Ode to a childhood dream

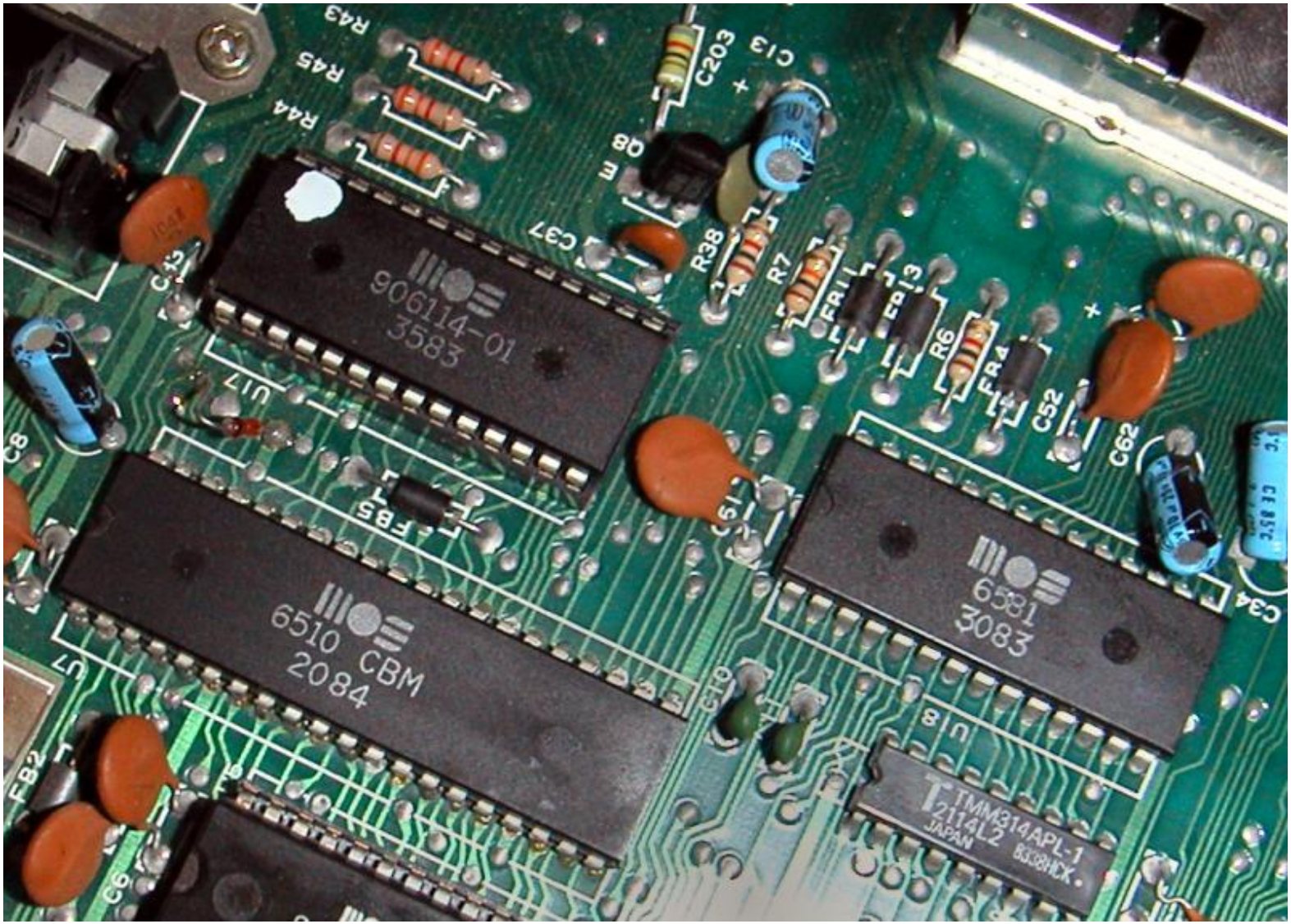
Cross-compiling GNU epsilon to the Commodore 64

Luca Saiu
positron@gnu.org
http://ageinghacker.net

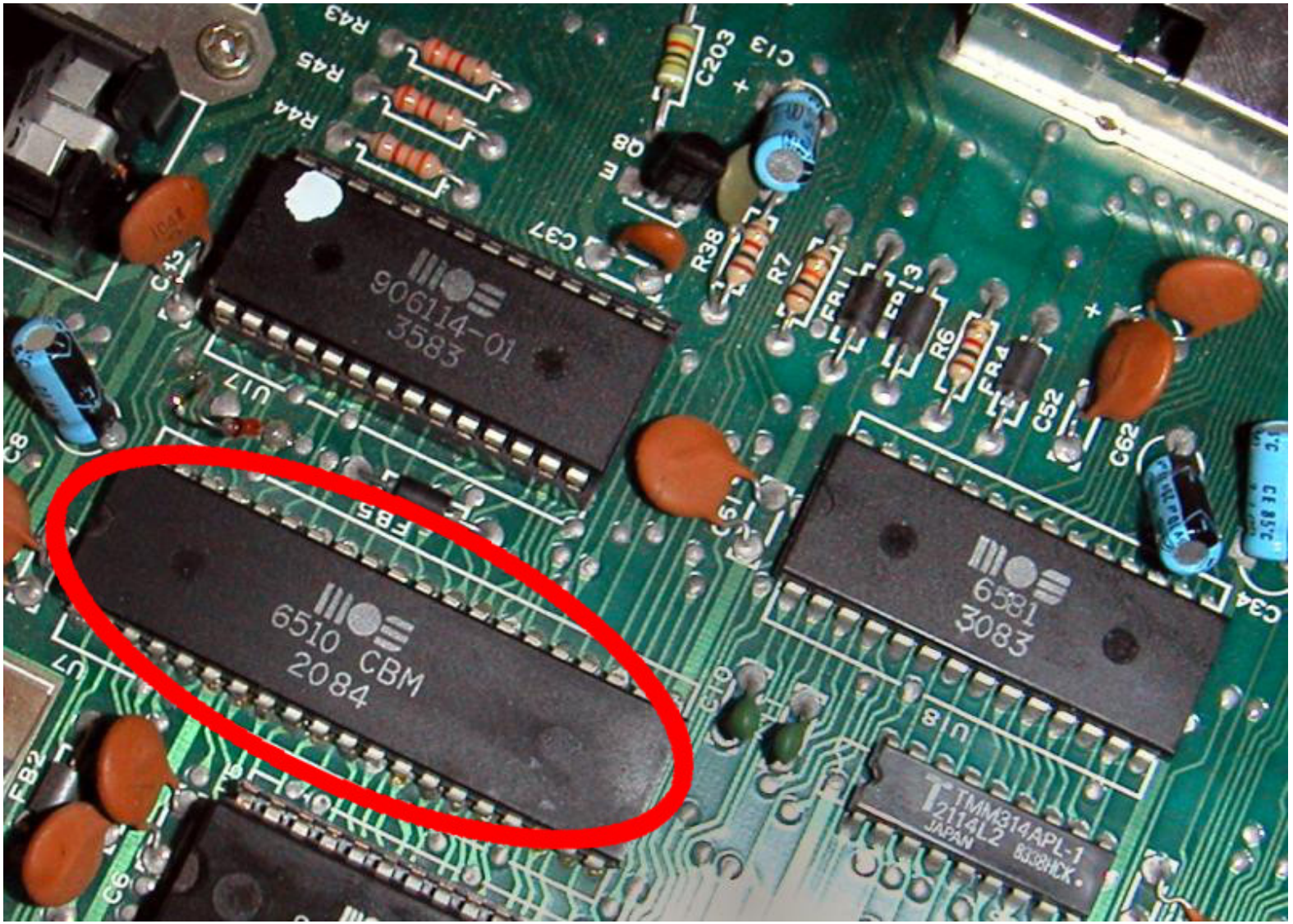
GNU Project

GNU Hackers' Meeting 2014
Technische Universität München — Garching, Germany
August 16th 2014

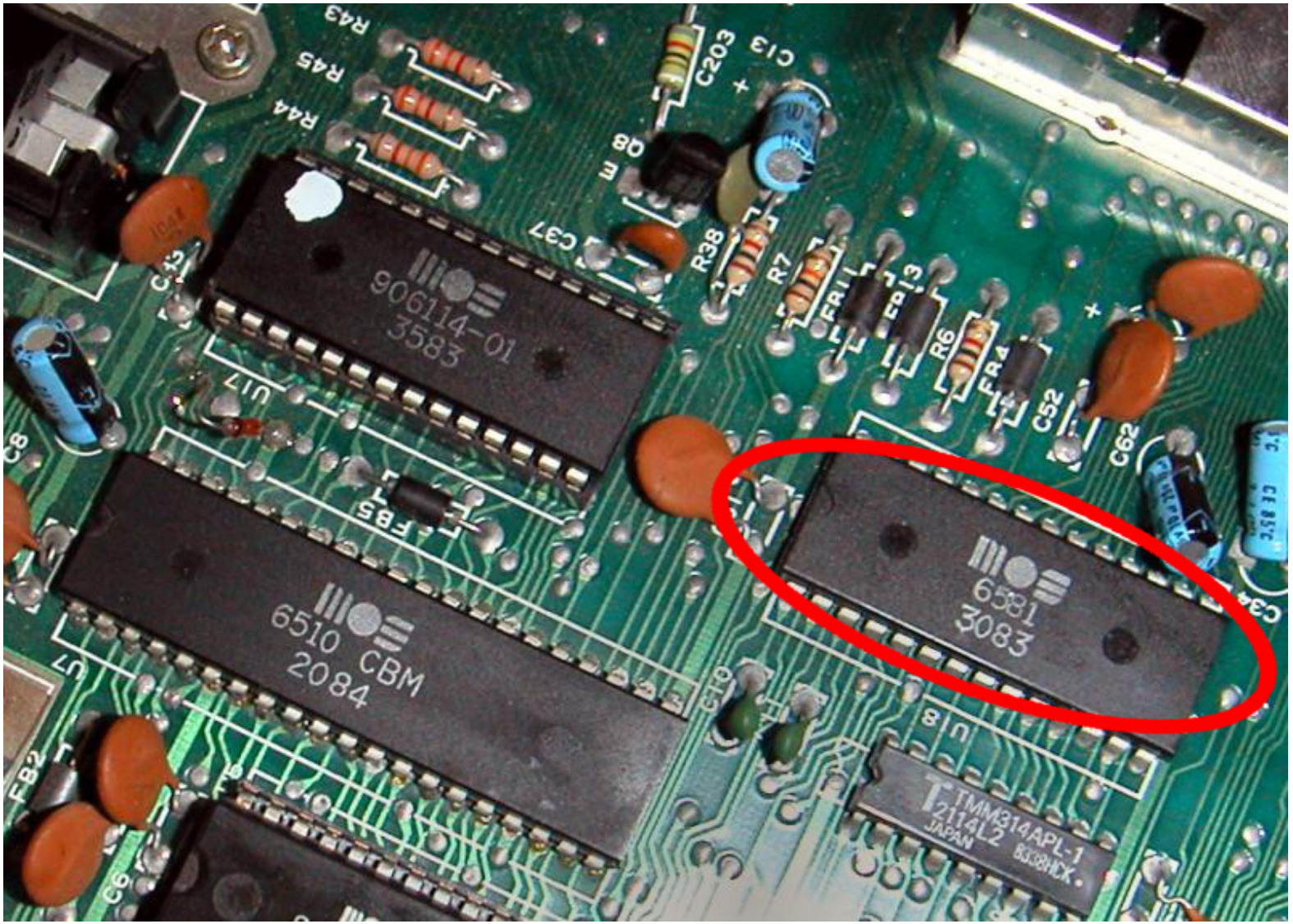
The Commodore 64 hardware — overview



The Commodore 64 hardware — overview

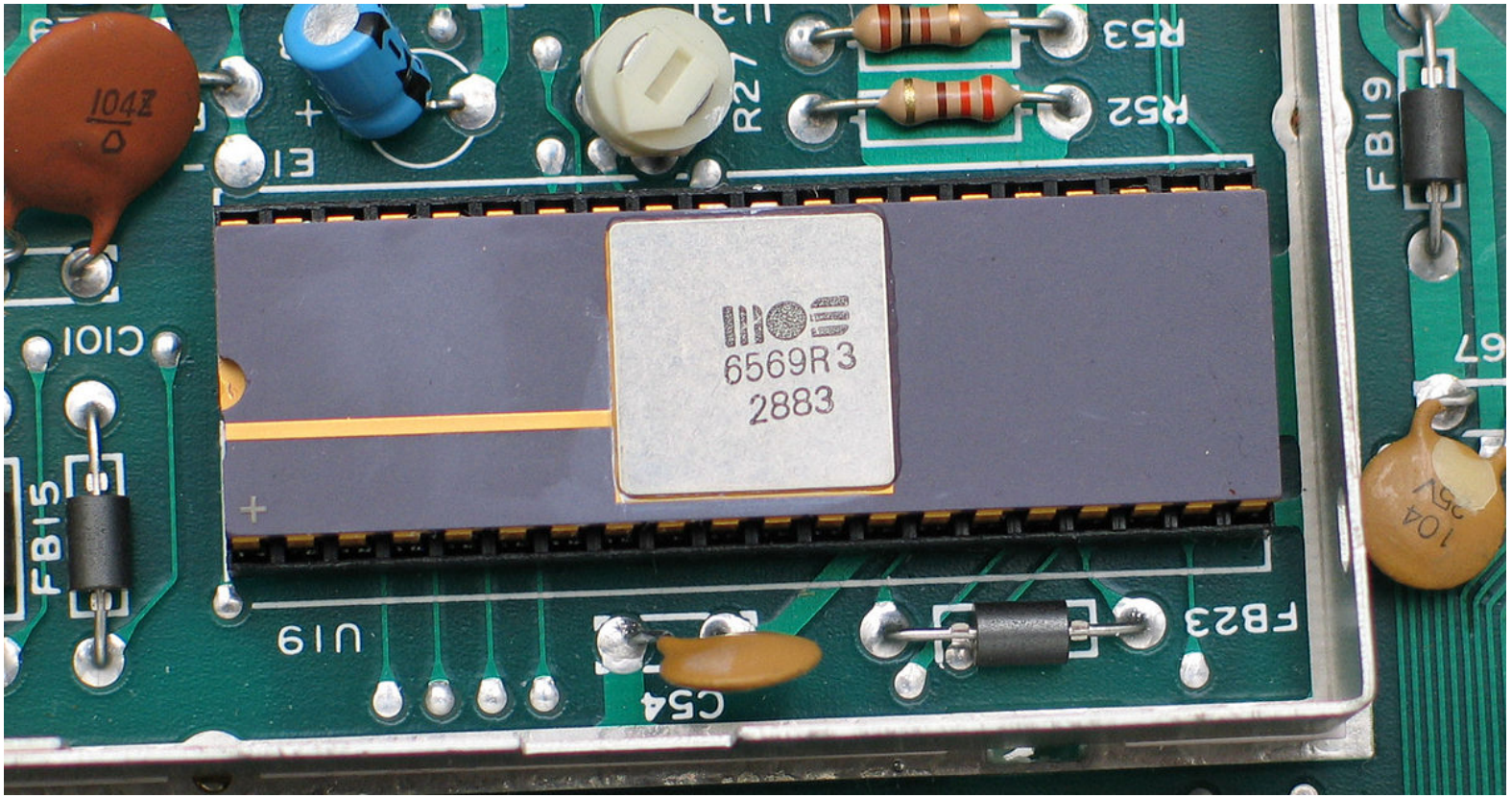


The Commodore 64 hardware — overview



The Commodore 64 hardware — overview

The VIC-II graphics chip



The Commodore 64 hardware — CPU

The CPU is a MOS 6510, essentially identical to the 6502.

A simple design from the late 1970s:

- 8-bit;
- accumulator CPU (A, two index registers X and Y, flags);
- no multiplication, no division, no floating-point;
- Low frequency:
  - 0.985 MHz (PAL version);
  - 1.023 MHz (NTSC version);
- 16-bit address bus:
  - 64KB address space, and 64KB RAM
  - clunky “zeropage” addressing modes, unsuited to pointers;
- memory-mapped I/O
  - (with bank switching);

The Commodore 64 hardware — multimedia

- SID audio chip (waveform), very nice for its time;
- VIC-II video chip:
  - hardware sprites (good !);
  - no linear framebuffer (very painful).
  - addresses 16KB out of the main 64KB memory
    - bank switching: any one of bank [0, 3].

The Commodore 64 hardware — other hardware

Quite reasonable keyboard (the computer is all in the keyboard).

Some hardware extensibility:

- Video interface: TV or dedicated monitor
- User port
- Memory-mapped ROM cartridges
- joystick, mouse, modem, printer
- secondary storage:
  - floppy drive
  - audio cassette interface

Motivation

Why all of this?

The Commodore 64 — how it “felt”

Commodore 64C (released 1986 — functionally identical to the 1982 model, updated motherboard making it cheaper to produce)

This is the same model I owned (but I had no floppy drive).

Commodore 64 system software

- KERNAL "operating system" in ROM:
  - memory-mapped, shadowing the last 8KB of RAM (unless disabled);
- "BASIC V2" interpreter in ROM:
  - memory-mapped, shadowing 8KB more;
  - very, very limited: PEEK and POKE;
  - BASIC starter;

[Little demo]

Commodore 64 system software

- KERNAL "operating system" in ROM:
  - memory-mapped, shadowing the last 8KB of RAM (unless disabled);
- "BASIC V2" interpreter in ROM:
  - memory-mapped, shadowing 8KB more;
  - very, very limited: PEEK and POKE;
  - BASIC starter;

[Little demo]

Commodore 64 system software

- KERNAL "operating system" in ROM:
  - memory-mapped, shadowing the last 8KB of RAM (unless disabled);
- "BASIC V2" interpreter in ROM:
  - memory-mapped, shadowing 8KB more;
  - very, very limited: PEEK and POKE;
  - BASIC starter;

[Little demo]

Extensible programming language

GNU epsilon:

- Extremely small core language ε_0
- Powerful extension capabilities
  - automatically rewrite programs into ε_0
  - only have to compile ε_0!
- Lispy high-level "personality" (but untyped)

Very simple, but little documented:

- Tutorial on my blog;
- café reports.

[Little demo]

Extensible programming language

GNU epsilon:

- Extremely small core language ε_0
- Powerful extension capabilities
  - automatically rewrite programs into ε_0
  - only have to compile ε_0!
- Lispy high-level "personality" (but untyped)

Very simple, but little documented:

- Tutorial on my blog;
- café reports.

[Little demo]

Extensible programming language

GNU epsilon:

- Extremely small core language ε_0
- Powerful extension capabilities
  - automatically rewrite programs into ε_0
  - only have to compile ε_0!
- Lispy high-level "personality" (but untyped)

Very simple, but little documented:

- Tutorial on my blog;
- café reports.

[Little demo]

Compiling epsilon

How I compile GNU epsilon:

- Build some desired global state to compile, by successive definitions;
- Visit the reachable data graph and generate (native) data.
- Translate procedure bodies (which are also data) into native code
  - At this point procedure bodies are ε_0 only — easy compilation!

Compiling epsilon

How I compile GNU epsilon:

- Build some desired global state to compile, by successive definitions;
- Visit the reachable data graph and generate (native) data.
- Translate procedure bodies (which are also data) into native code
  - At this point procedure bodies are ε_0 only — easy compilation!

Compiling epsilon

How I compile GNU epsilon:

- Build some desired global state to compile, by successive definitions;
- Visit the reachable data graph and generate (native) data.
- Translate procedure bodies (which are also data) into native code
  - At this point procedure bodies are ε_0 only — easy compilation!

6502 assembly: adding 16-bit numbers from epsilon stack

Macro parameters: `.fromslot1`, `.fromslot2`, `.toslot`.

```
;; Load the first operand low byte.
ldy #(.fromslot1 * 2)
lda (frame_pointer), y
;; Sum to second operand's first byte, store result.
clc ; Clear carry flag.
ldy #(.fromslot2 * 2)
adc (frame_pointer), y
ldy #(.toslot * 2)
sta (frame_pointer), y
;; Keep the carry bit and work with the high byte.
ldy #(.fromslot1 * 2 + 1)
lda (frame_pointer), y
ldy #(.fromslot2 * 2 + 1)
adc (frame_pointer), y
ldy #(.toslot * 2 + 1)
sta (frame_pointer), y
```

Demo

[demo]

This might be useful if you want to try yourself (c64 branch from epsilon git):

```
(e1:load "bootstrap/scheme/scratch-c64-demo.e")
(c (fio:write "foo" (i (fibo 10))))
(c (test-sprites-interactively))
```

```
acme -format cbm -o /tmp/q /tmp/q.a && x64 /tmp/q
```

Future work

- Make an official epsilon release (including Commodore 64 support);
- Don't compile procedure bodies as data;
  - ...this saves a whopping 30KB on my sprite example!
- Use zeropage "registers":
  - requires compiler work;
- Possibly make a real, elaborate Commodore 64 game.

Future work

- Make an official epsilon release (including Commodore 64 support);
- Don't compile procedure bodies as data;
  - ...this saves a whopping 30KB on my sprite example!
- Use zeropage "registers":
  - requires compiler work;
- Possibly make a real, elaborate Commodore 64 game.

Future work

- Make an official epsilon release (including Commodore 64 support);
- Don't compile procedure bodies as data;
  - ...this saves a whopping 30KB on my sprite example!
- Use zeropage "registers":
  - requires compiler work;
- Possibly make a real, elaborate Commodore 64 game.

Future work

- Make an official epsilon release (including Commodore 64 support);
- Don't compile procedure bodies as data;
  - ...this saves a whopping 30KB on my sprite example!
- Use zeropage "registers":
  - requires compiler work;
- Possibly make a real, elaborate Commodore 64 game.

The end

Ode to a childhood dream

Any questions?

Image credits

- `MOS_Technologies_large.jpg` by Nixdorf at the English language Wikipedia. *Creative Commons Attribution-Share Alike 3.0 Unported*.
- "MOS 6569R3 Video chip from a Commodore 64 main board (PAL version)" by Christian Taube. *This file is licensed under the Creative Commons Attribution-Share Alike 2.5 Generic license.*
- `https://en.wikipedia.org/wiki/File:C64c_system.jpg` by Bill Bertram. *This file is licensed under the Creative Commons Attribution-Share Alike 2.5 Generic license.*