Acorn Electron

DUNJUNZ

Tape To Disc Conversion

Someone over on the stardot forums posted a request for a tape to disc conversion of the Acorn Electron version of dunjunz.

Having a bit of spare time and already posessing some knowledge of the Acorn Electron and I figured I'd give this a go. It turned out to not be as difficult to do as I expected and I thought i'd document the process for anyone who might be interested.

I used the Elkulator emulator to aid me in the task as the debugger thats built in allowed me to easily inspect the machine as it was running.

First step was to obtain the tape image in uef format and I used the following program that i had written previously to copy each tape file in turn to disk and preseve the metadata.

```
  10 INPUT"RAM TO RESERVE IN KB";M%

  20 IF M%=0 M%=(HIMEM-TOP)/&400: PRINT M%;"KB RESERVED"

  30 IF HIMEM-M%*&400<TOP PRINT "NOT ENOUGH SPARE MEMORY!": GOTO 10

  40HIMEM=HIMEM-M%*&400

  50PRINT"CURRENT HIMEM: "+STR$~(HIMEM)

  60*T.

  70H%=HIMEM

80P%=&120:[OPT2:PHA:TXA:PHA:LDA&3CA:TAX:AND#&FE:STA&3CA:TXA:AND#1:ORA&70:STA&70:PLA:TAX:PLA:RTS:]
?&220=&20:?&221=1:*FX14,4

  90?&3CA=0:?&70=0

 100OSCLI "*L. "+CHR$(34)+CHR$(34)+" "+STR$~(H%)

 110A%=&3B2:A$=""

 120REPEAT

 130A$=A$+CHR$(?A%)
```

```
 140A%=A%+1

 150UNTIL ?A%=0 OR A%=&3BA

 160L%=!&3BE: IF L%>=&FF0000 L%=L%+&FF000000

 170X%=!&3C2: IF X%>=&FF0000 X%=X%+&FF000000

 180S%=?&3C6*256+?&3C8

 190IF ?&3C8=0 THEN S%=S%+256

 200PRINT"FILENAME: "+A$

 210PRINT"LOAD ADDR: "+STR$~(L%)

 220PRINT"SIZE: "+STR$~(S%)

 230PRINT"EXEC ADDR: "+STR$~(X%)

 240IF ?&70<>0 THEN PRINT"LOCKED:YES" ELSE PRINT "LOCKED:NO"

 250*DISC

 260*DIR$

 270 PRINT"SAVE?(Y/N)";

 280*FX15,1

 290B$=GET$

 300 PRINT B$

 310IF B$="Y" OR B$="y" OSCLI"*SAVE "+CHR$(34)+A$+CHR$(34)+" "+STR$~(H%)+" "+STR$~(H%+S%)+"
"+STR$~(X%)+" "+STR$~(L%)

 320 PRINT "CONTINUE?(Y/N)";

 330*FX 15,1

 340 B$=GET$

 350 PRINT B$

 360 IF B$<>"N" AND B$<>"n" GOTO 60
```

The program is best run with shadow memory enabled otherwise you may not have enough memory to be able to load larger files. The program grabs the file metadata (load address, length and start address) from the temporary workspace area used by the OS. It then changes to DFS and saves the file.

It also installs an interrupt driven piece of machine code that constantly clears the locked flag in memory allowing locked files to be loaded when they would not normally be able to.

Once you have extract all of the tape files you should have DUNK, LOADER, TITLE, LOADER, DUNJUNZ and 25 Level?? files.

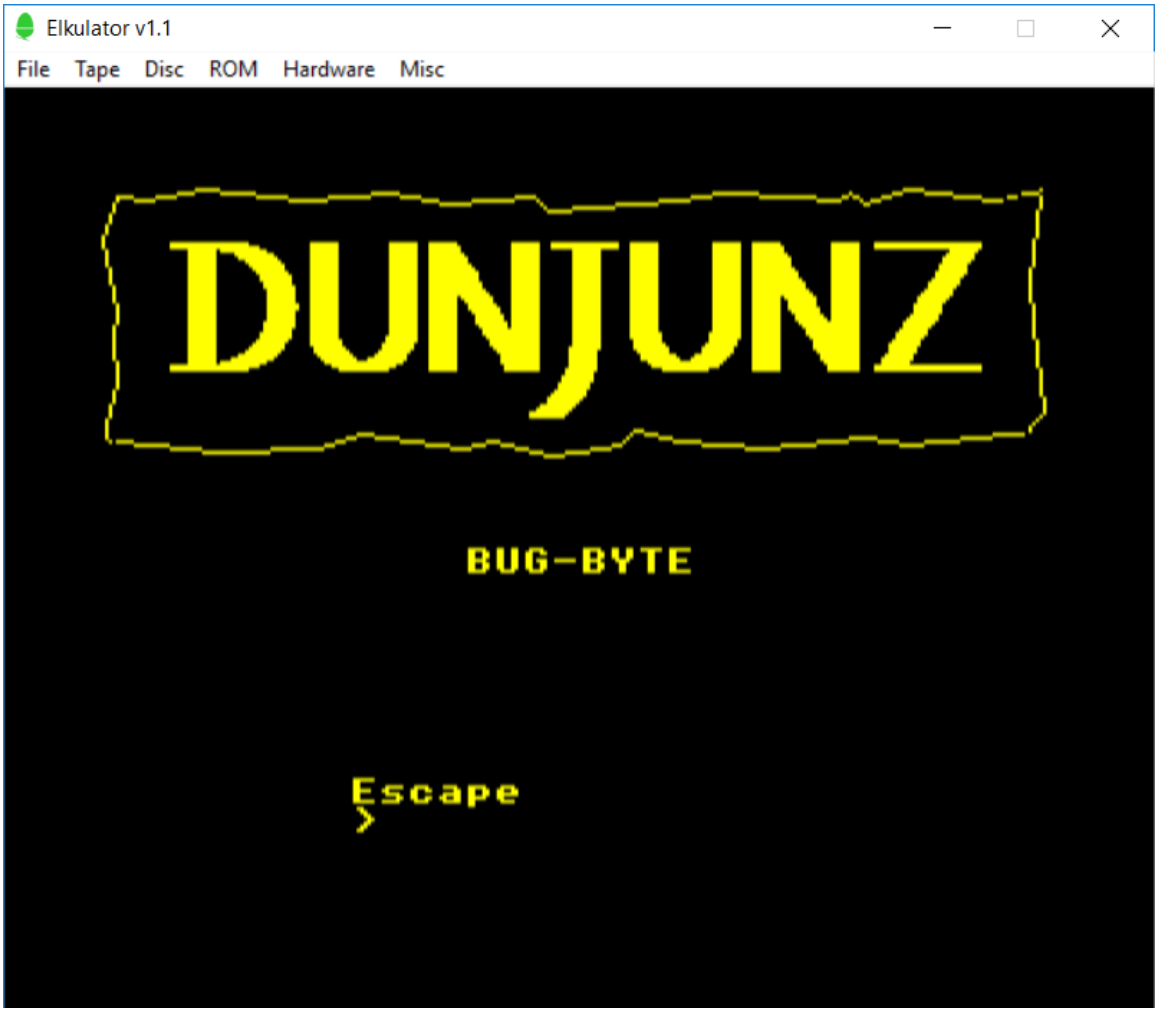
```
>*INFO*
$.Level25      006B00 006B00 0002B0 0AF
$.Level24      006B00 006B00 0002B0 0AC
$.Level23      006B00 006B00 0002B0 0A9
$.Level22      006B00 006B00 0002B0 0A6
$.Level21      006B00 006B00 0002B0 0A3
$.Level20      006B00 006B00 0002B0 0A0
$.Level19      006B00 006B00 0002B0 09D
$.Level18      006B00 006B00 0002B0 09A
$.Level17      006B00 006B00 0002B0 097
$.Level16      006B00 006B00 0002B0 094
$.Level15      006B00 006B00 0002B0 091
$.Level14      006B00 006B00 0002B0 08E
$.Level13      006B00 006B00 0002B0 08B
$.Level12      006B00 006B00 0002B0 088
$.Level11      006B00 006B00 0002B0 085
$.Level10      006B00 006B00 0002B0 082
$.Level9       006B00 006B00 0002B0 07F
$.Level8       006B00 006B00 0002B0 07C
$.Level7       006B00 006B00 0002B0 079
$.Level6       006B00 006B00 0002B0 076
$.Level5       006B00 006B00 0002B0 073
$.Level4       006B00 006B00 0002B0 070
$.Level3       006B00 0▫
```

```
$.Level19      006B00 006B00 0002B0 09D
$.Level18      006B00 006B00 0002B0 09A
$.Level17      006B00 006B00 0002B0 097
$.Level16      006B00 006B00 0002B0 094
$.Level15      006B00 006B00 0002B0 091
$.Level14      006B00 006B00 0002B0 08E
$.Level13      006B00 006B00 0002B0 08B
$.Level12      006B00 006B00 0002B0 088
$.Level11      006B00 006B00 0002B0 085
$.Level10      006B00 006B00 0002B0 082
$.Level9       006B00 006B00 0002B0 07F
$.Level8       006B00 006B00 0002B0 07C
$.Level7       006B00 006B00 0002B0 079
$.Level6       006B00 006B00 0002B0 076
$.Level5       006B00 006B00 0002B0 073
$.Level4       006B00 006B00 0002B0 070
$.Level3       006B00 006B00 0002B0 06D
$.Level2       006B00 006B00 0002B0 06A
$.Dunjunz      0008C0 0008C0 004D96 01C
$.TITLE        002000 002000 000D40 00E
$.LOADER       000400 000740 0003FF 00A
$.T2D          FF0E00 FF802B 0003B6 006
$.Level1       006B00 006B00 0002B0 003
$.DUNK         002000 000E00 000008 002
>_
```

If we now load the DUNK file we will see

This is no good for running from disc, so lets change this and save back the DUNK file.

Now if we try and CHAIN the DUNK file it does start loading but we get as far as



It looks as though it loaded the LOADER file, then loaded the TITLE file and displayed it but then it crashed out with an escape error for some reason even though we never touched the escape.

So now we need to start digging into the loader file to see whats in there.

Looking at the file list above we can see the LOADER file loads into address &400 and starts running at address &740. So i put a breakpoint im the elkulator debugger at &740 and CHAIN"DUNK". Now using the debugger disassembler we get the following code:

```
0740 : LDA #16
0742 : JSR FFEE   - OSWRCH
0745 : LDA #04
0747 : JSR FFEE   - OSWRCH   - VDU 22,4 (select MODE 4)
074A : LDA #13
074C : JSR FFEE   - OSWRCH
074F : LDA #07
0751 : JSR FFEE
0754 : LDA #03
0756 : JSR FFEE   - OSWRCH
0759 : LDA #00
075B : JSR FFEE   - OSWRCH
075E : JSR FFEE   - OSWRCH
0761 : JSR FFEE   - OSWRCH  - VDU 19,7,3,0,0,0 (define logical colour)
0764 : JSR 07BB   - define text window
0767 : LDX #D6
0769 : LDY #07
076B : JSR FFF7   - OSCLI (*LOAD Title)
076E : LDA #00
0770 : STA 72
0772 : LDA #C0
0774 : STA 70
0776 : LDA #20
0778 : STA 73
077A : LDA #5B   ; &70/71 = &5bc0
077C : STA 71    ; &72/73 = &2000
```

```
077E : LDY #00
0780 : LDA (72),Y ;get from source
0782 : STA (70),Y ;copy to screen
0784 : INY       ;next byte
0785 : BNE 0780   ;repeat 256 times
0787 : INC 73     ;next block of 256 from source
0789 : INC 71     ;next block of 256 to destination
078B : LDA 71
078D : CMP #69   ;repeat back until we get to the end
078F : BNE 0780
0791 : LDA #1A
0793 : JSR FFEE   - OSWRCH  - VDU 26 - restore default window
0796 : LDA #1F
0798 : JSR FFEE   - OSWRCH
079B : LDA #10
079D : JSR FFEE   - OSWRCH
07A0 : JSR FFEE   - OSWRCH  - VDU 31,16,16 - move text cursor to 16,16

;display BUG-BYTE text
07A3 : LDX #07
07A5 : LDA 07F5,X
07A8 : JSR FFEE   - OSWRCH
07AB : DEX
07AC : BPL 07A5

07AE : JSR 07BB   - define window
07B1 : LDX #E1
07B3 : LDY #07
07B5 : JSR FFF7   - OSCLI (*LOAD Dunjunz)
07B8 : JMP 0D00   - call main dunjunz code

;this does VDU 28,12,25,28,23 - define text window
07BB : LDA #1C
07BD : JSR FFEE   - OSWRCH
```

```
07C0 : LDA #0C

07C2 : JSR FFEE   - OSWRCH

07C5 : LDA #19

07C7 : JSR FFEE   - OSWRCH

07CA : LDA #1C

07CC : JSR FFEE   - OSWRCH

07CF : LDA #17

07D1 : JSR FFEE   - OSWRCH

07D4 : RTS


07D5 : 54 4C 4F 41 44 20 54 69 74 6C 65 0D 4C 4F 41 44   TLOAD Title.LOAD

07E5 : 20 44 75 6E 6A 75 6E 7A 0D 20 44 75 6E 6A 75 6E    Dunjunz. Dunjun

07F5 : 45 54 59 42 2D 47 55 42 0D 00 00 00 00 00 00 11   ETYB-GUB........
```

This is all pretty straightforward stuff. If we put a breakpoint at &7B5 and let the code run we can see that everything works fine right up to this point. If we now put a breakpoint at the next instruction &7B8 and start it running again we get the following.

```
   0812 : BRK
0740 : LDA #16       >break 7b5
   Breakpoint 1 set to 07B5
0740 : LDA #16       >c
   Break at 07B5
07B5 : JSR FFF7      >break 7b8
   Breakpoint 2 set to 07B8
07B5 : JSR FFF7      >c
BRK 0100! 0100 0000
0100 : BRK           >
```

This indicates something went wrong and the emulator has trapped a call to the error handler routine at &100. If you use C to continue execution you will then get the same error we saw previously.

Looking at the file list above we can see that the Dunjunz file loads into memory at address &8C0 so we can assume this is interfering with the disk system in some way and causing the error. We can confirm this by doing a *LOAD on that file and we see the same problem.

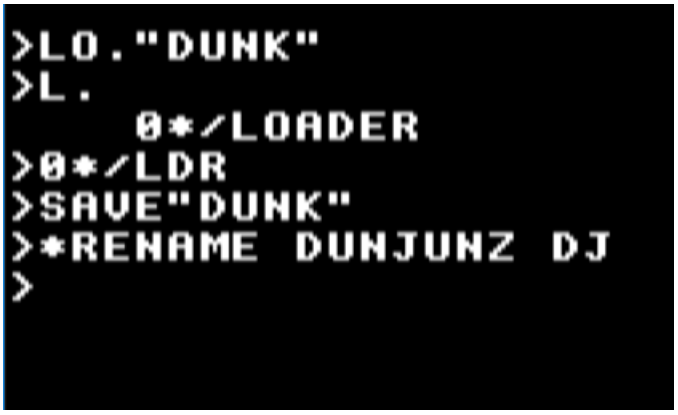
If we can modify this loader file to load the dunjunz file higher up in memory we can then relocate the file into the correct location before executing it as normal.

Here is the code i used to modify the LOADER file.

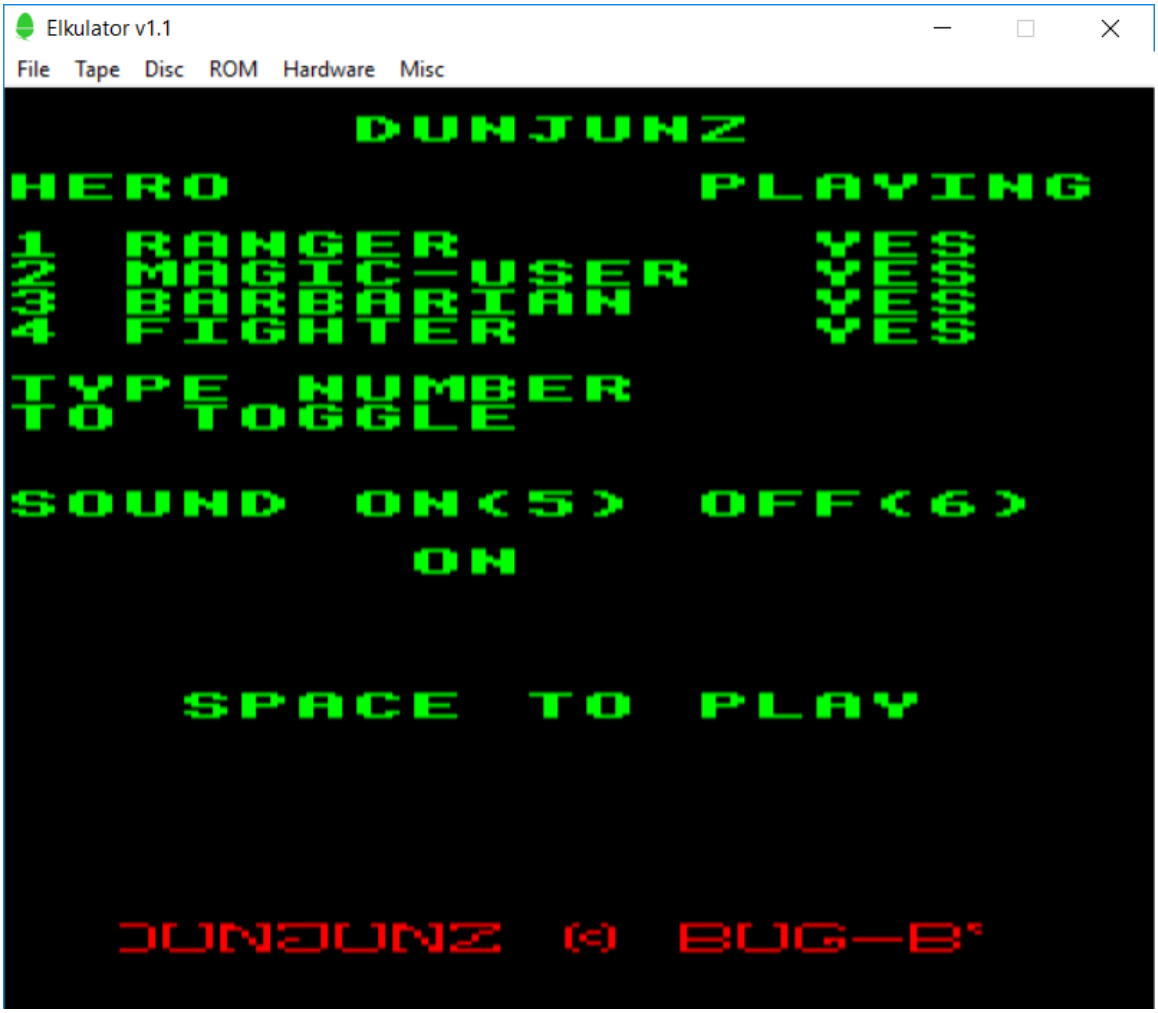
```
  10 P%=&1880

  20 *L.LOADER 1400     - load the original LOADER file to &1400

  30 ?&17BA=8

  40 ?&17B9=&80         - patch the original JUMP &D00 to jump to our new code at &880

  50 $&17E1="L.DJ E00"   - force the load to &E00, we have also renamed the Dunjunz file to DJ so
we can fit this string in place of the old one

  60 [

  70 LDA#0

  80 STA &72

  90 LDA #&E

 100 STA &73  - &72/73 = &E00

 110 LDA #&C0

 120 STA &70

 130 LDA #&8

 140 STA &71  - &70/71 = &7C0

 150.loop2

 160 LDY #0

 170.loop

 180 LDA (&72),Y - load a byte

 190 STA (&70),Y - copy to destination
```

```
200 INY
210 BNE loop    - repeat for 256 bytes
220 INC &73     - update source to next block
230 INC &71     - update destination to next block
240 LDA &71
250 CMP #&5C    - loop back until we are done
260 BNE loop2
270 JMP &D00    - start relocated code
280]
290*SAVE LDR 1400+500 740 400  - save updated loader as LDR
```

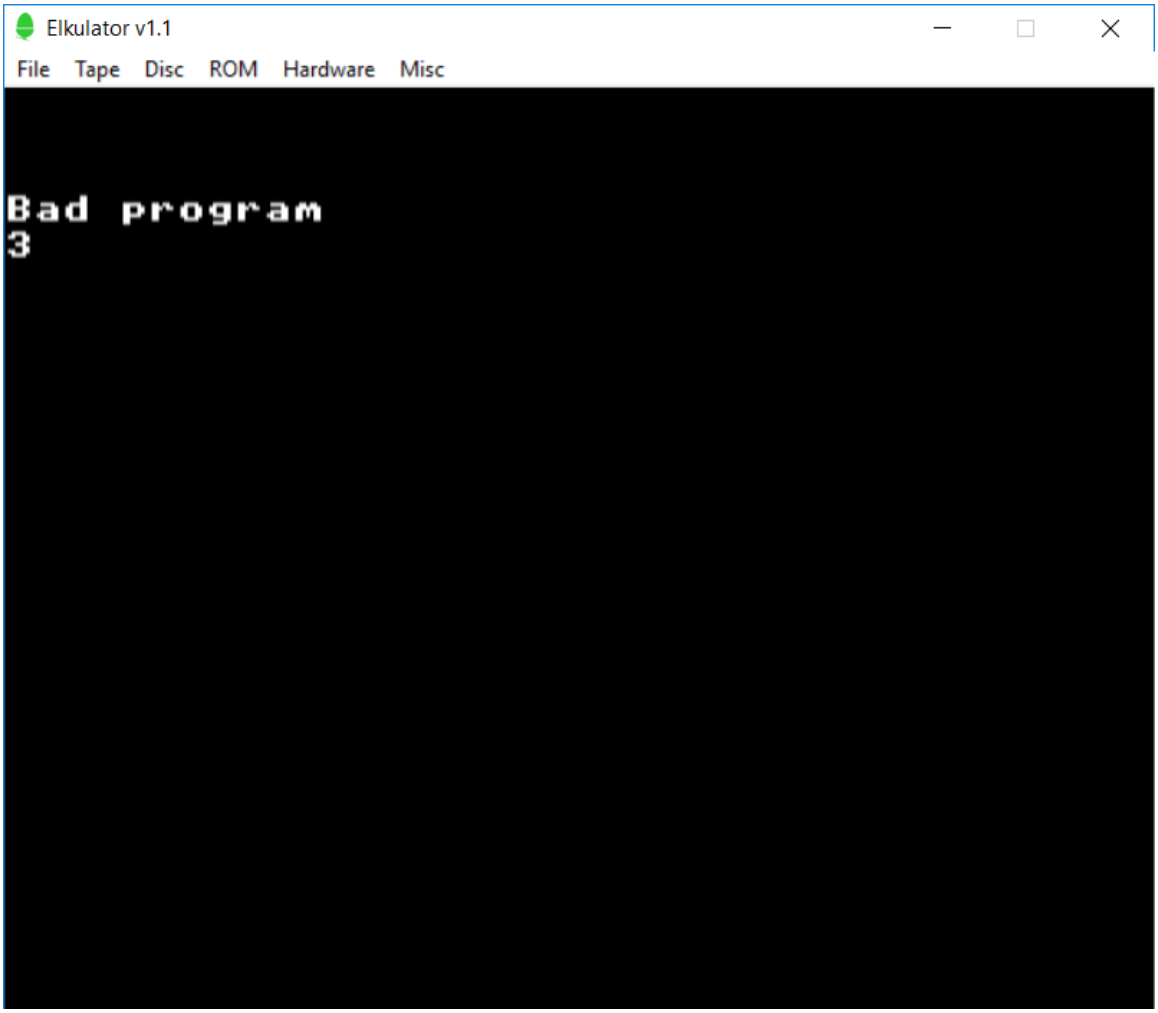
Now we can run the program to generate the new LDR program, then we update the DUNK program and rename the Dunjunz file to DJ



Now we can CHAIN"DUNK" and we are getting somewhere. The game loads right up to the main menu.



However, if we now try and start the game we get.



Its trying to load in the first level and crashing out. We need to figure out a way to allow it to load even though the memory used by the disc system has been overwritten.

From previous experience i figed out that page &D memory (&D00-&DFF) is critical when running disc operations and we already know that the DJ file exists in memory starting at address &7C0 and is &4D96 bytes in length so would definitely overwrite this area.

Do a reset and start the debugger and dump the area of &D00.

```
E2E8 : BCS E350     >m d00

   0D00 : 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  @...............

   0D10 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................

   0D20 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................

   0D30 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................

   0D40 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................

   0D50 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```
0D60 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0D70 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0D80 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0D90 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0DA0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0DB0 : 00 00 00 00 00 00 00 00 00 00 59 9B 03 68 96 03  ..........Y¢.hû.
0DC0 : 61 97 03 4F 98 03 9D 9B 03 D9 94 03 8A 9B 03 00  aù.Oÿ.¥¢.┘ö.è¢..
0DD0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0DE0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0DF0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

we can see that very little of this memory appears to be used. By running the below program we can see that clearing out this are does indeed prevent disc access from working correctly.

We can also try:

```
>?&D00=0
>*.
 (50) FM
Drive 0                    Option 3 (EXEC)
Dir. :0.$                  Lib. :0.$

     DJ                         DUNK
     HAK1                       LDR
     Level1                     Level10
     Level11                    Level12
     Level13                    Level14
     Level15                    Level16
     Level17                    Level18
     Level19                    Level2
     Level20                    Level21
     Level22                    Level23
     Level24                    Level25
     Level3                     Level4
     Level5                     Level6
     Level7                     Level8
     Level9                     LOADER
     TITLE
>_
```

and see that the first byte at &D00 does not seem to cause any issues so we can concentrate on &DB0-&DCF.

Using this program we can verify that by taking a copy of &DB0-&DCF and clearing out the whole of page &D and then restoring those bytes we can prove that we can do disc access afterwards.

```
    10 FOR I%=&DB0 TO &DCF
    20 ?(&4000+I%)=?I%
    30 NEXT
    40 FOR I%=&D00 TO &DFF
    50 ?I%=0
    60 NEXT
    70 FOR I%=&DB0 TO &DCF
    80 ?I%=?(&4000+I%)
    90 NEXT
>
```

So we now need to find the location where the level files are loaded in and hopefully patch the loader so that it can restore the necessary data in &DB0-&DCF before issuing the load command.

We also need to find some free memory where we can take a copy of this data before it is overwritten.

after some debugging I found the following code.

```
1479 : LDA #D0

147B : STA 3E60

147E : LDA #16

1480 : STA 3E61   ;get filename and save to control block filename

1483 : LDX #60

1485 : LDY #3E    ;control block for OSFILE

1487 : LDA #3B    ;get &3B

1489 : STA 3E63   ;save to load address (&3B00)

148C : LDA 0212   ; get OSFILE vector

148F : STA 149B   ; update JUMP below

1492 : LDA 0213   ; get OSFILE vector

1495 : STA 149C   ; update JUMP below

1498 : LDA #FF    ;OSFILE &FF (load file)

149A : JSR F27D   ;call OSFILE

149D : JMP 0450


16D0 : 4C 45 56 45 4C 31 0D 0D 70 00 8D 35 F8 00 00 00   LEVEL1..p.i5°...
```

I found this by loading the game from tape and breaking into the debugger during the level load and stepping through the rom code until I eventually ended up back in RAM at &149D.

So we can see that if we patch the code at &149A above to jump to our routine we can then restore the required memory and jump to OSFILE before returning to &149D.

We know that the game uses quite a bit of the memory from the files we have seen being loaded so far. I loaded up the game and played it for a while and did a dump of memory area &100-&1FF which is the stack. The bottom of the stack seems to be fairly free so as long as we keep our usage as small as possible we can try to fit our hack into this area.

So i wrote the following code:

```
10COLOUR 0

20FOR I%=0 TO 31:?(&100+I%)=?(&DB0+I%):NEXT

30P%=&120
```

```
 40[
 50OPT 2
 60PHA
 70TXA
 80PHA
 90LDX #0
100.loop
110LDA &100,X
120 STA &DB0,X
130INX
140CPX #32
150BNE loop
160PLA
170TAX
180PLA
190JMP &FFDD
200]
210*/LDR
```

And saved it over the original DUNK file.

Which we can see copies the memory area &DB0-&DCF to &100-&11F and then assembles a small piece of code to copy that data from &100 to &DB0 and calls OSFILE. We then run the usual LDR.

Ideally it would be better to swap over the memory areas &100-&11F and &DB0-&DCF to restore the memory before the load and then swap them back after the load. That way you could be sure that nothing was lost from the &DB0-&DCF. However in our case there is code that gets executed straight after the load routine is called that overwrites the &D00-&DFF memory anyway anyway, so we can save ourself a few bytes of space by not bothering to do that.

In order to patch the DJ file to call our code instead of calling OSFILE directly we could load up the file and make the necessary changes and write it back out again but since the file is so large i didnt want to risk corrupting the file so I took a slightly different approach.

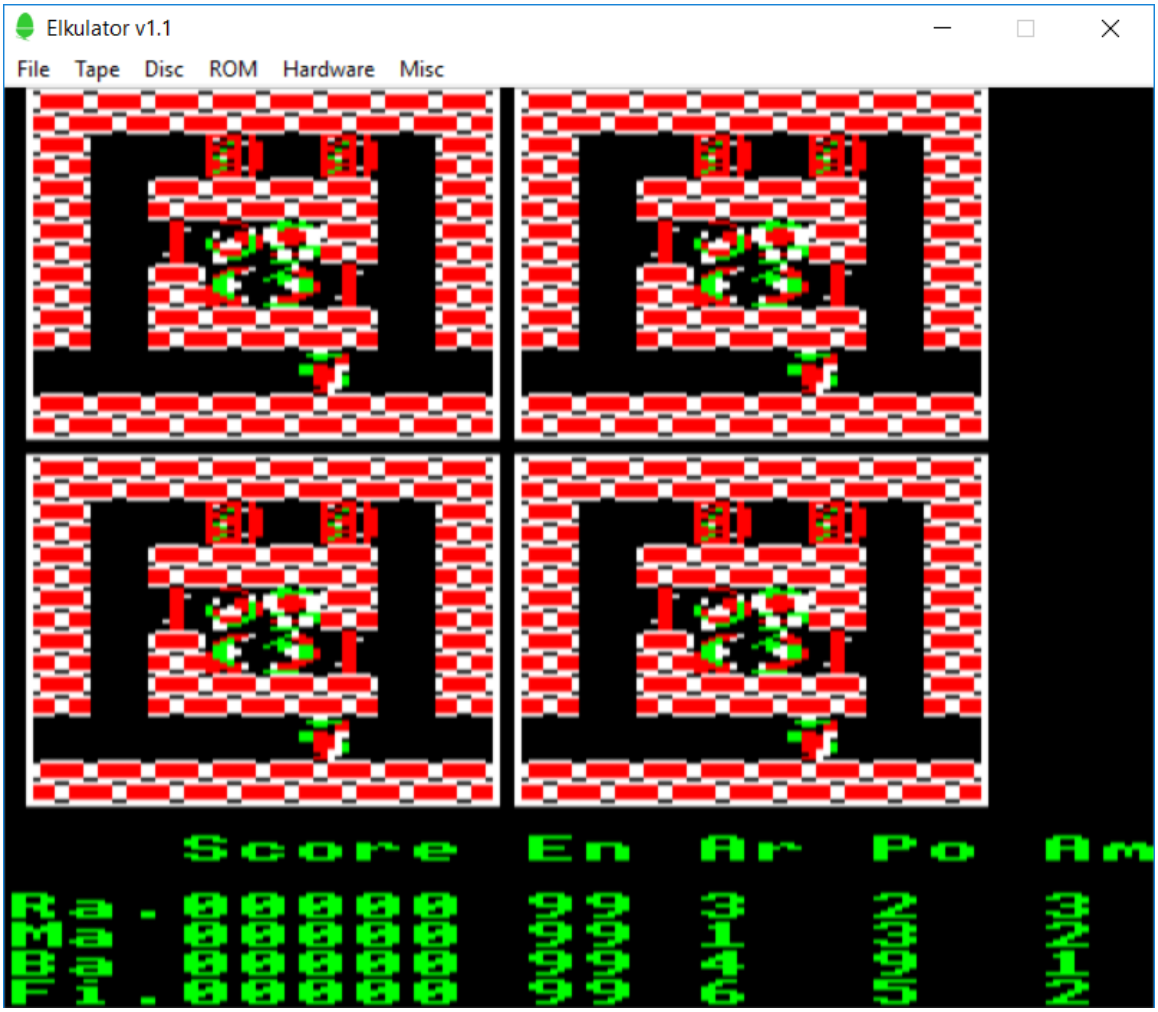
I modified our previous code to patch the loader to add an extra bit of code that patches the DJ file after it has been loaded.

```
 10 P%=&1880
 20 *L.LOADER 1400     - load the original LOADER file to &1400
```

```
 30 ?&17BA=8

 40 ?&17B9=&80        - patch the original JUMP &D00 to jump to our new code at &880

 50 $&17E1="L.DJ E00"  - force the load to &E00, we have also renamed the Dunjunz file to DJ so
we can fit this string in place of the old one

 60 [

 70 LDA#0

 80 STA &72

 90 LDA #&E

100 STA &73  - &72/73 = &E00

110 LDA #&C0

120 STA &70

130 LDA #&8

140 STA &71  - &70/71 = &7C0

150.loop2

160 LDY #0

170.loop

180 LDA (&72),Y - load a byte

190 STA (&70),Y - copy to destination

200 INY

210 BNE loop   - repeat for 256 bytes

220 INC &73    - update source to next block

230 INC &71    - update destination to next block

240 LDA &71

250 CMP #&5C   - loop back until we are done

260 BNE loop2

270 LDA #&EA   - get NOP opcode

280 LDX #0

290.loop3

300 STA &148C,X - blank code with NOP

310 INX

320 CPX #12

330 BNE loop3  - NOP out 12 bytes of code (&48C onwards)

340 LDA #32

350 STA &149B

360 LDA #1
```

```
370 STA &149C   - get the address of our routine &120 and update the JMP to OSFILE instruction
380 JMP &D00    - continue running
390]
400*SAVE LDR 1400+4BC 740 400
```

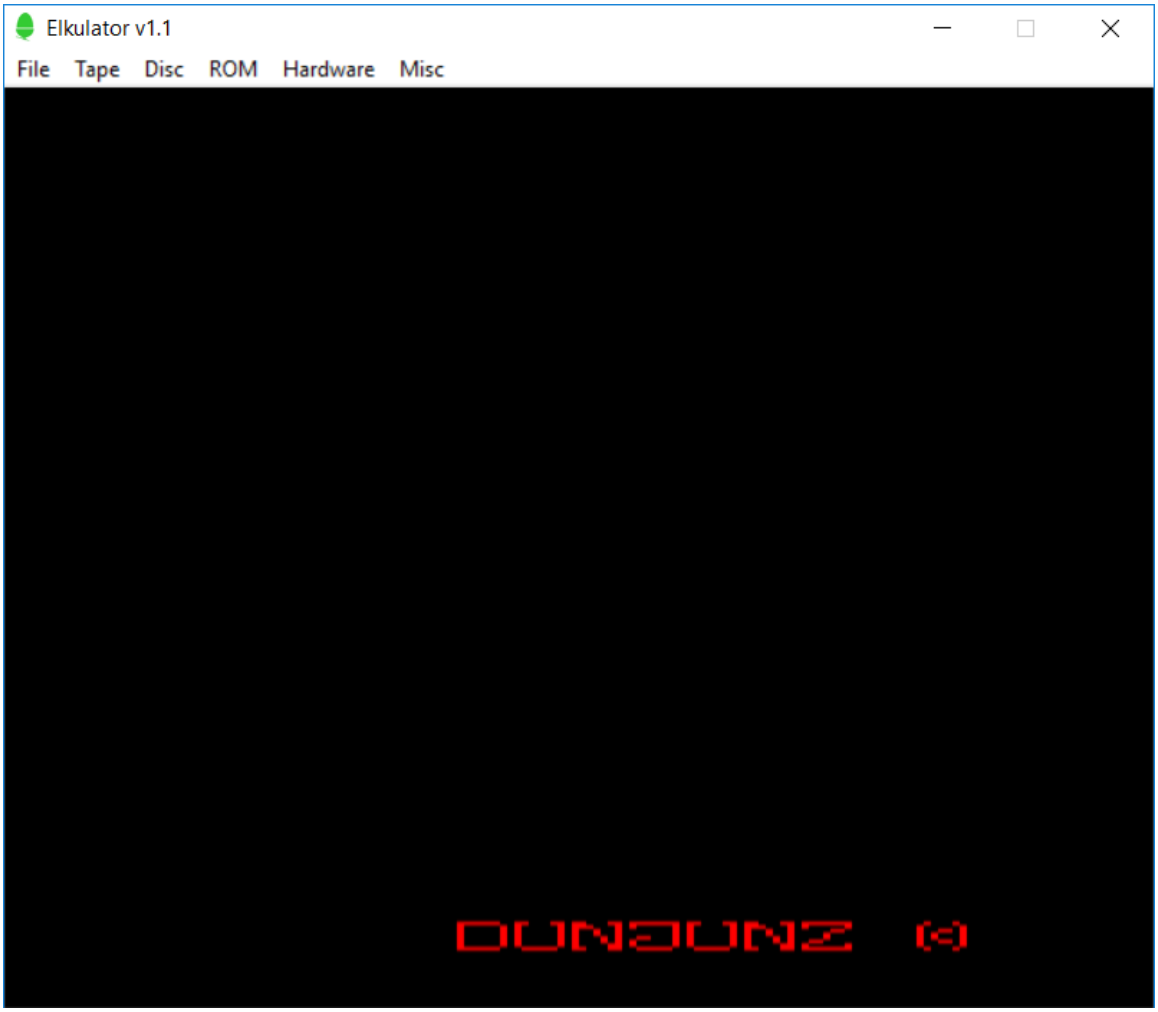
Now we can CHAIN"DUNK" and voila -



We are able to fully play the game.

All we need to do now is get the disc to auto boot and we are good to go. Create the !BOOT file and enable the

Everything should be working now but something slightly strange has happened.The main menu is not displaying correctly. It draws initially then blanks and all we get is the scrolly message.



I dug into why this was happening a little bit. I was not able to understand the exact cause but it appeared to be that if the prgram is run directly from !BOOT rather than from BASIC it did not load correctly. If i omitted the /*LDR part at the end of the basic code I was then able to isse /*LDR from basic and everything was fine.

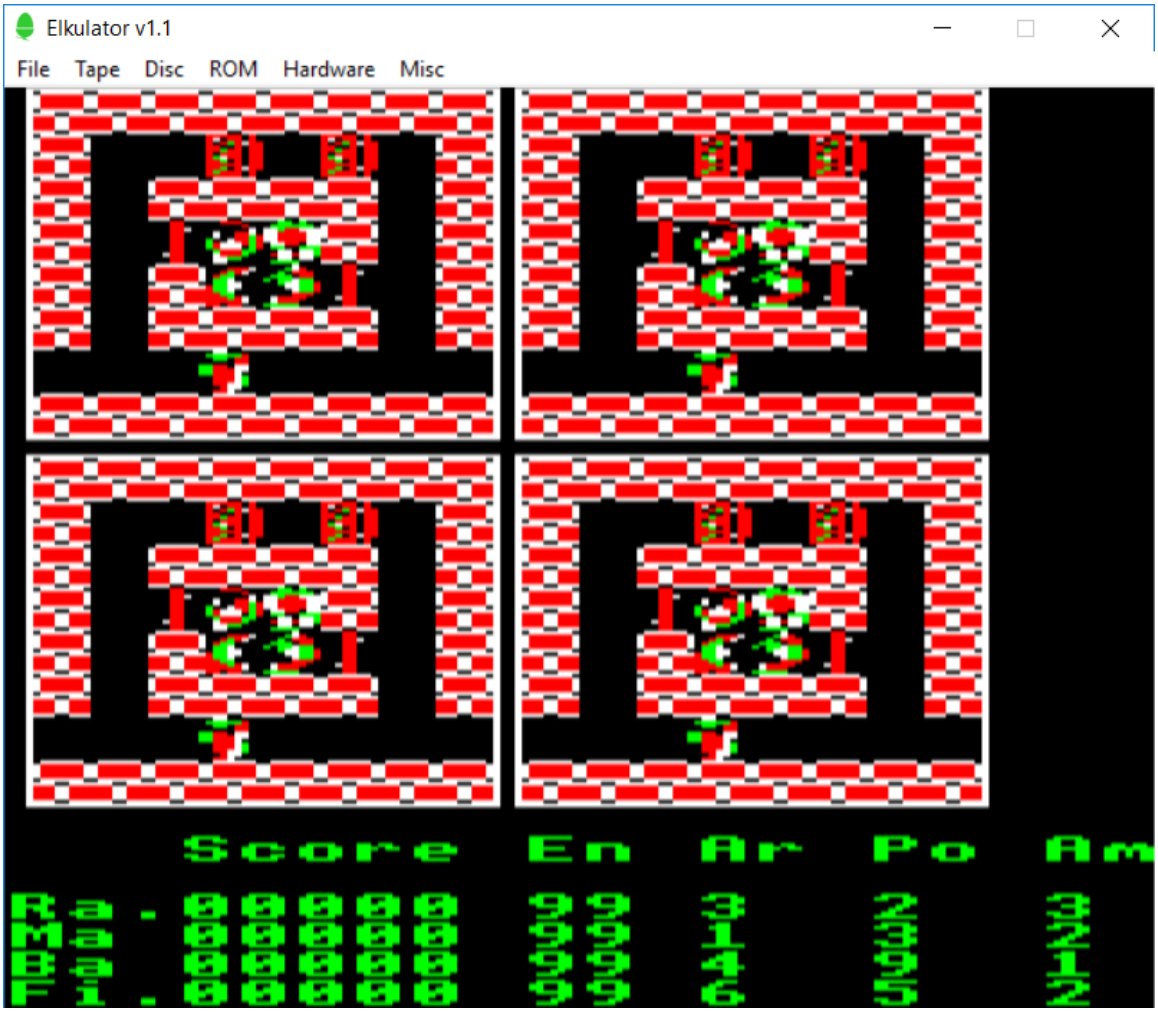
Instead of attempting to figure out why this was the case I found a workaround that enabled it to work correctly. I simply removed the /*LDR command and replaced it with a sequence of coommands to insert /*LDR into the keyboard buffer before ending the program. That way it is the same as if the command was manually typed by the user.

Here is the final DUNK file with all the changes.

```
  10 COLOUR 0
  20FOR I%=0 TO 31:?(&100+I%)=?(&DB0+I%):NEXT
  30P%=&120
  40[
  50 OPT 2
  60PHA
  70TXA
  80PHA
  90LDX #0
 100.loop
 110LDA &100,X
 120 STA &DB0,X
 130INX
 140CPX #32
 150BNE loop
 160PLA
 170TAX
 180PLA
 190JMP &FFDD
 200]
 210*FX138,0,42
 220*FX138,0,47
 230*FX138,0,76
 240*FX138,0,68
```

```
  250*FX138,0,82

  260*FX138,0,13   - send the codes 42,47,76,68,82,13 to the keyboard
buffer eg. "/*LDR",13
```

Now boot from the disc again and finally we have a fully working version of Dunjunz booting from DFS disc.



Thats it. I've only tested a couple of levels so there's a chance there could be issues further into the game.

Enjoy

Darren