

R E B E L I O N E

# OLD SCHOOL CRACKER

VOLUME 1

O L D S C H O O L C R A C K E R

# OLD SCHOOL CRACKER

## V O L U M E 1

**Contact:**

REBEL 1 of Hokuto Force  
PLK 080675 C  
3500 Kassel  
West-Germany

*Thanks to: **ACIN** for being the best crew on earth; **Bacchus/Fairlight** for FAIRLIGHT TV; **FORUM64.de** for answering all my stupid questions (trb, Hoogo, 1570, JeeK, ogd, goloMAK, Ruudi, Krill, muffi, Chrille, atomcode, GoDot, CapFuture1975, cruncher, Acorn, Claus, 8R0TK4\$T3N); **csdb** for the archive; **Gideon** for the Ultimate II+; **icomp** for the Turbo Chameleon 64 V2; **Christoph Fichtenkopp & Baltruschard** for being the best of buddies.*

*"Real cracking is about technical achievement,  
pride, dignity, and a dedication to quality."*

QED/TRIANGLE

# PREFACE

Welcome to the first edition of the *Old School Cracker* (OSC). The OSC is for all those C64 enthusiasts who, back in the heady days of the C64, never mastered the craft of cracking but always wondered how it really worked. It's also for anyone curious about how things were done on the real machine – without modern tools like *VICE*. *OSC* represents authenticity: We only want to crack with tools that were available at the time the game was released. Anything else would be pointless.

OSC Volume 1 dives into cracking the tape version of *Pitfall II* released in 1984 by Activision. Why tape you might ask? *"In Europe, most software was released on tape before on disk (if ever). So, tapes were not only **cheaper** and **easier** to crack than disks, but also **earlier**. This made tapes the most attractive for most crackers"* (SLC of Kvasigen). So, there we have it!

We're starting with a relatively simple crack. The target audience is the beginner in the cracking business. In future OSC issues, we'll also tackle

more modern copy-protection schemes. But: one step at a time. Let's begin with something fairly straightforward. We'll learn a lot that will be useful for bigger tasks down the road.

OSC was born out of the idea to have a complete step-by-step description of a C64 cracking process, something understandable and relatable from which we can all learn. OSC documents a piece of how the old skill of cracking works and aims to preserve it for future C64 generations.

The logical structure of OSC will always be similar: cracking, training and hacking. We start with an analysis of the code, tracing the whole thing through to the start of the game. However, tracing down the whole thing does not mean that we have to understand the entire code of the game. We absolutely want to stay focused and only take care of the stuff that is relevant to us. In this way, we will identify the built-in protections and crack them. To make sure that our crack works 100%, we will build a trainer that makes it easier for us to play through the game (and to improve the game!). And finally, we'll hack around in the game, just for fun.

A note on my own behalf: OSC is a public domain work - you can download it as a free PDF. Share it if you find it useful. I definitely recommend that you print it out and have it bound as an A5 hardcover. It looks better and you can work with it much more easily. If you find any errors or would like to suggest improvements, please drop me a message.

*"Cracking a program is more than just making a backup copy.  
It involves learning about the program."*

THE DOCTOR, PPM II

# INTRO

1989. The door of the kids' bedroom slams shut with a resounding echo. The college jacket is carelessly tossed onto the untidy bed, while our precious bounty, a package bulging with 5 1/4 inch floppy disks picked up from the post office (keyword: PLK!), finds its place on the desk. Those with keen eyes will notice the thin layer of hairspray delicately coating the stamp, and grins begin to spread across our faces.

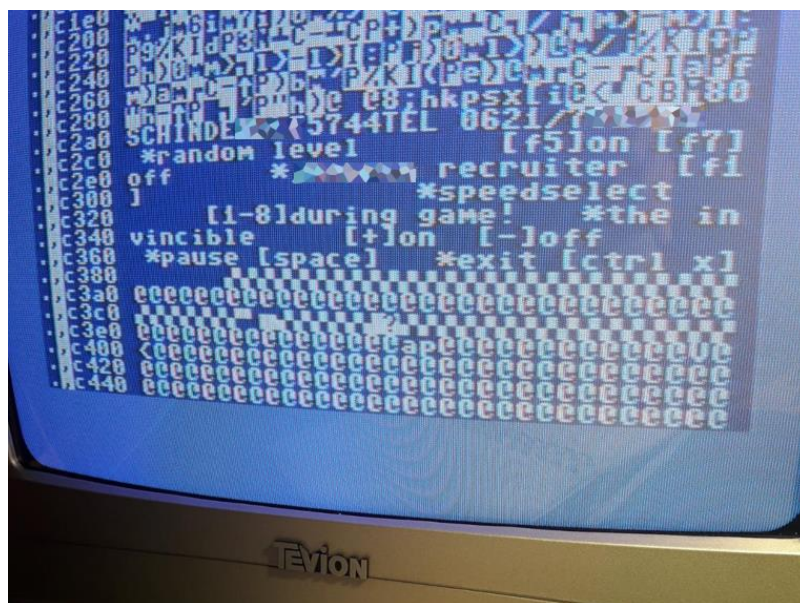
"Stamps back!" declares the instruction on the enclosed pirate's letter, its graffiti-like handwriting adorned with sci-fi comic characters. This promises to be a perfect afternoon, destined to stretch seamlessly into the depths of the night. Outside, it's December, it's already freezing cold and it's even getting dark again. But inside here in the room it's an oasis of time brimming with the latest and hottest stuff: vibrant intros, mesmerizing raster effects, scroll-text greetings, and juicy gossip gleaned from the top-notch disk magazines. And let's not forget the real

gems—cracked games that captivate our attention. A league of smart, cool youngsters, well integrated into an underground infrastructure, with secret informants infiltrating the most exclusive software circles. They've dissected the cryptic codes of cutting-edge software, outsmarting copy protection mechanisms with finesse, all for the thrill of endless duplication. Damn, that's the shit!

As the 1541 disk drive purrs and whirs, the *Action Replay MK6* kicks into action, swiftly loading almost 200 blocks into the memory of our beloved bread box (well, almost swiftly!). But hold on, what's that? The doorbell chimes two times, signaling the arrival of our buddy, eager not to miss out on the latest developments. And this is how this wonderful afternoon goes: Games are played, scroll-texts devoured, and SID music cranked up to the max. How did they pull off those tricks again? Such vibrant logos, so many sprites (more than 8 in the border, wtf?!), and that dancing scroll text—impossible!

"Hey, how does cracking even work?" inquires our buddy. "I'm not entirely sure, Dude. You need a machine language monitor, and assembler is the magic key. You've got to locate the copy protection and outsmart it!" I respond. "Cool... We should give it a shot." Indeed, he's onto something. How epic would it be? The mental movie begins: Two young guns from the suburban trenches, cracking the first game and unleashing it upon the world with a slick cracktro. All the "*lamers*" in town would wonder who these mysterious maestros were, just as we pondered at that very moment.

Deep in that night, after all the latest games have been played, I fire up the machine language monitor, and me and my buddy delve into the intricate code. Cryptic characters dance before our eyes until, at line \$C2a0, we're jolted from our trance by plain text: "Tel 0621/7...".  
"...Sheesh...!"<sup>1</sup>



**Figure 1.** A message from the past. I have pixelated parts of the name and phone number to protect the “innocent”.

---

<sup>1</sup> I spent some time with another classic c64 game in 2023. In this context, I downloaded all available cracks from csdb. I wanted to see if there were any re-cracks among them. Actually, what I found hidden in the ML code (to my greatest surprise) was a phone number from *my home town* (sic!) at the aforementioned memory address of one version. You can imagine how strange that felt: You look into the past with your machine language monitor - 40 years back - and find a reference to a cracker who lived in the same town as you do now. I really thought that Paola and Kurt Felix were about to turn up... Next morning, I called the phone number of course with great doubts about success in my mind: How likely is it that a 40-year-old phone number still works today? If it still works, how likely is it that the owner of the number at the time still has it? It rings.... An older gentleman answers *with the name* from the C64 memory. I gulp. “Mr. XYZ, ...I found your number in the machine code of a C64 game”. You can literally hear the confusion at the other end of the line. To cut a long story short. The gentleman told me that it must be one of his sons - he himself has nothing to do with the c64. He gave me the phone numbers and I actually spoke to the cracker at the time. Isn't it great that things like that happen!



This scene played out in countless teenager's rooms across the globe in the mid-80s, each trajectory as unique as the home-computer users themselves. My journey fell somewhere in the middle—I was a tad too young and lacked the right connections to dive headfirst into assembler and cracking. Instead, I found my niche designing logos for some little (unknown) groups, trading with a handful of swappers, and relishing the thrill of the postal exchange. With Speeder and Spirou I was at least once in Venlo in 1989 at one of the infamous copy parties. But as 1990 dawned, the allure faded, and new interests took center stage.

Fast forward almost four decades, and here I am, with another bread box on my desk (and four more stowed away in the basement). Much has changed for the better—I no longer rely on a floppy drive, thanks to modern replacements like the *Turbo Chameleon 64 V2* or the *Ultimate II+*, which embrace SD cards and USB sticks. Rediscovering my favorite old games, demos, intros, and tools, I realize the *scene* is alive and kicking. *Fairlight* reigns supreme on YouTube, while *Genesis Project*, *Triad*, *Censor Design*, *Finnish Gold*, *Arsenic* and others continue to push the boundaries with demos, graphics, SID music and games.

So now, armed with a (hopefully) more mature brain, I revisit the question that lingered in my mind all those years ago: "How does cracking actually work?" This time, I'm determined to find the answer. I want to delve into the past, crack my favorite games, and share my newfound knowledge with the world. And so, dear reader, what you hold in your hands today is the culmination of a journey that began forty years ago—a journey into the heart of cracking, as we unravel the

secrets of *Pitfall II* step by step. And it's a bit like the starting image in *Pitfall II*: me, the little green man, no plan, no manual and the goal (Quickclaw below me) so close and yet out of reach. I hope you enjoy the ride.

Why *Pitfall II*? Well, there are several good reasons for that. Firstly, I have the fondest memories of this game. I still vividly remember how I dove right in and immediately fell in love with the gameplay mechanics. The animation of the little green figure seemed relatively smooth, and while the enemies (bats, vultures, electric eels (WOW), scorpions, deadly frogs) weren't exactly a walk in the park, they were manageable. The “overworld” graphics with the towering trees, the water Pitfall Harry could swim in (second WOW), and of course, the first gold bars lying deep at the bottom of the water—just the right adventure. Of course, without a manual, I had no idea what I was supposed to do. This “twitching cat”, so close yet unreachable beneath me - what the hell does that mean? The white mouse in the second level—what the heck? And when I tried to approach the mouse (from the front) out of the water, it darted at me like stung by a scorpion and threw me back into the depths of the water... Dude! What was going on here? I remember searching for secret passages underwater. But I found nothing except gold. Then, when I had finally descended to the very bottom of the caverns for the first time and was waiting at the abyss not knowing what to do next, a balloon appeared from the left. A blue, promising balloon, and I made the highest jump of my life and... flew! It took me some time to figure out how to collect the diamond ring, and jumping through the

caves is always an absolute highlight. So, I thought, the game is simply worth it. Plus, I hoped that such an old game wouldn't have any insurmountable copy protection mechanisms...

Why the *tape version*? Admittedly, I never in my wildest dreams thought that I would ever fall back on using datasette again in my life. Agonizingly long loading times. The proverbial coffee break during loading is a joke — you could hold an entire traditional Japanese tea ceremony during the loading process! Do we need that? No way! But, as often in life, there's a little "but": Tapes are just much easier to crack than disks—but still challenging enough for beginners to learn a few important tricks. And that's what it's all about: we want to learn the ropes! So, back to the tape culture and tea ceremonies.

A few words about the making-of the text you currently hold in your hands. I've pilfered where I could, and if you're thinking, "Hey, I recognize this from Wikipedia" or something — well, you're probably right. The manual, of course, is swiped (thanks, internet!), and the nifty map of the caverns, too. My aim wasn't to pen a book about *Pitfall II*, but to compile all the information about *Pitfall II* that I deem crucial. I've slapped citations everywhere so you're always in the loop. The truly authentic aspect of this text is the cracking documentation. "Hey, how does cracking even work?" inquires our buddy. I have tried to answer this one question that has been nagging at me since it was first asked.

**REBEL ONE**

*"Don't mix up pressing the freeze button to enter the running program  
with freezing the game to disk."  
(BACCHUS/FLT)*

## METHODS

Which tools are we using and could we crack Pitfall II with tools available at the time when the game was released? Well, the answer to the later question is a clear yes! However, we may not have worked in the same way as described here. We may have reset the computer and simply searched for a SEI as the entry point. And, yes, this totally works fine and it's quick. We may have screened the memory and had good guesses about what part of the memory to save. We had a proper copy at the time of the *Pitfall II* release.

In OSC I we are using AR4.2 – why not AR6 you might ask? Well, AR6 is a powerful cartridge of 1988/89 equipped with a sophisticated freezer. With this tool we are able to trace the code by setting breakpoints (I am not talking about the dedicated freezepoint/breakpoint function but simple BRKs in the assembly code) at specific locations e.g. RTSs. The BRK fires up the AR6 freezer menu with its MLM in the freezer mode. Advantageous because we do not have to care about e.g. the actual

screen mode, stack, registers etc. With simple MLM or earlier AR versions (e.g. 4.2) a BRK does not fire up a freezer menu MLM but a "normal" MLM at the basic mode level. However, if the game shows a HIRES loader screen while the BRK hits, we will not be able to work without further ado. We have to modify the code e.g. to avoid HIRES mode etc. This will of course need a little more work. When you start thinking about what tools should be used, you will probably come to the answer that the only way would be to use tools available at the time point of the game release. Uhhhhmmm... back 1984/85. The earliest cartridges available were ISEPIC (1985), CODEBUSTER (1985), Formel-64 (1985)... well all a year after *Pitfall II* was released. The procedure described in OSC I will enable us to carry out the crack even with a simple software MLM from 1984.

Before we start, here's a handy list of essentials you might find useful. If you opt for emulation, you'll need nothing else but a PC and *VICE*, but if you crave that authentic experience, you'll want: a real C64, a *datasette* (yes, seriously!), preferable an *Ultimate II+* (for the full retro vibe), and an original tape version of *Pitfall II* (eBay is your friend). You should also put a machine language monitor (MLM) on your Ultimate. If you don't know what a MLM is, consult some introduction material. It is mandatory that you have an MLM and that you know how to use it. I'm a big fan of the *Action Replay* - the best cartridge in the world – At [pokefinder.org](http://pokefinder.org) you will find all the CRTs you need. Activate AR4.2.

I have chosen to purchase an original tape copy of *Pitfall II*. I didn't want a worn-out version, but the best one I could find. For me, these are

artifacts from another time and I love that. You'll only need the original for a short time- when it's cracked, you can put it away. But unlike downloading a tap-file, you can put your artifact on your shelf and look at it again from time to time and sniff at it. Sometimes it is these small things that keep our soul healthy.

Ok, now that we know the hardware and software we need, let's talk about our methodology- what we want to do (cracking!) and what we don't want to do (freezing). *“Cracking is the **removal of protections** and can only properly be done by **going through the loading process and stopping the program exactly on the very instruction the game starts.** [...] This is VERY different from freezing. Freezing interrupts a running program and stores it on disk in the state of the freeze.”* (Bacchus/FLT).

So, it may make sense to note that it is not pressing the freeze-button that is highly frowned but the very next step will decide whether you are a cheap freeze cracker or not. If you freeze and opt BACKUP you find yourself in the backup menu. Pressing C (=“Disk Save – Standard”) now seals the end of your career as a renowned cracker. You will get a rather large 117 Blocks file of *Pitfall II* which will start after typing RUN with the typical freezer artefacts. Properly cracked, the file will have 65 Blocks before crunching it down to a handy 38 Blocks file – and of course, no ugly artefacts. But, apart from being large and ugly, the freezed version works fine. You can copy it without any problems. However, have you learnt anything about copy protection removal, my friend? I am afraid that you are doomed to admit that this is not the case. The only thing

you have done is using the freeze function of your AR and saving it. No more, no less.

The freeze function of your cartridge is an absolutely legitimate tool for cracking, mainly because it stores automatically all the important information (e.g. stack, status etc.) more or less untouched and gives us back control and the possibility to inspect the current state of the C64. In practice, our mission of cracking *Pitfall II* will start with setting a BRK to \$104b, that is after decryption of the next data bunch. This BRK will fire up the AR6 because the BREAK vector is modified by the AR to be \$dfcd:

```
dfcd      A9 23          LDA #$23
dfcf      8D 00 DE      STA $DE00
```

Writing \$23 to \$DE00 will result in a jump to the AR freezer menu if PC is above \$0FFF. The good news is that important data as the stack will be stored for us (actually it is stored for continuing the program after our job is done). In order to trace the code of *Pitfall II* after decryption, this is very handy. You can now enter the MLM, read out the stack, manipulate the next code snippets and set the next precise BRK point at strategically meaningful points in code (e.g. after loading in new data bunches or right before starting the game). Often an RTS is a good place for placing a BRK. The freeze functionality of your AR makes your work much easier. It saves you from having to manually store the status register, the accumulator and what the hell else whenever you want to

set breakpoints. Without the freeze function, e.g. when using an AR4.2 or a simple software MLM, you need to do a little bit more (see below).

I hope you understand my point here: it is absolutely fine to use the available tools in a smart way to develop your skills and really crack! Only make the world as difficult as you have to (don't cheat yourself). Think a while about what is your *ambition*. In my case for example, my ambition was to use *real hardware* and tools available at the time of the game release. Don't get me wrong! Apart from the *Pitfall II* project I am intensively using VICE and its monitor and it is the best tool ever. But my ambition was to learn how cracking was done in the old days. Perhaps some of you find this incomprehensible and argue that the VICE monitor gives you a much more valid insight into the code processes. In addition, the VICE monitor allows us to do real tracing. Or you can argue that cross-platform development was also practiced at that time. Yes, that's all absolutely true. But I've staked out the playing field this way and I think it best fulfils my requirements.

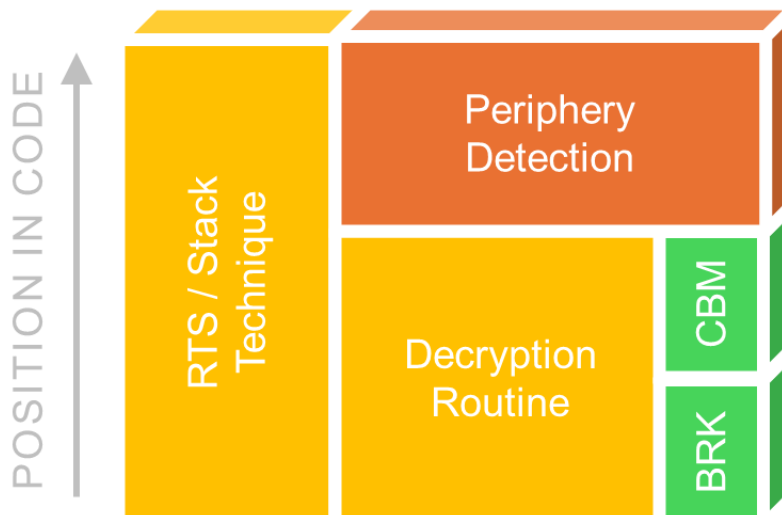
I ended up using the "bare ass on bed of nails" variant – using a MLM without the freezer functionality. In that way the crack is possible with a simple software MLM from 1984.



*"\*Anyone\* can press a button."*  
JAN LUND THOMSEN (AKA QED/TRIANGLE)

# CRACKING

*Pitfall II* is an educationally useful example of how to learn our cracker skills. It's not too easy, but not too hard either. It includes 5 different protection techniques, which are shown in figure 2.



**Figure 2.** The five different protection techniques in *Pitfall II* are listed chronologically according to their position in the code from bottom (beginning) to top (end). Coloring represents the difficulty from easy (green) to pain-in-the-ass (brownish).

## How does the program behave?

After putting the tape into the datasette and a shift-runstop, I pressed play and waited an impressive 8-9 minutes until the game started. No other periphery was connected to my C64. I switched off the computer and connected my Ultimate64-II in order to prepare a look into the code via the machine language monitor. I re-winded the tape and started loading again. Everything went exactly the same as in the first trial. However, at tape position 140 the tape stopped and the screen freezed. hmmm.... I replicated this behavior and remembered the loading instruction text in the manual which states *"make sure you have disconnected all peripheral equipment such as disc drive and printer"*. Also a connected and active AR-MK5 (real hardware) leads to a crash – this time directly after starting the first file. So, we can derive some hypotheses about what is going on here, right? Very likely disk drive & cartridge detection. Ok, let's load the header file using SYS 63276<sup>2</sup>: Start address \$0801 – no autostart. Nothing to re-allocate. Good. So, I loaded the loader using SYS 62828 and LIST showed 10 SYS 4096 (=\$1000). I sniffed through the code from \$1000 (see next page) using the MLM and, to my surprise, found nothing that looked like a loader routine or code that prepares the loading screen (Activision Logo and text). No, only this naughty stuff was lying around there:

---

<sup>2</sup> We do not want to start here with Adam and Eve but you will remember the specific noise of a c64 tape when played on a cassette recorder. There is this unique 10s synch sound and after that the so-called header block follows containing start and end address and the filename etc. Normally (when you shift-runstop or load"\*"), the load process is not under control of us. With sys63276 (\$F72C) you will be able to separately read the header from datasette. After the loading process, control is back in your hands and you can inspect the header information with your MLM and manipulate if necessary. Check out the episodes of Fairlight TV that deal with tape cracking.

# \$1000

1000	A9 00	LDA #\$00	
1002	8D 20 D0	STA \$D020	: black FRAME
1005	8D 11 D0	STA \$D011	: SCREEN OFF etc
1008	78	SEI	: Turn off interrupts
1009	08	PHP	: Processor status to STACK
100A	A9 10	LDA #\$10	: load \$10
100C	48	<b>PHA</b>	: to stack 1
100D	A9 FF	LDA #\$FF	: \$ff
100F	48	<b>PHA</b>	: to stack 2
1010	A9 2F	LDA #\$2F	: \$2f / 47
1012	85 00	STA \$00	: to zeropage \$00 (default=47)
1014	A5 01	LDA \$01	: load value from zp \$01
1016	09 02	ORA #\$02	: HIRAM - default 1
1018	85 01	STA \$01	: set HIRAM to default
101A	AD 2D 10	LDA \$102D	: load \$c9 (201) 11001001
101D	48	<b>PHA</b>	: to stack 3
101E	AD 2C 10	LDA \$102C	: load \$50
1021	48	<b>PHA</b>	: to stack 4
1022	AD 19 03	LDA \$0319	: load JumpVec.
1025	C9 E0	CMP #\$E0	: compare with e0 (why?)
1027	B0 05	BCS \$102E	: A >= E0?
1029	4C CA 0F	JMP \$0FCA	: true → 0FCA
102C	50 C9	BVC \$0FF7	: false(?) → 0ff7
102E	A9 CA	LDA #\$CA	: load \$CA (202)
1030	8D 18 03	STA \$0318	: in Jumpvek.
1033	A9 0F	LDA #\$0F	: load \$0f
1035	8D 19 03	STA \$0319	: 0FCA JumpVec. → kill memory!
1038	A9 00	LDA #\$00	: init. \$00fb free zp addy 1
103A	85 FB	STA \$FB	:
103C	A9 11	LDA #\$11	: \$11 in free zp addy 2
103E	85 FC	STA \$FC	:
1040	68	<b>PLA</b>	: pull from stack 1 (\$50)
1041	85 FD	STA \$FD	: store to free addy 3
1043	68	<b>PLA</b>	: pull C9
1044	85 FE	STA \$FE	: c9 to free addy 4
1046	A2 01	LDX #\$01	: init X=1
<b>1048</b>	<b>20 E8 0F</b>	<b>JSR \$0FE8</b>	: <b>DECRYPTION</b>
104B	60	RTS	: last 2 bytes from stack, i.e. \$10ff → jump to \$1100

Notes. FB/FC = \$1100

Let's summarize up to this point: The first file after the header contains a basic line: 10 SYS 4096. So, we start the MLM and look at \$1000. Why don't we see preparation of the screen etc.? Well, what we are confronted with here is a flipping *decryption routine* (see \$1048). This means that our Activision Loader program does not allow us to see the real interesting code without further ado. If we look further into the code (check e.g. \$1100-1120), only cryptic crap comes up that doesn't really make sense. Now we understand why. How do you recognize an encryption/decryption routine? Well, take a look at the subroutine called in the penultimate line (\$1048) from \$0FE8 onwards:

## \$0FE8

```

0FE8  A0 00      LDY #$00
0FEA  A5 FD      LDA $FD          : ($50?)
0FEC  18        CLC
0FED  65 FE      ADC $FE          : ($C9)
0FEF  85 FD      STA $FD          : =281 ($0119)
0FF1  B1 FB      LDA ($FB),Y      : $1100
0FF3  45 FD      EOR $FD          : DECRYPT
0FF5  91 FB      STA ($FB),Y      : $1100
0FF7  C8        INY
0FF8  D0 F0      BNE $0FEA
0FFA  E6 FC      INC $FC          : FC = $11 +1
0FFC  CA        DEX              : X defined in $1046
0FFD  D0 E9      BNE $0FE8
0FFF  60        RTS

```

The bold section loads bytes into the accumulator via an *indirectly indexed address* \$FB, scrambles this data with an EOR \$FD and then saves the decrypted data directly back into the same address! Ok, so far so good. But what do we do now? Well, it's getting a bit flimsy, I admit it. A somewhat unfair move towards the programmers will follow now.

We start the program (with RUN, SYS4096 or g \$1000), wait 1 second until the screen is built up (actually until the decryption is finished) and before the next loading process is started, we press freeze button (only for educational purposes). Now we can look at the decrypted parts of the loaded code at our leisure and see much more clearly. For example, the Activision screen can finally be localized:



**Figure 3.** Data from \$2000-\$4000 of the decrypted first data file visualized by INFILTRATOR V1.

How is the encryption routine working in detail you may ask? Well, it is quite easy. Get yourself a copy of *ppm-ii* and have a look at page 85:

```

C000 A6 FD LDX $FD      : Number of pages to process
C002 A0 00 LDY #$00     : Start with Zero
C004 B1 FB LDA ($FB),Y  : Load A indirect, indexed
C006 45 FE EOR $FE      : EOR A with content of $FE
C008 91 FB STA ($FB),Y  : Replace encrypted byte
C00A C8      INY        : Increase offset
C00B D0 F7 BNE $C004    : Repeat if not done with page
C00D E6 FC INC $FC      : Set pointer to next page
C00F CA      DEX        : Decrease # pages left to do
C010 D0 F2 BNE $C004    : Repeat if not done
C012 00      BRK        : Jump back to monitor

```

(Source: "Encrypt any" from *ppm-ii*, 1985)

With this little (and very similar looking) piece of code you can do the following. Produce a little bit of assembly code e.g. at \$C030

```

C030 A9 00      LDA #$00
C032 8D 20 D0    STA $D020
C035 8D 21 D0    STA $D021
C038 EA          NOP
C039 EA          NOP
C03A 4C 3A C0    JMP $C03A

```

Init 00fb onwards and start \$C000. Now look again at \$C030. Depending on your EOR value you will get something like this:

```

C030 D1 78      CMP ($78),Y
C032 F5 58      SBC $58,X
C034 A8          TAY
C035 F5 59      SBC $59,X
C037 A8          TAY
C038 92          ???
C039 92          ???
C03A 34          ???
C03B 42          ???
C03C B8          CLV

```

Ugly, eh? This is very similar to what the *Pitfall II* code is looking like from \$1100 onwards before you start the routine at \$1000. The good part of the story is that when you run again the “encrypt any” routine with the identical EOR value, you will get back to the original code – **it is reversible!** Technically, EOR’ed bit codes are 1 if they are different and 0 when they are not (e.g. 1010 1010 EOR 1111 0000 = 0101 1010). Here a comparison of OR vs. EOR truth tables

OR			EOR		
A	B	Q	A	B	Q
0	0	0	0	0	0
1	0	1	1	0	1
0	1	1	0	1	1
1	1	1	1	1	0

Ok, but now let’s go one major step and look at the decrypted stuff in more detail (bare in mind that we COULD just freeze after the full decryption and we are done). This is the content of 00fb on after init:

00fb 00 11 50 c9      than 0FE8 changes FD to      00fb 00 11 19 c9

The starting address for the decryption stuff is \$1100 and the EOR is \$19. And to make things even worse, EOR changes for every flipping byte. Urgh..... look at the loop again:

0FE8	A0 00	LDY #\$00	←
0FEA	A5 FD	LDA \$FD	
0FEC	18	CLC	
0FED	65 FE	ADC \$FE	
0FEF	85 FD	STA \$FD	
<b>0FF1</b>	<b>B1 FB</b>	<b>LDA (\$FB),Y</b>	
<b>0FF3</b>	<b>45 FD</b>	<b>EOR \$FD</b>	
<b>0FF5</b>	<b>91 FB</b>	<b>STA (\$FB),Y</b>	
0FF7	C8	INY	
0FF8	D0 F0	BNE \$0FEA	
0FFA	E6 FC	INC \$FC	
0FFC	CA	DEX	
0FFD	D0 E9	BNE \$0FE8	
0FFF	60	RTS	

Therefore, the first 3 bytes from \$1100 on, i.e. B4 CF BA, translate to AD D2 11. This in turn is an LDA \$112D. Let us check that before we go on by looking into the monitor when the loading screen appears. And voila! There it is: \$1100 AD 2D 11 LDA \$112D. The next decrypted opcode is a \$48 (PHA) and so on. Here the first 6 EORs – make sure that you understand how they are calculated: 19 E2 AB 74 3D 06. The good news is: You don't have to decrypt the code yourself (but it doesn't hurt to understand it). You can have the code decrypted and then view it with a “BRK” at an appropriate place (\$104b). This principle is important: Understand what is done when in the code and set BRK points at strategically important places to gain back control. Often RTS is a good place for a BRK.



## Gain Control

The art of cracking is to gain control over the processes in your C64. Software programmers try to block your ability to e.g. intervene in the loading process. If you simply load the game, you normally have no control over the process at all. If the program autostarts, then you are out of the game after you have pressed "Play". You must try to avoid this. What options do you have?

1) Use SYS 63276 and SYS 62828 to load the header file and the actual data file separately. This is mandatory if your program loads code into the autostart area. Remember this and try it out - but you don't need it for our first project *Pitfall II*. Here a simple LOAD is fine.

2) Set BREAKS at strategically important places in the machine code. The program then terminates in a controlled manner at the point you specify and gives you back control of the processes. For example, you can replace RTS with BRK to read out the next address to which the program jumps. That allows you to trace the code.

Now, that we know how EOR encryption/decryption works, let's look at some other important little things the \$1000 routine does. E.g. this nasty part which is called whenever an NMI is triggered (e.g. banging RESTORE).

## \$0FCA – Killer Code

0FCA	78	SEI	
0FCB	A9 FF	LDA #\$FF	
0FCD	8D 03 0C	STA \$0C03	
0FD0	A9 00	LDA #\$00	: prep. to destroy
0FD2	85 FB	STA \$FB	: from \$1000
0FD4	A9 10	LDA #\$10	
0FD6	85 FC	STA \$FC	
0FD8	A9 00	LDA #\$00	: overwrite with 00
0FDA	A8	TAY	
0FDB	91 FB	STA (\$FB),Y	: from \$1000 onwards
0FDD	E6 FB	INC \$FB	
0FDF	D0 FA	BNE \$0FDB	
0FE1	E6 FC	INC \$FC	: until end of memory
0FE3	D0 F6	BNE \$0FDB	
0FE5	18	CLC	
0FE6	90 E2	BCC \$0FCA	

# RTS/Stack Technique

Ok, after the whole flipping EOR wars are over and the encrypted code is successfully decrypted, how and why the hack is the program started?! After decryption we are going back to \$104B. Here we find a lonely RTS. But where are we jumping to now? We called this routine via SYS 4096 not via a JSR...: The RTS works perfectly well without any JSR, by simply accessing the stack (**\$01ff-\$0100**). This so-called “**stack technique**” can be used over and over to make tracing more difficult. As long as you keep careful track of the contents of the stack, however, you should be able to follow the program flow through any number of RTSs” (ppm-ii, 1985).

Remember what was placed to the stack before \$FD and \$FE: a causal \$10ff. RTS adds 1 to the address and happily hops on to \$1100. Before we go there, here a snippet for your understanding:

```
c000  A9 22      LDA #$22
c002  48        PHA           : push $22 to the stack
c003  A9 33      LDA #$32
c005  48        PHA           : push $32 to the stack
c006  60        RTS

2233  EE 21 D0    INC $D021
2236  4C 33 22    JMP $2233
```

The RTS jumps to \$2233 where a flashing routine is executed. **Note that RTS increases the lowbyte by 1.** Check the stack pointer (SP): it is pointing to the next free slot in the stack (i.e. SP+1 and SP+2 = last address little endian!). So, now that we got that we can look at \$1100 to further hack on *Pitfall II*.

# \$1100

```

1100 AD 2D 11 LDA $112D : $C6
1103 48 PHA : Push C6 to stack
1104 AD 15 03 LDA $0315 : Compare content of $0315 (ea?)
1107 C9 E0 CMP #$E0 : with $E0
1109 B0 03 BCS $110E : if bigger than $110E
110b 4C CA 0F JMP $0FCA : if not → kill memory!
110e AD 17 03 LDA $0317 : same with $0317
1111 C9 E0 CMP #$E0 : ACTION: Change to a lower number
1113 B0 03 BCS $1118
1115 4C CA 0F JMP $0FCA : if not → kill memory!
1118 AD 2C 11 LDA $112C : $1D
111b 48 PHA : to stack
111c A2 04 LDX #$04 : init X to $4
111e BD E2 11 LDA $11E2,X : load "CBM80" from $11E2 - ACTION!
1121 DD 04 80 CMP $8004,X : compare to $8004 (check for cartridge)
1124 D0 08 BNE $112E
1126 CA DEX
1127 10 F5 BPL $111E
1129 4C CA 0F JMP $0FCA : if cartridge is detected, kill memory!

112e A5 01 LDA $01
1130 29 FE AND #$FE
1132 85 01 STA $01
1134 68 PLA : $1D
1135 85 FD STA $FD : to $FD
1137 68 PLA : $C6
1138 85 FE STA $FE : to $FE
113a A9 7D LDA #$7D
113c 8D FF 01 STA $01FF : $7D to 01FF
113f A9 00 LDA #$00 : FB = 00
1141 85 FB STA $FB
1143 A9 1B LDA #$1B : FC = 1B
1145 85 FC STA $FC
1147 A2 04 LDX #$04
1149 20 E8 0F JSR $0FE8 : DECRYPT $1B00
114c A9 FF LDA #$FF : Set a BRK here
114e 8D FE 01 STA $01FE : store FF to 01FE
1151 AD 85 1E LDA $1E85 : $4F
1154 48 PHA : to stack
1155 20 00 1B JSR $1B00

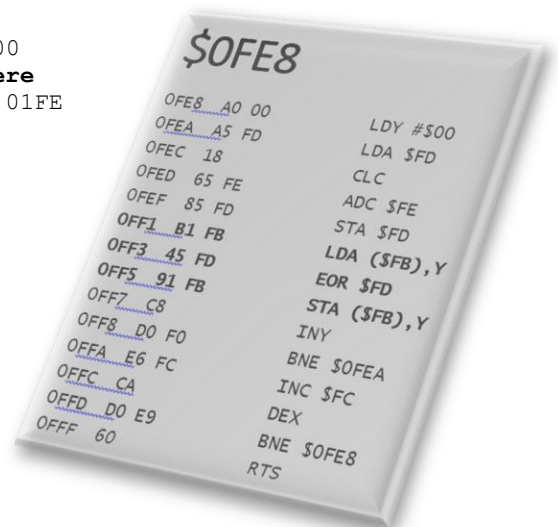
```

\$11E2 : c3 c2 cd 38 30 → CBM80

Change e.g. \$11E6 to \$31

Decrypting check

\$1b00 : 4A 80 → A9 29 LDA #\$29



## BRK-Vector & CBM80

If the game code detects that the break vector (\$0316/0317) is not default (i.e. \$FE66), it kills the code (see \$1111). If you have an Action Replay cartridge running the break vector will not be default (simply check that at \$0316/17 where you will find e.g. dfcd). Change the value in \$1112 to something that is lower than \$0317. Interestingly, my AR5 (and also the AR4.2) has a \$DF25 in the BRK vector- if you read \$0317 using LDA and save it somewhere, there is a "01" in it. For whatever reason, if you enter a "00" at \$1112 you are safe.

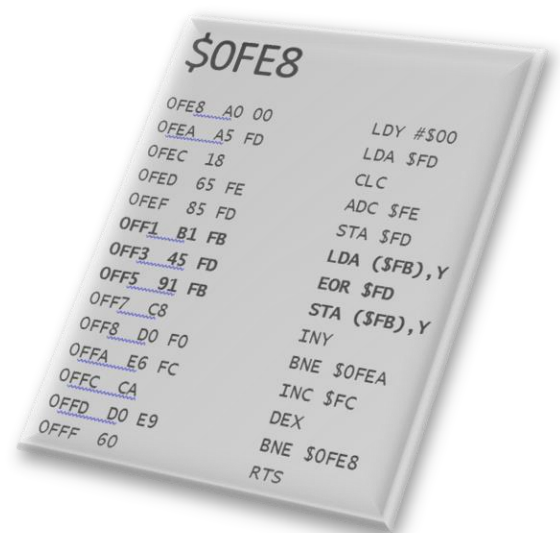
The CBM80 signature of your connected cartridge (at \$8004) will be checked at \$1121 and compared to \$11e2-11e6 (look at this with your MLM (type M 11E2 in your MLM and you will find "CBM80"). Simply change \$11e6 to 31 and all is fine. This is the first character that will be checked (see \$111c) and the routine will be discontinued if it is not equal to "0" (see \$1124). The CBM80 signature is routinely scanned for when the C64 is switch on (and also during reset procedure) and if found, it autostarts the cartridge stuff. *Pitfall II* wants to protect itself from a connected ML cartridge.

## \$1158 - now it's getting interesting!

```

1158 78          SEI
1159 A9 00       LDA #$00
115b AA         TAX
115c A8         TAY
115d A9 20       LDA #$20
115f 85 FB       STA $FB
1161 88         DEY
1162 D0 FD       BNE $1161
1164 CA         DEX
1165 D0 FA       BNE $1161
1167 C6 FB       DEC $FB
1169 D0 F6       BNE $1161
116b A5 01       LDA $01
116d 09 04       ORA #$04
116f 85 01       STA $01
1171 A9 00       LDA #$00
1173 8D 20 D0     STA $D020
1176 A9 20       LDA #$20
1178 8D 11 D0     STA $D011
117b A9 00       LDA #$00
117d 85 FB       STA $FB
117f A2 04       LDX #$04
1181 A9 08       LDA #$08
1183 85 FC       STA $FC
1185 A9 00       LDA #$00
1187 A0 00       LDY #$00
1189 91 FB       STA ($FB),Y
118b 88         DEY
118c D0 FB       BNE $1189
118e E6 FC       INC $FC
1190 CA         DEX
1191 D0 F4       BNE $1187
1193 AD 84 1E     LDA $1E84
1196 48         PHA
1197 A9 02       LDA #$02
1199 A2 01       LDX #$01
119b A0 01       LDY #$01
119d 20 BA FF     JSR $FFBA
11a0 A9 0C       LDA #$0C
11a2 A2 D6       LDX #$D6
11a4 A0 11       LDY #$11
11a6 20 BD FF     JSR $FFBD
11a9 A9 00       LDA #$00
11ab 20 D5 FF     JSR $FFD5
11ae A9 00       LDA #$00
11b0 8D 20 D0     STA $D020
11b3 AD 11 D0     LDA $D011
11b6 29 EF       AND #$EF
11b8 8D 11 D0     STA $D011
11bb 68         PLA

```



```

: A = length of filename
: filename at $11d6
: filename at $11d6

```

```

: (JMP $F4E9) load the data

```

ROM Routine FFD5: load RAM from a device. This routine will load data bytes from any input device directly into the memory of the computer.

```

11bc 85 FD      STA $FD
11be 68         PLA
11bf 85 FE      STA $FE
11c1 A9 00      LDA #$00
11c3 85 FB      STA $FB
11c5 A9 7E      LDA #$7E
11c7 85 FC      STA $FC
11c9 A2 12      LDX #$12
11cb 20 E8 0F   JSR $0FE8      : decrypt loaded data
11ce A2 FD      LDX #$FD
11d0 78         SEI
11d1 9A         TXS
11d2 D8         CLD
11d3 A2 00      LDX #$00
11d5 60         RTS      : good place for a BRK;
                          : look into stack ($7e00)

```

As described above without modern freezer MLM functionality you need to take care of screen mode (\$D011) and screen/char RAM (\$D018). They are modified in code section \$1B00 (**see excerpt**) and \$1158 where the loading screen is set up. To modify the bits in \$1B00, we need to add an extra BRK after \$1B00 has been decrypted (e.g. \$114C). Now, we change the particular STAs \$D018/\$D011 to LDAs and we are done. Our BRKs will give us now control back even with a software MLM of 1984.

## \$1B00 (excerpt)

```

1b00 A9 29      LDA #$29      : 00101001 → Scr. Mem. to $0800
1b02 8D 18      STA $D018      : Change $1B02 to AD
...
1db6 A9 3B      LDA #$3B      : 00111011 to D011 = HIRES on;
1db8 8D 11 D0   STA $D011      : SCR on; 25 lines; Change $1B02
to AD
1dbb 60         RTS      : Back to $1158

```

We have now analyzed the first file of *Pitfall II* sufficiently and we should finally get down to the real job: The cracking! Let's get started Dude! Simply follow the instructions. If you don't understand something, look again in the code descriptions above.

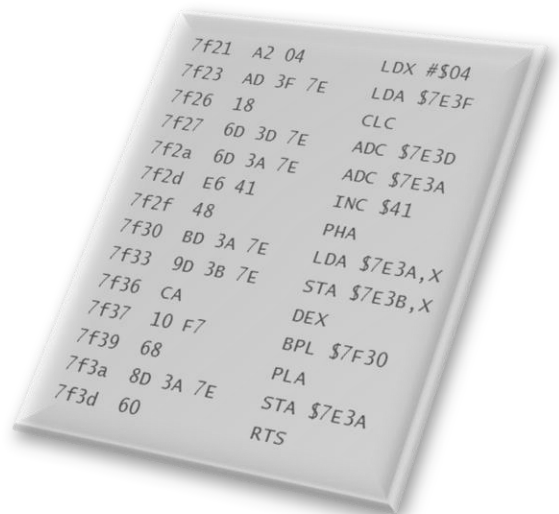
- 00) Start C64 with an active AR4.2 + datasette + nothing else
- 01) Put in the original *Pitfall II*, type Load + Return, Press Play
- 02) When loading is done → type MON
- 03) \$104b RTS → BRK (get back control)
- 04) G 1000 (let the C64 decrypt the code from \$1100 on)
- 05) D 1111: CMP #\$E0 → CMP #\$00
- 06) M 11E6: 30 → 31 (hide your cartridge)
- 07) A 114C BRK
- 08) G 1100 (let the C64 decrypt the code at \$1B00)
- 09) A 114C LDA#\$FF (set back to the original entry)
- 10) D 1B02: STA → LDA
- 11) D 1DB8: STA → LDA
- 12) D 1178: STA → LDA
- 13) D 11B8: STA → LDA
- 14) D 11D5: RTS → BRK (get back control)
- 15) G 114C [after ~6 few seconds loading continues...]
- 16) Make coffee
- 17) D 11D5: BRK → RTS [not necessarily needed]
- 18) If start address is known (see below) goto Step #59**

Now we have to take care of the stack (Mommy AR6 is no longer there). Check the stack (M 01FE), note the next address (\$7E00), assemble and execute a 4-liner (“stack cleaner” and “X-register saver”), replace the RTS with a BRK in the upcoming code section and go ahead.



## \$7E00 - 2nd data bunch

7e00	A9 7F	LDA #\$7F	
7e02	8D 19 03	STA \$0319	
7e05	A9 6F	LDA #\$6F	
7e07	8D 18 03	STA \$0318	: NMI vector to \$7f6f
7e0a	A9 59	LDA #\$59	
7e0c	85 3B	STA \$3B	
7e0e	18	CLC	
7e0f	69 E8	ADC #\$E8	
7e11	AA	TAX	
7e12	A9 7E	LDA #\$7E	
7e14	85 3C	STA \$3C	
7e16	69 FF	ADC #\$FF	
7e18	48	PHA	
7e19	8A	TXA	
7e1a	48	PHA	
7e1b	A9 7F	LDA #\$7F	
7e1d	85 8C	STA \$8C	
7e1f	A9 01	LDA #\$01	
7e21	85 8B	STA \$8B	
7e23	A2 3B	LDX #\$3B	
7e25	A0 02	LDY #\$02	
7e27	B5 00	LDA \$00,X	
7e29	95 FE	STA \$FE,X	
7e2b	E8	INX	
7e2c	88	DEY	
7e2d	D0 F8	BNE \$7E27	
7e2f	84 41	STY \$41	
7e31	E6 39	INC \$39	
7e33	D0 02	BNE \$7E37	
7e35	E6 3A	INC \$3A	
7e37	A0 3C	LDY #\$3C	
<b>7e39</b>	<b>60</b>	<b>RTS</b>	



## \$7E00 - Decryption

7e42	20 21 7F	JSR \$7F21	
7e45	51 39	EOR (\$39),Y	
7e47	91 39	STA (\$39),Y	
7e49	20 21 7F	JSR \$7F21	
7e4c	51 3B	EOR (\$3B),Y	
7e4e	88	DEY	
7e4f	D0 F6	BNE \$7E47	
7e51	91 39	STA (\$39),Y	
7e53	A5 3C	LDA \$3C	
7e55	48	PHA	
7e56	A5 3B	LDA \$3B	
7e58	48	PHA	
<b>7e59</b>	<b>60</b>	<b>RTS</b>	

If everything has worked out so far: Congratulations. Now let's work through the second pile of data. Continue cracking...

19)	M 01FE (FF 7D)	: check stack
20)	Assemble: A C000 LDX#\$FF	: 4-liner that
21)	C002 TXS	: clears the stack
22)	C003 LDX#\$00	: and sets back x
23)	C005 BRK	: to its original value
24)	G C000	: exec. C000
25)	D \$7E00 \$7E39	: list the next code section
26)	\$7E39 RTS → BRK	: replace RTS with BRK
27)	G 7E00	: exec. 7E000
28)	\$7E39: BRK → RTS	: replace BRK with RTS
29)	M 01FC (41 7E)	: check stack
30)	Modify C003 (X=3D) & G C000	: set X and exec. C000
31)	D 7E42 7E5A	: list next code section
32)	\$7E59: RTS → BRK	: ...
33)	G 7E42	:
34)	\$7E59: BRK → RTS	:
35)	M 01FC (59 7E)	:
36)	Modify C003 (X = FF) & G C000	:

In the next code section (\$7E5A) we have to modify 3 locations because our AR4.2 puts a "08" to \$00BA (you don't need to do this when you are using a software MLM. Here \$00BA should be 01 as long as no floppy is active – check it!). \$00BA is the Current Device Number (01 = Datasette, 08 = Floppy). *Pitfall II* expects "01" here (me too!), but we can deal with this...

## \$7E5A

7e5a	BA	TSX	: Stack pointer to X (FF)
7e5b	CA	DEX	: FE
7e5c	86 B9	STX \$B9	: FE
7e5e	CA	DEX	: FD
7e5f	A4 BA	LDY \$BA	: → Change to LDY #\$01
7e61	B9 E6 7E	LDA \$7EE6,Y	
7e64	D0 07	BNE \$7E6D	
7e66	B9 00 00	LDA \$0000,Y	
7e69	29 10	AND #\$10	
7e6b	F0 03	BEQ \$7E70	
7e6d	6C 8B 00	JMP (\$008B)	
7e70	9A	TXS	: FD
7e71	A9 7E	LDA #\$7E	
7e73	91 B9	STA (\$B9),Y	: → MSB to stack \$FF: : Change to PHA and 1 NOP
7e75	88	DEY	
7e76	A9 41	LDA #\$41	
7e78	91 B9	STA (\$B9),Y	: → LSB to stack \$FE: : Change to PHA and 1 NOP
7e7a	A5 39	LDA \$39	
7e7c	18	CLC	
7e7d	69 3D	ADC #\$3D	
7e7f	85 39	STA \$39	
7e81	A5 3A	LDA \$3A	
7e83	69 00	ADC #\$00	
7e85	85 3A	STA \$3A	
7e87	A5 3B	LDA \$3B	
7e89	18	CLC	
7e8a	69 3D	ADC #\$3D	
7e8c	85 3B	STA \$3B	
7e8e	A5 3C	LDA \$3C	
7e90	69 00	ADC #\$00	
7e92	85 3C	STA \$3C	
7e94	A0 4D	LDY #\$4D	
7e96	60	RTS	: → Change to BRK

Continue cracking...

- 37) G 7E5A
- 38) \$7E96: BRK → RTS
- 39) M 01Fa (41 7E)
- 40) Modify C003 (X is now FB) and G C000
- 41) D 7E42 7E5A
- 42) \$7E59 RTS → BRK

- 43) G 7E42
- 44) \$7E59 BRK → RTS : **Start address known → Goto step 52**
- 45) M 01FC (96 7E)
- 46) Modify C003 (X is now FF) and G C000
- 47) D 7E97 7EE5
- 48) \$7EE4 RTS → BRK
- 49) G 7E97
- 50) M 01FA → We see kernel address (E6 FF) and (9E 7E)

The section \$7E9F-7EE4 crazily calls now dozens of kernal routines and starts again at \$7E9F. The purpose of this is to find out whether we a floppy drive is connected. If so, it crashes. With AR4.2 or software MLM we need to assemble one “kernal routine caller”:

- 51) Assemble: C006 JSR \$FFE7
- 52) C009 BRK
- 53) G C006
- 54) Modify C003 (X is now 03) and G C000
- 55) \$7EBD: JMP \$7E21 → BRK
- 56) \$7EE4: BRK → RTS
- 57) G 7E9F
- 58) Read the stack (M 01FC) and find \$86B4
- 59) \$86F7: RTS → BRK
- 60) G 86B4
- 61) Read the stack (M 01FC) and find \$8009
- 62) s”pitfall ii crack”,08,8009,c000
- 63) g 8009 and the game starts – this is the starting address we have looked for after all checks have been done.

# \$7E9F

: within the next section the **peripheral check** is performed via 44 loops from \$7EE4-to Kernal and back

7e9f	A4 3F	LDY \$3F	: current DATA Line number	←
7ea1	88	DEY	: =03, 02	
7ea2	10 1C	BPL \$7EC0	: if result = pos. in trace mode this is taken!	
7ea4	29 80	AND #\$80		
7ea6	D0 03	BNE \$7EAB		
7ea8	6C 8B 00	JMP (\$008B)		
7eab	A6 3D	LDX \$3D		
7ead	CA	DEX		
7eae	10 E9	BPL \$7E99		
7eb0	A5 8B	LDA \$8B		
7eb2	18	CLC		
7eb3	69 B2	ADC #\$B2		
7eb5	AA	TAX		
7eb6	A5 8C	LDA \$8C		
7eb8	69 07	ADC #\$07		
7eba	48	PHA		
7ebb	8A	TXA		
7ebc	48	PHA		
7ebd	4C 21 7F	<b>JMP \$7F21</b>		
<b>7ec0</b>	<b>84 3F</b>	<b>STY \$3F</b>	: \$3f = 03, 02	
7ec2	BE E5 7E	LDX \$7EE5,Y	: y = 03,02, X=7e,00	
7ec5	D0 06	BNE \$7ECD		
7ec7	A9 0F	LDA #\$0F		
7ec9	38	SEC		
7eca	E5 3D	SBC \$3D	: 07	
7ecc	AA	TAX	: A=08, X=08	
7ecd	A9 7E	LDA #\$7E	: Prepare the RTS Jumpback from Kernal	
to \$7e9f				
7ecf	48	PHA		
7ed0	A9 9E	LDA #\$9E	: Prepare the RTS Jumpback from Kernal	
to \$7e9f				
7ed2	48	PHA		
7ed3	B9 EA 7E	LDA \$7EEA,Y	: Prepare KERNAL Routine RTS Jump (ff)	
7ed6	48	PHA		
7ed7	B9 F9 7E	LDA \$7EF9,Y	: Prepare KERNAL Routine RTS Jump (bc)	
7eda	48	PHA		
7edb	B9 F4 7E	LDA \$7EF4,Y	: 03	
7ede	48	PHA		
7edf	B9 EF 7E	LDA \$7EEF,Y	: fe	
7ee2	A8	TAY	: y=fe	
7ee3	68	PLA	: 03 (A)	
7ee4	60	RTS	: \$ffe7 (STACK: 01fc e6 ff 9e 7e)	

**which kernal routines are called?**

Lines \$7ed3 and \$7ed7 define the Kernal routine addresses:

Ffb7, ffc0, ffba, ffbd and ffe7

## Final walk through this section

7e9f	A4 3F	LDY \$3F	: 00	
7ea1	88	DEY	: FF	
7ea2	10 1C	BPL \$7EC0	:	
7ea4	29 80	AND #\$80	:	
7ea6	D0 03	BNE \$7EAB	:	
7ea8	6C 8B 00	JMP (\$008B)	:	
7eab	A6 3D	LDX \$3D	: 00	
7ead	CA	DEX	: FF	
7eae	10 E9	BPL \$7E99	:	
7eb0	A5 8B	LDA \$8B	: A: 01	
7eb2	18	CLC	:	
7eb3	69 B2	ADC #\$B2	: A: B3	
7eb5	AA	TAX	: X: B3	
7eb6	A5 8C	LDA \$8C	: A: 7F	
7eb8	69 07	ADC #\$07	: A: 86	
7eba	48	PHA	: 86	} \$86b4
7ebb	8A	TXA	:	
7ebc	48	PHA	: B3	
7ebd	4C 21 7F	JMP \$7F21		

Finally, we have \$86b3 in the stack which RTS's to \$86b4. Phew...!!

7f21	A2 04	LDX #\$04	
7f23	AD 3F 7E	LDA \$7E3F	
7f26	18	CLC	
7f27	6D 3D 7E	ADC \$7E3D	
7f2a	6D 3A 7E	ADC \$7E3A	
7f2d	E6 41	INC \$41	
7f2f	48	PHA	
7f30	BD 3A 7E	LDA \$7E3A, X	
7f33	9D 3B 7E	STA \$7E3B, X	
7f36	CA	DEX	
7f37	10 F7	BPL \$7F30	
7f39	68	PLA	
7f3a	8D 3A 7E	STA \$7E3A	
7f3d	60	RTS	: to \$86b4

Note: Before I found out that you can set a BRK at position \$7EBD, I went manually through the 44 (or however many there really are) loops step by step by alternately setting 2 BRKs at position \$7EE4 and \$7E9F. Then I did this until the game started and counted the repetitions and started all again. In the last step before starting the game, I then read out the stack. After that I found out that this is unnecessary if you set a BRK at \$7EBD.

We have overcome all protection checks and were able to follow the course of the code step by step in a controlled manner. We have identified the starting address (\$8009). Equipped with the start address, we reset the C64. Now we connect and fire up our floppy drive and insert a blank disk. We now do everything again until the 2nd decryption (\$7E42) has run (Step #43). Then save (\$8009-\$C000) to disk and you are done (s"pitfall ii crack",08,8009,c000). Reset your computer one really last time. Load the cracked *Pitfall II* file from the floppy and SYS32777 (or in MLM g 8009). Open the beer and start playing. Exomize it and spread it in the schoolyards and send it to your dudes (don't forget to cover the stamps with hairspray...)!

Ah, navigating through the intricacies of code, tinkering with the BRK vector query and the CBM80 identifier query, and braving the perilous RTS/stack wilderness. It's abundantly clear that the programmer's intention was to test your resolve, to push you to your limits. Especially if you go through the periphery detection loop step by step, there is a high probability that you will abort the entire project. After 10, maybe 20 exhaustive traces, they probably expected you to grumble, "Oh, darn

it... I must have goofed somewhere," and by the 30th trace, to exclaim, "Geez... this feels like going in circles!" But they messed with the wrong one!

So Dude, here is the summary of our efforts in the form of a step-by-step guide. If you do everything right, you'll end up with a perfect crack of Pitfall II.

- 01) Start C64 with an active AR4.2 + datasette + nothing else
- 02) Put in the original *Pitfall II*, type Load + Return, Press Play
- 03) When loading is done → type MON
- 04) \$104b RTS → BRK (get back control)
- 05) G 1000 (let the C64 decrypt the code)
- 06) D 1111: CMP #\$E0 → CMP #\$00
- 07) M 11E6: 30 → 31 (hide your cartridge)
- 08) A 114C BRK
- 09) G 1100 (let the C64 decrypt the code at \$1B00)
- 10) A 114C LDA#\$FF (set back to the original entry)
- 11) D 1B02: STA → LDA
- 12) D 1DB8: STA → LDA
- 13) D 1178: STA → LDA
- 14) D 11B8: STA → LDA
- 15) D 11D5: RTS → BRK (get back control)
- 16) G 114C [after ~6 few seconds, loading continues]
- 17) Make coffee
- 18) D 11D5: BRK → RTS [not necessarily needed]
- 19) If start address is known goto Step #60!**
- 20) M 01FE (FF 7D) : check stack
- 21) Assemble: C000 LDX#\$FF : 4-liner that
- 22) C002 TXS : clears the stack
- 23) C003 LDX#\$00 : and sets back x
- 24) C005 BRK : to its original value
- 25) G C000 : exec. C000
- 26) D \$7E00 \$7E39 : list the next code section
- 27) \$7E39 RTS → BRK : replace RTS with BRK
- 28) G 7E00 : exec. 7E000
- 29) \$7E39: BRK → RTS : replace BRK with RTS



30) M 01FC (41 7E) : check stack  
 31) Modify C003 (X=3D) & G C000 : set X and exec. C000  
 32) D 7E42 7E5A : list next code section  
 33) \$7E59: RTS → BRK : ...  
 34) G 7E42  
 35) \$7E59: BRK → RTS  
 36) M 01FC (59 7E)  
 37) Modify C003 (X = FF) & G C000  
 38) G 7E5A  
 39) \$7E96: BRK → RTS  
 40) M 01Fa (41 7E)  
 41) Modify C003 (X is now FB) and G C000  
 42) D 7E42 7E5A  
 43) \$7E59 RTS → BRK  
 44) G 7E42  
 45) \$7E59 BRK → RTS  
 46) M 01FC (96 7E)  
 47) Modify C003 (X is now FF) and G C000  
 48) D 7E97 7EE5  
 49) \$7EE4 RTS → BRK  
 50) G 7E97  
 51) M 01FA → We see kernel address (E6 FF) and (9E 7E)  
 52) Assemble: C006 JSR \$FFE7  
 53) C009 BRK  
 54) G C006  
 55) Modify C003 (X is now 03) and G C000  
 56) \$7EBD: JMP \$7E21 → BRK  
 57) \$7EE4: BRK → RTS  
 58) G 7E9F  
 59) Read the stack (M 01FC) and find \$86B4  
 60) \$86F7: RTS → BRK  
 61) G 86B4  
 62) Read the stack (M 01FC) and find \$8009  
 63) s"pitfall ii crack",08,8009,c000  
 64) g 8009 and the game starts – this is the starting address  
 we have looked for after all checks have been done  
 65) Open a beer.

# TRAINING

Hey, hold on a second... In all our excitement about our awesome Pitfall II crack, did we forget something important? Oh, right, there's one more thing: Can we be sure that the crack works just like the original? Can it be played through without any nasty side effects? We've all heard the stories about sneaky programmers who embedded a tricky routine that activates late in the game, preventing the rewarding end screen from appearing. There are plenty of horror stories about *Cauldron*: enemies supposedly become super tough, and you unfairly lose a lot more points when you touch them. All of this for one reason: to make sure playing the original is a huge advantage, and that cracks are just poor, unenjoyable versions that don't work right. Honestly, it's not a bad strategy! The original is cool, the cracks are cheap junk!

To cut to the chase, as a fresh new cracker, you need to ensure you have a 100% version—or better yet, a 101% version. Okay, cool—we're on it! But how? The easiest way is to play through *Pitfall II*. It shouldn't take more than 15 minutes—but you have to be really careful not to fall into a pit and lose points, or get stuck on a scorpion. There are many

reasons why it might end up taking more than 15 minutes, possibly even a few days depending on your schedule. So, let's make it easier on ourselves and add a trainer (think "101% version")!

Now, seasoned Pitfall players might say, "Why add a trainer? You have unlimited lives!" That's true. But when you touch an enemy, you get sent back to the last red cross and lose precious points. The solution? An *invincibility trainer*. This way, we become invincible and breeze through the levels. The only things that can cost us points are falls and hard landings. But we can handle that, right?

Looking through the literature, we find enticing titles like "*Crack and Train like a Pro*" or excellent forum introductions by Wanderer. They tell us roughly what to do, but just when things get interesting, they stop, leaving it up to us. Fair enough, that's how it goes.

I spent days scrolling through the code, trying to figure out where to hook in to make our Harry invincible. A first (bad!) solution comes up fairly quickly. We C64 enthusiasts know there's a sprite-sprite collision register at \$D01E. So, start the MLM and happily hunt for it. And: found it! At address \$9818

```
9818  AD 1E D0      LDA $D01E
981b  95 93        STA $93,X
981d  60           RTS
```

We notice that this subroutine reads from D01E and writes to \$93,X. My first thought was, why not just boldly turn the whole thing off (you can guess where this is going, right?). So let's just do this:

```
9818  AD 1E D0      LDA $D01E
981b  95 93        LDA $93,X
981d  60           RTS
```

We fire up the modified game and test out the bat in the second screen. We stroll to the center of the second screen and wait for the bat to disappear off the right side of the screen, only to reappear on the left. The deadly bat flutters towards us, silent and swift. Just a few pixels away, and it's about to touch us. And then, it happens—whoosh! The critter flutters right through us without a peep. WOW. Let's try the vulture next. We head over to Screen No. 3 and walk right into the vulture... with the same effect! The bird sails through us without a fuss!! We're truly invincible. What an amazing feeling. Okay, with a spring in our step, we dive into the water after the vulture and swim to Screen 4 to claim our first well-deserved gold bar. Uh... What's this? Harry swims right through the gold bar. A second attempt ends with the clear realization that turning off collision detection also makes it impossible to collect "good" sprites. That means no gold and especially no balloon ride. So, our first naive "trainer" has effectively ruined the game and made it unplayable. Reset and back to square one.

What followed were weeks of code-searching orgies. Hours spent poring over code snippets, tracking attempts, freezing on collision, post-collision, comparing before and after... all without success. It should be so simple: somewhere in the code, there must be a straightforward distinction like: If Harry collides with bat, then warp back. If Harry collides with gold, then bonus points! This distinction

must be findable... or so I thought. But honestly, after a few days, I threw in the towel and did something sacrilegious (don't tell anyone). I grabbed the REMEMBER version of *Pitfall II* and froze at the invincibility trainer check. And presto, there it was:

```

1072  20 E4 FF      JSR $FFE4
1075  C9 4E          CMP #$4E
1077  F0 09          BEQ $1082
1079  C9 59          CMP #$59
107b  D0 F5          BNE $1072
107d  A9 A5          LDA #$A5
107f  8D B9 9A      STA $9AB9
1082  A9 00          LDA #$00
1084  20 36 E5      JSR $E536
1087  A2 00          LDX #$00
1089  8A            TXA
108a  9D 00 D8      STA $D800,X
108d  9D 00 D9      STA $D900,X
1090  9D 00 DA      STA $DA00,X
1093  9D 00 DB      STA $DB00,X
1096  E8            INX
1097  D0 F1          BNE $108A
1099  4C 09 80      JMP $8009

```

This small but mighty modification turns STA \$A1...

```

9ab7  A9 98          LDA #$98
9ab9  85 A1          STA $A1
9abb  B5 94          LDA $94,X
9abd  29 60          AND #$60
9abf  C9 60          CMP #$60

```

...into an LDA \$A1—and everything's peachy! Let's give it a whirl: it works like a charm. Enemies can't touch us, and we can collect all the gold and other bonus sprites, ending the game with 199000 points. Harry jumps for joy, and we're over the moon! We lean back in our chair in front of the computer, feeling like champions, mentally patting ourselves on the back. Well done, you old pro! But lurking in the back

of our minds is that familiar doubt, uncertainty, the buzzkill. And it softly whispers the same simple question over and over again: "Why?"

To answer this straightforward yet crucial question, we need to once again dive deep into the code...

After weeks of further digging, I had some insights: *Pitfall II* utilizes the zero page for storing crucial game parameters. For example, \$45-\$47 holds the score (which is initially stored in \$9CBC-9CBE), and \$A1 holds Harry's status. A value of 84/85 indicates "moving right/left"; 80/81 means "facing right/left while standing"; 06/07 signifies "jumping right/left"; 88/89 represents "swimming right/left"; 8E stands for "hard landing after falling"; and a "98" denotes "enemy contact and death". Writing \$98 to \$A1 happens at \$9ab7 or \$9ab9. Preventing this write operation results in the desired invincible status. My realization regarding the zero page naturally tempted me to manipulate the values and analyze the consequences.

\$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	2f	36	80	00	a9	01	00	01	f4	00	e0	1f	01	7d	bf	02
0010	60	01	ff	ff	ff	28	00	00	00	31	7f	03	01	01	08	00
0020	1d	16	1c	15	18	14	0f	aa	c0	00	00	00	5a	50	00	00
0030	00	00	00	00	60	10	06	02	00	00	00	19	08	03	5a	00
0040	2e	20	00	00	a9	00	40	00	00	00	01	00	00	00	00	00
0050	00	00	00	08	cf	7e	f7	7a	00	00	00	00	00	00	00	03
0060	0b	02	03	07	02	22	86	5a	44	9f	50	a0	a4	9d	04	00
0070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0090	00	00	00	00	05	00	00	00	00	00	00	82	c6	c7	82	00
00A0	40	80	20	88	00	00	00	00	00	00	00	00	00	00	00	00
00B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

\$19: x-coordinate Harry; \$1A: y-coordinate Harry

00	60	10	06	02
----	----	----	----	----

Table X: Sprites at \$0033-\$0036

Nibble 1	Nibble 2
1x = Quickclaw	x1 = Scorpion
2x = Rhonda	x2 = Bat
3x = Gold left	x3 = Balloon
4x = Gold right	x4 = Rat
5x = Raj	x5 = Condor
6x = Cross	x6 = Frog
7x = Waterfall	x7 = Eel

82	c6	c7	82
----	----	----	----

Called by: [to find out]

## Secret path found after 40 years...

How cool would that be, right? Every Pitfall fan knows about the legendary Easter egg level on the Atari. Unfortunately, us C64 users miss out on that. It's a bummer, but hey, it is what it is. But imagine how awesome it would be if there were a secret passage. As a kid, I spent hours trying to reach that rat. I searched underwater for secret passages, but sadly, found nothing. However, recently, there was finally a breakthrough. Check this out:



*Secret path found after 40 years...* 🤔



You can simply reach Quickclaw from the starting level by pressing the joystick down directly above the nonexistent ladder. Harry will climb up the invisible ladder, allowing you to collect Quickclaw and even grab the rat from behind. Why has no one ever discovered this before? Well, because this secret path doesn't actually exist. At least not without our intervention. But with our knowledge of the relevant memory locations in the zero page, it's a minimally invasive operation. With a little "new school" trickery, I traced the writing to \$9b and found the \$969b, where A9E5 is loaded. This is where we install our invisible ladder!

Go ahead and give it a try yourself:

```
$A9E5 C3 (instead of 82)
```

Have fun tricking your buddies with a "Hey, did you know about this trick?" They'll be amazed. 😊

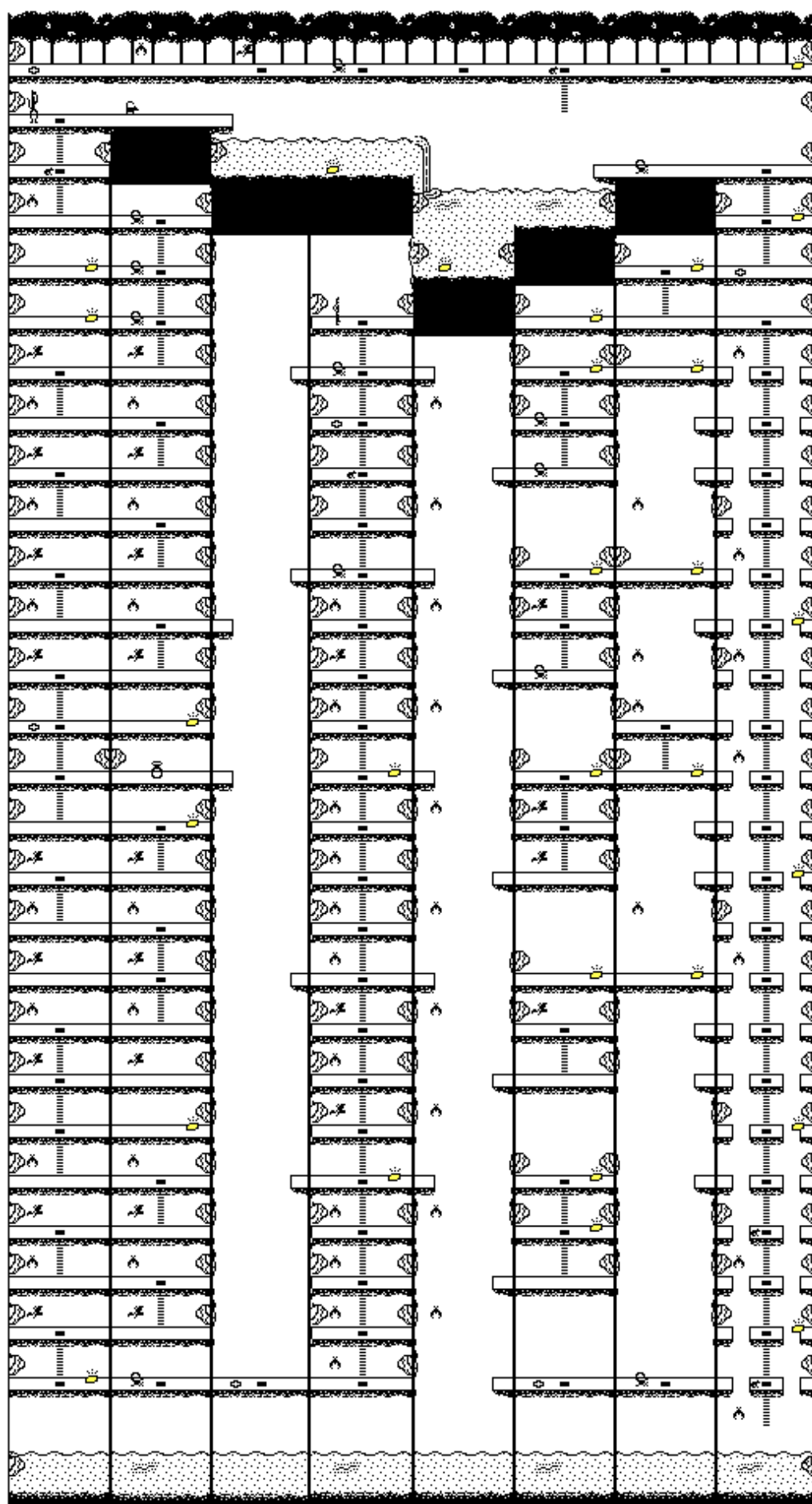
## Changing the Screen

What you see on the screen is divided into different levels. The bottom layer is in the range \$1300-\$140F. If you fill this with \$00, for example, you have deleted it completely. Try this out.

If you want to 'build' a ladder that will take you from your starting level to Quickclaw, you can do the following:

```
$1423 06 07  
$1451 06 07  
$1473 06 07  
$14A1 06 07
```

This makes you a nice ladder - but you still have to make the entry possible. You do that in 009b.



*"The path to knowledge is a rewarding one."*  
K.J. REVEALED TRILOGY

# REFERENCES

Berg, P. (1996) *Software analysis and reconstructive therapy, a historical view on "cracking"*. disC=overy, Issue 1

Berg, P. (2016). *Which was the best cracking cartridge on the Commodore 64?*  
<https://bergatrollet.se/blog/2016/01/c64-cartridges/>

Gelfand, R., Felt, J., Strauch, M., Krsnik, D. (1988) *Das Anti-Cracker Buch*. DATA BECKER, Düsseldorf

Kracker Jax Protection Busters (1990) *The Kracker Jax Revealed Trilogy*. Kracker Jax Protection Busters

Morrow S. (2019) *Pitfall II Game Reverse Engineered*. [www.c64brain.com](http://www.c64brain.com)

Mr. Nop (2015) *Crack and Train like a Pro*. <https://csdb.dk/release/?id=139238>

QED (1997) *Defeating Digital Corrosion (aka "cracking") - a beginners guide to software archiving*. disC=overy, Issue 3

Simstad, T.N. (1984) *Program Protection Manual for the C-64 Volume I*.

Simstad, T.N. (1985) *Program Protection Manual for the C-64 Volume II*. CSM Software, Inc.

SLC (2010) *Copy Protection on Tape*. Recollection issue 3

Wanderer (2006) *Beginner Tutorial - Training games for infinite lives*.  
<https://www.lemon64.com/forum/viewtopic.php?t=20313>

**MANUAL**



We were all pretty worried about Pitfall Harry. -  
We sent him and his niece, Rhonda, along with  
Quickclaw the cat, on a treacherous journey to  
an underground cavern. Well, not a word was  
heard from him—until today. We now present  
you with Pitfall Harry's diary—the journal he's  
been keeping in the lost caverns. It arrived this  
morning by carrier condor. Typical of Harry.

Before you start reading, let's get your gear together. Here's the basic set-up:

- **Insert cartridge** into your game system with the power OFF. Then, turn power ON.
- **Plug in the left Joystick Controller only.** This is a solo expedition.
- **The difficulty and game select switches are not used.**
- **To begin a new game,** press the reset switch.
- **To start the action,** move the Joystick.
- **The Joystick Controller** does many things. Certain maneuvers really require practice.
  - \* To move Pitfall Harry left or right, move the Joystick left or right.
  - \* To jump, press the red button. For a running jump, press the red button while holding the Joystick left or right.
  - \* To descend a ladder, pull Joystick back **just before** Pitfall Harry reaches the hole. To ascend a ladder, push Joystick forward.
  - \* When Pitfall Harry travels by balloon (see 'Balloons') the balloon will follow the left and right movements of the Joystick. To speed it up, push Joystick forward; to slow it down, pull Joystick back.
- **There is no time limit.** You and Pitfall Harry can explore the lost caverns as long as you wish.
- **The journey ends** the moment Rhonda, the Raj diamond and Quickclaw have all been found.

**And now...the diary.**



Lost Cavern,  
Machu Pichu, Peru

Lat.  $13.31^{\circ}\text{S}$ , Long.  $71.59^{\circ}\text{W}$

### IN CASE YOU'RE WONDERING WHY I'M HERE

Perhaps I've gone too far. I'm in an underground cavern beneath Peru. It seems to be a complex maze, perhaps eight chambers wide and over three times as deep. Niece Rhonda has disappeared, along with Quickclaw, our cowardly cat. I am beset by all manner of subterranean creatures in this vast, ancient labyrinth. And all because of a rock -- the Raj diamond. It was stolen a century ago, and hidden here.

Old friends, if ever you see this diary, I hope you'll read it and come to my aid. Help me find Rhonda, Quickclaw and the diamond. On the way, let's also look for a stone-aged rat. A large university wants it for research. Finally, lots of stolen U.S. gold bars were ditched here. The more bars we recover, the more "brownie" points we'll get at journey's end.

What to  
get:



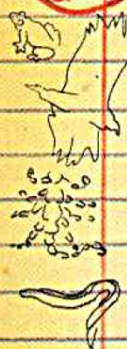
For the record, if I want a perfect evaluation (199,000 points), I must find Rhonda, Quickclaw, the diamond, all 28 gold bars, and the rat. And never fall victim to a single danger. Again though, my contract only requires the recovery of



Rhonda, Quickclaw and the diamond.

Everything else is gravy.

What to avoid:



But oh... danger prevails. Poisonous frogs, bats, condors, electric eels, albino scorpions. And leaps over dark voids that dare me to fall to their fathomless depths. All of these pitfalls must be avoided. I'm not really sure what'll happen, should I succumb, but it can't be good.

I'll check it out tomorrow.



### NEXT DAY: THE CONSEQUENCES



Red Crosses are as good as gold-- go for them! They'll never believe this when I get back but, it's like this: Whenever I succumb to any danger, I'm not put out of commission, as one might think. Instead, I'm magically transported back to the last red cross I touched!

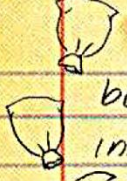


Here's my theory: These caverns are part of a land long inhabited by the Incas. This great civilization must have energized healing centers, and these ancient crosses mark their locations. Their magic is still potent.

### BALLOONS!!??

Today I was really out on a ledge. Suddenly, a balloon floated overhead-- a perfect cross-cavern transport.




 I wasn't about to ask questions, but I was curious. Upon closer inspection, I discovered that these "balloons" were actually the specimen bags Rhonda was carrying. She must've inflated them with steam from an underground geyser, to signal that she was still alive. Dear, ever-resourceful Rhonda.

Anyway, just jump up and hang on. Float above the ledge you're headed to and wait for a bat to burst the balloon (that's the only way to let go). You'll gently fall onto the ledge. Careful now. You want the bat to get the balloon. Not you.

Oh yes -- Balloons only appear up and down the length of one, specific shaft.

## GREAT LEAPS IN FAITH

 Courage and confidence may be necessary to leap across fathomless voids but, believe me, you can do it!

When you want to cross a shaft, especially the one with no balloons, stand at a ledge's edge and jump diagonally down to a ledge across the way. Press the red button right before you jump and hold it down as you move the joystick in the direction you're heading. You'll float to the other side or bounce off the shaft walls.



## JUST REWARDS

Sure, I love adventure. But let's give credit where credit is due. These are the amounts I agreed to before I left:

- \* We're given an advance of 4000 points as soon as we begin.

Thereafter, we'll receive:

- \* 5,000 points for every gold bar
- \* 15,000 points for the primitive cave rat
- \* 20,000 points for the Raj diamond
- \* 10,000 points for Rhonda's safe return
- \* 10,000 points for Quickclaw's safe return

Now hold on. Each time I succumb to a hazard, I'll continually lose points as I regress back to the last healing station (red cross) I contacted. The longer that journey, the more points I'll lose. Also, 100 points are deducted for every unintentional fall.

## HARRY STEPS ASIDE



Hint:  
Draw  
a  
map

While floating on a balloon today, I realized that it is unfair of me to take all the credit for what we do together. I hereby establish the Activision Cliffhangers -- open to any co-adventurer who collects 99,000 points or more. Send me a photo of the TV screen showing your qualifying score, along with your name and address. I'll send you the official Cliffhangers emblem. Be sure to write "Pitfall II" and your score on the bottom corner of the envelope. OK?



## FAMOUS LAST WORDS OR WHAT DAVID CRANE TOLD ME BEFORE I LEFT

Since David taught me everything I know,  
I thought I'd share this letter.

Dear Pitfall,

Good luck in the Lost Caverns! Here are some tips that'll help you out:

Time your approach to condors and bats so that you run exactly below their highest elevation.

A free-fall down an entire shaft can be a short-cut to the river below. This particular jump must be timed, though, to avoid colliding with bats on the way down.

If you're **unintentionally** falling down a chute of ladders or past many levels, hold the Joystick to the left or right. The underground wind will **slowly** move you in that direction.

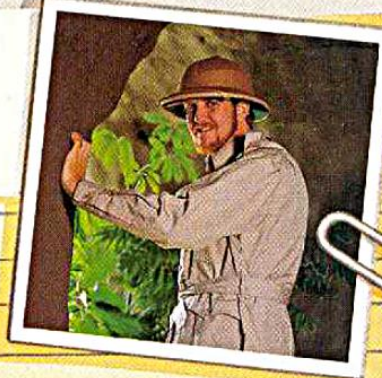
Don't get discouraged if a bat gets you whenever you go from a ladder to a gold bar. Listen up. Stay low on the ladder, wait until a bat is just over you, **then** climb up quickly and run to the gold bar. You'll barely miss the next bat, but miss it you will.

Finally, not everything you **see** is always easy to **get** to. Some things can be so close, yet so far away. Like Quickclaw, for instance. And the rat...who, incidentally, can only be subdued from behind.

Good luck and don't forget to write.

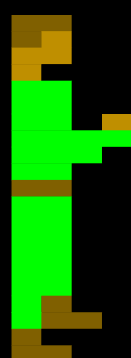
David Crane

David is one of the most highly awarded video game designers in the world. Born in Indiana, he now lives in California and is an avid tennis player. His numerous works include The Activision Decathlon and, of course, the original Pitfall!"

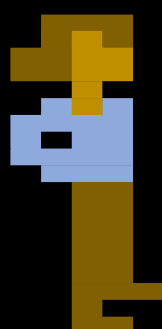


With that, I'll bid farewell, my  
friends. I hope to see you soon!  
Pitfall Harry.

P.S. Please write to David for me. I've run out of paper!



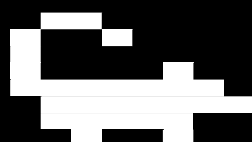
HARRY



RHONDA



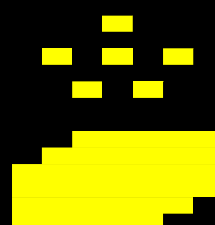
QUICKCLAW



RAT



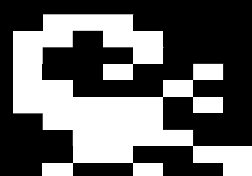
RAJ



GOLD



FROG



SCORPION



CROSS



BAT



EEL



CONDOR

00	00	00	00	00	11	11	00	00	00	00	00
00	00	00	00	00	11	01	00	00	00	00	00
00	00	00	00	00	01	01	00	00	00	00	00
00	00	00	00	00	01	00	00	00	00	00	00
00	00	00	00	00	10	10	00	00	00	00	00
00	00	00	00	00	10	10	00	00	00	00	00
00	00	00	00	00	10	10	00	01	00	00	00
00	00	00	00	00	10	10	10	10	00	00	00
00	00	00	00	00	10	10	10	00	00	00	00
00	00	00	00	00	10	10	00	00	00	00	00
00	00	00	00	00	11	11	00	00	00	00	00
00	00	00	00	00	10	10	00	00	00	00	00
00	00	00	00	00	10	10	00	00	00	00	00
00	00	00	00	00	10	10	00	00	00	00	00
00	00	00	00	00	10	10	00	00	00	00	00
00	00	00	00	00	10	10	00	00	00	00	00
00	00	00	00	00	10	10	00	00	00	00	00
00	00	00	00	00	10	10	00	00	00	00	00
00	00	00	00	00	10	10	00	00	00	00	00
00	00	00	00	00	10	10	00	00	00	00	00
00	00	00	00	00	10	11	00	00	00	00	00
00	00	00	00	00	10	11	11	00	00	00	00
00	00	00	00	00	11	00	00	00	00	00	00
00	00	00	00	00	11	11	00	00	00	00	00

	BINARY		HEX
00000000	00111100	00000000	00 3c 00
00000000	00110100	00000000	00 34 00
00000000	00010100	00000000	00 14 00
00000000	00010000	00000000	00 10 00
00000000	00101000	00000000	00 28 00
00000000	00101000	00000000	00 28 00
00000000	00101000	01000000	00 28 40
00000000	00101010	10000000	00 2a 80
00000000	00101010	00000000	00 2a 00
00000000	00101000	00000000	00 28 00
00000000	00111100	00000000	00 3c 00
00000000	00101000	00000000	00 28 00
00000000	00101000	00000000	00 28 00
00000000	00101000	00000000	00 28 00
00000000	00101000	00000000	00 28 00
00000000	00101000	00000000	00 28 00
00000000	00101000	00000000	00 28 00
00000000	00101000	00000000	00 28 00
00000000	00101100	00000000	00 2c 00
00000000	00101111	00000000	00 2f 00
00000000	00110000	00000000	00 30 00
00000000	00111100	00000000	00 3c 00

```

0900 .....####..... 00 3c 00
0903 .....#.#..... 00 34 00
0906 .....#.#..... 00 14 00
0909 .....#..... 00 10 00
090c .....#.#..... 00 28 00
090f .....#.#..... 00 28 00
0912 .....#.#.....#..... 00 28 40
0915 .....#.#.#.#..... 00 2a 80
0918 .....#.#.#..... 00 2a 00
091b .....#.#..... 00 28 00
091e .....####..... 00 3c 00
0921 .....#.#..... 00 28 00
0924 .....#.#..... 00 28 00
0927 .....#.#..... 00 28 00
092a .....#.#..... 00 28 00
092d .....#.#..... 00 28 00
0930 .....#.#..... 00 28 00
0933 .....#.#..... 00 2c 00
0936 .....#.####..... 00 2f 00
0939 .....#.#..... 00 30 00
093c .....####..... 00 3c 00

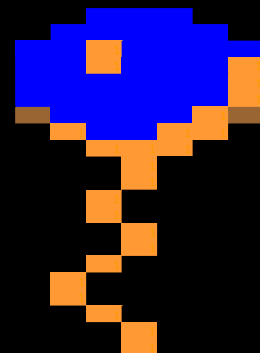
```

\$0940) m 0900 093e

```

0900 00 3c 00 00 34 00 00 14 00 00 10 00 00 28 00 00 28 00 00 3c
0910 28 00 00 28 40 00 2a 80 00 2a 00 00 28 00 00 28 00 00 3c
0920 00 00 28 00 00 28 00 00 28 00 00 28 00 00 28 00 00 3c
0930 00 28 00 00 2c 00 00 2f 00 00 30 00 00 3c 00

```



BALLOON

# The Sprites

We want to be good at documenting the knowledge we have acquired. Even the small bites are important to us and they are all too easily forgotten. It is therefore advantageous if we can always look it up quickly. Everything is difficult if you don't understand it. Understanding multicolor sprite data by trial and error, for example, is almost impossible. So, you have to have it explained to you once: Single color sprites can be displayed in highres with a resolution of 24x21 pixels. However, if you want to display colorful sprites, the horizontal resolution is halved to 12 2x1 sized pixels. The 4 possible sprite colors are coded in each 2x1 pixel using 2 bits (00, 01, 10, 11). In turn, 8 bits are (as you know) best stored as hex numbers. Therefore, you will find 3x21 hex numbers that encode a multicolor sprite in the memory of the C64. Three hex numbers are 3x4 double bits. Take a look at Harry with the superimposed double bit combinations and you will understand how it works. If you use multicolor sprites, note that the 4 colors have a few restrictions: (1) there is a "background color"- this is actually not a real sprite color, but is transparent and shows the background color; (2) color 2 and 3 can be freely selected from the 16 possible colors of the breadbox, but are fixed for all sprites; (3) there is at least 1 color that can be set individually for a sprite. For Harry, for example, this is "green". Take a look at Harry, Rhonda and Quickclaw. All have brown and light brown parts (fixed color 2+3) and 1 individual (at least Rhonda).

## In case you are using AR6 (which makes no sense, but ok....)

- 1) Disconnect your floppy; load the first file (until tape counter ~045)
- 2) Start the ML monitor and do the following manipulations:
  - a. Replace the RTS in \$104B with a BRK and g 1000
- 3) The monitor will be activated again
  - a. Replace the BRK in \$104B back to RTS
  - b. Check the SP and the jump address of RTS (\$10ff [+1])
  - c. Disassemble from \$1100 (d 1100 1120)
  - d. Replace #\$E0 in \$1111 with #\$00 (something < \$0317)
  - e. M \$11e2 (CBM80); change \$11e6 to \$31 (CBM80 check)
  - f. Replace RTS in \$11d5 to BRK
  - g. G 1100 (ACTIVISION SCREEN + LOAD) - \$11d5 back to RTS → in case start address is known goto step "s" now!
  - h. Check the stack (FE) & note jump address (i.e. \$7E00 = \$7DFF +1)
  - i. Change RTS in \$7e39 to BRK & G 11d5 & Change BRK in \$7e39 to RTS
  - j. Check the stack & note the jump address (i.e. \$7E42)
  - k. RTS \$7e59 to BRK & G 7e39 & BRK \$7e59 to RTS
  - l. Check the stack & note the jump address (i.e. \$7E5a)
  - m. 7e96 RTS to BRK → G 7e59 → 7e96 BRK to RTS
  - n. JMP address 7e42
  - o. \$7e59 RTS to BRK → G 7E96 → \$7E59 BRK → RTS
  - p. Jmp address 7e97
  - q. 7ee4 RTS to BRK → G 7E59 → STACK: e6 ff 9e 7e → 7EE4 BRK→RTS
  - r. Set BRK to \$7ebd → G 7EE4 and read stack (**\$86B4**)
  - s. Set a break to \$86f7, g \$86b4, check stack (**\$8009**), set \$86f7 to RTS
  - t. G 8009 will start Pitfall II (start address = \$8009). Turn off and on computer, connect & activate floppy with a blank diskette. Repeat until step "g" and goto "s".
  - u. Save to disk from \$8009 to \$c000, (Start address is \$8009/SYS 32777)- pack it and **you are done!** BTW: if you are using *exomizer* you'll get a 38 Block version.

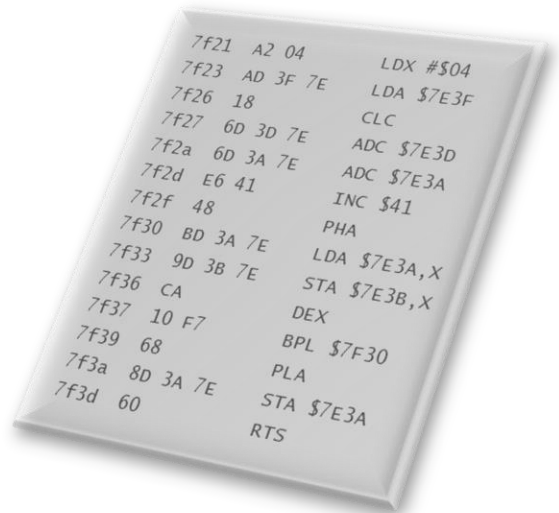
PITFALL II

# THE CODE BOOK



## \$7E00 - 2nd data bunch

7e00	A9 7F	LDA #\$7F	
7e02	8D 19 03	STA \$0319	
7e05	A9 6F	LDA #\$6F	
7e07	8D 18 03	STA \$0318	: NMI vector to \$7f6f
7e0a	A9 59	LDA #\$59	
7e0c	85 3B	STA \$3B	
7e0e	18	CLC	
7e0f	69 E8	ADC #\$E8	
7e11	AA	TAX	
7e12	A9 7E	LDA #\$7E	
7e14	85 3C	STA \$3C	
7e16	69 FF	ADC #\$FF	
7e18	48	PHA	
7e19	8A	TXA	
7e1a	48	PHA	
7e1b	A9 7F	LDA #\$7F	
7e1d	85 8C	STA \$8C	
7e1f	A9 01	LDA #\$01	
7e21	85 8B	STA \$8B	
7e23	A2 3B	LDX #\$3B	
7e25	A0 02	LDY #\$02	
7e27	B5 00	LDA \$00,X	
7e29	95 FE	STA \$FE,X	
7e2b	E8	INX	
7e2c	88	DEY	
7e2d	D0 F8	BNE \$7E27	
7e2f	84 41	STY \$41	
7e31	E6 39	INC \$39	
7e33	D0 02	BNE \$7E37	
7e35	E6 3A	INC \$3A	
7e37	A0 3C	LDY #\$3C	
<b>7e39</b>	<b>60</b>	<b>RTS</b>	
7e42	20 21 7F	JSR \$7F21	
7e45	51 39	EOR (\$39),Y	
7e47	91 39	STA (\$39),Y	
7e49	20 21 7F	JSR \$7F21	
7e4c	51 3B	EOR (\$3B),Y	
7e4e	88	DEY	
7e4f	D0 F6	BNE \$7E47	
7e51	91 39	STA (\$39),Y	
7e53	A5 3C	LDA \$3C	
7e55	48	PHA	
7e56	A5 3B	LDA \$3B	
7e58	48	PHA	
<b>7e59</b>	<b>60</b>	<b>RTS</b>	



7e60	BA	TSX	: encrypted before \$7E42 is exec.
7e61	B9 E6 7E	LDA	\$7EE6,Y
7e64	D0 07	BNE	\$7E6D
7e66	B9 00 00	LDA	\$0000,Y
7e69	29 10	AND	#\$10
7e6b	F0 03	BEQ	\$7E70
7e6d	6C 8B 00	JMP	(\$008B)
7e70	9A	TXS	
7e71	A9 7E	LDA	#\$7E
7e73	91 B9	STA	(\$B9),Y
7e75	88	DEY	
7e76	A9 41	LDA	#\$41
7e78	91 B9	STA	(\$B9),Y
7e7a	A5 39	LDA	\$39
7e7c	18	CLC	
7e7d	69 3D	ADC	#\$3D
7e7f	85 39	STA	\$39
7e81	A5 3A	LDA	\$3A
7e83	69 00	ADC	#\$00
7e85	85 3A	STA	\$3A
7e87	A5 3B	LDA	\$3B
7e89	18	CLC	
7e8a	69 3D	ADC	#\$3D
7e8c	85 3B	STA	\$3B
7e8e	A5 3C	LDA	\$3C
7e90	69 00	ADC	#\$00
7e92	85 3C	STA	\$3C
7e94	A0 4D	LDY	#\$4D
<b>7e96</b>	<b>60</b>	<b>RTS</b>	: back to \$7e42; then to \$7e97!
7e97	A2 07	LDX	#\$07
7e99	86 3D	STX	\$3D
7e9b	A9 05	LDA	#\$05
7e9d	85 3F	STA	\$3F

# \$7E9F

: Within the next section the **peripheral check** is performed via 44 loops from \$7EE4-to Kernal and back

7e9f	A4 3F	LDY \$3F	: current DATA Line number	←
7ea1	88	DEY	: =03, 02	
7ea2	10 1C	BPL \$7EC0	: if result = pos. in trace mode this is taken!	
7ea4	29 80	AND #\$80		
7ea6	D0 03	BNE \$7EAB		
7ea8	6C 8B 00	JMP (\$008B)		
7eab	A6 3D	LDX \$3D		
7ead	CA	DEX		
7eae	10 E9	BPL \$7E99		
7eb0	A5 8B	LDA \$8B		
7eb2	18	CLC		
7eb3	69 B2	ADC #\$B2		
7eb5	AA	TAX		
7eb6	A5 8C	LDA \$8C		
7eb8	69 07	ADC #\$07		
7eba	48	PHA		
7ebb	8A	TXA		
7ebc	48	PHA		
7ebd	4C 21 7F	<b>JMP \$7F21</b>	<b>: Good place to set a BRK!</b>	
→ 7ec0	<b>84 3F</b>	<b>STY \$3F</b>	<b>: \$3f = 03, 02</b>	
7ec2	BE E5 7E	LDX \$7EE5,Y	: y = 03,02, X=7e,00	
7ec5	D0 06	BNE \$7ECD		
7ec7	A9 0F	LDA #\$0F		
7ec9	38	SEC		
7eca	E5 3D	SBC \$3D	: 07	
7ecc	AA	TAX	: A=08, X=08	
→ 7ecd	A9 7E	LDA #\$7E	: Prepare the RTS Jumpback from Kernal to \$7e9f	
7ecf	48	PHA		
7ed0	A9 9E	LDA #\$9E	: Prepare the RTS Jumpback from Kernal to \$7e9f	
7ed2	48	PHA		
7ed3	B9 EA 7E	LDA \$7EEA,Y	: Prepare KERNAL Routine RTS Jump (ff)	
7ed6	48	PHA		
7ed7	B9 F9 7E	LDA \$7EF9,Y	: Prepare KERNAL Routine RTS Jump (bc)	
7eda	48	PHA		
7edb	B9 F4 7E	LDA \$7EF4,Y	: 03	
7ede	48	PHA		
7edf	B9 EF 7E	LDA \$7EEF,Y	: fe	
7ee2	A8	TAY	: y=fe	
7ee3	68	PLA	: 03 (A)	
7ee4	60	RTS	: \$ffe7 (STACK: 01fc e6 ff 9e 7e)	→

**Which Kernal routines are called?**

Lines \$7ed3 and \$7ed7 define the Kernal routine addresses:

Ffb7, ffc0, ffba, ffbd and ffe7

## Final walk through this section

7e9f	A4 3F	LDY \$3F	: 00	
7ea1	88	DEY	: FF	
7ea2	10 1C	BPL \$7EC0	:	
7ea4	29 80	AND #\$80	:	
7ea6	D0 03	BNE \$7EAB	:	
7ea8	6C 8B 00	JMP (\$008B)	:	
7eab	A6 3D	LDX \$3D	: 00	
7ead	CA	DEX	: FF	
7eae	10 E9	BPL \$7E99	:	
7eb0	A5 8B	LDA \$8B	: A: 01	
7eb2	18	CLC	:	
7eb3	69 B2	ADC #\$B2	: A: B3	
7eb5	AA	TAX	: X: B3	
7eb6	A5 8C	LDA \$8C	: A: 7F	
7eb8	69 07	ADC #\$07	: A: 86	
7eba	48	PHA	: 86	} \$86b4
7ebb	8A	TXA	:	
7ebc	48	PHA	: B3	
7ebd	4C 21 7F	JMP \$7F21		

Finally, we have \$86b3 in the stack which RTS's to \$86b4. Phew...!!

7f21	A2 04	LDX #\$04	
7f23	AD 3F 7E	LDA \$7E3F	
7f26	18	CLC	
7f27	6D 3D 7E	ADC \$7E3D	
7f2a	6D 3A 7E	ADC \$7E3A	
7f2d	E6 41	INC \$41	
7f2f	48	PHA	
7f30	BD 3A 7E	LDA \$7E3A, X	
7f33	9D 3B 7E	STA \$7E3B, X	
7f36	CA	DEX	
7f37	10 F7	BPL \$7F30	
7f39	68	PLA	
7f3a	8D 3A 7E	STA \$7E3A	
7f3d	60	RTS	: to \$86b4

8009	78	SEI	: disables interrupts
800a	A9 E2	LDA #\$E2	: NMI control vector
800c	8D 18 03	STA \$0318	: to \$FCE2
800f	A9 FC	LDA #\$FC	: checks \$8000
8011	8D 19 03	STA \$0319	: autostart ROM
8014	D8	CLD	: deletes the decimal flag
8015	A9 2F	LDA #\$2F	: 00101111
8017	85 00	STA \$00	: set to default
8019	A5 01	LDA \$01	: Default: \$37, %00110111
801b	29 FE	AND #\$FE	: 11111110 → 00110110 (\$36)
801d	85 01	STA \$01	: store back / turn BASIC off?
801f	A9 00	LDA #\$00	
8021	AA	TAX	
8022	9D 00 D0	STA \$D000,X	: set to zero
8025	9D 00 D4	STA \$D400,X	: set to zero
8028	9D 00 DC	STA \$DC00,X	: set to zero
802b	E8	INX	: set to zero
802c	D0 F4	BNE \$8022	: set to zero
802e	A9 3F	LDA #\$3F	: 00111111
8030	8D 02 DD	STA \$DD02	: Port A data direction register
8033	29 07	AND #\$07	: 00000111
8035	8D 11 D0	STA \$D011	: Screen control register
8038	A9 02	LDA #\$02	: 00000010
803a	8D 00 DD	STA \$DD00	:
803d	A9 00	LDA #\$00	: set 02-FF to zero
803f	A2 FF	LDX #\$FF	: indicating that 02-FF is
8041	9A	TXS	: heavily used for game
8042	CA	DEX	: purposes
8043	95 01	STA \$01,X	:
8045	CA	DEX	:
8046	D0 FB	BNE \$8043	:

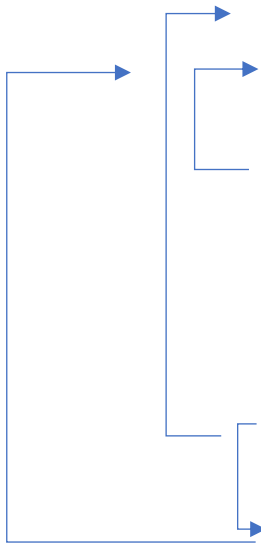
8048	2C 11 D0	BIT \$D011	: waits while raster beam
804b	30 FB	BMI \$8048	: <b>B</b> branch on <b>MI</b> nus
804d	2C 11 D0	BIT \$D011	: waits while raster beam
8050	10 FB	BPL \$804D	: <b>B</b> branch on <b>PL</b> us
8052	2C 11 D0	BIT \$D011	: waits while raster beam

947c	A9 10	LDA #\$10
947e	8D 04 D4	STA \$D404
9481	8D 0B D4	STA \$D40B
9484	8D 12 D4	STA \$D412
9487	60	RTS

USED ZEROPAGE VARs:

\$0C, \$0D, \$0E

8055	10 09	BPL \$8060	: Branch on <b>PL</b> us
8057	AD 12 D0	LDA \$D012	: Read: Current raster line
(bits #0-#7).			
805a	C9 18	CMP #\$18	
805c	90 F4	BCC \$8052	: Branch on <b>Car</b> ry Clear
805e	E6 0C	INC \$0C	: inc 000C
<b>8060</b>	<b>20 7C 94</b>	<b>JSR \$947C</b>	
8063	A9 18	LDA #\$18	: 00011000
8065	8D 16 D0	STA \$D016	: Multicolor ON + 40 Col
ON			
8068	A9 FF	LDA #\$FF	
806a	85 0D	STA \$0D	: store FF in 000D
806c	A9 D7	LDA #\$D7	
806e	85 0E	STA \$0E	: store D7 in 000E
8070	A9 08	LDA #\$08	: Define color
8072	A2 03	LDX #\$03	
8074	A0 00	LDY #\$00	
8076	88	DEY	: Y=FF
8077	91 0D	STA (\$0D),Y	: MSB: \$0E = D7; LSB:
\$0D=FF			
8079	D0 FB	BNE \$8076	
807b	E6 0E	INC \$0E	: D8, D9, DA, <b>DB</b>
807d	CA	DEX	
807e	30 0A	BMI \$808A	
8080	F0 03	BEQ \$8085	
8082	4C 74 80	JMP \$8074	
8085	A0 E9	LDY #\$E9	
8087	4C 76 80	JMP \$8076	
808a	A9 08	LDA #\$08	
808c	8D 21 D0	STA \$D021	: Define color
808f	A9 09	LDA #\$09	
8091	8D 22 D0	STA \$D022	: Define color
8094	A9 06	LDA #\$06	
8096	8D 23 D0	STA \$D023	: Define color



8099	A9 40	<b>LDA #\$40</b>	: set pointer 0e/0d to \$4000
809b	85 0E	STA \$0E	: where sprites are placed
809d	A9 00	<b>LDA #\$00</b>	
809f	85 0D	STA \$0D	
80a1	A8	TAY	
80a2	A2 2F	LDX #\$2F	
80a4	91 0D	STA (\$0D),Y	: MSB: \$40; LSB: \$00;
80a6	88	DEY	
80a7	D0 FB	BNE \$80A4	
80a9	E6 0E	INC \$0E	
80ab	CA	DEX	
80ac	10 F6	BPL \$80A4	
80ae	A9 00	LDA #\$00	
80b0	85 0D	STA \$0D	
80b2	A9 70	LDA #\$70	
80b4	85 0E	STA \$0E	
80b6	A9 C3	LDA #\$C3	
80b8	85 0F	STA \$0F	
80ba	A9 B2	LDA #\$B2	
80bc	85 10	STA \$10	
80be	A2 07	LDX #\$07	
80c0	A0 00	LDY #\$00	
80c2	B1 0F	LDA (\$0F),Y	
80c4	91 0D	STA (\$0D),Y	
80c6	88	DEY	
80c7	D0 F9	BNE \$80C2	
80c9	E6 0E	INC \$0E	
80cb	E6 10	INC \$10	
80cd	CA	DEX	
80ce	10 F0	BPL \$80C0	
80d0	A9 00	LDA #\$00	
80d2	85 0D	STA \$0D	



80d4	A9 40	LDA #\$40
80d6	85 0E	STA \$0E
80d8	A9 CD	LDA #\$CD
80da	85 0F	STA \$0F
80dc	A9 AA	LDA #\$AA
80de	85 10	STA \$10
80e0	A0 00	LDY #\$00
80e2	B1 0F	LDA (\$0F),Y
80e4	C9 FF	CMP #\$FF
80e6	D0 03	BNE \$80EB
80e8	4C 72 81	JMP \$8172
80eb	C8	INY
80ec	18	CLC
80ed	71 0F	ADC (\$0F),Y
80ef	85 11	STA \$11
80f1	B1 0F	LDA (\$0F),Y
80f3	18	CLC
80f4	69 02	ADC #\$02
80f6	85 12	STA \$12
80f8	85 13	STA \$13
80fa	A4 12	LDY \$12
80fc	88	DEY
80fd	C0 01	CPY #\$01
80ff	F0 0E	BEQ \$810F
8101	B1 0F	LDA (\$0F),Y
8103	84 12	STY \$12
8105	A4 11	LDY \$11
8107	88	DEY
8108	91 0D	STA (\$0D),Y
810a	84 11	STY \$11
810c	4C FA 80	JMP \$80FA
810f	A5 13	LDA \$13
8111	18	CLC
8112	65 0F	ADC \$0F

8114	85 0F	STA \$0F
8116	90 02	BCC \$811A
8118	E6 10	INC \$10
811a	A5 0D	LDA \$0D
811c	85 11	STA \$11
811e	18	CLC
811f	69 40	ADC #\$40
8121	85 0D	STA \$0D
8123	A5 0E	LDA \$0E
8125	85 12	STA \$12
8127	90 02	BCC \$812B
8129	E6 0E	INC \$0E
812b	88	DEY
812c	A9 02	LDA #\$02
812e	85 13	STA \$13
8130	85 14	STA \$14
8132	B1 11	LDA (\$11),Y
8134	85 16	STA \$16
8136	A9 00	LDA #\$00
8138	85 17	STA \$17
813a	A2 03	LDX #\$03
813c	A5 16	LDA \$16
813e	06 17	ASL \$17
8140	06 17	ASL \$17
8142	46 16	LSR \$16
8144	46 16	LSR \$16
8146	29 03	AND #\$03
8148	05 17	ORA \$17
814a	85 17	STA \$17
814c	CA	DEX
814d	10 ED	BPL \$813C
814f	48	PHA
8150	C8	INY
8151	C6 13	DEC \$13

8153	10 DD	BPL \$8132
8155	88	DEY
8156	88	DEY
8157	88	DEY
8158	68	PLA
8159	91 0D	STA (\$0D),Y
815b	C8	INY
815c	C6 14	DEC \$14
815e	10 F8	BPL \$8158
8160	C0 3F	CPY #\$3F
8162	90 C8	BCC \$812C
8164	A5 0D	LDA \$0D
8166	18	CLC
8167	69 40	ADC #\$40
8169	85 0D	STA \$0D
816b	90 02	BCC \$816F
816d	E6 0E	INC \$0E
816f	4C E0 80	JMP \$80E0
8172	A9 FF	LDA #\$FF
8174	A2 26	LDX #\$26
8176	9D 80 59	STA \$5980,X
8179	CA	DEX
817a	10 FA	BPL \$8176
817c	A9 00	LDA #\$00
817e	A2 27	LDX #\$27
8180	9D 00 10	STA \$1000,X
8183	CA	DEX
8184	10 FA	BPL \$8180
8186	A9 02	LDA #\$02
8188	A2 27	LDX #\$27
818a	9D 28 10	STA \$1028,X
818d	CA	DEX
818e	10 FA	BPL \$818A
8190	A9 03	LDA #\$03

8192	A2 27	LDX #\$27
8194	9D 50 10	STA \$1050,X
8197	CA	DEX
8198	10 FA	BPL \$8194
819a	A9 01	LDA #\$01
819c	A2 27	LDX #\$27
819e	9D 78 10	STA \$1078,X
81a1	CA	DEX
81a2	10 FA	BPL \$819E
81a4	A9 01	LDA #\$01
81a6	A2 27	LDX #\$27
81a8	9D A0 10	STA \$10A0,X
81ab	CA	DEX
81ac	10 FA	BPL \$81A8
81ae	A9 05	LDA #\$05
81b0	8D B3 10	STA \$10B3
81b3	8D B4 10	STA \$10B4
81b6	A9 00	LDA #\$00
81b8	A2 27	LDX #\$27
81ba	9D C8 10	STA \$10C8,X
81bd	CA	DEX
81be	10 FA	BPL \$81BA
81c0	A9 06	LDA #\$06
81c2	8D DB 10	STA \$10DB
81c5	A9 07	LDA #\$07
81c7	8D DC 10	STA \$10DC
81ca	A2 00	LDX #\$00
81cc	BD B3 82	LDA \$82B3,X
81cf	85 0D	STA \$0D
81d1	BD F3 82	LDA \$82F3,X
81d4	85 0E	STA \$0E
81d6	BD 33 82	LDA \$8233,X
81d9	85 0F	STA \$0F
81db	BD 73 82	LDA \$8273,X

81de	85 10	STA \$10
81e0	A0 27	LDY #\$27
81e2	B1 0D	LDA (\$0D),Y
81e4	91 0F	STA (\$0F),Y
81e6	88	DEY
81e7	10 F9	BPL \$81E2
81e9	BD 33 83	LDA \$8333,X
81ec	10 1A	BPL \$8208
81ee	86 11	STX \$11
81f0	29 3F	AND #\$3F
81f2	0A	ASL A
81f3	0A	ASL A
81f4	AA	TAX
81f5	A9 03	LDA #\$03
81f7	85 12	STA \$12
81f9	A0 00	LDY #\$00
81fb	BD 73 83	LDA \$8373,X
81fe	91 0F	STA (\$0F),Y
8200	E8	INX
8201	C8	INY
8202	C6 12	DEC \$12
8204	10 F5	BPL \$81FB
8206	A6 11	LDX \$11
8208	BD 33 83	LDA \$8333,X
820b	29 7F	AND #\$7F
820d	69 40	ADC #\$40
820f	10 1A	BPL \$822B
8211	86 11	STX \$11
8213	29 3F	AND #\$3F
8215	0A	ASL A
8216	0A	ASL A
8217	AA	TAX
8218	A9 03	LDA #\$03
821a	85 12	STA \$12

821c	A0 24	LDY #\$24
821e	BD AF 83	LDA \$83AF,X
8221	91 0F	STA (\$0F),Y
8223	E8	INX
8224	C8	INY
8225	C6 12	DEC \$12
8227	10 F5	BPL \$821E
8229	A6 11	LDX \$11
822b	E8	INX
822c	E0 40	CPX #\$40
822e	90 9C	BCC \$81CC
<b>8230</b>	<b>4C EB 83</b>	<b>JMP \$83EB</b>
8233	F0 18	BEQ \$824D
8235	40	RTI
8236	68	PLA
8237	90 B8	BCC \$81F1
8239	E0 08	CPX #\$08
823b	30 58	BMI \$8295
823d	80 A8	NOOP #\$A8

**\$83EB**

83eb	A9 30	LDA #\$30	: sprite "game info" pos.
83ed	8D 02 D0	STA \$D002	
83f0	0A	ASL A	: Arithmetic Shift Left
83f1	8D 00 D0	STA \$D000	
83f4	A9 64	LDA #\$64	
83f6	8D 04 D0	STA \$D004	
83f9	A9 01	LDA #\$01	
83fb	8D 27 D0	STA \$D027	
83fe	8D 28 D0	STA \$D028	
8401	A9 07	LDA #\$07	
8403	8D 1D D0	STA \$D01D	
8406	A2 80	LDX #\$80	
8408	8E F8 7B	STX \$7BF8	
840b	8E F8 7F	STX \$7FF8	
840e	E8	INX	
840f	8E F9 7B	STX \$7BF9	
8412	8E F9 7F	STX \$7FF9	
8415	A9 09	LDA #\$09	
8417	8D 26 D0	STA \$D026	
841a	A9 08	LDA #\$08	
841c	8D 25 D0	STA \$D025	
841f	A9 FC	LDA #\$FC	
8421	8D 1C D0	STA \$D01C	
8424	A9 05	LDA #\$05	
8426	8D 2B D0	STA \$D02B	
8429	A9 06	LDA #\$06	
842b	8D 2C D0	STA \$D02C	
842e	A9 AC	LDA #\$AC	
8430	8D 06 D0	STA \$D006	
8433	A9 00	LDA #\$00	
8435	8D 2A D0	STA \$D02A	
8438	A2 0F	LDX #\$0F	
843a	20 6E 9C	JSR \$9C6E	
843d	A9 28	LDA #\$28	



843f	8D 05 D4	STA \$D405
8442	8D 0C D4	STA \$D40C
8445	8D 13 D4	STA \$D413
8448	A9 88	LDA #\$88
844a	8D 06 D4	STA \$D406
844d	8D 0D D4	STA \$D40D
8450	8D 14 D4	STA \$D414
8453	A9 0F	LDA #\$0F
8455	8D 18 D4	STA \$D418
8458	A9 BF	LDA #\$BF
845a	8D 14 03	STA \$0314
845d	A9 9C	LDA #\$9C
845f	8D 15 03	STA \$0315
8462	A9 F6	LDA #\$F6
8464	8D 12 D0	STA \$D012
8467	A9 01	LDA #\$01
8469	8D 1A D0	STA \$D01A
846c	AE 19 D0	LDX \$D019
846f	8D 19 D0	STA \$D019
8472	58	CLI
8473	A9 F6	LDA #\$F6
8475	CD 12 D0	CMP \$D012
8478	B0 FB	BCS \$8475
847a	A5 51	LDA \$51
847c	D0 0A	BNE \$8488
847e	A5 1E	LDA \$1E
8480	85 53	STA \$53
8482	20 48 94	JSR \$9448
8485	4C D7 84	JMP \$84D7
8488	C9 01	CMP #\$01
848a	D0 14	BNE \$84A0
848c	A5 4D	LDA \$4D
848e	20 3D 94	JSR \$943D
8491	20 E0 92	JSR \$92E0

8494	20 00 93	JSR \$9300
8497	E0 0C	CPX #\$0C
8499	D0 F9	BNE \$8494
849b	E6 51	INC \$51
849d	4C D7 84	JMP \$84D7
84a0	C9 02	CMP #\$02
84a2	D0 0E	BNE \$84B2
84a4	A2 0C	LDX #\$0C
84a6	20 00 93	JSR \$9300
84a9	E0 18	CPX #\$18
84ab	90 F9	BCC \$84A6
84ad	E6 51	INC \$51
84af	4C D7 84	JMP \$84D7
84b2	C9 03	CMP #\$03
84b4	D0 13	BNE \$84C9
84b6	A5 1E	LDA \$1E
84b8	29 FC	AND #\$FC
84ba	85 53	STA \$53
84bc	20 48 94	JSR \$9448
84bf	E6 53	INC \$53
84c1	20 48 94	JSR \$9448
84c4	E6 51	INC \$51
84c6	4C D7 84	JMP \$84D7
84c9	E6 53	INC \$53
84cb	20 48 94	JSR \$9448
84ce	E6 53	INC \$53
84d0	20 48 94	JSR \$9448
84d3	A9 00	LDA #\$00
84d5	85 51	STA \$51
84d7	A5 02	LDA \$02
84d9	18	CLC
84da	E9 80	SBC #\$80
84dc	F0 3D	BEQ \$851B
84de	A5 05	LDA \$05

84e0	AA	TAX
84e1	A9 80	LDA #\$80
84e3	85 0D	STA \$0D
84e5	BD 11 85	LDA \$8511,X
84e8	85 0E	STA \$0E
84ea	A9 00	LDA #\$00
84ec	85 0F	STA \$0F
84ee	BD 16 85	LDA \$8516,X
84f1	85 10	STA \$10
84f3	A4 04	LDY \$04
84f5	B1 0D	LDA (\$0D),Y
84f7	99 80 5A	STA \$5A80,Y
84fa	B1 0F	LDA (\$0F),Y
84fc	99 80 5B	STA \$5B80,Y
84ff	E6 04	INC \$04
8501	D0 0B	BNE \$850E
8503	A6 05	LDX \$05
8505	E8	INX
8506	E0 05	CPX #\$05
8508	90 02	BCC \$850C
850a	A2 00	LDX #\$00
850c	86 05	STX \$05
850e	4C 23 85	JMP \$8523
8511	55 57	EOR \$57,X
8513	58	CLI
8514	56 54	LSR \$54,X
8516	56 58	LSR \$58,X
8518	59 57 55	EOR \$5557,Y
851b	A6 44	LDX \$44
851d	9D 80 5A	STA \$5A80,X
8520	9D 80 5B	STA \$5B80,X
8523	A2 FF	LDX #\$FF
8525	8E 02 DC	STX \$DC02
8528	8E 00 DC	STX \$DC00

852b	E8	INX
852c	8E 03 DC	STX \$DC03
852f	AD 01 DC	LDA \$DC01
8532	29 1F	AND #\$1F
8534	85 0A	STA \$0A
8536	29 10	AND #\$10
8538	F0 05	BEQ \$853F
853a	86 48	STX \$48
853c	4C 50 85	JMP \$8550
853f	A5 48	LDA \$48
8541	F0 09	BEQ \$854C
8543	A5 0A	LDA \$0A
8545	09 10	ORA #\$10
8547	85 0A	STA \$0A
8549	4C 50 85	JMP \$8550
854c	A9 01	LDA #\$01
854e	85 48	STA \$48
8550	A5 0A	LDA \$0A
8552	C9 1F	CMP #\$1F
8554	F0 06	BEQ \$855C
8556	86 49	STX \$49
8558	86 4A	STX \$4A
855a	86 4B	STX \$4B
855c	A5 02	LDA \$02
855e	24 4C	BIT \$4C
8560	30 04	BMI \$8566
8562	C9 80	CMP #\$80
8564	F0 1D	BEQ \$8583
8566	C9 C0	CMP #\$C0
8568	D0 15	BNE \$857F
856a	A9 03	LDA #\$03
856c	C5 51	CMP \$51
856e	D0 02	BNE \$8572
8570	85 27	STA \$27

8572	A2 17	LDX #\$17
8574	A5 79	LDA \$79
8576	29 08	AND #\$08
8578	D0 07	BNE \$8581
857a	A2 15	LDX #\$15
857c	4C 81 85	JMP \$8581
857f	A2 1F	LDX #\$1F
8581	86 0A	STX \$0A
8583	A2 FF	LDX #\$FF
8585	8E 03 DC	STX \$DC03
8588	CA	DEX
8589	8E 02 DC	STX \$DC02
858c	A9 EF	LDA #\$EF
858e	8D 01 DC	STA \$DC01
8591	AD 00 DC	LDA \$DC00
8594	C9 FE	CMP #\$FE
8596	D0 0F	BNE \$85A7
8598	A9 80	LDA #\$80
859a	85 02	STA \$02
859c	0A	ASL A
859d	85 03	STA \$03
859f	85 04	STA \$04
85a1	85 05	STA \$05
85a3	A2 2E	LDX #\$2E
85a5	D0 18	BNE \$85BF
85a7	A5 44	LDA \$44
85a9	D0 2D	BNE \$85D8
85ab	24 02	BIT \$02
85ad	30 19	BMI \$85C8
85af	E6 03	INC \$03
85b1	A5 03	LDA \$03
85b3	29 07	AND #\$07
85b5	85 03	STA \$03
85b7	D0 0F	BNE \$85C8

85b9	A9 C0	LDA #\$C0
85bb	85 02	STA \$02
85bd	A2 0F	LDX #\$0F
85bf	20 F9 87	JSR \$87F9
85c2	20 6E 9C	JSR \$9C6E
85c5	4C 16 86	JMP \$8616
85c8	E6 4A	INC \$4A
85ca	D0 0C	BNE \$85D8
85cc	E6 49	INC \$49
85ce	A5 49	LDA \$49
85d0	C9 04	CMP #\$04
85d2	90 04	BCC \$85D8
85d4	A9 80	LDA #\$80
85d6	85 49	STA \$49
85d8	A5 51	LDA \$51
85da	D0 25	BNE \$8601
85dc	A9 7F	LDA #\$7F
85de	8D 01 DC	STA \$DC01
85e1	AD 00 DC	LDA \$DC00
85e4	C9 7F	CMP #\$7F
85e6	F0 0D	BEQ \$85F5
85e8	C9 7D	CMP #\$7D
85ea	F0 09	BEQ \$85F5
85ec	A5 4B	LDA \$4B
85ee	29 BF	AND #\$BF
85f0	85 4B	STA \$4B
85f2	4C 01 86	JMP \$8601
85f5	24 4B	BIT \$4B
85f7	70 08	BVS \$8601
85f9	A5 4B	LDA \$4B
85fb	49 80	EOR #\$80
85fd	09 40	ORA #\$40
85ff	85 4B	STA \$4B
8601	A9 50	LDA #\$50

8603	CD 12 D0	CMP \$D012
8606	B0 FB	BCS \$8603
8608	20 9D 9B	JSR \$9B9D
860b	20 D5 94	JSR \$94D5
860e	20 F9 87	JSR \$87F9
<b>8611</b>	<b>20 BA 99</b>	<b>JSR \$99BA</b>
8614	E6 44	INC \$44
8616	A5 A6	LDA \$A6
8618	D0 07	BNE \$8621
861a	24 4B	BIT \$4B
861c	30 03	BMI \$8621
861e	20 00 88	JSR \$8800
8621	20 CD 9A	JSR \$9ACD
8624	A5 1A	LDA \$1A
8626	8D 09 D0	STA \$D009
8629	A6 51	LDX \$51
862b	E0 03	CPX #\$03
862d	D0 1B	BNE \$864A
862f	A6 52	LDX \$52
8631	BC C2 A2	LDY \$A2C2,X
8634	8A	TXA
8635	F0 0A	BEQ \$8641
8637	A6 1C	LDX \$1C
8639	B5 33	LDA \$33,X
863b	C9 04	CMP #\$04
863d	D0 02	BNE \$8641
863f	A0 1A	LDY #\$1A
8641	84 19	STY \$19
8643	A5 4D	LDA \$4D
8645	85 4E	STA \$4E
8647	20 B1 99	JSR \$99B1
864a	24 4B	BIT \$4B
864c	30 17	BMI \$8665
864e	E6 27	INC \$27

8650	A5 A1	LDA \$A1
8652	29 7F	AND #\$7F
8654	C9 18	CMP #\$18
8656	D0 04	BNE \$865C
8658	A5 A5	LDA \$A5
865a	D0 06	BNE \$8662
<b>865c</b>	<b>20 1E 98</b>	<b>JSR \$981E</b>
<b>865f</b>	<b>20 9D 98</b>	<b>JSR \$989D</b>
<b>8662</b>	<b>20 F8 86</b>	<b>JSR \$86F8</b>
8665	A5 19	LDA \$19
8667	0A	ASL A
8668	8D 08 D0	STA \$D008
866b	B0 08	BCS \$8675
866d	A9 EF	LDA #\$EF
866f	2D 10 D0	AND \$D010
8672	4C 7A 86	JMP \$867A
8675	A9 10	LDA #\$10
8677	0D 10 D0	ORA \$D010
867a	8D 10 D0	STA \$D010
867d	A5 A1	LDA \$A1
867f	4A	LSR A
8680	C9 09	CMP #\$09
8682	F0 0B	BEQ \$868F
8684	A9 00	LDA #\$00
8686	8D FD 7B	STA \$7BFD
8689	8D FD 7F	STA \$7FFD
868c	4C B1 86	JMP \$86B1
868f	AD 09 D0	LDA \$D009
8692	38	SEC
8693	E9 15	SBC #\$15
8695	8D 0B D0	STA \$D00B
8698	AD 08 D0	LDA \$D008
869b	8D 0A D0	STA \$D00A
869e	AD 10 D0	LDA \$D010



86a1	AA	TAX
86a2	29 10	AND #\$10
86a4	F0 05	BEQ \$86AB
86a6	8A	TXA
86a7	09 20	ORA #\$20
86a9	D0 03	BNE \$86AE
86ab	8A	TXA
86ac	29 DF	AND #\$DF
86ae	8D 10 D0	STA \$D010
86b1	4C 7A 84	JMP \$847A
86b4	A9 80	LDA #\$80
86b6	48	PHA
86b7	A9 08	LDA #\$08
86b9	48	PHA
86ba	18	CLC
86bb	90 02	BCC \$86BF
86bd	4D C9 AD	EOR \$ADC9
86c0	BE 86 48	LDX \$4886,Y
86c3	AD BD 86	LDA \$86BD
86c6	48	PHA
86c7	A9 92	LDA #\$92
86c9	48	PHA
86ca	A9 80	LDA #\$80
86cc	48	PHA
86cd	A9 80	LDA #\$80
86cf	85 FB	STA \$FB
86d1	68	PLA
86d2	85 FD	STA \$FD
86d4	A9 19	LDA #\$19
86d6	85 FC	STA \$FC
86d8	68	PLA
86d9	85 FE	STA \$FE
86db	A0 7F	LDY #\$7F
86dd	B1 FB	LDA (\$FB),Y

86df	91 FD	STA (\$FD),Y
86e1	88	DEY
86e2	10 F9	BPL \$86DD
86e4	A9 00	LDA #\$00
86e6	85 FB	STA \$FB
86e8	A9 90	LDA #\$90
86ea	85 FC	STA \$FC
86ec	68	PLA
86ed	85 FD	STA \$FD
86ef	68	PLA
86f0	85 FE	STA \$FE
86f2	A2 30	LDX #\$30
86f4	20 E8 0F	JSR \$0FE8
<b>86f7</b>	<b>60</b>	<b>RTS</b>
86f8	A5 51	LDA \$51
86fa	D0 56	BNE \$8752
86fc	A5 28	LDA \$28
86fe	30 52	BMI \$8752
8700	E6 59	INC \$59
8702	A5 4E	LDA \$4E
8704	0A	ASL A
8705	0A	ASL A
8706	0A	ASL A
8707	45 1E	EOR \$1E
8709	4A	LSR A
870a	4A	LSR A
870b	4A	LSR A
870c	29 01	AND #\$01
870e	A8	TAY
870f	A6 28	LDX \$28
8711	BD 65 87	LDA \$8765,X
8714	18	CLC
8715	79 63 87	ADC \$8763,Y
8718	85 5E	STA \$5E

871a	BD 7E 87	LDA \$877E,X
871d	85 5D	STA \$5D
871f	A5 59	LDA \$59
8721	29 38	AND #\$38
8723	4A	LSR A
8724	4A	LSR A
8725	4A	LSR A
8726	AA	TAX
8727	BD 5B 87	LDA \$875B,X
872a	A8	TAY
872b	B9 97 87	LDA \$8797,Y
872e	85 5B	STA \$5B
8730	B9 99 87	LDA \$8799,Y
8733	85 5C	STA \$5C
8735	BD 53 87	LDA \$8753,X
8738	18	CLC
8739	65 5B	ADC \$5B
873b	85 5B	STA \$5B
873d	90 02	BCC \$8741
873f	E6 5C	INC \$5C
8741	A6 5A	LDX \$5A
8743	BC 9B 87	LDY \$879B,X
8746	BD B3 87	LDA \$87B3,X
8749	AA	TAX
874a	B1 5B	LDA (\$5B),Y
874c	91 5D	STA (\$5D),Y
874e	88	DEY
874f	CA	DEX
8750	10 F8	BPL \$874A
<b>8752</b>	<b>60</b>	<b>RTS</b>

87d2	A5 78	LDA \$78
87d4	29 01	AND #\$01
87d6	F0 20	BEQ \$87F8
87d8	A5 7C	LDA \$7C
87da	29 08	AND #\$08
87dc	F0 1A	BEQ \$87F8
87de	A5 85	LDA \$85
87e0	29 02	AND #\$02
87e2	F0 14	BEQ \$87F8
87e4	A9 1F	LDA #\$1F
87e6	85 0A	STA \$0A
87e8	A5 A1	LDA \$A1
87ea	29 01	AND #\$01
87ec	09 02	ORA #\$02
87ee	24 A4	BIT \$A4
87f0	10 02	BPL \$87F4
87f2	09 80	ORA #\$80
87f4	85 A1	STA \$A1
87f6	C6 4C	DEC \$4C
<b>87f8</b>	<b>60</b>	<b>RTS</b>
87f9	A5 07	LDA \$07
87fb	D0 FC	BNE \$87F9
87fd	E6 07	INC \$07
<b>87ff</b>	<b>60</b>	<b>RTS</b>

# \$92D9

92d9	85 A1	STA \$A1
92db	60	RTS

: this little snippet stores A in \$A1  
: remember that #\$98 in \$A1 mean death!

# \$99BA

**Valid Code: \$83eb-\$8752      -      called by \$8230 JMP\$83eb**

99ba	A5 A1	LDA \$A1	<i>called in \$8611</i>
99bc	C9 18	CMP # \$18	
99be	D0 01	BNE \$99C1	
99c0	60	RTS	
99c1	A2 04	LDX # \$04	
99c3	B5 94	LDA \$94,X	
99c5	29 07	AND # \$07	
99c7	F0 03	BEQ \$99CC	
99c9	4C C6 9A	JMP \$9AC6	
99cc	B5 94	LDA \$94,X	
99ce	29 90	AND	
99d0	C9 90	CMP # \$90	
99d2	D0 64	BNE \$9A38	
99d4	B5 33	LDA \$33,X	
99d6	29 F0	AND # \$F0	
99d8	F0 5E	BEQ \$9A38	
99da	C9 60	CMP # \$60	
99dc	90 25	BCC \$9A03	
99de	D0 58	BNE \$9A38	
99e0	A5 4E	LDA \$4E	
99e2	C5 4F	CMP \$4F	
99e4	85 4F	STA \$4F	
99e6	F0 05	BEQ \$99ED	
99e8	A9 05	LDA # \$05	
99ea	20 9A 95	JSR \$959A	
99ed	A5 1E	LDA \$1E	
99ef	C9 02	CMP # \$02	
99f1	A5 1D	LDA \$1D	
99f3	65 1C	ADC \$1C	
99f5	C5 1B	CMP \$1B	
99f7	85 1B	STA \$1B	
99f9	F0 05	BEQ \$9A00	

99fb	A9 05	LDA #\$05
99fd	20 9A 95	JSR \$959A
9a00	4C 38 9A	JMP \$9A38
9a03	4A	LSR A
9a04	4A	LSR A
9a05	4A	LSR A
9a06	4A	LSR A
9a07	A8	TAY
9a08	B9 1E 9D	LDA \$9D1E,Y
9a0b	20 52 9C	JSR \$9C52
9a0e	A5 1E	LDA \$1E
9a10	C9 02	CMP #\$02
9a12	8A	TXA
9a13	65 1D	ADC \$1D
9a15	85 0D	STA \$0D
9a17	A8	TAY
9a18	B9 74 00	LDA \$0074,Y
9a1b	A4 4E	LDY \$4E
9a1d	19 BA A2	ORA \$A2BA,Y
9a20	A4 0D	LDY \$0D
9a22	99 74 00	STA \$0074,Y
9a25	A9 00	LDA #\$00
9a27	C5 70	CMP \$70
9a29	F0 05	BEQ \$9A30
9a2b	85 70	STA \$70
9a2d	20 91 94	JSR \$9491
9a30	20 D2 87	JSR \$87D2
9a33	A9 02	LDA #\$02
9a35	20 9A 95	JSR \$959A
9a38	B5 94	LDA \$94,X
9a3a	29 50	AND #\$50
9a3c	C9 50	CMP #\$50
9a3e	D0 7B	BNE \$9ABB
9a40	B5 33	LDA \$33,X



9a42	29 0F	AND #\$0F
9a44	C9 04	CMP #\$04
9a46	D0 30	BNE \$9A78
9a48	A5 51	LDA \$51
9a4a	D0 7A	BNE \$9AC6
9a4c	A5 52	LDA \$52
9a4e	F0 76	BEQ \$9AC6
9a50	A5 77	LDA \$77
9a52	09 02	ORA #\$02
9a54	85 77	STA \$77
9a56	B5 33	LDA \$33,X
9a58	29 F0	AND #\$F0
9a5a	95 33	STA \$33,X
9a5c	A9 00	LDA #\$00
9a5e	95 29	STA \$29,X
9a60	A9 51	LDA #\$51
9a62	20 52 9C	JSR \$9C52
9a65	A9 00	LDA #\$00
9a67	C5 70	CMP \$70
9a69	F0 05	BEQ \$9A70
9a6b	85 70	STA \$70
9a6d	20 91 94	JSR \$9491
9a70	A9 02	LDA #\$02
9a72	20 9A 95	JSR \$959A
9a75	4C C6 9A	JMP \$9AC6
9a78	C9 03	CMP #\$03
9a7a	D0 1C	BNE \$9A98
9a7c	B5 33	LDA \$33,X
9a7e	29 F0	AND #\$F0
9a80	95 33	STA \$33,X
9a82	A9 00	LDA #\$00
9a84	95 38	STA \$38,X
9a86	A5 A1	LDA \$A1
9a88	6A	ROR A

9a89	A9 09	LDA #\$09	
9a8b	2A	ROL A	
9a8c	85 A1	STA \$A1	
9a8e	A9 03	LDA #\$03	
9a90	85 70	STA \$70	
9a92	20 91 94	JSR \$9491	
9a95	4C C6 9A	JMP \$9AC6	
9a98	A5 A1	LDA \$A1	
9a9a	29 7E	AND #\$7E	
9a9c	C9 12	CMP #\$12	
9a9e	D0 17	BNE \$9AB7	
9aa0	A5 A1	LDA \$A1	
9aa2	29 01	AND #\$01	
9aa4	09 8E	ORA #\$8E	
9aa6	85 A1	STA \$A1	
9aa8	A9 01	LDA #\$01	
9aaa	85 70	STA \$70	
9aac	20 91 94	JSR \$9491	
9aaf	A9 03	LDA #\$03	
9ab1	20 9A 95	JSR \$959A	
9ab4	4C C6 9A	JMP \$9AC6	
9ab7	A9 98	LDA #\$98	: 98="dead"
<b>9ab9</b>	<b>85 A1</b>	<b>STA \$A1</b>	
9abb	B5 94	LDA \$94,X	
9abd	29 60	AND #\$60	
9abf	C9 60	CMP #\$60	
9ac1	D0 03	BNE \$9AC6	
9ac3	4C A0 9A	JMP \$9AA0	
9ac6	CA	DEX	
9ac7	30 03	BMI \$9ACC	
9ac9	4C C3 99	JMP \$99C3	
9acc	60	RTS	

\$9C52

Add points

: POINTS are stored in \$45-\$47

9c52	24 02	BIT \$02
9c54	10 15	BPL \$9C6B
9c56	70 13	BVS \$9C6B
9c58	F8	SED
9c59	85 0D	STA \$0D
9c5b	29 F0	AND #\$F0
9c5d	18	CLC
9c5e	65 46	ADC \$46
9c60	85 46	STA \$46
9c62	A5 0D	LDA \$0D
9c64	29 0F	AND #\$0F
9c66	65 47	ADC \$47
9c68	85 47	STA \$47
9c6a	D8	CLD
9c6b	60	RTS

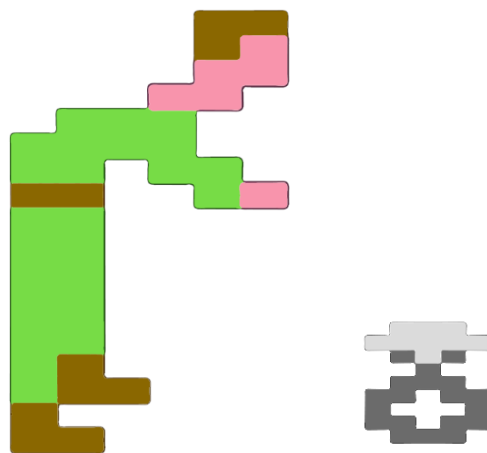
\$9C6C

Initialize \$45-  
47

9c6c	A2 2E	LDX #\$2E
9c6e	BD 90 9C	LDA <b>\$9C90</b> ,X : initial data is stored
9c71	95 19	STA \$19,X
9c73	CA	DEX
9c74	E0 FF	CPX #\$FF
9c76	D0 F6	BNE \$9C6E
9c78	A2 5F	LDX #\$5F
9c7a	A9 00	LDA #\$00
9c7c	95 48	STA \$48,X
9c7e	CA	DEX
9c7f	E0 FF	CPX #\$FF
9c81	D0 F9	BNE \$9C7C
9c83	20 7C 94	JSR \$947C
9c86	20 10 93	JSR \$9310
9c89	20 B1 99	JSR \$99B1
9c8c	20 91 94	JSR \$9491
9c8f	60	RTS

\$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
9c90	31	7f	03	01	01	08	00	1d	11	15	15	10	1a	09	00	c0
9ca0	00	00	00	4d	28	00	00	00	00	00	00	60	10	06	02	00
9cb0	00	00	16	00	16	4d	00	2e	1e	00	00	00	<b>00</b>	<b>40</b>	<b>00</b>	
													<b>TE</b>	<b>HU</b>	<b>TH</b>	

This is copied to \$19-\$47



**Contact:**

REBEL 1 of Hokuto Force  
PLK 080675 C  
3500 Kassel  
West-Germany

