Behind the scenes of a C64 demo

Ninja / The Dreams

28C3

About me

- Linux kernel hacker
- embedded systems
- low level computing
- prefers limited machines

and that's mainly because...

About me

- coding on the C64 since 1988
- mainly demo-scene related
- Zen(?)-like passion
  (searching enlightment via excessive practice :))

Some more notes

This is a group effort

- Thanks, Oswald
- Thanks, Edhellon
- Thanks, Bubis
- Thanks, Leon
- Thanks, AMN
- Thanks, Dalezy
- Thanks, Doc Bacardi

This is a case study!

- Yeah, we rock, but others do, too
- Yeah, C64 rocks, but other platforms do, too

C64 facts

- CPU: 6502-based, 985248 Hz, three 8-bit registers, 56+ opcodes, 13 addressing modes, no multiplication and such
- Memory: 64KB RAM, forget the ROM
- Graphics: 160x200x16 (4 colors per 4x8 cell), Sprites
- Sound: 3 voices (Triangle. Pulse. . .), Modulation, Filters
- Disk: 170KB per disk side, serial interface, handshake
- Other: 4 cycle-based timers, chainable

Design choices

- flow dictated by music
- high pace, effect after effect
- no obvious filling material
- avoid empty screens as much as possible
- no story
- "right in your face"

Let's get serious

Showtime!

Warning: Current working state

Some terminology

Speedcode

- unrolled loops
- usually generated at runtime

```
LDA sine,x
ADC sine,y
STA pixel
LDA sine+1,x
ADC sine+1,y
STA pixel+1
```

Some terminology

Speedcode

- unrolled loops
- usually generated at runtime

```
LDA sine,x
ADC sine,y
STA pixel
LDA sine+1,x
ADC sine+1,y
STA pixel+1
```

Illegal opcodes

- undocumented opcodes
- only few useful, but those should be used

Some terminology

Speedcode

- unrolled loops
- usually generated at runtime

```
LDA sine,x
ADC sine,y
STA pixel
LDA sine+1,x
ADC sine+1,y
STA pixel+1
```

Illegal opcodes

- undocumented opcodes
- only few useful, but those should be used

Raster time

- cycles

Things we do

Self modifying code

- can't do without
- code is data, data is code

Things we do

Self modifying code

- can't do without
- code is data, data is code

Abuse the stack

- `LDA table,x; INX` costs 6 cycles
- `PLA` costs 4 cycles (and leaves X free!)
- needs sliding window for interrupts

Things we do

Self modifying code

- can't do without
- code is data, data is code

Abuse the stack

- `LDA table,x; INX` costs 6 cycles
- `PLA` costs 4 cycles (and leaves X free!)
- needs sliding window for interrupts

Anarchy!

Assume $DC06 is our jitter counter	LDA #$4C STA $DC04 LDA #$00 STA $DC05 ...	LDA #$04 STA $FFFE LDA #$DC STA $FFFF ...

Tabled based effects

- lots of speedcode and tables
- ”magic” is in the tables and code generators
- runs in main loop
- code too slow, effect too slow
- chunky modes are common
- speed measured in fps (50 = good, 25 = well. . .)
- ”newschool” effects

VIC (register) based effects

- less speedcode and tables
- "magic" is in the actual code
- runs in interrupt, must often be cycle exact
- code too slow, effect impossible
- usually full resolution (or even increased)
- speed measured in rasterlines
- "oldschool" effects

Our team can do both \o/
Example: Bump mapper

Tools/Equipment used

Mostly cross-development

- GFX: ProjectOne, Grafx2, *AmicaPaint*, *Sprite-Char-Edit*, *other GFX tools*
- Music: GoatTracker
- Code: AS, DreamAss, CA65, *TASS*
- Custom Tools: mostly data converters or generators in C, Perl, VB, Java, *CBM BASIC V2*
- Linking: make, exomizer
- Developer machines: *C64*, Linux, Win, Mac

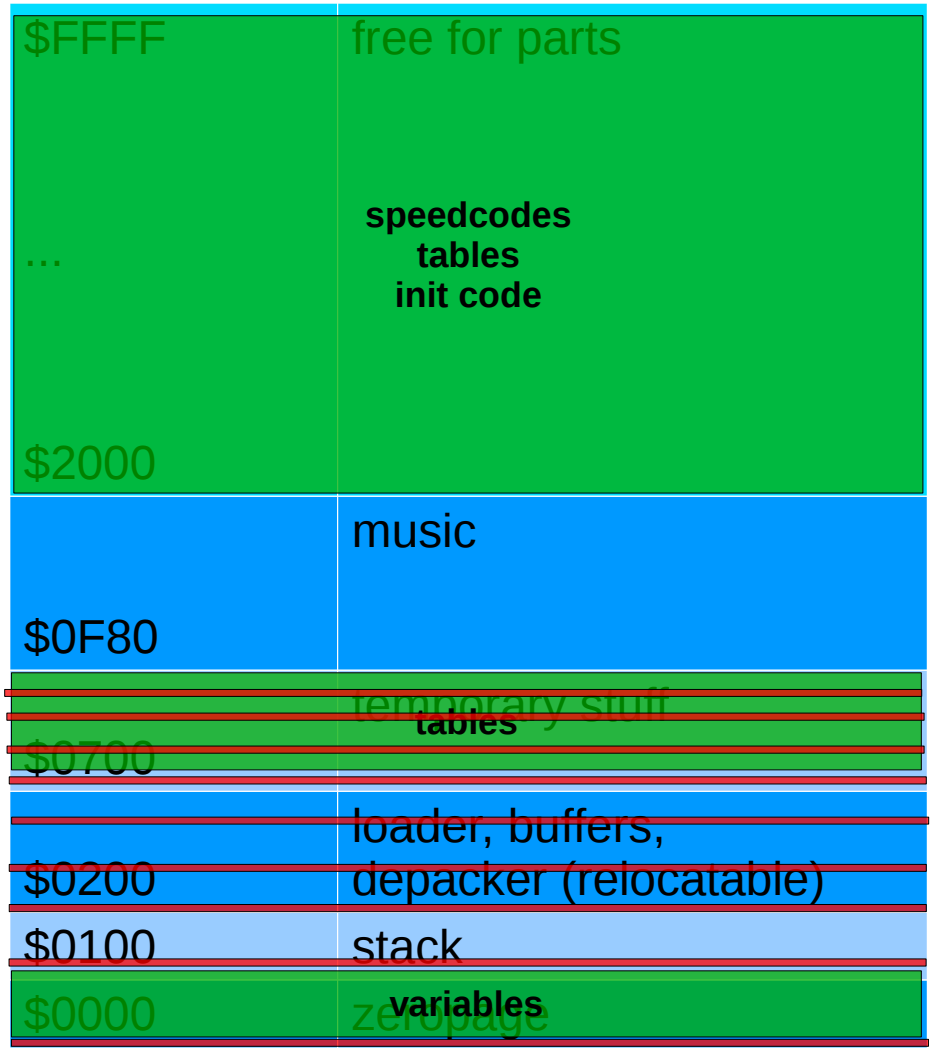
  EOL can still be annoying
- Master machine: Pentium MMX233 running DOS

Memory layout

$FFFF ... $2000	free for parts
$0F80	music
$0700	temporary stuff
$0200	loader, buffers, depacker (relocatable)
$0100	stack
$0000	zeropage

Memory layout (Rekmap)

Memory layout (Rekmap)



Loading

Custom load routines

- Serial(!), asynchronous(!!) protocol [rant, rant]
- demos usually use 2bit + ATN
- you need custom code in the drive

  Some demos use the drive for calculations
- was cool to have your own loader, but lots of them crashed
- a few survived and are commonly used these days
- Here, our DreamLoad is used

  Special feature: Can also run from HD and MMC

You want good compression to minimize loading.

Packing

Pimped exomizer (cross, of course)

- best packer around

  can take surprisingly long to pack 40KB
- depacking too slow for fast paced demos
- custom depacker, faster but nearly twice as large (500 byte)
- tweak the packer: use literals if the gain is not at least 'n' bit
- tweaked the parameters for every part

Packing

Pimped exomizer (cross, of course)

- best packer around

  can take surprisingly long to pack 40KB
- depacking too slow for fast paced demos
- custom depacker, faster but nearly twice as large (500 byte)
- tweak the packer: use literals if the gain is not at least 'n' bit
- tweaked the parameters for every part

A priori knowledge (manual)

- know your data
- find symmetries
- use delta-packing
- derive tables from other tables ...

Linking

Putting it all together

- usually parts are developed independently
- Linking means
  - Proper init/clean exit
  - rewrite code generators
  - Sync to music
  - Find/create spot to load/depack next part
  - assign resources (memory!)
  - don't crash
  - don't create visual/audial glitches
  - don't be boring
- everybody hates it
- no fame involved, although it *is* pretty hard

Syncing

Music is the reference

- Music player is called once per frame (usually)
- a simple counter is used for synchronization
- loading times are variable since drives are mechanic!
- don't even think about the time-of-day clocks, too coarse
- music players use "instruments" internally so you can do something on e.g. every drum

Showtime again

Now with pauses and comments

Thanks!

ninja@the-dreams.de