Vierte Dimension

1/2026

Das Forth–Magazin

für Wissenschaft und Technik, für kommerzielle EDV,
für MSR–Technik, für den interessierten Hobbyisten

In dieser Ausgabe:

Dienstleistungen und Produkte

tematik GmbH
Technische
Informatik

Feldstraße 143
D–22880 Wedel
Fon 04103 – 808989 – 0
Fax 04103 – 808989 – 9
mail@tematik.de
`http://www.tematik.de`

Seit 2001 entwickeln und vertreiben wir unter dem Markennamen „Servonaut“ Baugruppen für den Funktionsmodellbau wie Fahrtregler, Lichtanlagen, Soundmodule und Funkmodule. Unsere Module werden vorwiegend in LKW–Modellen im Maßstab 1:14 bzw. 1:16 eingesetzt, aber auch in Baumaschinen wie Baggern, Radladern etc. Wir entwickeln mit eigenen Werkzeugen in Forth für die Freescale–Prozessoren 68HC08, S08, Coldfire sowie Atmel AVR.

Forth–Schulungen

Möchten Sie die Programmiersprache Forth erlernen oder sich in den neuen Forth–Entwicklungen weiterbilden? Haben Sie Produkte auf Basis von Forth und möchten Mitarbeiter in der Wartung und Weiterentwicklung dieser Produkte schulen?

Wir bieten Schulungen in Legacy–Forth–Systemen (FIG–Forth, Forth83), ANSI–Forth und nach den neusten Forth–200x–Standards. Unsere Trainer haben über 20 Jahre Erfahrung mit Forth–Programmierung auf Embedded–Systemen (ARM, MSP430, Atmel AVR, M68K, 6502, Z80 uvm.) und auf PC–Systemen (Linux, BSD, macOS und Windows).

Carsten Strotmann carsten@strotmann.de
`https://forth-schulung.de`

RetroForth
Linux · Windows · Native
Generic · L4Ka::Pistachio · Dex4u
Public Domain
`http://www.retroforth.org`
`http://retro.tunes.org`

Diese Anzeige wird gesponsort von:
EDV–Beratung Schmiedl, Am Bräuweiher 4,
93499 Zandt

Cornu GmbH
Ingenieurdienstleistungen
Elektrotechnik

Weitlstraße 140
80995 München
`sales@cornu.de`
`www.cornu.de`

Unser Themenschwerpunkt ist automotive SW unter AutoSAR. In Forth bieten wir u. a. Lösungen zur Verarbeitung großer Datenmengen, Modultests und modellgetriebene SW, z. B. auf Basis eCore/EMF.

KIMA Echtzeitsysteme GmbH

Güstener Straße 72 52428 Jülich
Tel.: 02463/9967–0 Fax: 02463/9967–99
`www.kimaE.de` info@kimaE.de

Automatisierungstechnik: Fortgeschrittene Steuerungen für die Verfahrenstechnik, Schaltanlagenbau, Projektierung, Sensorik, Maschinenüberwachungen. Echtzeitrechnersysteme: für Werkzeug– und Sondermaschinen, Fuzzy Logic.

FORTecH Software GmbH

Tannenweg 22 m D–18059 Rostock
`https://www.fortech.de/`

Wir entwickeln seit fast 20 Jahren kundenspezifische Software für industrielle Anwendungen. In dieser Zeit entstanden in Zusammenarbeit mit Kunden und Partnern Lösungen für verschiedenste Branchen, vor allem für die chemische Industrie, die Automobilindustrie und die Medizintechnik.

Konsulting
Klaus Kohl–Schöpe

Tel.: **(0 82 66)–36 09 862**
Prof.–Hamp–Str. 5
D–87745 Eppishausen

Professionelle Entwicklung für Steuerungs– und Messtechnik. Literaturservice, Datenblätter; auch alles zu Forthsystemen und MCUs.

Mikrocontroller–Verleih
Forth–Gesellschaft e. V.

Wir stellen hochwertige Evaluation–Boards, auch FPGA, samt Forth–Systemen zur Verfügung: Cypress, RISC–V, TI, MicroCore, GA144, SeaForth, MiniMuck, Zilog, 68HC11, ATMEL, Motorola, Hitachi, Renesas, Lego ...
`https://wiki.forth-ev.de/doku.php/mcv:mcv2`

Titelbild: Zensus unter der Lupe AUTOR: M.Kalus

Quelle: Eigenes Werk (Lupenbild CC)

Alle Texte in diesem Heft wurden von den Autoren vor Veröffentlichung gegengelesen und die Abdruckerlaubnis gegeben.

Die Bilder in diesem Heft sind Werke der jeweiligen Autoren.
Wurden Bilder von anderen Autoren verwendet, ist dies am Bild angegeben.

Bildquellenverzeichnis:
David Levy, Photograph of an Osbourne 1, the first truly portable computer; https://commons.wikimedia.org/wiki/File:Osborne_1_open.jpg
Computer Disks — Osborne 1, 1982. Source: Museums Victoria; Copyright Museums Victoria / CC BY (Licensed as Attribution 4.0 International)
Joker.mg, B3eck–bochum, CC BY–SA 3.0 https://creativecommons.org/licenses/by-sa/3.0, via Wikimedia Commons https://commons.wikimedia.org/wiki/File:B3eck_bochum.total12.JPG#Licensing
Raenmaen, GFDL, https://commons.wikimedia.org/wiki/File:Bochum_Jugendherberge.jpg
Chatsam, Thermoskop des Gallileo; PD Wikipedia...MuséedesArtsetMétiersthermoscopedegalilée1592(cropped).JPG.
Dr. Manuel, Ole Römer (Gemälde);https://de.wikipedia.org/wiki/Datei:Ole_R%C3%B8mer_(Gem%C3%A4lde).jpg
Randall Munroe "Superstition" CC BY–NC 2. https://xkcd.com/3191/

Impressum

Name der Zeitschrift

Vierte Dimension

Herausgeberin

Forth–Gesellschaft e. V.
Postfach 1030
48481 Neuenkirchen
E–Mail: `Secretary@forth-ev.de`
`Direktorium@forth-ev.de`
Bankverbindung: Deutsche Skatbank
IBAN: DE27 8306 5408 0006 8751 14
BIC: GENO DEF1 SLR

Redaktion & Layout

Bernd Paysan, Ulrich Hoffmann
E–Mail: `4d@forth-ev.de`

Anzeigenverwaltung

Büro der Herausgeberin

Redaktionsschluss

Januar, April, Juli, Oktober jeweils in der dritten Woche

Erscheinungsweise

1 Ausgabe / Quartal

Einzelpreis

4,00 € + Porto u. Verpackung

Liebe Leser,

ihr habt rasch reagiert Ende letzten Jahres und eure Mitgliedsbeiträge nachgezahlt. So ist der Mitgliederstand der Vorjahre praktisch erreicht worden. Vielen Dank für euer Engagement in der Forth–Gesellschaft.

Es hat sich gezeigt, dass unser Verein weiter bestehen bleiben kann. Zwar nicht auf dem Niveau der Gründerzeit, da hatte die FG mal an die 200 Mitglieder, wenn die alten Listen stimmen. Doch dank euch trägt sich die FG nach wie vor, auch wegen der vielen Freiwilligen und ehrenamtlich Tätigen. **Kommt nach Ostern zur MV nach Bochum**, dort könnt ihr sie alle persönlich antreffen.

Im *Ergebnis des Zensus* findet ihr weitere Überlegungen dazu.

Hans Hübner hat gezaubert und das *Gründungsprotokoll* und noch andere hübsche Sachen von einer alten Diskette gerettet. Dabei ist auch Klaus Schleisieks *CASE* wiedergefunden worden, eine Technik, die sich eingebürgert hat.

Hans Eckes zauberte ebenfalls. Dabei kamen die Datentypen `structs` und `peripherals` aus dem Hut auf den Tisch in Burladingen 2025 und von dort ins *Project Forth Works*. Hier im Heft bekommt ihr den Appetit, euch das im *Forth–Wiki* näher anzusehen. Danke, Hans, für die Mühe.

Und auch Philip Zembrod scheint Magie zu verwenden, wenn er einen *Small–C–Compiler in Forth* in eine so kleine Maschine wie den C64 unterkriegt. Dass es dabei doch mit Know–how zugeht, wird ersichtlich, folgt ihr seinen Spuren durch den Linux–Editor in die *make–automatisierten Emulatoren.*

Rafael Deliano hingegen zaubert mit Hardware. Sein Mini–Terminal (Breadboard–Terminal) lockt den Betrachter auf Ausstellungen an, wie auch den Entwickler von Testsoftware in Laboren.

Und ich selbst habe diesmal auch etwas verfasst und das Format der *Kolumne* wiederbelebt. Da findet ihr was zum *Codeberg*, über *Fahrenheit und Celsius*, und über *Fledermäuse im Regen.*

Ich hoffe, das Heft gefällt. Es war recht zeitig im Jahr fertig dank eurer Einsendungen. Mir scheint, die Weihnachtsfeiertage 2025, mitten in der Woche, haben euch gutgetan.

So, und jetzt **kommt bitte alle zur *Jahrestagung 2026* nach Bochum**. Wer noch nicht angemeldet ist, hole das unverzüglich nach, bevor alle Plätze weg sind — die Rückseite bringt dich da hin.

Euer Michael

Die Quelltexte in der VD müssen Sie nicht abtippen. Sie können sie auch von der Web–Seite des Vereins herunterladen.
`http://fossil.forth-ev.de/vd-2026-01`

Die gem. Forth–Gesellschaft e. V. wird durch ihr Direktorium vertreten:

Ulrich Hoffmann Kontakt: `Direktorium@Forth-ev.de`
Bernd Paysan
Gerald Wodni

Wem gehört ein AI–generiertes Bild?

Hier Grok zu fragen, entspricht in etwa, einen Dieb zu fragen, wem die Beute gehört. Es ist nicht zu erwarten, dass dabei etwas Wahres und Ehrliches herauskommt: AIs trainieren sich massiv mit urheberrechtlich geschütztem Material. Entscheiden, wem das Bild tatsächlich gehört, wird daher nicht Grok, Gemini oder ChatGPT, sondern ein *Gericht*. Wahrscheinlich eines in den USA, denn dort fing gerade die erste Klage eines Titanen der Filmwirtschaft gegen einen Titanen der AI–Wirtschaft an:

`https://www.golem.de/news/star-wars-marvel-und-co-in-ki-systemen-disney-wirft-google-massive-urheberrechtsverletzungen-vor-2512-203185.html`

Und wenn es in den USA scheitert, könnte Disney auch versuchen, die deutsche Rechtsprechung zu nutzen.

Zunächst argumentieren die natürlich so, dass man durch entsprechende Prompts Teile ihrer Werke reproduzieren kann. Deshalb gehe ich davon aus, dass die Klage erfolgreich sein wird. Und dass sich daraus dann auch ein Graubereich ergeben wird, in dem zwar nicht eindeutig und direkt ein Werk zitiert wird, aber doch erkennbar davon abgeleitet wird, und schon das ist ein Problem. Die AI kann sich, da sie selbst kein „Schöpfer“ im Sinne des Urheberrechts ist, auch nicht auf Fair Use beziehen (das es ohnehin nur in den USA gibt, und nicht in Deutschland), denn ihr Eigenanteil an der Schöpfung ist, egal, wie viel sie rekombiniert und kreativ neu arrangiert, im Sinne des Urheberrechts immer null: Sie erzeugt neue Bilder aus vorhandenen, „gelernten“ Bildern. Da müsste man dann nachweisen, dass die Trainingsdaten nicht verunreinigt sind, also nur aus frei nutzbaren Werken bestehen. Damit lassen sich die Ergebnisse aber nicht erklären.

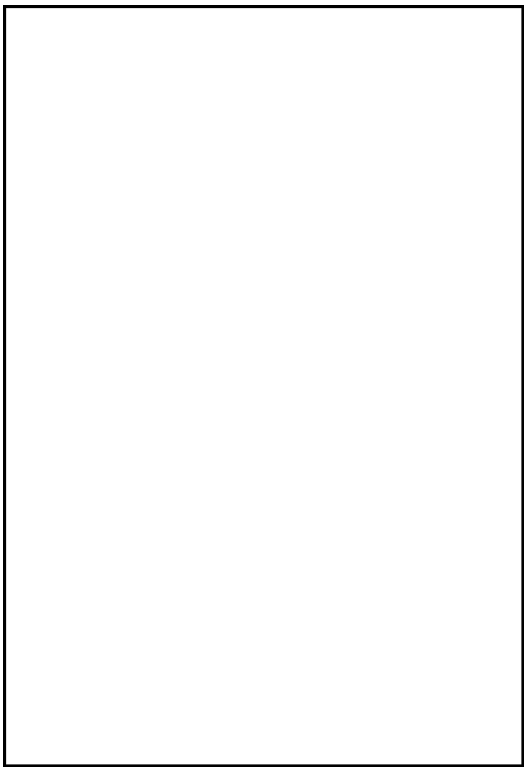
Beispiel–Prompt (für Grok und Gemini Text to Image):

> „Ein Laserschwertkampf zwischen Hua Mulan und Darth Vader vor der Kulisse des Auenlandes, also grüne, idyllische Wiesen mit Hobbit–Löchern. Beide Darsteller bitte mit Laserschwertern.“

Die Ergebnisbilder sind der Redaktion bekannt, das von Gemini gefällt mir besser.[1] Das Hua–Mulan–Kostüm bei Gemini ist nicht rot, und offensichtlich aus der chinesischen Monumental–Verfilmung von 2009 mit Zhao Wei als Hua Mulan entlehnt, nicht aus der Disney–Version. Das kann man sicher mit einem etwas anders formulierten Prompt ändern. Das kann jeder mal selbst ausprobieren, die Ergebnisse hängen natürlich auch vom Zufall ab.

Disney will hier offensichtlich Lizenzgebühren von Google und anderen AI–Firmen eintreiben. Das wird dann dafür sorgen, dass auch andere Firmen diesen Weg gehen, und erst, wenn alle drei Eigentümer dieser Immaterialgüter gültige Lizenzen vergeben haben, könnte man die Bilder ohne Bedenken veröffentlichen.

Deshalb gibt es für die VD eine **Keine–AI–Bilder–Politik**. Es gibt keine Möglichkeit, rechtssicher AI–Bilder zu bekommen, da die AI selbst kein Schöpfer im Sinne des Urheberrechts ist, das Bild also nicht als ihr Werk ausgeben und lizenzieren kann. Gleichzeitig ist es offensichtlich, dass die Trainingsdaten nicht nur aus frei verwendbaren Bildern bestehen. Die Beweislast der korrekten Nutzung liegt beim Verletzer. Das Ganze öffnet Tür und Tor für Abmahnungen ähnlicher Bilder. Bernd Paysan



Cover-Bild leider nicht nutzbar

Hacker's Delight

Die erste Auflage des Buches von Henry S. Warren, Jr. war bereits 2002 ein Erfolg [2], da sich was Rang und Namen hatte für das Buch einsetzte. Das Vorwort stammt von Guy Steele.

Die 2. Auflage ist im Umfang deutlich gewachsen [1]. Die Code–Schippsel sind in C oder Assembler unter Annahme einer 32–Bit–RISC–CPU geschrieben, ergänzt durch reichlich Illustrationen. Leichte Kost ist die Prosa des IBMers trotzdem nicht, hat wohl zu oft für akademische Publikationen gearbeitet.

Die Darstellung ist zwar knapp, aber gut strukturiert. Man wird leicht fündig, wenn man nach bestimmtem Thema sucht. Auf 490 Seiten wird einiges geboten. Der Schwerpunkt liegt neben Low–Level–Routinen bei Integer–Arithmetik. Rafael Deliano

[1] Warren „Hacker's Delight“ 2ed Addison Wesley 2013 ca. 52 EUR
[2] `en.wikipedia.org/wiki/Hacker's_Delight`

[1] Beim Autor auf Nachfrage einsehbar. (Hab die gesehen und kann bestätigen, was Bernd hier beschreibt. Der Sätza.)

Das FG–Gründungsprotokoll ist aufgetaucht

Wie Hans Hübner jenes Protokoll aus 1984 von Klaus' alter *Osborn–1–Diskette* gerettet hat, berichtet er euch in diesem Heft in einem kleinen Aufsatz. Das Protokoll haben sie, die sechs Gründer, gleich in Englisch verfasst, vermutlich um es in die USA an das FIG–Chapter zu schicken.

> "On April, 28th, sixteen people got together to get the ball rolling for a German FIG chapter. Three of us are 'old FIG hands' who once upon a time keyed the FIG listings into their machines. We are going to work as a chapter under the name of *Forth Gesellschaft* und we will publish a newsletter under the name of *Vierte Dimension*. The first issue will be mailed by the end of may and it will feature a translation of B. Ragsdale's interview from FD5/6[2]. Six working groups constituted themselves:

1. **Forth–83** We will be distributing H. Laxen's and M. Perry's *F83 model* and work is underway to translate the documentation into German.
2. **Leibniz** One of the major difficulties in gaining wide acceptance for Forth is constituted by the language barrier. Hence we are going to create a 'new' programming language under the name of Leibniz, which will have German names and follow the Forth–83 Standard semantically. Leibniz will be derived from the version of Forth which was developed by K. Schleisiek and it will be put into the public domain.
3. One working group will keep tabs of publications/products on Forth and compile an index.
4. **Newsletter** Vierte Dimension will be published approximately four times a year as a forum and communications vehicle for Forth users in Germany. Initial subscription is 23,– DM for individuals and 55,– DM for corporations.
5. **Fifth Dimension** This is a 'brainstorming' group which is going to envision the future developements of Forth towards friendly and not necessarily Forth–like user interfaces and higher level constructs. We will see what they are going to come up with. They want to take Forth as a basis on which to build more sophisticated structures.
6. **Operations/management** Horst–Günter Lynsche was elected secretary of Forth Gesellschaft and he will be responsible for the management aspects — keeping tabs of the subscribers, publishing Vierte Dimension etc.

> Meetings will be held every fourth saturday of a month at 15.00 hours. Ask Horst–Günter for the location and direct any questions to: ...[3]" Klaus Schleisiek

[2] Forth Dimension, Volume 5, Number 6

[3] [The old address is known to the editorial staff.]

Das F83 wurde damals wirklich weit verbreitet, ich denke, jeder Forthler kennt es noch.

Leibniz fand keine so weite Verbreitung, blieb irgendwie Horst–Günter's Sache; ich selbst hab das seinerzeit auf einer der ersten Euro–Forth–Tagungen in Aktion gesehen und war sehr beeintruckt von der damals sehr modernen, farbigen Fensteroberfläche — toll!

Die Literatursammlung wurde durch das Forth–Magazin transportiert und war weit verbreitet und fortgeschrieben. Heute liegt sie auf `https://wiki.forth-ev.de/doku.php/projects:litlist`.

Na ja, und den Newsletter haltet ihr ja grad in Händen. Das Projekt läuft nun im 42. Jahr! Habt ihr damals also gut gemacht, Leute!

Die „fünfte Dimension" hingegen ward nicht gekommen — oder weiß darüber jemand mehr als ich und mag berichten?

Das Management wanderte durch viele Hände in all den Jahren. Es erhielt unterwegs den Namen „Büro der Forth Gesellschaft", oder kurz *FG–Büro* und ist nach wie vor erreichbar per `secretary@forth-ev.de`, was auch eine erstaunliche Leistung durch all die Jahre ist. Liebe Gründer, das habt ihr gut gemacht damals.

Mit auf jener Diskette war übrigens noch ein Interview mit K. Schleisiek zu Thema der „Standardisierungskonferenz für FORTH–83" im Oktober 1982; auch das Standard–Kommitee gibt es heute noch. Und ein Interview mit Charles Moore, übersetzt ins Deutsche, das so beginnt:

> Frage: „Ich habe mir erzählen lassen, wie die *Forth Interest Group* anfing, aber ich konnte nie herausfinden, warum ..."
>
> CM: „Warum? Es gibt kein Warum. Es ist einfach so passiert und fing einfach an zu wachsen ..."

mk

SUBLEQ eForth

Richard James Howe hat neulich (2024/2025) „just for fun" eine 16–Bit–SUBLEQ–CPU geschrieben, in der eForth abläuft.

> "This project contains a working (self–hosting) Forth interpreter that runs on top of a SUBLEQ 16–bit machine. SUBLEQ machines belong to the class of One Instruction Set Computers, they only execute a single instruction but are still Turing Complete. The Forth system, specifically a variant of eForth, is provided as subleq.dec, passing this image to the tiny (~ 600 bytes) SUBLEQ C virtual machine allows you to run eForth on the machine. `https://github.com/howerj/subleq`"

SUBLEQ (Subtract and Branch if Less than or Equal to zero) ist eine Variante eines *One Instruction Set Computers* (OISC), einer Turing–vollständigen Rechnerarchitektur mit nur einer einzigen Instruktion.

SUBLEQ hat keinen einzelnen, eindeutig benannten Erfinder und keine dramatische Entdeckungsgeschichte. Es entstand als Konsequenz aus theoretischen Überlegungen zur Minimalität von Rechnerarchitekturen in den 1980er—1990er Jahren.

Die Idee basiert auf den Registermaschinen mit Inkrement/Dekrement und bedingtem Sprung, die Marvin Minsky in den 1960er Jahren beschrieben hat.[4] Aus solchen Minimalmodellen leitet sich ab, dass man mit einer einzigen Subtraktions–Operation und bedingtem Sprung auskommt. Die Addition lässt sich durch Subtraktion negativer Zahlen simulieren.

Der Name SUBLEQ und die populäre Beschreibung als OISC tauchten in den späten 1990er oder frühen 2000er Jahren in der Bewegung für esoterische Programmiersprachen (esolangs) auf, insbesondere auf Plattformen wie dem Esolang–Wiki.

Eine frühe formelle Erwähnung findet sich in einem Paper von 1988 („URISC“[5] von Mavaddat und Parhami), das eine ähnliche subtraktions–basierte Ein–Instruktions–Maschine beschreibt.

Lieber Carsten, danke für den Hinweis.

`https://en.wikipedia.org/wiki/One-instruction_set_computer#Subtract_and_branch_if_less_than_or_equal_to_zero`
`https://esolangs.org/wiki/Main_Page`

und Recherche mit Grok, Ergebnisse im Internet nachgesehen. mk

Kolumnisten

Wie ihr schon im Inhaltsverzeichnis gesehen habt, versuche ich mich gerade im Format der Kolumne, das zwischen dem klassischen Aufsatz eines Autors und dem Leserbrief oder einer knappen Meldung steht.

In unserem Forth–Magazin gab es das schon mal, von 2002 bis 2010, *Gehaltvolles — zusammengestellt und übertragen von Fred Behringer*. Fred, als Rentner, hatte die Zeit dazu und Spass daran auch. Nach seinem Tode hat das bis heute noch keiner wieder aufgegriffen.

Nun bin ich doch auch inzwischen Rentner mit Forth–Hobby. Also, was ist das?

> „Eine *Kolumne* ist ein regelmäßig erscheinender, kurzer Meinungsbeitrag in einer Zeitung, Zeitschrift oder Online–Medien, der meist von einem festen Autor, dem *Kolumnisten*, verfasst wird. Sie erscheint an derselben Stelle und spiegelt die persönliche Sichtweise des Autors wider, oft in Ich–Form und mit humorvoller oder polemischer Note.
>
> Der Begriff stammt vom lateinischen *columna* („Säule“) und bezieht sich ursprünglich auf die Spalte im Drucksatz — also die vertikale Textaufteilung einer Zeitungsseite.
>
> Heute steht „Kolumne“ vor allem für ein journalistisches Format, das unabhängig von aktuellen Nachrichten sein kann, aber Leserbindung fördert und Diskussionen anregt.“ [Wikipedia]

Na, dann versuche ich das auch mal. Doch etwas anders, als Fred es gemacht hat, sollen die einzelnen Themen im Inhaltsverzeichnis zu erkennen sein. Daher gibt es also gleich mehrere „Kolumnen“. Und unser bewährtes Aufsatz–Format passt da ganz gut, finde ich. Hoffentlich gefällts euch.

Dass ihr ebenfalls in diesem Format hier schreiben könnt, muss ich ja nicht extra erwähnen. mk

Kommentare im Forth–Quellcode

In *Gforth*, wie in den meisten ANS–Forth–Systemen, gibt es *keine* eingebaute Syntax für echte Block–Kommentare wie `/* ... */` in C oder Pascal.

Es gibt zwei Standardmöglichkeiten in Forth.

```
( ... )
```

Das ist der klassische, meist kurze Forth–Kommentar. Doch er kann in Dateien auch über *mehrere* Zeilen gehen, solange eine schließende) gefunden wird. Beispiel in einer `.fs`-Datei:

```
( Dies ist ein Kommentar
  über mehrere Zeilen.
  Er endet hier: )
: hallo ." Hallo Welt!" ;
```

Wichtig: Es *muss* ein Leerzeichen nach der öffnenden Klammer stehen (␣ und diese Art Kommentare sind *nicht* schachtelbar.

```
\
```

Kommentar bis zum Zeilenende, sehr praktisch für mehrzeilige Blöcke und auch die empfohlene Methode zum Auskommentieren größerer Code–Blöcke:

```
\ Langer Kommentar oder auskommentierter Code
\ Zeile 2
\ Zeile 3
```

[4] Der Hauptbeitrag Minskys zu diesem Thema findet sich in seinem klassischen Buch: Computation: Finite and Infinite Machines (Prentice–Hall, 1967).

In diesem Werk beschreibt und beweist er detailliert die Universalität von Registermaschinen mit minimalen Instruktionen. Es enthält die Beweise zur Simulation beliebiger Computer mit nur zwei Registern und einfachen Operationen. Das Buch ist ein Standardwerk zur Berechenbarkeitstheorie und diskutiert explizit minimale Maschinenmodelle, inklusive Varianten, die zu OISC–Ideen führen.

[5] Ultimate Reduced Instruction Set Computer

```
\ : alte-definition ... ;
\ Diese Definition wird ignoriert
```

Auch dabei gilt, nach dem *Backslash* muss ein Leerzeichen stehen \␣, weil es ein ausführbares Forthwort ist.

Was passiert da?

```
see \
: \
blk @
IF >in @ c/l / 1+ c/l * >in ! EXIT
THEN
source >in ! drop ; immediate
ok
```

Nun, `blk @` liefert den Indikator der Herkunft des Quellcodes, entweder ein Block–File–System oder eine Textdatei.

Falls es ein Block ist, wird der Input auf den Anfang der nächsten Zeile im Block gesetzt. Im anderen Fall wird der Input einfach an das Ende der aktuellen `source` gesetzt, die sich damit erschöpft hat.

Und das Wörtchen `immediate` sorgt schließlich dafür, dass diese Definition immer sogleich ausgeführt und nicht compiliert wird.

Der Rest der aktuelle Zeile wird also einfach übersprungen.

```
see (
: (
loadfile @ 0=
IF POSTPONE ( EXIT THEN
BEGIN
 >in @ 41 parse nip >in @ rot - =
 WHILE  refill 0=
  WHILE  warnings @
   WHILE  .\" warning: ')' missing" cr
   THEN
  EXIT
 THEN
REPEAT ; immediate ok
```

Hier läuft es etwas anders. Die öffnende Klammer `(` ist ein kleiner Parser.

Im Fall der Eingabe auf der Konsole wird die Ausführung zurückgestellt bis zum `<ret>` und dann erst passiert die Magie: Alles zwischen den Klammern wird ignoriert.

```
11 22 33 ( 44 55 ) .s <3> 11 22 33 ok
```

Im Fall eines Einstroms von Quellcode hingegen sucht `41 parse` die schließende Klammer.[6] Dabei wird der *input stream* solange fortgeschrieben, bis das Zeichen da oder das File erschöpft ist. Letzteres löst die Warnung aus.

[6] `decimal char ) . 41 ok`

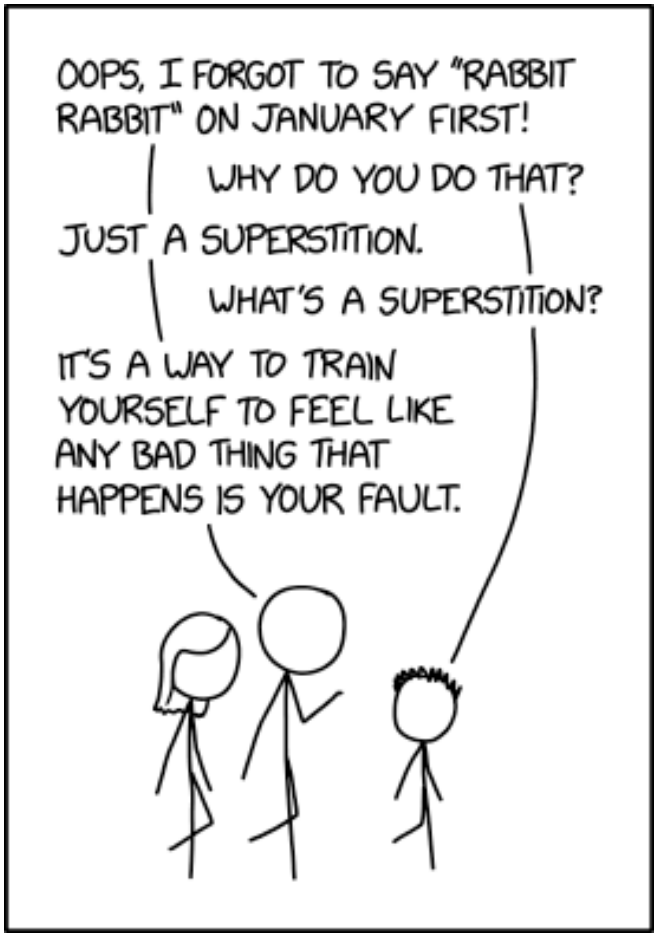
Interessant hier die verschachtelten `while`. Und vornedran der `exit` case, wie ihn Klaus Schleisiek seinerzeit einführte. mk

Neues Archiv für die „Vierte Dimension“

Es ist wieder komplett da, unser VD–Archiv! Zwar nicht, wie früher, unmittelbar einsehbar in unserem Wiki, sondern auf einer *eigenen Plattform*. Carsten Strotmann hat *Paperless–ngx* für euch angelegt und inzwischen auch alle Hefte eingelesen.

Der Zugang ist frei, im Sinne von „Freibier“, aber nicht öffentlich für jederman zugänglich. Ihr müsst euch euren Zugang geben lassen vom Forth–Büro oder von Carsten selbst. Warum das so ist in Deutschland, hat sich ja vermutlich inzwischen herumgesprochen.

Forth–Büro: `secretary@forth-ev.de`
VD–Archiv: `https://paperless.forth-ev.de/`
`https://docs.paperless-ngx.com/`



xkcd superstitious

Ergebnis des Zensus

Euer Kassenwart

Im letzten Heft riefen wir euch dazu auf, eure Mitgliedschaft zu erneuern und auch mal wieder Kontakt zum Forthbüro aufzunehmen. Sieht so aus, als sei Ersteres im Großen und Ganzen gelungen. Der Kontakt könnte noch besser werden.

94

Das ist die Anzahl der Mitglieder in der Forth–Gesellschaft e.V., Stand Januar 2026.

Jedenfalls, wenn man nur nach der schnöden Satzung geht, nach der nur die Beitragszahlenden gerechnet werden.

Wir haben in der Satzung keine Ehrenamtlichen vorgesehen, keine Veteranen oder sonstwie Assoziierten. Was eigentlich schade ist. Denn Forth–Seelen waren wir mal rund 200 nach der Gründung. Und sieht man sich so um in den Chats, sind da heute noch mehr, oder? Eigentlich sollten wir die Assoziierten einführen, nicht nur Euros zählen.

Theoretische Begründung für einen hybriden Mitgliedsbeitrag in einem Verein

Die Idee eines Mitgliedsbeitrags, der sowohl in monetärer Form (Euros) als auch durch praktische Arbeit geleistet werden kann, lässt sich elegant aus einer ökonomisch–soziologischen Perspektive begründen. Sie basiert auf dem Konzept des *sozialen Austauschs* und der *Ressourcenallokation*, inspiriert von Theorien wie der des Gemeinguts (Commons) von Elinor Ostrom und der Spieltheorie von John Nash.

In der Ökonomie wird der Beitrag zu einem Verein als Investition in ein kollektives Gut verstanden. Ein Verein ist ein *Kooperationssystem*, in dem Mitglieder Ressourcen einbringen, um gemeinsame Ziele zu erreichen (Forth ist Kultur). Geld (Euros) repräsentiert *abstrakte Wertschöpfung*, es ist liquide, standardisiert und ermöglicht dem Verein, externe Dienstleistungen zu erwerben wie Miete für Server oder Materialien. Praktische Arbeit hingegen ist *konkrete Wertschöpfung*. Sie schafft direkten Nutzen durch Zeit und Fähigkeiten, z. B. Organisation von Veranstaltungen oder Pflege von Anlagen wie Webseite, Wiki, Cloud, Teamwörks. Beide Formen sind äquivalent. Ein Mitglied, das 40 € zahlt, opfert Kaufkraft; eines, das 4 Stunden arbeitet, opfert Freizeit oder alternative Einkünfte. So könnten finanziell schwächere Mitglieder durch Arbeit kompensieren, ohne Diskriminierung — eine elegante Lösung für soziale Gerechtigkeit (Studenten und Rentner zahlen die Hälfte; dafür tun sie was im Verein, z. B. im Büro oder als Autoren des Forth–Magazins!

Vereine seien anfällig für das *Trittbrettfahrer–Problem* (Free–Riding), heißt es, bei dem Individuen von kollektiven Gütern profitieren, ohne beizutragen. Etwas, dass ich in der FG bisher aber nicht erlebt habe. Ihr?

Ein reiner Geldbeitrag könnte zu Passivität führen, während reine Arbeit zur Überlastung weniger führt. Das erleben wir im Verein allerdings laufend. Es sind wenige im und um das Büro herum und Carsten Strotmann hat schießlich fast allein dagestanden.

Vereinsarbeit fördert *soziale Bindung*, da sie Interaktion schafft und Identifikation mit dem Verein vertieft. Geld sorgt für Effizienz. Diese Dualität minimiert Konflikte und maximiert Kooperation — ein „hübsches“ Gleichgewicht, das wie ein symmetrisches Mandala wirkt: Beide Seiten ergänzen sich zu einem Ganzen.

Ein großes Problem liegt in der räumlichen Zersplittertheit. In den Gründerzeiten der FG gab es den Versuch, das durch *lokale Gruppen* zu kitten. Unsere Satzung zeugt noch davon. In Hamburg, Mannheim, München und Wuppertal gab es die. Die in Wuppertal ist nach einigen Jahren zerfallen — es braucht immer einen, der das vorantreibt. Zieht Derjenige weg und es findet sich keine neuer „Motor“, ist es aus.

Inzwischen haben sich neue Moglichkeiten ergeben durch das schnelle Internet mit seinen Browsern. Videokonferenzen und Chats überbrücken die räumliche Entfernung. Es finden sich wieder lose Gruppen zusammen. Bernd Paysan und Wolfgang Strauss sind zwei solche modernen Forth–Gastgeber auf freien Plattformen außerhalb von Facecbook und Co. Es bleibt das Problem der Gleichzeitigkeit solcher Online–Treffen. Man muss nicht verreisen, aber anwesend sein dann eben doch.

Nachhaltigkeit und Gemeinschaftsbildung

Die Praxis begründet den theoretischen Kern: Erfolgreiche Selbstorganisation basiert auf *diversen Beitragsformen*. Ein Verein als Commons profitiert von Vielfalt: Monetäre Beiträge sichern finanzielle Stabilität, praktische Arbeit baut Humankapital auf. Dies fördert Resilienz gegen externe Schocks (z. B. Pandemien).

Die Hybridform transformiert den Beitrag von einer Last zu einer investiven Handlung, persönliches Wachstum darin ist möglich, z. B. durch erlernte Fähigkeiten in der Vereinsarbeit. In einer Ära des Individualismus (kennst du ein Forth, kennst du *ein* Forth) stärkt dies Gemeinschaften ohne Zwang — eine poetische[1] Balance, die den Forth–Verein zu einem lebendigen Organismus macht.

Lange Rede, kurzer Sinn: Euch allen meinen herzlichen Dank für eure Beiträge — in der einen oder anderen Form.
mk

[1] Der Ursprung liegt im altgriechischen Verb *poiein*, was so viel bedeutet wie machen, verfertigen, erschaffen, hervorbringen.

Datenarchäologie — Wie man Osborne–Disketten heute einlesen kann

Hans Hübner

Klaus[1] *brachte einige Disketten mit, die er in den frühen 1980er Jahren mit einem Osborne 1 verwendet hatte und auf denen er interessante Dokumente vermutete. Da ich schon einige Erfahrung mit dem Einlesen alter Disketten habe, holte ich die dafür notwendige Hardware aus dem Keller und wir befassten uns mit der Problematik, während wir Kaffee und Lebkuchen verspeisten.*

Die Hardware / Einlesen der Daten

Zum Einlesen alter Disketten verwendet man heutzutage am besten einen *Flux–Leser*, der die magnetischen Flusswechsel ohne weitere Interpretation vom Diskettenlaufwerk einliest und an einen modernen Rechner zur Verarbeitung weitergibt. Ich verwende meistens ein *Greaseweazle*, ich habe auch mit der *Fluxengine* gute Ergebnisse erzielt. Das Greaseweazle wurde über USB mit dem Mac verbunden. Mit einem normalen Floppykabel haben wir ein gewöhnliches 5.25”–HD–Laufwerk angeschlossen. Mit diesem Setup kann man nun die Disketten einlesen, wobei die Greaseweazle–Software bereits die physikalischen Osborne–Formate beherrscht und uns somit direkt die Fluxdaten dekodiert. Wir erhalten also die Datenblöcke so, wie sie auf den Disketten abgespeichert wurden. Um eine normale Diskette mit 100 kB (einseitig, einfache Dichte) einzulesen, ist dieses Kommando zu verwenden:

```
gw read --format=occ1.sd
        --tracks=c=0-39:h=0:step=2
        diskette-1a.img
```

Dabei ist `occ1.sd` das Osborne–Format (es gibt auch `occ1.dd` für doppelte Dichte). Die Angabe `step=2` ist notwendig, um 40–Spur–Formate mit einem 80–Spur–Laufwerk einzulesen. Bei fast allen Disketten funktionierte dieses Verfahren einwandfrei. Eine sperrte sich, daher haben wir die Fluxdaten ohne Interpretation für eine spätere Analyse ausgelesen. Das entsprechende Format ist `raw.250`.

Foto: David Levy

Abbildung 1: Osborne 1

[1] Klaus Schleisiek

Auswerten der Daten

Die Image–Dateien der Disketten von Klaus enthielten teilweise CP/M–Dateisysteme, teilweise enthielten sie FORTH–Blöcke. Letztere lassen sich einfach so mit einem FORTH benutzen, das Block–I/O noch beherrscht. Um Dateien aus den CP/M–Dateisystem–Images zu extrahieren, habe ich *cpmtools* verwendet. Die passende Formatdefinition ist

```
diskdef osb1sssd
  seclen 256
  tracks 40
  sectrk 10
  blocksize 2048
  maxdir 64
  skew 2
  boottrk 3
  os 2.2
end
```

Sie muss im `diskdefs`–File für die cpmtools–Installation hinterlegt sein (bei mir ist das die Datei `/opt/homebrew/share/diskdefs`). Mit `cpmls` kann man nun die Dateien in einem der Images anzeigen:

```
% cpmls -f osb1sssd diskette-1a.img
0:
4dim.txt
```

Das Image enthält eine Datei mit dem Namen `4dim.txt` im Benutzerbereich `0`, die nun mit `cpmcp` ins lokale Dateisystem kopiert werden kann:

```
% cpmcp -f osb1sssd diskette-1a.img 0:4dim.txt .
% ls -l 4dim.txt
-rw-r--r-- 1 hans staff 44544
           Dec 7 11:10 4dim.txt
```

Konvertierung von WordStar–Dateien

Klaus hat seine Texte mit WordStar geschrieben, und diese Dateien lassen sich nicht ohne weiteres mit modernen Textverarbeitungen öffnen. Die Formatierungsmöglichkeiten von WordStar waren aber ohnehin beschränkt, daher bot es sich an, die Formatierungssteuerzeichen einfach zu entfernen. Weiterhin muss das Bit 7 bei einigen Zeichen gelöscht werden, weil es von WordStar für die Markierung weicher Umbruchpositionen verwendet wird, und die

7–Bit–Umlaute müssen nach UTF–8 konvertiert werden. Ich habe dazu alle WordStar–Dateien in ein temporäres Verzeichnis kopiert und dort dieses Perl–Skript laufen lassen:

```
#!/usr/bin/perl

# Entfernen der Wordstar-Formatierungen
# aus allen Dateien im aktuellen Verzeichnis
# und Umwandlung nach UTF-8.

foreach $file (<*>) {
    next if (! -f $file or $file =~ /\.cnv/);
    open OUTFILE, ">$file.cnv";
    open INFILE, "<$file";
    while (<INFILE>)
    {
        tr [\200-\377] [\000-\177];
        s/\[/Ä/g;
        s/\\/Ö/g;
        s/\]/Ü/g;
        s/{/ä/g;
        s/\|/ö/g;
        s/}/ü/g;
        s/@/§/g;
        s/~/ß/g;
        s/[\000-\011\013-\037]//g;
        print OUTFILE $_;
    }
    close INFILE;
    close OUTFILE;
    print "  Read $file, wrote $file.txt ...\n";
}
```

In den resultierenden Dateien muss man teilweise noch ein paar Müllzeichen am Anfang editieren, aber das war mit den paar Texten rasch erledigt. Es gibt angeblich auch fertige Werkzeuge, die WordStar–Dateien einlesen und konvertieren können, aber ich habe nichts gefunden, das auf der Kommandozeile einfach so funktionieren würde.

Links

`https://github.com/keirf/greaseweazle` Tools for accessing a floppy drive at the raw flux level.

`https://cowlark.com/fluxengine/index.html` The FluxEngine hardware is a very cheap USB floppy disk interface capable of reading and writing exotic non–PC floppy disk formats.

Anmerkung

In der aktuellen Ausgabe des Forth–Magazins findet ihr gleich zwei der ausgegrabenen Fundstücke: das klassische `CASE`–Statement, wie es Klaus Schleisiek seinerzeit vorgeschlagen hatte sowie das Protokoll der Gründungssitzung der Forth–Gesellschaft. mk

Museum Victoria

Abbildung 2: Computer Disks — Osborne 1, 1982

Structs und Peripherals

Projekt Forth Works Kolumne

Hans Eckes *stellte auf der Forthtagung in Burladingen 2025 diese Datenstrukturen vor, um den vielen Adressen aller Register Herr zu werden, mit denen MCUs wie ein Arm Cortex–M7 heute*[1] *daherkommen. Er schrieb den Quelltext auf SwiftForth 3.12 und hat ihn nun auch im PFW veröffentlicht. Sein Quelltext ist reich kommentiert und ein kompletter Aufsatz zu dem Thema.*

> „`Structs` gruppieren zusammengehörende Variablen und verbessern so Lesbarkeit von Programmcode. Priorität des weiter unten stehenden Codes ist das Schreiben von lesbarem Quelltext, nicht ein möglichst forthiges Herangehen ans Thema ‚Struct'.
>
> `Peripherals` sprechen Peripheriebausteine an, haben ansonsten aber das gleiche Innenleben wie Structs ... Im Prinzip muss `IS-STRUCT?` nur an die Stelle hinter die Abfrage, ob das neue Wort vielleicht eine Zahl war, eingefügt werden."

So beginnt er und damit ist auch schon klar, wo die Reise hingeht. Der Forth–Parser lernt einen weiteren Delimiter — den *Punkt*. Beispiel:

```
SETTINGS.win.position.top
```

Hans taucht euch also tief ein in den Compilerbau. Im klassischen Forth ist der Parser einfach gehalten. Ein `WORD` im Quellcode reicht bis zum nächsten Leerzeichen. Diese kurze Zeichenkette wird im Dictionary aufgesucht und das Wort ausgeführt. Gibt es das Wort nicht, wird noch untersucht, ob es eine `NUMBER` sein könnte, die dann auf den Stack kommt — das ist die *runtime*. Und dann gibt es bekanntlich einen 2. Zustand der Forth–Engine, die *compiletime*. Dabei wird das `WORD` oder die `NUMBER` als neue Struktur im Dictionary angelegt — ein neues Forthwort ist entstanden. Und ist es weder das eine noch das andere, wird eine Fehlermeldung ausgegeben. In Gforth z. B:

```
qwe
:1: Undefined word
>>>qwe<<<
Backtrace:
$75D208007A68 throw
$75D20801DDF0 no.extensions
$75D208007D28 interpreter-notfound1
```

IS-STRUCT?

`Nun ahnt ihr schon: IS-STRUCT?` wird aufgerufen, wenn das Wort nicht im Dictionary gefunden wurde. Es wurde ins `INTERPRET` — genauer gesagt ins `NOT-DEFINED` — von SwiftForth eingehängt[2]. Und das geht so:

```
' IS-STRUCT? UNDEFINED >CHAIN
```

Es enthält `STRUCT?` und prüft, ob das unbekannte Wort vielleicht zu einem *Struct* gehört.

Wie man die Structs gebraucht, schildert Hans ebenfalls im Vorspann seines Quelltextes. Da habt ihr beides zusammen, die Erklärung und wie es gemacht worden ist.

Ähnlich zu Forth gibt es *Defining Structs*, mit denen die Structmemberlisten (in ASCII) erzeugt werden und die damit angelegten `structs`, was die ausführbaren Worte sind.

Hier ist ein kurzes Beispiel zur Definition von Structs:

```
STRUCT Header \ definiere das Struct "Header"
  Int32 Command
  Int32 ID
  Int32 Len
  Int32 Checksum
END-STRUCT

STRUCT Inputconfig
  Header IHeader \ <-- baue "Header" in ein
                 \ anderes Struct ein
  Int32 PreAmp
  Int16 Filter
  Int16 unused
  Int32 Muxer
END-STRUCT
```

PERIPHs

Hans lieferte dann auch gleich praktische Beispiele, Wörter für den einfachen Zugriff auf die Peripherie von Mikrocontrollern. Sein Ziel war es auch hierbei, einen „lesbaren" Zugriff auf Peripherieregister zu bekommen.

Syntax am Beispiel UART:

```
0 UART3.CTRL !
UART3.DATA @
```

Für den Interpreter sind *PERIPHs* auch nur *Structs*, es gelten die gleichen Syntaxregeln wie bei den Structs.

[1] *2025*

[2] Achtung: SwiftForth prüft erst, ob der String eine Zahl ist, bevor es die \ UNDEFINED–Chain abarbeitet.

Hier ein Beispiel für einen Quad–Timer

`Group` sei die Registergruppe für *einen* Timer in der erwähnten MCU, die in Gruppen zu 4 Timern zusammengefasst werden (Quad–Timer). Und der Controller hat drei solche Quad–Timer. Entsprechend viele Register sind zu handhaben. Über diese neue Syntax ist der Zugriff auf das Register `HOLD` für Timer2 vom QuadTimer1 einfach so:

```
QTMR1.2.HOLD \ Achtung, 16-Bit-Register!
```

Jedenfalls, wenn diese Peripherie zuvor angelegt worden ist. Was der Fall ist im SwiftForth.

```
PERIPH GROUP    \ definiere ein Peripheral
  REG $00 COMP1 \ hinter REG steht der Offset
                 \ zur Startadresse
  REG $02 COMP2 \ hinter dem Offset steht dann
                 \ der Registername
  \ und alle Registernamen samt Offset landen
  \ in der Memberlist von GROUP ...
  REG $04 CAPT
  REG $06 LOAD
  REG $08 HOLD
  REG $0A CNTR
  REG $0C CTRL
  REG $0E SCTRL
  REG $10 CMPLD1
  REG $12 CMPLD2
  REG $14 CSCTRL
END-PERIPH

PERIPH TMR
  GROUP $00 0 \ <- Peripheral nesting
  REG $1E ENBL
  GROUP $20 1
  GROUP $40 2
  GROUP $60 3
  END-PERIPH
```

[3] Gibt es im Forth–Büro; Anschrift im Impressum.

```
401DC000 TMR QTMR1  \ Quad-Timer 1 liegt
                     \ an Adresse 0x401DC000
401E0000 TMR QTMR2
401E4000 TMR QTMR3



\ Lese nun im Quad-Timer1
\ das Hold-Register von Timer0

QTMR1.0.HOLD W@
```

Die ganze Magie dahinter entfaltet Hans im Quelltext [1].

Hilfsworte

Manchmal muss der Inhalt von einem Struct in einen anderen Struct kopiert werden. Beide Structs müssen dabei vom gleichen *Defining Struct* angelegt worden sein, dann geht das.

Zum Gebrauch von `STRUCT>` und `>STRUCT` siehe Quelltext.

Auch `ERASE-STRUCT` und `sizeof` helfen beim Anlegen und Erproben der Strukturen.

Link

[1] `https://wiki.forth-ev.de/doku.php/en:pfw:struct`
Solltet ihr noch kein Browserzertifikat[3] für diesen direkten Zugang zum Wiki haben, geht es auch über `https://forth-ev.de/`. Dort zum *Wiki* navigieren, dann *B — Projekt Forth Works* und dort bei den *Data Structures* ist auch `structs` zu finden. Die Botschaft von Hans Eckes dort:

> „Bitte verwende den Quelltext in diesem Sinne: Lesen, anpassen, verwenden, weitergeben.“

mk

cc64 — Small–C–Compiler in Forth

Philip Zembrod

Einen C–Compiler in Forth zu schreiben, ist zunächst alles andere als naheliegend. Und doch war es dieses Projekt, cc64 [1], das mich Ende der 80er auf dem C64 zu Forth brachte. Was ich dabei gelernt habe, war ein entscheidender Schritt auf meinem Weg zum Software–Engineer, und ich stelle heute fest, dass mir das Design von cc64 auch nach fast 40 Jahren immer noch gefällt. In einer losen Artikelfolge möchte ich das Projekt daher ein wenig genauer vorstellen. Dieser erste Artikel soll einen Überblick über das Design und die Überlegungen geben, die zu den verschiedenen Designentscheidungen geführt haben.

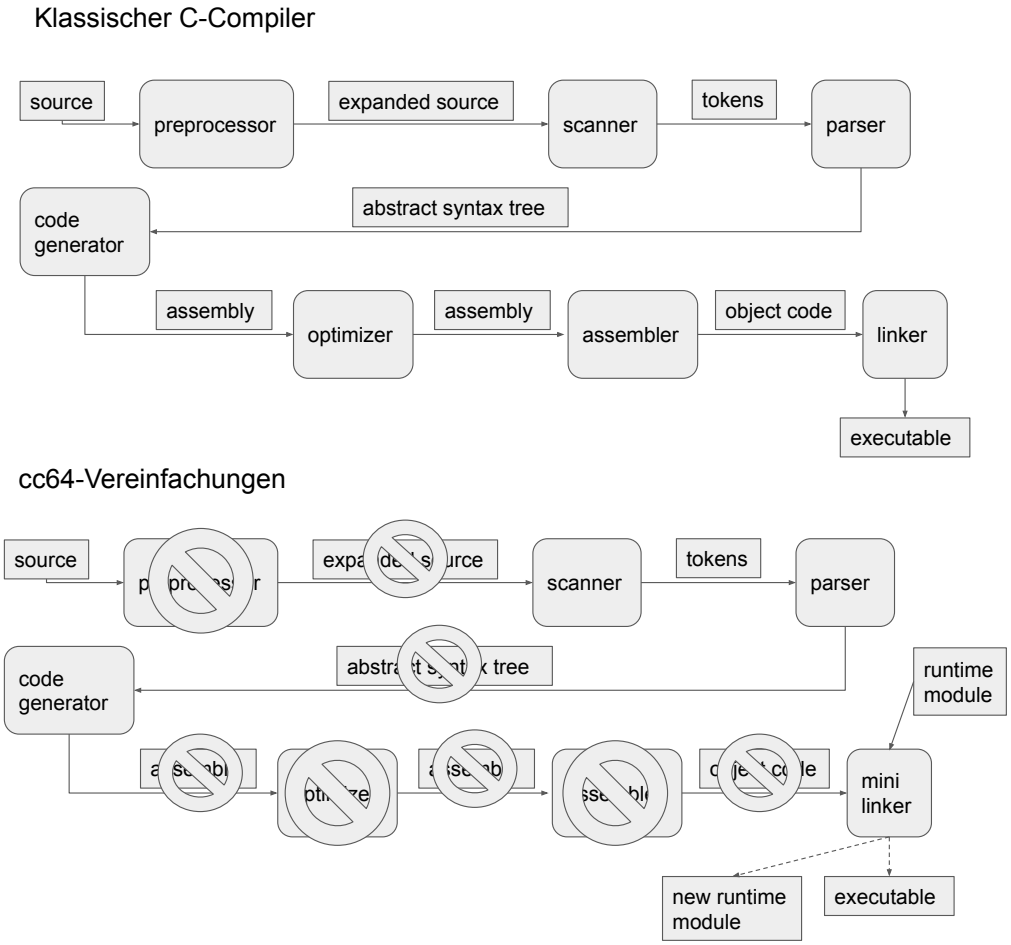


Abbildung 1: Oben: die Zwischenschritte, die Code bei einem klassischen C–Compiler von Source zum Executable durchläuft. Unten: was beim cc64 „wegvereinfacht“ wurde.

Warum in Forth

Die Entscheidung, diesen Compiler in Forth zu schreiben, war eng verbunden mit dem Grund, überhaupt einen C–Compiler schreiben zu wollen. Mein primärer Computer war 1989 immer noch ein C64, was er übrigens nicht zuletzt wegen dieses Compilerprojektes auch blieb, bis 1992 ein 486 bei mir einzog. Meine Suche nach einem guten Hochsprachen–Compiler war so alt wie mein C64 selbst. Ich hatte schon vorher auf der PDP11 unserer Schule Pascal kennengelernt, war begeistert von der Klarheit dieser strukturierten Sprache, und wollte so etwas auch zuhause haben, was sich als schwierig herausstellte. Das mit Abstand beste Pascal, das ich nach Jahren gefunden hatte, war Kyan–Pascal, welches aber recht langsamen Code erzeugte, Faktor 35 beim Sieb des Eratosthenes, verglichen mit handgeschriebenem 6502–Assembler. Ähnliches galt für einen C–Compiler von Data Becker mit einem Faktor von 33. Beides waren offenbar P–Code–Compiler, und ich wollte einen Compiler, der 6502–Maschinencode erzeugte.

Also begann ich mit eigenen Compiler–Experimenten. Eine meiner Lernquellen über Compiler war das berühmte Dragon Book [2]. Eine weitere Lernquelle und eine Inspiration war das Buch „C Tools“ [3] über den Small–C–Compiler von Ron Cain und James E. Hendrix, und ich beschloss, mich an einem Compiler für das gleiche, Small–C [4] genannte Subset von C zu versuchen: nur `int` und

`char` als grundlegende Datentypen, keine `structs` oder `unions`, keine `typedefs` oder `enums`, nur eine Ebene von Pointern, also keine Pointer of Pointer, und entsprechend auch nur eindimensionale Arrays. Und kein `goto`.

Mein erstes Experiment bestand aus Makros für Dirk Zabels ausgezeichneten Makroassembler *ASSI/M*, dessen Möglichkeiten von Stringbearbeitung von Makroparametern und bedingter Assemblierung viel ermöglichte. Semantisch kam ich damit schon sehr nah an Small–C heran, mit Code, der ähnlich schnell war wie Assembler. Ermutigt begann ich, in der „Sprache“ dieser Makrobibliothek eine Symboltabelle und einen Scanner für einen echten Compiler zu schreiben, und stellte fest, dass das Assemblieren dieser „Sprache“ entsetzlich langsam war; die Makros expandierten im Mittel jede Zeile Quelltext zu etwa 100 Zeilen. Noch problematischer: Der generierte Code war schnell, aber lang; es war absehbar, dass ich mit dem verfügbaren Speicher des C64 nicht auskommen würde.

Auf der Suche nach einer praxistauglichen Sprache zum Implementieren meines C–Compilers stieß ich in einer Anzeige, es muss in der 64‘er oder c‘t gewesen sein, auf *UltraForth*, wie das C64–VolksForth damals hieß. Ich bestellte es, probierte es aus und stellte fest: nur um einen Faktor 8 langsamer als Assembler beim Sieb–Benchmark, und sehr kompakter Code; das sah nach einer passenden Kombination aus. Also setze ich mein Compiler–Projekt noch einmal neu auf, diesmal in *Forth*.

Einfachheit & begrenzter Umfang

Ein wichtiges Designziel war von Anfang an Einfachheit. Das hatte, neben höheren Zielen wie Eleganz etc., ganz praktische Gründe: Einen C–Compiler zu schreiben, musste als Hobbyprojekt zeitmäßig neben mein Physikstudium passen, ich war als Programmierer ja Autodidakt und hatte nur begrenzte theoretische Kenntnisse über Compilerbau. Der Compiler sollte in die etwa 50 kB freien Speicher des C64 passen — nachladbare Overlays wollte ich vermeiden. Alles Gründe, die Aufgabe so einfach wie möglich zu gestalten. Zumal ich schon den Anspruch hatte, dass am Ende etwas herauskommen sollte, das praktisch nutzbar war, speziell, was den generierten Code betraf.

Die erste schon erwähnte Einschränkung, die ich deshalb machte, war die Reduktion des Sprachumfangs auf Small–C [4].

Eine weitere Einschränkung rührt daher, dass es auf dem C64 zu jener Zeit keinen Standardlinker und kein Format für relozierbare Objektdateien gab, jedenfalls kannte ich keinen. Auch war mir kein frei verfügbarer Assembler bekannt, den ich gut mit meinem Compiler hätte bündeln können. Beides selbst zu schreiben, hätte sicher meinen Zeitrahmen noch mehr gesprengt, als es das Projekt ohnehin schon tat. Also verlegte ich mich auf einen von frühen Versionen von Turbo Pascal inspirierten Modus: Der Compiler sollte aus den C–Sourcen direkt ausführbaren binären Code erzeugen, ohne einen Assembler oder symbolischen Linker als Zwischenstufen.

Schließlich machte ich beim Präprozessor Abstriche, den ich auf `#define`–Befehle nur für einfache Konstanten und auf `#include`–Befehle beschränkte. Später stellte ich fest, dass ich auch noch ein `#pragma` brauchte, aber das war es dann.

Abb. 1 zeigt, welche Teile eines klassischen C–Compilers unter anderem durch diese Einschränkungen und Abstriche bei cc64 einfach wegfallen.

1–Pass–Quelle zu Binary

cc64 ist ein 1–Pass–Compiler, er liest den Quelltext nur einmal ein und erzeugt daraus direkt (fast) ausführbaren Binärcode. Dieser Code muss absolute Adressen verwenden, da der 6502 nur extrem begrenzt relative Adressierung unterstützt (nur bei bedingten Sprüngen), und um diese Adressen direkt generieren zu können, muss die Startadresse des Codes schon direkt zu Anfang des Compilierens feststehen. cc64 erreicht das dadurch, dass der Binärcode an das Ende eines Runtime–Moduls angefügt wird, welches grundlegende Laufzeitroutinen enthält, wie z. B. Multiplikation oder Division. Mit dem festen Ende des Runtime–Moduls ist also die Adresse des ersten zu generierenden Codes von Anfang an gegeben.

Um einen Sprung mit absoluter Adresse zu generieren, braucht man die Zieladresse. Bei Sprüngen zurück, z. B. vom Ende einer Schleife zurück an den Schleifenanfang, und bei Aufrufen von bereits compilierten Funktionen sind die Zieladressen bekannt, der Code wurde ja bereits generiert. Aber wie sieht das aus bei Sprüngen vorwärts, z. B. hinter das Ende eines `if`–Zweiges oder einer Schleife, und bei Aufrufen von noch nicht definierten Funktionen? In beiden Fällen muss zu noch nicht generiertem Code gesprungen werden.

Sprünge vorwärts

Um Kontrollstrukturen, also Sprünge innerhalb einer Funktion, gut handhaben zu können, habe ich mich entschieden, den Binärcode einer Funktion vollständig im Speicher zu halten und erst auf Diskette zu schreiben, wenn die Funktion vollständig übersetzt ist. Dies begrenzt natürlich die maximale Größe, die eine Funktion haben kann, aber es macht es sehr einfach, während z. B. ein `while` compiliert wird, einen Sprung hinter das noch unbekannte Schleifenende zu erzeugen: Der Sprung erhält zunächst einfach die Zieladresse `0`, und nachdem der Schleifenkörper fertig generiert und die Zieladresse dahinter bekannt ist, wird diese Adresse einfach über die `0` geschrieben, die sich ja noch im Speicher befindet. VolksForth und viele andere Forth–Systeme machen das beim Compilieren von Kontrollstrukturen übrigens genauso.

Vorwärts–Aufrufe von noch nicht definierten Funktionen müssen anders gehandhabt werden. Auch hier wird der Aufruf zunächst mit `0` als Platzhalter für die Zieladresse

generiert. Diese Aufrufe mit Platzhalter sind es übrigens, weshalb der im Compile–Lauf erzeugte Binärcode nur „fast“ ausführbar ist. Das Auflösen der Platzhalter kann im Zweifelsfall erst passieren, nachdem alle Funktionen übersetzt wurden; deshalb merkt sich cc64 in einer Liste die Adressen, an denen die aufzulösenden Platzhalter liegen. Diese werden dann in einem minimalistischen „Linker“–Lauf gepatcht, der nichts weiter macht, als das Runtime–Modul und den compilierten Binärcode aneinanderzuhängen und eben dabei die Platzhalter der Vorwärtsreferenzen zu patchen.

Adressen von Variablen

Eine weitere Voraussetzung dafür, direkt lauffähigen Code zu generieren, ist, dass globalen und statischen Variablen absolute Adressen zugewiesen werden, sobald diese Variablen im Code definiert werden. Code, der auf diese Variablen zugreift, soll schließlich gleich mit der richtigen absoluten Adresse generiert werden. Um statische Variablenadressen sofort zuweisen zu können, legt cc64 diese ab dem Ende des verfügbaren Variablenspeichers absteigend an. Das Ende dieses Speichers liegt nämlich schon vor Beginn des Compilerlaufes fest, während der Anfang des Variablenspeichers in der Regel mit dem Ende des erzeugten Binärcodes zusammenfällt und damit erst nach dem Compilerlauf bekannt ist.

Damit der Compiler alle diese Adressen gleich zu Beginn der Übersetzung eines Programmes zur Verfügung hat, habe ich die spezielle Präprozessordirektive `#pragma cc64` eingeführt, die im Programm vor der ersten Zeile echten Codes stehen muss. `#pragma cc64` setzt alle nötigen Adressen sowie den Dateinamen des zu verwendenden Laufzeitmodules. In der Regel hat jedes Laufzeitmodul eine zugehörige Include–Datei, in der am Anfang das passende `#pragma cc64` steht, so dass ein Hauptprogramm nur ein `#include` des gewünschten Laufzeitmoduls benötigt, damit Compiler und Mini–Linker arbeiten können.

Mit dieser einfachen Schnittstelle zwischen Laufzeitmodul und Compiler und Mini–Linker ist es einfach, verschiedene Zielplattformen zu definieren und zu implementieren. Auch ROM–fähiger Code ist möglich.

Bibliotheken via Abkürzung

Bibliotheken, die gerade C ja braucht, sind natürlich eine Herausforderung, wenn kein regulärer Linker verfügbar ist. Wie gelangen die Funktionen, die in `stdio.h`, `ctype.h` oder `string.h` referenziert werden, tatsächlich in ein fertiges Binary, und wie kann der Compiler die Adressen der Funktionen vor der Übersetzung des Hauptprogrammes kennen?

Es zeigt sich, dass sich das Konzept des Laufzeitmoduls zu einem Bibliothekskonzept erweitern lässt, wenn man eine Einschränkung akzeptiert: Ein regulärer Linker bindet nur die Objektdateien aus einer Bibliothek in das lauffähige Binary ein, die von diesem auch tatsächlich gebraucht werden, und lässt unbenutzte Objektdateien weg. Unter Verzicht auf diese Flexibilität wurde Folgendes möglich:

Wenn cc64 ein Hauptprogramm übersetzt, das keine `main()`–Funktion enthält, dann erzeugt der Mini–Linker kein ausführbares Binary. Stattdessen werden das zugrundeliegende Laufzeitmodul und der neu compilierte Code zu einem neuen Laufzeitmodul gebündelt, auf das wiederum ein anderes Hauptprogramm aufsetzen kann. Die exportierten Symbole, speziell die Funktionen, werden in einer speziellen Syntax mit ihren Adressen in die Headerdatei des neuen Laufzeitmoduls geschrieben. Ein Programm, das mit diesem erweiterten Laufzeitmodul compiliert wird, kann jetzt diese exportierten Funktionen als Bibliothek nutzen.

Das Problem, dass ein gegebenes Binary mit diesem Bibliotheksmodell unter Umständen ungenutze Bibliotheksfunktionen enthält, lässt sich mit der Behelfslösung umgehen, ein angepasstes Laufzeitmodul für dieses Binary zu erzeugen mit nur den verwendeten Bibliothekssourcen.

Zukünftig könnte die *Schichtenarchitektur von cc64*, speziell die Schnittstelle von v–assembler (s. u.), eine Möglichkeit bieten, ein forth–gestütztes Format für relozierbaren Code zu entwickeln und damit flexibleres Linken zu ermöglichen.

Sourcenüberblick

Als Abschluss dieses Übersichtartikels will ich versuchen, einen Überblick über die Architektur zu geben, indem ich die wichtigsten Sourcen kurz beschreibe. Vergleiche auch Abb. 2.

`symboltable.fth`

enthält die globale und die lokale Symboltabelle. Beide teilen sich einen Speicherblock, wobei die globale Symboltabelle vom unteren Ende des Blocks nach oben und die lokale Symboltabelle vom oberen Ende nach unten wächst. Die globale Symboltabelle hat zusätzlich eine Hashtabelle für schnelleren Zugriff.

Das Interface der Symboltabelle hat Worte zum Setzen und Suchen lokaler und globaler Variablen sowie von Funktionen und Funktionsparametern, und außerdem Worte zum Öffnen und Schließen von Geltungsbereichen lokaler Variablen.

`input.fth`

liest zeilenweise die C–Sourcen, ruft den Präprozessor, bietet dem Scanner eine zeichenweise Sicht auf die Sourcen, und regelt das Öffnen, Schließen und Lesen von Include–Dateien.

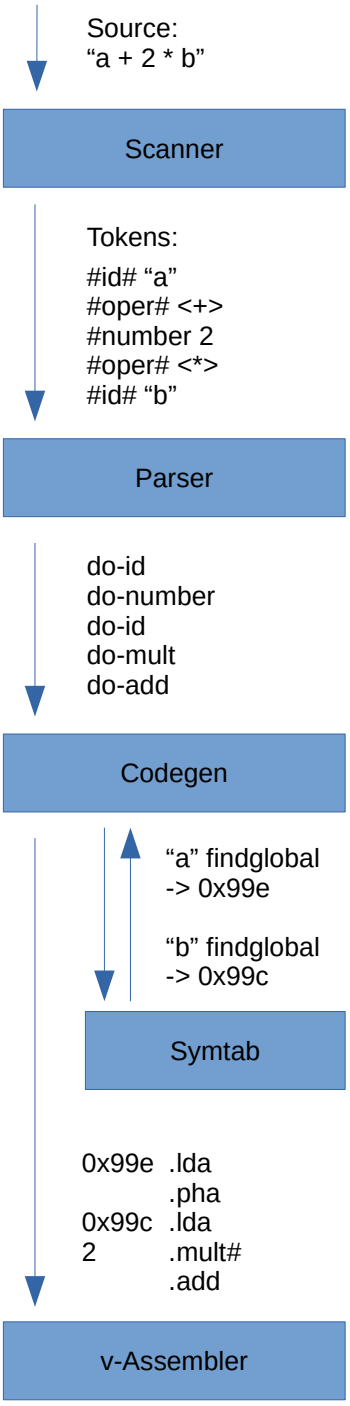


Abbildung 2: Am Beispiel eines einfachen Ausdrucks wird der Fluss des Codes vom Quelltext über eine Tokensequenz und eine Folge von Codegen–Aufrufen bis zum Aufruf der entsprechenden Assembler–Templates gezeigt. Erkennbar ist, wie der Parser nach der Punkt–vor–Strich–Regel den Infix–Ausdruck in Codegen–Aufrufe für stackbasierten Postfix–Code umsetzt, und wie auf v–Assembler–Ebene die Multiplikation mit dem konstanten Wert 2 umgesetzt wird: Die 2 wird nicht erst auf den Stapel geschoben, sondern direkt an `.mult#` übergeben, das den Akkumulator mit einem Immediate–Wert multipliziert, ohne den Stapel zu bemühen.

`preprocess.fth`

Der cc64–Präprozessor ist sehr einfach und ein wenig ein Hack. Er verarbeitet keine ganzen Quelldateien, sondern wird von `input.fth` jeweils für nur eine Quelltextzeile gerufen. Und er versteht nur die folgenden Direktiven:

`#include` öffnet, via `input.fth`, eine Include–Datei.
`#define` ist besonders hacky: Es erzeugt reguläre C–Konstanten in der globalen Symboltabelle, d. h., es führt keine Makro–Substitution durch.
`#pragma cc64` wählt und konfiguriert das zu verwendende Laufzeitmodul und setzt die absoluten Adressbereiche für Code und Variablen, damit der Codegenerator direkt lauffähigen Binärcode erzeugen kann, wie im Abschnitt „1–Pass–Quelle zu Binary“ beschrieben.

`scanner.fth`

Der Scanner, zuständig für die lexikalische Analyse, zerlegt den C–Quelltext in Tokens, d. h. in Schlüsselworte, Identifier, Zahlen, Operatoren und sonstige Zeichen. Sein Interface für den Parser besteht aus 4 Worten:

`thisword` liefert das aktuelle Token, ohne es zu „verbrauchen“

`accept` „verbraucht“ das aktuelle Token und geht weiter zum nächsten Token

`mark` merkt sich die aktuelle Tokenposition

`advanced?` prüft, ob seit dem letzten `mark` Tokens verbraucht wurden oder ob `thisword` immer noch auf dieselbe Stelle im Quelltext zeigt.

`parser.fth`

Der Parser, zuständig für die syntaktische Analyse, ist bei cc64 ein Top–Down–Parser nach dem Prinzip des rekursiven Abstiegs (recursive decent). Er besteht aus drei Teilen: für Ausdrücke, für Befehle, d. h. im Wesentlichen für Kontrollstrukturen, und für Definitionen.

Die Syntax von Sprachen wie C folgt den Regeln einer sogenannten kontextfreien Grammatik, und syntaktisch analysierter C–Quelltext hat die Struktur eines Baumes, genannt abstrakter Syntaxbaum. Dass cc64 direkt beim Parsen des Quelltextes fast ausführbaren Code erzeugt, bedeutet, dass der abstrakte Syntaxbaum nicht als Datenstruktur im Speicher materialisiert wird. Stattdessen werden die Daten jedes logischen Baumknotens sofort an den Codegenerator weitergegeben, der Syntaxbaum wird quasi beim Aufbauen sofort wieder konsumiert. Dies ist eine weitere Abkürzung, die cc64 nimmt und die auch in Abb. 1 dargestellt ist.

`codegen.fth`

enthält Logik zum Erzeugen von Code für C–Ausdrücke. Die Codegenerierung für Ausdrücke erwies sich als deutlich komplexer als die für Kontrollstrukturen, deswegen ein separates Modul.

Codegen ist parallel zum Parser für Ausdrücke aufgebaut. Viele Parserworte haben ein Partnerwort im Codegenerator. In der Regel ist ein solches Parserwort für eine bestimmte Grammatikregel zuständig, etwa für Addition. Wenn das Parserwort `sum` den +–Operator für eine Addition findet, ruft es das Codegen–Wort `do-add` auf, das mithilfe von Templates aus `v-assembler.fth` den Binärcode für diese Addition erzeugt.

Codegen berechnet auch konstante Ausdrücke und die konstanten Anteile von Ausdrücken, soweit sie zur Compilezeit bereits ermittelt werden können.

Code für Kontrollstrukturen ist wesentlich einfacher als Code für Ausdrücke und kann vom Parser direkt mittels `v-assembler`–Templates erzeugt werden.

`v-assembler.fth`

enthält die untere Ebene des Codegenerators und bietet als Interface so etwas wie eine virtuelle CPU mit einem 16–Bit–Akkumulator, implementiert mit binären 6502–Code–Templates, die mit VolksForths 6502–Assembler erstellt sind. Die virtuelle CPU „erbt“ den Hardware–Stack des 6502, der für den Stack–Code genutzt wird, in den C–Infix–Ausdrücke übersetzt werden. Außerdem gibt es ein virtuelles Stack–Frame–Register für einen Software–Stack, auf dem lokale Variable und Funktionsparameter abgelegt werden.

`minilinker.fth`

enthält den bereits erwähnten Mini–Linker. Parser & Codegen erzeugen zwei Dateien:

`%%code` enthält den erzeugten Binärcode, und `%%init` enthält die Initialisierungswerte für statische Variable. Laufzeitmodule enthalten entsprechend auch eine Codedatei (*.o) und eine Initialisierungsdatei (*.i). Wenn der Mini–Linker ein ausführbares Programm — mittels `main()`–Funktion — erstellt, werden alle diese 4 Dateien in einer einzigen Datei verbunden. Wenn eine neue Bibliothek erzeugt wird — ohne `main()`–Funktion — dann werden aus beiden Codedateien eine neue `o`–Datei und aus beiden Initialisierungsdateien eine neue `i`–Datei erzeugt.

In beiden Fällen werden die Vorwärtsreferenzen der Funktionsaufrufe aufgelöst, indem die Nullen, die als Zieladressenplatzhalter in Sprüngen stehen, deren Zielfunktion beim Compilieren noch nicht definiert war, durch die richtige Funktionsadresse ersetzt werden.

Beim Erzeugen von ausführbaren Programmen wird außerdem die Adresse der `main()`–Funktion und einige andere Adressen an definierten Stellen im Runtime–Modul eingetragen, damit dieses beim Start die statischen Variablen korrekt initialisieren und dann die `main()`–Funktion aufrufen kann.

Für eine neue Bibliothek wird außerdem eine Headerdatei (*.h) erzeugt mit der korrekten `#pragma cc64`–Direktive und mit Definitionen für alle Symbole, die die Bibliothek exportiert. Für diese Symbole nutzt cc64 eine Nicht–Standard–Syntax: Wird ein globales Symbol mit dem Operator `*=` initialisiert, wird der Initialisierungswert als absolute Adresse der Variable oder Funktion interpretiert. Ein Hack, der aber sehr gut funktioniert.

`invoke.fth`

enthält das Top–Level–Wort `cc`, welches Compiler und Mini–Linker aufruft und damit alles bündelt.

`shell.fth`

schließlich macht alles für den Benutzer zugänglich. Es definiert ein Vokabular mit einem Alias auf `cc` sowie mit einigen nützlichen Worten wie `dir`, `list` oder `help` und, sofern vorhanden, auch einem Alias zum Aufruf eines Editors. In diesem Vokabular befindet sich der Benutzer nach dem Start von cc64, es stellt also die Benutzeroberfläche dar.

Weitere Sourcen

Es gibt noch eine Reihe weiterer Sourcen, die z. B. Dinge wie Speicher– und Listenverwaltung, Stringverarbeitung und Stringtabellen, Dateiverwaltung, Fehlermeldungen und noch ein bisschen dies und das bereitstellen und auf die ich vielleicht in einem späteren Artikel zu sprechen komme.

Anmerkung

Ich betreibe momentan keinen aktiven C64–Arbeitsplatz. Die Entwicklung läuft komplett im Linux–Editor und auf make–automatisierten Emulatoren.

Für einen echten Aufbau habe ich aber noch zwei C64 mit 1541 und einen Dell 2001FP mit FBAS– und S–Video–Eingang; selten bei LCD–Monitoren.

Quellen

[1] `http://github.com/pzembrod/cc64`
[2] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, „Compilers, Principles, Techniques and Tools“ aka The Dragon Book, 1986, Addison–Wesley, ISBN 0–201–10194–7
[3] „Dr. Dobb‘s Journal C Tools“, 1986, Markt–und–Technik–Verlag, ISBN 3–89090–190–5
[4] `https://en.wikipedia.org/wiki/Small-C`

CASE — Ein ganz einfacher Fall[1]

Klaus Schleisiek

Forth hat in seinem Kern kein `CASE`*–Statement. Dies etwa nicht deshalb, weil es besonders schwierig wäre, oder weil man es vergessen hätte; nein, der Hauptgrund, warum es nicht im Kern enthalten ist, besteht darin, dass es einfach zu selten benötigt wird, so dass es sinnlos ist, es immer „mitzuschleppen" — ganz abgesehen davon, dass es grundsätzlich drei verschiedene Arten von* `CASE`*–Statements gibt und deshalb nach den murphischen Gesetzen im gegebenen Fall immer der falsche Typus zur Hand wäre.*

Im Folgenden werde ich die simpelste Art eines `CASE`*–Statements vorstellen, die ich kenne; die in all den Fällen eingesetzt werden kann, wenn z. B. in Abhängigkeit von einzelnen Keys von der Tastatur einige wenige, unterschiedliche Aktionen ausgelöst werden sollen, wie dies z. B. in dem Forth–Wort* `EXPECT` *notwendig ist — dort nämlich die Spezialbehandlung von* `<space>` *,* `<backspace>` *und* `<return>` *.*

Zunächst aber etwas Theorie, nämlich die drei verschiedenen Typen von `CASE`–Statements:

positional case Im Falle `0` soll die eine, im Falle `1` eine andere, im Falle `2` noch eine andere usw. Aktion ausgelöst werden. D. h., die einzelnen Fälle sind fortlaufend „durchnummeriert". Darauf will ich heute nicht besonders eingehen, obwohl dieser Fall in Forth einfach und effizient zu lösen ist. Aber merke: Der erste Fall ist Fall `0`, wie überhaupt in Forth grundsätzlich ab `null` gezählt wird — wer das nicht beherzigt, der zahlt dadurch, dass er diverse `1-` und `1+` in seinem Code versprenkeln muss.

keyed case Im Falle `12` soll eine, im Falle `534` eine andere usw. Aktion ausgelöst werden, und wenn keiner der aufgeführten Fälle zutrifft, dann irgendeine Standardaktion. Es werden unterschiedliche Code–Sequenzen ausgeführt in Abhängigkeit einer Zahl, die mit vorgegebenen, gestreuten Werten verglichen wird. Eine solche Struktur kann in einfacher Weise mit dem Wort `case?` und den üblichen Kontrollstrukturen implementiert werden. Davon später mehr.

range case Im Falle 13 . . . 20 dies, bei 21 . . . 39 das, etc. und ansonsten `default`. Selten gebraucht.

Als Motivation nun zunächst ein kleines Spiel, aus dem die Benutzung von `case?` deutlich werden sollte.

```
013 CONSTANT <CR>
: wastun  ( Zahl - Flag)
   ASCII j case? IF  ." ja"     False EXIT THEN
   ASCII n case? IF  ." nein"   False EXIT THEN
      <CR> case? IF  ." fertig" True  EXIT THEN
   emit ."  ungültig" False ;
: Spiel   BEGIN KEY wastun UNTIL ;
```

Ich hoffe, dass es ziemlich offensichtlich ist, was passiert; wers nicht durchschaut, muss halt ausprobieren, dass ihn nur ein `<cr>` aus dem Spiel befreit.

Ungewohnt mag die Benutzung von `EXIT` erscheinen (in FIG–Systemen `;S` genannt). Mit `EXIT` wird `wastun` verlassen und als nächstes wird dann wieder ein logischer Wert `UNTIL` zum Fraß vorgeworfen. Allgemein kann `EXIT` benutzt werden, um `ELSE`–Verzweigungen zu vermeiden.

Merke: Jeder `ELSE`–Zweig, der nicht vorkommt, braucht auch nicht durchdacht zu werden, wenn man sich später einmal das Programm anschaut, oder wenn etwa jemand anderes es verstehen soll. Wers nicht einsieht, der erinnere sich einmal, welche Editierbemühungen es kostet, eine verschachtelte Struktur deutlich zu machen — viel einfacher ist das Denken in geschichteten Strukturen. Psychologische Untersuchungen legen den Schluss nahe, dass der Mensch mit fünf bis sieben abstrakten Objekten „jonglieren" kann. Das bestätigt die gefühlsmäßige Erfahrung, dass sich gutes Forth durch viele kurze Definitionen auszeichnet, die aufeinander aufbauen — die Alternative ist „Spaghetticode", Nährboden zahlreicher *pretty printer* und anderer Analyseprogramme.

Was `False` und `True` im Programm zu suchen haben? — das sind simple Konstanten.

```
0 CONSTANT False   1 CONSTANT True
```

Ähem, ab dem 83er–Standard gilt allerdings:

```
-1 CONSTANT True
```

Bevor ich nun die Definition von `case?` hinschreibe, zunächst einmal einige inzwischen weitverbreitete Worte, um den Stack zu manipulieren; die werden so häufig gebraucht, dass es sich lohnt, sie mit in den Kern zu packen.

```
: nip   ( n0 n1 - n1)          swap drop ;
: under ( n0 n1 - n1 n0 n1)    swap over ;
: stash ( n0 n1 - n0 n0 n1)    over swap ;
```

Nach diesen ganzen Vorbereitungen denn nun:

```
: case? ( n0 n1 - tf / n0 ff)
  over = dup IF nip THEN ;
```

Zwei Zahlen liegen auf dem Stack, `n0` ist meine „Eingabe", die einen Fall auswählen soll und `n1` ist ein Vergleichswert. Stimmen `n0` und `n1` überein, dann habe ich meinen gesuchten Fall gefunden und `case?` kommt einfach mit der „True Flag" `tf` zurück, sodass das im Allgemeinen folgende `IF` passiert werden kann; stimmen `n0` und `n1` nicht überein, nun, dann wird `n0` für weitere Vergleiche aufbewahrt und eine „False Flag" `ff` ermöglicht das Überspringen des folgenden `IF`'s.

[1] (c) 1984, K. Schleisiek. Reprint in dieser Ausgabe wurde vom Autor gestattet.

Forth–Projekte auf Codeberg

Rezension

Codeberg ist eine gemeinnützige, communitygetriebene Plattform für die Zusammenarbeit und das Hosting von Git–Repositories, speziell für freie und quelloffene Softwareprojekte, basiert auf dem Open–Source–Tool *Forgejo*, und wird vom Codeberg e.V. betrieben. `https://de.wikipedia.org/wiki/Codeberg`

Carsten Strotmann war dort für euch unterwegs. Hier seine Funde im Dezember 2025 und meine Notizen dazu.

Akumuli Stack oriented programming language inspired by forth written in nim `https://codeberg.org/tttardigrado/Akumuli`

Kostprobe gefällig?

```
################################################
# Calculator
# - - - - - - - - - - - - - - - - - - - - - - -
# Input, Output, Stack, Alt Stack, Ints, Floats
################################################

# Takes a value as an argument
# Checks if it is equal to the top value
# of the alt stack
# (Without consuming that stack value)
# (x)+(y) -b-> (x.y)+(_) -d-> (x.y.y)+(_)
#   -c-> (x.y)+(y) -e-> (x==y)+(y)
@ is
  back dup cross eq
;

# Get the first value
'' ':' 'A' echo input toF

# Get the second value
'' ':' 'B' echo input toF

# Get operation to the alt stk
'' ':' 'p' 'O' echo input cross

'+' is if
       add print
else '-' is if
  swap sub print
else '*' is if
       mul print
else '/' is if
  swap div print
else
  '\n' 'r' 'o' 'r' 'r' 'E' echo
;;;;
```

Und da sage noch einer, Forth sei esoterisch …

forthstrap A simple Forth with the goal of being used in the Linux bootstrap process. `https://codeberg.org/notgull/forthstrap`

> "forthstrap starts from a `hex0` loader image. This image is highly platform dependent; however, it is only required to implement the Forth interpreter, compiler and a small number of built–in words. In addition, the resulting Forth interpreter is somewhat slow. From here, we eventually move to: Jumping to Protected Mode from Real Mode (where applicable). Moving to a more advanced interpreter with better performance. Implementing memory management, virtual memory, paging and file system access. Setting up a small operating system, with a user mode and basic drivers. Setting up a more advanced Forth compiler, used to compile userspace applications. A simple userspace with some Unix–like utilities, like an assembler and a linker. Setting up a C compiler … Inspiration is taken from sectorforth, jonesforth and eulex."

Ambitioniertes Vorhaben, oder? Mal sehen, was draus wird.

Flood–Game `https://codeberg.org/krisbug/forthgames/src/branch/main/flood.fth`

> "The goal of the game is to fill the most tiles with a single color."

```
\ 1-5: Fill with color
\ q: Quit
\ Type PLAY to play
...
```

55 Zeilen kompakter Forth–Code. Hübsch gemacht von Krisbug („I make stuff sometimes"). Um das in Gforth laufen zu lassen, braucht es etwas Anpassung. Sein `INCLUDE common.fth` gibts da so ja nicht und so fehlt sein `RAND` und schon geht das Grübeln los — ein schönes Puzzle. Krisbug hat da ja noch mehr eingestellt, mal sehen, ob jemand fündig wird und das `comon.fth` findet.

BugForth Experimental Forth system written in Forth. `https://codeberg.org/krisbug/bugforth`

Auch von Krisbug — ein fleißiger Tscheche, wie mir scheint. Er benutzt Gforth dabei und M4 dafür.

llforth Minimalist, highly extensible lua flavoured forth `https://codeberg.org/ufrag/llforth`

Sehr speziell für Lua–Freunde. Ein interessanter Weg, ein Forth hochzuziehen mittels Lua in C.

BLEcli Bluetooth LE command line interface. An iOS app that allows command line programming Bluetooth LE using a Forth engine. `https://codeberg.org/cayeric/BLEcli`

"The command line interface commands can be divided into three parts, each corresponding to a dedicated functionality:

system–specific words: `SYS!`, `SYS@`, `SYS?` — send out a message to the system interface, retrieve incoming system messages, get the number of still pending messages

bluetooth specific words: `scan-on`, `scan-off`, `scanner`: starts resp. ends listening to Bluetooth LE advertising packages, print out the name of advertising devices general

Forth words: the integrated Forth engine implements a 16–bit Forth comprising of control words, in–/output, arithmetic, stack handling.

See `Reference.md` for a list of available Forth words ..."

vulpforth Forth with some rather odd design decisions `https://codeberg.org/xfnw/vulpforth`

Assembliert. "takes inspiration from: miniforth (dtc using lods, dedicating a register to the value at the top of the working stack), duskos ..."

CHIP–8 Interpreter written in Forth. `https://codeberg.org/krisbug/chip4th`

"CHIP–8 is a virtual machine and interpreted programming language designed in 1977 by RCA engineer Joe Weisbecker for the COSMAC VIP microcomputer, intended to simplify video game development on 8–bit systems. It operates as a platform–independent environment, making it a precursor to modern virtual machines like Java VM or .NET CLR. The language features a 64×32 pixel monochrome display, a 16–key hexadecimal keypad, 16 general–purpose 8–bit registers, one 16–bit index register, and a program counter, with instructions stored as 16–bit opcodes. CHIP–8 programs are written in a low–level, assembler–like syntax and executed by an interpreter, which allows the same code to run across different hardware platforms as long as a CHIP–8 interpreter is available. Although originally used on kits like the COSMAC VIP and Telmac 1800, CHIP–8 has seen revivals on devices such as the HP–48 graphing calculator and remains a popular educational project for learning emulation and virtual machine implementation ... [Grok]

Mir scheint, der fleißige Herr Krisbug hat studiert, wie ein Chip–8 gemacht ist, und siehe da, geht auch in Forth.

forth–gen Small Forth–based OS (aarch64 und x86_64): `https://codeberg.org/rune-os/forth-gen`

"Forth OS is a portable, bare–metal operating system designed from the ground up around the principles of the Forth programming language. Its architecture is guided by three core tenets:

Portability: The kernel is designed to be largely architecture–agnostic, with a clean separation between generic code and hardware–specific implementations.

Security: The system employs a dual–stack model, a classic and robust design that provides strong protection against common memory corruption exploits.

Simplicity: The design favors clarity and minimalism, using a small set of powerful abstractions to build a complete system ..."

Danny Milosavljevic hat da viel Mühe in die Dokumentation gesteckt. Sehr beachtliches Werk. Er bezieht sich auf das Buch: *Intel® 64 and IA–32 Architectures Software Developer's Manual Volume 1: Basic Architecture.* Wer das also nachvollziehen möchte, findet hier viele Anregungen.

Nucleus Commandline–based interactive 16–bit Forth system for 8–bit AVR microcontroller `https://codeberg.org/cayeric/NUCLEUS`

Cay–Eric Schimanski versucht hier, auf dem MacOS einem ATMega Forth beizubringen — amForth, CamelForth und JonesForth stehen Pate.

mk

Von Fahrenheit, Celsius und der Bruchrechnung

Euer Kolumnist

Bei unseren Nachbarn im Westen war in den ForthWords_2025–10 eine hübsche Aufgabe gestellt worden. Der bekannte Code zur Umrechnung von Temperaturen in Celsius nach Fahrenheit und umgekehrt sollte für genauere Ergebnisse umgeschrieben werden, z. B. für Zehntelgrade. Die Berechnung der Ausgabe als Dezimalzahl mit Nachkommastellen sollte ohne Gleitkommazahlen mit Brüchen erfolgen und war die Zusatzaufgabe.

Abonniert die ForthWords, es kommen dort immer wieder schöne Aufgaben zum Mitmachen.

Es gab tatsächlich eine ganze Reihe von Einsendungen: Bryan Tevreden, Henny Luijkx, Jeroen Hoekstra, Willem Jager, Willem Ouwerkerk und auch Ulrich Hoffmann haben die Aufgabe eigentlich alle recht ähnlich gelöst.

C>F und zurück

Im Prinzip sahen die Lösungen so aus[1], einmal ganzzahlig und dann in Form von Brüchen:

```
decimal
: C>F ( C -- F ) 9 5 */ 32 + ;
: F>C ( F -- C ) 32 - 5 9 */ ;

\ t/n numerator/denominator
: C>>F ( ct cn -- ft fn )
  5 * swap 9 * over 32 * + swap ;

: F>>C ( ft fn -- ct cn )
  tuck 32 * - 5 * swap 9 * ;
```

Einige haben statt `swap` und `tuck` auf dem Stack mit `>r` und `r>` jongliert, um den einen in den anderen Bruch umzuwandeln.

.Q

Auch der zweite Teil wurde ähnlich gelöst. Man dividiert zunächst den Zähler (numerator) durch den Nenner (denominator), schreibt einen Punkt und verfährt mit dem Rest wie bei der manuellen Division so oft, wie Nachkommastellen gefordert sind.

```
\ Print t/n (the quotient of t and n)
 \ as a decimal fraction
: .Q ( t n #dec -- )
  \ #dec = number of decimal places
  >r
  tuck >r s>d r>
  sm/rem 0 .r   \ print signed prefix
  [char] . emit
  abs
  r> 0 ?DO \ print out places
  ( t n ) over >r 10 um*
  ( t d ) r> um/mod 0 .r
  ( t n ) LOOP
  2drop ;
```

Wem `tuck` fehlt, der macht sich eins. Sofern `locals` existieren in Form der geschweiften Klammern, ist das sehr einfach:

```
: tuck { a b -- b a b } b a b ;
```

Aber der Klassiker tut es natürlich auch:

```
: tuck ( a b -- b a b ) swap over ;
```

Oder man benutzt `need` in der Hoffnung, ein modernes Forth zu haben, das mit Bibliotheken daherkommt.

Und natürlich muss man sich in die Divisionsroutinen seines Forthsystems vertiefen, um die richtigen herauszufinden. Aber darum ging es ja in der Aufgabe. :)

Beifang <rtn>

Mit ins Netz gegangen ist mir dabei noch Ulrichs Art der Sichtprüfung einer Zahlenausgabe (s. Listing).

```
: <rtn> ( <num> -- )
  bl word count ." =?= " type cr ;

21 c>f . <rtn> 69
22 c>f . <rtn> 71
...
```

Das ist ja interessant. Wir probieren:

```
2 10 * cr . <rtn> 20
20 =?= 20
ok

3 10 * cr . <rtn> 40
30 =?= 40
ok
```

`2*10` soll `20` ergeben. Tut es offensichtlich! Und beim zweiten Beispiel soll `40` herauskommen, was offensichtlich nicht der Fall ist — da stimmt wohl was nicht in der Multiplikation. Und man kann in die Fehlersuche gehen. Einfach und verblüffend.

Die Phrase `bl word` nimmt die nächst folgende Zeichenkette aus dem *input stream* und liefert dessen Adresse und Länge an `count` ab. Die dann von `type` gedruckt wird. Man hat praktisch eine String–Konstante eingebaut, die mit der gewünschten Ausgabe vergleichen werden kann. Herrlich, oder?

[1] Habe mal Ulrichs Beispiel genommen. Er benutzte Gforth, das habe ich auch hier.

Listing

```
\ C>>F F>>C .Q Albert's challenge 8/2025      uho 2025-10-11

\ see breuken.pdf

\ F = 9/5 * C + 32
\ C = (F-32) * 5/9

need .r  \ 0 .r allows to print numbers without trailing space
need tuck

decimal
: C>F ( C -- F ) 9 5 */ 32 + ;
: F>C ( F -- C ) 32 - 5 9 */ ;

\ t/n numerator/denominator
: C>>F ( ct cn -- ft fn )  5 *  swap 9 *  over 32 * +  swap ;

{ 0 1 c>>f  / -> 32 }
{ 30 1 c>>f / -> 86 }
{ 1000 10 c>>f / -> 212 }


: F>>C ( ft fn -- ct cn )  tuck 32 * -  5 *  swap 9 * ;

{ 86 1 f>>c / -> 30 }
{ 32 1 f>>c / -> 0 }
{ 388 2 f>>c / -> 90 }


\ Print t/n (the quotient of t and n) as a decimal fraction
: .Q ( t n #dec -- ) \ #dec = number of decimal places
  >r
  tuck >r s>d r> sm/rem 0 .r [char] . emit  \ print signed prefix
  abs
  r> 0 ?DO ( t n ) over >r 10 um*  ( t d ) r> um/mod 0 .r ( t n ) LOOP \ print out places
  2drop ;


\ Examples

: <rtn> ( <num> -- ) bl word count ."  =?= " type cr ;

21 c>f . <rtn> 69
22 c>f . <rtn> 71

21  1  c>>f 3 .q  <rtn> 69,800
22  1  c>>f 4 .q  <rtn> 71.6000
215 10 c>>f 5 .q  <rtn> 70.70000
707 10 f>>c 6 .q  <rtn> 21.500000
355 113    32 .q  <rtn> 3.14159292035398230088495575221238
1000 998   32 .q  <rtn> 1.00200400801603206412825651302605

22  7       4 .q  <rtn> 3.1428
-5  2       4 .q  <rtn> -2.5000
23  1  f>>c 4 .q  <rtn> -5.0000
194 10 f>>c 4 .q  <rtn> -7.0000
```

Historisches am Rande

Wenn ich mich in sowas vertiefe, interessieren mich auch immer die beteiligten Leute hinter den altehrwürdigen Namen und die verschlungenen Pfade ihrer Wissenschaft und die Vokabeln, die sie wählten.

Temperatur

Ein Maß dafür, wie „heiß“ oder „kalt“ etwas ist. Genauer gesagt beschreibt sie die durchschnittliche kinetische Energie der Teilchen (Atome/Moleküle) in einem Stoff oder System. Und jeder Körper leuchtet Infrarot. Je wärmer ein Objekt (z. B. Stirn, Trommelfell, heiße Pfanne), desto intensiver seine Infrarotstrahlung.[2]

Doch bis zu dieser Betrachtungsweise war es ein langer Weg. Im Alltag benutzen wir heute beides selbstverständlich nebeneinander, physikalische Thermometer wie die von Fahrenheit oder Celsius und elektronische digitale Thermometer. Doch das war nicht immer so.

Unser Wort *Temperatur* stammt — wie in fast allen europäischen Sprachen — aus dem Lateinischen. „Temperatura“ bedeutet wörtlich „die richtige Mischung“ oder „das Gemischte“ — wie noch bei den Tempera–Farben in Gebrauch.[3]

„Temperare“ selbst kommt wahrscheinlich von einer indogermanischen Wurzel *temp–/tem–* mit der Grundbedeutung „(ab)schneiden, abmessen, in richtiger Weise teilen“, verwandt mit lateinisch *tempus* „Zeit, Zeitabschnitt“ — weil Zeit „abgemessen“ wird.

Von der Antike bis zum Mittelalter wurde unter „Temperatur“ das richtige Mischungsverhältnis der vier Säfte im Körper[4] nach der Humoralpathologie verstanden. Erst im 19. Jahrhundert wurde sie durch die Zellpathologie[5] und die Entdeckung von Bakterien, Viren usw. endgültig abgelöst. Trotzdem leben viele Begriffe weiter — „cholerisch“, „melancholisch“, „blutarm“, „gute/schlechte Laune“ = „guter/schlechter Saft“.

Eine „gute Temperatur“ bedeutete Gesundheit, Ausgeglichenheit (daher auch „temperamentvoll“ = gut gemischt).

Erst mit der neuzeitlichen Physik brauchte man ein Wort für den „gemessenen Grad der Wärme oder Kälte“ und hat eine „Mischung“ in der Materie vermutet, nannte Geräte zur Objektivierung der sonst nur *erfühlten* Materiezustände „Mischungsanzeiger“. Der Begriff „Thermoskop“ fiel. Und propagierte von Galilei und seinen Schülern bis zu Fahrenheit, Celsius und Kelvin.[6]

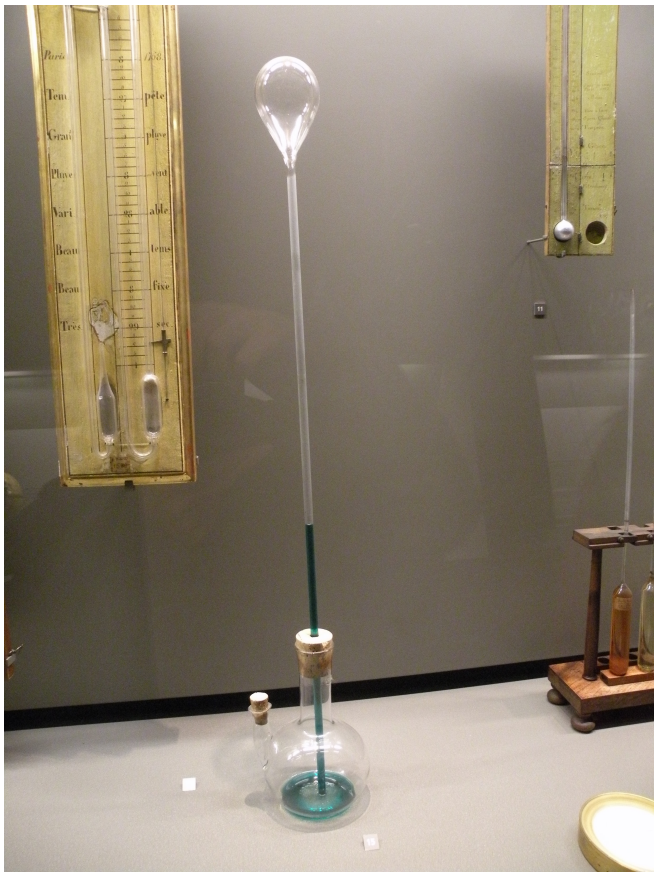


Foto: Chatsam

Abbildung 1: Musée des Arts et Métiers: Thermoscope de galilée 1592

Vom Thermoskop zum Thermometer

Galilei hat kein Thermometer im heutigen Sinne erfunden, mit Skala und genauen Gradzahlen, sondern das erste bekannte Thermoskop (ca. 1593–1600) — ein Gerät, das Temperaturänderungen sichtbar, aber noch nicht genau messbar machte. Es war ein langes, dünnes Glasrohr, mit einer Luft gefüllten faustgroßen Glaskugel oben und unten offen. Das offene Ende wurde in ein Gefäß mit farbigem Wasser getaucht. Durch leichtes Erwärmen der Glaskugel mit der Hand wurde ein Teil der Luft ausgetrieben. Lies man los stieg das Wasser ein Stück ins Glasrohr hoch und blieb dort stehen. Je kälter es wurde, desto höher stieg die Wassersäule und umgekehrt. Galileis Schüler und die *Accademia del Cimento*[7] in Florenz bauten das Gerät weiter. Sie drehten es um. Die Glaskugel kam nach unten, das Rohr nach oben, aber verschlossen. Sie füllten es vollständig mit Alkohol und brachten erste Skalen an.

[2] Wellenlänge ca. 5–14 µm; Planck'sches Strahlungsgesetz.

[3] Tempera–Farben = wasserlösliche, schnell trocknende, matte Farben mit einem emulsifizierenden Bindemittel (meist Eigelb) + Farbpigment. Sie ergeben eine sehr präzise, detailreiche Malweise und eine unverwechselbare, samtig–matte Oberfläche.

[4] Blut, Schleim, gelbe Galle, schwarze Galle.

[5] Rudolf Virchow, 1858

[6] Galileo di Vincenzo Bonaiuti de' Galilei, *15. Februar 1564 in Pisa, Italien. Daniel Gabriel Fahrenheit, *24. Mai 1686 in Danzig (heute Gdańsk, Polen). Anders Celsius, *27. November 1701 in Uppsala, Schweden. William Thomson, 1. Baron Kelvin, *26. Juni 1824 in Belfast (heute Nordirland).

[7] Die Accademia del Cimento war 1657–1667 in Florenz die weltweit erste „reine Experimentier–Akademie“ unter Medici–Schirmherrschaft und ein wichtiger Wegbereiter der modernen Wissenschaft. Hat leider nur 10 Jahre Bestand gehabt, die Kirche war dagegen!

Das war dann schon der direkte Vorläufer der modernen Flüssigkeitsthermometer von Fahrenheit und Celsius.

Fahrenheit

Über ihn gibs viel zu lesen im Internet. Danach wurde er Glasbläser aus einer Mischung aus Neugier, Notwendigkeit und den Anforderungen seiner wissenschaftlichen Arbeit — ein Autodidakt. Doch welche Begegnungen dazu führten, dass er von der kaufmännischen Ausbildung abließ, um sich mit gläsernen Apparaturen zu befassen, und solche selbst aus Glas zu formen, dazu schweigt seine Biografie.

Um präzise Thermometer und Barometer herstellen zu können, musste er extrem genaue und gleichmäßig dünne Glasröhren ziehen — etwas, das ihm die Glasbläser seiner Zeit meist nicht in der nötigen Qualität lieferten. Deshalb lernte er selbst das Glasblasen in Amsterdam und auch auf Reisen in England und in Dänemark bei Ole Rømer. Er perfektionierte die Technik so weit, dass er sehr dünne und gleichmäßige Glasröhren herstellen konnte, ein hochpräziser wissenschaftlicher Instrumenten–Glasbläser.

Fahrenheit–Skala

Die Röhrchen machen, in die sich eine Flüssigkeit ausdehnen und damit Temperarturen anzeigen kann, ist das eine. Das Prinzip ist einfach: Die Flüssigkeit in der kleinen Glasblase am Ende des Röhrchens kann sich nur in den freien Raum, in die Kapilare hinein ausdehnen. Je feiner das Röhrchen, je mehr Weg macht selbst die kleinste Ausdehnung.

Er nahm Quecksilber als Ausdehnungsflüssigkeit, wohl, weil dessen Ausdehnungskoeffizient kaum temperaturabhängig ist, im Gegensatz zu Alkohol, und es größere Messbereiche ermöglichte. Quecksilber gefriert bei –38,8 °C und siedet erst bei 357 °C.

Eine Skala an diese Röhrchen anzubringen, ist die andere Sache. Ich schätze mal, jedes einzelne Thermometer musste geeicht werden, selbst wenn die Röhrchen noch so fein gemacht wurden, genau gleich werden sie nicht gewesen sein. Und wie macht man mit geometrischen Mitteln, also Zirkel und Lineal, fein unterteilte gleichlange Einteilungen?

Interessant fand ich, dass Fahrenheit Vielfache von 6 genommen hat: 96=6x16. Die Sechstel ergeben sich im Kreis durch den fortgesetzten Zirkelschlag mit dem Radius, die bekannte Rosette. Gedreht um einen rechten Winkel, ergibt es die bekannten 12 Teile der Uhr usw. Und mit der 96–Teilung kann man schon feinteilige Skalen drucken. Mit einem kleineren und einem größeren solchen Maß und einer Drahtharfe dazwischen aufgespannt kann man Röhrchen variierender Längen passabel skalieren, wenn man zwei praktische Fixpunkte an Temperaturen hat.

Bleibt noch das Problem der praktischen Eichung. Einfach zu handhaben und billig herzustellen. Die eigene Körpertemperatur als den einen Endpunkt der Skala zu benutzen, ist genial, die hat ja jeder Handwerker immer dabei. Der andere Endpunkt, ein möglichst tiefer Wert, war schon schwieriger. Es musste auf jeden Fall etwas sein, dass kälter als Wasser am Gefrierpunkt[8] ist. Denn es war schon klar, dass Eis auch kälter als Wasser an seinem Gefrierpunkt werden kann, und somit Messungen in den Bereich hinein erforderlich waren.

Mit der Kältemischung aus Eis + Salz + Salmiak (NH_4Cl) erreichte er stabil etwa –17,8 °C (0 °F). Er schrieb explizit, dass man diese Mischung „jederzeit“ herstellen könne, auch im Sommer.

Man nahm dazu zerstoßenes Eis oder Schnee, mischte es in einem Holzgefäß mit dem Salz, rührte kräftig um und tauchte das Thermometerrohr direkt in die breiige Mischung, bis die Flüssigkeitssäule stehen blieb. Das dauerte nur wenige Minuten und war das ganze Jahr über möglich — solange man Eis im Keller lagerte oder im Winter erntete. Eiskeller sollen damals in Amsterdam, London und Paris normal gewesen sein, sagt man. Also erst rein in den Bottich, danach am Körper warm gemacht, schon hat man den unteren und oberen Punkt am Röhrchen markiert. An die Drahtharfe gehalten, lässt sich sodann die 96er–Skala auswählen in Form des passenden Elfenbeinplättchens.[9]

Feinmechanik mit einfachen Mitteln und dennoch genauer als alles, was es damals gemeinhin gab.

Celsius

Er wurde nicht aus handwerklicher oder instrumentenbaulicher Leidenschaft Thermometerbauer. Celsius kam zum Thermometerbau aus einem anderen Grund.

Als er Professor für Astronomie in Uppsala wurde, übernahm er auch die Leitung des neuen Observatoriums (fertig 1741). Er wollte dort tägliche, langjährige und international vergleichbare Wetteraufzeichnungen einführen — das war damals ein großes europäisches Projekt (z. B. auch in Paris, London, St. Petersburg); Klimaforschung sozusagen. Dafür brauchte er viele identische, stabile und einfach zu vergleichende Thermometer.

Auf seiner großen Europareise — Berlin, Paris, London, Italien — traf er die führenden Physiker und Instrumentenmacher. Dort sah er die neuesten Quecksilberthermometer von Réaumur (mit 80°–Skala) und die Probleme der alten Alkoholthermometer (unterschiedliche Eichungen, starke Temperaturabhängigkeit des Alkohols).

Die in Schweden verfügbaren Thermometer, meist mit Alkohol, teils Réaumur–Skala, teils Fahrenheit–Kopien, wichen um 5–10° voneinander ab. So ließ er in Stockholm und Uppsala selbst Quecksilberthermometer bauen, hauptsächlich durch den Instrumentenmacher Daniel

[8] Schnee–Wasser–Gemisch

[9] Aus Elfenbein geschnitzte Skalen waren damals bei Hofe beliebt. Einfache Leute konnten sich Fahrenheits Thermometer eh nicht leisten.

Ekström. und er wollte eine Skala, die „einfach, dezimal und an die Naturphänomene gebunden“ war, also „100° zwischen den beiden wichtigsten Fixpunkten des Wassers“.

100° = Schmelzpunkt von Eis — 0° = Siedepunkt von Wasser

Ja, so herum dachte er sich das, also genau umgedreht, wie wir es heute kennen!

Das war damals logisch für ihn. Der tiefere Punkt sollte die höhere Zahl haben. Außerdem war der Siedepunkt wetterabhängig (Luftdruck!), der Eispunkt aber „jederzeit reproduzierbar“ und daher sollte er die „sichere“ 100 haben.

Celsius selbst hat nie die umgedrehte Skala benutzt, sondern seine „verkehrte“.

Umgedreht haben das wohl erst nach seinem Tod zwei Personen fast gleichzeitig. 1743/44 Christin in Lyon (ein Schüler von Réaumur) und 1745–1750 Carl von Linné und Pehr Elvius in Uppsala, weil es ihnen praktischer erschien (0° unten, 100° oben). Das setzte sich weltweit durch und wurde dann „Celsius–Skala“ genannt.

Foto: Dr. Manuel

Abbildung 2: Ole Rømers

Ole Rømers[10]

Und wenn wir schon dabei sind, sollte auch dieser Astronom erwähnt werden. Er entwickelte eines der ersten wirklich brauchbaren und reproduzierbaren Thermometer mit fester Skala mehrere Jahrzehnte vor Fahrenheit. Rømer stellte allerdings Thermometer her mit rot gefärbtem Weinspiritus, bereits mit zwei fixen Punkten und präziser Glasbläserei. Er verwendete auch schon die Kältemischung aus Eis + Salz, dem tiefsten Punkt, den man im Winter erreichen konnte — 0 °Rø (ca. –17,8 °C) und legte den Siedepunkt von Wasser bei Normaldruck auf 60 °Rø (100 °C).

Der junge Fahrenheit hat Ole Rømer in Kopenhagen besucht und sah dort die Idee der zwei reproduzierbaren Fixpunkte und die extrem präzise Glasbläserei. Fahrenheit übernahm und verfeinerte das, hat also die Rømer–Skala im Prinzip nur „gestreckt“, aber eben auch die Eichung der Röhrchen vereinfacht und sozusagen die Serienproduktion erst ermöglicht.

[Anmerkung: 1676 wurde Rømer zum königlich–dänischen Astronomen ernannt. Als solcher wurde man wohl gemalt. Von Fahrenheit und Celsius gibt es keine solchen schönen Bilder für die Nachwelt — und die Fotografie war noch nicht erfunden. J. N. Niépces „Blick aus dem Arbeitszimmer“ war 1826. Es ist die älteste erhaltene Fotografie, auf einer asphaltbeschichteten Zinnplatte gemacht, aber ich schweife ab . . .]

Und heute . . .

. . . wissen wir vom Zusammenhang von Ausdehnung und Strahlung. So ein elektronisches Thermometer, das die Wärmestrahlung misst, also *ohne Kontakt* zur Oberfläche, heißt Infrarot–Thermometer, Pyrometer oder umgangssprachlich „Fieberthermometer mit Stirnscan“ und „Ohrthermometer“ oder „Infrarot–Temperatursensor“.

Über eine kleine Linse wird die Infrarotstrahlung von einem definierten Messfleck (z. B. auf der Stirn) auf den Temperatur–Sensor fokussiert. Die Differenz zwischen der Strahlung des Objekts und der eigenen Gehäusetemperatur ergibt die Oberflächentemperatur. Darum ist ein zweiter Temperatursensor im Gerät selbst und ein Mikrocontroller berechnet daraus die tatsächliche Oberflächentemperatur am zu messenenden Objekt. Und macht natürlich die Anzeige der Temperatur auf einem kleinen Display.

Womit wir wieder bei Forth und `EMIT` angekommen sind. Werft doch mal einen Blick in unser *Forth–Wiki*. Im *Projekt Forth Works* gibts auch allerlei zu Sensoren. Und ein Blick auf eure Lieblings–MCU zeigt vielleicht sogar den darin verbauten Temperatursensor an einem internen ADC–Portpin.

mk

[10] Ole Rømer (1644–1710), Däne; bekannt durch die erste Messung der Lichtgeschwindigkeit.

25 Jahre AATiS e.V.

Euer Kolumnist

Ihr wisst noch, was uns, die Forth–Gesellschaft, mit denen, dem Arbeitskreis Amateurfunk und Telekommunikation in der Schule e.V., verbindet? Genau, steht in unserer Satzung! Jener Verein wird uns dereinst beerben. Und ich vermute mal, den AATiS e.V. wird es dann auch tatsächlich noch geben ...

Und zwar, weil der AATiS e.V. jünger ist als wir es sind, denn sein Nachwuchs kommt direkt aus den Schulen. Die Gründungsmitglieder waren weitsichtig.

> „Vor 25 Jahren, am 24. September 1994 wurde in Harsum bei Hildesheim der Arbeitskreis Amateurfunk und Telekommunikation in der Schule als eingetragener Verein konstituiert. Ein Teil der zwölf Gründungsmitglieder, darunter Wolfgang Lipps DL4OAD, Harald Görlich DK9AC, Oliver Amend DG6BCE, Jörg Stotz DL6OAA und Carsten Böker DG6OU, hatten schon seit mehr als 15 Jahren intensiv und engagiertest zusammengearbeitet. Im Laufe der Jahre ist der Arbeitskreis auf über 600 Mitglieder im gesamten Bundesgebiet und mehreren angrenzenden Ländern angewachsen.
>
> Eine jährlich wiederkehrende Veranstaltung vom AATiS e.V. ist der „Bundeskongress für Amateurfunk und Telekommunikation an Schulen“, der im Jahr 2020 zum 35. Mal stattfindet und die bereits lange vor der Vereinsgründung zum jährlichen Angebot gehörte.
>
> Aus den Bundeskongressen ging auch der „blaue Ordner“ Amateurfunk in Schule und Weiterbildung mit dem DARC hervor. Aus ihm entstanden dann die Praxishefte, von denen inzwischen 29 Ausgaben erschienen sind. Die Jubiläumsausgabe mit der Nummer 30 wird dann zum 35. Bundeskongress erscheinen.
>
> Seit fünf Jahren wird der Verein durch Harald Schönwitz, DL2HSC, geführt und hat deshalb seinen Sitz in Börnichen / Erzgebirge.“ [Zitat: `https://www.aatis.de/content/25-jahre-aatis-ev`]

Na gut, in unserer Satzung heißt es noch:

> „16. Auflösung
>
> ... fällt das Vermögen des Vereins an den Arbeitskreis Amateurfunk in der Schule AATis eV, *Sedanstr. 24, 31177 Harsum*, der es ausschließlich und unmittelbar für gemeinnützige Zwecke zur Förderung von Wissenschaft und Forschung oder Bildung zu verwenden hat.“

Ich denke, die genaue Adresse kann man ersetzen durch so etwas wie „Sitz wie im Vereinsregister angegeben“, die aktuelle Anschrift wird dort verzeichnet sein. Auch wir wandern ja durch die Zeit und verlegen aus ebensolchen praktischen Gründen den Vereinssitz besser dorthin, wo die Vereinsführung inzwischen ist.

Das war nun für lange Zeit Hamburg, weil die Gründungsmitglieder Ulrich Hoffmann, Klaus Schleisiek, Andreas Goppold und andere aus Hamburg waren. Ulrich ist dem Verein treu geblieben, lange auch als Vorstand. Doch zurück zum AATiS.

Der AATiS stellt sich vor ...

> „Der Arbeitskreis Amateurfunk und Telekommunikation in der Schule (AATiS) e.V. als gemeinnütziger Verein ist ein kompetenter Partner für Lehrer, Jugendleiter, Ausbilder in der Industrie und weitere Interessenten sowie Schüler und Jugendliche. Zur Nachwuchsarbeit schult und bedient er sich Multiplikatoren, weil dadurch effektives Arbeiten gewährleistet ist. Die von ihm ausgearbeitete und erprobte Seminardidaktik wird geschätzt ...
>
> Der AATiS e.V. beschäftigt sich intensiv mit den folgenden Bereichen:
>
> - Amateurfunkanwendungen
> - Telekommunikation und Netze
> - Meteorologie, Aerologie, Klimatologie
> - Geo–/ Raumwissenschaften, Radioastronomie
> - Elektronik, Sensorik, Mikrocontroller, Robotik
>
> u. a. m.“ [Zitat: `https://www.aatis.de/content/ueber_den_aatis_ev`]

Ihr seht die Schnittmenge mit Forth: Radioastronomie, Elektronik, Sensorik, Mikrocontroller, Robotik. Und mit IoT funkt Forth — notgedrungen oder vergnüglich — immer mehr.

Fledermäuse im Regen

Ääh? — doch. Auch sowas wird da abgehandelt, Funkverbindungen und Sensorik eben.

> „Zu diesem Zweck sollen Akustiklogger für Ultraschallrufe in Kombination mit Regensensoren und weiteren Parametern wie Temperatur und Luftfeuchte auf Friedhöfen installiert werden. Friedhöfe wurden deshalb ausgewählt, weil sich diese Habitate für die Fledermäuse über Jahrzehnte kaum verändert haben und nachts relativ ungestört sind ...“ [ebenda]

Also klickt euch mal da durch bei unseren Erben, es ist viel los auf deren Webseite. mk

Mini–Terminal

Rafael Deliano

Manchmal ist ein Laptop zu unförmig. Nicht für die Entwicklung, wohl aber für den Betrieb, reichen einige farbige Funktionstasten und ein Display für die Steuerung von Geräten (Abb. 1).

Große, helle alphanumerische LED–Anzeigen sind oft besser ablesbar als LCDs, werden aber nicht mehr hergestellt. Obschon obsolet, dennoch erhältlich. Für Geräte, die nicht in Serie produziert werden, ist das kein Nachteil.

Die ältere Variante DL1414 von Litronix wurde 1978 im AIM65 Kleincomputer verwendet. Ihre LEDs in der „british flag“ Anordnung (Abb. 2, links) stellen Buchstaben nur leidlich gut dar. Hauptproblem ist die Linsenfunktion, die eine etwa 1mm starke rote Frontplatte in passendem Abstand benötigt (Abb. 2, rechts). Zudem fielen nach langer Betriebszeit oft einzelne Segmente aus.

Alternative und Nachfolger ist die 5x7–Matrix. Wurden von HP und Siemens (die Litronix gekauft hatten) hergestellt. Es gab verschiedene Serien, die letzte Variante hat die größte Bauform und hellsten LEDs (Abb. 6). Die Pinbelegung und Ansteuerung sind identisch (Tabelle 1). Das Teil ist in Europa noch häufig für etwa 25 EUR erhältlich. Die Stromaufnahme ist abhängig von der Zahl der aktiven LEDs, Nennwert etwa 150 mA. Bei langen Zeilen können sie also weder vom Preis noch Stromverbrauch mit den hellen VFL–Displays konkurrieren. Die ICs für LCD/VFL sind CMOS, die für LEDs aber bipolar. Das begrenzt die Komplexität. Sie stellen ASCII, deutsche Umlaute und einige Symbole dar, haben aber keine per RAM definierbare Zeichen. Die wären aber oft nützlich, wenn man Messwert, alternativ zu numerisch, umschaltbar auch als Bargraph darstellen will.

Die Ansteuerung erfolgt über einen Bus, der aber am Controller gleich zwei Ports belegt (Abb. 3). Für Blinkfunktionen haben die ICs eine gemeinsame Taktleitung (Abb. 4).

Tastatur

Auch der National MM74C922 ist inzwischen obsolet, aber in DIP und SO noch gut erhältlich. Die maximal 16 Tasten sind als Matrix verschaltet (Abb. 5). Tastendruck wird am IRQ–Pin gemeldet, ihr Code kann dann über den Datenbus gelesen werden.

Das Wichtigste dabei ist eine ordentliche Taste, hier wird der Klassiker *Digitast* verwendet.

Für Funktionstasten sind unterschiedliche Farben wünschenswert. Mit Schneidplotter kann man zusätzlich Symbole aus schwarzer Folie[1] fertigen. Da der obere Bereich der Hebeltaste nicht berührt wird, sind sie dort recht dauerhaft angebracht (Abb. 7).

klein	mittel	groß	
PD2435	PD3535	PD4435	H
PD2436	PD3536	PD4436	R
PD2437	PD3537	PD4437	G

Tabelle 1: Siemens Displays. H=high efficiency red, R=red, G=green

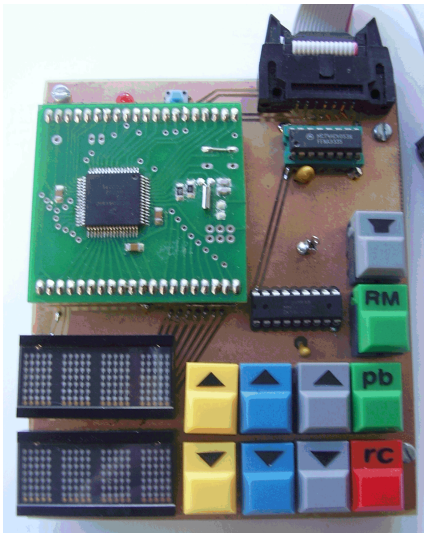


Abbildung 1: Breadboard–Terminal

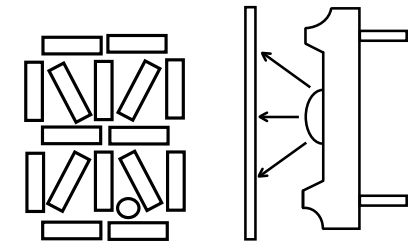


Abbildung 2: Links: British–Flag–Buchstabe. Rechts: DL1414 mit Plexiglas

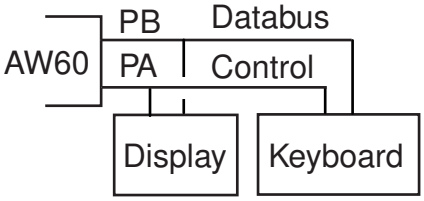


Abbildung 3: Blockschaltbild

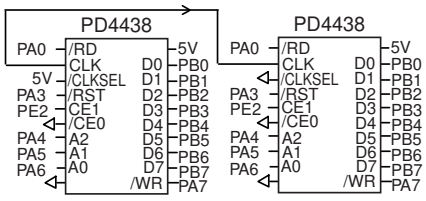


Abbildung 4: Schaltung Displays

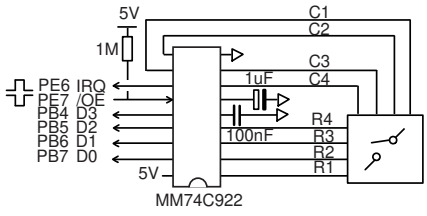


Abbildung 5: Schaltung Tastatur

[1] Oracal 631.

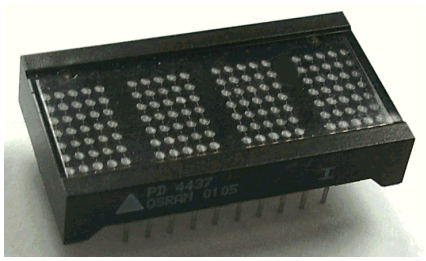


Abbildung 6: PD4435

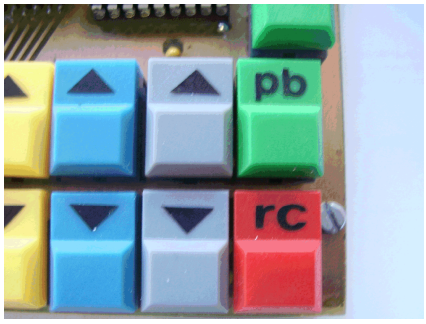


Abbildung 7: Tasten mit Aufkleber

Abbildung 8: SP86 12 mm

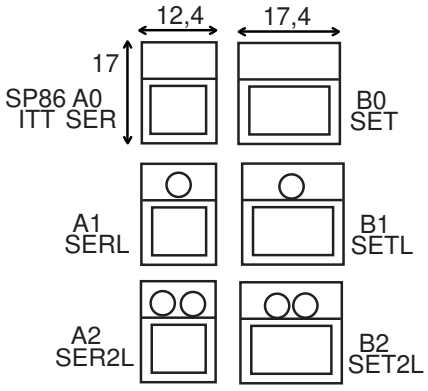


Abbildung 9: Grundtypen

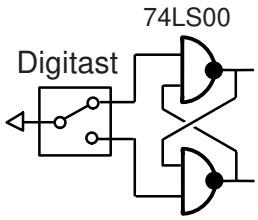


Abbildung 10: Entprellte Taste

Digitast

Diese deutschen Taster der 70er Jahre schienen auszusterben. Die Retro–Welle lebt vom Image (Abb. 11). Das verspricht hier klares Design und Qualität. Scheint kostendeckend zu sein, denn die Chinesen haben sie geclont. Vor allem für Kleinserie sind sie damit wieder interessant geworden.

Der Grundtyp wurde 1975 von ITT–Schadow Berlin eingeführt und speziell mit Anwendung für digitale Logik–ICs beworben. Prellfreie mechanische Tasten gibt es nicht. Aber ein Umschalter kann ein RS–FlipFlop steuern (Abb. 10), ein 74LSxx benötigt nicht mal Pullups. Die 17–mm–Version entspricht PC–Tasten, während die kleine 12–mm–Variante die Abmessungen eines DIL14–Sockels hat (Abb. 9), aber nicht in diesen gesteckt werden kann. Rote 5–mm–LEDs waren anfangs Hightech und wurden direkt eingebaut (Abb. 8). Während man bei deren Farben auf rot und grün beschränkt war, wurde die Kappe bald in allen poppigen Farben der 70er Jahre angeboten. Inklusive orange, das fertigen die Chinesen heute aber nicht mehr.

Mechanik und Kappen waren vom Anwender montierbar und wurden auch getrennt angeboten. Mit dem Erfolg am Markt entstand für die Kappen bald eine Unzahl mechanischer Varianten, um jede Anwendung abdecken zu können. Heute ein Albtraum für jeden, der Ersatzteile beschaffen will. ITT–Schadow wurde 2007 an C&K verkauft. Die Tasten sind weiterhin hierzulande erhältlich, aber für 5. . . 7 EUR unattraktiv.

Die SP86–Serie aus China für ca. 1 EUR beschränkt sich auf die Grundtypen in 7 gängigen Farben. Die LEDs sind rot oder rot/grün. Neben dem spürbaren Schaltpunkt ist auch der dezente akustische „Click“ erhalten geblieben.

Abbildung 11: Messgerät 80er Jahre

Listing 1

```
<| \ Display.txt

\ PB 0 ... PB 7 databus input

\ PA 4 DCONSTANT A2-DSP    \ o 0
\ PA 5 DCONSTANT A1-DSP    \ o 0
\ PA 6 DCONSTANT A0-DSP    \ o 0
\ PA 7 DCONSTANT /WR-DSP   \ o 1
\ PA 0 DCONSTANT /RD-DSP   \ o 1
\ PA 3 DCONSTANT /RST-DSP  \ o 0
\ PE 2 DCONSTANT CE1-DSP1  \ o 0
\ PE 3 DCONSTANT CE1-DSP2  \ o 0


: +DISP1  CE1-DSP1 B1! ; \ (  ---  )
: +DISP2  CE1-DSP2 B1! ;
: -DISP1  CE1-DSP1 B0! ;
: -DISP2  CE1-DSP2 B0! ;

\ addr
\ 000  control
\ 100  digit 0
\ 101  digit 1
\ 110  digit 2
\ 111  digit 3

\ control word
\ 7 6 5 4 3 2 10
\              01  0= off ; 1 = dim ;
\                  2 = medium ; 3 = bright
\        000       attributes disabled
\     0            blink disabled
\   0              lamp test disabled
\ 0                clear disabled

: (ADR!) \ ( adr bit mask --- )
AND IF B1! ELSE B0! THEN ;

: ADR!         \ ( adr --- )
A0-DSP HOPP B% 001 (ADR!)
A1-DSP HOPP B% 010 (ADR!)
A2-DSP ROT  B% 100 (ADR!) ;

: DISP-C!      \ ( UC1 adr --- )
FF DPB C! ADR! PB C!
/WR-DSP B0! /WR-DSP B1! 00 DPB C! ;

: DISP-C@      \ (     adr --- UC1 )
ADR! /RD-DSP B0! PB C@ /RD-DSP B1! ;

: INIT-DISP    \ (  ---  )
/RST-DSP B0! 1 MSEC /RST-DSP B1!
+DISP1
01 0 DISP-C!
41 B% 111 DISP-C!
42 B% 110 DISP-C!
43 B% 101 DISP-C!
44 B% 100 DISP-C!
-DISP1
+DISP2
01 0 DISP-C!
30 B% 111 DISP-C!
31 B% 110 DISP-C!
32 B% 101 DISP-C!
33 B% 100 DISP-C!
-DISP2 ;

\ (  ---  )
: OFF
  +DISP2 0 0 DISP-C! -DISP2 +DISP1 0 0 DISP-C! -DISP1 ;
: DIM
  +DISP2 1 0 DISP-C! -DISP2 +DISP1 1 0 DISP-C! -DISP1 ;
: MEDIUM
  +DISP2 2 0 DISP-C! -DISP2 +DISP1 2 0 DISP-C! -DISP1 ;
: BRIGHT
  +DISP2 3 0 DISP-C! -DISP2 +DISP1 3 0 DISP-C! -DISP1 ;

\ INIT INIT-DISP
|>

```

Listing 2

```
<|  \ Keyboard.txt

HEX

\ PB4 .. PB7 databus input
\ PE 6 DCONSTANT IRQ-KB  \ i
\ PE 7 DCONSTANT /OE-KB  \ o 1

: KB?   \ ( --- UC1 1 ) UC1 = 0...F
        \   ---     0 )
IRQ-KB B@ IF
/OE-KB B0! PB C@ /OE-KB B1!
4SHIFT> 1 ELSE 0 THEN ;

: TEST-KB  \ (  --- )
INIT
BEGIN
KB? IF CR CH. D% 1000 MSEC THEN
AGAIN ;
|>
```

Anmerkung zur Historie

Die Rudolf Schadow GmbH, ursprünglich aus Berlin, 1960er, wurde nach 2000 von ITT übernommen, daraus entstand ITT–Schadow, die 2007 an C&K Components ging, zusammen mit Jeanrenaud aus Frankreich.

International Telephone and Telegraph (ITT) war ein großer, historischer US–amerikanischer Mischkonzern (gegründet 1920), der in vielen Bereichen, Telekommunikation, Elektronik, Automotive, aktiv war.

C&K steht für die beiden Gründer Charles A. Coolidge Jr. und Marshall Kincaid, 1957, USA. Der Name wurde auch nach allen Eigentümerwechseln beibehalten.

Heute gehört C&K zu *Littelfuse*, ein amerikanischer Industrie–Konzern, der auf elektronische Schutz– und Steuerungskomponenten spezialisiert ist. Gegründet 1927 in Chicago als „Littelfuse Laboratories“ für kleine Messgeräte–Sicherungen, hat er heute ca. 19K Mitarbeiter in >50 Produktions– und Vertriebsstandorten in Amerika, Europa, Asien mit einem Umsatz von mehreren Milliarden USD jährlich. Starkes Wachstum durch Zukäufe, kürzlich Basler Electric (2025), früher Zilog, C&K Switches und diverse andere Schalter– und Sensorfirmen. mk

Forth–Gruppen regional

Bitte erkundigt euch vorab bei den Veranstaltern, ob die Treffen stattfinden.

Mannheim **Thomas Prinz**
Tel.: (0 62 71) – 28 30$_p$
Ewald Rieger
Tel.: (0 62 39) – 92 01 85$_p$
Treffen: jeden 1. Dienstag im Monat
Vereinslokal Segelverein Mannheim e.V. Flugplatz Mannheim–Neuostheim

München **Bernd Paysan**
Tel.: (0 173) – 82 06 874
`bernd@net2o.de`
Treffen: Jeden 4. Donnerstag im Monat um 20:00 auf `http://public.senfcall.de/forth-muenchen`, Passwort `over+swap`, im Sommer auch wieder um 19:00 im La Capannina, Weitlstraße 142, 80995 München

Hamburg **Ulrich Hoffmann**
Tel.: (04103) – 80 48 41
`uho@forth-ev.de`
Treffen alle 1–2 Monate in loser Folge
Termine unter: `http://forth-ev.de`

Ruhrgebiet **Carsten Strotmann**
`ruhrpott-forth@strotmann.de`
Derzeit keine Treffen.

Dienste der Forth–Gesellschaft

Nextcloud `https://cloud.forth-ev.de`

GitHub `https://github.com/forth-ev`

Twitch `https://www.twitch.tv/4ther`

µP–Controller–Verleih **Kohl–Schöpe, Strotmann**
`microcontrollerverleih@forth-ev.de`
`mcv@forth-ev.de`

Spezielle Fachgebiete

Forth–Hardware in VHDL microcore (uCore)	**Klaus Schleisiek** Tel.: **(0 58 46) – 98 04 00 8**$_p$ `kschleisiek@freenet.de`
KI, Object Oriented Forth, Sicherheitskritische Systeme	**Ulrich Hoffmann** Tel.: **(0 41 03) – 80 48 41** `uho@forth-ev.de`
Forth–Vertrieb volksFORTH ultraFORTH RTX / FG / Super8 KK–FORTH	Ingenieurbüro **Klaus Kohl–Schöpe** Tel.: **(0 82 66)–36 09 862**$_p$

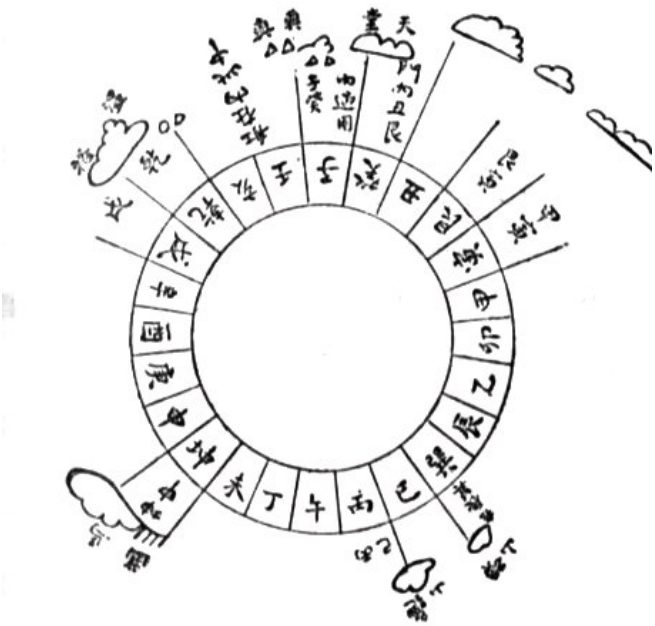
Termine

Donnerstags ab 20:00 Uhr
Forth–Chat net2o `forth@bernd` mit dem Key `keysearch kQusJ`, voller Key:
`kQusJzA;7*?t=uy@X}1GWr!+0qqp_Cn176t4(dQ*`

Jeder 1. Montag im Monat ab 20:30 Uhr
Forth–Abend
Videotreffen (nicht nur) für Forthanfänger
Info und Teilnahmelink: E–Mail an `wost@ewost.de`

Jeder 2. Samstag in ungeraden Monaten
ZOOM–Treffen der Forth2020 Facebook–Gruppe
Infos zur Teilnahme: `www.forth2020.org`

Details zu den Terminen unter `http://forth-ev.de`



Möchten Sie gerne in Ihrer Umgebung eine lokale Forthgruppe gründen, oder einfach nur regelmäßige Treffen initiieren? Oder können Sie sich vorstellen, ratsuchenden Forthern zu Forth (oder anderen Themen) Hilfestellung zu leisten? Möchten Sie gerne Kontakte knüpfen, die über die VD und das jährliche Mitgliedertreffen hinausgehen? Schreiben Sie einfach der VD — oder rufen Sie an — oder schicken Sie uns eine E–Mail!

Hinweise zu den Angaben nach den Telefonnummern:
Q = Anrufbeantworter
p = privat, außerhalb typischer Arbeitszeiten
g = geschäftlich
Die Adressen des Büros der Forth–Gesellschaft e.V. und der VD finden Sie im Impressum des Heftes.

Jahrestagung 2026

Schnell noch anmelden!

Am Wochenende nach Ostern 09. – 12. April

findet die nächste Jahrestagung und die Mitgliederversammlung der Forth–Gesellschaft e.V. statt. Die Einladung zur MV liegt dem Heft bei. Wie ihr dort seht, gibt es einiges zu regeln.

Programm und Preise?

`tagung.forth-ev.de`

Wir haben Halbpensionspreise, also Übernachtung inklusive Frühstück.

Es wird eine Tagungspauschale[1] erhoben. Diese beinhaltet den Konferenzraum und dessen Ausstattung, das Mittagessen, die Erfrischungsgetränke, Obst, Kaffee und Kuchen in Pausen. Am Tag der An– und Abreise wird nur die halbe Pauschale erhoben.

Das Abendessen können wir im Haus haben oder auswärts Essen gehen.

Gleich nebenan im Bermuda–Dreieck sind die Restaurants und Bars von früh bis spät geöffnet.

Foto: Joker.mg

Wo?

DJH Jugendherberge
Humboldtstraße 59–63
44787 Bochum, Germany.

Das ist mitten in **Bochum.**

Foto: Raenmaen

Buchen!

Beeil dich, einige Plätze gibt es noch.

`tagung.forth-ev.de`

Oder E–Mail an `secretary@forth-ev.de`

Oder du rufst mich an und wir klären das:
049 157 5505 1777

Anreise

Die DJH Jugendherberge Bochum ist etwa 2 km vom Bochumer Hauptbahnhof entfernt. Mit dem Taxi sind es ungefähr 6 Minuten.

Parkhäuser für den eigenen Wagen sind gegenüber der Herberge und nebenan in der City und rund um die Uhr geöffnet.

Und wer will, kann über Dortmund einfliegen, von da sind es 35 km zum Tagungsort, mit dem Auto ungefähr 30 Minuten und es gibt eine direkte S–Bahn–Verbindung zum Hbf–Bochum, die Fahrt dauert etwa 40 Minuten.

Auch über den Flughafen Düsseldorf ginge das, die Fahrt mit der Bahn von dort zum Hbf–Bochum dauert etwa 50 Minuten.

Wir sehen uns!

mk

[1] Damit kann man an der Tagung auch teilnehmen, ohne im Haus zu übernachten.