Street Fighter 2
by CAPCOM

Cracking the MFM & Copylock protection
Extending functionality to use up to 4 disk drives

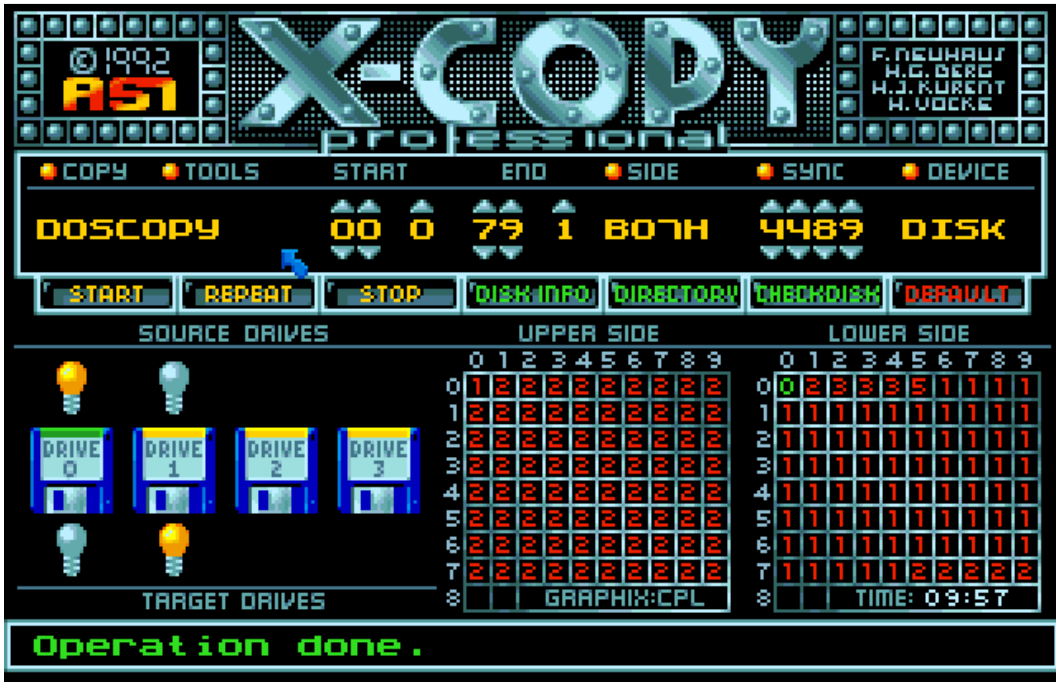
Contents

0 Introduction

We will need:

Street Fighter 2 - The World Warrior (CAPS image 1371 or original disks)
Amiga 500 with 1MB Chip RAM (for game)
Amiga 1200 with 2MB Chip RAM (for assembler)
Action Replay 3
Alpha One's Track Loader Pro
Alpha One's Track Loader Pro X (Multi Drive Support)
ASM One Assembler
Mr. Larmer's Copylock Decoder (optional)

By attempting to copy one of the disks with a disk copier tool like X-Copy brings us to the conclusion that the game is using some kind of custom MFM disk format which cannot be easily copied.



But fortunately there is a way around it to make a functional copy of the game, let's take a closer look at how to do it..

1 Analysis of boot block

The disk used to boot the game always needs to have a standard AmigaDOS track containing a boot block. The boot block occupies 2 out of 11 sectors on track 0, resulting in a size of $400 bytes.
Analysis of the boot block will shed some light onto the bootstrapping process of the game.
We would expect the boot block to contain a loader used to load data from the disk.

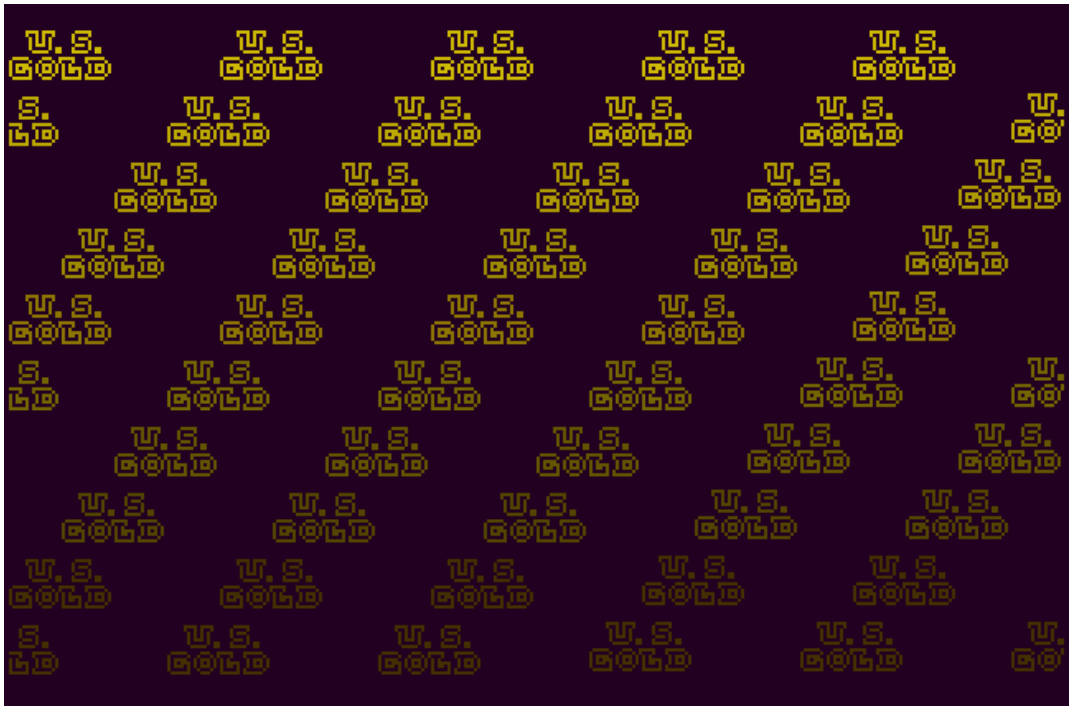
We have to keep in mind that the boot block gets loaded to some arbitrary address location in memory when the boot process starts.

We load the first track to address $10000 via AR3 for analysis:
RT 0 1 10000

```
~01000C LEA     10016(PC),A0
~010010 MOVE.L  A0,00000080.S
~010014 TRAP    #0
~010016 MOVE.W  #2700,SR
~01001A LEA     0007FFF0,A7
~010020 LEA     00DFF000,A6
~010026 MOVE.W  #7FFF,D6
~01002A MOVE.W  D6,96(A6)
~01002E MOVE.W  D6,9A(A6)
~010032 MOVE.W  D6,9C(A6)
~010036 LEA     10050(PC),A0
~01003A LEA     00075044,A1
~010040 MOVE.W  #1D5,D0
~010044 MOVE.W  (A0)+,(A1)+
~010046 DBF     D0,00010044
~01004A JMP     00075044
```

$1000C - $10014	Sets supervisor mode, continues execution at $10016
$10016 - $10032	Sets registers: $DFF096 DMACON, $DFF09A INTENA, $DFF09C INTREQ
$10036 - $10046	Copies rest of boot block to $75044 starting from relative $10050
$1004A	Jumps to $75044

Boot up the game, following loading screen is shown:



Freeze the game and start disassembling the routine at address $75044.

D 75044

```
~075044 MOVEQ   #28,D7
~075046 LEA     00004000.S,A0
~07504A MOVEA.L A0,A1
~07504C MOVE.W  #270F,D0
~075050 CLR.L   (A0)+
~075052 DBF     D0,00075050
~075056 LEA     000753AC,A0
~07505C MOVEQ   #F,D0
~07505E MOVEA.L A1,A2
~075060 MOVEQ   #9,D1
~075062 MOVEQ   #4,D2
~075064 MOVE.L  (A0),(A2)+
~075066 ADDQ.L  #4,A2
~075068 DBF     D2,00075064
~07506C ADDA.L  #2F6,A2
~075072 DBF     D1,00075062
~075076 ADDA.L  D7,A1
~075078 ADDQ.L  #4,A0
~07507A DBF     D0,0007505E
~07507E MOVE.W  #3881,8E(A6)
~075084 MOVE.W  #C1,90(A6)
~07508A MOVE.W  #38,92(A6)
~075090 MOVE.W  #D0,94(A6)
~075096 MOVE.W  #4200,100(A6)
~07509C MOVE.L  #7532E.80(A6)
~0750A4 MOVE.W  #0,88(A6)
~0750AA MOVE.W  #83D0,96(A6)
~0750B0 LEA     00018828,A0
~0750B6 MOVEQ   #2,D0
~0750B8 MOVEQ   #49,D1
~0750BA MOVE.W  #0,000753A8
~0750C2 BSR     000750D4
~0750C6 MOVE.W  #7FFF,00DFF096
~0750CE JMP     00018828
```

Different chip registers are being set up to display the loading screen:

Chip register	Value	Description
$DFF08E DIWSTART	#3881	Display window start
$DFF090 DIWSTOP	#C1	Display window stop
$DFF092 DDFSTRT	#38	Display data fetch start
$DFF094 DDFSTOP	#D0	Display data fetch stop
$DFF100 BPLCON0	#4200	Bitplane control
$DFF080 COP1LCH	#7532E	Coprocessor 1st location
$DFF088 COPJMP1	#0	Coprocessor restart at 1st location
$DFF096 DMACON	#83D0	DMA control write
$DFF096 DMACON	#7FFF	DMA control write

Addresses $750B0 - $750C2 set the needed parameters for the loader and afterwards the loader is called. When the loader subroutine returns, program execution jumps to $18828 and the game starts.

After playing the game until the end, it is apparent that the first disk is only used up to this point, where the animation and the title screen is shown. So let's start to take a closer look how we can make a working copy of the first disk.

2 Ripping first disk

We'll change the jump to $18828 to jump to itself instead, this way we can easily rip the data from memory. Load the boot block into AR3, fix the jump instruction, recalculate the checksum and write it

back to disk.

```
RT 0 1 10000
Disk ok

D 100DA
^0100DA JMP     00018828

A 100DA
^0100DA JMP 750CE
^0100E0

BOOTCHK 10000
Old checksum was 8CEA93AB, now is set to 8CE4CB05

WT 0 1 10000
Disk ok
```

Boot the disk and wait until track loading is finished and we're stuck in the loop.
Enter AR3 and check.

```
D
^0750CE JMP     000750CE

M 18828
:018828 60 00 00 F4 3F 00 2F 08 10 39 00 BF ED 01 6A 78 `...?./..9....jx
:018838 08 00 00 03 67 72 42 40 33 C0 00 01 8A B4 10 39 ....grB@3......9
:018848 00 BF EC 01 13 FC 00 57 00 BF EE 01 41 F9 00 01 .....ü.W....A...
:018858 8A DC 42 39 00 BF EC 01 E2 48 64 0E 13 C0 00 01 .üB9.....Hd.....
:018868 8A B4 11 BC 00 01 00 00 60 06 11 BC 00 00 00 00 ........`.......
:018878 08 39 00 03 00 BF ED 01 67 F6 4E 71 4E 71 4E 71 .9......göNqNqNq
:018888 4E 71 4E 71 4E 71 4E 71 4E 71 4E 71 4E 71 4E 71 NqNqNqNqNqNqNqNq
:018898 4E 71 4E 71 4E 71 4E 71 4E 71 4E 71 4E 71 4E 71 NqNqNqNqNqNqNqNq
:0188A8 4E 71 42 39 00 BF EE 01 3D 7C 00 08 00 9C 20 5F NqB9....=|.... _
:0188B8 30 1F 4E 73 08 39 00 05 00 DF F0 1F 67 4E 33 EE 0.Ns.9...ß..gN3.
:0188C8 00 16 00 01 8A B0 52 79 00 01 8A B2 3D 7C CC 01 ......Ry....=|..
:0188D8 00 34 61 00 04 44 4E B9 00 02 06 1C 4A 79 00 01 .4a..DN.....Jy..
:0188E8 8A D6 67 28 52 79 00 01 8A D6 0C 79 00 32 00 01 .Ög(Ry...Ö.y.2..
:0188F8 8A D6 66 18 33 FC 00 01 00 01 8A D6 53 79 00 01 .Öf.3ü.....ÖSy..
:018908 8A D8 64 08 33 FC 00 00 00 01 8A D8 33 FC 00 70 ..d.3ü......3ü.p
:018918 00 DF F0 9C 4E 73 21 FC 00 01 89 28 00 80 4E 40 .ß..Ns!ü...(..N@
:018928 4F F9 00 07 FF E0 61 00 00 0C 61 00 02 C8 4E F9 O.....a...a...N.
:018938 00 01 98 36 46 FC 27 00 4D F9 00 DF F0 00 13 FC ...6Fü'.M..ß...ü
:018948 00 02 00 BF E0 01 3D 7C 7F FF 00 9A 3D 7C 7F FF ......=|....=|..
:018958 00 9C 3D 7C 7F FF 00 96 21 FC 00 01 89 1C 00 60 ..=|....!ü.....`
:018968 21 FC 00 01 89 1C 00 64 21 FC 00 01 89 1C 00 70 !ü.....d!ü.....p
```

To determine the amount of data that was read from disk to memory, we scan the memory starting from $18828 until we reach the end.

NQ 18828

The end is reached at $753E8, but keep in mind that the loader occupies the memory space from $75044 to $753E8.

```
0728A8 ;.▒.▒.?.?.P▒.?.?.;.▒.?.!.?.;.▒h.▒.?.w~.?.ws.ü.;.'.▒.Ä.?.ö. .?.<.;.~.Q?.Q?
07295C .>.▒X.;.p.~?.üü?.ü. .;.~5.?ü.\.;.xGB=.ID?.?.{o.üC.;.q.ö.▒ö.Q.ö. .;.p/.pö.
072A1D t.%.pßy.Ä.~.;.@.s.ü'.q.▒.z.ü. .;.:.s.P.P.w.o.3A.|;.ß.G.<.g.?ü@ß.;.▒.'.o.u
072ACF .w._.u.'.ß.+.j._0_.3.!.@.>.z.?.6▒.ü.5?.<.Z.?.x@.B. .!.ö.8{Q.ü.ü.P.▒.▒.>P.
072BA0 ö.ßü.V.<.^.ü.~.o.ü.>.ü.p.5.0.>.=/h.▒.ü@.ß.\.ß.\.:.t.w.~.B.ü.ö.B.ü.o.▒.>.P
072C36 .▒.>.{.t.t.>ß.H.?}.>.'.P.>.'.P.ü.o./.i.▒.l▒.l.z.ß. .{.W.{.W.W.C.ü.]lß.?.ß
072D0F .?.?.ü.g. . .'.▒. .8.ü.~.~.ü.y.H.x.H.x. .o. .ßl.H.H.▒.ü.o.x.~.".?.?. ▒.▒.
072DEA ~.H.(@.(@. ▒.l.=>.j.▒.z▒.x.2.(.p.(.p.'.ß. . .ᵀ._.@.▒.p.k.@.@.+.Yh.Z.D.>.l.ü
072F0E h...h.Ö.p.{.p.@.l.ü.ü .▒k. .?.0.@.▒.@.▒./ü.ᵀ.Jß.l.in.a.8.Y@.w. .?. .?.'L.
072FF6 @.[.l.>.@.▒.@.'.P. .".2.'.0.?.0.?.'.>.<.?.p.▒.0.▒.x.▒.▒.▒.'.▒ß.x.Ä.▒.▒.
073147 ~.▒ß.x.Q.@.ß.l.▒.@.}.}.}ü.ü.ü.w.8.@. . . ._.l.ü.ß.▒s.?.?.?.?.?.X.<.ü.ü.=.;.
07320A =.;.=.'.<.'ü.0.?.q.?.O▒.l.l.p.?.▒.P†B.ᵀB.†.?.~.ß@. .B.B.P.?.<.p.P.H b.H`b
0732D6 .P.o.@P.D.D. .p./.†.x.ü.▒.0.x. .ü.ü.<. ▒.ü.y. . .?.ü.ü.ü.?.▒._.?ü.ü.~.ü.7
0735B4 .l.7.ü.?.~(A.@."H0('.B.Q.üA.S.p.$Ir.t.$.X.Q.ü.öQ.X.Q.=l8.=l.=T.8.=l.=lB.-
07509D l.S..=l.=l.A.(p.rI3ü.S.a.3ü▒.ß.N.P.$Ak2a.k.3ü.S.H§./.a,.a:k *Ha.k. MXOL.R
075105 @Q.öx.a.JDNu _L.$y.S.k.f.P.S.'.=l@.$=l.-l. =lh.=l.=lD.~=l.$=l.$p.Ä8..D.V.
07516E g.=l@.$x.Nu=l@.$x.Nu.,.g.p.a.'.By.S.NuI.*<UUUU49.S.v.VB.49.S.g.S.l.l.▒.F.
0751D9 C.4<.8.Ä.Ä.,.W.f.x.Nu.l.l.l.ß.l.l.l.Nu<.Jy.S.j.a.\.y.S.gL.y.S.0.j.D@.9.S.
075248 '.9.S.F.F.S@.B.'.'.A.6<.ÄQ.üQ.äNuE.ZD.g.UJ Jar29.S.A.y.S.f.üA.a.X..A.2<(.
0752B8 a8.f.A.0<.aHx.Nu.$,".(.g. .NuäI .UA$.Q.Nu ."NuC.äHS@".$.Ä.*.Q.Nu.@.ä._@.
075344 ~.K.'.p.p.'.P.@.0.J.0.l9.E.D.~8.R.P.((.'.P.
Quick-dump up to address:09CFB6
Ready.
```

The actual loaded data ends at address $735E6.
We will dump the loaded data to a newly formatted disk, we'll call from now on: **SF2_DATA_D1**.

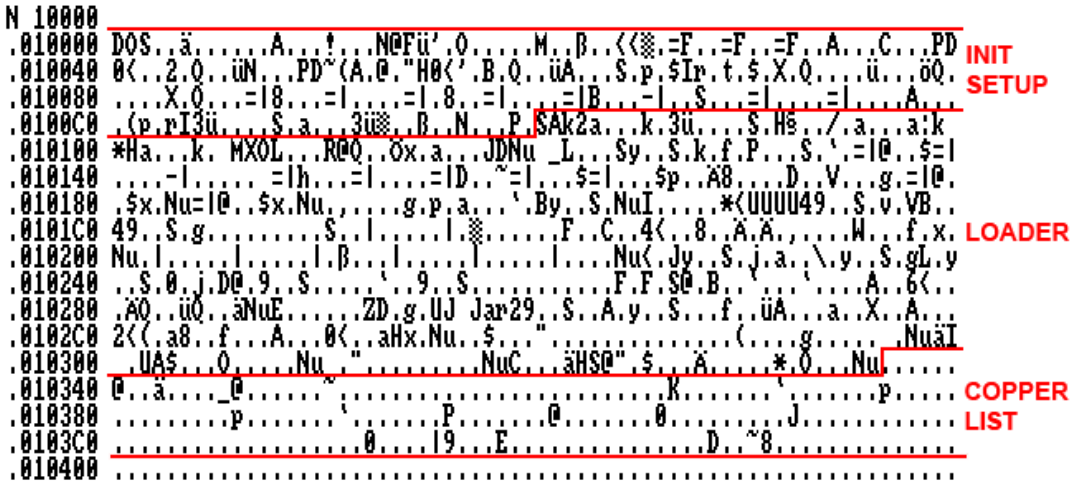
```
SM DISK1DATA,18828 735E6
Disk ok
```

We restore the jump modification in the boot block, recalculate the checksum and then write it back to the original disk.

A 100DA -> JMP 18828
BOOTCHK 10000
WT 0 1 10000

3 Replace loader in boot block

The boot block is divided in 3 major parts:



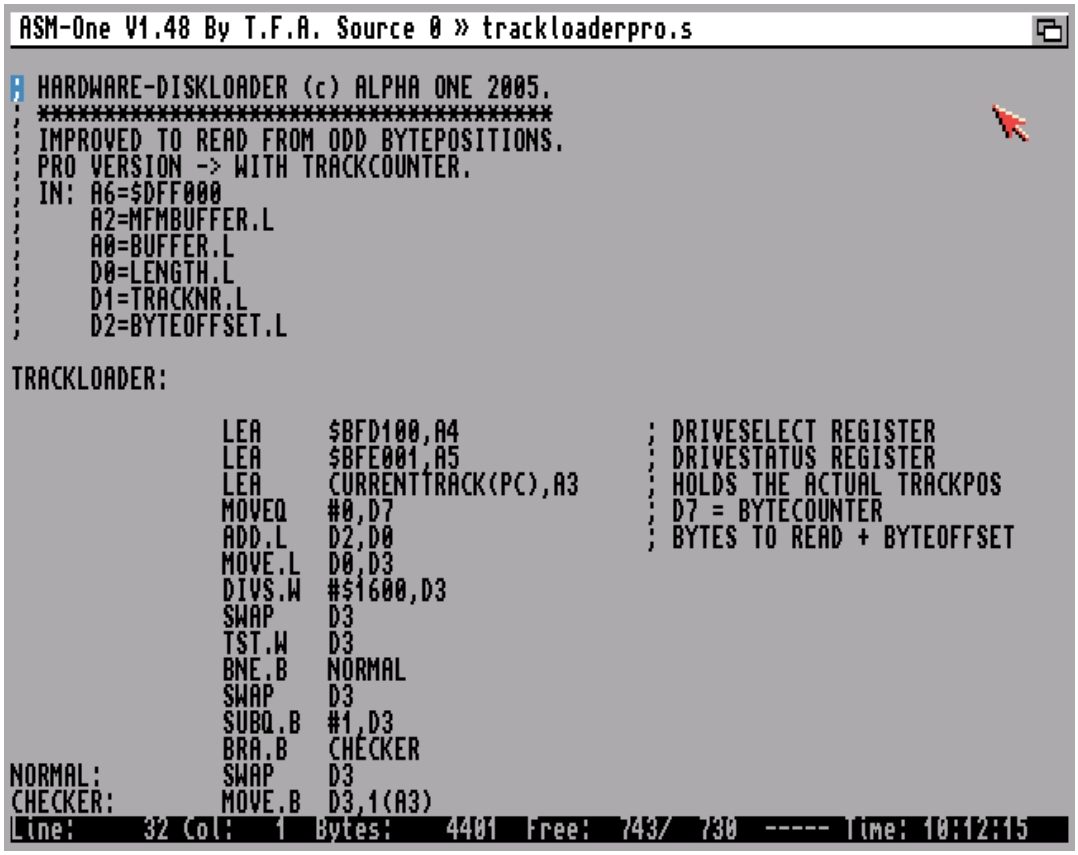
In order to be able to load the previously ripped data, the current loader has to be replaced by a standard track loader with a track size of $1600.

The current loader is located between $100E0 and $1033A. (length 25A)
We'll clear the memory occupied by the current loader:

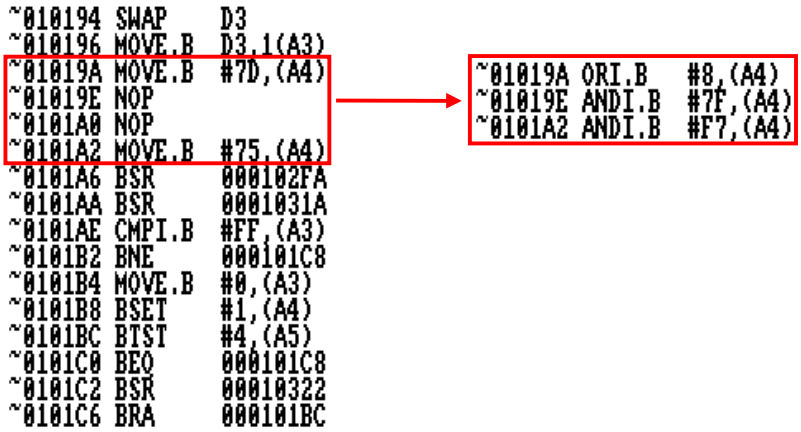
```
RT 0 1 10000
Disk ok

O 0,100E0 1033A
Ready.

BOOTCHK 10000
Old checksum was 8CE4CB05, now is set to C67BEE44

WT 0 1 10000
Disk ok
```

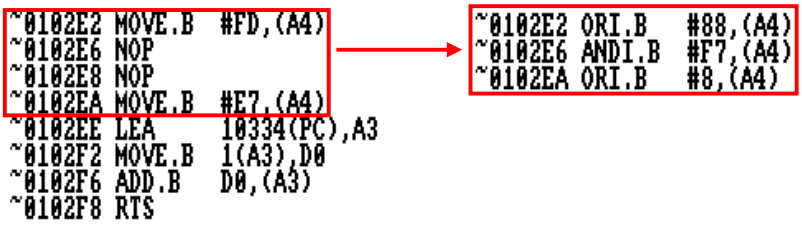
Assemble Alpha One's Track Loader Pro in ASM One and load the binary to the address $1016E. The size of the binary is $1CC bytes.

$1033A - $1CC = $1016E (Start address of new loader)



We need to apply 2 small fixes in the loader where the DF0 disk motor is switched on respectively switched off. Either do it in the assembler before assembling the binary or patch it afterwards in AR3:





A good explanation of how to program Amiga disk drives can be found in Codetapper's "How to HD install Pacland (MFM Format) using WHDLoad" tutorial.
http://zakalwe.fi/~shd/amiga-cracking/mfminstalling.txt

Insert the new loader and write the modified boot block to **SF2_DATA_D1**:

```
RT 0 1 10000
Disk ok

LM ALPHA1_TL,1016E
Loading from 01016E to 01033A
Disk ok

BOOTCHK 10000
Old checksum was C67BEE44, now is set to 25070A63


SM DISK1BOOT,10000 11600
Disk ok
```

Please note: The checksums may differ from yours, because some steps were applied in a different order than illustrated here.

4 Reconstruct first disk

Now we will take the modified boot block and the ripped data and create a new disk image.
Label a new floppy as **SF2_CRACKED_D1.**

Boot Amiga and freeze Action Replay.

Insert the **SF2_DATA_D1** and load **DISK1BOOT** and **DISK1DATA** into memory.

```
LM DISK1BOOT,10000
Loading from 010000 to 011600
Disk ok

LM DISK1DATA,11600
Loading from 011600 to 06C3BE
Disk ok
```

Now we need to write some code, where we pass the necessary parameters to the newly added loader:

A6=$DFF000
A2=MFMBUFFER.L $C8B8 (Like in original loader)
A0=BUFFER.L $18828
D0=LENGTH.L $5ADBE
D1=TRACKNR.L $1
D2=BYTEOFFSET.L $0

```
~0100C2 MOVEQ   #2,D0
~0100C4 MOVEQ   #49,D1
~0100C6 MOVE.W  #0,000753A8
~0100CE BSR     000100E0
~0100D2 MOVE.W  #7FFF,00DFF096
~0100DA JMP     000750CE
;=====================================
~0100E0 MOVEM.L D0-D7/A0-A6,-(A7)
~0100E4 LEA     00DFF000,A6
~0100EA LEA     00018828,A0
~0100F0 LEA     0000C8B8,A2
~0100F6 MOVE.L  #5ADBE,D0
~0100FC MOVE.L  #1,D1
~010102 MOVE.L  #0,D2
~010108 BSR     0001016E
~01010A MOVEM.L (A7)+,D0-D7/A0-A6
~01010E RTS
;=====================================
~010110 ORI.B   #0,D0
~010114 ORI.B   #0,D0
~010118 ORI.B   #0,D0
~01011C ORI.B   #0,D0
~010120 ORI.B   #0,D0
~010124 ORI.B   #0,D0
~010128 ORI.B   #0,D0
```

Now make sure to restore the jump at $100DA to jump to address $18828, fix the boot block and write it back to disk **SF2_CRACKED_D1**:

```
A 100DA
^0100DA JMP 18828
^0100E0

BOOTCHK 10000
Old checksum was 25070A63, now is set to 8BD2663B

WT 0 !159 10000
Disk ok
```

Boot up **SF2_CRACKED_D1** it loads the data as expected and the game starts!
The original first disk from Street Fighter 2 is no more needed!

5 Ripping data from original data disks

Once the game is started select a fighter. During the versus screen you will notice that the track counter jumps to track 0 and then you will be prompted to change the disk.
This could be a Copylock based disk protection, let's keep that in mind.

To further nail down the location of the track loader we insert the requested disk, press fire and then freeze the cartridge when track loading starts.

We will probably end up somewhere between the addresses $192E6 and $1934A:

```
~0192E6 MOVEA.L (A7)+,A0
~0192E8 MOVEM.W (A7)+,D0-D1
~0192EC SUBQ.W  #1,00040606
~0192F2 BMI     000192DE
~0192F4 BNE     000192B8
~0192F6 ST      0004060A
~0192FC BRA     000192B8
;=====================================
~0192FE MOVE.W  #4000,24(A6)
~019304 MOVE.W  #2,9C(A6)
~01930A MOVE.L  #7B000,20(A6)
~019312 MOVE.W  #6800,9E(A6)
~019318 MOVE.W  #9500,9E(A6)
~01931E MOVE.W  #4489,7E(A6)
~019324 MOVE.W  #9414,24(A6)
~01932A MOVE.W  #9414,24(A6)
~019330 MOVEQ   #FFFFFFFF,D0
~019332 MULU.W  D4,D4
~019334 MOVE.W  1E(A6),D4
~019338 ANDI.W  #2,D4
~01933C DBNE    D0,00019332
~019340 BEQ     0001934C
~019342 MOVE.W  #4000,24(A6)
~019348 MOVEQ   #0,D4
~01934A RTS
```

When we execute the RTS at $1934A we will end up at either $192C4 or $192CC (which is in middle of the track loading routine)

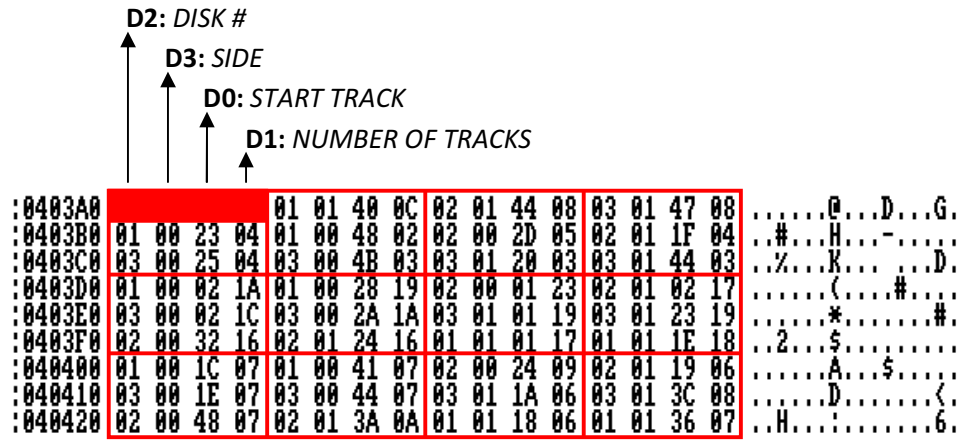
```
~0192A6 SUBQ.W  #1,D1
~0192A8 BMI     000192DC
~0192AA BSR     0001936E
~0192AE BMI     000192DE
~0192B0 MOVE.W  #2,00040606
~0192B8 MOVEM.W D0-D1,-(A7)
~0192BC MOVE.L  A0,-(A7)
~0192BE BSR     000193EE
~0192C2 BSR     000192FE
~0192C4 BMI     000192E6
~0192C6 MOVEA.L A0,A5
~0192C8 BSR     00019452
~0192CC BMI     000192E6
~0192CE MOVEA.L A5,A0
~0192D0 ADDQ.W  #4,A7
~0192D2 MOVEM.W (A7)+,D0-D1
~0192D6 ADDQ.W  #1,D0
~0192D8 DBF     D1,000192B0
~0192DC MOVEQ   #0,D4
~0192DE BSR     000193C8
~0192E2 TST.W   D4
~0192E4 RTS
```

When we investigate the code a bit prior the track loading routine, we will come across 2 interesting places

```
~019102 LEA     00040400,A1
~019108 ADD.W   D7,D7
~01910A ADD.W   D7,D7
~01910C MOVEQ   #0,D0
~01910E MOVEQ   #0,D1
~019110 MOVEQ   #0,D2
~019112 MOVEQ   #0,D3
~019114 MOVE.B  0(A1,D7.W),D2
~019118 CMP.B   0001920B,D2
~01911E BNE     00019142
~019120 MOVE.B  1(A1,D7.W),D3
~019124 MOVE.B  2(A1,D7.W),D0
~019128 MOVE.B  3(A1,D7.W),D1
  ...

~01919E LEA     000403D0,A1
~0191A4 ADD.W   D7,D7
~0191A6 ADD.W   D7,D7
~0191A8 MOVEQ   #0,D0
~0191AA MOVEQ   #0,D1
~0191AC MOVEQ   #0,D2
~0191AE MOVEQ   #0,D3
~0191B0 MOVE.B  0(A1,D7.W),D2
~0191B4 MOVE.B  1(A1,D7.W),D3
~0191B8 MOVE.B  2(A1,D7.W),D0
~0191BC MOVE.B  3(A1,D7.W),D1
  ...
```

Data from the addresses $40400 and $403D0 are read here and stored in the registers D0-D3.
Register D7 serves as pointer and is multiplied by 4 every pass (D7+D7+D7+D7).
D0-D3 are initially cleared and then populated with the data.

These 2 addresses point to a table of information needed for loading data from the disks:



DISK#	Data disk 1-3
SIDE	Lower side: 0 | Upper side: 1
START TRACK	Start track of track loading
NUMBER OF TRACKS	Number of tracks to load

Now that we understand the structure of the data table and know approximately where the disk loading is happening, we can start with the data ripping from the data disks.

Within the highlighted area the data is loaded from the data disk on a track per track basis.
Register D1 contains the current track to load and is decremented after each loop iteration.

```
~0192A6 SUBQ.W  #1,D1
~0192A8 BMI     000192DC
~0192AA BSR     0001936E
~0192AE BMI     000192DE
~0192B0 MOVE.W  #2,00040606
~0192B8 MOVEM.W D0-D1,-(A7)
~0192BC MOVE.L  A0,-(A7)
~0192BE BSR     000193EE
~0192C2 BSR     000192FE
~0192C4 BMI     000192E6
~0192C6 MOVEA.L A0,A5
~0192C8 BSR     00019452
~0192CC BMI     000192E6
~0192CE MOVEA.L A5,A0
~0192D0 ADDQ.W  #4,A7
~0192D2 MOVEM.W (A7)+,D0-D1
~0192D6 ADDQ.W  #1,D0
~0192D8 DBF     D1,000192B0
~0192DC MOVEQ   #0,D4
~0192DE BSR     000193C8
~0192E2 TST.W   D4
~0192E4 RTS
```

Set a breakpoint at $192A6 and check the register contents.

```
R
D0=00000002 0000001A 00000001 00000000  00000000 00000000 00000081 00000078
A0=00088000 00DFF1C0 00041A5C 0004111C  00047EE2 0004055C 00DFF000 0007FFD0
PC = 000192A6 USP = 000018B2 SR = 2100 T=0 S=1 I=001 X=0 N=0 Z=0 V=0 C=0
```

As we see the registers from D0-D3 contain all information needed for ripping the data:

Register D0: START TRACK
Register D1: NUMBER OF TRACKS
Register D2: DISK #
Register D3: SIDE

Register A0 contains the address $88000 ($208000 if Fast RAM is available) where the data will be loaded to.

Set another breakpoint at $192DC (end of loop) and check the register contents again.

```
R
D0=0000001C 0000FFFF 00000000 0000FFFF  00000000 55555555 00000001 54545554
A0=000A8800 0007D812 0007B002 0004111C  00BFD000 000A8800 00DFF000 0007FFD0
PC = 000192DC USP = 000018B2 SR = 2100 T=0 S=1 I=001 X=0 N=0 Z=0 V=0 C=0
```

Register D0 contains D0+D1.
Register D1 will be negative.
Register A0 points to: $A8800 + (amount of tracks to load * track size of $1400)
Register A2 points to the MFM buffer starting at $7B000.

Basically all we need to do now is to set these two breakpoints and set the information about the data we want to load into registers D0-D3 and dump it to disk at breakpoint $192DC.

If we peek into address $88000 we will see that a RNC ProPack file was loaded from the disk:

```
M 88000
:088000 52 4E 43 02 00 03 02 32 00 01 FF 8B 05 F0 48 C0 RNC....2......H.
:088010 04 11 03 00 00 1D 9F 8E 03 AE 38 03 8A 03 98 C7 ..........8.....
:088020 03 1E 09 1C 03 0E 0B 71 BC 03 C3 C7 03 CA 1C 03 .......q........
:088030 D8 03 71 E5 03 F2 BC 03 08 00 1D 71 FB 2F 02 C7 ..q........q./..
:088040 03 23 1C 03 3D 03 71 48 03 53 C7 03 5E 1C 03 83 .#..=.qH.S..^...
:088050 03 49 A3 07 03 00 1E 6C 8E 03 13 38 03 72 03 18 .I.....l...8.r..
:088060 E3 03 75 8E 03 8B 39 03 AC 03 C3 83 1E B0 8E 03 ..u...9.........
:088070 B2 38 03 B4 03 B6 E3 03 B8 8E 03 BA 70 03 8B 1E .8..........p...
...
```

Prepare 3 formatted AmigaDOS disks:
SF2_DATA_D2
SF2_DATA_D3
SF2_DATA_D4

Now rip every single entry from the data table by accordingly setting the registers and following the procedure as described above.

Save the files using a filename notation, like for example: DISKNO-SIDE-STARTTRACK-NUMBEROFTRACKS

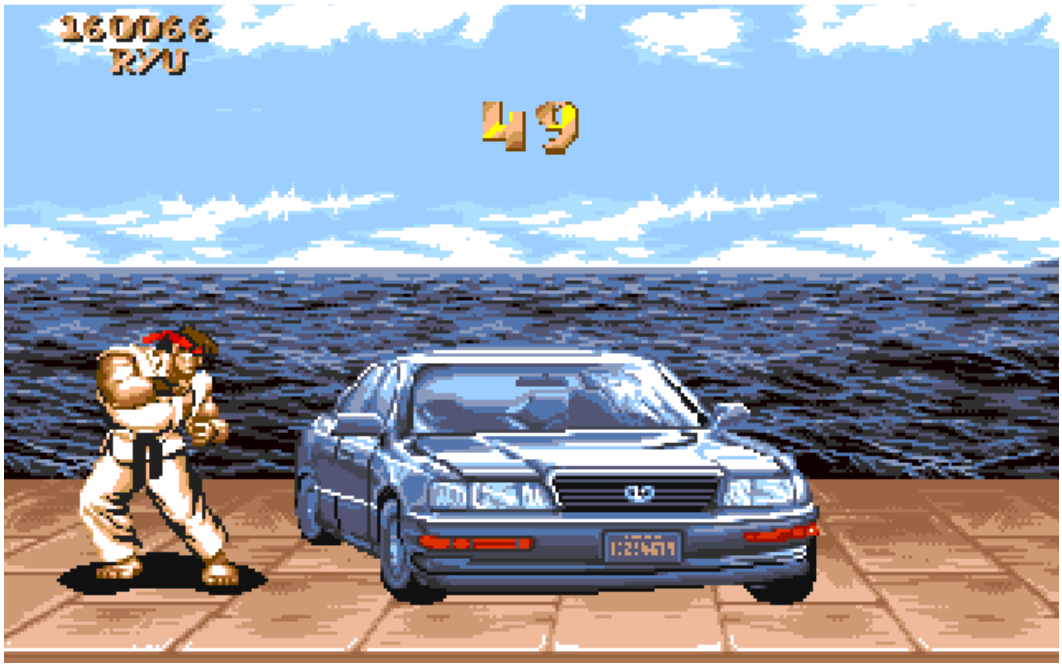
After dumping and saving a single RNC file, I restarted the game and repeated the whole process. This can be quite time consuming, so feel free to automate the process.

The 24 files from the middle and lower section of the data table are fighter and stage files.

After we ripped and saved the data we play test the game with the original disks, shortly before we arrive at the special stage (car demolition) after 4 won fights, we notice that registers D0-D3 contain entries from the upper section of the data table.

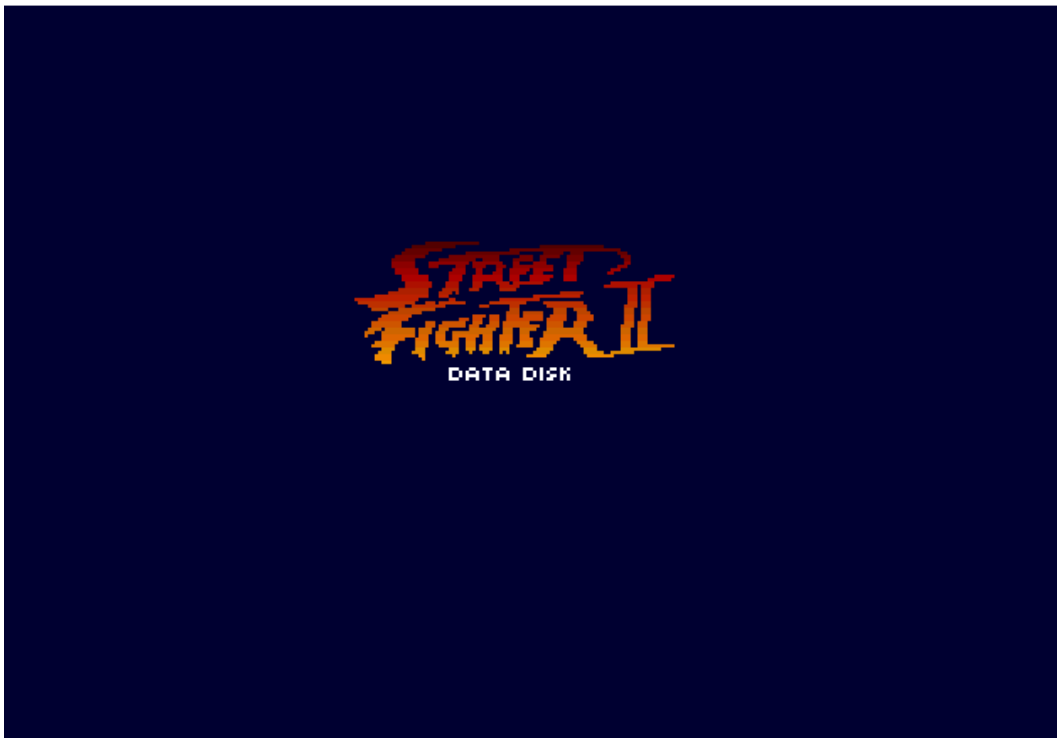
When the breakpoint $192A6 triggers pay attention to the registers D0-D3:

```
Breakpoint raised at address: 000192A6

R
D0=00000040 0000000C 00000001 00000001  00000000 00000000 0000FFFF 0000FFFF
A0=0005F8DC 00DFF1C0 00041E2C 00018A50  000476C2 00042DF2 00DFF000 0007FFD0
PC = 000192A6 USP = 000018B2 SR = 2100 T=0 S=1 I=001 X=0 N=0 Z=0 V=0 C=0
```



The game consists of 3 special stages, so for each one there is a separate file and the remaining 8 files are the ending screens of each of the selectable fighters. Rip the remaining 11 files.

Each of the data disks contain a boot block:



We rip the boot block of one of the data disks and save it to **SF2_DATA_D2** (all 3 data disks have the same boot block)

```
RT 0 1 10000
Disk ok

SM DATADISKBOOT,10000 11600
Disk ok
```

6 Data disk image reconstruction

Now as we have ripped all the files from the original disks, we will start to reconstruct the 3 data disks to exactly match the original's structure.

As we know now the original track size is $1400 bytes. A standard AmigaDOS loader uses $1600 bytes per track, so will need to add some arithmetic later to calculate the correct track numbers and offsets.

Get some newly formatted standard disks and label them:
SF2_CRACK_D2
SF2_CRACK_D3
SF2_CRACK_D4

Now we restart the Amiga and immediately hit the freeze button.
We will reconstruct the disk images in memory and then write them back to new standard disks.

Let's start by loading the ripped boot block and the files to their correct calculated positions:

Disk 2

File name:	Base address:	Side offset:	Offset:	Actual position:
DATADISKBOOT	$10000	$0	$0	**$10000**
01-00-23-04	$10000	$0	$2BC00	**$3BC00**
01-00-48-02	$10000	$0	$5A000	**$6A000**
01-01-40-0C	$10000	$64000	$50000	**$C4000**
01-00-02-1A	$10000	$0	$2800	**$12800**
01-00-1C-07	$10000	$0	$23000	**$33000**
01-00-28-19	$10000	$0	$32000	**$42000**
01-00-41-07	$10000	$0	$51400	**$61400**
01-01-01-17	$10000	$64000	$1400	**$75400**
01-01-18-06	$10000	$64000	$1E000	**$92000**
01-01-1E-18	$10000	$64000	$25800	**$99800**
01-01-36-07	$10000	$64000	$43800	**$B7800**

Disk 3

File name:	Base address:	Side offset:	Offset:	Actual position:
DATADISKBOOT	$10000	$0	$0	**$10000**
02-00-2D-05	$10000	$0	$38400	**$48400**
02-01-1F-04	$10000	$64000	$26C00	**$9AC00**
02-01-44-08	$10000	$64000	$55000	**$C9000**
02-00-01-23	$10000	$0	$1400	**$11400**
02-00-24-09	$10000	$0	$2D000	**$3D000**
02-00-32-16	$10000	$0	$3E800	**$4E800**
02-00-48-07	$10000	$0	$5A000	**$6A000**
02-01-02-17	$10000	$64000	$2800	**$76800**
02-01-19-06	$10000	$64000	$1F400	**$93400**
02-01-24-16	$10000	$64000	$2D000	**$A1000**
02-01-3A-0A	$10000	$64000	$48800	**$BC800**

Disk 4

File name:	Base address:	Side offset:	Offset:	Actual position:
DATADISKBOOT	$10000	$0	$0	**$10000**
03-00-25-04	$10000	$0	$2E400	**$3E400**
03-00-4B-03	$10000	$0	$5DC00	**$6DC00**
03-01-20-03	$10000	$64000	$28000	**$9C000**
03-01-44-03	$10000	$64000	$55000	**$C9000**
03-01-47-08	$10000	$64000	$58C00	**$CCC00**
03-00-02-1C	$10000	$0	$2800	**$12800**
03-00-1E-07	$10000	$0	$25800	**$35800**
03-00-2A-1A	$10000	$0	$34800	**$44800**
03-00-44-07	$10000	$0	$55000	**$65000**
03-01-01-19	$10000	$64000	$1400	**$75400**
03-01-1A-06	$10000	$64000	$20800	**$94800**
03-01-23-19	$10000	$64000	$2BC00	**$9FC00**
03-01-3C-08	$10000	$64000	$4B000	**$BF000**

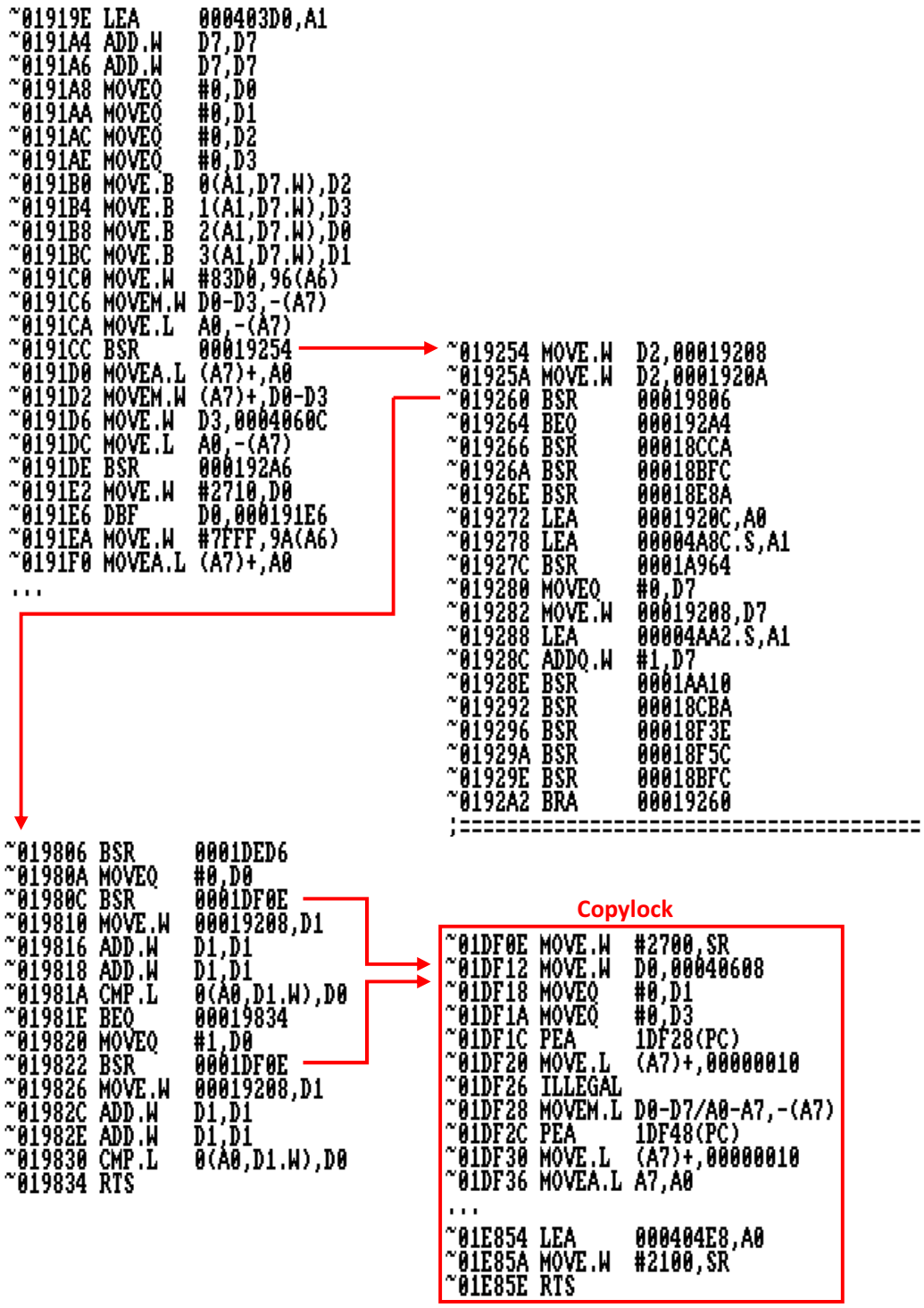
Base address: $10000
Side offset: Side x $50 tracks x $1400
Offset: Start track x $1400
Actual position: Base address + Side offset + offset

Finally we will write back each memory dump back to its disk:
WT 0 !159 10000

We're almost done with the data disks..

7 Copylock analysis

After the loader parameters have been copied to the data registers, a Copylock routine is executed. This explains why the track counter jumps to 0 when a disk swap is requested.



There are 2 jumps to the Copylock routine coming from $1980C and $19822.
Notice that register D0 is set to $0 before the first call and before the second call it is set to $1.
When we follow the calls to the Copylock routine, we'll see that the content of register D0 is stored to $40608 as a word. Shortly after the calls return some new content of register D0 is compared against memory address $404E8 with some calculated offset and then returning from the subroutine when equal otherwise the second call to the Copylock is taken.

Let's figure out what the Copylock does.
Set 2 breakpoints after the Copylock was executed at addresses $19810 and $19826.
Run the game, select Ryu as fighter and wait until one of the breakpoints fires.

```
BS 19810
Breakpoint inserted
Ready.

BS 19826
Breakpoint inserted
Ready.

X
No known virus in memory!
Ready.
Breakpoint raised at address: 00019810

R
D0=992D049E 00000000 00000001 00000000  00000000 00000000 0000FFFF 00000000
A0=000404E8 000403D0 00041A5C 00018A50  00047942 0004055C 00DFF000 0007FFC4
PC = 00019810 USP = 000018B2 SR = 2100 T=0 S=1 I=001 X=0 N=0 Z=0 V=0 C=0
```

The breakpoint at $19810 fires.
The content of D0 is $992D049E, trace 3 more instructions until the comparison.
Register D0 is compared to the value being stored at $404E8+4 which equals to $DD739D27.
Exit AR3, second breakpoint will fire at $19826, this time register D0 contains a value of $0.
The second comparison will fail as well.

In case you own a second disk drive, you may have noticed that its drive motor was briefly switched on and switched off. Now we can assume that D0 contains the requested drive number prior calling the Copylock routine: $0 for DF0 and $1 for DF1. Shortly before executing the Copylock routine, the disk drive to be used will be stored at $40608 (as word).

Exit AR3 again and you'll see that disk 2 was requested.

So let's see what happens if we insert the requested disk and press any key:

```
Breakpoint raised at address: 00019810

R
D0=DD739D27 00000000 0000000F 00000000  00000000 00000000 000000C3 000000B3
A0=000404E8 00DFF1C0 00041A5C 0004111C  000479E2 0004055C 00DFF000 0007FFC4
PC = 00019810 USP = 000018B2 SR = 2100 T=0 S=1 I=001 X=0 N=0 Z=0 V=0 C=0
```

This time the D0 register's content matches what it will be compared against.
Hmm interesting.. It looks like that the Copylock key returned by the Copylock routine is used for identifying whether the requested disk was inserted.

To verify this theory, we exit AR3 again, wait until the track loading finishes and the next disk is being requested.

Now the game asks for disk 4. Insert it and press any key.
As expected the breakpoint at $19810 triggers:

```
R
D0=B1BDE8CB 00000000 0000000F 00000000  000B0000 00080000 000000B2 000000E6
A0=000404E8 00DFF1C0 00041A5C 0004116C  00088744 000A8800 00DFF000 0007FFC4
PC = 00019810 USP = 000018B2 SR = 2100 T=0 S=1 I=001 X=0 N=0 Z=0 V=0 C=0
```

Notice the value of $B1BDE8CB in register D0. Take a closer look at the next 4 instructions.

```
D 19810
~019810 MOVE.W  00019208,D1
~019816 ADD.W   D1,D1
~019818 ADD.W   D1,D1
~01981A CMP.L   0(A0,D1.W),D0
~01981E BEQ     00019834

M 19208
:019208 00 03 00 03 50 4C 45 41 53 45 20 49 4E 53 45 52 ....PLEASE INSER
:019218 54 20 44 49 53 4B 20 4E 52 2E 00 00 20 20 20 20 T DISK NR...
:019228 49 4E 54 4F 20 54 48 45 20 44 52 49 56 45 00 00 INTO THE DRIVE..
:019238 00 00 54 48 45 4E 20 50 52 45 53 53 20 41 4E 59 ..THEN PRESS ANY
:019248 20 4B 45 59 FF 50 41 55 53 45 FF 00 33 C2 00 01  KEY.PAUSE..3...
```

Memory address $19208 holds the requested data disk (0-3), it is then multiplied by 4 and used to point at the Copylock key belonging to the requested disk.

The Copylock key table at $404E8 (register A0) looks like this:

```
:0404E8 3B E4 6B 16 DD 73 9D 27 66 15 2A BE B1 BD E8 CB ;äk..s.'f.*.....
```

Disk keys:
Disk 2: $DD739D27
Disk 3: $66152ABE
Disk 4: $B1BDE8CB

Our cracked data disks will of course not return the correct keys when the Copylock routine is run.
In order to retain most of the game's disk key recognition mechanism, we'll store the correct key directly on the disk. Later we will patch the Copylock routine to read the hardcoded Copylock key from the data disks and thereby emulating the original Copylock behavior.

8 Replace main loader

As we know from our previous data ripping step, the main loader loop is located between $192A6 and $192E4:

```
~0192A6 SUBQ.W  #1,D1
~0192A8 BMI     000192DC
~0192AA BSR     0001936E
~0192AE BMI     000192DE
~0192B0 MOVE.W  #2,00040606
~0192B8 MOVEM.W D0-D1,-(A7)
~0192BC MOVE.L  A0,-(A7)
~0192BE BSR     000193EE
~0192C2 BSR     000192FE
~0192C4 BMI     000192E6
~0192C6 MOVEA.L A0,A5
~0192C8 BSR     00019452
~0192CC BMI     000192E6
~0192CE MOVEA.L A5,A0
~0192D0 ADDQ.W  #4,A7
~0192D2 MOVEM.W (A7)+,D0-D1
~0192D6 ADDQ.W  #1,D0
~0192D8 DBF     D1,000192B0
~0192DC MOVEQ   #0,D4
~0192DE BSR     000193C8
~0192E2 TST.W   D4
~0192E4 RTS
```

We will replace the original main loader with Alpha One's Track Loader Pro X.

The new loader will be used for 2 purposes:
- Read game data from our 3 ripped data disks
- Read the hardcoded Copylock key from our 3 cracked data disks

In case you wonder why we don't reuse the first loader from the boot block, it is because its memory range will be cleared after game has been loaded (although it could be resolved by relocation). But more importantly we want to retain the multiple drive support of the game and the multiple drive capable loader is larger than the available space in the boot block.

Let's find out the size of the original loader, so we can make sure we have enough space to embed our new loader there.

If you trace each call within the main loader loop, you'll notice that the highest address being used is $194FE. Interestingly the code between $19500 and $19804 also seems to be loader related, although it is not being used anywhere (verified by clearing the mentioned memory range and play test the game until the end). As we figured out earlier at address $19806 the Copylock routine for disk key retrieval begins. That means we have the memory range from $192A6 to $19806 for our disposal (1376 bytes).

We start by resetting the Amiga, read the second track (first track contains boot block) from **SF2_CRACKED_D1** and clear the mentioned memory range:

```
RT 1 1 18828
Disk ok

0 0,192A6 19806
Ready.
```

Since we now have to deal with $1600 sized tracks instead of $1400 tracks, there is some interface code we need to write in order to correctly calculate the start track, length and the offset of the individual track loading operation.

The formula for the calculation goes like this:

$$\frac{(\text{START TRACK} * \$1400) + (\text{SIDE} * \$1400 * \$50)}{\$1600} = \text{TRACKNR (Remainder: BYTEOFFSET)}$$

And the parameters have to be transformed / set like this:

Original loader	Description		New loader	Description
D0	START TRACK	→	D1	TRACKNR.L
D1	NUMBER OF TRACKS	→	D0	NUMBER OF TRACKS * $1400 = LENGTH.L
D2	DISK #	→	D5	DISKDRIVE.B ($40609)
D3	SIDE	→	$4060C	ADDRESS CONTAINING SIDE (WORD)

D2	BYTEOFFSET.L (CALCULATED)
A0	BUFFER.L (ALREADY SET BY GAME)
A2	MFMBUFFER.L ($7B000)
A6	$DFF000

With the above mentioned information we write the following interface code beginning at address $192A6:

```
~0192A6 MOVEM.L D0-D7/A0-A6,-(A7)
~0192AA CLR.L   D3
~0192AC LEA     00DFF000,A6
~0192B2 LEA     0007B000,A2
~0192B8 MULU.W  #1400,D0
~0192BC MOVE.W  0004060C,D3
~0192C2 MULU.W  #1400,D3
~0192C6 MULU.W  #50,D3
~0192CA ADD.L   D0,D3
~0192CC DIVU.W  #1600,D3
~0192D0 MULU.W  #1400,D1
~0192D4 MOVE.L  D1,D0
~0192D6 CLR.L   D1
~0192D8 CLR.L   D2
~0192DA MOVE.W  D3,D1
~0192DC SWAP    D3
~0192DE MOVE.W  D3,D2
~0192E0 MOVE.B  00040609,D5
~0192E6 BSR     000192EE
~0192E8 MOVEM.L (A7)+,D0-D7/A0-A6
~0192EC RTS
```

Then take Alpha One's Track Loader Pro X and copy it to the address range $192EE to $19540:

```
~0192EE LEA     00BFD100,A4
~0192F4 LEA     00BFE001,A5
~0192FA ANDI.L  #FF,D5
~019300 LEA     1953E(PC),A1
~019304 MOVE.B  D5,(A1)
~019306 MOVE.B  D5,D3
~019308 ADDI.B  #3,D3
~01930C LEA     1953F(PC),A1
~019310 MOVE.B  D3,(A1)
~019312 BSET    D3,(A4)
~019314 ANDI.B  #7F,(A4)
~019318 BCLR    D3,(A4)
~01931A BSR     000194F2
~01931E BTST    #5,(A5)
~019322 BEQ     0001932C
~019326 BSR     000194AA
~01932A RTS
'=====================================
~01932C LEA     19532(PC),A3
~019330 ADD.B   D5,D5
~019332 ADDA.W  D5,A3
~019334 MOVEQ   #0,D7
~019336 ADD.L   D2,D0
~019338 MOVE.L  D0,D3
'''
```

Finally we write the changes back to our disk:

```
WT 1 1 18828
Disk ok
```

9 Copylock patch

Now as we understand how the Copylock works, we can start to patch it.
We figured out that the Copylock only serves as a means of identifying whether the correct disk was inserted.

(I decoded and checked the content of the Copylock with Mr. Larmer's Copylock Decoder and couldn't see anything else important being done inside the Copylock)

Next step is to write the Copylock keys to our 3 cracked data disks and then we can consider them as finished! Ideally we write the key directly after the boot block at $400.

Disk 2:

```
RT 0 1 10000
Disk ok

M 10400
:010400 DD 73 9D 27 00 00 00 00 00 00 00 00 00 00 00 00 ................
:010410 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

WT 0 1 10000
Disk ok
```

Disk 3:

```
RT 0 1 10000
Disk ok

M 10400
:010400 66 15 2A BE 00 00 00 00 00 00 00 00 00 00 00 00 ................
:010410 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

WT 0 1 10000
Disk ok
```

Disk 4:

```
RT 0 1 10000
Disk ok

M 10400
:010400 B1 BD E8 CB 00 00 00 00 00 00 00 00 00 00 00 00 ................
:010410 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

WT 0 1 10000
Disk ok
```

Now we need to add some code where we tell the loader where to load the Copylock key from.
We start by loading the tracks 1-5 from the first disk into memory and then clear the whole Copylock routine.

```
RT 1 5 18828
Disk ok

Q 0,1DF0E 1E860
Ready.
```

Add this interface code at $1DF0E:

```
~01DF0E MOVE.L  #0,00000060.S       Clear Copylock key
~01DF16 MOVE.W  D0,00040608         Move disk drive number to $40608
~01DF1C MOVEM.L D0-D7/A0-A6,-(A7)   Push all registers onto stack
~01DF20 LEA     00DFF000,A6         Load chip registers base address
~01DF26 LEA     00000060,A0         Load Copylock key store address
~01DF2C LEA     0007B000,A2         Load MFM buffer address
~01DF32 MOVE.L  #4,D0               4 bytes to read
~01DF38 MOVE.L  #0,D1               Track number 0
~01DF3E MOVE.L  #400,D2             Byte offset $400
~01DF44 MOVE.B  00040609,D5         Pass disk drive number to read from
~01DF4A BSR     000192EE            Call loader
~01DF4E MOVEM.L (A7)+,D0-D7/A0-A6   Pop all registers from stack
~01DF52 MOVE.L  00000060.S,D0       Copy retrieved Copylock key to data register D0
~01DF56 LEA     000404E8,A0         Load Copylock key table
~01DF5C MOVE.W  #2100,SR            Set status register
~01DF60 RTS                         Return to caller
```

And finally write it back to our disk:

```
WT 1 5 18828
Disk ok
```

That's it, we have now a fully functional cracked copy of Street Fighter 2!

Optionally we can extend the game to support up to 4 disk drives instead of 2 disk drives.
Read tracks 1-5 into memory $18828:

```
RT 1 5 18828
Disk ok
```

Extend the disk drive selection routine to also support DF2 and DF3.

```
~0197D6 BSR     0001DED6
~0197DA MOVEQ   #0,D0
~0197DC NOP
~0197DE NOP
~0197E0 BSR     0001DF0E
~0197E4 MOVE.W  00019208,D1
~0197EA ADD.W   D1,D1
~0197EC ADD.W   D1,D1
~0197EE CMP.L   0(A0,D1.W),D0
~0197F2 BEQ     00019834
~0197F4 MOVEQ   #1,D0
~0197F6 BSR     0001DF0E
~0197FA MOVE.W  00019208,D1
~019800 ADD.W   D1,D1
~019802 ADD.W   D1,D1
~019804 CMP.L   0(A0,D1.W),D0
~019808 BEQ     00019834
~01980A MOVEQ   #2,D0
~01980C BSR     0001DF0E
~019810 MOVE.W  00019208,D1
~019816 ADD.W   D1,D1
~019818 ADD.W   D1,D1
~01981A CMP.L   0(A0,D1.W),D0
~01981E BEQ     00019834
~019820 MOVEQ   #3,D0
~019822 BSR     0001DF0E
~019826 MOVE.W  00019208,D1
~01982C ADD.W   D1,D1
~01982E ADD.W   D1,D1
~019830 CMP.L   0(A0,D1.W),D0
~019834 RTS
```

Fix the call to the disk drive selection routine, write it back to disk and we're done.
We now just added new functionality to the game!

```
FA 19806
Search from: 000000 to: 100000
019260 BSR      00019806
Searched up to adr: 01B60A
Ready.

A 19260
^019260 BSR 197D6
^019264

WT 1 5 18828
Disk ok
```

Get a cool cracktro from www.flashtro.com and finalize your crack!
Let's have a play!

Greetings go out to the great Flashtro community for keeping the Amiga spirit alive!
Special thanks to Alpha One for his loader and his MFM tutorials!

scenex 2015