CIA-B / PRB, bit0 set to 0, STEP, Head pre/move
700DC    BSET     #0,BFD100         ; CIA-B / PRB, bit0 set to 0, STEP, Head move
700E4    BSR      70170             ; GoSub ➔ #WAIT
700E8    MOVE.L   A0,-(A7)          ; Save A0 into stack
700EA    LEA      70070(PC),A0      ; 70070(PC)=A0 Adress into the table
700EE    ADDQ.W   #1,(A0)           ; (A0)=A0+1 Word
                                    ; We moved one Track inward and (A0) $70070 is incremented by 1
                                    ; $70070 so it's a Track pointer.
                                    ; We can also speak of 'cylinder' because we move only on one side.
                                    ; So, when we move forward one track, we also move forward 1 cylinder.
                                    ;
700F0    MOVEA.L  (A7)+,A0          ; Restore A0 from stack
700F2    BRA      70164             ; GoTo #Update_table
```

#Move Outside.

```
700F6    BSET     #1,BFD100        ; CIA-B / PRB, bit1 set to 0, DIR= Towards the outside
700FE    BCLR     #0,BFD100        ; CIA-B / PRB, bit0 set to 0, STEP, Head pre/move
70106    BSET     #0,BFD100        ; CIA-B / PRB, bit0 set to 0, STEP, Head move
7010E    BSR      70170            ; GoSub → #WAIT
70112    MOVE.L   A0,-(A7)         ; Save A0 into stack
70114    LEA      70070(PC),A0     ; 70070(PC)=A0 Adress in table
70118    SUBQ.W   #1,(A0)          ; We moved one Track inward and (A0) $70070 is decremented by 1
                                   ; $70070 so it's really a Track pointer.
                                   ; We can also speak of 'cylinder' because we move only on one side.
                                   ; So, when we move forward one track, we also move forward 1 cylinder.
                                   ;
7011A    MOVEA.L  (A7)+,A0         ; Restore A0 from stack
7011C    BRA      70164            ; GoTo → #Update_table
```

#Position Reach?

```
70120    CMP.W    70070(PC),D0     ; Compare D0 with $70070(PC), adress read from the table
70124    BEQ      70138            ; If equal then GoTo → #Move_Forward/back
70126    BGT      7013C            ; If N=0, (greater than), then GoTo #GoTo_Position
70128    MOVE.W   70070(PC),D6     ; D6=70070(PC) = Adress in table
7012C    SUB.W    D0,D6            ; D6=D6-D0
7012E    SUBQ.W   #1,D6            ; D6=D6-1 in word
70130    BSR      700F6            ; → GoSub → #Move Outside.
70132    DBF      D6,70130         ; ← D6=D6-1, loop as long D6 is different from -1.
70136    RTS                       ; E.T Back Home
```

#Move_Forward/back

```
70138    BSR      700CC            ; GoSub → #Move Inside.
7013A    BRA      700F6            ; GoTo → #Move Outside.   (This explains the jerking noises during loading.)
```

#GoTo_Position

```
7013C    SUB.W    70070(PC),D0     ; D0 'word' = Table's word in 70070(PC)-D0
70140    SUBQ.W   #1,D0            ; D0=D0-1
70142    BSR      700CC            ; → GoSub → #Move Inside.
70144    DBF      D0,70142         ; ← D0=D0-1, loop as long D0 is different from -1, we loop.
70148    RTS                       ; E.T Back Home
```

#Return_T00

```
7014A    MOVE.B   BFE001,D         ; D0=CIA PRA
70150    BTST     #4,D0            ; Test of bit4, TK0, Head on T00 ?
70154    BEQ      7015E            ; Yes ? GoTo → #Update_table
70156    BSR      700F6            ; No ? GoSub → #Move Outside
70158    BSR      70170            ; GoSub #WAIT
7015C    BRA      7014A            ; GoTo → #Return_T00                 // We loop to Return_T00
```

#Update_table

```
7015E    LEA      70070(PC),A3     ; A3=position in the table
70162    CLR.W    (A3)             ; Clear the Word in (A3)
70164    BTST     #5,BFE001        ; → CIA-A / PRA, test bit5, Drive Ready ?
7016C    BNE      70164            ; ← loop as long Drive is not ready.
7016E    RTS                       ; E.T Back Home
```

#WAIT

```
70170    MOVE.W   D7,-(A7)         ; Save D7 into stack
70172    MOVE.W   #1388,D7         ; → D7=1388
70176    DBF      D7,70176         ; ← D7=D7-1, loop as long D7 is different from -1.
7017A    MOVE.W   (A7)+,D7         ; Restore D7 from stack
7017C    RTS                       ; E.T Back Home
```

#DF0__SIDE_DOWN__MOTOR_OFF__DIR_EXT

```
7017E    MOVE.B   #FD,BFD100          ; Conf CIA-B / PRB
                                      ; MOTOR OFF, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 UNSELECT, SIDE DOWN, DIR EXT., STEP TRACK=1
                                      ;
70186    MOVE.W   #100,D0             ; → D0=100
7018A    DBF      D0,7018A            ; ← D0=D0-1, loop as long D0 is different from -1 (It's just a pause.)
7018E    MOVE.B   #F5,BFD100          ; Conf CIA-B / PRB
                                      ; MOTOR OFF, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 SELECT, SIDE DOWN, DIR EXT., STEP TRACK=1
                                      ;
70196    MOVE.W   #B000,D0            ; D0=B000
7019A    DBF      D0,7019A            ; ←→ D0=D0-1, loop as long D0 is different from -1 (It's just a pause.)
7019E    LEA      70072(PC),A3        ; A3=position in the table
701A2    CLR.W    (A3)                ; Clear the Word in (A3)
701A4    RTS                          ; E.T Back Home
```

#DF0__SIDE_DOWN__MOTOR_ON__DIR_EXT

```
701A6    MOVE.B   #7D,BFD100          ; Conf CIA-B / PRB
                                      ; MOTOR ON, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 UNSELECT, SIDE DOWN, DIR EXT., STEP TRACK=1
701AE    NOP                          ;
701B0    NOP                          ;
701B2    MOVE.B   #75,BFD100          ; Conf CIA-B / PRB
                                      ; MOTOR ON, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 SELECT, SIDE DOWN, DIR EXT., STEP TRACK=1
701BA    MOVE.W   #B000,D7            ; D7=B000
701BE    DBF      D7,701BE            ; ←→ D7=D7-1, loop as long D7 is different from -1 (again, it's a pause)
701C2    RTS                          ; E.T Back Home
```

#DF0__SIDE_UP__MOTOR_ON__DIR_EXT

```
701C4    MOVE.B   #79,BFD100          ; Conf CIA-B / PRB
                                      ; MOTOR ON, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 UNSELECT, SIDE UP, DIR EXT., STEP TRACK=1
701CC    NOP                          ;
701CE    NOP                          ;
701D0    MOVE.B   #71,BFD100          ; Conf CIA-B / PRB
                                      ; MOTOR ON, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 SELECT, SIDE UP, DIR EXT., STEP TRACK=1
701D8    MOVE.W   #B000,D7            ; D7= B000
701DC    DBF      D7,701DC            ; ←→ D7=D7-1, loop as long D7 is different from -1 (again, it's a pause)
701E0    RTS                          ; E.T Back Home
```

#Trackloader_Start   **With**   **A0**=Memory_Adress   **A1**=Table pointer(Adress_Start_raw)

```
701E2    MOVE.L   A0,-(A7)            ; Save A0 into stack
701E4    BSR      701EC               ; GoSub → #Read_Table_and_Start_Trackload_Or_Not
701E8    MOVEA.L  (A7)+,A0            ; Restore A0 from stack
701EA    RTS                          ; E.T Back Home
```

#Read_Table_and_Start_Trackload_Or_Not

```
701EC    LEA      70072(PC), A3       ; A3=70072        (Position in the table)           // Marker Ready to Trackload or not
701F0    TST.W    (A3)                ; Test of A3 with zero
701F2    BNE      701F8               ; If different of zero then GoTo → #Side_Select
701F4    BSR      70078               ; else, GoSub → #Trackloader_Init

#Side_Select
701F8    MOVE.L   (A1)+,D0            ; Copy the LongWord (A1) into D0 then A1=A1+4
701FA    CMP.L    #84468,D0           ; D0-84468 and change flags et change les Flag accordingly.
                                      ; Test to know which side to use.

                                      ; According to result, we'll go to Side_UP or Side_Down
                                      ; so D0, alias A1, i.e $70060 indicates a starting position on the disk.
                                      ; ak : Address_Start_raw

70200    BGE      70206               ; If greather than, then.
                                      ; GoTo → #Recover_Info_for Trackloader_1 and #DF0__SIDE_UP__MOTOR_ON__DIR_EXT
                                      ;
70202    BSR      701A6               ; Else GoSub → #DF0__SIDE_DOWN__MOTOR_ON__DIR_EXT
70204    BRA      70208               ; And then, GoTo → #Recover_Info_pour_Trackloader_2
```

#Recover_Info_For_Trackloader_1

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

#Recover_Info_For_Trackloader_2

```
70208    MOVE.L   D0,D3              ; At this step, D0=(LongWord of $70060) so the value in the table=$48810
                                     ; To be precise, Adress_Start_raw
                                     ; D3=D0, so D3=$48810
                                     ;
7020A    DIVU.W   #189C,D0           ; D0=D0/$189C, this validates two things :
                                     ; + The size of custom track here is $189C
                                     ; + D0 indicates a physical address on the disk.
                                     ; D0 now indicates Start_Track
                                     ;
7020E    CMP.W    #56,D0             ; Compare D0 with $56
70212    BLT      7021E              ; if smaller than, then GoTo $7021E
70214    SUBI.L   #84468,D3          ; else, D3=D3-$84468
7021A    SUBI.W   #56,D0             ; D0=D0-$56
                                     ;
7021E    MOVEQ    #0,D2              ; D2=00
70220    MOVE.W   D0,D2              ; D2=D0
70222    BSR      70120              ; GoSub ➔ #Position Reach?
                                     ;
70226    MULU.W   #189C,D2           ; At this point, D2 = $2F = !47 = position reach = Head position = Track in progress
                                     ; We multiply it by the size of a Track, it becomes the size in Position_raw_reach
                                     ; Position where we are, in this instance : $484A4
                                     ;
7022A    SUB.L    D2,D3              ; In this Step D3=(LongWord of $70060) so table's value = $48810
                                     ; or if D0 Start_Track > $56, we subtract the position_raw_reach of D3
                                     ; In fact, according to Adress_Start_raw in the table, we're going to use a specific side.
                                     ;
                                     ; So this operation carried out on D3 allows to reposition
                                     ; Adress_Start_raw while changing face.
                                     ; Example, D3=$48810(table's value in $70060)-$484A4(position reach)=$36C
                                     ;
7022C    MOVE.L   (A1),D4            ; Copy the longWord (A1) into D4
                                     ; A1=$70064=(Length_To_Read) = $24284 so D4=$24284
                                     ;
7022E    LSR.L    #2,D4              ; Shift of 2 Bits to the right the LongWord D4, which is equivalent to dividing D4 by 4,
                                     ; In our case, we give D4=90A1
                                     ;
70230    SUBQ.L   #1,D4              ; D4=D4-1, We remove 1 : D4=$90A0 (this allows you to get an even number)
                                     ;
70232    BSR      7023A              ; ➔      GoSub ➔ #Trackload_Base
70234    BSR      700CC              ;        GoSub ➔ #Move Inside
70238    BRA      70232              ; 🡐      GoTo  ➔ #Trackload_Base (Indirect)
```

#Trackload_Base

```
7023A    MOVEQ    #F,D2              ; D2=F
7023C    MOVE.L   #18B8,D0           ; ➔ D0=$18B8
70242    MOVEA.L  70074(PC),A6       ;   A6=Table's adress=0006A000
70246    MOVE.W   #2,DFF09C          ;   Conf INTREQ, Disk Block Finished Interupt
7024E    MOVE.L   A6,DFF020          ;   Conf DSKPTH (conf comes from the table), So DSKPTH=$6A000
70254    MOVE.W   #8210,DFF096       ;   Conf DMACON, DSKEN enable, ALL DMA enable
7025C    MOVE.W   #4489,DFF07E       ;   Conf DSKSYNC = $4489 (standard AmigaDos)
70264    MOVE.W   #7F00,DFF09E       ;   Conf ADKCON specific
7026C    MOVE.W   #B500,DFF09E       ;   Again to start the transfer.
                                     ;   FAST=2us, WORDSYNC=active, MFM precomp, recomp=140ns
                                     ;
70274    MOVE.W   #4000,DFF024       ;    Conf DSKLEN, Write enable (ram or disk)
                                     ;
                                     ;    Tst CIA Ready
7027C    MOVE.B   BFDD00,D7          ;    D7=BFDD00
70282    MOVE.B   BFDD00,D7          ;    D7=ICR register of CIA-B
70288    BTST     #4,D7              ;    ➔ Test bit4 (FLAG) of ICR
7028C    BEQ      70282              ;    🡐 Interruption generated ? We loop
7028E    ADDI.W   #8001,D0           ;    ADD signed on D0 (which is $18B8 see a few lines above)
                                     ;    Whith $8001, what gives us : $98B9 and flag C=0
                                     ;
70292    MOVE.W   D0,DFF024          ;    CONF DSKLEN, Disk DMA Enable, Length of DMA data=$18B9
70298    MOVE.W   D0,DFF024          ;    Again to start the transfer.
7029E    MOVE.W   DFF01E,D0          ;    ➔ D0=INTREQR
702A4    BTST     #1,D0              ;       Test Bit1 of INTREQR, Level 1 Disk Block Finished Interupt
702A8    BEQ      7029E              ;    🡐 Test ?, We loop
702AA    MOVE.L   (A6)+,D0           ;    Value of DSKPTH in D0, then A6=A6+4 (so A6=DFF024)
702AC    MOVE.L   (A6)+,D7           ;    Value of DSKLEN in D7, then A6=A6+4 (so A6=DFF028)
702AE    ADD.L    D0,D0              ;    D0=D0+D0
702B0    ANDI.L   #AAAAAAAA,D0       ;    Post_ Processing MFM bit odd in D0
702B6    ANDI.L   #55555555,D7       ;    Post_ Processing MFM bit even in D7
702BC    OR.L     D7,D0              ;    MFM Processing
702BE    CMP.L    #42535432,D0       ;    Compare D0 with 42 53 54 32, In Ascii = BST2.
                                     ;    If the signature is not found, we check again.
                                     ;    and finally, if still not good, 'start' on the red screen routine below.
                                     ;
702C4    BEQ      702D4              ;    YES ? Signature found GoTo ➔ #Processing_MFM_BASE
                                     ;    We start the actual data processing and trackload.
702C6    DBF      D2,7023C           ; 🡐  D2=D2-1, loop as long D2 is different from -1.
                                     ;    (let's go for a ride)
                                     ;
702CA    MOVE.W   #F00,DFF180        ; ➔ GGRRRR, 'bad move', red Screen.
702D2    BRA      702CA              ; 🡐 DeadLoop on the red Screen. (death loop)
```

#Processing_MFM_BASE

```
702D4     MOVEQ    #0,D2               ; D2=00
702D6     MOVE.W   #626,D2             ; D2=$626
                                       ;
702DA     TST.W    D3                  ; Test D3 with zero                          // D3=Delta calculated beforehand
702DC     BEQ      702E8               ; If equal then GoTo ➔ #Check Processing MFM
702DE     ADD.W    D3,D3               ; Else D3=D3+D3                              // x2 on D3
                                       ; Reminder, D3=((Table's value 70060)-(reached's position))
                                       ;
702E0     ADDA.L   D3,A6               ; A6=A6+D3                                   // A6=DSKPTH+decoding in progress
702E2     LSR.W    #3,D3               ; Shift The LongWord D3 of 3 Bits to the left        // Equals dividing by 8 D3
                                       ; So delta between position reached and (Value in $70064)/8
                                       ;
702E4     SUB.W    D3,D2               ; D2=D2-D3
                                       ;                            // D2(current_raw_position)-Delta calculated above
702E6     MOVEQ    #0,D3               ; D3=0                                       // We reset D3 to Zero
                                       ;
#Check_Processing_MFM
702E8     CMP.L    D2,D4               ; Compare D4-D2
702EA     BGT      702F2               ; If the result is greater than, GoTo ➔ #Processing MFM bit even
702EC     MOVE.W   D4,D2               ; D2=D4
702EE     ADDQ.W   #4,A7               ; A7=A7+4
702F0     BRA      702F6               ; GoTo ➔ #Start_Decoding
```

#Processing MFM bit even

```
702F2     SUB.L    D2,D4               ; D4=D4-D2
702F4     SUBQ.L   #1,D4               ; D4=D4-1
#Start_Decoding
702F6     MOVE.L   #55555555,D5        ; D5=55555555, mask MFM bit even
702FC     MOVE.L   (A6)+,D0            ; ➔  Copy (A6) into D0 then A6=A6+4
702FE     MOVE.L   (A6)+,D7            ;     Copy (A6) into D7 then A6=A6+4
70300     AND.L    D5,D0               ;     Processing MFM
70302     AND.L    D5,D7               ;     Processing MFM
70304     ADD.L    D0,D0               ;     Processing MFM
70306     OR.L     D7,D0               ;     Processing MFM
70308     MOVE.L   D0,(A0)+            ;     Copy D0 at the address pointed by A0, then A0=A0+4
7030A     DBF      D2,702FC            ; 🡐 D2=D2-1 and as long as D2 is different from -1, we loop
7030E     RTS                          ; E.T Back Home
```

#Decrypt_decomp_Init

```
70310     MOVEQ    #0,D7               ; D7=0
70312     MOVEA.L  A0,A2               ; A2=A0
70314     MOVE.L   (A0),D0             ; Copy of the LongWord pointed by A0 in D0
70316     BTST     #0,D0               ; Test D0 with 0
7031A     BEQ      70324               ; If  equal, then GoTo ➔ #Decrypt_Decomp_Base
7031C     MOVEA.L  A1,A3               ; A3=A1
7031E     NOT.W    D7                  ; ...
70320     ANDI.W   #FFFE,D0            ; ... A whole phase of data processing.

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

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

#Processing_D0_D1
70352     MOVEQ    #7,D1               ; ➔
70354     LSR.L    #1,D0               ;        ➔ Shift 1 bit to the right of D0
70356     BNE      7035A               ;          GoTo ➔ #Decrypt_D2
70358     BSR      703AE               ;          GoSub ➔ #Decrypt_D5
7035A     ROXL.L   #1,D2               ;          1 bit Extended left rotation of D2
7035C     DBF      D1,70354            ;        🡐
70360     MOVE.B   D2,-(A2)            ;
70362     DBF      D3,70352            ; 🡐 GoTo ➔ #Processing_D0_D1
70366     BRA      7039C               ; GoTo ➔ #Check_A1_A2
```

#D1_D4_8Byte

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

#Decrypt_phase1

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

#Decrypt_phase2

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

70392    SUBQ.L   #1,A2                ; →
70394    MOVE.B   0(A2,D2.W),(A2)      ;   Copy the result into (A2)
70398    DBF      D3,70392             ; ←

#Check_A1_A2
7039C    CMPA.L   A2,A1                ;
7039E    BLT      70334                ; GoTo → #Decrypt_Start
703A0    TST.L    D5                   ;
703A2    BNE      703D0                ; GoTo → #D0=-1
703A4    TST.W    D7                   ;
703A6    BEQ      703AA                ; GoTo → #RTZ_D0
703A8    BRA      703D4                ; GoTo → #Move_Decrypt_base
```

#RTZ_D0

```
703AA MOVEQ       #0, D0               ; Reset to Zero of D0
703AC RTS                              ; E.T Back Home
```

#Decrypt_D5

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

#Decrypt_D0_D2

```
703B8    SUBQ.W   #1,D1                ; D1=D1-1
703BA    CLR.W    D2                   ; D2=0 in Word
703BC    LSR.L    #1, D0               ; → Shift 1 bit to the right of D0
703BE    BNE      703C8                ;    GoTo → #Decrypt_D0_D2b
703C0    MOVE.L   -(A0),D0             ;
703C2    EOR.L    D0,D5                ;
703C4    MOVE.W   D6,CCR               ;
703C6    ROXR.L   #1,D0                ;

#Decrypt_D0_D2b
703C8    ROXL.L   #1,D2                ;
703CA    DBF      D1,703BC             ; ← D1=D1-1 and as long as D1 is different from -1, we loop
703CE    RTS                           ; E.T Back Home
```

#D0=-1

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

#Move_Decrypt_base

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

#Move_Decrypt_Start
70400    CMPA.L   A2,A1               ; Compare A1 with A2 (Test if decryption is complete.)
70402    BLE      7043E               ; GoTo ➔ #Decrypt_Done, A2 or A1 now contains the ‘Start addr.’ of data decomp/decrypt
70404    MOVE.B   -(A0),D0            ;
70406    CMP.B    D0,D4               ;
70408    BEQ      70430               ; GoTo ➔ #COPY_Decrypt
7040A    CMP.B    D0,D5               ;
7040C    BEQ      7041A               ; GoTo ➔ #Move_Decrypt_D0
7040E    CMP.B    D0,D6               ;
70410    BEQ      70424               ; GoTo ➔ #Move_Decrypt_D2
70412    CMP.B    D0,D7               ;
70414    BEQ      7042A               ; GoTo ➔ #Move_Decrypt_D3
70416    MOVE.B   D0,-(A1)            ;
70418    BRA      70400               ; GoTo ➔ #Move_Decrypt_Start
```

#Move_Decrypt_D0

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

#Move_Decrypt_D2

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

#Move_Decrypt_D3

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

#COPY_Decrypt

```
70430    MOVEQ    #0,D0               ; RTZ of D0
70432    MOVE.B   -(A0),D1            ; Update of D1, our data to copy.
70434    MOVE.B   -(A0),D0            ; Update of D0, our counter.

70436    MOVE.B   D1,-(A1)            ; ➔ Copy D1 into A1 then A1=A1-1
70438    DBF      D0,70436            ; ← D0=D0-1 and as long as D0 is different from -1, we loop
7043C    BRA      70400               ; GoTo ➔ #Move_Decrypt_Start
```

#Decrypt_Done

```
7043E    MOVEQ    #0,D0               ; D0=0
70440    RTS                          ; E.T Back Home
```

Conclusion :

We have here a TrackLoader that :

- Operates by 'Side', Upper or Lower (selection limit value=*$84468*)
  Works with a table that is updated before each TrackLoad. (at least for now)
- Custom Track Size of *$189C*
- 2 tracks to avoid on the 1st Disk (Track 00=BootSector, Track 01=SignatureDisk)
- 1 track to avoid on the second Disk (Track 00=SignatureDisk)
- **DSKPTH** configuration performed specific to each TrackLoad through a table.
- '**StartDisk**' adress specific to each TrackLoad through a table.
- '**Data Length**' specific to TrackLoad through a table.
- A Destination memory in **A0** before calling the TrackLoad routine.
- Compressed/encrypted data on the original disk.

The table concerned, presented in *$70060* and updated methodically.

#Table_#01

```
70060    00 04 88 10       ← Adress_Start_raw, (Minimum $189C, size of a Custom Track)
                             Requirement to start in cylinder 1 minimum (Track in 0 Specific)
70064    00 02 42 84       ← Length_To_Read
70068    00 06 CA 94       ← Put in $C2, no idea yet.
7006C    00 00 9B 78       ← Put in $C6, no idea yet
70070    00 00             ← Track Pointer in progress. (Track not half Track because we working by side)
70072    00 00             ← Marker to Trackload or not.
70074    00 06 A0 00       ← Conf DSKPTH
```

The process as seen in *$70000*

1	**A0** to define the 'Destination adr.' for the TrackLoader.	7003A	LEA	45610,A0
2	**A1 to define** 'adress of the table' to retrieve the informations : **DSKPTH** and **'Length to read'**	70040	LEA	70060(PC),A1
3	Execution of TrackLoading.	70044	BSR.W	701E2
	Trackloaded Data available in memory at A0			
4	**A1** to define the 'Destination adr.' for the decomp. Process.	70048	LEA	2B0.S,A1
5	Execution of decompression	7004C	BSR.W	70310
	Uncompressed Data available in memory at A1			
6	Update of **A0** for the Next Trackload.	70050	LEA	70068(PC),A0
7	Don't forget to update also the markers in memory $C2 and $C6 Data that will be retrieved before the next TrackLoad.	70054 70058	MOVE.L MOVE.L	(A0)+,C2.S (A0)+,C6.S
8	Execution of the code at the decomp. address.	7005C	JMP	2B0.S

Start of 'Psygnosis Animation'

Part 7 Test of the TrackLoader #1 and extract of the 'Psygnosis animation'

Reboot your Amiga on our backup floppy disk previously made under **X-Copy.**
Very quickly we reach our green wallpaper, **swap the floppy disk by the original Disk1 and enter the AR.**

The original code is restored, **Type in : A 70000**

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

A quick reminder: Normally the next **TrackLoad** will be done in :

Adr. Source disk	:	*$48810*
Size	:	*$24284*
DSKPTH	:	*$6A000*
Dest. Memory	:	*$45610*

A *BreakPoint* is taken just before the decompression/decryption phase. **Type in : BS 7004C**
The Destination Memory Area is filled with a small margin : ($45610-$20) → ($45610+$24284+$20) ***$455F0 → $698B4***
Type in : O "PaTtErN", 455F0 698B4

You can also check if you want to, as shown in the picture below.

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

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

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

And we go back to the code with the command **X**
After a trackload, our *BreakPoint* is reached, we automatically enter the AR.

We check the memory area again, **Type in : M 455F0** then **M 698B4-40**

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

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

Data have been loaded from ***$45610*** up to ***$69894*** as expected. This validates our analysis of the **TrackLoader's** operation

We are left with the decompression/decryption part :
The Destination Memory Area is filled with a small margin : ($2B0-$20) → ($45610) ***$290 → $45610***
Type in: O "PaTtErN", 290 45610

A *BreakPoint* is taken just after the decompression/decryption phase. **Type in : BS 70050** and return to the code with the command **X**
After a few seconds, our **BreakPoint** is reached, the decompression/decryption phase.

We check the decompression memory area, **Type in : M 455F0** then **M 698B4-40**

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

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

Data is now available from $2B0, , but not up to $45610
$2B0+$24284=$24534
Type in : N 290 and scroll down in the code until you see our pattern ‘PaTtErN’

```
.028034 .@@... $.....'....... ...@..  ...'P............... ...........@
.028074 ........................................@@.00... .0..........0..
.0280B4 .''.. ...... . ...................... . .....................
.0280F4 ...@. ..................0...@.@....@........$....@.@.... @...p.
.028134 ..".....@0......@........"....@ ...... ......"....@!. ...... ....
.028174  .....A.'...... p...!.....B.@......A...1.............C...1....
.0281B4 ...............0...............(..........C...(..........a...(....
.0281F4 ......!...$.....0..........$.......  .........$......@@....@..."..
.028234 ..........@..."..........  ..."......!..... ...!.....b.....0...!.
.028274 ....F.........!............... ......'....... ..............1...
.0282B4 .(#...@.........0....F...D.................d...............$....'
.0282F4 ........"........0............................'...............'....
.028334 .........8......................................PaTtErNPaTtErNPa
.028374 TtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaT
.0283B4 tErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTt
.0283F4 ErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtE
.028434 rNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtEr
.028474 NPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErN
.0284B4 PaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNP
.0284F4 aTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPa
.028534 TtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaT
.028574 tErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTt
.0285B4 ErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtE
.0285F4 rNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtEr
.028634 NPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErNPaTtErN
```

More precisely with the command **M 28334**

```
m 28334
:028334 09 8E 00 01 03 80 87 00 00 38 8C 00 00 09 CE 00 .........8......
:028344 00 04 A6 00 00 04 A6 00 00 04 A6 00 00 02 A6 00 ................
:028354 00 02 A6 00 00 01 A6 00 00 01 A6 00 00 01 81 00 ................
:028364 50 61 54 74 45 72 4E 50 61 54 74 45 72 4E 50 61 PaTtErNPaTtErNPa
:028374 54 74 45 72 4E 50 61 54 74 45 72 4E 50 61 54 74 TtErNPaTtErNPaTt
:028384 45 72 4E 50 61 54 74 45 72 4E 50 61 54 74 45 72 ErNPaTtErNPaTtEr
```

So we have ‘an end’ of the decompressed/decrypted data in $28364 starting at $2B0 (so $28364-$2B0=$280B4)
We start from a **trackLoaded** data size of : $24284 to obtain a decompressed size of $280B4
Either $3E30 (15920 Octets, approx 15,5 Ko) more, (This gives us a gain of about 10% of the file) which is really not bad.
We'll see later if we can do better with another compression routine. The goal is to save as much space as possible.

We take this opportunity to save this data which corresponds to the 1st **Psygnosis** animation.
#FORMAT, allows you to format a floppy disk in AmigaDOS format.
#SM, alias SaveMemory. Allows you to save a memory area to a file.

Remove the original disk from the floppy drive and insert a blank disk.
Type in: FORMAT SOTB2-FileD1 and we save it all **SM PSYGNOSIS, 2B0 28364**

```
format SOTB2-FileD1
Ready to format disk in drive DF0: (y/n)?
y
Disk ok

sm PSYGNOSIS, 2B0 28364
Disk ok


dir
Directory of (SOTB2-FileD1)
        164020  PSYGNOSIS
1414 blocks free, 19.6 % of disk used
Disk ok
```

Part 8 Analyse of TrackLoader #2

Now, the only thing left to do is to take a look at the code in *$2b0*, and obviously we're still dealing with a **TrackLoader.**
Type in: D 2B0

```
d 2B0
~0002B0 ORI.B   #0,D0
~0002B4 ORI.B   #0,D0
~0002B8 ORI.B   #26,D0
~0002BC BSR     000003B2
~0002C0 JMP     00001072

d 3B2
~0003B2 MOVE.B  00BFE001,D0
~0003B8 BTST    #4,D0
~0003BC BEQ     000003C6
~0003BE BSR     0000035E
~0003C0 BSR     000003D8
~0003C4 BRA     000003B2
;========================================

d 1072
~001072 MOVE.W  #2700,SR
~001076 MOVEA.L #1764,A7
~00107C MOVE.B  #7F,00BFED01
~001084 MOVE.B  #7F,00BFDD00
~00108C MOVE.W  #0,00DFF100
```

Let's go into more detail and here's what we can remember.

#2B0

```
002B0    ORI.B    #0,D0
002B4    ORI.B    #0,D0
002B8    ORI.B    #26,D0                       ; 26 ? Marker of something ?
002BC    BSR      3B2                          ; GoSub ➔ #Return_T00
002C0    JMP      1072                         ; Goto ➔ #Base_Pre-Anim
```

#Base_Pre-Anim

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

1094     LEA      14E4,A0                      ; A0=$14E4
109A     MOVE.W   #1F,D7                       ; D7=1F
109E     MOVE.L   #0,(A0)+                     ; ➔ Clear (A0), then A0=A0+4
10A4     DBF      D7,109E                      ; ← D7=D7-1, as long as D7 is different from -1, we loop
                                               ; That's $20 times, so $20*4=$80 Bytes erased from 14E4  (14E4 -> 15E4)

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

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

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

#ERASE_A0

```
00EFC    MOVE.W   #270F,D7                     ; D7=270F
00F00    MOVE.L   #0,(A0)+                     ; ➔ Erase (A0), then A0=A0+4
00F06    DBF      D7,F00                       ; ← D7=D7-1, as long as D7 is different from -1, we loop
00F0A    RTS                                   ;
```

#Base_Anim

```
00598   MOVE.W   #4,D88                  ; copy 0004 to adress D88
005A0   MOVE.W   #96,D86                 ; copy 0096 to adress D86
005A8   MOVE.W   #FFFF,D8A               ; copy FFFF to adress D8A
005B0   CLR.W    1DD2                    ; Nettoie le World en 1DD2
005B6   BSR      CE6                     ; GoSub $CE6
005BA   JSR      1D0A                    ; GoTo $1D0A
005C0   MOVE.W   #FFFF,ABE               ; copy FFFF to adress $ABE
005C8   MOVE.W   #1,D0                   ; D0=0001
005CC   BSR      F44                     ; GoTo $F44
005D0   BTST     #7,BFE001               ; Fire Button pressed?
005D8   BNE      5E4                     ; - If not then GoTo ➔ #Start_Animation
005DA   MOVE.B   #1,176B                 ; - If Yes then, copy 01 to adress $176B
005E2   BRA      638                     ; Then... GoTo ➔ #Last_Load_Disk1 - InsertDisk2
```

#Start_Animation

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

005EE   BTST     #7,BFE001               ; Fire Button pressed?
005F6   BEQ      638                     ; GoTo ➔ #Last_Load_Disk1 - InsertDisk2

#Loading_Animation Part #1               // End Animation Psyngosis, Start Loading Anim Part #1
005F8   MOVE.L   1DCA,578.S              ; Update of table #1, DSKPTH in relation to the memory content in $1DCA

00600   LEA      16B14,A0                ; A0=16B14,  Adr_memory_destination
00606   LEA      580.S,A1                ; A1=$580    Adress of the Table : Adress_Start_raw
0060A   BSR      44A                     ; GoSub ➔ #Trackloader_Start

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

#Loading Animation Part #2               // End Anim Part #1, Start Loading Anim Part #2
00618   MOVE.L   1DCA,578.S              ; Update of table #1, DSKPTH in relation to the memory content in $1DCA

00620   LEA      16B14,A0                ; A0=16B14,  Adr_memory_destination
00626   LEA      588.S,A1                ; A1=$588    Adress of the Table : Adress_Start_raw
0062A   BSR      44A                     ; GoSub ➔ #Trackloader_Start

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

#Last_Load_Disk1 - InsertDisk2           // End Anim Part #2, Start Loading Last_Load
00638   LEA      256.S,A7                ; Change address of the stack to $256

0063C   MOVE.L   #50000,578.S            ; End of animations. Before next, update of table #02
00644   MOVE.L   C2.S,590.S              ; Copy of the Word previously put in C2 into current table
0064A   MOVE.L   C6.S,594.S              ; Copy of the Word previously put in C6 into current table

00650   LEA      40000,A0                ; A0=memory Destination for the TrackLoader
00656   LEA      590(PC),A               ; A1=$590, adr. of the table that will contain what we have previously put in $C2
                                         ; values that was in $70068 in the previous Trackloader

0065A   BSR      44A                     ; GoSub ➔ #Trackloader_Start
                                         ; A0=Adr_dest_memory           A1=Table pointer (Adress_Start_raw)
                                           $590=(Disk Pos.=0006CA94 Size=00009B78)
                                           Data loaded : $40000 ➔ $49B78

#copy_2C6_to_4F000                       // Last Load, copy code to 4F000, Recopy
0065E   LEA      2C6(PC),A0              ; A0=Source address for the copy loop=2C6
00662   LEA      4F000,A1                ; A1=Dest. Address=4F000
00668   MOVE.W   #6,D0                   ; D0=6, counter

0066C   MOVE.L   (A0)+,(A1)+             ; ➔ copy loop of (A0) to (A1)
0066E   DBF      D0,66C                  ; ← D0=D0-1, as long as D0 is different from -1, we loop
                                         ; copy of 7 long word, so 28 Bytes

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

#End of the copy loop, we waiting the end of the counter to display the message 'INSERT DISK 2'

```
00676   CLR.B    176A           ; ➔ We clear the Byte in $176A
0067C   TST.B    176A           ;        ➔ Test the Byte in $176A
00682   BEQ      67C            ;        ← If = 0, we loop, it must be an end pointer of something…
                                         In this case, timer of a few seconds.

00684   BTST     #7,BFE001      ; CIA-A / PRA, test of the bit7, Button Fire1 pressed ?
0068C   BEQ      692            ; Yes ?, We jump just after this loop.
0068E   DBF      D1,676         ; ← No, not pressed ? We loop (we wait a few seconds before passing this test)

00692   TST.B    176B           ; Again the test of $176B
                                ; obviously there is something going on in the background at this address…

00698   BNE      69E            ; If not = 0, we are going to $69E
0069A   BSR      988            ; GoSub #$988                        // Off topic of the hack, no interest

0069E   MOVE.W   #7FFF,DFF09A   ; DFF09A=Conf. INTENA
006A6   MOVE.W   #7FFF,DFF096   ; DFF096=Conf. DMACON
006AE   JMP      4F000          ; Jump to 4F000 ====> Display of the message 'INSERT DISK 2'
```

 # Table_#02 Information from the table retrieved with the **BS 494** command before each loading.

```
00578    00 05 00 00     ← Conf DSKPTH 'Last_Load', DMA DATA=50000
0057C    00 00           ← Marker to Trackload or not
0057E    00 00           ← Current Track Pointer

00580    00 00 18 9C     ← Adress_Start_raw Anim_Part_#01,          (Minimum $189C, track size)
                           Note : we can't start in Cylinder 0, we have to start in Cylinder 1 minimum

00584    00 04 6F 74     ← Length_To_Read    Anim_Part_#01

00588    00 08 5D 04     ← Adress_Start_raw  Anim_Part_#02

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

00590    00 04 88 10     ← Adress_Start_raw  Last_Load_D1 →@ end of anim = 00 06 CA 94 (previously copied from the old table)

00594    00 00 BB 80     ← Length_To_Read    Last_Load_D1 →@ end of anim  = 00 00 9B 78 (previously copied from the old table)
```

#Copy_Post_LastLoad // code executed after **#Loading Last Data of Disk1 - InsertDisk2**

```
002C6    LEA      40000,A0        ; A0=$40000              // Source Address
002CC    LEA      256.S,A1        ; A1=$256                // Destination Address
002D0    MOVE.L   C6.S,D0         ; D0=C6.S
002D4    LSR.L    #2,D0           ; Shift of 2 bits to the right of D0, what gives us D0=26DE

002D6    MOVE.L   (A0)+,(A1)+     ; → Copy (A0) to (A1) then, A0=A0+4 and A1=A1+4
                                  ; As we copy in LongWord, this gives us (26DE*4)+4=$9B7C of copied data.
                                  ; $256+$9B7C=$9DD2         // Area copied (destination) = $256 → $9DD2

002D8    DBF      D0,2D6          ; ← D0=D0-1, as long as D0 is different from -1, we loop
002DC    JMP      256.S           ; Jump to $256
```

#Trackloader_Init

```
0002E0   MOVE.W   #10,DFF096      ; Conf DMACON
0002E8   MOVE.W   #2,DFF09C       ; Conf INTREQ
0002F0   MOVE.W   #7FFF,DFF09E    ; Conf ADKCON
0002F8   MOVE.W   #8100,DFF09E    ; Conf ADKCON
000300   ORI.B    #78,BFD100      ; 'OR' binary betweeb 0111 1000 and $BFD100, so DF0 to DF3 = select and Motor ON
000308   BCLR     #7,BFD100       ; CIA-B / PRB, bit7 set to 0, motor On
000310   BCLR     #3,BFD100       ;

000318   MOVE.W   #BB8,D6         ; → D6=BB8
00031C   DBF      D6,31C          ; ← D6=D6-1, as long D6 is different from -1, we loop. (we have a 'pause' here)

000320   BTST     #5,BFE001       ; → CIA-A / PRA, Test of bit5, FloppyDisk Ready ?
000328   BNE      320             ; → as long FloppyDisk is not ready, we loop.

00032A   LEA      57C(PC),A3      ; A3=57C(PC) Address in the table, put 70072 into A3 (table address)
0003EE   MOVE.W   #1,(A3)         ; Copy the word 01 to (A3), so put 00 01 into the table in $57C
000332   RTS                      ; E.T back home
```

#Move inside

```
00334    BCLR     #1,BFD100       ; CIA-B / PRB, bit1 set to 0, DIR= towards the inside.
0033C    BCLR     #0,BFD100       ; CIA-B / PRB, bit0 set to 0, STEP, Heads pre/move
00344    BSET     #0,BFD100       ; CIA-B / PRB, bit0 set to 0, STEP, Head move.
0034C    BSR.W    3D8             ; GoSub → #WAIT
00350    MOVE.L   A0,-(A7)        ; Save A0 in stack
00352    LEA      57E(PC),A0      ; Current Cylinder Pointer put in A0
00356    ADDQ.W   #1,(A0)         ; (A0)=A0+1 Word
                                  ; We moved from a Track to the inside and (A0), $57E is increased by 1
                                  ; $57E so it's a Track pointer.

00358    MOVEA.L (A7)+, A0        ; Restore A0 from the stack
0035A    BRA      3CC             ; GoTo → #Disk Ready? without passing through the modification of A3
```

#Move Outside

```
00035E   BSET     #1,BFD100       ; CIA-B / PRB, bit1 set to 1, DIR= towards the outside
000366   BCLR     #0,BFD100       ; CIA-B / PRB, bit0 set to 0, STEP, Heads pre/move
00036E   BSET     #0,BFD100       ; CIA-B / PRB, bit0 set to 0, STEP, Head move.
000376   BSR      3D8             ; GoSub → #WAIT
00037A   MOVE.L   A0,-(A7)        ; Save A0 in stack
00037C   LEA      57E(PC),A0      ; A0=57E(PC), ...again, must be in table.
000380   SUBQ.W   #1,(A0)         ; (A0)=A0-1 Word, We move from a Track to the outside and (A0), $57E is decreased by 1
                                  ; $57E , so that confirm it's a Track Pointer.

000382   MOVEA.L  (A7)+,A0        ; Restore A0 from stack
000384   BRA      3CC             ; GoTo → #Disk Ready? Without passing trough the modification of A3
```

#Position Reach?

```
00388   CMP.W    57E(PC),D0       ; Check $57E (Track Pointer) with D0
0038C   BEQ      3A0              ; If equal then GoTo → #Forward_Backward
0038E   BGT      3A4              ; if N=0, (greater than), then GoTo → #Go_To_Track
00390   MOVE.W   57E(PC),D6       ; D6=$57E=Track pointer
00394   SUB.W    D0,D6            ; D6=D6-D0
00396   SUBQ.W   #1,D6            ; D6=D6-1
00398   BSR      35E              ; → GoSub #Move Outside
0039A   DBF      D6,398           ; ← D6=D6-1, as long D6 is different from -1, we loop.
0039E   RTS                       ; E.T back home
```

#Forward_Backward

```
003A0   BSR      334              ; GoSub → #Move Inside
003A2   BRA      35E              ; GoTo → #Move Outside
```

#GoTo_Position

```
003A4   SUB.W    57E(PC),D0       ; D0=D0-( content of $57E), aka, current position
003A8   SUBQ.W   #1,D0            ; D0=D0-1
003AA   BSR      334              ; → GoSub → #Move Inside
003AC   DBF      D0,3AA           ; ← D0=D0-1, as long D0 is different from -1, we loop.
003B0   RTS                       ; E.T back home
```

#Return_T00

```
0003B2  MOVE.B   BFE001,D0        ; → PRA CIA-A dans D0
0003B8  BTST     #4,D0            ;     Test Bit 4 of D0, Namely: Disk on T00?
0003BC  BEQ      3C6              ;     Yes ? GoTo → #Disk Ready?
0003BE  BSR      35E              ;     GoSub → #Move Outside
0003C0  BSR      3D8              ;     GoSub → #WAIT
0003C4  BRA      3B2              ; ← we loop on #Return_T00
```

#Disk Ready?

```
0003C6  LEA      57E(PC),A3       ; A3=57E(PC), again a table ?
0003CA  CLR.W    (A3)             ; Clear the word at (A3)
0003CC  BTST     #5,BFE001        ; → Test Bit 8 of PRA, Namely : Disk_Ready ?
0003D4  BNE      3CC              ; ← No ? We loop until it does.
0003D6  RTS                       ; E.T back home
```

#WAIT

```
003D8   MOVE.W   D7,-(A7)         ; Save D7 in stack
003DA   MOVE.W   #1388,D7         ; D7=1388
003DE   DBF      D7,3DE           ; ↔ D7=D7-1, as long D7 is different from -1, we loop.
003E2   MOVE.W   (A7)+,D7         ; Restore D7 from stack
003E4   RTS                       ; E.T back home
```

#DF0__SIDE_DOWN__MOTOR_OFF__DIR_EXT

```
003E6   MOVE.B   #FD,BFD100       ; Conf CIA-B / PRB
                                  ; MOTOR OFF, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 UNSELECT, SIDE DOWN, DIR EXT., STEP TRACK=1
003EE   NOP                       ;
003F0   NOP                       ;
003F2   NOP                       ;
003F4   NOP                       ;
003F6   MOVE.B   #F5,BFD100       ; Conf CIA-B / PRB
                                  ; MOTOR OFF, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 SELECT, SIDE DOWN, DIR EXT., STEP TRACK=1
003FE   MOVE.W   #B000,D0         ; D0=B000
00402   DBF      D0,402           ; ↔ D0=D0-1, as long D0 is different from -1, we loop. (we have a 'pause' here)
00406   LEA      57C(PC),A3       ; A3=57C, Marker for Trackload or not
0040A   CLR.W    (A3)             ; We clear the Marker
0040C   RTS                       ; E.T back home
```

#DF0__SIDE_DOWN__MOTOR_ON__DIR_EXT

```
0040E   MOVE.B   #7D,BFD100       ; Conf CIA-B / PRB
                                  ; MOTOR ON, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 UNSELECT, SIDE DOWN, DIR EXT., STEP TRACK=1
00416   NOP                       ;
00418   NOP                       ;
0041A   MOVE.B   #75,BFD100       ; Conf CIA-B / PRB
                                  ; MOTOR ON, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 SELECT, SIDE DOWN, DIR EXT., STEP TRACK=1
00422   MOVE.W   #B000,D7         ; D7=B000, 'pause' counter
00426   DBF      D7,426           ; ↔ D7=D7-1, as long D7 is different from -1, we loop. (we have a 'pause' here)
0042A   RTS                       ; E.T back home
```

#DF0__SIDE_UP__MOTOR_ON__DIR_EXT

```
0042C    MOVE.B   #79,BFD100        ; Conf CIA-B / PRB
                                    ; MOTOR ON, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 UNSELECT, SIDE UP, DIR EXT., STEP TRACK=1
00434    NOP                        ;
00436    NOP                        ;
00438    MOVE.B   #71,BFD100        ; Conf CIA-B / PRB
                                    ; MOTOR ON, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 SELECT, SIDE UP, DIR EXT., STEP TRACK=1
00440    MOVE.W   #B000,D7          ; D7=B000, 'pause' counter
00444    DBF      D7,444            ; ←→ D7=D7-1, as long D7 is different from -1, we loop. (we have a 'pause' here)
00448    RTS                        ; E.T back home
```

#Trackloader_Start A0=Adr_dest_memory A1=Table pointer(Adress_Start_raw)

```
0044A    MOVE.L   A0,-(A7)          ; Save A0 in stack
0044C    BSR      454               ; GoSub → #Read_Table_and_Start_Trackload_Or_Not
00450    MOVEA.L  (A7)+,A0          ; Restore A0 from stack
00452    RTS                        ; E.T back home
```

#Read_Table_and_Start_Trackload_Or_Not

```
00454    LEA      57C(PC),A3        ; A3=57C (Adress in the table)          // Marker ready to track or not
00458    TST.W    (A3)              ; Test A3 content with zero
0045A    BNE      460               ; If not equal then GoTo → #Side_Select
0045C    BSR      2E0               ; GoSub → #Trackloader_Init

#Side_Select
00460    MOVE.L   (A1)+,D0          ; D0=A1 then A1=A1+4
00462    CMP.L    #84468,D0         ; D0-84468 and modifies the Flags accordingly.
                                    ; Test to know on which side we will use.
                                    ; Same value as the 1er TrackLoader, namely 84468

                                    ; Depending on the result, we will go to the Side_UP or Side_Down routine
00468    BGE      46E               ; if 'greater than' GoTo → #Recover_Info_to_Trackloader_1 and #DF0__SIDE_UP__MOTOR_ON__DIR_EXT

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

0046C    BRA      470               ; And afterwards GoTo → #Recover_Info_to_Trackloader_2
```

#Recover_Info_to_Trackloader_1

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

#Recover_Info_to_Trackloader_2
00470    MOVE.L   D0,D3             ; In this step D0=(LongWord of $580), value in the table=$189C
                                    ; namely 'Adress_Start_raw'
                                    ; D3=D0, so D3=$189C, allows to know which Track is concerned.

00472    DIVU     #189C,D0          ; D0/$189C and puts the result in D0
00476    CMP.W    #56,D0            ; Compare D0 with the value $56
0047A    BLT      486               ; If 'less than', 'branch' in 846
0047C    SUBI.L   #84468,D3         ; Otherwise, D3=D3-$84468
00482    SUBI.W   #56,D0            ; D0=D0-$56
00486    MOVEQ    #0,D2             ; D2=00
00488    MOVE.W   D0,D2             ; D2=D0
0048A    BSR      $388              ; GoSub → #Position Reach?
0048E    MULU     #189C,D2          ; At this moment, D2=position reached = Heads position=Track in progress
                                    ; Multiplied by the size of a Track,
                                    ; it becomes the size in position_raw where we are.

00492    SUB.L    D2,D3             ; In this step D3=(LongWord of $580), value in the table=$189C
                                      or if D0 Start_Track was larger than $56
                                      we subtract the position_raw_reach of D3

                                      In fact, according to Adress_Start_raw in the table, we will use a specific side.
                                      And so this operation performed on D3 allows
                                      to reposition Adress_Start_raw at the beginning and change the side at the same time.
                                      D3=(value in table $580)-(position reach)

00494    MOVE.L   (A1),D4           ; Copy the LongWord content in A1 into D4
                                    ; 1er call (loading Anim_Part_#01, A1=584=(Length_To_Read) = $46F74
                                    ; 2em call (loading Anim_Part_#02, A1=58C=(Length_To_Read) = $48DDE
                                    ; 3em call (loading Last_Load,     A1=594=(Length_To_Read) = $9B78

00496    LSR.L    #2,D4             ; Shift of 2 bits to the right of D4, is equivalent to dividing D4 by 4
00498    SUBQ.L   #1,D4             ; D4=D4-1

0049A    BSR      4A2               ; → GoSub → #Trackload_Base
0049C    BSR      334               ;     GoSub → #Move Inside
004A0    BRA      49A               ; ← Branch in 49A, we loop to #Trackload_Base
```

#Trackload_Base

```
004A2   MOVEQ    #0F,D2          ; D2=F
004A4   MOVE.L   #18B8,D0        ; → D0=$18B8
004AA   MOVEA.L  578(PC),A6      ; A6=Address in table=000578
004AE   MOVE.W   #2,DFF09C       ; Conf INTREQ, Disk Block Finished Interupt
004B6   MOVE.L   A6,DFF020       ; Conf DSKPTH    (conf which comes from table)
004BC   MOVE.W   #8210,DFF096    ; Conf DMACON, DSKEN enable, ALL DMA enable
004C4   MOVE.W   #4489,DFF07E    ; Conf DSKSYNC = $4489 (standard AmigaDos)
004CC   MOVE.W   #7F00,DFF09E    ; Conf ADKCON specific
004D4   MOVE.W   #B500,DFF09E    ; One more time to start the transfer.
                                   FAST=2us, WORDSYNC=active, MFM precomp, recomp=140ns

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

#Test CIA Ready
004E4   MOVE.B   BFDD00,D7       ; D7=BFDD00
004EA   MOVE.B   BFDD00,D7       ;        → D7=ICR register CIA-B
004F0   BTST     #4,D7           ;          Test bit4 (FLAG) of ICR
004F4   BEQ      4EA             ;        ← Interruption generated ? We loop

004F6   ADDI.W   #8001,D0        ; ADD signed on D0 (which is $18B8 see a few lines above) with $8001
                                   Which gives us: $98B9 and flag C=0

004FA   MOVE.W   D0,DFF024       ; CONF DSKLEN, Disk DMA Enable, Length of DMA data=$18B9
00500   MOVE.W   D0,DFF024       ; One more time to trigger the reading

00506   MOVE.W   DFF01E,D0       ;        → D0=INTREQR
0050C   BTST     #1,D0           ;          Test Bit1 of INTREQR, Level 1 Disk Block Finished Interupt
00510   BEQ      506             ;        ← Test ?, We loop

00512   MOVE.L   (A6)+,D0        ; Value of DSKPTH in D0, (A6=DFF024) then A6=A6+4
00514   MOVE.L   (A6)+,D7        ; Value of DSKLEN in D7, (A6=DFF028) then A6=A6+4
00516   ADD.L    D0,D0           ; D0=D0+D0
00518   ANDI.L   #AAAAAAAA,D0    ; Post_ Treatment MFM odd bit in D0
0051E   ANDI.L   #55555555,D7    ; Post_ Treatment MFM even bit in D7
00524   OR.L     D7,D0           ; MFM processing
00526   CMP.L    #42535432,D0    ; Compare D0 with 42535432
                                   42 53 54 32 in Ascii = BST2
                                   If the signature is not found, we recheck and finally,
                                   If still not good, will go on the red screen routine below.

0052C   BEQ      53C             ; Yes ? Signature found, GoTo → #Processing_MFM_BASE
                                   We start the processing of the Real Data and the Trackload.

0052E   DBF      D2,4A4          ; ← D2=D2-1, as long D2 is different from -1, we loop at 4A4
                                   Let's go for a ride again.

00532   MOVE.W   #F00,DFF180     ; → Red Background
0053A   BRA      532             ; ← DeadLoop 'Red Background'
```

#Processing_MFM_BASE

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

00542   TST.W    D3              ; Test D3 with zero                // D3= Delta previously calculated.
00544   BEQ      550             ; If equal thens GoTo → #Check_Processing_MFM
00546   ADD.W    D3, D3          ; Otherwise D3=D3+D3               // We do x2 on D3

                                   Reminder, D3=((Table value Address_Start_raw)-(value position reach))

00548   ADDA.L   D3,A6           ; A6=A6+D3                                   // A6=DSKPTH+decoding in progress
0054A   LSR.W    #3,D3           ; Shifts 3 Bits to the left of LongWord D3  // Equals to dividing D3 by 8
                                   So delta between position reach and (value in Length_To_Read)/8)

0054C   SUB.W    D3,D2           ; D2(current_raw_position)-delta calculated above
0054E   MOVEQ    #0,D3           ; D3=0                                       // D3 is reset to Zero

#Check_Processing_MFM
00550   CMP.L    D2,D4           ; Compare D4-D2
00552   BGT      55A             ; If result greater than GoTo → #MFM bit even processing
00554   MOVE.W   D4,D2           ; D2=D4
00556   ADDQ.W   #4,A7           ; A7=A7+4
00558   BRA      55E             ; GoTo → #Start_Decoding
```

#MFM bit even processing

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

#Start_Decoding
0055E   MOVE.L   #55555555,D5    ; D5=55555555, MFM mask bit even
00564   MOVE.L   (A6)+,D0        ; → Copy (A6) into D0 then A6=A6+4
00566   MOVE.L   (A6)+,D7        ;     Copy the contents of A6 into D7 then A6=A6+4
00568   AND.L    D5,D0           ;     Processing MFM
0056A   AND.L    D5,D7           ;     Processing MFM
0056C   ADD.L    D0,D0           ;     Processing MFM
0056E   OR.L     D7,D0           ;     Processing MFM
00570   MOVE.L   D0,(A0)+        ;     Copy D0 to the address pointed by A0 then A0=A0+4
00572   DBF      D2,564          ; ← D2=D2-1, as long D2 is different from -1, we loop
00576   RTS                      ; E.T  back home
```

#Decrypt/Decomp_&_execution

```
006B4    MOVE.L   (A0)+,800          ; Copy the LongWord from the address pointed by A0 to address $800
                                       $800=$46F74     and $5DA88 decomp anim_part_#01

006BA    ADDI.L   #16B14,800         ; Add $16B14 to this one at $800
006C4    MOVE.L   A0,7F8             ; Copy A0 at $7F8
006CA    MOVE.L   #93C,7FC           ; Put the LongWord 93C to address $7FC
006D4    CLR.W    AAE                ; Clear the word at $AAE
006DA    CLR.W    AAC                ; Clear the word at $AAC
006E0    CLR.W    AB0                ; Clear the word at $AB0
006E6    CLR.W    AB2                ; Clear the word at $AB2
006EC    CLR.W    AB4                ; Clear the word at $AB4
006F2    CLR.W    AB8                ; Clear the word at $AB8
006F8    MOVE.W   #3,AB6             ; Put the word $0003 to address $AB6

00700    BTST     #6,BFE001          ; Fire Button pressed ?
00708    BNE      71A                ; No ? GoTo → $71A

0070A    MOVE.W   #FFF,DFF180        ; White background when we press the mouse button.
00712    MOVE.B   #1,1ABE            ; Put the Byte 01 into $1ABE

0071A    TST.B    1529               ; Compare 0 with the content of the address $1529
00720    BEQ      730                ; If equal then GoTo → $730
00722    CLR.B    1529               ; Clear the Byte in $1529

00728    TST.B    1529               ; → Compare 0 with the content of the address $1529
0072E    BEQ      728                ; ← If equal then GoTo → $728

00730    TST.W    AAC                ; Compare 0 with the content of the address $AAC
00736    BNE      700                ; If different then GoTo → $700

00738    TST.W    AB8                ; Compare 0 with the content of the address $AB8
0073E    BNE      752                ; If different then GoTo → $752

00740    MOVE.W   AB6,AB8            ; Copy the word in $0AB6 to address $AB8
0074A    MOVE.L   7FC,A1             ; A1=$7FC
00750    JSR      (A1)               ; JSR (A1), jumps to the address pointed to in A1 (defined earlier in the code)

00752    TST.W    AB2                ; Compare 0 with the content of the address $AB2
00758    BEQ      772                ; If equal then GoTo → $772

0075A    TST.W    AB0                ; Compare 0 with the content of the address $AB0
00760    BNE      772                ; If different then GoTo → $772

00762    MOVE.W   AA8,D0             ; Put the Word $0AA8 into D0
00768    BSR      F44                ; GoSub → $F44
0076C    CLR.W    AB2                ; Clear the Word in $AB2

00772    TST.W    AAE                ; Compare 0 with the content of the address $AAE
00778    BPL      786                ;

0077A    MOVE.L   7FC, D0            ; Put the LongWord $7FC into D0
00780    CMP.L    93C, D0            ; Compare the LongWord $93C with D0
00786    BNE      700                ; If different then GoTo → $700

0078A    CLR.W    AAE                ; Clear the Word in $AAE
00790    MOVE.L   7F8,A6             ; Put the LongWord 7F8 into A6
00796    MOVE.W   (A6)+,D0           ; Copy (A6) into D0 then A6=A6+2
00798    BMI      7AA                ;
0079A    MOVEA.L  10(PC,D0.W),A1     ;
0079E    JSR      (A1)               ; A1=$16B18
007A0    MOVE.L   A6,7F8             ; Copy A6 to address $7F8
007A6    BRA      700                ; Branch to $700

007AA    RTS                         ;
```

We stop here our disassembling because it becomes too long, we have about all that we need.
Theoretically, the decompression routine ranges from $6B4 → $FBC   so 2312 bytes, but whatever...

Note : We can add an option to skip the animations.
Instead of having a **white background**, we can change to jump directly to $638 to make the last load.

Therefore, replace

```
0070A    MOVE.W   #FFF,DFF180        ; White background when we press the mouse button
```

by

```
0070A    BRA      638                ; GoTo → #Last_Load_Disk1 - InsertDisk2
```

Here is a table of the main *Trackloader* subroutines.
The objective is to see if we can overwrite these without having to make too many changes in the original code.

Memory Adr	Sub Routine	Called by	Info	In the Potential Area ?	Area Potential
2E0-332	#Trackloader_Init	45C	Not required	**inside**	
334-35A	#Move Inside	3A0 3AA 49C	Not required	**inside**	
35E-384	#Move Outside.	398 3A2 3B2	Not required	**inside**	
388-39E	#Position Reach?	48A	Not required	**inside**	
3A0-3A2	#Forward_Backward	38C	Not required	**inside**	
3A4-3B0	#GoTo_Position	38E	Not required	**inside**	
3B2-3C4	#Return_T00	**2BC** 3C4	Not required	**To PATCH**	
3C6-3D6	#Disk Ready?	3BC	Not required	**inside**	
3D8-3E4	#WAIT	34C 376 3C0	Not required	**inside**	
3E6-40C	#DF0__SIDE_DOWN__MOTOR_OFF__DIR_EXT	**NONE**	Not required	**inside**	
40E-42A	#DF0__SIDE_DOWN__MOTOR_ON__DIR_EXT	46A	Not required	**inside**	
42C-448	#DF0__SIDE_UP__MOTOR_ON__DIR_EXT	46E	Not required	**inside**	
44A-452	#Trackloader_Start	60A 62A 65A	No longer necessary	**inside**	
454-46C	#Read_Table_and_Start_Trackload_Or_Not	44C	Not required	**inside**	
46E-470	#Recover_Info_to_Trackloader_1	468	Not required	**inside**	
470-4A0	#Recover_Info_to_Trackloader_2	46C	Not required	**inside**	
4A2-4DC	#Trackload_Base	49A	Not required	**inside**	
4E4-53A	#Test CIA Ready	**NONE**	Not required	**inside**	
53C-54E	#Processing_MFM_BASE	52C	Not required	**inside**	
550-558	#Check_Processing_MFM	544	Not required	**inside**	
55A-55E	#MFM bit even processing	552	Not required	**inside**	
55E-576	#Start_Decoding	558	Not required	**inside**	

So we can easily 'overwrite' the original code with our own *trackloader*.

Part 9 Analyse of the last loaded code : Last_Load and TrackLoader #3

If a BreakPoint is set before the last trackload, i.e. in **$656**
Fill the destination area with any pattern with a small margin: **O "PaTtErN", 3FFF0 49B88**
And by placing a BreakPoint just after the Trackload, i.e. in **$65E**, then we return to the code.
Once the trackload is finished and our breakPoint *$65E* rechead, we can clearly see that the trackloading DATA is pure code, no compression or encryption phase and that the filled area is the expected one : *$40000 → $49B78*

Then a few lines of code later, we execute this copied code: **006AE JMP 4F000**
This is equivalent to analyzing the previously trackloaded code in *$40000* which is now in place in *$4F000*

Execution of the routine **#Recopy_2C6_to_4F000** which happens right after copy the routine **#Copy_Post_LastLoad** in *$4F000*

```
#Recopy_Copy_Post_LastLoad_to_4F000
4F000   LEA      40000,A0          ; A0=$40000                // Source Address
4F006   LEA      256.S,A1          ; A1=$256                  // Destination Address
4F00A   MOVE.L   C6.S,D0           ; D0= Counter for the following DBF loop
                                   ; We now understand the importance of the value of $C6

4F00E   LSR.L    #2,D0             ; Shift 2 bits to the right of D0, what gives us D0=$26DE

4F010   MOVE.L   (A0)+,(A1)+       ; → Copy (A0) to (A1) then A0=A0+4 and A1=A1+4
                                   ;    Copy in LongWord, what gives us (26DE*4)+4=$9B7C of copied data.
                                   ;    $256+$9B7C=$9DD2      // Copied area (destination) = $256 → $9DD2
                                   ;    Reminder : Size of the previously trackloaded code in $40000 = $BB80
                                   ;    So the entire trackload code is moved to $256

4F012   DBF      D0,4F010          ; ← D0=D0-1, as long D0 is different from -1, we loop
4F016   JMP      256.S             ; Jump to $256
```

After examining the code, here is what we can learn. (namely, another **TrackLoader**, almost identical)
We will go back to the main menu of the game AND the loading of the 1st level
All this is performed with the commands **BS** and **ST**

Type in: D 256

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

#WaitSyncV

```
25A     MOVEQ    #0,D1                         ; D1=00
25C     MOVE.B   BFE801,D1                     ; D1=todlo (vsync)
262     BSR      26A                           ; → GoSub #Working_on_BE2_Mark
264     DBF      D1,262                        ; ← We loop as long as necessary.
268     RTS                                    ; E.T back home
```

#Working_on_BE2_Mark

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

#Working_on_BE2_Mark_#2

```
00286   BTST     #1,D0                         ; test Bit1 of D0 with zero
0028A   BNE      28E                           ; If different then GoTo → #Working_on_BE2_Mark_#4

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

#Working_on_BE2_Mark_#4
0028E   MOVE.L   D0,BE2                        ; Copy the LongWord D0 to address $BE2
00294   RTS                                    ; E.T back home
```

#Base_LastLoad

```
1DE0    MOVE.B   C0.S,D0                  ; D0=$C0
1DE4    MOVE.B   D0,BBF.S                 ; copy D0 at $BBF
1DE8    MOVE.W   #2700,SR                 ; Conf. Status register
1DEC    MOVE.W   #7FFF,D0                 ; D0=$7FFF
1DF0    MOVE.W   #7C7F,DFF096             ; Conf DMAcon
1DF8    MOVE.W   D0,DFF09A                ; Conf INTENA
1DFE    MOVE.W   D0,DFF09C                ; Conf INTREQ
1E04    LEA      256.S,A7                 ; A7=256
1E08    BSR      25A                      ; GoSub ➔ #WaitSyncV
1E0C    MOVE.B   #7F,BFDD01               ; ??, programming trick to access bits 0 to 7 of CIA-A ? or bug ?
                                          ; In any case, disabling all the Interupts. From CIA-B
                                          ;
1E14    MOVE.B   #88,BFED01               ; Conf CIAA / ICR
1E1C    MOVE.B   #0,BFEE01                ; Conf CIAA / CRA

1E24    MOVE.L   #99A8,6C.S               ; Copy $99A8 at $6C       ?? // marker for ?
1E2C    MOVE.L   #9A70,68.S               ; Copy $9A70 at $68       ?? // marker for ?

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

1E48    JSR      19C4.S                   ; GoSub ➔ #Trackloader_2_Init
1E4C    JSR      1A96.S                   ; GoSub ➔ #Return_T00
1E50    JSR      8C30                     ; GoSub ➔ #BASE_INSERT_DISK_ASKED
1E56    BSR      9D3E                     ; GoSub ➔ #Wait_BB6

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

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

#Trackloader_2_Init

```
19C4    MOVE.W   #10,DFF096               ; Conf DMACON
19CC    MOVE.W   #2,DFF09C                ; Conf INTREQ
19D4    MOVE.W   #7FFF,DFF09E             ; Conf ADKCON
19DC    MOVE.W   #8100,DFF09E             ; Conf ADKCON
19E4    ORI.B    #78,BFD100               ; 'Or' binary betwwen 0111 1000 and  $BFD100, so DF0 to DF3 select and Motor ON
19EC    BCLR     #7,BFD100                ; CIA-B / PRB, bit7 set to 0, motor On
19F4    BCLR     #3,BFD100                ; CIA-B, bit3 set to 0, DF0 selected
19FC    MOVE.W   #BB8,D6                  ; D6=BB8
1A00    DBF      D6,1A00                  ; ➔➔ D6=D6-1, as long D6 is different from -1, we loop. (we have a 'pause' here)


1A04    BTST     #5,BFE001                ; ➔ CIA-A / PRA, Test of bit5, FloppyDrive Ready ?
1A0C    BNE      1A04                     ; ← As long as the FD is not ready, we loop to wait for it to be ready

1A0E    LEA      C1E(PC),A3               ; A3=$C1E(PC) Address in table.
1A12    MOVE.W   #1,(A3)                  ; Copy of the word 01 into (A3), so put 00 01 in the table in $C1E
1A16    RTS                               ; E.T back home
```

#Return_T00

```
1A96    MOVE.B   BFE001,D0                : D0=CIA PRA
1A9C    BTST     #4,D0                    ; Test of bit4, TK0, Head on T00 ?
1AA0    BEQ      1AAA                     ; Yes ? GoTo ➔ #Disk Ready?
1AA2    BSR      1A42                     ; No ? GoSub ➔ #Move Outside.
1AA4    BSR      1ABC                     ; GoSub ➔ #WAIT
1AA8    BRA      1A96                     ; GoTo ➔ #Return_T00
```

#WAIT

```
1ABC    MOVE.W   D7,-(A7)                 ; Save D7 in stack
1ABE    MOVE.W   #1388,D7                 ; D7=$1388
1AC2    DBF      D7,1AC2                  ; ←➔ Decrease D7, if D7 is different from -1, we loop
1AC6    MOVE.W   (A7)+, D7                ; Restore D7 from the stack
1AC8    RTS                               ; E.T back home
```

#Disk Ready?

```
1AAA    LEA      C20(PC),A3               ; A3=position in table
1AAE    CLR.W    (A3)                     ; Clear the Word at $C20             // Marker Track in progress

#Disk Ready?
1AB0    BTST     #5,BFE001                ; ➔ Test Bit 8 of PRA, namely Disk_Ready
1AB8    BNE      1AB0                     ; ← No ? GoTo ➔ $1AB0
1ABA    RTS                               ; E.T Back Home
```

#Move Outside.

```
1A42    BSET    #1,BFD100                 ; CIA-B / PRB, bit1 set to 1, DIR= toward the outside
1A4A    BCLR    #0,BFD100                 ; CIA-B / PRB, bit0 set to 0, STEP, Head pre/move
1A52    BSET    #0,BFD100                 ; CIA-B / PRB, bit0 set to 0, STEP, Head move
1A5A    BSR     1ABC                      ; GoSub → #WAIT
1A5E    MOVE.L  A0,-(A7)                  ; Sauve A0 on stack
1A60    LEA     C20(PC),A0                ; Pointer 'Current cylinder' set to A0
1A64    SUBQ.W  #1,(A0)                   ; (A0)=A0-1 Word, We have moved one Track to the Outside and (A0)
                                            $C20 is decreasing by 1
                                            $C20 so it is a TRACK pointer, in this case 'TRACK' because the
                                            trackloader works in SIDE.

1A66    MOVE.L  (A7)+,A0                  ; Restore A0 from stack
1A68    BRA     1AB0                      ; Branch in sub-routine GoTo → #Disk Ready?
```

#BASE_INSERT_DISK_ASKED

```
8C30    MOVEQ   #1,D0                     ; D0=01            // Choice of requested disk (00=Disk1, 01=Disk2)

#BASE_INSERT_DISK_ASKED_WITHOUT_FLOPPY_DISK_MARKER
8C32    MOVE.W  D0,A0.S                   ; Copy D0 to the memory address $A0        (marker 'Disk requested')
8C36    BSR     9D3E                      ; GoSub → #Wait_BB6
8C3A    LEA     DFF000,A6                 ; A6=DFF000
8C40    MOVE.W  #1A0,96(A6)               ; Conf of DMACON
8C46    MOVE.L  #8D1E,DFF080              ; Conf of Coperlist / COP1LCH //Conf. of the Copperlist(1) address = $8D1E

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

8C6E    LEA     30000, A0                 ; A0=30000
8C74    MOVE.W  #7CF,D1                   ; D1=7CF
8C78    CLR.L   (A0)+                     ; → Loop for erasing the Memory area (A0)
8C7A    DBF     D1,8C78                   ; ← Decrease D1, if D1 is different from -1, we loop

8C7E    TST.W   A0.S                      ; Test of Word at $A0          // Disk read buffer (signature)
8C82    BEQ     8C8C                      ; If equal to 0 then GoSub → #CHECK_DISK
8C84    LEA     8D2A,A0                   ; A0=8D2A
8C8A    BRA     8C92                      ; GoTo → #CHECK_DISK_A0
```

#Wait_BB6

```
9D3E    CLR.B   BB6.S                     ; Clear the Byte at BB6
9D42    TST.B   BB6.S                     ; → Byte test at BB6
9D46    BEQ     9D42                      ; ← As long as different from zero, we loop (so waiting marker '$BB6')
9D48    RTS                               ; E.T Back Home
```

#CHECK_DISK

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

#CHECK_DISK_A0
8C92     LEA      30E16,A1                   ; A1=30E16
8C98     MOVEQ    #6,D1                      ; D1=06, counter #1
8C9A     MOVEQ    #0D,D2                     ; → D2=0D, counter #2
8C9C     MOVE.W   (A0)+,(A1)+                ;     → Copy loop from (A0) to (A1) defined above
8C9E     DBF      D2,8C9C                    ;     ← as long as D2 is different from -1, we loop
8CA2     ADDA.W   #C,A1                      ;     A1=A1+$C
8CA6     DBF      D1,8C9A                    ; ← as long as D1 is different from -1, we loop

8CAA     LEA      31021,A1                   ; A1=31021
8CB0     LEA      8EB2,A0                    ; A0=8EB2
8CB6     MOVEQ    #6,D1                      ; D1=06, counter #1
8CB8     MOVEQ    #15,D2                     ; → D2=15, counter 2
8CBA     MOVE.B   (A0)+,(A1)+                ;    → Copy loop from (A0) to (A1) defined above
8CBC     DBF      D2,8CBA                    ;    ← as long as D2 is different from -1, we loop
8CC0     ADDA.W   #12,A1                     ;       A1=A1+$C
8CC4     DBF      D1,8CB8                    ; ← as long as D1 is different from -1, we loop

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

#DISK2_OR_DISK1_INSERTED?
8CDC     BSR      1ACA                       ; GoSub → #DF0__SIDE_DOWN__MOTOR_OFF__DIR_EXT
8CE0     TST.B    BB4.S                      ; → Waits for FIRE to be pressed
8CE4     BEQ      8CE0                       ; ← As long as you don't press, you loop
8CE6     BSR      19C4                       ; GoSub → #Trackloader_2_Init
8CEA     BSR      1A96                       ; GoSub → #Return_T00

8CEE     LEA      B0.S,A0                    ; A0=B0                 Memory address destination
8CF2     LEA      8D16,A1                    ; A1=8D16               Table Pos.      // Check_Signature DISK, 1 WORD
8CF8     BSR      1B2E                       ; GoSub → #TrackLoader_2

8CFC     TST.W    A0.S                       ; Test bits of address $A0
                                               $A0 = Disk read buffer (signature)
8D00     BEQ      8D0C                       ; If bits to zero then GoTo → #Check_Signature_DISK_01

#Check_Signature_DISK_02
8D02     CMPI.W   #D0D2,B0.S                 ; Otherwise, compare the word D0D2 (which is the signature of the Original Disk N°2)
8D08     BNE      8CDC                       ; If Disk2 signature not found, loop on #DISK2_OR_DISK1_INSERTED?
8D0A     RTS                                 ; Signature found, we return to the code
```

#Check_Signature_DISK_01

```
8D0C     CMPI.W   #4,B0.S                    ; Compare the word 0004 with the content read in address $B0 (last load)
8D12     BNE.B    8CDC                       ; So different, we go back for a ride
8D14     RTS                                 ; E.T Back Home
```

#DF0__SIDE_DOWN__MOTOR_OFF__DIR_EXT

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

1AD2     MOVE.W   #100,D0             ; D0=100
1AD6     DBF      D0,1AD6             ; ←→ D0=D0-1, as long as D1 is different from -1, it's a pause
1ADA     MOVE.B   #F5,BFD100          ; Conf CIA-B / PRB
                                        MOTOR OFF, DF3 UNSELECT, DF2 UNSELECT, DF1 UNSELECT, DF0 SELECT, SIDE DOWN, DIR EXT., STEP TRACK=1

1AE2     MOVE.W   #B000,D0            ; D0=B000
1AE6     DBF      D0,1AE6             ; ←→ D0=D0-1, as long D0 is different from -1, we loop. (we have a 'pause' here)
1AEA     LEA      C1E(PC),A3          ; A3=position in the table
1AEE     CLR.W    (A3)                ; We clear the marker
1AF0     RTS                          ; E.T Back Home
```

#Trackloader_Start   A0=Memory_Adr_Destination   A1=Table Pointer(Adress_Start_raw)

```
1B2E     MOVE.L   A0,-(A7)            ; Save A0 in stack
1B30     BSR      1B38                ; GoSub → #Read_Table_and_Start_Trackload_Or_Not
1B34     MOVE.L   (A7)+,A0            ; Restore A0 from stack
1B36     RTS                          ; E.T Back Home
```

#Read_Table_and_Start_Trackload_Or_Not

```
1B38     LEA      C1E(PC),A3         ; A3=C1E (Address in the table)
1B3C     TST.W    (A3)               ; Test content of A3 with zero
1B3E     BNE      1B44               ; If not equal then GoTo → #Side_Select
1B40     BSR      19C4               ; GoSub → #Trackloader_2_Init

#Side_Select
1B44     MOVE.L (A1)+, D0            ; D0=A1 then A1=A1+4
1B46     CMP.L    #84468,D0          ; D0-84468 and modifies the Flag accordingly
                                     ; Test to determine on which side we will use
                                     ; Same value as the other TrackLoader, namely 84468
                                     ; (Depending on the result, we will go to to Side_UP or Side_Down

1B4C     BGE      1B52               ; If > then GoTo  → #Recover_Info_to_Trackloader_2 et #DF0__SIDE_UP__MOTOR_ON__DIR_EXT

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

1B50     BRA      1B54               ; And then, next, GoTo → #Recover_Info_to_Trackloader_2
```

#Recover_Info_to_Trackloader_1

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

#Recover_Info_to_Trackloader_2
1B54     MOVE.L   D0,D3              ; At this point, First call, D0=(LongWord of $189C) so value in table =$189C
                                     ; namely 'Adress_Start_raw'
                                     ; D3=D0, so D3=$189C

1B56     DIVU.W   #$189C,D0          ; D0=D0/$189C and set the result into D0

1B5A     CMP.W    #56,D0             ; Compare D0 with the value 56
1B5E     BLT      1B6A               ; If less than, we branch to 1B6A
1B60     SUBI.L   #84468,D3          ; Otherwise, D3=D3-$84468
1B66     SUBI.W   #56,D0             ; D0=D0-$56
1B6A     MOVEQ    #0,D2              ; D2=00
1B6C     MOVE.W   D0,D2              ; D2=D0
1B6E     BSR      1A6C               ; GoSub → #Position Reach?

1B72     MULU     #189C,D2           ; At this point, D2=$189C = position reached = Head position
                                     ; multiplied by the size of a Track
                                     ; it becomes the size in position_raw_reach
                                     ; Position where we are, in this case : $189C

1B72     SUB.L    D2,D3              ; This Step, D3=(LongWord of 189C)
                                       Or if D0 Start_Track was larger than $56
                                       we subtract the position_raw_reach of D3

                                       In fact, according to the 'Address_Start_raw' in the table, we will use a specific face.

                                       This operation performed on D3 allows
                                       to reposition the Address_Start_raw changing face at the same time.

1B78     MOVE.L   (A1),D4            ; Copy the longWord content at address A1 into D4

1B7A     LSR.L    #2,D4              ; Shift of 2 bits to the right of D4, equal to dividing by D4 by 4
1B7C     SUBQ.L   #1,D4              ; D4=D4-1

1B7E     BSR      1B86               ; → GoSub → #Trackload_Base
1B80     BSR      1A18               ;     GoSub → #Move Inside.
1B84     BRA      1B7E               ; ← Branch to 1B7E, we loop on #Trackload_Base
```

#Trackload_Base

```
1B86    MOVEQ    #0F,D2              ; D2=F
1B88    MOVE.L   #18B8,D0            ; ➔ D0=$18B8
1B8E    MOVE.L   C1A(PC),A6          ; A6=Address in the table=0C1A=$00018000    //Memory Adr. for DSKPTH
1B92    MOVE.W   #2,DFF09C           ; Conf INTREQ, Disk Block Finished Interupt
1B9A    MOVE.L   A6,DFF020           ; Conf DSKPTH = $18000
1BA0    MOVE.W   #8210,DFF096        ; Conf DMACON, DSKEN enable, ALL DMA enable
1BA8    MOVE.W   #$4489,DFF07E       ; Conf DSKSYNC = $4489 (standard AmigaDos)
1BB0    MOVE.W   #7F00,DFF09E        ; Conf ADKCON specific
1BB8    MOVE.W   #B500,DFF09E        ; Once again to start the transfer.
                                       FAST=2us, WORDSYNC=active, MFM precomp, recomp=140ns

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

#Test CIA Ready
1BC8    MOVE.B   BFDD00,D7           ; D7=BFDD00
1BCE    MOVE.B   BFDD00,D7           ;       ➔ D7=ICR register of CIA-B
1BD4    BTST     #4,D7               ;         Test bit4 (FLAG) of ICR
1BD8    BEQ      1BCE                ;       ➔ Interruption generated ? We loop

1BDA    ADDI.W   #8001,D0            ; ADD signed on D0 (which is $18B8 see a few lines above) with $8001
                                       what gives us : $98B9 and flag C=0

1BDE    MOVE.W   D0,DFF024           ; CONF DSKLEN, Disk DMA Enable, Length of DMA data=$18B9
1BE4    MOVE.W   D0,DFF024           ; Once again to trigger the reading

1BEA    MOVE.W   DFF01E,D0           ;       ➔ D0=INTREQR
1BF0    BTST     #1,D0               ;         Test Bit1 of INTREQR, Level 1 Disk Block Finished Interupt
1BF4    BEQ      1BEA                ;       ← Test ?, We loop

1BF6    MOVE.L (A6)+, D0             ; Value of DSKPTH into D0, (A6=DFF024) then A6=A6+4
1BF8    MOVE.L (A6)+, D7             ; Value of DSKLEN into D7, (A6=DFF028) then A6=A6+4
1BFA    ADD.L    D0,D0               ; D0=D0+D0
1BFC    ANDI.L   #AAAAAAAA,D0        ; MFM Post_Treatment odd bit into D0
1C02    ANDI.L   #55555555,D7        ; Post_MFM bit even processing into D7
1C08    OR.L     D7,D0               ; MFM treatment
1C0A    CMP.L    #42535432,D0        ; Compare D0 with 42535432
                                       42 53 54 32 In Ascii = BST2

1C10    BEQ      1C20                ; Yes ? 'Signature' found GoSub ➔ #Processing_MFM_BASE
                                       We start the real processing of the data and the Trackload

1C12    DBF      D2,1B88             ; ← D2=D2-1, as long as D2 is different from -1, we loop to 1B88
                                         Let's go for a ride

1C16    MOVE.W   #F00,DFF180         ; ➔ Background in red
1C1E    BRA      1C16                ; ← DeadLoop Red background
```

#Move Inside.

```
1A18    BCLR     #1,BFD100           ; CIA-B / PRB, bit1 set to 0, DIR= towards the inside.
1A20    BCLR     #0,BFD100           ; CIA-B / PRB, bit0 set to 0, STEP, pre/move heads
1A28    BSET     #0,BFD100           ; CIA-B / PRB, bit0 set to 0, STEP, head movement.
1A30    BSR      1ABC                ; GoSub ➔ #WAIT
1A34    MOVE.L   A0,-(A7)            ; Save A0 in stack
1A36    LEA      C20(PC),A0          ; 00C20(PC)=A0 Address in the table
1A3A    ADDQ.W   #01,(A0)            ; (A0)=A0+1 Word
                                     ; We moved from a Track to the inside and (A0), $C20 is increased by 1
                                     ; Here $C20 it's a Track pointer.

1A3C    MOVE.L   (A7)+,A0            ; Restore A0 from stack
1A3E    BRA      1AB0                ; GoTo ➔ #Disk Ready? without passing through the modification of A3
```

#Position Reach?

```
1A6C    CMP.W    C20(PC),D0          ; Check D0 $C20 (Track Pointer) with D0
1A70    BEQ      1A84                ; If equal then GoTo ➔ #Forward_Backward
1A72    BGT      1A88                ; If N=0, (greater than), then GoTo ➔ #GoTo_Position
1A74    MOVE.W   C20(PC),D6          ; D6=$C20= Track pointer
1A78    SUB.W    D0,D6               ; D6=D6-D0
1A7A    SUBQ.W   #1, D6              ; D6=D6-1
1A7C    BSR      1A42                ; ➔ GoSub ➔ #Move Outside.
1A7E    DBF      D6,1A7C             ; ← D6=D6-1, as long D6 is different from -1, we loop.
1A82    RTS                          ; E.T back home
```

#Forward_Backward

```
1A84    BSR      1A18                ; GoSub ➔ #Move Inside.
1A86    BRA      1A42                ; GoTo ➔ #Move Outside.
```

#GoTo_Position

```
1A88    SUB.W     C20(PC),D0         ; D0=D0-( content of $C20), aka, current position
1A8C    SUBQ.W    #1,D0              ; D0=D0-1
1A8E    BSR       1A18               ; → GoSub → #Move Inside.
1A90    DBF       D0,1A8E            ; ← D0=D0-1, as long D0 is different from -1, we loop.
1A94    RTS                          ; E.T back home
```

#DF0__SIDE_DOWN__MOTOR_ON__DIR_EXT

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

1B06    MOVE.W    #B000,D7           ; D7=B000, pause counter
1B0A    DBF       D7,1B0A            ; ←→ D7=D7-1, as long D7 is different from -1, we loop. (we have a ‘pause’ here)
1B0E    RTS                          ; E.T back home
```

#DF0__SIDE_UP__MOTOR_ON__DIR_EXT

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

1B24    MOVE.W    #B000,D7           ; D7=B000, pause counter
1B28    DBF       D7,1B28            ; ←→ D7=D7-1, as long D7 is different from -1, we loop. (we have a ‘pause’ here)
1B2C    RTS                          ; E.T back home
```

#Processing_MFM_BASE

```
1C20    MOVEQ     #0,D2              ; D2=00
1C22    MOVE.W    #626,D2            ; D2=626
1C26    TST.W     D3                 ; Test D3 with zero            // D3= Delta previously calculated.
1C28    BEQ       1C34               ; If equal then GoTo → #Check_Processing_MFM
1C2A    ADD.W     D3,D3              ; Otherwise D3=D3+D3                              // on fait x2 sur D3

                                       Reminder, D3=((Table value Address_Start_raw)-(value position reached))

1C2C    ADDA.L    D3, A6             ; A6=A6+D3                          // A6=DSKPTH+decoding in progress
1C2E    LSR.W     #3,D3              ; Shift 3 Bits to the left of LongWord D3// Equals to dividing D3 by 8
                                       So delta between position reached and  (value Length_To_Read)/8

1C30    SUB.W     D3,D2              ; D2(current_raw_position)- Delta calculated above
1C32    MOVEQ     #0,D3              ; D3=0                              // We reset D3 to Zero

#Check_Processing_MFM
1C34    CMP.L     D2,D4              ; Compare D4-D2
1C36    BGT.B     1C3E               ; if result greater than, then GoTo → #MFM bit even processing
1C38    MOVE.W    D4,D2              ; D2=D4
1C3A    ADDQ.W    #4,A7              ; A7=A7+4
1C3C    BRA       1C42               ; GoTo → #Start_Decoding
```

#MFM bit even processing

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

#Start_Decoding
01C42   MOVE.L    #55555555,D5       ; D5=55555555, MFM bit even mask
01C48   MOVE.L    (A6)+,D0           ; → Copy contained of A6 into D0 then A6=A6+4
01C4A   MOVE.L    (A6)+,D7           ;     Copy contained of A6 into D7 then A6=A6+4
01C4C   AND.L     D5,D0              ;     MFM treatment
01C4E   AND.L     D5,D7              ;     MFM treatment
01C50   ADD.L     D0,D0              ;     MFM treatment
01C52   OR.L      D7,D0              ;     MFM treatment
01C54   MOVE.L    D0,(A0)+           ;     Copy D0 to the address provided by A0 then A0=A0+4
01C56   DBF       D2,1C48            ; ← D2=D2-1, as long D2 is different from -1, we loop.
01C5A   RTS                          ; E.T back home
```

#Decomp/Decrypt

```
                                      #Decompression Adr. = Same as the address Source of the Trackload (so A0)
1824    MOVEA.L  A0,A1                ; A1=A0=Adr. Source, we decompress to the same adr. Memory as the TrackLoad
                                        So we overwrite the 'source' area.

#Decomp/Decrypt_By_Adress

                                      #Decompression Adr. = A1
1826    MOVEQ    #0,D7                ; D7=00
1828    MOVE.L   A0,A2                ; A2=A0          A2=A0=Adr. Source
182A    MOVE.L   (A0),D0              ; Copy the first longWord of A0 into D0 // (Length compressed+1 for info)
182C    BTST     #0,D0                ; Test this one
1830    BEQ      183A                 ; If = 0000 then branch to $138A
1832    MOVE.L   A1,A3                ; A3=A1=Source Address
1834    NOT.W    7                    ; Binary Operation, NOT on the Word D7
1836    ANDI.W   #FFFE,D0             ; Binary Operation, AND between D0 and FFFE(result put into D0)

#Lenght_Indicated_in_D0
183A    ADDA.L   D0,A0                ; In this case, we add D0 to A0 (so theoretically, it is the Length of the data.)
183C    MOVE.L   -(A0),(A2)           ; Copy the longword of A0 into A2 then A0=A0-4
183E    MOVE.L   -(A0),A2             ; Copy the longword of A0 to A2, So the second LongWord contains the information put in A2
1840    ADDA.L   A1,A2                ; A2=A2+A1
1842    MOVE.L   -(A0),D5             ; Copy the longword of A0 to D5 then A0=A0-4
1844    MOVE.L   -(A0),D0             ; Copy the longword of A0 to D0 then A0=A0-4
1846    MOVEQ    #10,D6               ; D6=$10
1848    EOR.L    D0,D5                ; Binary Operation, EOR between D0 and D5
184A    LSR.L    #1,D0                ; Shift 1 bit of D0 to the right
184C    BNE      1850                 ; test flag
184E    BSR      18C4                 ; ...
1850    BCS      1884                 ; ... ...
1852    MOVEQ    #8,D1
1854    MOVEQ    #1,D3
1856    LSR.L    #1,D0
1858    BNE      185C
185A    BSR      18C4
185C    BCS      18A6
185E    MOVEQ    #3,D1
1860    MOVEQ    #0,D4
1862    BSR      18CE
1864    MOVE.W   D2,D3
1866    ADD.W    D4,D3
1868    MOVEQ    #7,D1
186A    LSR.L    #1,D0
186C    BNE      1870
186E    BSR      18C4
1870    ROXL.L   #1,D2
1872    DBF      D1,186A
1876    MOVE.B   D2,-(A2)
1878    DBF      D3,1868
187C    BRA      18B2
===========================================================================
187E    MOVEQ    #8,D1
1880    MOVEQ    #8,D4
1882    BRA      1862
===========================================================================
1884    MOVEQ    #2,D1
1886    BSR      18CE
1888    CMP.B    #2,D2
188C    BLT      189E
188E    CMP.B    #03,D2
1892    BEQ      187E
1894    MOVEQ    #8,D1
1896    BSR      18CE
1898    MOVE.W   D2,D3
189A    MOVEQ    #C,D1
189C    BRA      18A6
===========================================================================
189E    MOVEQ    #9,D1
18A0    ADD.W    D2,D1
18A2    ADDQ.W   #2,D2
18A4    MOVE.W   D2,D3
18A6    BSR      18CE
18A8    SUBQ.L   #1,A2
18AA    MOVE.B   0(A2,D2),(A2)
18AE    DBF      D3,18A8
18B2    CMPA.L   A2,A1
18B4    BLT      184A
18B6    TST.L    D5
18B8    BNE      18E6
18BA    TST.W    D7
18BC    BEQ      8C0
18BE    BRA      18EA
===========================================================================
18C0    MOVEQ    #0,D0
18C2    RTS
===========================================================================
18C4    MOVE.L   -(A0),D0
18C6    EOR.L    D0,D5
18C8    MOVE.W   D6,CCR
18CA    ROXR.L   #1,D0
18CC    RTS
===========================================================================
```

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

#Decomp/Decrypt_02

```
                                        #Decompression Adr. = Same as the address Source of the Trackload (so A0)
1958     MOVE.L    A0,A1                ; A1=A0=Adr. Source
195A     MOVE.L    A1,A2                ; A2=A1
195C     MOVE.L    A0,A3                ; A3=A0         In brief... all in one place
195E     MOVE.L    (A0),D0              ; First LongWord to D0

1960     LSR.L     #8,D0                ; Shifts the result one byte to the right
1962     ADDA.L    D0,A0                ; And we add it to A0
1964     MOVE.B    -(A0),(A3)+
1966     MOVE.B    -(A0),(A3)+
1968     MOVE.B    -(A0),(A3)
196A     MOVEQ     #0,D1
196C     MOVE.B    -(A0),D1

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

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

#Base_Treatment
1984     CMPA.L    A2,A1
1986     BLE       19C2                 ; GoTo → End_of_routine_and_RTS
1988     MOVE.B    -(A0),D0
198A     CMP.B     D0,D4
198C     BEQ       19B4                 ; GoTo → #Processing_1
198E     CMP.B     D0,D5
1990     BEQ       199E                 ; GoTo → #Processing_2
1992     CMP.B     D0,D6
1994     BEQ       19A8                 ; GoTo → #Processing_3
1996     CMP.B     D0,D7
1998     BEQ       19AE                 ; GoTo → #Processing_4
199A     MOVE.B    D0,-(A1)
199C     BRA       1984                 ; GoTo → #Base_Treatment
===========================================================================
                                        # Processing_2
199E     MOVE.B    -(A0),D0
19A0     MOVE.B    D0,-(A1)
19A2     MOVE.B    D0,-(A1)
19A4     MOVE.B    D0,-(A1)
19A6     BRA       1984                 ; GoTo → #Base_Treatment
===========================================================================
                                        #Processing_3
19A8     MOVE.B    D2,-(A1)
19AA     MOVE.B    D2,-(A1)
19AC     BRA       1984                 ; GoTo → #Base_Treatment
===========================================================================
                                        #Processing_4
19AE     MOVE.B    D3,-(A1)
19B0     MOVE.B    D3,-(A1)
19B2     BRA       1984                 ; GoTo → #Base_Treatment
===========================================================================
                                        #Processing_1
19B4     MOVEQ     #0,D0
19B6     MOVE.B    -(A0),D1
19B8     MOVE.B    -(A0),D0
19BA     MOVE.B    D1,-(A1)             ; → Loop to A1
19BC     DBF       D0,19BA              ; ←
19C0     BRA       1984                 ; GoTo → #Base_Treatment
===========
================================================================
19C2     RTS                            ; E.T back home, End of Sub-routine decomp/decrypt
===========================================================================
```

#Loading_Phase_#1

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

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

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

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

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

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

                                  A BreakPoint in $1956 followed by the command ST
                                  will end up with a return in $1E66 namely, #Loading_Phase_#2_1/2
```

#Loading_Phase_#2_1/2

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

#Loading_Phase_#2_1/2_bis
1E74    MOVE.L  #7C000,C1A.S    ; Copy $7C000 to the address $C1A, Update of the DSKPTH
1E7C    BSR     1D48            ; GoSub → #LoadPhase_#1_End_Part1/2
1E80    BSR     1A96            ; GoSub → #Return_T00

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

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


1EA8    BSR     1CE0            ; GoSub → #Data_Already_Loaded?   If it not the case, continue to loading phase #1
                                                                  in 1CE6, 'Loading_Phase_#2_2/2'

1EAC    JSR     1ACA.S          ; GoSub → #DF0__SIDE_DOWN__MOTOR_OFF__DIR_EXT
1EB0    LEA     DFF180,A0       ; A0=DFF180
1EB6    MOVEQ   #F,D0           ; D0=$F, counter
1EB8    CLR.L   (A0)+           ; → Clear (A0)
1EBA    DBF     D0,1EB8         ; ← D0=D0-1, as long D0 is different from -1, we loop. (so 16 times)

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

```
1EC2     LEA      DFF000,A6          ; A6=DFF000
1EC8     MOVE.L   #441B8,80(A6)      ; Conf copper  // COP1LCH //Conf. address of the Copperlist(1) = $441B8
1ED0     MOVE.W   #5200,100(A6)      ; Conf BitPlan // BPLCON0
1ED6     CLR.L    102(A6)            ; Conf BitPlan // BPLCON1
1EDA     CLR.L    108(A6)            ; Conf BitPlan // BPL1MOD
1EDE     MOVE.L   #2C81F4C1,8E(A6)   ; Conf Display // DIWSTRT
1EE6     MOVE.L   #3800D0,92(A6)     ; Conf Display // DDFSTRT
1EEE     MOVEQ    #FFFFFFFF,D0       ; D0=FFFFFFFF
1EF0     MOVE.L   D0,44(A6)          ; Conf BLTAFWM=D0
1EF4     LEA      BC2.S,A0           ; A0=$BC2
1EF8     MOVE.L   #ED04,(A0)+        ; Copy ED04 into (A0) (table ?), then A0=A0+4
1EFE     MOVE.L   #F66E,(A0)+        ; Copy F66E into (A0), then A0=A0+4
1F04     MOVE.L   #F67C,(A0)+        ; Copy F67C into (A0), then A0=A0+4
1F0A     MOVE.L   #F68A,(A0)+        ; Copy F68A into (A0), then A0=A0+4
1F10     MOVE.L   #F698,(A0)+        ; Copy F698 into (A0), then A0=A0+4
1F16     MOVE.L   #F3BE,(A0)+        ; Copy F3BE into (A0), then A0=A0+4
1F1C     MOVE.L   #F3C2,(A0)         ; Copy F3C2 into (A0)
1F22     TST.B    BE0.S              ; Byte Test of $BE0
1F26     BEQ      1F48               ; If equal to Zero then GoTo #1F48
1F28     MOVEQ    #1,D0              ; D0=1
1F2A     MOVE.B   D0,F3C6            ; Copy D0 to $F3C6
1F30     MOVE.B   D0,BC0.S           ; Copy D0 to $BC0
1F34     MOVE.B   BBF.S,F3C3         ; Copy BBF to $F3C3
1F3C     JSR      EB7E               ; GoSub → #EB7E
1F42     MOVE.B   #1,BBE.S           ;
1F48     MOVE.L   #794C4,42E.S       ;
1F50     BSR      9D3E               ; GoSub → #ERASE_BB6
1F54     MOVE.W   #8180,DFF096       ;
1F5C     LEA      44FD2,A0           ;
1F62     JSR      4504E              ; GoSub → #COLOR_TABLE_AND_CO
1F68     MOVEQ    #2,D5              ;
1F6A     MOVE.W   #12C,D0            ;
1F6E     TST.B    BB4.S              ; → Test FIRE button pushed ?
1F72     BNE      24CC               ;   Yes ? then GoTo → #LOADING_LEVEL1
1F76     BSR      9D3E               ;   Otherwise -----→  GoSub → #ERASE_BB6
1F7A     DBF      D0,1F6E            ; ← D0=D0-1, as long D0 is different from -1, we loop.
1F7E     LEA      23E0,A0            ;
1F84     MOVE.W   #4CD4,2(A0)        ;
1F8A     BSR      22FE               ; GoSub → #BLITTER_CONF_#01
1F8E     BSR      2334               ; GoSub → #BLITTER_CONF_#02
1F92     BSR      23B4               ; GoSub → #START_LEVEL1_OR_NOT
1F96     MOVEQ    #28,D7             ;
1F98     TST.B    BB4.S              ; → Test FIRE button pushed ?
1F9C     BNE      24CC               ;   Yes ? then GoTo → #LOADING_LEVEL1
1FA0     BSR      9D3E               ;   Otherwise -----→  GoSub → #ERASE_BB6
1FA4     DBF      D7,1F98            ; ← D7=D7-1, as long D7 is different from -1, we loop.
1FA8     MOVEQ    #3C,D7             ;

1FAA     MOVE.L   DFF004,D0          ; →
1FB0     LSR.L    #8,D0              ;
1FB2     ANDI.W   #1FF,D0            ;
1FB6     CMP.W    #8E,D0             ;
1FBA     BLT      1FAA               ;       ← We loop as long as D0 does not fit

1FBC     BSR      237E               ;       GoSub → #BLITTER_CONF_#00
1FC0     SUBI.W   #28,2(A0)          ;
1FC6     BSR      22FE               ;       GoSub → #BLITTER_CONF_#01
1FCA     BSR      2334               ;       GoSub → #BLITTER_CONF_#02
1FCE     MOVE.L   DFF004,D0          ;
1FD4     LSR.L    #8,D0              ;
1FD6     ANDI.W   #1FF,D0            ;
1FDA     CMP.W    #1E,D0             ;
1FDE     BLT      1FCE               ;       ← We loop as long as D0 does not fit
1FE0     CMP.W    #32,D0             ;
1FE4     BGT      1FCE               ;       ← We loop as long as D0 does not fit
1FE6     TST.B    BB4.S              ;       Test FIRE button pushed ?
1FEA     BNE      24CC               ;       Yes ? then GoTo → #LOADING_LEVEL1
1FEE     DBF      D7,1FAA            ; ← D7=D7-1, as long D7 is different from -1, we loop.
1FF2     MOVEQ    #4B,D7             ;
1FF4     TST.B    BB4.S              ; → Test FIRE button pushed ?
1FF8     BNE      24CC               ;   Yes ? alors GoTo → #LOADING_LEVEL1
1FFC     BSR      9D3E               ;   Otherwise -----→  GoSub → #ERASE_BB6
2000     DBF      D7,1FF4            ; ← D7=D7-1, as long D7 is different from -1, we loop.
2004     LEA      23F4,A0            ;
200A     MOVE.W   #491C,2(A0)        ;
2010     MOVE.W   #0400,A(A0)        ;
2016     MOVE.W   #DE1C,E(A0)        ;
201C     MOVEQ    #7,D7              ;
201E     BRA      2030               ; GoTo → $2030
...
```

#Decrypt_RAW

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

5724     MOVE.W   (A0)+,D0          ; ➔ Copy (A0) into D0 then A0=A0+2
5726     MOVE.W   D0,D1             ;    D1=D0
5728     LSR.W    #1,D0             ;    Shift 1 Bits to the right the LongWord D0// Equivalent to dividing D0 by 2
572A     SUBQ.W   #1,D0             ;    D0=D0-1
-----------------------------------------------------
572C     MOVE.L   A2,A3             ;    A3=A2
572E     ADDA.W   D1,A3             ;    A3=A3+D1

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

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

5738     MOVE.L   A5,A6             ;          A6=A5
573A     ADDA.W   D1,A6             ;          A6=A6+D1
-----------------------------------------------------
573C     MOVE.W   (A2)+,D2          ;        ➔ D2=(A2) then A2=A2+2
573E     OR.W     (A3)+,D2          ;          OR of (A3) with D2, result in D2 then A3=A3+2
5740     OR.W     (A4)+,D2          ;          OR of (A4) with D2, result in D2 then A3=A3+2
5742     OR.W     (A5)+,D2          ;          OR of (A5) with D2, result in D2 then A3=A3+2
5744     OR.W     (A6)+,D2          ;          OR of (A6) with D2, result in D2 then A3=A3+2
5746     MOVE.W   D2,(A1)+          ;          Copy D2 into (A1) then A1=A1+2
5748     DBF      D0,573C           ;        ← D0=D0-1, as long D0 is different from -1, we loop.
574C     MOVE.L   A6,A2             ;    A2=A6
574E     DBF      D3,5724           ; ← D3=D3-1, as long D3 is different from -1, we loop.
5752     RTS                        ; E.T back home
```

#Data_Already_Loaded?

```
1CE0     TST.B    BE0.S             ; Bits test of $BE0, Tag Data already loaded
1CE4     BEQ      1D46              ; If equal to 0, then GoTo ➔ 1D46 (which is a RTS)

#Loading_Phase_#2_2/2
1CE6     LEA      EB7E,A0           ; A0=EB7E                    Memory Adr 'destination'
1CEC     LEA      1686.S,A1         ; A1=1686                    Table Pos.          // Loading Phase#1 12/16
1CF0     BSR      1B2E              ; GoSub ➔ #Trackloader_Start
1CF4     BSR      1824              ; GoSub ➔ #Decomp/Decrypt

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

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

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

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

1D40     MOVE.B   #1,BE1.S          ; Copy the Byte 01 to the address $BE1, it's for sure an end of load marker
1D46     RTS                        ; E.T back home
```

#LoadPhase_#1_End_Part1/2

```
01D48    MOVE.W   #775,D0            ; D0=0775, counter
01D4C    LEA      43C.S,A0           ; A0=$43C
01D50    CLR.B    (A0)+              ; → Clear the Byte (A0)
01D52    DBF      D0,1D50            ; ← D0=D0-1, as long D0 is different from -1, we loop.
------------------------------------------------------------------------
01D56    LEA      2A8.S,A0           ; A0=2A8                          Memory Adr 'destination'
01D5A    LEA      17C6.S,A1          ; A1=17C6                         Table Pos.              // Loading Phase#1 09/16
01D5E    BSR      1B2E               ; GoTo → #Trackloader_Start
------------------------------------------------------------------------

==========================================================================
01D62    MOVE.W   #0,DFF180          ; Black background
01D6A    BSR      1824               ; GoSub → #Decomp/Decrypt

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

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

#RTZ_Tables

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

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

7B5A     LEA      35E.S,A0           ; A0=35E
7B5E     MOVEQ    #3,D1              ; D1=03, counter
7B60     MOVE.L   D0,(A0)+           ; → RTZ (A0)
7B62     MOVE.L   (A0),A1            ;    Copy (A0) to A1
7B64     MOVE.L   D0,(A1)            ;    Copy D0 into (A1)
7B66     MOVE.L   #515DC,(A0)+       ;    Copy $515DC into (A0) then A0=A0+4
7B6C     MOVE.L   D0,(A0)+           ;    RTZ (A0)
7B6E     DBF      D1,7B60            ; ←  D1=D1-1, as long D1 is different from -1, we loop.

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

7B7E     CLR.B    3AB.S              ; RTZ Table
7B82     CLR.B    357.S              ; RTZ Table
7B86     CLR.L    358.S              ; RTZ Table
7B8A     CLR.B    35C.S              ; RTZ Table
7B8E     RTS                         ; E.T back home
```

#Update_Table_CDC_CEC__D2C_D38

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

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

#Update_Table_10E4_10F0__1094_10A0

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

#JSR(A0)_or_Erase

```
5B10    MOVE.L   3A2.S,A0                   ; A0=$3A2
5B14    CMPA.L   #0,A0                      ; Compare A0 with value $00000000
5B1A    BEQ      5B1E                       ; If equal then GoSub $5B1E
5B1C    JSR      (A0)                       ; Otherwise, GoSub (A0)

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

#Update_Table_A0_With_3A0&3E0__E58_1210_E5C_1214

```
5AC4    MOVE.W   3A0.S,A0                   ; Copy the word from $3A0 to A0
5AC8    MOVE.W   A0,E58.S                   ; Copy A0 to $E58
5ACC    MOVE.W   A0,1210.S                  ; Copy A0 to $1210

5AD0    ADDA.W   #50,A0                     ; A0=A0+50        // $3E0
5AD4    MOVE.W   A0,E5C.S                   ; Copy A0 to $E5C
5AD8    MOVE.W   A0,1214.S                  ; Copy A0 to $1214
5ADC    RTS                                 ; E.T back home
```

#EB7E_Post_Processing

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

#ERASE_BB6

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

#COLOR_TABLE_AND_CO

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

#LOADING_LEVEL1

```
24CC     MOVE.L   #70400,C1A.S           ; Update DSKPTH to $70400
24D4     LEA      515DC.A0               ; A0=515DC                Memory Adr 'destination'
24DA     LEA      16AE.S,A1              ; A1=16AE                 Table Pos.              // Loading Level#1_01/06
24DE     BSR      1B2E                   ; GoSub → #Trackloader_Start
24E2     BSR      1824                   ; GoSub → #Decomp/Decrypt
24E6     LEA      4D800,A0               ; A0=4D800                Memory Adr 'destination'
24EC     LEA      16B6.S,A1              ; A0=16B6                 Table Pos.              // Loading Level#1_02/06
24F0     BSR      1B2E                   ; GoSub → #Trackloader_Start
24F4     BSR      1824                   ; GoSub → #Decomp/Decrypt
24F8     TST.B    C0.S                   ; Test of $C0             marker 'data already loaded'
24FC     BNE      24CC                   ; GoTo → #LOADING_LEVEL1
24FE     LEA      2AFBE,A0               ; A0=2AFBE                Memory Adr 'destination'
2504     LEA      16BE.S,A1              ; A1=16BE                 Table Pos.              // Loading Level#1_03/06
2508     BSR      1B2E                   ; GoSub → #Trackloader_Start
250C     BSR      1824                   ; GoSub → #Decomp/Decrypt
2510     LEA      AC00,A0                ; A0=AC00                 Memory Adr 'destination'
2516     LEA      16C6.S,A1              ; A1=16C6                 Table Pos.              // Loading Level#1_04/06
251A     BSR      1B2E                   ; GoSub → #Decomp/Decrypt
251E     MOVE.B   #1,F3BE                ;
2526     LEA      29654,A0               ; A0=29654                Memory Adr 'destination'
252C     LEA      16D6.S,A1              ; A1=16D6                 Table Pos.              // Loading Level#1_05/06
2530     BSR      1B2E                   ; GoSub → #Trackloader_Start
2534     BSR      1824                   ; GoSub → #Decomp/Decrypt
2538     LEA      6DE1C,A0               ; A0=6DE1C                Memory Adr 'destination'
253E     LEA      16CE.S,A1              ; A1=16CE                 Table Pos.              // Loading Level#1_06/06
2542     BSR      1B2E                   ; GoSub → #Trackloader_Start
2546     BSR      1824                   ; GoSub → #Decomp/Decrypt
254A     MOVEA.L  C1A.S,A0               ; Retreives DSKPTH and put it into A0
254E     BSR      9B24                   ; GoSub → #RTZ_Trackloader
...
```

#BLITTER_CONF_#00

```
02 37E    MOVE.W    #9F0,40(A6)           ; A6=DFF000, so Conf of BLTCON0
02384     CLR.W     42(A6)                ; Conf BLTCON1
02388     CLR.W     64(A6)                ; Conf BLTAMOD
0238C     MOVE.W    10(A0),66(A6)         ; Conf BLTDMOD
02392     MOVE.L    4(A0), 50(A6)         ; Conf BLTAPTH
02398     MOVE.L    (A0),A1               ; A1=(A0)
0239A     MOVEQ     #4,D0                 ; Counter in D0
0239C     MOVE.L    A1,54(A6)             ; → Conf BLTDPTH
023A0     BSR       5B42                  ; GoTo →
023A4     MOVE.W    12(A0),58(A6)         ; Conf BLTSIZE
023AA     ADDA.W    #1F40,A1              ; A1=A1+1F40
023AE     DBF       D0,239C               ; ← D0=D0-1, as long D0 is different from -1, we loop.
023B2     RTS                             ; E.T back home
```

#BLITTER_CONF_#01

```
022FE     MOVE.W    #9F0,40(A6)           ; A6=DFF000, so Conf of BLTCON0
02304     CLR.W     42(A6)                ; Conf BLTCON1
02308     MOVE.W    10(A0),64(A6)         ; Conf BLTAMOD
0230E     CLR.W     66(A6)                ; Conf BLTDMOD
02312     MOVE.L    4(A0),54(A6)          ; Conf BLTDPTH
02318     MOVE.L    (A0),A1               ; A1=(A0)
0231A     MOVEQ     #4,D0                 ; Counter in D0
0231C     MOVE.L    A1,50(A6)             ; → Conf BLTAPTH
02320     BSR       5B42                  ;    GoSub →
02324     MOVE.W    12(A0),58(A6)         ;    Conf BLTSIZE
0232A     ADDA.W    #1F40,A1              ;    A1=A1+1F40
0232E     DBF       D0,231C               ; ← D0=D0-1, as long D0 is different from -1, we loop.
02332     RTS                             ; E.T back home
```

#BLITTER_CONF_#02

```
02334     MOVE.W    #FCA,40(A6)           ; A6=DFF000, so Conf of BLTCON0
0233A     CLR.W     42(A6)                ; Conf BLTCON1
0233E     CLR.W     64(A6)                ; Conf BLTAMOD
02342     CLR.W     62(A6)                ; Conf BLTBMOD
02346     MOVE.W    10(A0),60(A6)         ; Conf BLTCMOD
0234C     MOVE.W    10(A0),66(A6)         ; Conf BLTDMOD
02352     MOVE.L    8(A0),4C(A6)          ; Conf BLTBPTH
02358     MOVE.L    (A0),A1               ; A1=(A0)
0235A     MOVEQ     #4,D0                 ; Counter in D0
0235C     MOVE.L    C(A0),50(A6)          ; → Conf BLTAPTH
02362     MOVE.L    A1,48(A6)             ;    Conf BLTCPTH
02366     MOVE.L    A1,54(A6)             ;    Conf BLTDPTH
0236A     BSR       5B42                  ;    GoSub →
0236E     MOVE.W    12(A0),58(A6)         ;    Conf BLTSIZE
02374     ADDA.W    #1F40,A1              ;    A1=A1+1F40
02378     DBF       D0,235C               ; ← D0=D0-1, as long D0 is different from -1, we loop.
0237C     RTS                             ; E.T back home
```

#START_LEVEL1_OR_NOT

```
023B4     MOVEQ     #F,D0                 ; Compteur en D0
023B6     MOVE.W    #FFF,D1               ;
023BA     MOVE.W    D1,196(A6)            ; →
023BE     BSR       9D3E                  ;   GoSub → #ERASE_BB6
023C2     TST.B     BB4.S                 ;   Test FIRE pushed ?
023C6     BNE       24CC                  ;   Pushed ? then  GoTo → #LOADING_LEVEL1
023CA     BSR       9D3E                  ;   if not ---→ GoSub → #ERASE_BB6
023CE     TST.B     BB4.S                 ;   Test FIRE pushed
023D2     BNE       24CC                  ;   GoTo → #LOADING_LEVEL1
023D6     SUBI.W    #111, D1              ;   D1=D1-111
023DA     DBF       D0,23BA               ; ← D0=D0-1, as long D0 is different from -1, we loop.
023DE     RTS                             ; E.T back home
```

#RTZ_TRACKLOADER

```
09B24     MOVE.L    A0,1818.S             ; Update DSKPTH at address $1818 of the table      isn't it strange to go
                                                                                              through A0 to do this ?

09B28     BSR       9B40                  ; GoSub → #Pre_Base_TrackLoadX
09B2C     BSR       9B8A                  ; GoSub → #Motor_ON
09B30     BSR       9BBA                  ; GoSub → #Return_T00
09B34     BSR       9C24                  ; GoSub → #Pre_Base_TRK_X2
09B38     BSR       9C94                  ; GoSub → #Base_Conf_Interupt
09B3C     BRA       9CDC                  ; GoTo  → #Processing_Trait_01
```

#Pre_Base_TrackLoadX

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

09B60    MOVE.W   #10,DFF096                     ; Conf DMACON
09B68    MOVE.W   #2,DFF09C                      ; Conf INTREQ
09B70    MOVE.W   #7FFF,DFF09E                   ; Conf ADKCON
09B78    MOVE.W   #8100,DFF09E                   ; Conf ADKCON
09B80    ORI.B    #$78,BFD100                    ; 'Or' binary betwwen 0111 1000 and  $BFD100, so DF0 to DF3 = select and Motor ON
09B88    RTS                                     ; E.T back home
```

#Motor_ON

```
09B8A    BCLR     #7,BFD100                      ; CIA-B / PRB, bit7 set to 0, motor On
09B92    MOVEQ    #64,D0                         ; D0=64, counter
09B94    BCLR     #3,$BFD100                     ; → CIA-B, bit3 set to 0, DF0 selected
09B9C    DBF      D0,9B94                        ; ← D0=D0-1, as long D0 is different from -1, we loop.
09BA0    BSR      9BA6                           ; GoSub → #Floppy_Ready?
09BA4    RTS                                     ; E.T back home
```

#Floppy_Ready?

```
09BA6    MOVE.W   #BB8, D6                       ; D6=BB8, counter
09BAA    DBF      D6,9BAA                        ; ↔ D6=D6-1, as long D0 is different from -1, we loop. (it's a break)
09BAE    BTST     #5,BFE001                      ; → Test Bit 5 of BFE001=Floppy Ready?
09BB6    BNE      9BAE                           ; ← as long as the is not ready, we loop
09BB8    RTS                                     ; E.T back home
```

#Return_T00

```
09BBA    BTST     #4,BFE001                      ; CIA-A / PRA, test bit4, Head on T00 ?
09BC2    BEQ      9BE0                           ; If equal to zero, then GoTo $9BE0 (Which is RTS)
09BC4    BSET     #1,BFD100                      ; CIA-B / PRB, bit1 set to 1, DIR= towards the outside
09BCC    BCLR     #0,BFD100                      ; CIA-B / PRB, bit0 set to 0, STEP, pre-movement head
09BD4    BSET     #0,BFD100                      ; CIA-B / PRB, bit0 set to 0, STEP, movement head
09BDC    BSR      9BA6                           ; GoTo → #Floppy_Ready?
09BDE    BRA      9BBA                           ; We loop, GoSub → #Return_T00
```

```
09BE0    RTS
```

#Return_Home

```
09BE2    BTST     #4,BFE001                      ; CIA-A / PRA, Test bit4, Head on T00 ?
09BEA    BEQ      9BF0                           ; GoTo → #GoTo_Position_base
09BEC    BSR      9BBA                           ; GoSub → #Return_T00
09BEE    BRA      9BE2                           ; We loop, GoTo → #Return_Home
```

#GoTo_Position_base

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

#GoTo_Position_delta_D2
09BF4    TST.W    D2                             ; Test of the Word D2
09BF6    BEQ      9C22                           ; If equal to Zero then GoTo $9C22 (which is RTS)
09BF8    BCLR     #1,BFD100                      ; CIA-B / PRB, bit1 set to 0, DIR= towards the inside
09C00    NOP                                     ;
09C02    NOP                                     ;
09C04    NOP                                     ;
09C06    BCLR     #0,BFD100                      ; CIA-B / PRB, bit0 set to 0, STEP, pre/move heads
09C0E    NOP                                     ;
09C10    NOP                                     ;
09C12    NOP                                     ;
09C14    BSET     #0,BFD100                      ; CIA-B / PRB, bit0 set to 0, STEP, Heads movement
09C1C    BSR      9BA6                           ; GoSub → #Floppy_Ready?
09C1E    SUBQ.W   #1,D2                          ; D2=D2-1
09C20    BRA      9BF4                           ; GoTo → #GoTo_Position_delta_D2
```

```
09C22    RTS                                     ; E.T back home
```

#Pre_Base_TRK_X2

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

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

#INTREQ_TEST
09C66     MOVE.W   DFF01E,D0                             ; D0=INTREQR
09C6C     SUBQ.L   #1,D1                                 ; D1=D1-1
09C6E     BEQ      9C26                                  ; If Flag Z=1 so let's go for a ride, GoTo → #Pre_Base_TRK_X2_bis
09C70     BTST     #1,D0                                 ; Test Bit 1 of D0
09C74     BEQ      9C66                                  ; If equal to Zero, we loop GoTo → #INTREQ_TEST
09C76     MOVE.W   #2,DFF09C                             ; Conf INTREQ, Clear DSKBLK (Disk Block Finished Interupt)
09C7E     RTS                                            ; E.T Back Home
```

#ICR_TEST

```
09C80     MOVE.B   BFDD00,D6                             ; D6=CIA-B ICR
09C86     MOVE.B   BFDD00,D6                             ; →  D6=CIA-B ICR
09C8C     BTST     #4,D6                                 ;    test bit 4 of ICR
09C90     BEQ      9C86                                  ; ← we loop if necessary
09C92     RTS                                            ; E.T Back Home
```

#Base_Conf_Interupt

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

#Processing_Trait_01

```
09CDC     MOVE.L   1818.S,A0                             ; A0=1818
09CE0     MOVE.L   #3778,D3                              ; Counter in D3
09CE6     MOVE.L   (A0),D0                               ; →
09CE8     ADDQ.L   #2,A0                                 ; A0=A0+2
09CEA     MOVE.L   #F,D2                                 ; Counter in D2
09CF0     MOVE.L   D0,D1                                 ;                →
09CF2     SWAP     D1                                    ;                Reverse in longword D1
09CF4     CMPI.W   #4454, D1                             ;                Compare with 4454 (Word of Custom Synchro?)
09CF8     BEQ      9D08                                  ;                If identical then GoTo → #Processing_Trait_02
09CFA     ADD.L    D0,D0                                 ;                D0=D0+D0
09CFC     DBF      D2,9CF0                               ;                ← D2=D2-1, as long D2 is different from -1, we loop.
09D00     DBF      D3,9CE6                               ; ← D3=D3-1, as long D3 is different from -1, we loop.
09D04     BRA      9D38                                  ; GoTo → #SET_D0_To_1
```

#Processing_Trait_02

```
09D08     MOVEQ    #0,D5                                 ;
09D0A     MOVE.L   (A0),D0                               ; → D0=(A0)
09D0C     ADDQ.L   #2,A0                                 ;   A0=A0+2
09D0E     MOVE.L   #F,D2                                 ;   Counter in D2
09D14     MOVE.L   D0,D1                                 ;                →
09D16     SWAP     D1                                    ;                Reverse in longword D1
09D18     CMPI.W   #4454,D1                              ;                Compare with 4454 (Word of Custom Synchro?)
09D1C     BEQ      9D2C                                  ;                If identical then GoTo → #Processing_Trait_03
09D1E     ADD.L    D0,D0                                 ;                D0=D0+D0
09D20     DBF      D2,9D14                               ;                ← D2=D2-1, as long D2 is different from -1, we loop.
09D24     ADDQ.L   #1,D5                                 ;    D5=D5+1
09D26     DBF      D3,9D0A                               ; ← D3=D3-1, as long D3 is different from -1, we loop.
09D2A     BRA      9D38                                  ; GoTo → #SET_D0_To_1
```

#Processing_Trait_03

```
09D2C     SUBI.L   #1A2C,D5                              ; D5=D5-1A2C
09D32     BMI      9D38                                  ; If negatif result, then GoTo → #SET_D0_To_1
09D34     CLR.W    D0                                    ; otherwise, we clear the Word D0
09D36     RTS                                            ; E.T Back Home
```

#SET_D0_To_1

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

This finally gives us the following table of calls to the main menu.

#Table_#02

Call Order	Call Addr	A0 Memory Adr	A1 Pos. Table	DSKPTH	(A1) LENGTH	Memory loading area address	Decomp/ Decrypt	NFO	Memory Decompr.area address	LENGTH Décomp
00	8CF8	000B0	8D16	7C100/18000	00004	000B0-000B4	NO	Check Signature Disk	N/A	N/A
01	1C66	46FB4	17D6	18000	03A90	46FB4-4AA44	BSR 1824	Load_Menu_01/16	46FB4-4D800	0684C
02	1C78	70000	17DE	18000	00250	70000-70250	BSR 1824	Load_Menu_02/16	70000-70390	00390
03	1C8A	45F34	17E6	18000	00438	45F34-4636C	BSR 1824	Load_Menu_03/16	45F34-46AD4	00BA0
04	1C9C	50000	17EE	18000	00124	50000-50124	BSR 1824	Load_Menu_04/16	50000-5019C	0019C
05	1CAE	515DC	17F6	18000	00664	515DC-51C40	BSR 1826	Load_Menu_05/16	42000-4324B	0124B
06	1CC6	50398	17FE	18000	00268	50398-50600	BSR 1824	Load_Menu_06/16	50398-509D7	0063F
07	1CD8	457A4	1806	18000	003FC	457A4-45BA0	BSR 1824	Load_Menu_07/16	457A4-45F34	00790
08	1E70	4324C	180E	18000	01EBC	4324C-45108	NO	Load_Menu_08/16	N/A	N/A
09	1D5E	002A8	17C6	7C000	00060	002A8-00308	BSR 1824	Load_Menu_09/16	002A8-0043C	00194
10	1E8E	641DC	1676	7C000	04C18	641DC-68DF4	BSR 1824	Load_Menu_10/16	641DC-6DE1C	9C40
11	1EA0	70400	167E	7C000	0271C	70400-72B1C	BSR 1824	Load_Menu_11/16	70400-794C4	090C4
12	1CF0	0EB7E	1686	7C000	007FC	0EB7E-0F37A	BSR 1824	Load_Menu_12/16	0EB7E-0F920	00DA2
13	1D02	0F920	168E	7C000	0E9D8	0F920-1F240	BSR 1824	Load_Menu_13/16	0F920-26488	16B68
14	1D14	26488	1696	7C000	01408	26488-27890	BSR 1824	Load_Menu_14/16	26488-27BC2	0173A
15	1D26	27BC2	169E	7C000	00E10	27BC2-289D2	BSR 1958	Load_Menu_15/16	27BC2-28AB2	00EF0
16	1D38	28AB2	16A6	7C000	00B48	28AB2-295FA	BSR 1958	Load_Menu_16/16	28AB2-29654	00BA2

And don't forget the following table for the next Loading, namely the TrackLoad of **Level #01**

#Table_#03

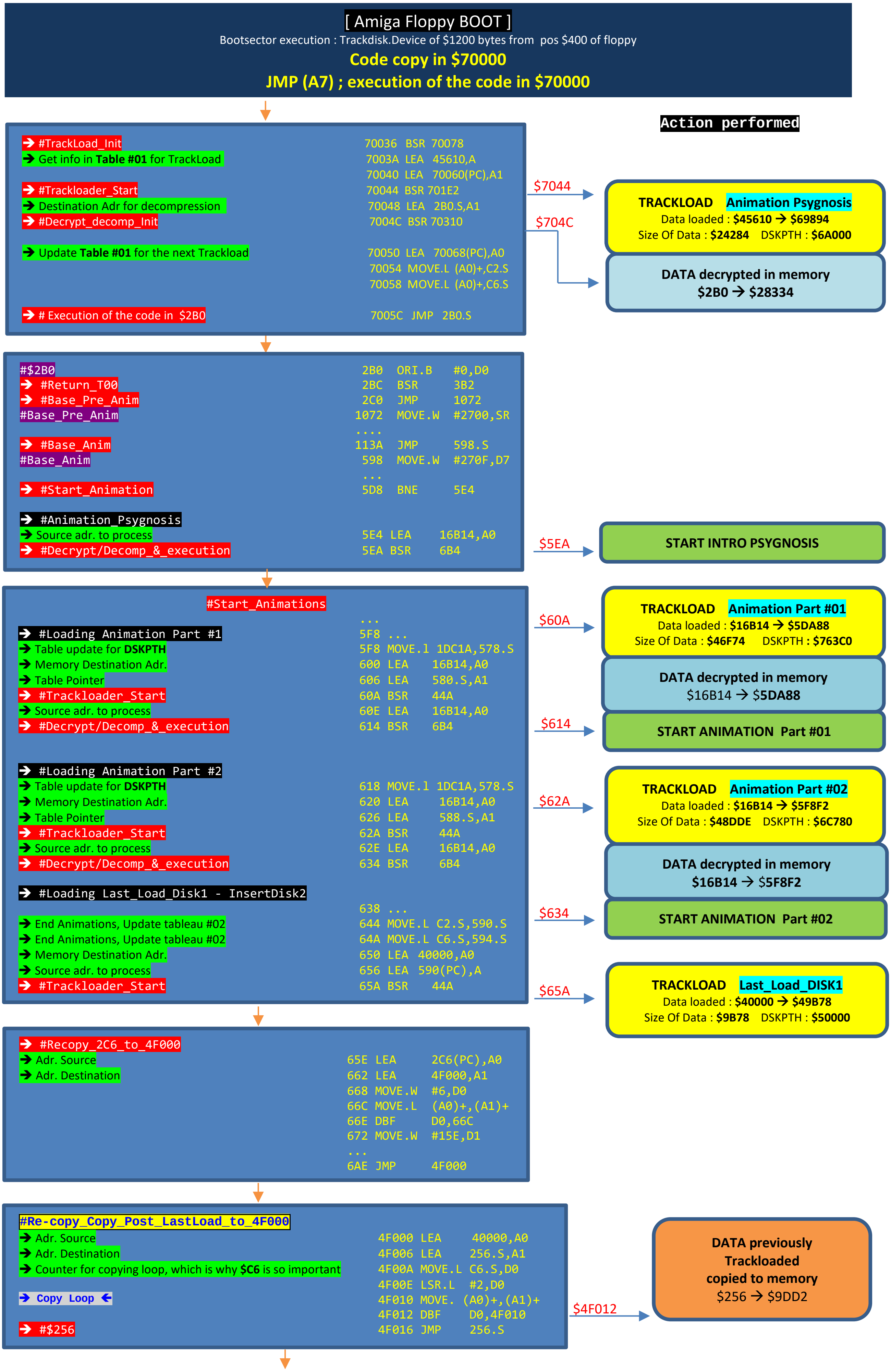
Call Order	Call Addr	A0 Memory Adr	A1 Pos. Table	DSKPTH	(A1) LENGTH	Memory loading area address	Decomp/ Decrypt	NFO	Memory Decomp. area address	LENGTH Décomp
17	24DE	515DC	16AE	70400	0CAA8	515DC-5E084	BSR 1824	Level#1_01/06	515DC-641DC	12C00
18	24F0	4D800	16B6	70400	00E84	4D800-4E684	BSR 1824	Level#1_02/06	4D800-50000	02800
19	2508	2AFBE	16BE	70400	0DDC8	2AFBE-38D86	BSR 1824	Level#1_03/06	2AFBE-42000	17042
20	251A	0AC00	16C6	70400	03C28	0AC00-0E828	No	Level#1_04/06	N/A	N/A
21	2530	29654	16D6	70400	01178	29654-2A7CC	BSR 1824	Level#1_05/06	29654-2AFBE	0196A
22	2542	6DE1C	16CE	70400	003A0	6DE1C-6E1BC	BSR 1824	Level#1_06/06	6DE1C-6FD9C	01E80

Note also that there is a base Track counter in memory **$C20**
/! It would be interesting to look at it a little later to see if it is not used during the game **out of the TrackLoad internal routine**.
Anyway, why wait? Just enter the AR once the Level is loaded and observe.

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

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

And indeed, the counter seems to be well used outside the Trackload routines, we will have to take this into account and recreate it if necessary.

Part 10 Diagram Part #01 (ak : the scary organization chart...)




#$256
➔ #Base_LastLoad
256 BRA 1DE0
#Base_LastLoad
1DE0 MOVE.B C0.S,D0
...
➔ #Trackloader_2_Init
1E48 JSR 19C4.S
➔ #Return_T00
1E4C JSR 1A96.S
➔ #BASE_INSERT_DISK_ASKED
1E50 JSR 8C30
#CHECK_DISK
DISK 2 inserted ? // Fire pressed
NO
YES
➔ #Wait_BB6
1E56 BSR 9D3E
➔ #Loading_Phase_#1
1E62 BSR 1C5C
Area of TRACKLOAD
#Loading_Phase_#1
➔ Memory Destination Adr.
1C5C LEA 46FB4,A0
➔ Table Pointer
1C62 LEA 17D6.S,A1
➔ #Trackloader_Start
1C66 BSR 1B2E
➔ #Decomp/Decrypt
1C6A BSR 1824
Loading_Phase#1 01/16
Data loaded : $46FB4 ➔ $4AA4A
Decomp : $46FB4 ➔ $4D800
➔ Memory Destination Adr.
1C6E LEA 70000,A0
➔ Table Pointer
1C74 LEA 17DE.S,A1
➔ #Trackloader_Start
1C78 BSR 1B2E
➔ #Decomp/Decrypt
1C7C BSR 1824
Loading_Phase#1 02/16
Data loaded : $70000➔ $70250
Decomp : $70000 ➔ $70390
➔ Memory Destination Adr.
1C80 LEA 45F34,A0
➔ Table Pointer
1C86 LEA 17E6.S,A1
➔ #Trackloader_Start
1C8A BSR 1B2E
➔ #Decomp/Decrypt
1C8E BSR 1824
Loading_Phase#1 03/16
Data loaded : $45F34➔ $4636C
Decomp : $45F34 ➔ $46AD4
Memory Destination Adr.
1C95 LEA 50000,A0
➔ Table Pointer
1C98 LEA 17EE.S,A1
➔ #Trackloader_Start
1C9C BSR 1B2E
➔ #Decomp/Decrypt
1CA0 BSR 1824
Loading_Phase#1 04/16
Data loaded : $50000 ➔ $50124
Decomp : $50000 ➔ $5019C
➔ Memory Destination Adr.
1CA4 LEA 515DC,A0
➔ Table Pointer
1CAA LEA 17F6.S,A1
➔ #Trackloader_Start
1CAE BSR 1B2E
➔ Memory Destination Adr. for decomp/decrypt
1CB2 LEA 42000,A1
➔ #Decomp/Decrypt_By_Adress
1CB8 BSR 1826
Loading_Phase#1 05/16
Data loaded : $515DC ➔ 51C40
Decomp : $42000 ➔ $4324B
➔ Memory Destination Adr.
1CBC LEA 50398,A0
➔ Table Pointer
1CC2 LEA 17FE.S,A1
➔ #Trackloader_Start
1CC6 BSR 1B2E
➔ #Decomp/Decrypt
1CCA BSR 1824
Loading_Phase#1 06/16
Data loaded : $50398 ➔ 50600
Decomp : $50398 ➔ $509D7
➔ Memory Destination Adr.
1CCE LEA 457A4,A0
➔ Table Pointer
1CD4 LEA 1806.S,A1
➔ #Trackloader_Start
1CD8 BSR 1B2E
➔ #Decomp/Decrypt
1CDC JMP 1824
Loading_Phase#1 07/16
Data loaded : $457A4 ➔ 45BA0
Decomp : $457A4 ➔ $45F34
Ending with a RTS to return to $E166
#Loading_Phase_#2 1/2
➔ Memory Destination Adr.
1E66 LEA 434C,A0
➔ Table Pointer
1E6C LEA 180E.S,A1
➔ #Trackloader_Start
1E70 BSR 1B2E
Loading_Phase#1 08/16
Data loaded : $4324C ➔ $45108
1E74 MOVE.L #7C000,C1A.S
➔ #LoadPhase_#1_End_Part1/2
1E7C BSR 1D48
#LoadPhase_#1_End_Part1/2
#$1D48
1D48 MOVE.W #775,D0
...
Memory Destination Adr.
1D56 LEA 2A8.S,A0
➔ Table Pointer
1D5A LEA 17C6.S,A
➔ #Trackloader_Start
1D5E BSR 1B2E
...
➔ #Decomp/Decrypt
1D6A BSR 1824
Loading_Phase#1 09/16
Data loaded : $2A8 ➔ $308
Decomp : $2A8 ➔ $43C
Ending with a RTS to return to #1E80

#Loading_Phase_#2 1/2

```
→ #Back_T00                          1E80 BSR 1A96

Memory Destination Adr.              1E84 LEA 641DC,A0
→ Table Pointer                      1E8A LEA 1676.S,A1
→ #Trackloader_Start                 1E8E BSR 1B2E
→ #Decomp/Decrypt                    1E92 BSR 1824

→ Memory Destination Adr.            1E96 LEA 70400,A0
→ Table Pointer                      1E9C LEA 167E.S,A1
→ #Trackloader_Start                 1EA0 BSR 1B2E
→ #Decomp/Decrypt                    1EA4 BSR 1824

→ #data already loaded?              1EA8 BSR 1CE0
```

Loading_Phase#1 10/16
Data loaded : $641DC → $68DF4
Decomp : $641DC → $6DE1C

Loading_Phase#1 11/16
Data loaded : $70400 → $72B1C
Decomp : $70400 → $794C4

#Data already loaded?

```
Marker' test data already loaded ?                 1CE0 TST.B BE0.S
If egale to 0 then → $1D46 wich is a RTS           1CE4 BEQ 1D46
In the present situation, this is not the case.
So we unroll the code.
```

#Loading_Phase_#2 2/2

```
→ Memory Destination Adr.            1CE6 LEA EB7E,A0
→ Table Pointer                      1CEC LEA 1686.S,A1
→ #Trackloader_Start                 1CF0 BSR 1B2E
→ #Decomp/Decrypt                    1CF4 BSR 1824

→ Memory Destination Adr.            1CF8 LEA F920,A0
→ Table Pointer                      1CFE LEA 168E.S,A1
→ #Trackloader_Start                 1D02 BSR 1B2E
→ #Decomp/Decrypt                    1D06 BSR 1824

Memory Destination Adr.              1D0A LEA 26488,A0
→ Table Pointer                      1D10 LEA 1696.S,A1
→ #Trackloader_Start                 1D14 BSR 1B2E
→ #Decomp/Decrypt                    1D18 BSR 1824

→ Memory Destination Adr.            1D1C LEA 27BC2,A0
→ Table Pointer                      1D22 LEA 169E.S,A1
→ #Trackloader_Start                 1D26 BSR 1B2E
→ #Decomp/Decrypt_02                 1D2A BSR 1958

→ Memory Destination Adr.            1D2E LEA 28AB2,A0
→ Table Pointer                      1D34 LEA 16A6.S,A1
→ #Trackloader_Start                 1D38 BSR 1B2E
→ #Decomp/Decrypt_02                 1D3C BSR 1958

Data loaded, the marker is modified  1D40 MOVE.B #1,BE1.S
                                     1D46 RTS

Will end up in RTS to return to #1EAC
```

Loading_Phase#1 12/16
Data loaded : $EB7E → $F37A
Decomp : $EB7E → $F920

Loading_Phase#1 13/16
Data loaded : $F920 → $1F240
Decomp : $ 0F920-26488

Loading_Phase#1 14/16
Data loaded : $26488 → $27890
Decomp : $ 26488-27BC2

Loading_Phase#1 15/16
Data loaded : $27BC2 → $289D2
Decomp : $27BC2 → $28AB2

Loading_Phase#1 16/16
Data loaded : $28AB2 → $295FA
Decomp : $28AB2 → $29654

#Loading_Phase_#2 1/2

```
#$1EAC                               ...
→ #Decrypt_RAW                       1EBE BSR 5710
                                     ...
Various routines to finish to        ...
→LOADING_LEVEL1                      ... BNE 24CC
```

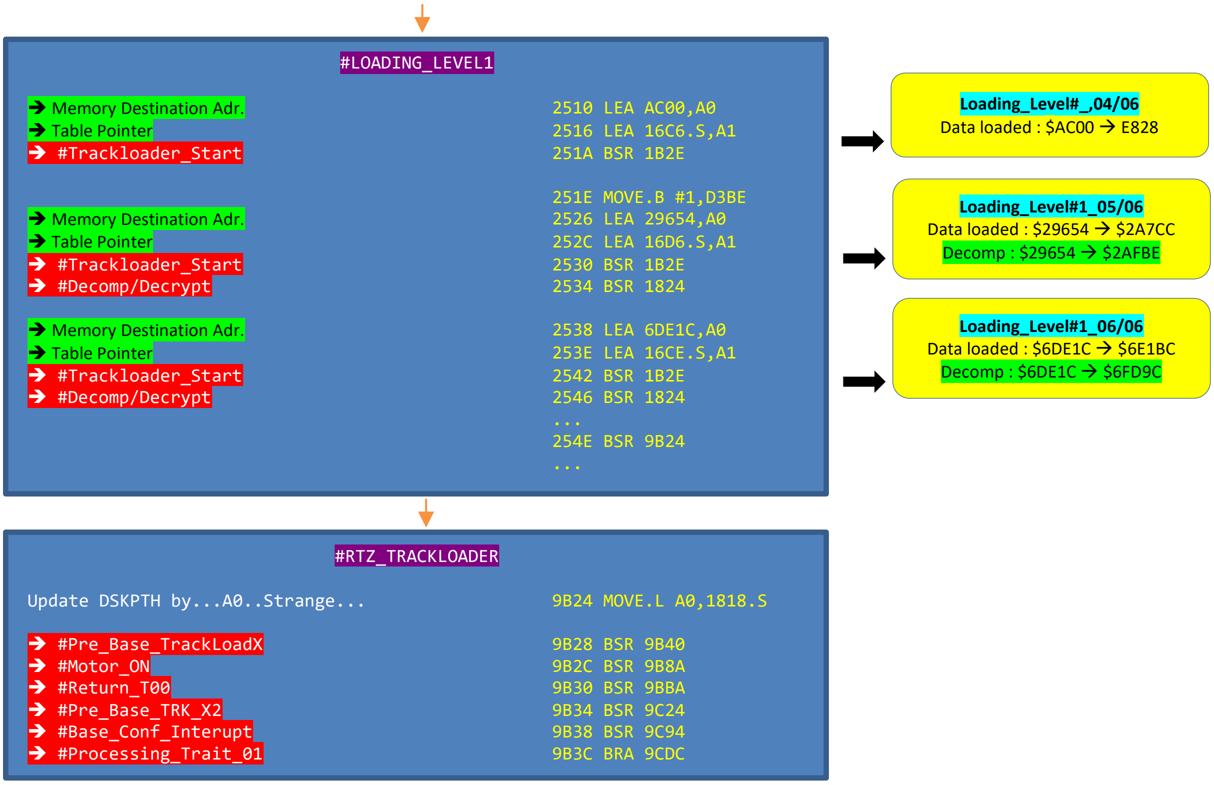
#LOADING_LEVEL1

```
                                     24CC MOVE.L #70400,C1A.S

→ Memory Destination Adr.            24D4 LEA 515DC,A0
→ Table Pointer                      24DA LEA 16AE.S,A1
→ #Trackloader_Start                 24D2 BSR 1B2E
→ #Decomp/Decrypt                    24E2 BSR 1824

→ Memory Destination Adr.            24E6 LEA 4D800,A0
→ Table Pointer                      24EC LEA 16B6.S,A1
→ #Trackloader_Start                 24F0 BSR 1B2E
→ #Decomp/Decrypt                    24F4 BSR 1824

→ Memory Destination Adr.            24FE LEA 2AFBE,A0
→ Table Pointer                      2504 LEA 16BE.S,A1
→ #Trackloader_Start                 2508 BSR 1B2E
→ #Decomp/Decrypt                    250C BSR 1824
```

Loading_Level#1_01/06
Data loaded : $515DC → $5E084
Decomp : $515DC → $641DC

Loading_Level#1_02/06
Data loaded : $4D800 → $4E684
Decomp : $4D800 → $50000

Loading_Level#1_03/06
Data loaded : $2AFBE → $38D86
Decomp : $2AFBE → $42000



Part 11 Disk call from the game. (ak : the tableau of death)

Congratulations to you if you are still here and if you have followed everything, which is not necessarily obvious because of the amount of information.

So... there are several ways to list all the *TrackLoader* calls and rip them.
The 1st one is to use command **FA 1B2E** and fill a table accordingly by analyzing the code around these calls.
The second, used below, is to use *Cheat-Modes* (Energie and money) to finish completly the game with BreakPoints well positioned.
namely, when calling the *TrackLoader*: **$1B2E**
At the output of the *TrackLoader*: **$1B36**

And at each call, look at the register values **A0** and **A1**
The Program *Counter (**PC**)*
The value in memory of the requested address.
The instructions that precede and follow the call to *1B2E*

TIPS : Breakpoints will interfere with the disk change routine during the game (which happens 2 or 3 times)
In this case, you will have to insert the requested disk in the drive and then enter the AR and jump directly to the code return.

If it is the **disk1** requested → type in: **G8D14**
If it is the **disk2** requested → type in: **G8D0A**

For more info, see the code in question above part 9 of the tutorial.

You should find the following table:

Disk2

Call Order	Disk Side	Call Addr	A0 Memory Addr	A1 Pos. Table	DSKPTH	(A1)+ LENGTH	(A1) Real Disk Position	Memory Loading area Address	Decomp/ Decrypt	Memory Decomp. Area Address	LENGTH Decomp	NFO
Load_Menu_10	D2-LOW	1E8E	641DC	1676	7C000	04C18	00000-04C17	641DC-68DF4	BSR 1824	641DC-6DE1C	9C40	Load_Menu_10/16 // Return_Menu_02/08
00	D2-LOW	8CF8	000B0	8D16	7C100 8000	00004	00000-00004	000B0-000B4	No	N/A	N/A	Check Signature Disk
Load_Menu_11	D2-LOW	1EA0	70400	167E	7C000	0271C	04C18-07333	70400-72B1C	BSR 1824	70400-794C4	090C4	Load_Menu_11/16, Return_Menu_03/08
Load_Menu_12 In_Game_50	D2-LOW	1CF0	0EB7E	1686	7C100 7C000	007FC	07334-07B2F	0EB7E-0F37A	BSR 1824	0EB7E-0F920	00DA2	Load_Menu_12/16, Back_CRYSTAL_CAVERNS_08/12 Return_Menu_04/08
Load_Menu_13 In_Game_51	D2-LOW	1D02	0F920	168E	7C100 7C000	0E9D8	07B30-16507	0F920-1F240	BSR 1824	0F920-26488	16B68	Load_Menu_13/16, Back _CRYSTAL_CAVERNS_09/12 Return_Menu_05/08
Load_Menu_14 In_Game_52	D2-LOW	1D14	26488	1696	7C100 7C000	01408	16508-1790F	26488-27890	BSR 1824	26488-27BC2	0173A	Load_Menu_14/16, Back _CRYSTAL_CAVERNS_10/12 Return_Menu_06/08
Load_Menu_15 In_Game_53	D2-LOW	1D26	27BC2	169E	7C100 7C000	00E10	17910-1871F	27BC2-289D2	BSR 1958	27BC2-28AB2	00EF0	Load_Menu_15/16, Back _CRYSTAL_CAVERNS_11/12 Return_Menu_07/08
Load_Menu_16 In_Game_54	D2-LOW	1D38	28AB2	16A6	7C100 7C000	00B48	18720-19267	28AB2-295FA	BSR 1958	28AB2-29654	00BA2	Load_Menu_16/16, Back _CRYSTAL_CAVERNS_12/12 Return_Menu_08/08 End Loading Phase1, display Menu of the game
Load_Level1_01	D2-LOW	24DE	515DC	16AE	70400	0CAA8	19268-25D0F	515DC-5E084	BSR 1824	515DC-641DC	12C00	Level#1_01/06
In_Game_61	D2-LOW	762C	515DC	16AE	7C100	0CAA8	19268-25D0F	515DC-5E084	BSR 1824	515DC-641DC	12C00	Return_Level#1_02/07 (Coming from WATER_VORTEX)
In_Game_55-RET	D2-LOW	7800	515DC	16AE	74300	0CAA8	19268-25D0F	515DC-5E084	BSR 1824	515DC-641DC	12C00	Level#1_return_from_1st_Bridge_on_the_right_01/04 (Coming from the old man's hut)
In_Game_31-A	D2-LOW	7966	515DC	16AE	74300	0CAA8	19268-25D0F	515DC-5E084	BSR 1824	515DC-641DC	12C00	Level#1_return_towards_Aggressive_Tree_01/04 (Coming from the left, from the shark bridge)
In_Game_44	D2-LOW	7C80	515DC	16AE	7C100	0CAA8	19268-25D0F	515DC-5E084	BSR 1824	515DC-641DC	12C00	Return_CRYSTAL_CAVERNS_02/12
Load_Level1_02	D2-LOW	24F0	4D800	16B6	70400	00E84	25D10-26B93	4D800-4E684	BSR 1824	4D800-50000	02800	Level#1_02/06
In_Game_63	D2-LOW	763E	4D800	16B6	7C100	00E84	25D10-26B93	4D800-4E684	BSR 1824	4D800-50000	02800	Return_Level#1_03/07 (Coming from WATER_VORTEX)
In_Game_45	D2-LOW	7C9A	4D800	16B6	7C100	00E84	25D10-26B93	4D800-4E684	BSR 1824	4D800-50000	02800	Return_CRYSTAL_CAVERNS_03/12
In_Game_56-RET	D2-LOW	7812	4D800	16B6	74300	00E84	25D10-26B93	4D800-4E684	BSR 1824	4D800-50000	02800	Level#1_return_from_1st_Bridge_on_the_right_02/04 (Coming from the old man's hut)
In_Game_12	D2-LOW	8A9C	4D800	16B6	74300	00E84	25D10-26B93	4D800-4E684	BSR 1824	4D800-50000	02800	Return_Level#2_04/06
Load_Level1_03	D2-LOW	2508	2AFBE	16BE	70400	0DDC8	26B94-3495B	2AFBE-38D86	BSR 1824	2AFBE-42000	17042	Level#1_03/06

Call Order	Disk Side	Call Addr	A0 Memory Addr	A1 Pos. Table	DSKPTH	(A1)+ LENGTH	(A1) Real Disk Position	Memory Loading area Address	Decomp/ Decrypt	Memory Decomp. Area Address	LENGTH Decomp	NFO
In_Game_46	D2-LOW	7CAC	2AFBE	16BE	7C100	0DDC8	26B94-3495B	2AFBE-38D86	BSR 1824	2AFBE-42000	17042	Return_CRYSTAL_CAVERNS_04/12
In_Game_57-RET	D2-LOW	7824	2AFBE	16BE	74300	0DDC8	26B94-3495B	2AFBE-38D86	BSR 1824	2AFBE-42000	17042	Level#1_return_from_1st_Bridge_on_the_right_03/04 (Coming from the old man's hut)
In_Game_31-B	D2-LOW	7980	2AFBE	16BE	74300	0DDC8	26B94-3495B	2AFBE-38D86	BSR 1824	2AFBE-42000	17042	Level#1_return_towards_Aggressive_Tree_02/04 (Coming from the left, from the shark bridge)
Load_Level1_04	D2-LOW	251A	0AC00	16C6	70400	03C28	3495C-38583	0AC00-0E828	No	N/A	N/A	Level#1_04/06
In_Game_58-RET	D2-LOW	7836	0AC00	16C6	74300	03C28	3495C-38583	0AC00-0E828	No	N/A	N/A	Level#1_return_from_1st_Bridge_on_the_right_04/04 (Coming from the old man's hut)
In_Game_31-C	D2-LOW	7992	0AC00	16C6	74300	03C28	3495C-38583	0AC00-0E828	No	N/A	N/A	Level#1_return_towards_Aggressive_Tree_03/04 (Coming from the left, from the shark bridge)
In_Game_48	D2-LOW	7CD0	0AC00	16C6	7C100	03C28	3495C-38583	0AC00-0E828	No	N/A	N/A	Return_CRYSTAL_CAVERNS_06/12
Load_Level1_06	D2-LOW	2542	6DE1C	16CE	70400	003A0	38584-38923	6DE1C-6E1BC	BSR 1824	6DE1C-6FD9C	01E80	Level#1_06/06
In_Game_47	D2-LOW	7CBE	6DE1C	16CE	7C100	003A0	38584-38923	6DE1C-6E1BC	BSR 1824	6DE1C-6FD9C	01E80	Return_CRYSTAL_CAVERNS_05/12
In_Game_30	D2-LOW	8244	6DE1C	16CE	641DE	003A0	38584-38923	6DE1C-6E1BC	BSR 1824	6DE1C-6FD9C	01E80	Return_Level4_02/02
In_Game_22	D2-LOW	84C2	6DE1C	16CE	641DE	003A0	38584-38923	6DE1C-6E1BC	BSR 1824	6DE1C-6FD9C	01E80	Return_Level#3_Cage_Lift_02/02
In_Game_14	D2-LOW	8AC0	6DE1C	16CE	74300	003A0	38584-38923	6DE1C-6E1BC	BSR 1824	6DE1C-6FC9C	01E80	Return_Level#2_06/06
In_Game_65	D2-LOW	7668	6DE1C	16CE	7C100	003A0	38584-38923	6DE1C-6E1BC	BSR 1824	6DE1C-6FC9C	01E80	Return_Level#1_05/07 (Coming from WATER_VORTEX)
Load_Level1_05	D2-LOW	2530	29654	16D6	70400	01178	38924-39A9B	29654-2A7CC	BSR 1824	29654-2AFBE	0196A	Level#1_05/06
In_Game_49	D2-LOW	7CDE	29654	16D6	7C100	01178	38924-39A9B	29654-2A7CC	BSR 1824	29654-2AFBE	0196A	Return_CRYSTAL_CAVERNS_07/12
In_Game_31-D	D2-LOW	79A0	29654	16D6	74300	01178	38924-39A9B	29654-2A7CC	BSR 1824	29654-2AFBE	0196A	Level#1_ return_towards_Aggressive_Tree_04/04 (Coming from the left, from the shark bridge)
In_Game_18	D2-LOW	7A3C	291D2	16D6	74300	01178	38924-39A9B	291D2-2A34A	BSR 1824	29654-2AB3C	014E8	Level#1_After_block_which_falls_04/04 (to the left, after the enemy 'stone-pusher)
In_Game_10	D2-LOW	8A70	515DC	16DE	74300	04F08	39A9C-3E9A3	515DC-564E4	BSR 1824	515DC-5A5DC	09000	Return_Level#2_02/06
In_Game_01	D2-LOW	719A	515DC	16DE	74300	04F08	39A9C-3E9A3	515DC-564E4	BSR 1824	515DC-5A5DC	09000	Level#1_Aggressive_Tree_01/04
In_Game_02	D2-LOW	71AC	5A5DC	16E6	74300	054A0	3E9A4-43E43	5A5DC-5FA7C	BSR 1824	5A5DC-641DC	09C00	Level#1_Aggressive_Tree_02/04 (to_the_left, to shark bridge)
In_Game_11	D2-LOW	8A82	5A5DC	16E6	74300	054A0	3E9A4-43E43	5A5DC-5FA7C	BSR 1824	5A5DC-641DC	09C00	Return_Level#2_03/06
In_Game_29	D2-LOW	8232	5A5DC	16E6	641DE	054A0	3E9A4-43E43	5A5DC-5FA7C	BSR 1824	5A5DC-641DC	09C00	Return_Level4_01/02

Call Order	Disk Side	Call Addr	A0 Memory Addr	A1 Pos. Table	DSKPTH	(A1)+ LENGTH	(A1) Real Disk Position	Memory Loading area Address	Decomp/ Decrypt	Memory Decomp. Area Address	LENGTH Decomp	NFO
In_Game_21	D2-LOW	84B0	5A5DC	16E6	641DE	054A0	3E9A4-43E43	5A5DC-5FA7C	BSR 1824	5A5DC-641DC	09C00	Back _Level#3_Cage_Lift_01/02
In_Game_23	D2-LOW	7AD0	2D2E0	16EE	74300	0B26C	43E44-4F0AF	2D2E0-3854C	BSR 1824	2D2E0-42000	14D20	Level#1_Return_block_which_falls_01/02
In_Game_13	D2-LOW	8AAE	2D2E0	16EE	74300	0B26C	43E44-4F0AF	2D2E0-3854C	BSR 1824	2D2E0-42000	14D20	Return_Level#2_05/06 (Coming from the left)
In_Game_03	D2-LOW	71C6	2D2E0	16EE	74300	0B26C	43E44-4F0AF	2D2E0-3854C	BSR 1824	2D2E0-42000	14D20	Level#1_Aggressive_Tree_03/04 (to_the_left, to shark bridge)
In_Game_04	D2-LOW	71D8	0AA00	16F6	74300	03A34	4F0B0-52AE3	0AA00-0E434	BSR 8C0A No	N/A	N/A	Level#1_Aggressive_Tree_04/04 (to_the_left, to shark bridge)
In_Game_24	D2-LOW	7AE2	0AA00	16F6	74300	03A34	4F0B0-52AE3	0AA00-0E434	No	N/A	N/A	Level#1_Return_block_which_falls_02/02
In_Game_15	D2-LOW	7A0A	32A4C	16FE	74300	07DC8	52AE4-5A8AB	32A4C-3A814	BSR 1824	32A4C-42000	0F5B4	Level#1_After_block_which_falls _01/04 (Coming from the left)
In_Game_16	D2-LOW	7A1C	2AB3C	1706	74300	041C8	5A8AC-5EA73	2AB3C-2ED04	BSR 1824	2AB3C-32A4C	07F10	Level#1_After_block_which_falls_02/04 (to the left, after the 'stone-pusher' enemy)
In_Game_17	D2-LOW	7A2E	0A300	170E	74300	045EC	5EA74-6305F	0A300-0E8EC	No	N/A	N/A	Level#1_After_block_which_falls_03/04 (to the left, after the enemy 'stone-pusher')
In_Game_19	D2-LOW	8360	5A5DC	1716	641DE	0353C	63060-6659B	5A5DC-5DB18	BSR 1824	5A5DC-5DBE7	0360B	Level#3_Cage_Lift_01/02
In_Game_28	D2-LOW	8150	6DE1C	171E	641DE	00588	6659C-66B23	6DE1C-6E3A4	BSR 1824	6DE1C-6FD9C	01E80	Level#4_After_Password_of_Barloom_Projection_04/04
In_Game_20	D2-LOW	8372	6DE1C	171E	641DE	00588	6659C-66B23	6DE1C-6E3A4	BSR 1824	6DE1C-6FD9C	01E80	Level#3_Cage_Lift_02/02
In_Game_25	D2-LOW	811A	5A5DC	1726	641DE	01CE4	66B24-68807	5A5DC-5C2C0	BSR 1824	5A5DC-5D5DC	03000	Level#4_After_Password_of_Barloom_Projection_01/04
In_Game_26	D2-LOW	812C	5E1DC	172E	641DE	001E0	68808-689E7	5E1DC-5E3BC	BSR 1824	5E1DC-5E7DC	00600	Level#4_After_Password_of_Barloom_Projection_02/04
In_Game_27	D2-LOW	813E	61DDC	1736	641DE	00B5C	689E8-69543	61DDC-62938	BSR 1824	61DDC-63D5C	01F80	Level#4_After_Password_of_Barloom_Projection_03/04
In_Game_05 In_Game_09-RET In_Game_59 In_Game_60	D2-LOW	7E68	30000	173E	74300 7C100	02EEC	69544-6C42F	30000-32EEC	BSR 1826	641DC-6A5DC	06400	Enter_House_GoTo_Level#2_01/04, Return_Level#2_01/06 CRYSTAL_CAVERNS_01/11 (Coming from Down Stairs & I/O on Disk1) Level_6_01/04 (Water Vortex) & Return_Level#1_01/07
In_Game_06	D2-LOW	8918	515DC	1746	74300	02C44	6C430-6F073	515DC-54220	BSR 1824	515DC-58C5D	07681	Enter_House_GoTo_Level#2_02/04
In_Game_07	D2-LOW	892A	5F9DC	174E	74300	00B70	6F074-6FBE3	5F9DC-6054C	BSR 1824	5F9DC-641DC	04800	Enter_House_GoTo_Level#2_03/04
In_Game_08	D2-LOW	893C	4E0C0	1756	74300	0011C	6FBE4-6FCFF	4E0C0-4E1DC	BSR 1824	4E0C0-4E700	00640	Enter_House_GoTo_Level#2_04/04

Disk2

Call Order	Disk Side	Call Addr	A0 Memory Addr	A1 Pos. Table	DSKPTH	(A1)+ LENGTH	(A1) Real Disk Position	Memory Loading area Address	Decomp/ Decrypt	Memory Decomp. Area Address	LENGTH Decomp	NFO
In_Game_54b	D1-LOW	7E68	30000	175E	7C100	4228	6FD00-73F87	30000-34288	BSR 1826	641DC-6A5DC	6400	Enter Vortex
In_Game_60	D1-LOW	74FA	5955C	1766	74300	04228	73F88-781AF	5955C-5D784	BSR 1824	5955C-641DC	0AC80	Level_6_02/04
In_Game_61	D2-LOW	750C	4E840	176E	74300	001D0	781B0-7837F	4E840-4EA10	BSR 1824	4E840-4FA50	01210	Level_6_03/04
In_Game_62	D2-LOW	751E	6DE1C	1776	74300	00628	78380-789A7	6DE1C-6E444	BSR 1824	6DE1C-6FE5C	02040	Level_6_04/04
						D35C	789A8-85D03			UNUSED AREA ~52Ko – Specific change of Side		
In_Game_67	D2-UP	768C	2E6EC	177E	7C100	0B00C	85D04-90D0F	2E6EC-396F8	BSR 1824	2E6EC-42000	13914	Return_Level#1_07/07 (Coming from WATER_VORTEX)
In_Game_57	D2-UP	709E	2E6EC	177E	74300	0B00C	85D04-90D0F	2E6EC-396F8	BSR 1824	2E6EC-42000	13914	Level#1_Right_bridge 03/04 (to old man hut)
In_Game_58	D2-UP	70B0	0AC00	1786	74300	030D4	90D10-93DE3	0AC00-0DCD4	No	N/A	N/A	Level#1_Right_bridge 04/04 (to old man hut)
In_Game_56	D2-UP	708C	4D800	178E	74300	00478	93DE4-9425B	4D800-4DC78	BSR 1824	4D800-4F740	01F40	Level#1_Right_bridge 02/04 (to old man hut)
In_Game_66	D2-UP	767A	4D800	178E	7C100	00478	93DE4-9425B	4D800-4DC78	BSR 1824	4D800-4F740	01F40	Return_Level#1_06/07 (Coming from WATER_VORTEX)
In_Game_55	D2-UP	7074	38000	1796	74300	05914	9425C-99B6F	38000-3D914	BSR 1826	55DDC-5F85C	09A80	Level#1_Right_bridge 01/04 (to old man hut)
In_Game_64	D2-UP	7650	6A5DC	1796	7C100	05914	9425C-99B6F	6A5DC-6FEF0	BSR 1826	55DDC-5BF50	06174	Return_Level#1_04/07 (Coming from WATER_VORTEX)
N/A	D2-UP	78E4	38000	1796		05914	9425C-99B6F	38000-3D914	BSR 1826	55DDC-5F85C	9A80	Never called. Position in $2BE map, Never reached because castle appears and prevents from going to this position. (10 steps further than the castle entrance)
END_01	D2-UP	4F7E	0F0C6	179E	7C100	00308	99B70-99E77	0F0C6-0F3CE	BSR 1824	0F0C6-0F57C	04B6	Quit_Game_01/04
END_02	D2-UP	4F90	0F57C	17A6	7C100	32A84	99E78-CC8FB	0F57C-42000	No	N/A	N/A	Quit_Game_02/04
END_03	D2-UP	5010	70400	17AE	6AFE0	0F778	CC8FC-DC073	70400-7FB78	No	N/A	N/A	Quit_Game_03/04
END_04	D2-UP	501E	641DC	17B6	6AFE0	06E04	DC074-E2E77	641DC-6AFE0	BSR 1958	641DC-6DE1B	09C3F	Quit_Game_04/04
In_Game_68	D2-UP	78B0	58DDC	17BE	7C100	04C98	E2E78-E7B0F	58DDC-5DA74	BSR 1824	58DDC-6405C	0B280	CASTLE_01/01 (Reveals the entrance of the castle)
Load_Menu_09	D2-UP	1D5E	002A8	17C6	7C000	00060	E7B10-E7B6F	002A8-00308	BSR 1824	002A8-0043C	00194	Load_Menu_09/16 & Return_Menu_01/08
In_Game_69 [END]	D2-UP	85EE	515DC	17CE	74300	04520	E7B70-EC08F	515DC-55AFC	BSR 1824	515DC-5B21C	09C40	END_GAME_01/01
Load_Menu_01	D2-UP	1C66	46FB4	17D6	18000	03A90	EC090-EFB1F	46FB4-4AA44	BSR 1824	46FB4-4D800	0684C	Load_Menu_01/16
Load_Menu_02	D2-UP	1C78	70000	17DE	18000	00250	EFB20-EFD6F	70000-70250	BSR 1824	70000-70390	00390	Load_Menu_02/16

Call Order	Disk Side	Call Addr	A0 Memory Addr	A1 Pos. Table	DSKPTH	(A1)+ LENGTH	(A1) Real Disk Position	Memory Loading area Address	Decomp/ Decrypt	Memory Decomp. Area Address	LENGTH Decomp	NFO
Load_Menu_03	D2-UP	1C8A	45F34	17E6	18000	00438	EFD70-F01A7	45F34-4636C	BSR 1824	45F34-46AD4	00BA0	Load_Menu_03/16
Load_Menu_04	D2-UP	1C9C	50000	17EE	18000	00124	F01A8-F02CB	50000-50124	BSR 1824	50000-5019C	0019C	Load_Menu_04/16
Load_Menu_05	D2-UP	1CAE	515DC	17F6	18000	00664	F02CC-F092F	515DC-51C40	BSR 1826	42000-4324B	0124B	Load_Menu_05/16
Load_Menu_06	D2-UP	1CC6	50398	17FE	18000	00268	F0930-F0B97	50398-50600	BSR 1824	50398-509D7	0063F	Load_Menu_06/16
Load_Menu_07	D2-UP	1CD8	457A4	1806	18000	003FC	F0B98-F0F93	457A4-45BA0	BSR 1824	457A4-45F34	00790	Load_Menu_07/16
Load_Menu_08	D2-UP	1E70	4324C	180E	18000	01EBC	F0F94-F2E4F	4324C-45108	No	N/A	N/A	Load_Menu_08/16

Disk1

Call Order	Disk Side	Call Addr	A0 Memory Addr	A1 Pos. Table	DSKPTH	(A1)+ LENGTH	(A1) Real Disk Position	Memory Loading area Address	Decomp/ Decrypt	Memory Decomp. Area Address	LENGTH Decomp	NFO
Anim part #01	D1-LOW	60A	16B14	580	763C0	46F74	0189C-4880F	16B14-5DA88	BSR 6B4	N/A	N/A	Animation Part #01 of Shadow Of The Beast II
Anim Psygnosis	D1-LOW	70044	45610	N/A	6A000	24284	48810-6CA93	45610-69894	BSR 70310	002B0-28364	280B4	Animation Psygnosis
Last_Load_Disk1	D1-LOW	65A	40000	590	50000	9B78	6CA94-7660B	40000-49B78	N/A	N/A	N/A	Last code loaded after end of animation
In_Game_37	D1-LOW	72F0	0A800	161E	7C100	04074	76FC0-7B033	0A800-0E874	No	N/A	N/A	CRYSTAL_CAVERNS_06/11 (Comming From Down Stairs)
						ACCF	7B034-85D03					UNUSED AREA End of Disk
Anim part #02	D1-UP	62A	16B14	588	6C780	48DDE	85D04-CEAE1	16B14-5F8F2	BSR 6B4	N/A	N/A	Animation Part #02 of Shadow Of The Beast II
						1476	CEAE1-CFF57					UNUSED AREA ~5Ko
In_Game_43	D1-UP	7C44	31F40	1626	7C100 74300	0280C	CFF58-D2763	31F40-3474C	No	N/A	N/A	Return_CRYSTAL_CAVERNS_01/12 (Then will ask to insert the Disk2)
In_Game_32	D1-UP	7E68	30000	1626	7C100	0280C	CFF58-D2763	30000-3280C	BSR 1826	641DC-6A5DC	06400	CRYSTAL_CAVERNS_01/11 (Coming from Down Stairs)
In_Game_33	D1-UP	72A8	515DC	162E	7C100	0C3C8	D2764-DEB2B	515DC-5D9A4	BSR 1824	515DC-641DC	12C00	CRYSTAL_CAVERNS_02/11 (Coming from Down Stairs)
In_Game_34	D1-UP	72BA	4E700	1636	7C100	00524	DEB2C-DF04F	4E700-4EC24	BSR 1824	4E700-4F600	00F00	CRYSTAL_CAVERNS_03/11 (Coming from Down Stairs)
In_Game_35	D1-UP	72CC	6DE1C	163E	7C100	0151C	DF050-E056B	6DE1C-6F338	BSR 1824	6DE1C-6FC9C	01F80	CRYSTAL_CAVERNS_04/11 (Coming from Down Stairs)
In_Game_36	D1-UP	72DE	2E248	1646	7C100	0A73C	E056C-EACA7	2E248-38984	BSR 1824	2E248-42000	13DB8	CRYSTAL_CAVERNS_05/11 (Coming from Down Stairs)
In_Game_38	D1-UP	7308	0F38A	164E	7C100	007F0	EACA8-EB497	0F38A-FB7A	BSR 1824	0F38A-1013A	00DB0	CRYSTAL_CAVERNS_07/11 (Coming from Down Stairs)
In_Game_39	D1-UP	731A	1013A	1656	7C100	10410	EB498-FB8A7	1013A-2054A	BSR 1824	1013A-2AF4A	1AE10	CRYSTAL_CAVERNS_08/11 (Coming from Down Stairs)
In_Game_40	D1-UP	732C	2AF4A	165E	7C100	01828	FB8A8-FD0CF	2AF4A-2C772	BSR 1958	2AF4A-2C7B6	0186C	CRYSTAL_CAVERNS_09/11 (Coming from Down Stairs)
In_Game_41	D1-UP	733E	2C7B6	1666	7C100	00E10	FD0D0-FDEDF	2C7B6-2D5C6	BSR 1958	2C7B6-2D6A6	00EF0	CRYSTAL_CAVERNS_10/11 (Coming from Down Stairs)
In_Game_42	D1-UP	7350	2D6A6	166E	7C100	00B48	FDEE0-FEA27	2D6A6-2E1EE	BSR 1958	2D6A6-2E247	00BA1	CRYSTAL_CAVERNS_11/11 (Coming from Down Stairs)

Part 12a Rip Full Disk1

Rip Disk1 Side Lower :

The disk 1 also uses a custom *MFM format* (see part 2 IPF image analysis)

Question : Is it really useful to rip the whole Disk1 ?
We know that it is mainly used for the animations at the beginning of the game, so we will use another method to rip these.

*(we have already ripped the Psygnosis animation so we will only have 2 more animations to rip)

But for now, let's deal with the complete disk Side by Side (The *Trackloader* works by Side)

The disk 1 is bootable, so the Track 0.0 is in standard 'AmigaDos' format and cannot be read by our Custom Trackloader.
So we have to rip : **(!80-1)*!6300 = $79824 = !497 700 Bytes**

Review of the Functioning of the SOTB2 Custom Trackloader:

A0 = Memory Destination
A1 = Adr in the table (and change the length in the table)

The custom *trackloader* buffer is located in **$6A000**

On an A500, the potential free memory area with margin is therefore : **$1000 → $69000 = $68000** '425 984 Bytes'
This will not be enough for a complete rip of a side, let's see how many 'custom tracks' can be read in '425 984 Bytes

!425984 / !6300 = !67
!67 * !6300 = $68000 = !422 100 Bytes

Let's Rock'n roll
Just boot on the original Disk1 of SOTB2, **enter** in the AR **during the** trackload (before displaying the 1st animation)

Type in :
```
A 7003A
$7003A  LEA     $1000,A0                ; Memory Destination

A 70048
$70048  MOVE.W  #F0,DFF180              ; → Green screen at the end of the trackload
$70050  NOP                             ;   Wait
$70052  MOVE.W  #0,DFF180               ;   Black screen
$7005A  BRA     70048                   ; ← Dead loop
```

Then, type :
```
M 70060       Put     00 00 18 9C     ; physical addr start on disk (Start pos.+1 track Custom)*
M 70064       Put     00 06 80 00     ; Length to read
```

* As explained above, we start at the 1st Track Custom which is the second track of the side.
The 1st track is non-custom because it is the boot track, it is in AmigaDOS format.

And we start the *trackload*, **Type in** : **G 7003A**

Trackload of Tracks from 01 → 68 without any problem and we reach our background in green, the trackload is finished.
Enter in the AR.

We save everything on a new blank disk inserted in DF1.
Type : SM 1:Disk1_lower_00, 1000 1000+68000

We do the rest :
```
Total to rip       : $79824
Already riped      : $68000
Remains            : $79824-$68000    = $11824
```

Type :
```
M 70060       Put     00 06 98 9C     ; physical addr start on ($189C+$68000)
M 70064       Put     00 01 18 24     ; Length to read
```

And we start our *trackloader*, **Type in** : **G 7003A**
Trackload of the last Tracks without any problem and we arrive on our background in green, the trackload is finished.
Enter in the AR.

SM 1:Disk1_lower_01, 1000 1000+11824

We join this two files :
Remove the disk in DF1 and insert it in DF0 in place of the Shadow Of The Beast Original Disk N°1.

```
O 00, 1000 80000                    ; Small cleaning.
LM Disk1_lower_00, 1000             ; Rip D1 Lower phase 1
LM Disk1_lower_01, 69000            ; Rip D1 Lower phase 2
Format SOTB_Disk1_lower             ; Format a new floppy disk if necessary
SM Disk1_Lower.bin,1000 7A824       ; Full Backup of 'side Lower 1st floppy'
```

It remains to make the other side.

Rip Disk1 Side Upper :

On the second side there is no AmigaDOS track BUT there is our famous protection/signature track.
So, it's ripping exactly the same amount of data on that side.

reminder of how the code works for the selection of Side, **Upper** or **Lower** = $08 44 68

```
701FA CMP.L #$84468, D0        ; Compare D0 with $84468
70200 BGE.B $70206             ; If greater           then  GoTo #DF0__SIDE_UP__MOTOR_ON__DIR_EXT
70202 BSR.B $701A6             ; otherwise                  GoTo #DF0__SIDE_DOWN__MOTOR_ON__DIR_EXT
```

The 1st Track in Side Upper starts at the address $84468
* We will start at the second track as explained above so at the address : $84468 + $189C = $85D04

Let's Go
Boot on the original Disk1 of SOTB2, **enter** in the AR **during the** trackload (before displaying the 1st animation)

Type :

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

A 70048
$70048  MOVE.W  #F0,DFF180            ; → Green screen at the end of the trackload
$70050  NOP                           ;    Wait
$70052  MOVE.W  #0,DFF180             ;    Black Screen
$7005A  BRA     70048                 ; ← Dead loop
```

Then, type :

```
M 70060      Put      00 08 5D 04   ; physical adr of 'starting address' on disk. *( see explanation above)
M 70064      Put      00 06 80 00   ; Length to read
```

And we launch the *trackload*, **Type in** : **G 7003A**

Trackload of Tracks from 01 → 68 without any problem and we reach our green background, the trackload is finished.
Enter in the AR.

We save everything on a new blank disk inserted in DF1.
Type in : SM 1:Disk1_upper_00, 1000 1000+68000

We do what is left :

```
Total to rip      : $79824 (79 Tracks of $189C, see above)
Already Riped     : $68000
Remains           : $79824-$68000   = $11824
```

Type in :

```
M 70060      Put      00 0E DD 04   ; physical addr start on ($85D04+$68000)
M 70064      Put      00 01 18 24   ; Length to read
```

We launch the *trackload*, **Type in** : **G 7003A**
Trackload of the last Tracks without any problem and we reach our green background, the *trackload* is finished.
Enter in the AR.

SM 1:Disk1_upper_01, 1000 1000+11824

We join this two files :

Remove the disk in DF1 and insert it in DF0 in place of the Shadow Of The Beast Original Disk N°1.

```
O 00, 1000 80000                    ; Small cleaning.
LM Disk1_upper_01, 1000             ; Rip D1 Upper phase 1
LM Disk1_upper_02, 69000            ; Rip D1 Upper phase 2
Format SOTB2_Disk1_upper            ; Format a new floppy disk if necessary
SM Disk1_Upper.bin,1000 7A824       ; Full Backup of 'side Upper 1st floppy'
```

Part 12b Rip Full Disk2

We find exactly the same *MFM custom* format on disk2, we will always use the same method to rip its data.

Rip Disk2 Side Lower :

The disk 2 is not bootable, so the Track 0.0 is not in standard 'AmigaDos' format but in a *custom MFM format*
This gives us a total of track to rip:

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

Boot on the original Disk1 of SOTB2, **enter** in the AR **during the** trackload (before displaying the 1st animation)
Swap in the internal drive the disk1 by the original disk2 of SOTB2 and **insert a new blank disk in the external drive.**

Type :
```
A 7003A
$7003A  LEA     $1000,A0              ; Memory Destination

A 70048
$70048  MOVE.W  #F0,DFF180            ; → Green screen at the end of the trackload
$70050  NOP                           ;   Wait
$70052  MOVE.W  #0,DFF180             ;   Black Screen
$7005A  BRA     70048                 ; ← Dead loop
```

Then
```
M 70060        Put     00 00 00 00    ; physical adr of 'starting address' on disk.
M 70064        Put     00 06 80 00    ; Length to read
```

We launch the *trackload*, **Type in**: **G 7003A**

Trackload of Tracks from 01 → 68 without any problem and we reach our green background, the trackload is finished.
Enter in the AR.
We save everything on our new blank disk previously inserted in DF1.
Type in : SM 1:Disk2_lower_01, 1000 1000+68000

We do what is left :

```
Total to rip      : $7B0C0
Already Riped     : $68000
Remains           : $7B0C0-$68000   = $130C0
```

Type in :
```
M 70060        Put     00 06 80 00    ; physical adr of 'starting address' on disk.
M 70064        Put     00 01 30 C0    ; Length to read
```

We launch the *trackload*, **Type in**: **G 7003A**
Trackload of the last Tracks without any problem and we reach our green background, the *trackload* is finished.

Enter in the AR.
Type in : SM 1:Disk2_lower_02, 1000 1000+130C0

We join this two files :
Remove the disk in DF1 and insert it in DF0 in place of the Shadow Of The Beast Original Disk N°2.

```
O 00, 1000 80000                    ; Small cleaning.
LM Disk2_lower_01, 1000             ; Rip D2 Lower phase 1
LM Disk2_lower_02, 69000            ; Rip D2 Lower phase 2
Format SOTB2_Disk2_lower            ; re-format the floppy
SM Disk2_Lower.bin,1000 7A824       ; Full Backup of 'side Lower 2nd floppy'
```

Rip Disk2 Side Upper :

So....normally, 1 disk = 80 cylinder (0 to 79)
Except that we have the protection/signature track: Cylinder 0.1 (see code above or web analyzer)
This gives us 79 tracks to rip on the **'Upper'** side
!79*!6300 = *$79824* = **!497 700 Bytes**

The 'base' for 'trackloading' in **Upper** Side is *$84468* (see code above);

We add 1 Custom Track to it so that the trackloader works and starts reading on the second Track of the side upper.
So **$84468+$189C=***$85D04*, is the value to put in the table in $70060 as *'Address_Start_raw'* and as before, we rip in two phases.

Boot on the original Disk1 of SOTB2, **enter** in the AR **during the** trackload (before displaying the 1st animation)
Swap in the internal drive the disk1 by the original disk2 of SOTB2 and **insert a new blank disk in the external drive.**

Type in :

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

A 70048
$70048  MOVE.W  #F0,DFF180              ; → Green screen at the end of the trackload
$70050  NOP                             ;   Wait
$70052  MOVE.W  #0,DFF180               ;   Black screen
$7005A  BRA     70048                   ; ← Dead loop
```

Then

```
M 70060        Put     00 08 5D 04      ; physical adr of 'starting address' on disk.
                                          (limit+1 track)=84468+189C=85D04)
M 70064        Put     00 06 80 00      ; Length to Read
```

We launch the *trackload*, **Type in**: **G 7003A**
Trackload of Tracks from 01 → 68 without any problem and we reach our green background, the trackload is finished.
Enter in the AR.

We save the whole thing on our new blank disk inserted in DF1.
Type in :SM 1:Disk2_Upper_00, 1000 1000+68000

We do what is left:

```
Total to rip      : $79824 (79 Tracks of $189C, see above)
Already Riped     : $68000
Remains           : $79824-$68000   = $11824
```

Type in :

```
M 70060        Put     00 0E DD 04      ; physical adr of 'starting address' on disk. ($85D04+$68000)
M 70064        Put     00 01 18 24      ; Length to Read
```

We launch the *trackload*, **Type in** : **G 7003A**
Trackload of the last Tracks without any problem and we arrive on our green background, the *trackload* is finished.
Enter in the AR.

SM 1:Disk2_upper_01, 1000 1000+11824

We join this two files :
Remove the disk in DF1 and insert it in DF0 in place of the Shadow Of The Beast Original Disk N°2.

```
O 00, 1000 80000                        ; Small cleaning.
LM Disk2_upper_00, 1000                 ; Rip D2 Upper phase 1
LM Disk2_upper_01, 69000                ; Rip D2 Upper phase 2
Format SOTB2_Disk2_upper                ; re-format the floppy
SM Disk2_Upper.bin,1000 7A824           ; Full Backup of 'side Upper 2nd floppy'
```

Part 13 Rip Disk1 <FILE>

Animation RIP Part 01 & 02 and the last piece of code

Reboot your Amiga with Original Disk1in the internal floppy drive and a new floppy disk (yes again) in the external Floppy Reader.
Enter in your AR during animations of psygnosis.

We have already noted all the values (review Part 10 Diagram Part #01 (aka: the scary organization chart...))

But hey, just for fun:
Take a *breakpoint* just before the decompression phase:

Type in : **BS 6B4** then, return to the game code with : **X**

Our *breakpoint* is reached once the trackload is finished and we automatically enter the AR and as expected we will look at our values..
Type in: **R** and note **A0** which is our **address Data source**

Then we look at the address pointed to in **A1** to know the **length to save**.

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

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

And we save everything on our blank disk in **DF1**, **type in : SM 1:Anim_01, 16B14 16B14+46F74**

We delete our *breakpoint*, **type in** : **BDA** and return to the game code with : **X**

The original decompression is done and the 1st part of the animation is played.
Enter in the AR, we take advantage of it to put our two *breakpoints*.
Type in : BS 634 & BS 65A et and return to the game code with : **X**

Our *breakpoint* is reached once the trackload is finished and we automatically enter the AR and as expected we will look at our values.
Type in : **R** and we look at the same thing as before.

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

We save everything on our blank disk in **DF1**, **type in : SM 1:Anim_02, 16B14 16B14+48DDE**
Ok ? Let's time to return to the game code with : **X**

The animation continues and the last Trackload arrives and finally, our last *breakpoints* is reached, we automatically enter the AR.
Type in : **R** and we look at the same thing as before with a variant on the **memory address containing** the **length** (review code analysis)

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

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

We save everything on our floppy disk always present in DF1, **type in : SM 1:Last_Load, 40000 40000+9B78**

Part 13b Rip Disk2 < file by file >

There is another, shall we say, perfect possibility.
The file rip by setting Breakpoints for each TrackLoader call (as seen above)

Example, we start on the message 'insert Disk2' just after the end of the animations.
Reminder : Adr of the trackloader *$1B2E*

See analyse of **Decomp_Decrypt** :
A0=Source adr.
A1=Destination adr. ; But which is skewed by the 1st order in 01824 MOVE.L A0, A1
It will decompress at the same place as the source (it works in reverse so no worries for it)

We will use the call: **$1E8E**

```
1E96    LEA     70400,A0        ; Address of destination
1E9C    LEA     1676.S,A1       ; Position in the tableau        (Length data=271C)       2 tracks read
1EA0    BSR     1B2E            ; GoTo #TrackLoader
1EA4    BSR     1824            ; GoTo #Decomp_Decrypt
1EA8    ...
```

Value of **DSKPTH** is in *$18000*we leave ourselves a good step, we will use the memory from **$30000**

```
R A0 30000
O "PaTtErN", 30000 50000        ; Fill the destination area with a pattern
BS 1EA4
G 1E9C
```

The Loading is performed and our breakpoint is reached.
We save it all:

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

```
1st                         LongWord = 00 00 27 1D          ; Length of the compressed area
last                        LongWord = 00 1C 9B 46          ;
Second to last              LongWord = 00 00 38 D9          ; A2 before adding
Before before last.         LongWord = 98 D7 B8 1D          ; D5
Before before before last LongWord = 00 0C 94 07            ; D0
```

Let's uncrypt it all :

```
BDA
BS 1EA8                         ; BreakPoint at the end of decomp_decryt
R A0 30000                      ; Should already be at this value
X
```

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

End of data in 390C4, which gives us a length of **390C4-30000 = $90C4**

We save it all:

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

10012 Beast2_4C18
37060 Beast2D_4C18

It is also possible to automate the rip, there is a Textbox in *$2D90*, we can use it for now:

```
2D90    MOVE.W  #$00F,$DFF180   ; Blue background, we note A1
2D98    BTST    #6,$BFE001      ; Left Button
2DA0    BNE     2D90
2DA2    BSR     1B38
2DA6    MOVE.W  #$0F0,$DFF180   ; Green background, we save the data not decompressed
2DAE    BTST    #A,$DFF016      ; Right Button
2DB6    BNE     2DA6
2DB8    MOVE.W  #0,$DFF180      ; Return black screen
2DC0    RTS


1B30    BSR     1B38            ; to be replaced by a bsr 2D90     (original opcode=61 00 00 06 20)
1B34    MOVEA.L (A7)+,A0
1B36    RTS

8CE4    NOP                     ; Disable waiting for fire to be pressed for the insertion of the Disk2
```

(Reminder : The installation of the BS will cancel the signature check routine of the disk....(see above))
We put a BS in *1840* to read **A2** which is the end address of the compressed data (if we go through the decompression routine of course)

Blue background (left button)
Start trackloader routine, **look at A0** and see what it corresponds to in the table.
(or look directly in the Last_Load_Original file)

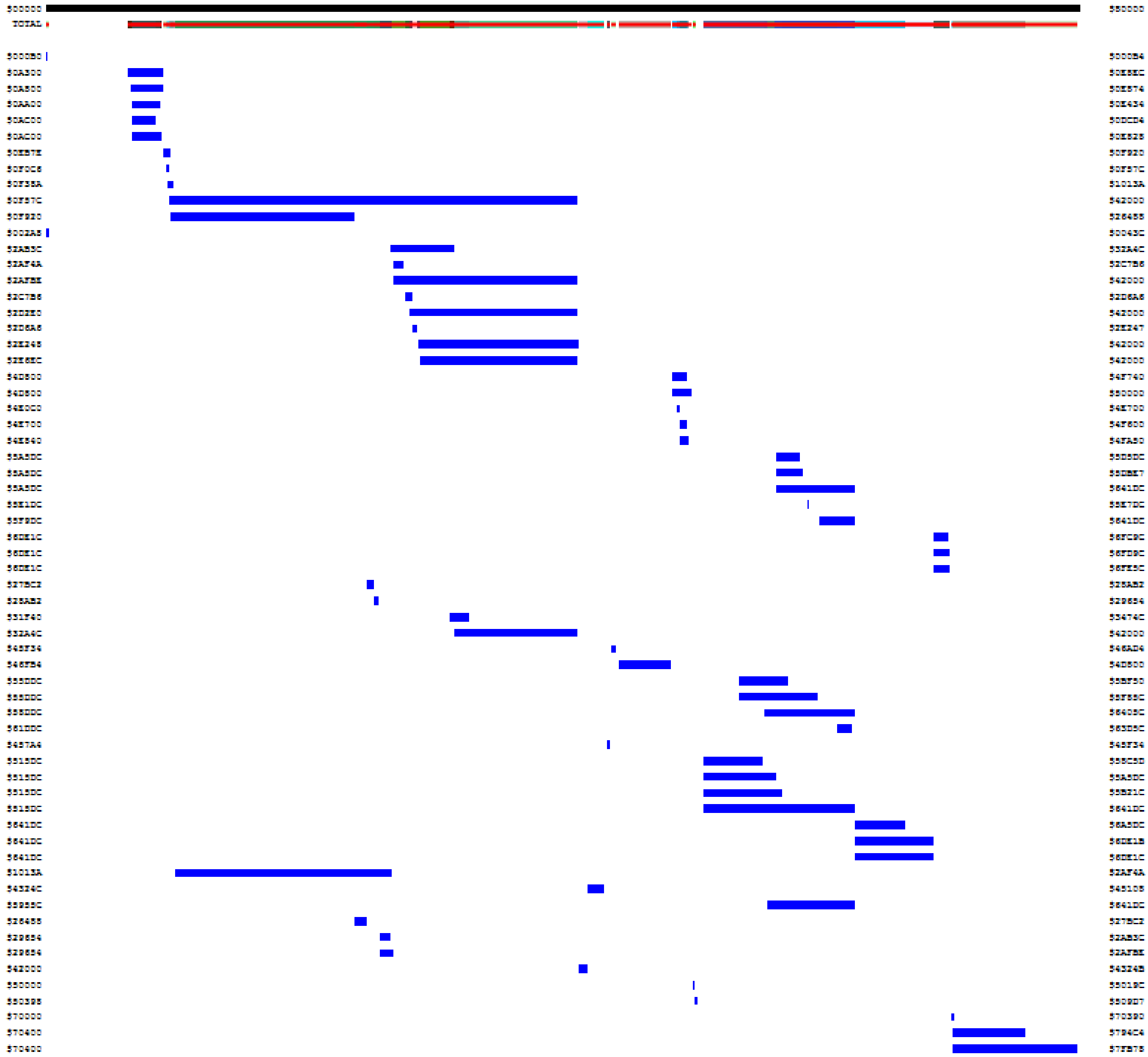
A0 indicates the destination address, where the data read from the disk will be copied.
In consideration of what we have noted, all the data will be copied after the code area located in the lower memory.
We can therefore fill the memory area from **A0** to the end with a **specific pattern** for **each blue screen**.

Press the left mouse button, the trackloader loads the data from the disk and decompresses/decrypts them

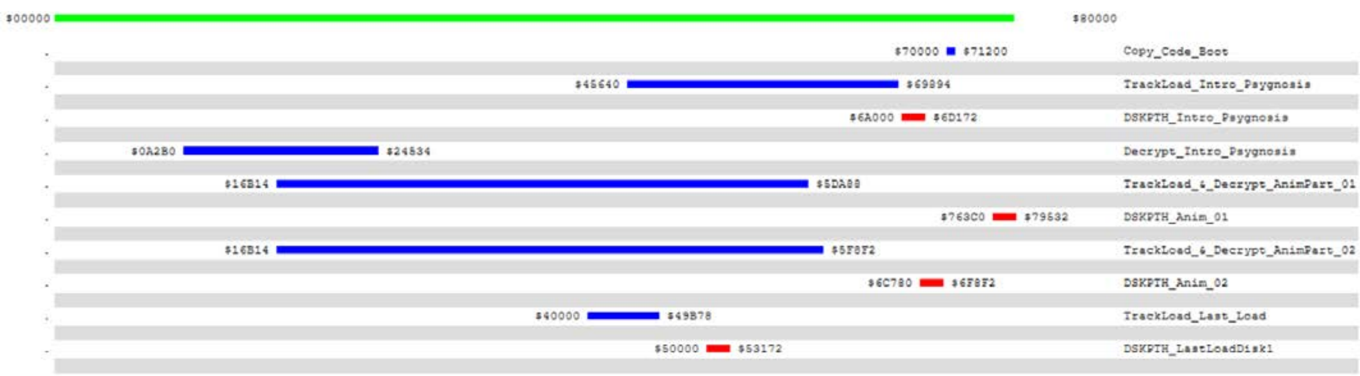
Green background (right button)
Uncompressed data loaded.
We **search** for **our pattern** in memory and **save everything** from **A0** to **our pattern**.
We press the right button of the mouse, and we start again. (and again, and again, ...)

Part 14 Memory usage table

Looking at the previous table of all calls (Part11), it is possible to draw a graph of the memory occupation during the game.
This gives us:



Focus on the 'specific' area of the 1st Disc



It seems clear, the 512Kb of the Amiga are well used...

Part 14B Code analysis during a game

Before looking at the organization that we will have to put in place for the data on our crack.
That is because it seems obvious that you can't copy all the data ripped on an AmigaDos disk 'in place' without making modifications.
We'll look at the code during the game phase and see if there's anything special that could cause us problems.

For that we will 'analyze' 2 Loadings starting from the starting point.
- The one from the left.
- The one on the right towards 'Crystal Cavern'.

it will give us an idea of the 'thing'.
Run the original game until you reach the beginning of LEVEL1, enter the AR and place a *BreakPoint* on the *TrackLoader*.
BS 1B2E & **BS 1B36**

We start to the left, very quickly our *BreakPoint* is reached and we automatically enter the AR
We take a look to see the return address located in the stack.
R
M 24E → **719E**

And we take a look at this code, type in **: D 719E-E**

```
07190    LEA $000515DC, A0                 ; We find our Loading planned with the Trackload phases and Decrypt
07196    LEA $16DE.W, A1                   ; Aggressive Tree 1/4
0719A    BSR.W $00001B2E                   ; GoTo #Trackloader_Start
0719E    BSR.W $00001824                   ; GoTo #Decomp/Decrypt

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

071B4    BSR.W $000054E2                   ; We do not spend time on subroutines, the goal is not to understand
071B8    BSR.W $00005588                   ; all the phases of the game but the important points of 'protection'
                                           ; here, no CMP or other, so we skip their analysis.

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

071CE    LEA $0000AA00, A0                 ;
071D4    LEA $16F6.W, A1                   ; Aggressive Tree 4/4
071D8    BSR.W $00001B2E                   ; GoTo #Trackloader_Start
                                           ; Well... so far, nothing spectacular
071E0    LEA      46A4.S,A0
071E4    LEA      466E.S,A1
071E8    BSR      4654                     ; Bla bla

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

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

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

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

07226    ADD.W (A0)+, D0                   ; → Oh the nice loop of a CRC calculation.
07228    CMPA.L A0, A1                     ;
0722A    BGT.B $00007226                   ; ← Basically, we loop until we get to A1 defined above (namely 9D4A)
                                           ;    Impact on the code? No idea but as a precaution we'll just disable it.

0722C    CMP.W #$3D69, D0                  ; We compare the total, D0 with 3D69
                                             which if it is good, will give us a couple A0, A1 and D0 expected.

07230    BNE.B $00007226                   ; ← If not equal (is that the CRC is not validated so)
                                               Web branch into 7226 (and here we go again for a little ride.)
=========================================
```

To disable this 'check' we will make it simple, instead of testing **D0** we will define it.
So we replace :

~~0722C CMP.W #$3D69, D0~~ ; OPCODE = **B0 7C 3D 69**
by
0722C MOVE.W #$3D69,D0
0730 NOP

It is quite common that one **CRC routine** hides another (well, usually there is more than one and it is quite common that the 'others' are modelled on the same operation as the one previously found.

So we search in memory if there is not another one of the same type.
Type in : F B0 7C 3D 69 → **722C** (the one we just found) and **7D52**
Let's take a look at the routine in **7D52**

```
========================================
07D3E    LEA $000099A8, A0               ; Exactly the same CRC routine seen in 7218
07D44    LEA $00009D4A, A1               ; lol
07D4A    MOVEQ #$00, D0                  ;
07D4C    ADD.W (A0)+,D0                  ;
07D4E    CMPA.L A0, A1                   ;
07D50    BGT.B $00007D4C                 ;
07D52    CMP.W #$3D69, D0                ;
07D56    BNE.B $00007D4C                 ;
========================================
```

So...Same solution.

We replace :
~~07D52 CMP.W #$3D69, D0~~
by
07D52 MOVE.W #$3D69,D0
07D56 NOP

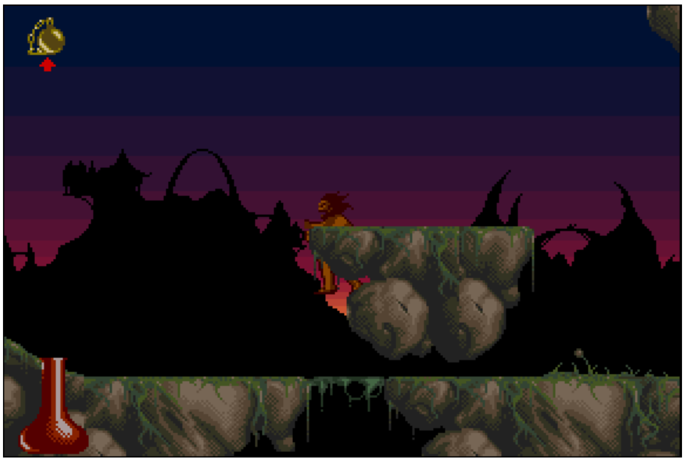
And we continue our analysis and we don't need to go far to see something interesting again (definitely)

```
07232    CMPI.W #$0035, $0C20.W          ; Compare the current 'Track' Pointer with the $35 value (piste !53)
07238    BNE.B $0000724A                 ; No  ? GoTo 724A
0723A    BSR.W $00007A90                 ; Yes ? GoTo 7A90
0723E    JSR $1ACA.W
07242    LEA $0256.W, A7
07246    BRA.W $00002758
```

So here, we will not look at the subroutines in 724A and 7A90, just remember, and this is IMPORTANT, that the code performs tests on the 'current track marker' (aka *$C20*)
this is VERY PROBLEMATIC because as we are going to recreate completely our crack disks on a standard AmigaDOS floppy disk and thus Run on a 'normal' side and not just on a side like the original trackload.
Moreover, it is more than likely that we will also move our data, they will not necessarily be in the same place as the original.

We will see later (at the end of this section), how to bypass this type of 'protection' (bbaaa yes, it is a protection).

So... we go back to the game and head to Crystal Cavern on the far right? I think it's a good idea to look at the code there.
Type in : BDA then **X** and go to the right.

Jump several times to make the ground give way under your feet.


Continue down the stairs that will lead us straight to the **'Crystal Level'** and ask us for the Disk1.

Enter the AR, we look where we are and what is the return address stored in the Stack. **Type in : R**
Without surprise we are in **8CE0**, in the routine : **#DISK2_OR_DISK1_INSERTED?**
With a return address located in **7254**, it is logical to expect the cascade loading of the 11 'files' of this Level, except that... not necessarily.

```
07254   BSR.W $00009D3E
07258   MOVE.W #$01A0, $00DFF096
07260   LEA $000515DC, A0
07266   BSR.W $00009B24                 ; GoSub → #RTZ_TRACKLOADER
                                        ; /!\ Routine to patch because all these sub-routines will no longer exist.

0726A   BNE.B $000072B0                 ; Here is a test on the Flag Z, which, if desired, will connect to 72B0
                                        ; We are in 72B0, It's the trackload of Crystal Cavern #03/11
                                        ; So if this test is valid, we will trackload directly the Part 03/11
                                        ; without going through the 01/11 and 02/11
                                        ;
                                        ; In our crack, it is for sure the guaranteed Bug since we will patch
                                        ; Routines in 72B0 and therefore no modification of Z
                                        ; /!\ It will have to be deactivated/modified
0726C   MOVE.B #$01, $0000F3BE
07274   MOVEQ #$02, D0
07276   MOVEQ #$01, D1
07278   LEA $1626.W, A1
0727C   LEA $00044EFE, A2

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

07286   TST.B $0BE0.W
0728A   BEQ.B $00007298
0728C   TST.B $0000F3C2                 ; in view of the code, Bof..not much more revealing than that one.
07292   BEQ.B $0000728C
07294   CLR.B $0BBE.W                   ; I don't really care about the BBE test
07298   TST.B $0BBF.W                   ; or BBF, they are really too numerous to be a 'protection' as such.
                                        ; It seems rather markers of Loading already done.
0729C   BNE.B $0000728C

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

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

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

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

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

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

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

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

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

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

We return a few minutes on the test seen on **C20** just above. Namely, a test on the counter of 'Track in progress'.

Type in : FA 0C20

```
7166     CMPI.W   #A,C20.S                     ; Unknown for now
7232     CMPI.W   #35,C20.S                    ; The one detected just above.
7FCE     CMPI.W   #4F,C20.S                    ; Unknown for now
9CD6     CLR.W    C20.S                        ; RTZ of the track counter, aka #Base_Conf_Interupt
                                               ; doesn't bother us more than that.

46F10    CMPI.W   #A,C20.S                     ; Unknown for now
```

Take a look at these 3 addresses with the command **D 7166**

```
==================================
07166    CMPI.W   #$000A, $0C20.S              ; Check on the track counter
0716C    BNE      $0000717E                    ; GoSub ➔ TrackLoad_Aggressive Tree 1/4
0716E    BSR      $00007A90                    ; We still find our subroutine in 7A90
07172    JSR      $1ACA.S
07176    LEA      $0256.S, A7
0717A    BRA      $00002758
==================================
```

D 717E

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

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

Let's continue : **D 7FCE**

Considering the code,
we go back to the beginning of the subroutine, it seems interesting.

```
==================================
07FC4    MOVE.W   $02BE.W, D1                  ; D1=$2BE
07FC8    CMP.W    #$454A, D1                   ; a Check on 2BE
07FCC    BNE      $00007FF4                    ; Well… depending on the state of 2BE, it will skip the tests below.

07FCE    CMPI.W   #$004F, $0C20.S              ; Again a check on the ‘track counter’
07FD4    BNE      $00007FC4
07FD6    CMPI.W   #$4150, $02C0.S              ; And another
07FDC    BGT      $0000805C
07FDE    CMPI.B   #$01, $0BC0.S
07FE4    BEQ      $0000805C
07FE6    MOVEQ    #$01, D0                     ; D0=01
07FE8    MOVE.B   D0, $0BC0.S                  ; Copy D0 to address $BC0
07FEC    MOVE.B   D0, $0000FBD4                ; Copy D0 to address $FBD4
07FF2    RTS
==================================
07FF4    CMP.W    #$422C, D1                   ; And again a test on 2BE
07FF8    BNE      $00008018
07FFA    CMPI.W   #$4142, $02C0.S              ; And again another on ‘track counter’
08000    BGT      $0000805C
08002    CMPI.B   #$04, $0BC0.S
08008    BEQ      $0000805C
0800A    MOVEQ    #$04, D0                     ; I don’t really care.
0800C    MOVE.B   D0, $0BC0.S                  ; Because it is not only 'testing', it is also positioned (as here)
08010    MOVE.B   D0, $0000FBDE                ; and the branching made after their tests are not indicative.
08016    RTS
==================================
805C     RTS
==================================
```

And the last, **D 46F10**

Considering the code,
we go back to the beginning of the subroutine, it seems interesting.

```
==================================
46F10    CMPI.W   #A,C20.S                     ; Again, a check on ‘track counter’
46F16    BNE      46F28
46F18    BSR      4783A
46F1C    JSR      1ACA.S
46F20    LEA      256.S,A7
46F24    BRA      42502

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

Well…It is clear that the marker in **C20** is important for the correct functioning of the game.
As seen above, we will have to find a method to make all the tests work well.
I don't really want to blindly patch the tests done on **C20**, and who knows, maybe there will be more!

We also saw a test on **2BE**, we linger 2s on this one, **Type in : M 2BE**
On my side, this value is set to **42 C8**

We go back to the game code, don't touch anything and wait a few seconds. We go back to the AR to look again in **2BE**
The value did not move.

Ok, we go back to the game and we go forward or backward a little, go back to the AR and we look again at the value of **2BE**
The value has changed this time.

You can do some tests and even return to the exact same place physically with your character,
You will find that in fact **the marker in 2BE** seems to indicate a **position of the character** in the **game MAP**.

Theoretically, this should have no impact on our crack (a position remains a position even in a crack).
Nevertheless, it is decent to know that it exists (this marker) and that it is positioned in **2BE**

Returns a small moment on the call towards **9B24** seen just above.

```
7266    BSR.W $00009B24          ; GoSub ➔ #RTZ_TRACKLOADER
726A    BNE.B $000072B0          ; Test on Z Flag, which, if necessary, will connect to 72B0
                                 ; Which, as we have seen, will crash the game.
```

It would be interesting to know when the routine is called to **#RTZ_TRACKLOADER**
If after each call to it, is made a **BNE**, and if so, if this BNE leaves on a crash of the game.
In this case, we would be in the presence of a protection and it will be necessary to look more in detail at the code in **9B24**
We start the investigations, **enter the AE during a game** and **type in** : **FA 9B24**
This gives us: (and we look at the code that gravitates around)

```
254E    BSR 9B24        In the area of Loading Level#1
7266    BSR 9B24        In the area of Loading Crystal_Cavern (view above)
74E8    BSR 9B24        In the area of Loading vortex (thank you to the table of death and A1)
```

ALL have a **BNE** after this instruction, and ALL with this BNE leave on a part of the code not expected if it is the case (Z=0)

245E If Z=0 then it leaves in a reLoading death loop.
7266 If Z=0 then zapping a whole part of the Loading... see Chapter 14B of the tutorial.
74E8 If Z=0 then zapping also part (or all) of the loading of the level in question.

I don't know about you, but it's looking more and more like a protection story. Shall we look at the code in **9B24**? Let's go !

#RTZ_TRACKLOADER

```
9B24  MOVE.L A0,1818.S    ;
9B28  BSR    9B40         ; GoSub ➔ #Pre_Base_TrackLoadX    To be deleted, already integrated in our TrackLoader.
9B2C  BSR    9B8A         ; GoSub ➔ #Motor_ON               To be deleted, same.
9B30  BSR    9BBA         ; GoSub ➔ #Return_T00             To be deleted, same.
9B34  BSR    9C24         ; GoSub ➔ #Pre_Base_TRK_X2        To be deleted, same.
9B38  BSR    9C94         ; GoSub ➔ #Base_Conf_Interupt     We keep the RTZ of $C20, all the rest is garbage.
9B3C  BRA    9CDC         ; GoTo  ➔ #Processing_Trait_01    See below ➔
```

Reminder of what we have seen above in the analysis of the **Trackloader 3 part 9 of the Tuto**

#Processing_Trait_01

```
09CDC   MOVE.L  1818.S,A0          ; A0=1818
09CE0   MOVE.L  #3778,D3           ; Counter into D3
09CE6   MOVE.L  (A0),D0            ; ➔
09CE8   ADDQ.L  #2,A0              ; A0=A0+2
09CEA   MOVE.L  #F,D2              ; Counter into D2
09CF0   MOVE.L  D0,D1              ;            ➔
09CF2   SWAP    D1                 ;            Inverse in longword D1
09CF4   CMPI.W  #4454, D1          ;            We compare with 4454 (Word of synchro Custom?)
09CF8   BEQ     9D08               ;            If identical then we GoTo ➔ #Processing_Trait_02
09CFA   ADD.L   D0,D0              ;            D0=D0+D0
09CFC   DBF     D2,9CF0            ;            ← D2=D2-1, as long D2 is different from -1, we loop.
09D00   DBF     D3,9CE6            ; ← D3=D3-1, as long D3 is different from -1, we loop.
09D04   BRA     9D38               ; GoTo ➔ #SET_D0_To_1
```

#Processing_Trait_02

```
09D08   MOVEQ   #0,D5              ;
09D0A   MOVE.L  (A0),D0            ; ➔ D0=(A0)
09D0C   ADDQ.L  #2,A0              ;   A0=A0+2
09D0E   MOVE.L  #F,D2              ;   Compteur en D2
09D14   MOVE.L  D0,D1              ;            ➔
09D16   SWAP    D1                 ;            Inverse in longword D1
09D18   CMPI.W  #4454,D1           ;            We compare with 4454 (Word of synchro Custom?)
09D1C   BEQ     9D2C               ;            If identical then we GoTo ➔ #Processing_Trait_03
09D1E   ADD.L   D0,D0              ;            D0=D0+D0
09D20   DBF     D2,9D14            ;            ← D2=D2-1, as long D2 is different from -1, we loop.
09D24   ADDQ.L  #1,D5              ;    D5=D5+1
09D26   DBF     D3,9D0A            ; ← D3=D3-1, as long D3 is different from -1, we loop.
09D2A   BRA     9D38               ; GoTo ➔ #SET_D0_To_1
```

#Processing_Trait_03

```
09D2C    SUBI.L   #1A2C,D5                          ; D5=D5-1A2C
09D32    BMI      9D38                              ; If the result is negative, then GoTo → #SET_D0_To_1
09D34    CLR.W    D0                                ; Otherwise the Word D0 is cleared
09D36    RTS                                        ; E.T Back Home
```

#SET_D0_To_1

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

Well, it's nice to have all these subroutines, but what does it do? It doesn't write anything to a memory address and it seems to calculate something and according to the result **D0**=0 or **D0**=1

It would be interesting to know the 'normal' course of these subroutines and the output state of **D0** and **Flag Z**.
For that it's easy, you just have to play at the place of the test(s)

254E Loading Level#1, it will be enough to put a BS in 9D34 and 9D38 from the game menu and start Loading.
7266 Same with the **BS** except that the action is to be done before entering the Crystal Level
74E8 Well... it's going to be harder because you have to advance a lot in the game to get to this stage, so you'll take my word for it.

On each *Breakpoint* put here above, we are **ALWAYS** in the 'area' **9D34**, we never reach **9D38**
So the 'normal' output of the routine **#Processing_Trait_01** in **9CDC** should always be **D0=0** and **flag Z=1**

So, for sure, it's a protection.

We will not continue our investigation any further and go directly to the conclusion of this chapter.
We have seen:

- 1st of CRC routine 7218 → 07230 to be deactivated in 722C
- 2nd of CRC routine 7D3E → 07D56 to be deactivated in 7D52
- Game Operation with Checks made on the Track Marker in $C20
- A **BNE** to be deleted in 726A ~~BNE.B $000072B0~~
- A set of subroutines to be disabled when replacing the original trackloader.
- Position marker in MAP in $2BE
- A set of subtoutine to be disabled into $9B24
- But keep in it a **RTZ** of $C20, and a **RTZ** of **D0** (**/!** make sure to have a **Z=0** a the output)

All these modifications will be done during the HACK of the **Last_Load_Disk1** data

For the global operation of the game with the track marker in C20,
it would be **easier** to keep the **original positions** found in our **death table** listing all calls.
*Reminder : 1st long word of (A1), A1 which contains the **address** of the **table** for the **next trackload**.*

Then...to be determined according to the test of our crack, there may be other things to modify... ?!

Part 15 Reorganization of the data for the creation of our disk

The idea : Try to create a cracked version containing the whole game and its animations on 2 disks.

The problem is, if you look at the table of calls (Part 11 of the tutorial), it is clear that we are here on a significant overflow of data.
If you study this table you can see that the data are contiguous and use almost the whole disk. (by the way, incredible...)

If we move the loadings **Load_Menu_01 -> Load_Menu_07** of disk2 to the disk1, the data will fit on a standard *AmigaDOS*
It will of course be necessary to modify the calls for its loadings (but they are called only once, at the beginning of the Loading Menu) and also modify the call of the last : **Load_Menu_08**
Use a 'standard' *trackloader* by modifying its input to use exactly the same types of calls and use the original tables contained in memory or modify some calls to fit our disk
It will be much less work to do and it will be much easier (in theory)

All these calls are made with the last loading after the animations, namely : **Last_Load_Disk1** which clearly contains **code** and **not Data**, it will have to be modified.
This gives us this:

DISK2 – Our cracked version

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

Load_Level1_02	24F0	00E84	BSR 1824	00000E8598683F015B0D07E034E1A0FC063C341E403C2400A077E140F80422BC	D2	LOW	25D10-26B93	25D10-26B93	210F8-21F7B
In_Game_63	763E	00E84	BSR 1824						
In_Game_45	7C9A	00E84	BSR 1824						
In_Game_56-RET	7812	00E84	BSR 1824						
In_Game_12	8A9C	00E84	BSR 1824						
Load_Level1_03	2508	0DDC8	BSR 1824	0000DD7976FF8B66043E0C083C00F34501FCC1A641FAFF0712FAA50AE184EA4E	D2	LOW	26B94-3495B	26B94-3495B	21F7C-2FD43
In_Game_46	7CAC	0DDC8	BSR 1824						
In_Game_57-RET	7824	0DDC8	BSR 1824						
In_Game_31-B	7980	0DDC8	BSR 1824						
Load_Level1_04	251A	03C28	No	428E30004020444A3000402000050000046602130311042205320643075400001	D2	LOW	3495C-38583	3495C-38583	2FD44-3396B
In_Game_58-RET	7836	03C28	No						
In_Game_31-C	7992	03C28	No						
In_Game_48	7CD0	03C28	No						
Load_Level1_06	2542	003A0	BSR 1824	000003A1002C3FF00801031FFC8080EF8042E7F38080503F9CF8080C7FF97FE4	D2	LOW	38584-38923	38584-38923	3396C-33D0B
In_Game_47	7CBE	003A0	BSR 1824						
In_Game_30	8244	003A0	BSR 1824						
In_Game_22	84C2	003A0	BSR 1824						
In_Game_14	8AC0	003A0	BSR 1824						
In_Game_65	7668	003A0	BSR 1824						
Load_Level1_05	2530	01178	BSR 1824	00001178C20F103FFF08100C7038C5770A0417D9AC1F6F7E2FCFFF7CBA435DF0	D2	LOW	38924-39A9B	38924-39A9B	33D0C-34E83
In_Game_49	7CDE	01178	BSR 1824						
In_Game_31-D	79A0	01178	BSR 1824						
In_Game_18	7A3C	01178	BSR 1824						
In_Game_10	8A70	04F08	BSR 1824	00004F0900B2A10715B3F1AC0E03FE53F1FC09DE3F0E71DA0EB103F10FA1824F	D2	LOW	39A9C-3E9A3	39A9C-3E9A3	34E84-39D8B
In_Game_01	719A	04F08	BSR 1824						

In_Game_02	71AC	054A0	BSR 1824	000054A1D4FFF00021891B032AFD0E73C00180E0390B65BFDCAFDFFE17900203	D2	LOW	3E9A4-43E43	3E9A4-43E43	39D8C-3F22B
In_Game_11	8A82	054A0	BSR 1824						
In_Game_29	8232	054A0	BSR 1824						
In_Game_21	84B0	054A0	BSR 1824						
In_Game_23	7AD0	0B26C	BSR 1824	0000B26C0708A07FD8507086060A07086261C0708A068003C180BC84A0FD01C2	D2	LOW	43E44-4F0AF	43E44-4F0AF	3F22C-4A497
In_Game_13	8AAE	0B26C	BSR 1824						
In_Game_03	71C6	0B26C	BSR 1824						
In_Game_04	71D8	03A34	BSR 8C0A No	39F43000400000003000400000030000046603120423062308430964 0A870000	D2	LOW	4F0B0-52AE3	4F0B0-52AE3	4A498-4DECB
In_Game_24	7AE2	03A34	No						
In_Game_15	7A0A	07DC8	BSR 1824	00007DC9F610340060680040050640120C043FD8087FB010E16041B0C883C1B8	D2	LOW	52AE4-5A8AB	52AE4-5A8AB	4DECC-55C93
In_Game_16	7A1C	041C8	BSR 1824	000041C97C2938E07BCEF033E181DF8060404F93FE73E60019F3800CF9C0067C	D2	LOW	5A8AC-5EA73	5A8AC-5EA73	55C94-59E5B
In_Game_17	7A2E	045EC	No	348C400040B030E63000406000010000046603140212032204320542 06550000	D2	LOW	5EA74-6305F	5EA74-6305F	59E5C-5E447
In_Game_19	8360	0353C	BSR 1824	0000353D1A34BFE021A3037FC0FF57C494A1A103281BD90D281BBE0C24243420	D2	LOW	63060-6659B	63060-6659B	5E448-61983
In_Game_28	8150	00588	BSR 1824	00000589301A3A40165103600046DF1815807901CFC40D80011BACFEC908B220	D2	LOW	6659C-66B23	6659C-66B23	61984-61F0B
In_Game_20	8372	00588	BSR 1824		D2	LOW	6659C-66B23		
In_Game_25	811A	01CE4	BSR 1824	00001CE58A713BA1D1B45181A3488040F452499015D200E0868B5881811683E6	D2	LOW	66B24-68807	66B24-68807	61F0C-63BEF
In_Game_26	812C	001E0	BSR 1824	000001E1BF00A3648003095F041DF95C07E206A003D2C81FF54A18657FF21CB5	D2	LOW	68808-689E7	68808-689E7	63BF0-63DCF
In_Game_27	813E	00B5C	BSR 1824	00000B5D0F77FC7DE6F729300E96E4FFF26F7503007DEFDEC3C5A1618821C01E	D2	LOW	689E8-69543	689E8-69543	63DD0-6492B
In_Game_05 In_Game_09-RET In_Game_59 In_Game_60	7E68	02EEC	BSR 1826	00002EEDE25501646D52E50701E29085D20959856290995044 20E14940000AA9	D2	LOW	69544-6C42F	69544-6C42F	6492C-67817
In_Game_06	8918	02C44	BSR 1824	00002C4538E3F3FFE0E383806071DAB43E7037F0206FD01C2FE880C81F3E0C2F	D2	LOW	6C430-6F073	6C430-6F073	67818-6A45B
In_Game_07	892A	00B70	BSR 1824	00000B71644137B830701787CF8FF87DDB4FC313F33FDC1FFCF03FEFF7905F8E	D2	LOW	6F074-6FBE3	6F074-6FBE3	6A45C-6AFCB
In_Game_08	893C	0011C	BSR 1824	0000011DC300643E080808010380A07DFFC680303342E80303B42F40340181F4	D2	LOW	6FBE4-6FCFF	6FBE4-6FCFF	6AFCC-6B0E7
In_Game_54b	7E68	04228	BSR 1824	000042898020217801A4D81E55060FF8B0762BC0E078AFC1D11789822C62807C	D2	LOW	6FD00-73F87	6FD00-73F87	6B0E8-6F36F
In_Game_60	74FA	04228	BSR 1824	0000422970303260606 8FD40E401681C0C20180040E405684B0640281980B008	D2	LOW	73F88-781AF	73F88-781AF	6F370-73597

In_Game_61	750C	001D0	BSR 1824	000001D1C2163A3680184590C0CA01D1F604831789076F140C12B786064F040D	D2	LOW	781B0-7837F	781B0-7837F	73598-73767
In_Game_62	751E	00628	BSR 1824	00000629C02705880D3AA07C089084AF87FBFA7FE13406807FEBE05920806002	D2	LOW	78380-789A7	78380-789A7	73768-73D8F
In_Game_67	768C	0B00C	BSR 1824	0000B00D9227AD0DFE9BFD3BFA67F617ED17DC67B3C5B16B6757786ED71EDB9F	D2	UP	85D04-90D0F	00000-0B00B	73D90-7ED9B
In_Game_57	709E	0B00C	BSR 1824						
In_Game_58	70B0	030D4	No	5040300040F85124300040F80001000004660212032204320643076508880001	D2	UP	90D10-93DE3	0B00C-0E0DF	7ED9C-81E6F
In_Game_56	708C	00478	BSR 1824	00000479B08A01039602079C040E58081B20103960206C8040D4C081B2010353	D2	UP	93DE4-9425B	0E0E0-0E557	81E70-822E7
In_Game_66	767A	00478	BSR 1824						
In_Game_55	7074	05914	BSR 1826	0000591594AD694FAD804930120486007B3977B3FBFF1A100D24621561DECCFF	D2	UP	9425C-99B6F	0E558-13E6B	822E8-87BFB
In_Game_64	7650	05914	BSR 1826						
N/A	78E4	05914	BSR 1826				9425C-99B6F		
END_01	4F7E	00308	BSR 1824	00000309C54888005AA3BF41E01008040DF2801B043CC054020D489203280920	D2	UP	99B70-99E77	13E6C-14173	87BFC-87F03
END_02	4F90	32A84	No	67616D656F7665720000000000000000000623100000000000000000000000000	D2	UP	99E78-CC8FB	14174-46BF7	87F04-BA987
END_03	5010	0F778	No	060602FE040A0E120E060404FCFC060A0E0E08060804FCFE02060E0A08080802 (-B0)	D2	UP	CC8FC-DC073	46BF8-5636F	BA988-CA0FF
END_04	501E	06E04	BSR 1958	006E01073FCF8FDE3CE551901E005A0F8F021FBFF9CAA3201B005A01FC01FE02	D2	UP	DC074-E2E77	56370-5D173	CA100-D0F03
In_Game_68	78B0	04C98	BSR 1824	00004C981087B1FBEF784E70A83D19E00C30C050614020109787F2EF78780D15	D2	UP	E2E78-E7B0F	5D174-61E0B	D0F04-D5B9B
Load_Menu_09	1D5E	00060	BSR 1824	000000603790606040FF807848B5D9E050211706142C24000609274211C42570	D2	UP	E7B10-E7B6F	61E0C-61E6B	D5B9C-D5BFB
In_Game_69 [END]	85EE	04520	BSR 1824	00004521C3140C56F148B57CE0B2349645E52DD0E0698660185BB9C2C02CF1C8	D2	UP	E7B70-EC08F	61E6C-6638B	D5BFC-DA11B
	4F03			AREA not USED ~20Ko			EC090-F0F93		
Load_Menu_08	1E70	01EBC	No	4AB8041E67162078041E20B804262078042220B8042A700021C0041E41F80B1A	D2	UP	F0F94-F2E4F	6B290-68247	DA11C-DBFD7

As shown in the table above, we will simply copy the ripped data from **Disk2** to our **Disk2** but we will not start from the position **raw** $4C18 but $0000
We will just bypass the **UNUSED FIELD** at the end of the disk **(EC090-F0F93 aka : 'AREA not USED')** in order to enter all our data on our disk.

As you can see, the difference is **-4C18**.
We will have to do this subtraction before our *TrackLoad* routine or of course, directly in the code of our *TrackLoader 'easier solution'*.

Note that from the original calls of the second side (UP), we get a larger delta and this is normal.
Delta which corresponds to the end marker of the original disk: *$85D04* (review **Part 12a Rip Full Disk1** if necessary) minus the address of the beginning of the next data: *$73D8F* (see table above)
This gives us: **85D04-73D8F=$11F74**

We'll see later how we can manage these shifts simply with **CMP**.

DISK1 – Our cracked version

					ORIGINAL		Position in		
Call Order	Calling Adr.	LENGTH	Decomp/ Decrypt	Hexa Chain // Signature	Disk	SIDE	Real Disk	Ripped File	Our Cracked Disk
Boosector + code									
Load_Menu_10	1E8E	04C18	BSR 1824	00004C19441E01FB52EEBFB9E1A93F80945B24B05FC0EB206DBC0074CF981007	D2	LOW	00000-04C18	00000-04C18	00400-05017
Load_Menu_01	1C66	03A90	BSR 1824	00003A91C46244304A924783188E0649088F1EB8400482075E0B6AA411301800	D2	UP	EC090-EFB1F	6638C-69E1B	05018-08AA7
Load_Menu_02	1C78	00250	BSR 1824	00000250C1C0303200A06617C0080081880703380 1003A3ADBF0040020CF803F	D2	UP	EFB20-EFD6F	69E1C-6A06B	08AA8-08CF7
Load_Menu_03	1C8A	00438	BSR 1824	0000043980D7046478DA0F679AE33C6E0448C02100B33BDCC79A03DDDB9EBE7D	D2	UP	EFD70-F01A7	6A06C-6A4A3	08CF8-0912F
Load_Menu_04	1C9C	00124	BSR 1824	000001242180904B8A0A2188008C4042100048C0022183430283888A8181CD83	D2	UP	F01A8-F02CB	6A4A4-6A5C7	09130-09253
Load_Menu_05	1CAE	00664	BSR 1826	00000665E080500020003808C0003E024005BD05C203FE030107FE8A81077F99	D2	UP	F02CC-F092F	6A5C8-6AC2B	09254-098B7
Load_Menu_06	1CC6	00268	BSR 1824	00000269AC054004C00BDEA00BC9A8A01B90782D2C023041CA05B443CC737227	D2	UP	F0930-F0B97	6AC2C-6AE93	098B8-09B1F
Load_Menu_07	1CD8	003FC	BSR 1824	000003FDE5FF07F9078B40A078967827ED28B08F40700463F81FB0904C6C0482	D2	UP	F0B98-F0F93	6AE94-6B28F	09B20-09F1B
In_Game_37	72F0	04074	No	483E40704170454C40E04150000B00000466001300250136034705680 89B0000	D1	LOW	76FC0-7B033	75724-79797	09F1C-0DF8F
In_Game_43	7C44	0280C	No	0000280D1FA620D8E704C7069F08E04AF1D007055F823F9D84811F065C960EC0	D1	UP	CFF58-D2763	4A254-4CA5F	0DF90-1079B
In_Game_32	7E68	0280C	BSR 1826						
In_Game_33	72A8	0C3C8	BSR 1824	0000C3C90036A00C583E4000C8C80064780033E8B7761A16401908000CAA8DFD	D1	UP	D2764-DEB2B	4CA60-58E27	1079C-1CB63
In_Game_34	72BA	00524	BSR 1824	000005250068 3F6BFC1A029020908888F88A007F840E515082EB1D336034833A	D1	UP	DEB2C-DF04F	58E28-5934B	1CB64-1D087
In_Game_35	72CC	0151C	BSR 1824	0000151D45FDFF008B81FC01010120D7E7E01149DC1DFA8278761FCE6A24F9D0	D1	UP	DF050-E056B	5934C-5A867	1D088-1E5A3
In_Game_36	72DE	0A73C	BSR 1824	0000A73D2D07F1E3E20380E61205A207AA07825B781F3910F01F801FFC3FF96F	D1	UP	E056C-EACA7	5A868-64FA3	1E5A4-28CDF
In_Game_38	7308	007F0	BSR 1824	000007F1D55B6181EEC1031002A0807E83C0021368881FE49F0036087845C804	D1	UP	EACA8-EB497	64FA4-65793	28CE0-294CF
In_Game_39	731A	10410	BSR 1824	00010411B6A6A680A40303CC02409823C1203818701B2C62C8710E8164110041	D1	UP	EB498-FB8A7	65794-75BA3	294D0-398DF
In_Game_40	732C	01828	BSR 1958	0018262AC0E5EADFF6F6EE00FF13EF2823215040121F28281F1F152222003BF5	D1	UP	FB8A8-FD0CF	75BA4-773CB	398E0-3B107
In_Game_41	733E	00E10	BSR 1958	000E0D18191A150C02FEFDFE9A010200FDFAF9F7F8FD00030403020200010408	D1	UP	FD0D0-FDEDF	773CC-781DB	3B108-3BF17
In_Game_42	7350	00B48	BSR 1958	000B48F6F7F7FC00030A110D17161A2120212016120E0702FEF6F2EEE8E8E6E4	D1	UP	FDEE0-FEA27	781DC-78D23	3BF18-3CA5F

Then will come afterwards, the animations.
Preferably **Anim_part#01** and **Anim_part#02** first because we know that there are no changes to be made in these so we know exactly their compacted sizes.
While in **Anim_Psygnosis** and **Last_Load_Disk1**, we will have to do some. It is better to leave them last, it will be easier if you have to make several tries.
We will just have to rewrite these last 'two files'.

The best compression ratio is obtained with the **RNC PROPACK** compressor whose sources are **available on aminet**, thus impeccable
FYI, **Crunchmania** also does quite well BUT, not as well as **RNC** 😉

After a few tests, here is what the compression gives us in terms of rate/gain. (I bypass the compression part of these animations with **PPAMI** under workbench, the syntax is quite simple).

Info	Real Size	Compr. Size RNC ProPacker	Gain Calcul Bytes	Extra Nfo
Anim Psygnosis	164 022	133 499	30 523	Loaded by the bootsector, not in the table of game. (Trackdisk.Device)
Anim sotb2 #1	290 677	236 710	53 967	Loaded by the code in Anim_Psygnosis (Trackloader Custom #1)
Anim sotb2 #2	298 460	235 956	62 504	Loaded by the code in Anim_Psygnosis (Trackloader Custom #1)
Last Code	39 800	~~21 532 *~~	~~18 268 *~~	Loaded by the code in Anim_Psygnosis (Trackloader Custom #1)

*** /!** in order to keep a better flexibility for eventual modifications, we will **NOT compress** the data 'LAST CODE DISK1' **BUT** it remains 'possible'.

We are going to place our files as close as possible to each sector (multiple of 512 bytes) in order to save as much space as possible and to keep the operation with our Custom Trackloader #01

This finally gives us :

Info	Calling Order	LENGTH	Hexa Chain // Signature Decomp	Our cracked Disk	Disk Crack	Position Track + Offset
	~~In_Game_42~~	~~00B48~~	~~000B48F6F7F7FC00030A110D17161A2120212016120E0702FEF6F2EEE8E8E6E4~~	~~3BF18-3CA5F~~	~~D1~~	~~N/A~~
			927 Empty bytes	3CA60-3CDFF		
RNC	Anim part #01	**39CA6**	00046F740030FFFFFFFF0010003400030038004600 1C00140000000000464000A	3CE00-76AA5	D1	Track !44 + $600
			159 Empty bytes	76AA6-76BFF		
RNC	Anim part #02	**399B4**	00048DDE0030FFFFFFFF00100034000300380046001C00140000000004C60000	76C00-B05B3	D1	Track !86 + $800
			75 Empty bytes	B05B4-B05FF		
RNC	Anim Psygnosis	**~2097B**	0002410535D800359600F813200C62F729F35C0249D2D086CA06102083800C64	B0600-D0F7A	D1	Track !128 + $600
			644 Empty bytes	D0F7B-D11FF		
CODE	Last_Load_Disk1	**9B78**	60001B887200123900BFE801610651C9FFFC4E75203900000BE2D08008000017	D1200-DAD77	D1	Track !152 + $1400
			4744 Empty bytes to the end of DISK	DAD78-DC000		

This deserves some explanation.

Reminder: Size of standard AmigaDos track = $1600 or !5632 bytes Size of AmigaDos standard Sector= $200 ou !512 bytes Nbr of sectors in AmigaDos Standard Track = !11

We know that the end of our ripped data from SOTB2 on our **Disk1** ends up in Raw.pos **$3CA60**
In order to make things easier with our Custom #01 trackloader, we will work in *'sector'* mode, so the question to ask is.
What is the address of the next sector after the position Raw.pos **$3CA60** ?

The answer would be ... Still it would be necessary to know already on which Track and sector we are : **$3CA60 ... o_O'**

Easy : **$3CA60 / $1600 =** !44,11 So first, we have the Track, it's **!44**
Then just calculate the delta. : **!44*$1600= $3C800**

$3CA60 - $3C800 = $260
$260 / $200 = !1,19 (1 sect=512 Bytes), So 1 full secteur and plus again data on another sector, which gives us a total of **2 sectors.**

So logically, the '**next sector**' will be the 3rd in the Track!
In reality it will be the sector 2 of the Track !44 because we count the sectors from Zero. So the '3rd sector' is the sector N°2

This gives us in hexa : (!44 * $1600) + ($200 * 3) = $3C800 + $600 = **$3CE00**

So our first Data to be recorded, namely **Anim part #01** should be written in Raw.pos **$3CE00**
You simply use the length of the data to calculate the next position **and so on**.

With one small detail, we must modify our previously cracked file* : **Anim Psygnosis,** so we don't know 'exactly' its final size (although it won't vary by a lot)
We will therefore leave 1 more sector free in our table concerning the case. (Blue Bar)

Also, the next data to be written after this one, namely: **Last_Load_Disk1** will not be written 'at the next free sector' but 'at the next free sector **+ 1'**.
* and yes because this 'table' is contained in the code : **Anim Psygnosis** , we will have to update and re-compact it.

I hope I have been clear enough on the subject 😉
No ? Too bad 😉

So our future loading table in the **'Anim Psygnosis'** code will be :

8 Bytes (578 ->58F)

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

Part 15b Preparing the files for the creation of our cracked Disk1

Insert in **DF0** the floppy disk containing the rip file: **Disk2_Upper.bin**
And in DF1 a new blank disk formatted. **A small reboot and we enter the AR**

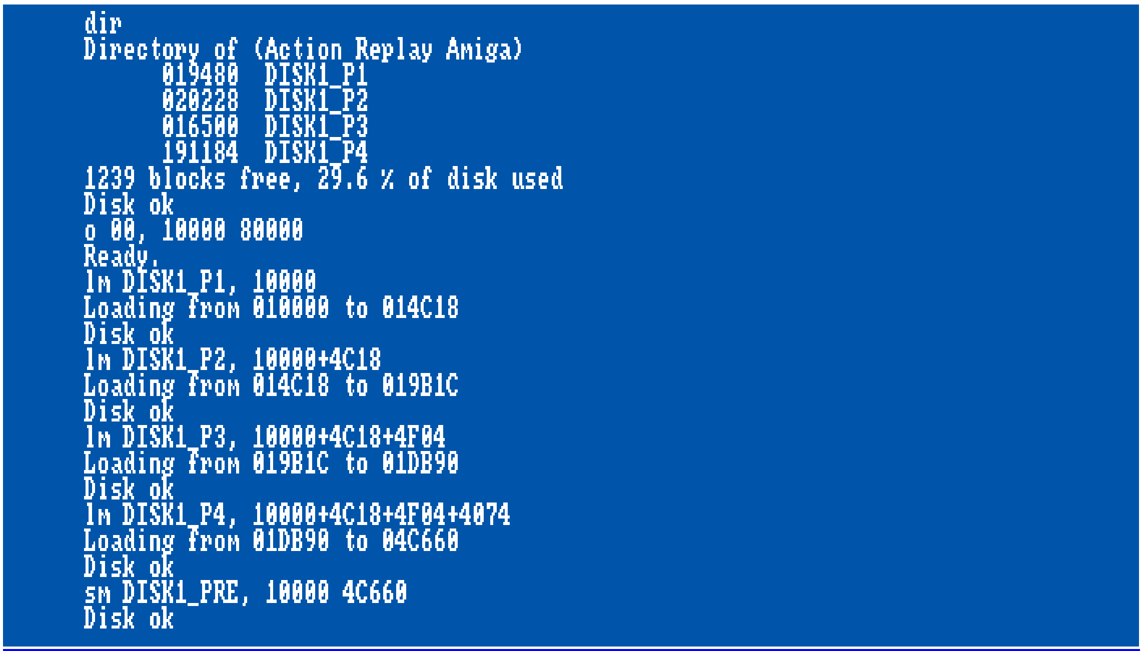
Type in :
```
O 00, 1000 80000
LM Disk2_Lower.bin, 1000
SM 1:DISK1_P1, 1000 1000+4C18
```

Swap the floppy disk in **DF0** by the one containing the file :**Disk2_Upper.bin**
Type in :
```
LM Disk2_Upper.bin, 1000
SM 1:DISK1_P2, 1000+6638C 1000+6638C+4F04            // $6B28F +1 - $6638C = $4F04
```

Swap the floppy disk in **DF0** by the one containing the file :: **Disk1_Lower.bin**
Type in :
```
LM Disk1_Lower.bin, 1000
SM 1:DISK1_P3, 1000+75724 1000+75724+4074
```

And to finish this section, swap the floppy disk in **DF0** by the one containing the file: **Disk1_Upper.bin**
Type in :
```
SM 1:DISK1_P4, 1000+4A254 1000+4A254+2EAD0           // $78D23 + 1 - $4A254 = $2EAD0
```

we join the entire thing, **Type in** :
```
O 00, 10000 80000
LM 1:DISK1_P1, 10000
LM 1:DISK1_P2, 10000+4C18
LM 1:DISK1_P3, 10000+4C18+4F04
LM 1:DISK1_P4, 10000+4C18+4F04+4074
SM 1:DISK1_PRE, 10000 4C660
```



Part 16 Modification and compilation of the AlphaONE Trackloader - Phase1

The AlphanOne trackloader version 2004 will do very well, we'll just make some modifications on it.
We remove **A5** from the AlphaOne Trackloader source code because at this stage it interferes with the original SOTB2 code

Reminder of their respective functioning:

Functioning of the original SOTB2 trackloade	Value of AlphaOne trackloader
A0=Memory Destination **A1**=Table Position	**A0**=Memory Destination **A1**=Not use **A2**=DSKPTH **D0**=Length to read **D1**=Number of starting track number **D2**=Offset in the, ZERO

Trackloader_SOTB2_#01.s

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

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

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

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

TRACKLOADER:
                MOVEM.L D0-D7/A0-A6,-(A7)           ; Save all register
                LEA     $DFF000,A6                  ; Required
 ;------------------------------------------------------------------------------------------------------------
                MOVE.L  (A1),D2                     ; Retrieval Raw Pos
                MOVE.L  4(A1),D0                    ; Retrieval Length
                MOVE.L  D2,D1                       ; Copy Raw Pos. to D1
                DIVU.W  #$1600,D1                   ; D1 / Actual size concerned
                AND.L   #$FF,D1                     ; Clean D1 // TRACKNR.L
                MOVE.L  D1,D3                       ; Copy D1 to D3
                MULU.W  #$1600,D3                   ; Delta
                SUB.L   D3,D2                       ; Calculed Offset // BYTEOFFSET.L
;------------------------------------------------------------------------------------------------------------
                LEA     $BFD100,A4                  ; DRIVESELECT REGISTER
                ;LEA    $BFE001,A5
                MOVEQ   #0,D7                       ; D7 = BYTECOUNTER
                ADD.L   D2,D0                       ; BYTES TO READ + BYTEOFFSET

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

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

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

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

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

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

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

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

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

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

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

TRACKLOADERENDE:
```

We type/load the whole under ASM-ONE, we assemble
and we save the compiled binary (from TRACKLOADER to TRACKLOADERENDE) as : **TRACKLOADER_AlphaOne_P1**

For more info on how to use ASM-one check out my other tutorials available on my Home Page.
For example, you can take a look of : R-Type II

Or if you prefer the already compiled version, attached as a Hexa suite.

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

Part 17 Compilation of the RNC PRO-PACK compression routine by Rob North.

As explained before, it is simply impossible to propose a crack of the game running on two floppies without going through a phase of compression of some files. For that we used **RCN pro-pack method 1** to compile our files.
It is now time to compile the decompression routine that we will use for our crack.
We are going to adapt it to our needs.

RNC_1.S

```
*-----------------------------------------------------------------------------
* PRO-PACK Unpack Source Code - MC68000, Method 1
*
* Copyright (c) 1991,92 Rob Northen Computing, U.K. All Rights Reserved.
*
* File: RNC_1.S
*
* MOD for 'absolute addressing' of buffer size and position
* Date: 24.3.92
*-----------------------------------------------------------------------------

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

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

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

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

NOT_PACKED       EQU     0
PACKED_CRC       EQU     -1
UNPACKED_CRC     EQU     -2

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

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

;RAW_TABLE       EQU     0                ; disabled. We will operate in absolute

;POS_TABLE       EQU     RAW_TABLE+16*8   ; disabled. We will operate in absolute

POS_TABLE        EQU     $6A220           ; replaced by: addr. of the buffer zone in absolute
                                          ; In this case in $6A220, see explanation at the end of this chapter.

LEN_TABLE        EQU     POS_TABLE+16*8   ; Do not change.

;                IFEQ    CHECKSUMS        ; Not used
BUFSIZE          EQU     16*8*3           ; 384 Bytes should indeed be enough.

;                ELSE                     ; Not used
;BUFSIZE         EQU     512              ; Not used
;                ENDC                     ; Not used

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

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

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

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

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

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

                 IFNE PROTECTED
                 move.w   d0,key
                 ENDC

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

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

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

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

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

unpack7
                moveq.l  #0,bit_count
                move.b   1(input),bit_buffer
                rol.w    #8,bit_buffer
                move.b   (input),bit_buffer
                moveq.l  #2,d0
                moveq.l  #2,d1
                bsr      input_bits
unpack8
                move.l   a2,a0
                bsr      make_huftable
;               lea      POS_TABLE(a2),a0           ; We no longer work with the couple POS_TABLE(A2)
                lea      POS_TABLE,a0               ; but directly with POS_TABLE
                bsr      make_huftable
;               lea      LEN_TABLE(a2),a0           ; Same
                lea      LEN_TABLE,a0               ;
                bsr      make_huftable
unpack9
                moveq.l  #-1,d0
                moveq.l  #16,d1
                bsr      input_bits
                move.w   d0,counts
                subq.w   #1,counts
                bra.s    unpack12
unpack10
;               lea      POS_TABLE(a2),a0           ; Same
                lea      POS_TABLE,a0               ;
                moveq.l  #0,d0
                bsr.s    input_value
                neg.l    d0
                lea      -1(output,d0.l),a1
                move.l   a1,$dff180                 ; ADD COLOR BAR
;               lea      LEN_TABLE(a2),a0           ; We no longer work with the couple LEN_TABLE(A2)
                lea      LEN_TABLE,a0               ; but directly with LEN_TABLE
                bsr.s    input_value
                move.b   (a1)+,(output)+
unpack11
                move.b   (a1)+,(output)+
                dbra     d0,unpack11
unpack12
                move.l   a2,a0
                bsr.s    input_value
                subq.w   #1,d0
                bmi.s    unpack13
                getrawREP
                move.b   1(input),d0
                rol.w    #8,d0
                move.b   (input),d0
                lsl.l    bit_count,d0
                moveq.l #1,d1
                lsl.w    bit_count,d1
                subq.w   #1,d1
                and.l    d1,bit_buffer
                or.l     d0,bit_buffer
unpack13
                dbra     counts,unpack10
                cmp.l    output_hi,output
                bcs.s    unpack8

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

```
unpack15

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

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

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

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

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

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

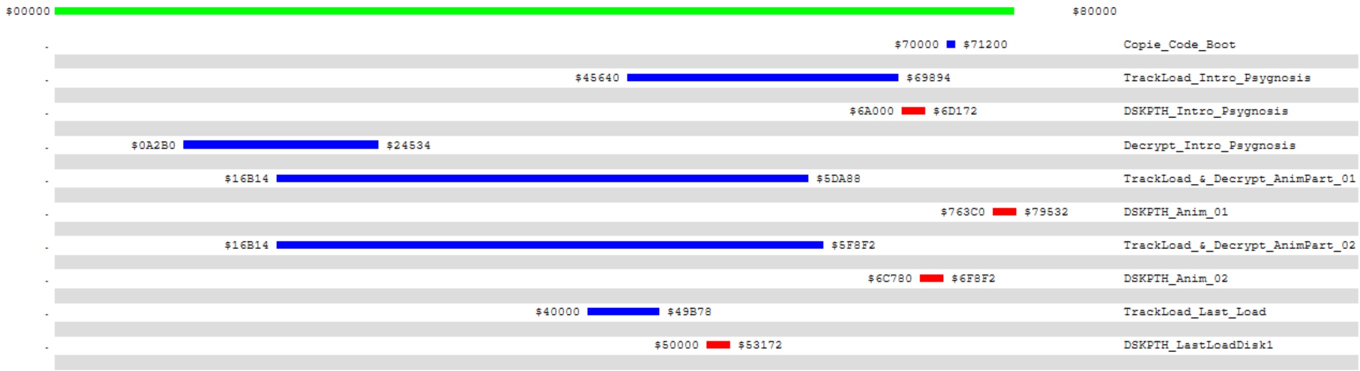
```
make_huftable3
                  moveq.l  #$f,d0
                  moveq.l  #4,d1
                  bsr.s    input_bits
                  move.b   d0,(a1)+
                  dbra     d2,make_huftable3
                  moveq.l  #1,d0
                  ror.l    #1,d0
                  moveq.l  #1,d1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

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

We type/load the complete code under ASM-ONE, assemble, save the compiled binary : **RNC_1**

Or if you prefer the already compiled version, attached as a Hexa suite.

```
48 E7 FF FE 4F EF FE 80 24 4F 22 48 61 00 01 76 72 00 0C 80 52 4E 43 01 66 00 01 02 61
00 01 66 2F 40 01 80 47 E8 00 0A 2A 49 4D F5 08 00 61 00 01 54 49 F3 08 00 42 67 BB CC
64 4C 70 00 10 2B FF FE 41 F6 08 00 B1 CC 63 3E 54 4F 20 0C 08 00 00 00 67 04 52 4C 52
48 20 08 08 00 00 00 67 02 52 48 70 00 BD C8 67 08 12 20 3F 01 52 00 60 F4 3F 00 D1 C0
49 EC FF E0 4C D4 00 FF 48 E0 FF 00 B9 CB 62 F0 97 CC D7 C8 7E 00 1C 2B 00 01 E1 5E 1C
13 70 02 72 02 61 00 00 C8 20 4A 61 00 00 F2 41 F9 00 06 A2 20 61 00 00 E8 41 F9 00 06
A2 A0 61 00 00 DE 70 FF 72 10 61 00 00 A6 38 00 53 44 60 26 41 F9 00 06 A2 20 70 00 61
5E 44 80 43 F5 08 FF 23 C9 00 DF F1 80 41 F9 00 06 A2 A0 61 4A 1A D9 1A D9 51 C8 FF FC
20 4A 61 3E 53 40 6B 1A 1A DB 51 C8 FF FC 10 2B 00 01 E1 58 10 13 EF A8 72 01 EF 69 53
41 CC 81 8C 80 51 CC FF B6 BB CE 65 88 30 1F 67 06 32 1F 53 00 66 FA 2F 41 01 80 4F EF
01 80 4C DF 7F FF 4E 75 30 18 C0 46 90 58 66 F8 12 28 00 3C 9E 01 6C 02 61 30 E2 AE 10
28 00 3D 0C 00 00 02 6D 16 53 00 12 00 14 00 30 28 00 3E C0 46 9E 01 6C 02 61 12 E2 AE
05 C0 4E 75 C0 46 9E 01 6C 02 61 04 E2 AE 4E 75 DE 01 EE AE 48 46 58 4B 1C 23 E1 5E 1C
23 48 46 92 07 7E 10 9E 01 4E 75 72 03 E1 88 10 18 51 C9 FF FA 4E 75 70 1F 72 05 61 CA
53 40 6B 7C 34 00 36 00 4F EF FF F0 22 4F 70 0F 72 04 61 B6 12 C0 51 CA FF F6 70 01 E2
98 72 01 74 00 48 E7 07 00 38 03 43 EF 00 0C B2 19 66 3A 7A 01 E3 6D 53 45 30 C5 2A 02
48 45 3E 01 53 47 E3 55 E2 56 51 CF FF FA 7A 10 9A 01 EA 6E 30 C6 11 41 00 3C 1A 03 9A
04 11 45 00 3D 7C 01 53 05 EB 6E 53 46 31 46 00 3E D4 80 51 CC FF C0 E2 88 52 01 0C 01
00 11 66 AE 4C DF 00 E0 4F EF 00 10 4E 75
```

Small reminder of the loading phases of the different parts of Disk1 of Shadow Of The Beast (already seen at the end of Part14)



To the question where to copy our *RNC decruncher* in memory the answer in view of the above diagram is quite complex.

Remember the analysis of the game trackloader, it works with one or more memory tables that contain the starting position on the disk as well as the size to be trackloaded, without forgetting, at least in the 1st part of the game, the loadings of disk1, a variable DSKPTH that is also contained in this table.

Basically, the **'buffer'** area that is used for the **MFM reading** of the disk moves according to the current trackload.

If we look closely, we can see that for the Psygnosis intro, it's set at the memory address *$6A000*
Review the '**#Trackload_Base**' section in the '**Part 6 TrackLoader #1 Analysis**' section.

The length (DFF024) is set to $18B9 and the register DFF024 (ak DSKLEN), indicates the number of words to transfer per call.

So
$18B9*2= $3172
$6A000+$3172= $6D172

So the 'buffer' area for the 1st 'official' trackload of the SOTB2 animation is located: **$6A000 → $6D172**

However, if we assume that we will perform this initial trackload not with the official TrackLoader of the game but simply from the boot-sector with the **TrackDisk.device**, this makes this memory area available

In addition, as seen in the above diagram of the memory areas, it is only used on the 1st trackload.
It suits us. 😉

We can plan to place our decruncher in **$6A200**
And as it has a size of 536 Bytes, it gives us the used memory area: **$6A000 → $6A218**
We leave ourselves a little margin and we place the buffer zone of the decruncher in **$6A220**

At this point you should have the following files:

```
------------------------------------------------------
Anim_01                                  290 676 Bytes
Anim_01.RNC                              236 710 Bytes
------------------------------------------------------
Anim_02                                  298 462 Bytes
Anim_02.RNC                              235 956 Bytes
------------------------------------------------------
Psygnosys                                164 022 Bytes        ; To be modified
Psygnosys.RNC                          ~ 133 501 Bytes
------------------------------------------------------
Last_Load                                 39 800 Bytes        ; Just for information
Last_Load.RNC                          ~  21 532 Bytes        ; because we don't compress.
------------------------------------------------------
Disk1_Lower_01                           425 984 Bytes
Disk1_Lower_02                            71 728 Bytes
Disk1_Lower.bin                          497 700 Bytes
------------------------------------------------------
Disk1_Upper_01                           425 984 Bytes
Disk1_Upper_02                            71 728 Bytes
Disk1_Upper.bin                          497 700 Bytes
------------------------------------------------------
Disk2_Lower_01                           425 984 Bytes
Disk2_Lower_02                            78 016 Bytes
Disk2_Lower.bin                          504 000 Bytes
------------------------------------------------------
Disk1_Upper_01                           425 984 Bytes
Disk1_Upper_02                            71 716 Bytes
Disk2_Upper.bin                          497 700 Bytes
------------------------------------------------------
TRACKLOADER_AlphaOne_P1                      434 Bytes
RNC_1                                        522 Bytes
DISK1_P1                                  1 9480 Bytes
DISK1_P2                                  20 228 Bytes
DISK1_P3                                  16 500 Bytes
DISK1_P4                                 191 184 Bytes
DISK1_PRE                                247 392 Bytes
------------------------------------------------------
```

Well, I'm not going to talk about inserting the floppy disk xyz anymore but rather loading the file xxx.
It's up to you to juggle with your floppy and if necessary to use others (and yes, still).

Part 18 Modification of the main file : Psygnosys

The trackload of the animations and the last loading before the message 'Insert Disk 1' is done directly in the **Psygnosis** 'file

So we will work on it, **boot on the AR** and **load the ripped file**.
Normally*, this file is loaded in *$2B0*. In order to facilitate the global understanding, we will load it in : *$20000+$2B0*
** Review 'Part 8 Analyse of TrackLoader #2' to understand the operation of the Psygnosys file if necessary.*

Type in :
```
LM Psygnosis_original, 20000+2B0
```

We clean the code, we remove the **BSR** towards the routine *Return_T00*, **Type in :**
```
A 20000+2B0+C
~202BC  NOP                     ; Previously 'GoTo #Return_T00'
~202BE  NOP                     ; 2 nop to deactivate it.
~202C0  <RETURN>
```

We replace the *TrackLoader* with the one we modified from *AlphaOne*.
```
LM TRACKLOADER_AlphaOne_P1, 20000+2B0+30                    // 202E0 →2049E
```

Modify the original table with our own values, each time coded on 8 Bytes
```
M 20000+2B0+2C8
```

```
m 20000+2B0+2C8
:020578 00 03 CE 00 00 03 9C A6 00 07 6C 00 00 03 99 B4 ..........l.....
:020588 00 0D 12 00 00 00 9B 78 00 04 88 10 00 00 BB 80 .......x........
```

Review the explanation in the table above.

We now modify the 3 calls and delete what is no longer necessary.
```
A 20000+2B0+348
```

#Loading Animation Part #1 ($5F8 original address)
```
~205F8  LEA    763C0,A2         ; Conf of DSKPTH in A2 for our TrackLoader
~205FE  LEA    16B14,A0         ; A0=16B14, Adr_memory_dest
~20604  LEA    578.S,A1         ; A1=$578,  New Pointer for our address of our new table.
~20608  BSR    202E0            ; Calling our new Trackloader d'AlphaOne.
~2060C  JSR    6A000            ; Calling our decompactor which also uses A0.

                                ; Why in 6A000 ? See end of section 'part 17' for more info.

~20612  BSR    206B4            ; Calling of original decomp/decrypt.
~20616  NOP                     ; See below
~20618  NOP                     ; See below
```

#Loading Animation Part #2 ($618 original address)
```
~2061A  LEA    6C780,A2         ; Conf of DSKPTH in A2 for our TrackLoader
~20620  LEA    16B14,A0         ; A0=16B14, Adr_memory_dest, doesn't change.
~20626  LEA    580.S,A1         ; A1=$580,  New Pointer for our address of our new table
~2062A  BSR    202E0            ; Calling our new Trackloader d'AlphaOne.
~2062E  JSR    6A000            ; Calling our decompactor which also uses A0.
~20634  BSR    206B4            ; Calling our decomp/decrypt original.
~20636  NOP                     ; See below
~20638  NOP                     ; See below
```

```
For information :
~20638  LEA    256.S,A7         ; Original code, We will move it AFTER our last TrackLoad.
                                ; Because it is not a good idea to change A7 before using our routines.
```

Loading last Data of Disk1 **LastLoad-Disk1**
```
~2063A  LEA    50000,A2         ; Conf of DSKPTH in A2 for our TrackLoader
~20640  LEA    40000,A0         ; A0=40000, Adr_dest_Memory, doesn't change.
~20646  LEA    588.S,A1         ; A1=$588,  New Pointer for our address of our new table
~2064A  BSR    202E0            ; Calling our new Trackloader d'AlphaOne
~2065E  JSR    6A000            ; Calling our decompactor which also uses A0.
~20654  LEA    256.S,A7         ; We restore the original code that we had deleted in 20638
~20658  NOP                     ;
~2065A  NOP                     ; Set of NOP's To get back on our feet
~2065C  NOP                     ; in $2065E
```

```
~2065E  LEA    202C6(PC),A0     ; Original Code, doesn't change.

A 20000+70A                     ; Without omitting to add the possibility of skipping the animations
~2070A  BRA    20638            ; replace the appearance of the white background when the mouse is pressed
~2070E  NOP                     ; by a direct connection to the 'end of animations' routine.
~20710  NOP
```

And of course, we save it all, **type in** : `SM Psygnosis_mod, 20000+2B0 20000+2B0+280B6`
And we pass it through the mill of the RNC Propack compressor to obtain the file **Psygnosys_mod.RNC** (taille !133502 Octets)

Part 19a Creation of our disk 1 of SOTB2

Reminder :

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

Reboot and enter the AR then Type in:
O 00, 20000 70000

Then we write our little piece of code :
A 20000

```
2000C   NOP                         ; We keep a small margin, in case we want to add a cracktro
2000E   NOP                         ;

                                    ; #TrackLoad of Psygnosis_mod.RNC
20010   MOVE.W   #2,1C(A1)          ; Trackdisk.device in reading mode
20016   MOVE.L   #40000,28(A1)      ; Memory Destination
2001E   MOVE.L   #21000,24(A1)      ; Size to trackload (a little more)
20026   MOVE.L   #B0600,2C(A1)      ; Position Raw Disk        // Track !128 and at the beginning of the 4th sector
                                                               (so sector 3, we start from 0)
2002E   MOVEA.L  4.S,A6             ; Just in case
20032   JSR      -1C8(A6)           ; Start the TrackLoad of the 'file Psygnosis_MOD.RNC      OPCODE=4E AE FE 38


                                    ; #Re-copy code of 'RNC decruncher' in memory 6A200
20036   LEA      200E6(PC),A0       ; A0=Addr. Memory source             $200E6
2003A   LEA      6A200,A1           ; A1=Adr. Memory Destination         $6A200
20040   MOVE.L   #85,D0             ; D0= Size to copy (in nbr of Longword to copy)      // 522 Octets so $82 LongWord
20046   MOVE.L   (A0)+,(A1)+        ; → Copy Source to Destination
20048   DBF      D0,20046           ; ← D0=D0-1, loop as long D0 is different from -1


                                    ; #Decompression of animation Psygnosis_MOD.RNC
2004C   LEA      40000,A0           ; A0=Addr. File' memory source Psygnosis_MOD.RNC
20052   JSR      6A200              ; GoSub #RNC_Decrunc


                                    ; #Re-copy original code of SOTB2
20058   LEA      2008C(PC),A0       ; A0= Addr. Memory source            $2008C
2005C   LEA      7FC08C,A1          ; A1= Adr. Memory Destination$7FC08C
20062   MOVE.W   #18,D0             ; D0= Size to copy (in nbr of Longword to copy)
20066   MOVE.L   (A0)+,(A1)+        ; → Copy Source to Destination
20068   DBF      D0,20066           ; ← D0=D0-1, loop as long D0 is different from -1

2006C   MOVE.L   #6CA94,C2          ; We reposition the two important original values of SOTB2
20076   MOVE.L   #9B78,C6           ; See part 'Part 6 Analysis of TrackLoader #1' in the analyzed section #Pre_Conf_TrackLoader
20080   JMP      7F08C              ; And we jump on the original code of SOTB2 copied in memory.
;=========================
                                    ; Original Code of SOTB2
20086   NOP                         ; We copy the original code just before the launch of the intro Psygnosis
                                    ; See part 'Part 6 Analysis of TrackLoader #1' in the analyzed section #Pre_Conf_TrackLoader
                                    ; Code that will be executed BEFORE launching the trackloaded code.
20088   NOP
2008A   NOP                         ; Small margin, just in case.

2008C   MOVE.W   #$7FFF,DFF09A      ; Conf INTENA, clear all Level
20094   MOVE.W   #$7FFF,DFF096      ; Conf DMACON, clear all DMA channels and BBUSY, BZERO, BLTPRI
2009C   LEA      200A8(PC),A0       ; A0=80, address start conf supervisor stack
200A0   MOVE.L   A0,20.S            ; copy of A0 to address =$20 for the supervisor conf
200A4   MOVE.W   #2700,SR           ; Supervisor Stak =$2700
200A8   MOVE.W   #2700,SR           ; Supervisor Stak =$2700
200AC   LEA      40000,A0           ; A0=Adr source      = $40000
200B2   LEA      2B0.S,A1           ; A1=Adr Destination= $2B0
200B6   MOVE.L   (A0)+,(A1)+        ; →        Copy
200B8   CMPA.L   #68200,A0          ; Compare $68200 with A0,this is the last address when everything is copied
                                    ; $68200-$40000=$28200 size

200BE   BNE      200B6              ; ← As long as we are not at the end of the data, we copy
200C0   MOVE.W   #8210,DFF096       ; Conf DMACON, Set Disk DMA, Enable all DMA
200C8   MOVE.W   #7FFF,DFF09A       ; Conf INTENA, clear all Level
200D0   MOVE.W   #7FFF,DFF09C       ; Conf INTREQ
200D8   CLR.W    DFF180             ; We erase the background.
200DE   LEA      256.S,A7           ; A7=256, adr. of stack (identical to the original)
200E2   JMP      2B0.S              ; We start the animation

;=========================
                                    ; Code of RNC decompactor      $200E6→$202F0      Size : 522 Bytes
200E6   MOVEM.L  D0-D7/A0-A6,-(A7)
200EA   LEA      -200(A7),A7
200EE   ...
```

Insert the disk containing the necessary files into the external drive (i.e. **DF1**) and insert a **formatted disk** into **DF0** : **SOTB2_D1_CRACK**
Swap when necessary the floppy in the external drive in order to load the correct file(s).

As expected we load our RNC decompactor in **$200E6**
LM 1:RNC_1, 200E6

We re-calculate the checksum
BOOTCHK 20000

And we rewrite the whole thing on our floppy disk
WT 0 1 20000

We now tackle the previously prepared data. *(Part 15b 'Preparing the files for the creation of our cracked Disk1')*

Tape in :

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

Now the animations.

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

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

Don't forget the modified and **compacted Psygnosis** animation code

```
O 00, 20000 60000
RT !128 1 20000
LM 1:Psygnosis_mod.RNC, 20000+600    ; PsygnosisMod RNC Size=$2097E+$600=$20F7E  $20F7E/$1600= !23,98 so !24
WT !128 !24 20000
```

/!\ We will only have LastLoad_Disk1 left which we will do later!

Part 19b Creation of our disk 2 of SOTB2

Review: *'Part 15 Reorganization of the data for the creation of our disk'* if necessary.

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

Insert the disk containing the necessary files* in the external drive (thus **DF1**)
and **insert a blank floppy disk in DF0: SOTB2_D2_CRACK**

As expected we load our ripped data and write everything to our new disk.

```
O 00, 10000 80000
LM 1:Disk2_Lower_01, 10000          // Size !425984    !425984-$4C18=!406504
WT 0 !73 10000+4C18                 // !406504/!5632 = 72.2 That is 73 Tracks
                                    // Without forgetting to 'skip' the 1er $4C18 Bytes
```

So now it's a little more complicated in the sense that **you have** to continue **EXACTLY** from this data and not from the '**track'**
However, with the *action replay* we only work in 'track' mode.

What we will do is to load the last track written -1 with the above command and load our data afterwards.

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

Small calculation of the offset : !5632*!72=**405504** and !406504 - !**405504**=!**1000**
A small check is necessary.

```
M 10000+!1000-!10
```

```
m 10000+!1000-!10
:0103DE 1F 56 33 CE 00 08 00 21 F4 1E 00 00 00 00 00 00 .V3....!........
:0103EE 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:0103FE 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:01040E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:01041E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:01042E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

The following data is then loaded.
Now insert the disk containing the file Disk_lower_02 into the external drive.
Type in :

```
LM 1:Disk2_Lower_02, 10000+!1000          // Size !78016
M 10000+!1000-!10
```

```
lm 1:Disk2_lower_02, 10000+!1000
Loading from 0103E8 to 0234A8
Disk ok
m 10000+!1000-!10
:0103DE 1F 56 33 CE 00 08 00 21 F4 1E 4A 7D 95 54 78 4D .V3....!..J}.TxM
:0103EE 15 00 80 82 EA 62 BF AA 2A F4 A7 CA 9A 40 F5 62 .....b..*.§..@.b
```

Data therefore loaded up to $234A8 and presents in memory of $10000 → $234A8
This gives us **$234A8-$10000=$134A8** $134A8/$1600= !14.03 **so !15**

```
WT !72 !15 10000
```

Now we need to copy our data ripped from the original 2 UPPER disk in a row.
But, there is a large original empty area there, so of course we should not leave it and copy our data exactly after of our last data '**In_Game_62**' (review table if necessary in: *Part 15 Reorganization of the data for the creation of our disk*

Position of our future data '**In_Game_62**' in position raw-disk **$73D90-7ED9B**, that gives us:
$73D90 = !474512 474512/5632= !84,25 !84*5632= !473088 !474512-!473088= !1424
So the next data start in Track !84, offset !1424

```
O 00, 10000 80000
RT !84 1 10000
M 10000+!1424-!10
```

```
rt !84 1 10000
Disk ok
m 10000+!1424-10
:010580 00 0F 84 07 5E 5A 14 19 00 00 09 74 00 90 2E 07 ....^Z.....t....
:010590 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:0105A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

We are correct, it only remains for us to load following this data.
Now insert the disk containing the file Disk_Upper_01 into the external drive.

```
LM 1:Disk2_Upper_01, 10000+!1424      // File Size !425984 + previous data !1424 = !427408
!427408 / !5632 = !75,9               so !76 Tracks to record
```

BUT, if we look carefully at what we plan to do: *Part 15 Reorganization of the data for the creation of our disk*
We stop on the data '**In_Game_69 [END]**', positioned in our rip file **Upper D2** in **raw** position: **$61E6C-$6638B**

We check the correct positioning of the ripped data in memory, the beginning of the data **In_Game_69 [END]**

M 10000+ !14242+61E6C // Beginning of working memory area + Previous offset + Pos. Start Expected in rip file

```
m 10000+!1424+61E6C
:0723FC 00 00 45 21 C3 14 0C 56 F1 48 B5 7C E0 B2 34 96 ..E!...V.H.|..4.
```

We fall well on our 'signatures' hexa previously raised, it is correct.

All that remains is to 'clean' our memory area at the end of this data, namely : $1000 + !1424 +$6638B+1
(Work memory area + previous offset + End of work data **In_Game_69 [END]** + 1)
O 00, 10000+!1424+6638B+1 80000 **// $7691C → $80000**

calculate how many tracks we have to write on our disk
($7691C-$10000)/$1600 = $4A exactly !74,6 so !75 or $4B, your choice.

WT !84 !75 10000
As expected, we arrive almost at the end of the disk (Track 79 in this case).
We only have to write the last data in a row, namely : **Load_Menu_08**
Data that are on the **original disk 2** on the **UPPER side** at the end of the disk, so in our rip file: **Disk2_Upper_02**

Now insert the disk containing the file Disk_Upper_02 into the external drive.
Okay, let's make this last data simpler.
We know its Hexa signature and we know its size, so we will load our dump file in memory
Search for our hex string, add its 'size' to its found position and save it to our external disk.

Hexa Signature : **4A B8 04 1E 67 16 20 78 04 1E 20 B8 04 26 20 78 04 22 20 B8 04 2A 70 00 21 C0 04 1E 41 F8 0B 1A**
Size : $1EBC

Type in :
O 00, 10000 80000
LM 1:Disk2_Upper_02, 10000
F 4A B8 04 1E 67 16 20 78 04 1E 20 B8 04 26 20 78

```
dir 1:
Directory of (empty)upper_02
       425984  Disk2_upper_01disk used
       071716  Disk2_upper_02
0722 blocks free, 58.1 % of disk used
Disk ok

lm 1:Disk2_upper_02, 10000
Loading from 010000 to 021824
Disk ok

f 4A B8 04 1E 67 16 20 78 04 1E 20 B8 04 26 20 78
Search from: 000000 to: 080000
013290
Ready.
sm 1:Load_Menu_08, 13290 13290+1EBC
Disk ok
```

And we save it all : **SM 1:Load_Menu_08, 13290 13290+1EBC**

Now we have to save these data after our data on our disk2.
Still based on our table, it should be present on our disk in raw position : $DA11C-$DBFD7
$DA11C = !893212 !893212 / !5632 = 158,6 !158 * !5632= !889856
!893212 - !889856 = !3356 **so Ofsset $D1C**

Type in :
O 00, 10000 80000
RT !158 1 10000
M 10000+D1C-10

```
o 00, 10000 80000
Ready.
rt !158 1 10000
Disk ok
m 10000+D1C-10
:010D0C 01 83 F8 87 19 A5 D9 FD 00 00 53 3C 00 CA 3C C0 ..........S<..<.
:010D1C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
:010D2C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Okay, we're still correct.

We load our file in the continuation and we save the whole, let's go.
LM 1:Load_Menu_08, 10000+D1C

$D1C + $1EBC = $2BD8 = !11224 !11224 / 5362 = 1,9 **so 2 Tracks**
WT !158 2 10000

```
dir 1:
Directory of (empty)
      425984  Disk2_upper_01
      071716  Disk2_upper_02
      007868  Load_Menu_08
0704 blocks free, 60.0 % of disk used
Disk ok

lm 1:Load_Menu_08, 10000+D1C
Loading from 010D1C to 012BD8
Disk ok

wt !158 2 10000
Disk ok
```

Pfiouuuu, It's done for this section 😊

Part 20a Analysis 'Free memory occupation' to insert our Final TackLoader

It's necessary to return on the analysis of the last Loading of Disk1.
If we summarize the memory positions of the different sections of this part of the code, it gives us the following table:
The goal is to understand the memory space occupied by each part and to find a 'free' place where we can insert our modified *TrackLoader*.

		Called by				
Memory Adr	**Sub-routine**	**INSERT DISK 2**	**IN GAME**	**Info**	**In the area potential ?**	**Zone Potential**
256-268	#WaitSyncV	4F016	4F016	REQUIRED	N/A	No
26A-284	#Working_on_BE2_Mark	262	many times	REQUIRED	N/A	No
286-294	#Working_on_BE2_Mark_#x	276	276	REQUIRED	N/A	No
446-BD6						No
						No
1824-1956	#Decomp/Decrypt	many times	many times	REQUIRED	N/A	No
1958-19C2	#Decomp/Decrypt_02	many times	many times	REQUIRED	N/A	No
19C4-1A16	#Trackloader_2_Init	1B40 1E48 4ECC 8CE6 41BF2	1B40 1E48 4ECC 8CE6 41BF2	Not required	TO PATCH	
1A18-1A3E	#Move Inside.	1A84 1A8E 1B80	1A84 1A8E 1B80	Not required	Yes	
1A6C-1A82	#Position Reach?	1B6E	1B6E	Not required	Yes	
1A84-1A86	#Forward_Backward	1A70	1A70	Not required	Yes	
1A88-1A94	#GoTo_Position	1A72	1A72	Not required	Yes	
1A96-1AA8	#Return_T00	1AA8 1E4C 1E80 4ED0 8CEA 41BF6	1AA8 1E4C 1E80 4ED0 8CEA	Not required	TO PATCH	
1ABC-1AC8	#WAIT	1A30 1A5A 1AA4	1A30 1A5A 1AA4		Yes	
1AAA-1ABA	#Disk Ready?	1AA0	1AA0	Not required	Yes	
1A42-1A68	#Move **Outside**.	1A7C 1A86 1AA2	1A7C 1A86 1AA2	Not required	Yes	
1ACA-1AF0	#DF0__SIDE_DOWN__MOTOR_OFF__DIR_EXT	many times	many times	Not required	TO PATCH	
1AF2-1B0E	#DF0__SIDE_DOWN__MOTOR_ON__DIR_EXT	1B4E	1B4E	Not required	Yes	
1B10-1B2C	#DF0__SIDE_UP__MOTOR_ON__DIR_EXT	1B52	1B52	Not required	Yes	
1B2E-1B36	#Trackloader_Start	many times	many times	Not required	TO PATCH	
1B38-1B40	#Read_Table_and_Start_Trackload_Or_Not	1B30	1B30	Not required	Yes	
1B44-1B50	#Side_Select	1B3E	1B3E	Not required	Yes	
1B52-1B84	#Recover_Info_to_Trackloader_x	1B4C	1B4C	Not required	Yes	
1B86-1BC0	#Trackload_Base	1B7E	1B7E	Not required	Yes	
1BC8-1C1E	#Test CIA Ready	N/A	N/A	Not required	N/A	
1C20-1C32	#Processing_MFM_BASE	1C10	1C10	Not required	Yes	
1C34-1C3C	#Check_Processing_MFM	1C28	1C28	Not required	Yes	
1C3E-1C5A	#MFM bit even processing	1C36	1C36	Not required	Yes	
1C5C-1CDC	#Loading_Phase_#1	1E62	1E62	REQUIRED	N/A	No
1CE0-1CE4	#Data_already_loaded?	1EA8 256C 7CE6	1EA8 256C 7CE6 4D86C	REQUIRED	N/A	No
1CE6-1D46	#Loading_Phase_#2_2/2	N/A	N/A	REQUIRED	N/A	No
1D48-1DDE	#LoadPhase_#1_End_Part1/2	1E7C	1E7C	REQUIRED	N/A	No
1DE0-1E62	#Base_LastLoad	256	256	REQUIRED	N/A	No
1E66-201E	#Loading_Phase_#2_1/2	N/A	N/A	REQUIRED	N/A	No
1E74-201E	#Loading_Phase_#2_1/2_bis	N/A	N/A	REQUIRED	N/A	No

22FE-2332	#BLITTER_CONF_#01	1F8A 1FC6 2030 2080 210E 211C 212A 2138 2226 2266 22F2	1F8A 1FC6 2030 2080 210E 211C 212A 2138 2226 2266 22F2	**REQUIRED**	N/A	**No**
2334-237C	#BLITTER_CONF_#02	1F8E 1FCA 2034 2084 2112 2120 212E 213C 222A 226A 22F6 **4C408**	1F8E 1FCA 2034 2084 2112 2120 212E 213C 222A 226A 22F6	**REQUIRED**	N/A	**No**
237E-23B2	#BLITTER_CONF_#00	1FBC 2020 20FA 21B2 21BC 21C6 21D0 21E4 21EE 2216 2256 22A8 22C2	1FBC 2020 20FA 21B2 21BC 21C6 21D0 21E4 21EE 2216 2256 22A8 22C2	**REQUIRED**	N/A	**No**
23B4-23DE	#START_LEVEL1_OR_NOT	1F92 2054	1F92 2054	**REQUIRED**	N/A	**No**
24CC-254E	#LOADING_LEVEL1	**many times**	N/A	**REQUIRED**	N/A	**No**
						No
3FCC-3FFC	#Update_Table_10E4_10F0__1094_10A0	7B76	7B76	**REQUIRED**	N/A	**No**
41FA-422A	#Update_Table_CDC_CEC__D2C_D38	7B72	7B72	**REQUIRED**	N/A	**No**
						No
5710-5752	#Decrypt_RAW	1EBE	1EBE	**REQUIRED**	N/A	**No**
5AC4-5ADC	#Update_Table_A0_With_3A0&3E0__E58_1210_E5C_1214	5B0E 5B38	5B0E 5B38	**REQUIRED**	N/A	**No**
5B10-5B38	#JSR(A0)_or_Erase	5AB6 7B7A	5AB6 7B7A	**REQUIRED**	N/A	**No**
						No

It seems that the **$19C4 → $1C5A** area is a correct candidate.

7B2E-7B8E	#RTZ_Tables	1DA2 4E3A 7EF4	1DA2 4E3A 7EF4	REQUIRED	N/A	No
8C30-8C8A	#BASE_INSERT_DISK_ASKED...	1E50 4EDC 7C48 41BFA	1E50 4EDC 7C48 41BFA	REQUIRED	N/A	No
8C8C-8D0A	#CHECK_DISK...	8C82	8C82	REQUIRED	N/A	No
8C92-8CD4	#CHECK_DISK_A0	8C8A	8C8A	REQUIRED	N/A	No
8CDC-8D00	#DISK2_OR_DISK1_INSERTED?	8D08 8D12	8D08 8D12	REQUIRED	N/A	No
8D02-8D0A	#Check_Signature_DISK_02	N/A	N/A	REQUIRED	N/A	No
8D0C-8D14	#Check_Signature_DISK_01	8D00	8D00	REQUIRED	N/A	No
9B24-9B3C	#RTZ_TRACKLOADER	254E 7266 7E48	254E 7266 7E48	REQUIRED	TO PATCH	No
9B40-9B88	#Pre_Base_TrackLoadX	9B28	9B28	Not required	TO PATCH	
9BA6-9BB8	#Floppy_Ready?	9BA0 9BDC 9C1C	9BA0 9BDC 9C1C	Not required	TO PATCH	
9B8A-9BA4	#Motor_ON	9B2C	9B2C	Not required	TO PATCH	
9BBA-9BE0	#Return_T00	9B30 9BDA 9BEC	9B30 9BDA 9BEC	Not required	TO PATCH	

It also seems that the **$9B40 → $9BA4** area is also available but in a smaller size.

19C4 → $1C5A = $296 = !662 Bytes Available
9B40 → $9BE0 =$A0 = !160 Bytes Available

Part 20b Modification and compilation of the AlphaONE v2 Trackloader - Phase2

This time we will take the **trackload** of **AlphaOne v2 of 2005**, it allows an addressing on the even and odd bytes.
As we have reorganized the positioning of the datas on our crack floppy disks, it will be necessary either to modify all the calls.
OR **to keep the original tables of the calls and to patch directly the new real positions.**

DISK2 – Cracked Version

		ORIGINAL		Position	Into	
Order Calling	Calling Address	Disk	SIDE	Disk Real	Our cracked Disk	Delta between Pos.Orig and Pos.Real
Load_Menu_11	1EA0	D2	LOW	04C18-07333	00000-0271B	-4C18
Load_Menu_12 In_Game_50	1CF0	D2	LOW	07334-07B2F	0271C-02F17	-4C18
Load_Menu_13 In_Game_51	1D02	D2	LOW	07B30-16507	02F18-118EF	-4C18
Load_Menu_14 In_Game_52	1D14	D2	LOW	16508-1790F	118F0-12CF7	-4C18
Load_Menu_15 In_Game_53	1D26	D2	LOW	17910-1871F	12CF8-13B07	-4C18
Load_Menu_16 In_Game_54	1D38	D2	LOW	18720-19267	13B08-1464F	-4C18
Load_Level1_01	24DE	D2	LOW	19268-25D0F	14650-210F7	-4C18
In_Game_61	762C					
In_Game_55-RET	7800					
In_Game_31-A	7966					
In_Game_44	7C80					
Load_Level1_02	24F0	D2	LOW	25D10-26B93	210F8-21F7B	-4C18
In_Game_63	763E					
In_Game_45	7C9A					
In_Game_56-RET	7812					
In_Game_12	8A9C					
Load_Level1_03	2508	D2	LOW	26B94-3495B	21F7C-2FD43	-4C18
In_Game_46	7CAC					
In_Game_57-RET	7824					
In_Game_31-B	7980					
Load_Level1_04	251A	D2	LOW	3495C-38583	2FD44-3396B	-4C18
In_Game_58-RET	7836					
In_Game_31-C	7992					
In_Game_48	7CD0					
Load_Level1_06	2542	D2	LOW	38584-38923	3396C-33D0B	-4C18
In_Game_47	7CBE					
In_Game_30	8244					
In_Game_22	84C2					
In_Game_14	8AC0					
In_Game_65	7668					
Load_Level1_05	2530	D2	LOW	38924-39A9B	33D0C-34E83	-4C18
In_Game_49	7CDE					
In_Game_31-D	79A0					
In_Game_18	7A3C					
In_Game_10	8A70	D2	LOW	39A9C-3E9A3	34E84-39D8B	-4C18
In_Game_01	719A					

In_Game_02	71AC	D2	LOW	3E9A4-43E43	39D8C-3F22B	-4C18
In_Game_11	8A82					
In_Game_29	8232					
In_Game_21	84B0					
In_Game_23	7AD0	D2	LOW	43E44-4F0AF	3F22C-4A497	-4C18
In_Game_13	8AAE					
In_Game_03	71C6					
In_Game_04	71D8	D2	LOW	4F0B0-52AE3	4A498-4DECB	-4C18
In_Game_24	7AE2					
In_Game_15	7A0A	D2	LOW	52AE4-5A8AB	4DECC-55C93	-4C18
In_Game_16	7A1C	D2	LOW	5A8AC-5EA73	55C94-59E5B	-4C18
In_Game_17	7A2E	D2	LOW	5EA74-6305F	59E5C-5E447	-4C18
In_Game_19	8360	D2	LOW	63060-6659B	5E448-61983	-4C18
In_Game_28	8150	D2	LOW	6659C-66B23	61984-61F0B	-4C18
In_Game_20	8372	D2	LOW	6659C-66B23		
In_Game_25	811A	D2	LOW	66B24-68807	61F0C-63BEF	-4C18
In_Game_26	812C	D2	LOW	68808-689E7	63BF0-63DCF	-4C18
In_Game_27	813E	D2	LOW	689E8-69543	63DD0-6492B	-4C18
In_Game_05 In_Game_09-RET In_Game_59 In_Game_60	7E68	D2	LOW	69544-6C42F	6492C-67817	-4C18
In_Game_06	8918	D2	LOW	6C430-6F073	67818-6A45B	-4C18
In_Game_07	892A	D2	LOW	6F074-6FBE3	6A45C-6AFCB	-4C18
In_Game_08	893C	D2	LOW	6FBE4-6FCFF	6AFCC-6B0E7	-4C18
In_Game_54b	7E68	D2	LOW	6FD00-73F87	6B0E8-6F36F	-4C18
In_Game_60	74FA	D2	LOW	73F88-781AF	6F370-73597	-4C18
In_Game_61	750C	D2	LOW	781B0-7837F	73598-73767	-4C18
In_Game_62	751E	D2	LOW	78380-789A7	73768-73D8F	-4C18
In_Game_67	768C	D2	UP	85D04-90D0F	73D90-7ED9B	-11F74
In_Game_57	709E					
In_Game_58	70B0	D2	UP	90D10-93DE3	7ED9C-81E6F	-11F74
In_Game_56	708C	D2	UP	93DE4-9425B	81E70-822E7	-11F74
In_Game_66	767A					
In_Game_55	7074	D2	UP	9425C-99B6F	822E8-87BFB	-11F74
In_Game_64	7650					
N/A	78E4			9425C-99B6F		
END_01	4F7E	D2	UP	99B70-99E77	87BFC-87F03	-11F74
END_02	4F90	D2	UP	99E78-CC8FB	87F04-BA987	
END_03	5010	D2	UP	CC8FC-DC073	BA988-CA0FF	-11F74
END_04	501E	D2	UP	DC074-E2E77	CA100-D0F03	
In_Game_68	78B0	D2	UP	E2E78-E7B0F	D0F04-D5B9B	-11F74
Load_Menu_09	1D5E	D2	UP	E7B10-E7B6F	D5B9C-D5BFB	
In_Game_69 [END]	85EE	D2	UP	E7B70-EC08F	D5BFC-DA11B	-11F74
				EC090-F0F93		
Load_Menu_08	1E70	D2	UP	F0F94-F2E4F	DA11C-DBFD7	-16E78

DISK1 – Cracked Version

Order Calling	Calling Address	ORIGINAL Disk	 SIDE	Position into Disk Real	 Our cracked Disk	Delta between Pos.Orig and Pos.Real
Boosecteur + code						
Load_Menu_10	1E8E	D2	LOW	00000-04C18	0400-05017	+400
Load_Menu_01	1C66	D2	UP	EC090-EFB1F	05018-08AA7	-E7078
Load_Menu_02	1C78	D2	UP	EFB20-EFD6F	08AA8-08CF7	-E7078
Load_Menu_03	1C8A	D2	UP	EFD70-F01A7	08CF8-0912F	-E7078
Load_Menu_04	1C9C	D2	UP	F01A8-F02CB	09130-09253	-E7078
Load_Menu_05	1CAE	D2	UP	F02CC-F092F	09254-098B7	-E7078
Load_Menu_06	1CC6	D2	UP	F0930-F0B97	098B8-09B1F	-E7078
Load_Menu_07	1CD8	D2	UP	F0B98-F0F93	09B20-09F1B	-E7078
In_Game_37	72F0	D1	LOW	76FC0-7B033	09F1C-0DF8F	-6D0A4
In_Game_43	7C44	D1	UP	CFF58-D2763	0DF90-1079B	-C1FC8
In_Game_32	7E68					
In_Game_33	72A8	D1	UP	D2764-DEB2B	1079C-1CB63	-C1FC8
In_Game_34	72BA	D1	UP	DEB2C-DF04F	1CB64-1D087	-C1FC8
In_Game_35	72CC	D1	UP	DF050-E056B	1D088-1E5A3	-C1FC8
In_Game_36	72DE	D1	UP	E056C-EACA7	1E5A4-28CDF	-C1FC8
In_Game_38	7308	D1	UP	EACA8-EB497	28CE0-294CF	-C1FC8
In_Game_39	731A	D1	UP	EB498-FB8A7	294D0-398DF	-C1FC8
In_Game_40	732C	D1	UP	FB8A8-FD0CF	398E0-3B107	-C1FC8
In_Game_41	733E	D1	UP	FD0D0-FDEDF	3B108-3BF17	-C1FC8
In_Game_42	7350	D1	UP	FDEE0-FEA27	3BF18-3CA5F	-C1FC8
					3CC60	

Info		Disk Crack	SIDE Orig.	Disk Real	Our cracked Disk	Delta between Pos.Orig and Pos.Real
Anim part #01		D1	LOW	0189C-4880F	Not concerned - TrackDiskDevice	
Anim part #02		D1	UP	85D04-CEAE1	Not concerned - TrackDiskDevice	
Anim Psygnosis		D1	LOW	48810-6CA93	Not concerned - TrackDiskDevice	
Last_Load_Disk1		D1	LOW	6CA94-7660B	Not concerned - TrackDiskDevice	

This gives us 8 possibilities : (7 + the disk signature which does not change **D2**)

```
If      (A1)                   = 189C  then    +0           // Signature Disk, remains in, $189C
If       A1                    = 180E  then    D2=D2-16E78
If       A1                    = 161E  then    D2=D2-6D0A4
If       A1                    = 1676  then    D2=D2+400
If       A1 between 1626 and 166E      then    D2=D2-C1FC8
If       A1 between 167E and 1776      then    D2=D2-4C18
If       A1 between 177E and 17CE      then    D2=D2-11F74
If       A1 between 17D6 and 1806      then    D2=D2-E7078
```

The option chosen here is to keep the original **raw.pos** and to patch them according to the above conditions.
In addition, as seen at *chapitre 14B*, it will be much less disruptive to the operation of the game using the Track marker.
Because, as we have reorganized the data on our Hack disks, doing an actual track count would serve **NO PURPOSE**.
Or you would have to patch every check made in **C20** in the whole game code.

The trick is simple, if you look carefully, no check is made on **C20** during the trackload, it is just updated according to **the head movement** +1 or -1 according to the displacement, or even a reset to zero if you are on T00

It is then sufficient to calculate on which track the original **trackload** should have ended.
It's easy as we keep the original **raw.pos** and we know the **SIZE** of the data to **trackload**.
It is enough to add the two to know the **raw.pos final** and divide it by the size of the custom track: 189C
Then copy this value to the address **$C20**

Note that the original game, the original trackload works by **SIDE** with a limit value of $84468
Therefore, everything smaller than $84468 will not need any correction in our future 'Original Track' calculation routine
And all that is greater than $84468 will need an adjustment, namely to perform a subtraction of 84468 from **raw.pos final** before dividing by 189C.

Note also that if we can integrate in our custom trackloader code the original routines that we have overwritten and that are useful, it would be nice.
To review:

1A96	Because called many times	-> **#Return_T00**	→ **RTZ $C20 and RTS**
1ACA	Because called many times	-> **#DF0__SIDE_DOWN__MOTOR_OFF__DIR_EXT**	→ **Simply RTS**
1B2E	Because called on each trackload	-> **#TrackLoader_2** → **to redirect to the beginning of our trackload routine**	

We will use the second part of free in memory **9B40 → $9BE0** for our **raw.pos** patches. (because our code is small and this area too)
This gives us the following code for the **Raw.Pos** modification as seen above.

DELTA.s **// 150 Bytes**

```
                                                    ; /!\ to load in $9B40
DELTA:                                              ; at this point
                                                    ; (A1) = Adr of table     and D2 = Raw.Pos of table
;-------------------------------------------
CASE1:   CMP.W     #$1676,A1                        ; Compare A1 with 1676 - LoadMenu_10
                   BNE       CASE2                  ; If different, we continue to → CASE2
                   ADD.L     #$400,D2               ; Otherwise, we add $400 to D2
;-------------------------------------------
CASE2:   CMP.W     #$180E,A1                        ; Compare A1 with 180E - LoadMenu_08
                   BNE       CASE3                  ; If different, we continue to → CASE3
                   SUB.L     #$16E78,D2             ; Otherwise, we subtract $16E78 of D2
;-------------------------------------------
CASE3:             CMP.W     #$161E,A1              ; Compare A1 with 161E - Crystal Cavern 06/11
                   BNE       NEXT                   ; If different, we continue to → NEXT
                   SUB.L     #$6D0A4,D2             ; Otherwise, we subtract $16E78 from D2
;-------------------------------------------
NEXT:
                   CMP.L     (A1),D2                ; Check if D2 is different (A1)
                   BNE       DONE                   ; If this is the case, is that none of the above cases occurred
                                                    ; and in this case, we jump directly to the end of the code.
;------------------------------------------------------------------------------------------------------------------
SIGNATUREDISK:
                   CMP.L     #$189C,(A1)        ; Compare Raw.Pos with 189C which corresponds to Raw.Pos of the Disk signature
                   BEQ       DONE               ; If identical, we don't change D2 and we jump to the end of our routine.
;------------------------------------------------------------------------------------------------------------------------

CASE4:             CMP.W     #$17D6,A1              ; Compare A1 with 17D6
                   BLT       CASE5                  ; If smaller then, we continue to → CASE5
                   SUB.L     #$E7078,D2             ; Otherwise D2=D2-E7078
                   BRA       DONE

CASE5:             CMP.W     #$177E,A1              ; Compare A1 with 177E
                   BLT       CASE6                  ; If smaller then, we continue to → CASE6
                   SUB.L     #$11F74,D2             ; Otherwise D2=D2-11F74
                   BRA       DONE

CASE6:             CMP.W     #$167E,A1              ; Compare A1 with 167E
                   BLT       CASE7                  ; If smaller then, we continue to → CASE7
                   SUB.L     #$4C18,D2              ; Otherwise D2=D2-4C18
                   BRA       DONE

                   NOP                              ; To get back on our feet in 9BBA
                   NOP
                   NOP
                   NOP
                   NOP
                   ;-------------------------------------------------------------------------------------
                   CLR.W     $C20                   ; RTZ of the original track counter
                   RTS                              ; Cancellation of the routine in 9BBA
                   ;-------------------------------------------------------------------------------------

CASE7:             CMP.W     #$1626,A1              ; Compare A1 with 1626
                   BLT       DONE                   ; If smaller then, wn continue to → DONE
                   SUB.L     #$C1FC8,D2             ; Otherwise D2=D2-C1FC8
                   BRA       DONE


DONE:              RTS                              ; back with in D2 the real Raw.Pos.

END:
```

If you prefer the already compiled version, attached as a Hexa suite.

DELTA

```
B2 FC 16 76 66 00 00 08 06 82 00 00 04 00 B2 FC 18 0E 66 00 00 08 04 82
00 01 6E 78 B2 FC 16 1E 66 00 00 08 04 82 00 06 D0 A4 B4 91 66 00 00 66
0C 91 00 00 18 9C 67 00 00 5C B2 FC 17 D6 6D 00 00 0C 04 82 00 0E 70 78
60 00 00 4A B2 FC 17 7E 6D 00 00 0C 04 82 00 01 1F 74 60 00 00 38 B2 FC
16 7E 6D 00 00 1E 04 82 00 00 4C 18 60 00 00 26 4E 71 4E 71 4E 71 4E 71
4E 71 42 79 00 00 0C 20 4E 75 B2 FC 16 26 6D 00 00 0C 04 82 00 0C 1F C8
60 00 00 02 4E 75
```

And the use of the **19C4 → $1C5A** memory range for **our custom** *trackloader*.
The code is commented for a better understanding ☺

TRACKLOADER_AlphaOnev2[2005].s **// 566 Bytes**

```
init:   move.w  $dff01c,oldintena
        move.w  $dff01e,oldintreq
        bset    #7,oldintena
        bset    #7,oldintreq
        move.w  #$7fff,$dff09a          ; interrupts aus.
        move.w  #$7fff,$dff09c          ;
;*****************************************************************
        lea     $dff000,a6
        lea     buffer(pc),a0
        lea     mfmbuffer,a2
        move.l  #5*$1600,d0
        move.l  #0,d1
        move.l  #$0,d2
        jsr     TRACKLOADER
;*****************************************************************
        move.w  oldintena(pc),$dff09a
        move.w  oldintreq(pc),$dff09c
        moveq   #0,d0
        rts
oldintena:      dc.w    0
oldintreq:      dc.w    0
buffer:         blk.b   130000,0
mfmbuffer:      blk.w   6400,0


; ===========================================================================
; HARDWARE-DISKLOADER (c) ALPHA ONE 2005. v2.0
; ***************************************
; IMPROVED TO READ FROM ODD BYTEPOSITIONS.
; PRO VERSION -> WITH TRACKCOUNTER.
; IN: A6=$DFF000
;     A2=MFMBUFFER.L
;     A0=BUFFER.L
;     D0=LENGTH.L
;     D1=TRACKNR.L
;     D2=BYTEOFFSET.L
TRACKLOADER:
        ;-----------------------------------------------------------------   /!\ to load in $19FA
        ; MOD for Shadow Of The Beast 2
        ;-----------------------------------------------------------------
        MOVE.L  $C1A,A2                 ; Retrieve DSKPTH from C1A
        LEA             $DFF000,A6      ; Required
        MOVE.L  (A1),D2                 ; Retrieve Raw Pos
        MOVE.L  D2,D4                   ; Copy Raw Pos. to D4
        MOVE.L  4(A1),D0                ; Retrieve Length
        ADD.L   D0,D4                   ; D4 = Final.Position after Trackload

        CMP.L   #$84468,D4              ; Compares with the value of the original 'side' limiter.
        BLT     CORRECTION              ; If it is smaller, then we skip the next steps
        SUB.L   #$84468,D4              ; Otherwise we remove from the equation 'a complete side'.

CORRECTION:
        DIVU.W  #$189C,D4               ; D4 / Custom size
        AND.L   #$FF,D4                 ; Clean D4
        MOVE.W  D4,$C20                 ; Save 'FinalTrack' for SOTB2
        ;-----------------------------------------------------------------
        JSR     $9B40                   ; GoSub #CalculDelta RawPos real      (9B40 -> 9BB2)
        ;-----------------------------------------------------------------
        MOVE.L  D2,D1                   ; Copy Raw Pos. to D1
        DIVU.W  #$1600,D1               ; D1 / Real Size concerned
        AND.L   #$FF,D1                 ; Clean D1 // TRACKNR.L
        MOVE.L  D1,D3                   ; Copy D1 to D3
        MULU.W  #$1600,D3               ; Delta
        SUB.L   D3,D2                   ; Offset calculation   // BYTEOFFSET.L
        ;-----------------------------------------------------------------
                LEA     $BFD100,A4              ; DRIVESELECT REGISTER
                LEA     $BFE001,A5              ; DRIVESTATUS REGISTER
                LEA     CURRENTTRACK(PC),A3     ; HOLDS THE ACTUAL TRACKPOS
                MOVEQ   #0,D7                   ; D7 = BYTECOUNTER
                ADD.L   D2,D0                   ; BYTES TO READ + BYTEOFFSET
                MOVE.L  D0,D3
                DIVS.W  #$1600,D3
                SWAP    D3                      ; --> Add, bug correction on track position
                TST.W   D3                      ;
                BNE.B   WITHOUT                 ;
                SWAP    D3                      ;
                SUBQ.B  #$01,D3                 ;
                BRA     WITH                    ;
WITHOUT:        SWAP    D3                      ;
```

```
WITH:           MOVE.B  D3,1(A3)               ; <--
                MOVE.B  #$7D,(A4)              ; SWITCH MOTOR DRIVE 0 ON
                NOP                            ; =======================
                NOP
                MOVE.B  #$75,(A4)
                BSR.W   DELAY
                BSR.W   DISKREADY
                CMP.B   #$FF,(A3)
                BNE.B   MOVETOCYLINDER
                BRA             NEXT
                ;-----------------------------------------------------------------
                CLR.W   $C20                   ; RTZ Track counter
                RTS                            ; Cancellation $1A96
                ;-----------------------------------------------------------------
NEXT:
                MOVE.B  #0,(A3)
                BSET    #1,(A4)                ; CHOSE DIRECTION TOWARDS 0
MOVETOZERO:     BTST    #4,(A5)                ; HEAD ON CYLINDER 0?
                BEQ.B   MOVETOCYLINDER
                BSR.W   MOVEHEAD               ; MOVE HEAD + DELAY
                BRA.B   MOVETOZERO
MOVETOCYLINDER: BSET    #2,(A4)                ; CHOOSE HEAD HIGH
                BTST    #0,D1                  ; EVEN OR ODD TRACK?
                BEQ.B   HEADOK                 ; TRACK IS EVEN, ALRIGHT!
                BCLR    #2,(A4)                ; CHOOSE HEAD LOW
HEADOK:         MOVEQ   #0,D3
                MOVEQ   #0,D5
                MOVE.B  D1,D3                  ; NEW TRACKNUM -> D3

                BRA             NEXT2
                ;---------------------------------------------------
                RTS                            ; Cancellation $1ACA
                ;---------------------------------------------------

NEXT2:
                MOVE.B  (A3),D5                ; CURRENT TRACK -> D5
                MOVE.B  D1,(A3)                ; REPLACE CURRENT TRACK
                LSR.W   #1,D3                  ; GET CYLINDER NUM NEW TRACK
                LSR.W   #1,D5                  ; GET CYLINDER NUM CUR TRACK
                SUB.W   D3,D5
                BEQ.B   READTRACK
                BMI.B   OTHERDIRECTION
                BSET    #1,(A4)                ; CHOOSE DIRECTION TOWARDS 0
                BRA.B   CHOSEN

OTHERDIRECTION: BCLR    #1,(A4)                ; CHOOSE DIRECTION OUTWARDS 0
                NEG.W   D5
CHOSEN:         BSR.W   MOVEHEAD
                SUBQ.B  #1,D5
                BNE.B   CHOSEN
READTRACK:      BSR.W   DISKREADY              ; EINEN TRACK LESEN
                MOVE.W  #$8210,$96(A6)         ; =================
                MOVE.W  #$7F00,$9E(A6)
                MOVE.W  #$8500,$9E(A6)
                MOVE.W  #$4489,$7E(A6)
                MOVE.W  #$4000,$24(A6)
                MOVE.L  A2,$20(A6)
                MOVE.W  #$9900,$24(A6)
                MOVE.W  #$9900,$24(A6)
                MOVE.W  #$2,$9C(A6)
                NOP             ; Pour retomber sur nos pattes en 1B2E
                NOP             ;
                BRA             TRACKREADY
                ;---------------------------------------------------------
                MOVEM.L D0-D7/A0-A6,-(A7)      ; Save all the registers
                BSR     TRACKLOADER            ; Patch to keep calls in $1B2E (origin of SOBT2)
                MOVEM.L (A7)+,D0-D7/A0-A6      ; and we restore it all.
                RTS
                ;---------------------------------------------------------

TRACKREADY:     BTST    #1,$DFF01F
                BEQ.B   TRACKREADY
                MOVE.W  #$4000,$24(A6)
                MOVEQ   #0,D5                  ; DECODE TRACK

DECODE:         MOVE.L  A2,A1                  ; ============
                MOVE.L  #$55555555,D4
```

```
FINDSYNC:       CMP.W   #$4489,(A1)+
                BNE.B   FINDSYNC
                CMP.W   #$4489,(A1)
                BEQ.B   FINDSYNC
                MOVE.L  (A1),D3
                MOVE.L  4(A1),D1
                AND.L   D4,D3
                AND.L   D4,D1
                ASL.L   #1,D3
                OR.L    D1,D3
                ROR.L   #8,D3
                CMP.B   D5,D3
                BEQ.B   SECTORFOUND
                ADD.L   #1086,A1
                BRA.B   FINDSYNC

SECTORFOUND:    ADD.L   #56,A1
                MOVE.L  #(512/4)-1,D6
DECODESECTOR:   MOVE.L  512(A1),D1
                MOVE.L  (A1)+,D3
                AND.L   D4,D3
                AND.L   D4,D1
                ASL.L   #1,D3
                OR.L    D1,D3
                LEA     STORE(PC),A3
                MOVE.L  D3,(A3)
                MOVEQ   #4-1,D3

LOOP:           CMP.L   D7,D0
                BEQ.B   READREADY
                CMP.L   D7,D2
                BGT.B   BELOW
                MOVE.B  (A3),(A0)+
BELOW:          ADDQ.L  #1,A3
                ADDQ.L  #1,D7
                DBF     D3,LOOP
                DBF     D6,DECODESECTOR
                ADDQ.B  #1,D5
                CMP.B   #11,D5
                BNE.W   DECODE
TRACKDONE:      CMP.L   D7,D0                   ; TRACK DONE, GET ONTO NEXT.
                BEQ.B   READREADY               ; ===========================
                BTST    #2,(A4)
                BEQ.B   ONTONEXT
                BCLR    #2,(A4)
                BSR.W   DELAY
                BRA.W   READTRACK
ONTONEXT:       BSET    #2,(A4)
                BSR.W   DELAY
                BCLR    #1,(A4)
                BSR.W   MOVEHEAD
                BRA.W   READTRACK
READREADY:      MOVE.B  #$FD,(A4)               ; SWITCH MOTOR DRIVE 0 OFF
                NOP                             ; =========================
                NOP
                MOVE.B  #$E7,(A4)
                LEA     CURRENTTRACK(PC),A3
                MOVE.B  1(A3),D0
                ADD.B   D0,(A3)
                RTS
DELAY:          CLR.B   $300(A4)                ; DELAY ROUTINE
                MOVE.B  #$19,$400(A4)           ; =============
                MOVE.B  #$1,$D00(A4)
WAIT:           BTST    #0,$C00(A4)
                BEQ.B   WAIT
                BCLR    #0,$D00(A4)
                RTS
DISKREADY:      BTST    #5,(A5)                 ; WAIT FOR DISK-READY
                BNE.B   DISKREADY               ; ===================
                RTS
MOVEHEAD:       BCLR    #0,(A4)
                BSR.W   DELAY
                BSET    #0,(A4)
                BSR.W   DELAY
                RTS
CURRENTTRACK:   DC.B    $FF,0
STORE:          DC.L    0

END:
; ============================================================================
```

We type/load everything under ASM-ONE, we assemble, we save the compiled binaries : **TRACKLOADER_AlphaOne_P2** and **DELTA**

If you prefer the already compiled version, attached as a Hexa suite.

TRACKLOADER_AlphaOnE_P2

```
24 79 00 00 0C 1A 4D F9 00 DF F0 00 24 11 28 02 20 29 00 04 D8 80 0C 84
00 08 44 68 6D 00 00 08 04 84 00 08 44 68 88 FC 18 9C 02 84 00 00 00 FF
33 C4 00 00 0C 20 4E B9 00 00 9B 40 22 02 82 FC 16 00 02 81 00 00 00 FF
26 01 C6 FC 16 00 94 83 49 F9 00 BF D1 00 4B F9 00 BF E0 01 47 FA 01 E4
7E 00 D0 82 26 00 87 FC 16 00 48 43 4A 43 66 08 48 43 53 03 60 00 00 04
48 43 17 43 00 01 18 BC 00 7D 4E 71 4E 71 18 BC 00 75 61 00 01 7C 61 00
01 98 0C 13 00 FF 66 20 60 00 00 0A 42 79 00 00 0C 20 4E 75 16 BC 00 00
08 D4 00 01 08 15 00 04 67 06 61 00 01 7C 60 F4 08 D4 00 02 08 01 00 00
67 04 08 94 00 02 76 00 7A 00 16 01 60 00 00 04 4E 75 1A 13 16 81 E2 4B
E2 4D 9A 43 67 16 6B 06 08 D4 00 01 60 06 08 94 00 01 44 45 61 00 01 42
53 05 66 F8 61 00 01 32 3D 7C 82 10 00 96 3D 7C 7F 00 00 9E 3D 7C 85 00
00 9E 3D 7C 44 89 00 7E 3D 7C 40 00 00 24 2D 4A 00 20 3D 7C 99 00 00 24
3D 7C 99 00 00 24 3D 7C 00 02 00 9C 4E 71 4E 71 60 00 00 10 48 E7 FF FE
61 00 FE C6 4C DF 7F FF 4E 75 08 39 00 01 00 DF F0 1F 67 F6 3D 7C 40 00
00 24 7A 00 22 4A 28 3C 55 55 55 55 0C 59 44 89 66 FA 0C 51 44 89 67 F4
26 11 22 29 00 04 C6 84 C2 84 E3 83 86 81 E0 9B B6 05 67 08 D3 FC 00 00
04 3E 60 D8 D3 FC 00 00 00 38 2C 3C 00 00 00 7F 22 29 02 00 26 19 C6 84
C2 84 E3 83 86 81 47 FA 00 A4 26 83 76 03 B0 87 67 46 B4 87 6E 02 10 D3
52 8B 52 87 51 CB FF F0 51 CE FF D6 52 05 0C 05 00 0B 66 00 FF 90 B0 87
67 26 08 14 00 02 67 0C 08 94 00 02 61 00 00 32 60 00 FF 1A 08 D4 00 02
61 00 00 26 08 94 00 01 61 00 00 46 60 00 FF 06 18 BC 00 FD 4E 71 4E 71
18 BC 00 E7 47 FA 00 44 10 2B 00 01 D1 13 4E 75 42 2C 03 00 19 7C 00 19
04 00 19 7C 00 01 0D 00 08 2C 00 00 0C 00 67 F8 08 AC 00 00 0D 00 4E 75
08 15 00 05 66 FA 4E 75 08 94 00 00 61 00 FF D2 08 D4 00 00 61 00 FF CA
4E 75 FF 00 00 00 00 00
```

Part 21 Hack of Last_Load-Disk1 part #01

As we have explained through various chapters and flowcharts of this tutorial, the last Loading performed on the original N°1 will load some code in memory and copy it to address **$256** and execute it.
The famous **'Last_Load-Disk1'** which is... tough.

For a better overall understanding we will load our ripped data at this address.
It's necessary to review the chapter *'Part 9 Analysis of the last loaded code : Last_Load'* in order to fully understand the code we are going to modify.

The goal is to preserve as much as possible the original code and to patch what is no longer necessary.

Now insert the disk containing the Last_Load file into the internal drive.
Type in :
LM Last_Load, 256 **// Loaded in memory $256 -> $9DCE**

Now insert the disk containing the file TRACKLOADER_AlphaOne_P2 and DELTA into the internal drive.
And as expected we insert our *trackloader* in **$19C4**
LM TRACKLOADER_AlphaOne_P2, 19F8
LM DELTA, 9B40

Below :

- **In blue** the original code that we keep.
- **In red** the original code that we do not keep.
- **In orange** the original code that we keep because it is deactivated through another routine.
- **In purple** our new replacement code using the **A** command of the Action Replay.
- And always in **yellow background** the commands to type in the AR.

```
256     BRA     1DE0            ; Original code that jumps to 1DE0
;================================
1DE0    MOVE.B  C0.S,D0         ;
...
1E48    JSR     19C4.S          ; GoSub → #Trackloader_2_Init
1E4C    JSR     1A96.S          ; GoSub → #Return_T00
1E50    JSR     8C30            ; GoSub → #BASE_INSERT_DISK_ASKED
...
1E48    BSR     9D3E            ; We prefer to delete the above routines, this allows us to save
                                ; some bytes and we shift the rest.
1E4C    MOVE.W  #1A0,DFF096     ;
1E54    BSR     1C5C            ; Jump in 1C5C to #Loading_Phase_#1
```

There follow some loading phases that we don't touch as expected, our DELTA routine will take care of calculating the real **Raw.Pos** before the actual *trackload*.

```
1C5C    LEA     46FB4,A0        ; Load_Menu_01
1C62    LEA     17D6.S,A1       ;
1C66    BSR     1B2E            ; Hexa Signature = 0003A91C46244304A924783188E0649088F1EB8400482075E0B6AA411301800
1C6A    BSR     1824            ;

1C6E    LEA     70000,A0        ; Load_Menu_02
1C74    LEA     17DE.S,A1       ;
1C78    BSR     1B2E            ; Hexa Signature = 00000250C1C0303200A06617C0080081880703380100 3A3ADBF0040020CF803F
1C7C    BSR     1824            ;

1C80    LEA     45F34,A0        ; Load_Menu_03
1C86    LEA     17E6.S,A1       ;
1C8A    BSR     1B2E            ; Hexa Signature = 0000043980D7046478DA0F679AE33C6E0448C02100B33BDCC79A03DDDB9EBE7D
1C8E    BSR     1824            ;

1C92    LEA     50000,A0        ; Load_Menu_04
1C98    LEA     17EE.S,A1       ;
1C9C    BSR     1B2E            ; Hexa Signature = 000001242180904B8A0A2188008C4042100048C0022183430283888A8181CD83
1CA0    BSR     1824            ;

1CA4    LEA     515DC,A0        ; Load_Menu_05
1CAA    LEA     $17F6.S,A1      ;
1CAE    BSR     1B2E            ; Hexa Signature = 00000665E080500020003808C0003E024005BD05C203FE030107FE8A81077F99
1CB2    LEA     42000,A1        ;
1CB8    BSR     1826            ;

1CBC    LEA     50398,A0        ; Load_Menu_06
1CC2    LEA     17FE.S,A1       :
1CC6    BSR     1B2E            ; Hexa Signature = 00000269AC054004C00BDEA00BC9A8A01B90782D2C023041CA05B443CC737227
1CCA    BSR     1824            ;

1CCE    LEA     457A4,A0        ; Load_Menu_07
1CD4    LEA     1806.S,A1       ;
1CD8    BSR     1B2E            ; Hexa  Signature = 000003FDE5FF07F9078B40A078967827ED28B08F40700463F81FB0904C6C0482
1CDC    JMP     1824            ; GoTo #Decomp/Decrypt then return after the BSR $1E54 performed above
```

Our next Loading if we believe the original process should concern the **'Load_Menu_08'** data
Except that in our case, these data are on **our disk2**.

We will have to use and modify the '**Shadow Of The Beast II disk change request routine'**

So, back in **$1E58** to change the code.

```
1E58    JSR     8C30                ; GoSub #BASE_INSERT_DISK_ASKED because files 8 and 9 are on our disk2
```

And we look in this **'disk change process'** if nothing is going to be a problem with our Hack.

#BASE_INSERT_DISK_ASKED

```
8C30    MOVEQ   #1,D0                       ; D0=01               // Choice of the requested disk (00=Disk1, 01=Disk2)

#BASE_INSERT_DISK_ASKED_WITHOUT_FLOPPY_DISK_MARKER
8C32    MOVE.W  D0,A0.S                     ;
8C36    BSR     9D3E                        ; GoSub ➔ #Wait_BB6
8C3A    LEA     DFF000,A6                   ;
8C40    MOVE.W  #1A0,96(A6)                 ; Nothing to
8C46    MOVE.L  #8D1E,DFF080                ; change here.
8C50    MOVE.W  #1200,100(A6)               ;
8C56    CLR.L   102(A6)                     ; We
8C5A    CLR.L   108(A6)                     ; don’t
8C5E    MOVE.L  #2C81F4C1,8E(A6)            ; touch
8C66    MOVE.L  #3800D0,92(A6)              ; anything

8C6E    LEA     30000, A0                   ; to nothing I said
8C74    MOVE.W  #7CF,D1                     ;
8C78    CLR.L   (A0)+                       ; etc
8C7A    DBF     D1,8C78                     ; etc

8C7E    TST.W   A0.S                        ;
8C82    BEQ     8C8C                        ; GoTo ➔ #CHECK_DISK
8C84    LEA     8D2A,A0                     ;
8C8A    BRA     8C92                        ; GoTo ➔ #CHECK_DISK_A0
```

We are still in the process of **'requesting a change of disc'**.

#CHECK_DISK

```
8C8C    LEA     8DEE,A0                     ;

#CHECK_DISK_A0
8C92    LEA     30E16,A1                    ; Same !
8C98    MOVEQ   #6,D1                       ; We don’t
8C9A    MOVEQ   #0D,D2                      ; touch anything
8C9C    MOVE.W  (A0)+,(A1)+                 ; here
8C9E    DBF     D2,8C9C                     ;
8CA2    ADDA.W  #C,A1                       ; etc
8CA6    DBF     D1,8C9A                     ;

8CAA    LEA     31021,A1                    ;
8CB0    LEA     8EB2,A0                     ; etc
8CB6    MOVEQ   #6,D1                       ;
8CB8    MOVEQ   #15,D2                      ; et
8CBA    MOVE.B  (A0)+,(A1)+                 ;
8CBC    DBF     D2,8CBA                     ; etc
8CC0    ADDA.W  #12,A1                      ;
8CC4    DBF     D1,8CB8                     ;

8CC8    MOVE.W  #F00,DFF182                 ; Display of the message ‘PLEASE INSER BEAST DISK’
8CD0    BSR     9D3E                        ; GoSub ➔ #Wait_BB6
8CD4    MOVE.W  #8180,DFF096                ;

#DISK2_OR_DISK1_INSERTED?
8CDC    BSR     1ACA                        ; GoSub ➔ #DF0__SIDE_DOWN__MOTOR_OFF__DIR_EXT     // useless but already patched
8CE0    TST.B   BB4.S                       ;
8CE4    BEQ     8CE0                        ;
8CE6    BSR     19C4                        ; GoSub ➔ #Trackloader_2_Init
8CEA    BSR     1A96                        ; GoSub ➔ #Return_T00                             // useless but already patched
...
8CE6    NOP                                 ; Previously GoSub ➔ #Trackloader_2_Init
8CE8    NOP                                 ;
8CEA    BSR     1A96                        ; We keep it, anyway it is disabled in our TrackLoader
...

8CEE    LEA     B0.S,A0                     ;
8CF2    LEA     8D16,A1                     ; Table Pos.      // Check_Signature DISK, 1 WORD
                                            ; :008D16 00 00 18 9C      //adr. Signature Disk that we don't change.
                                            ; Because it is managed directly in our DELTA routine (see below)

8CF8    BSR     1B2E                        ;
8CFC    TST.W   A0.S                        ;
8D00    BEQ     8D0C                        ; If bits to zero then GoTo ➔ #Check_Signature_DISK_01
```

(On the page, the lines 8CE6 BSR 19C4 are struck through.)

#CHECK_DISK (continued)

So, it is obvious that the **'disk signature'** is **not** the **same** as the **original** one because we have changed the organization of data on the disks.

Several solutions are possible to know these 2 words of each disk.

- Live by setting a *BreakPoint* in the execution of our code.
- Use a hex editor (e.g. Deksid) and go look at what is in position *$189C* on our two disks.
- Use the Replay action to read the Track in question and go to the relevant offset, etc.

BRIEF... Anyway I give you directly these two values.

Disk1 ➔ 64 A6 **Disk2 ➔ 44 CC**

#Check_Signature_DISK_02

```
~~8D02      CMPI.W    #D0D2,B0.S~~              ; Check Signature Disk2
8D02      CMPI.W    #44CC,B0.S                ; Our new signature

8D08      BNE       8CDC                      ; If Disk2 signature not found, we loop on #DISK2_OR_DISK1_INSERTED?
8D0A      RTS                                 ; Signature found, we return to the code
```

#Check_Signature_DISK_01

```
~~8D0C      CMPI.W    #4,B0.S~~                 ; Check Signature Disk1
8D0C      CMPI.W    #64A6,B0.S                ; Our new signature

8D12      BNE.B     8CDC                      ; If different, let's go for a ride
8D14      RTS                                 ; E.T Back Home
```

The process of requesting a change of disk being finished, the routine returns to **$1E6C** where we continue our Loading, namely the data Load_Menu_08 and Load_Menu_09 Present on our disk2.

We'll have to shift everything around because of our previous **JSR 8C30** in $1E58

```
1E5E      LEA       4324C,A0         ; Load_Menu_08
1E64      LEA       180E.S,A1        ;
1E68      BSR       1B2E             ; Hexa Signature = 4AB8041E67162078041E20B804262078042220B8042A700021C0041E41F80B1A
1E6C      MOVE.L    #7C000,C1A.S     ;
1E74      BSR       1D48             ; We jump to the second expected phase (we do not change anything)
```

We are now in the second phase of Loading.

LoadPhase_#1_End_Part1/2

```
1D48      MOVE.W    #775,D0
...
1D56      LEA       2A8.S,A0         ; Load_Menu_09
1D5A      LEA       17C6.S,A1        ;
1D5E      BSR       1B2E             ; Hexa Signature = 0000006037906060 40FF807848B5D9E050211706142C24000609274211C42570
1D62      MOVE.W    #0,dff180
1D6A      BSR       1824
...
1DDE      RTS                        ; Return to $1E86 because we have shifted everything.
```

Initially, if we had not shifted everything, we would have come back in $1E80 and branch to the **#Return_T00** routine
Of course we won't put it back because it has no interest in our crack and it still allows us to save some precious bytes.

```
Reminder
~~1E80      BSR       1A96~~             ; GoSub ➔ #Return_T00
```

So, here we are back in **$1E78** and we have to load the next data **'Load_Menu_10'** which are on .... The **disk1**
So we have to call the disk change routine again, but this time asking for **disk1**.

```
1E78      MOVEQ     #0,D0            ; D0=00                          // Choice of the requested disk (00=Disk1, 01=Disk2)
1E7A      JSR       8C32             ; GoSub #BASE_INSERT_DISK_ASKED because the ‘file’ 10 is on our disk1
```

Back on this set of **disk change routine**, we now need to load the 'Load_Menu_10' data

```
1E80      LEA       641DC,A0         ; Load_Menu_10
1E86      LEA       1676.S,A1        ;
1E8A      BSR       1B2E             ; Hexa Signature = 00004C19441E01FB52EEBFB9E1A93F80945B24B05FC0EB206DBC0074CF981007
1E8E      BSR       1824             ;
```

And all the rest of the loadings will now be done on **disk2**, so again, a **JSR 8C30** to perform.

```
1E92      JSR      8C30              ;
1E98      LEA      70400,A0          ; Load_Menu_11
1E9E      LEA      167E.S,A1         ;
1EA2      BSR      1B2E              ; Hexa Signature = 0000271DF050378932041AB57E0081AF0503A82E68250321D001030407C05839
1EA6      BSR      1824              ;

1EAA      BSR      1CE0              ; GoSub → #Data_already_loaded     // See a few lines below.
1EAE      NOP                        ; To get back on our feet and remove the GoSub #DF0__SIDE_DOWN__MOTOR_OFF__DIR_EXT

1EB0      LEA      DFF180,A0         ; A0=DFF180
1EB6      MOVEQ    #F,D0             ; D0=$F, counter
1EB8      CLR.L    (A0)+             ; → Clear (A0)
1EBA      DBF      D0,1EB8           ; ← D0=D0-1, as long D0 is different from -1, we loop. (so 16 times)
1EBE      BSR      5710              ; GoSub → #Decrypt_RAW
...
```

Then we are on the last part of the Loading, in **'Loading_Phase_#2_2/2'**.
This means *1CE0*

```
1CE0      TST.B    BE0.S             ; Bit Bits of $BE0, data marker already loaded
1CE4      BEQ      1D46              ; If equal to 0, then GoTo → 1D46 (which is RTS)

#Loading_Phase_#2_2/2
1CE6      LEA      EB7E,A0           ; Load_Menu_12
1CEC      LEA      1686.S,A1         ;
1CF0      BSR      1B2E              ; Hexa Signature = 000007FD55536181F6C1031007350001FA0F01054BEA7C121795BC00D821E107
1CF4      BSR      1824              ;

1CF8      LEA      F920,A0           ; Load_Menu_13
1CFE      LEA      168E.S,A1         ;
1D02      BSR      1B2E              ; Hexa Signature = 0000E9D9B686A6042EEEF60446D60E20E0EA16717274B3B734631FA61C8AA478
1D06      BSR      1824              ;

1D0A      LEA      26488,A0          ; Load_Menu_14
1D10      LEA      1696.S,A1         ;
1D14      BSR      1B2E              ; Hexa Signature = 00001408EFF1915FC81C0C34284FF8280FFF84107F800070797962B70F2F9004
1D18      BSR      1824              ;

1D1C      LEA      27BC2,A0          ; Load_Menu_15
1D22      LEA      169E.S,A1         ;
1D26      BSR      1B2E              ; Hexa Signature = 000E0D18191A150C02FEFDFE9A010200FDFAF9F7F8FD00030403020200010408
1D2A      BSR      1958              ;

1D2E      LEA      28AB2,A0          ; Load_Menu_16
1D34      LEA      16A6.S,A1         ;
1D38      BSR      1B2E              ; Hexa Signature = 000B48F6F7F7FC00030A110D17161A2120212016120E0702FEF6F2EEE8E8E6E4
1D3C      BSR      1958              ;

1D40      MOVE.B   #1,BE1.S          ;
1D46      RTS                        ; ← Return to 1EAE
```

Okay!
At this point we have patched all the loadings up to the Menu.

We are back in *1EAE* where the whole palette **is set to Zero**.
There follows a whole phase of work on the code in memory which does not interest us.
If necessary, review the code in memory **1EC2** → ... 'Analyse of the last loaded code : Last_Load and TrackLoader #3'

Part 21b Hack of the file Last_Load-Disk1 part #02

With the Main Menu of the game displayed and waiting for FIRE to be pressed to start loading Level 1.
Namely the routine: **#LOADING_LEVEL1**

We take a closer look at the code and change what needs to be changed.

#LOADING_LEVEL1

```
24CC     MOVE.L   #70400,C1A.S              ;
24D4     LEA      515DC.A0                  ; Loading Level#1_01/06
24DA     LEA      16AE.S,A1                 ;
24DE     BSR      1B2E                      ;
24E2     BSR      1824                      ;
24E6     LEA      4D800,A0                  ;
24EC     LEA      16B6.S,A1                 ; Loading Level#1_02/06
24F0     BSR      1B2E                      ;
24F4     BSR      1824                      ;
24F8     TST.B    C0.S                      ; Data marker already loaded ?     // No longer needed.
24FC     BNE      24CC                      ; GoTo ➔ #LOADING_LEVEL1
24F8     NOP                                ; Previously the TST
24FA     NOP                                ;
24FC     NOP                                ; Previously Branching to Level1 reLoading from the beginning

24FE     LEA      2AFBE,A0                  ;
2504     LEA      16BE.S,A1                 ; Loading Level#1_03/06
2508     BSR      1B2E                      ;
250C     BSR      1824                      ;

2510     LEA      AC00,A0                   ;
2516     LEA      16C6.S,A1                 ; Loading Level#1_04/06
251A     BSR      1B2E                      ;
251E     MOVE.B   #1,F3BE                   ;

2526     LEA      29654,A0                  ;
252C     LEA      16D6.S,A1                 ; Loading Level#1_05/06
2530     BSR      1B2E                      ;
2534     BSR      1824                      ;

2538     LEA      6DE1C,A0                  ;
253E     LEA      16CE.S,A1                 ; Loading Level#1_06/06
2542     BSR      1B2E                      ;
2546     BSR      1824                      ;

254A     MOVEA.L  C1A.S,A0                  ;
254E     BSR      9B24                      ; GoSub ➔ #RTZ_Trackloader          // See a few lines below
2552     BNE      2526                      ; loop reLoading Level1
2552     NOP                                ; Previously Branching to reLoading of level1 05/06
...
```

(Struck through: lines 24F8 TST.B C0.S, 24FC BNE 24CC, and 2552 BNE 2526.)

#RTZ_TRACKLOADER

```
9B24     MOVE.L   A0,1818.S                 ; The only 'thing' to keep.
                                            ; And still, it is only used in the subroutines that we will delete...
                                            ; Anyway... let's keep it.

9B28     BSR      9B40                      ; GoSub ➔ #Pre_Base_TrackLoadX      // To be deleted
9B2C     BSR      9B8A                      ; GoSub ➔ #Motor_ON                 // To be deleted
9B30     BSR      9BBA                      ; GoSub ➔ #Return_T00               // To be deleted
9B34     BSR      9C24                      ; GoSub ➔ #Pre_Base_TRK_X2          // To be deleted
9B38     BSR      9C94                      ; GoSub ➔ #Base_Conf_Interupt       // To be deleted
9B3C     BRA      9CDC                      ; GoTo ➔ #Processing_Trait_01       // To be deleted

9B28     NOP                                ;
9B2A     NOP                                ;
9B2C     NOP                                ;
9B2E     CLR.W    C20                       ; RTZ of the track counter to Zero
9B34     NOP                                ; RTZ of D0 and Set the Z flag Z=0
9B36     NOP                                ;
9B38     NOP                                ;
9B3A     NOP                                ;
9B3C     MOVEQ    #0,D0                     ;
9B3E     RTS                                ; E.T Back Home
```

(Struck through: lines 9B28 through 9B3C BSR/BRA.)

On the other hand, we must not forget that we have moved files from the Original Disk 1 to our Disk 2

```
07052    CMPI.W #$40B0, $02C0.W         ; Check if we go down to the 'CRYSTAL_CAVERNS' level
07058    BGE.W $0000724A                ; If this is the case we connect to $724A
→
0724A    MOVE.L $02CA.W, $0C1A.W
07250    BSR.W $00008C2C                ; GoTo #Set_Marker_Insert_Disk1
07254    BSR.W $00009D3E                ; Gosub #Erase_BB6
07258    MOVE.W #$01A0, $00DFF096       ; Conf DMACON, DSKEN enable, ALL DMA enable
07260    LEA $000515DC, A0              ; A0=515DC=DSKPTH
07266    BSR.W $00009B24                ; GoTo #Update_DSKPTH     // already patched
0726A    BNE.B $000072B0                ; Check flag Z of CCR, if 0 then
                                        ; we branch to the 3rd Trackload of the Crystal level...
                                        ; Not correct, to be deactivated by a NOP

0726A    NOP
0726C    MOVE.B   #1,F3BE
07274    MOVEQ    #2,D0
07276    MOVEQ    #1,D1
07278    LEA      1626.S,A1             ; For the next trackload of Crystal Level 01/11
...
```

We still have to disable the **CRC** routines seen in **'Part 14B'** of this tutorial.

```
0722C    CMP.W #$3D69, D0               ;
07230    BNE.B $00007226                ;

0722C    MOVE.W #$3D69,D0               ; We patch directly D0 with the correct value expected.
07230    NOP                            ; and disable the BNE
```

And the second.

```
07D52    CMP.W #$3D69, D0               ;
07D56    BNE.B $00007D4C                ;

07D52    MOVE.W #$3D69,D0               ; We patch directly D0 with the correct value expected.
07D56    NOP                            ; and disable the BNE
```

It is not necessary to disable the calls that we have overwritten with our *TrackLoader*, review the table in 'part 20a' of this tuto if necessary.
Namely: $4ED0 and $8CEA for the routine **#Return_T00**
Because we have already patched internally, in our trackloader, this routine 😉

However, we still have a free electron in 4ECC for a call to **#Trackloader_2_Init** which is not patched.
We must deactivate this call

```
04ECC    BSR      19C4                  ; Previously #Trackloader_2_Init

04ECC    NOP                            ; Branching
04ED0    NOP                            ; disable.
```

Earlier in our hack in this chapter, we shifted/modified the code from 1E48 to 1EAE
This concerns the following routines:
1ED0 #Base_LastLoad
1E66 #Loading_Phase_#2_1/2
1E74 #Loading_Phase_#2_1/2_bis

The question we have to ask ourselves is : Are there any calls to subroutines in the original code?
If yes, you have to patch them of course because the expected code is not in the same place anymore.
This is done with the commands :

FA 1ED0	→ **256 BRA 1DE0**	// Ok, it doesn't change, it's the beginning of the routine and it's still in the same place
FA 1E66	→ Nothing found, perfect.	
FA 1E74	→ **5086 BRA 1E74**	// This one is problematic because the code expected at this address has been moved.

#Loading_Phase_#2_1/2_bis

```
1E74     MOVE.L   #7C000,C1A.S      ; Copy $7C000 to the address $C1A, update of DSKPTH
1E7C     BSR      1D48              ; GoSub → #LoadPhase_#1_End_Part1/2       // TrackLoad of Load_Menu_09
1E80     BSR      1A96              ; GoSub → #Return_T00
```

Now this part of the code is in $1E6C
So all we have to do is replace the **BRA** address
~~5086 BRA 1E74~~
By a
5086 BRA 1E6C

We save our new patched file: **SM Last_Load-MOD, 256 9DCE**
We write everything on our crack **disk1 freshly inserted in DF0**

```
O 00, 20000 60000
RT !152 1 20000
LM 1:Last_Load_mod, 20000+200           ; PsygnosisMod    Size=$9B78$200=$9D78       $20F7E/$1600= !7,16 so !8
WT !152 !8 20000
```

Next section: testing our crack 😉

Part 22 Test of our crack

Insert our freshly made disk1 into the drive and reboot your amiga.
Do 2 tests.

The first one is to let the animations play until the end and continue the Loading until the Menu.
(by changing as requested from disk of course)

The second test is to bypass the animations by pressing Fire and also continue the Loading until the Menu.
This will validate our 1st part of the crack, everything works 😉

Once on the game menu, press FIRE again to start a game.

Let's go for some tests, already a series of trackload on the left.
Then we return on our steps and we go to Crystal cavern then, once the Crystal Level reached, (which validates the trackload), to go up the steps to return to the beginning.

OK... so far you shouldn't have any problems.

You can also test the end of a game by pressing the **F10** key during the game, which allows you to abandon the game in progress.
The end intro is loaded and launched (disk2 is requested if the last loading was on disk1)
It then returns well to the menu level after the expected floppy changes.
We restart a game... it's OK, everything works. 😉

So we just have to play the game to finish it or find other problems and patch them of course.
(which will not be the case for information)

The best way is to test the game completely until the end.
There are a few videos on the internet on the subject as well as complete 'tips' on where to go and when, etc.
An advice..., activate before the cheat mode (see section at the beginning of this tuto).