Vierte Dimension

3/2018

Das Forth–Magazin

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

In dieser Ausgabe:

Dienstleistungen und Produkte fördernder Mitglieder des Vereins

tematik GmbH
Technische
Informatik

Feldstraße 143
D–22880 Wedel
Fon 04103 – 808989 – 0
Fax 04103 – 808989 – 9
mail@tematik.de
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.

LEGO RCX–Verleih

Seit unserem Gewinn (VD 1/2001 S.30) verfügt unsere Schule über so ausreichend viele RCX–Komponenten, dass ich meine privat eingebrachten Dinge nun Anderen, vorzugsweise Mitgliedern der Forth–Gesellschaft e. V., zur Verfügung stellen kann.

Angeboten wird: Ein komplettes LEGO–RCX–Set, so wie es für ca. 230,–€ im Handel zu erwerben ist.

Inhalt:

1 RCX, 1 Sendeturm, 2 Motoren, 4 Sensoren und ca. 1.000 LEGO Steine.

Anfragen bitte an
Martin.Bitter@t-online.de

Letztlich enthält das Ganze auch nicht mehr als einen Mikrocontroller der Familie H8/300 von Hitachi, ein paar Treiber und etwas Peripherie. Zudem: dieses Teil ist „narrensicher"!

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

Ingenieurbüro
Klaus Kohl–Schöpe

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

FORTH–Software (volksFORTH, KKFORTH und viele PDVersionen). FORTH–Hardware (z.B. Super8) und Literaturservice. Professionelle Entwicklung für Steuerungs– und Meßtechnik.

KIMA Echtzeitsysteme GmbH

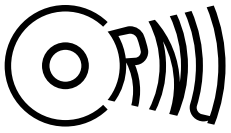
Güstener Straße 72 52428 Jülich
Tel.: 02463/9967-0 Fax: 02463/9967–99
`www.kimaE.de` info@kimaE.de

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

FORTecH Software GmbH
Entwicklungsbüro Dr.–Ing. Egmont Woitzel

Bergstraße 10 D–18057 Rostock
Tel.: +49 381 496800-0 Fax: +49 381 496800-29

PC–basierte Forth–Entwicklungswerkzeuge, comFORTH für Windows und eingebettete und verteilte Systeme. Softwareentwicklung für Windows und Mikrocontroller mit Forth, C/C++, Delphi und Basic. Entwicklung von Gerätetreibern und Kommunikationssoftware für Windows 3.1, Windows95 und WindowsNT. Beratung zu Software–/Systementwurf. Mehr als 15 Jahre Erfahrung.



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.

Hier könnte Ihre Anzeige stehen!

Wenn Sie ein Förderer der Forth–Gesellschaft e.V. sind oder werden möchten, sprechen Sie mit dem Forth–Büro über die Konditionen einer festen Anzeige.

`Secretary@forth-ev.de`

Titelbild Autor Text

Quelle

Impressum

Name der Zeitschrift

Vierte Dimension

Herausgeberin

Forth–Gesellschaft e. V.
Postfach 32 01 24
68273 Mannheim
Tel: ++49(0)6239 9201-85, Fax: -86
E–Mail: Secretary@forth-ev.de
Direktorium@forth-ev.de
Bankverbindung: Postbank Hamburg
BLZ 200 100 20
Kto 563 211 208
IBAN: DE60 2001 0020 0563 2112 08
BIC: PBNKDEFF

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

Manuskripte und Rechte

Berücksichtigt werden alle eingesandten Manuskripte. Leserbriefe können ohne Rücksprache wiedergegeben werden. Für die mit dem Namen des Verfassers gekennzeichneten Beiträge übernimmt die Redaktion lediglich die presserechtliche Verantwortung. Die in diesem Magazin veröffentlichten Beiträge sind urheberrechtlich geschützt. Übersetzung, Vervielfältigung, sowie Speicherung auf beliebigen Medien, ganz oder auszugsweise nur mit genauer Quellenangabe erlaubt. Die eingereichten Beiträge müssen frei von Ansprüchen Dritter sein. Veröffentlichte Programme gehen — soweit nichts anderes vermerkt ist — in die Public Domain über. Für Text, Schaltbilder oder Aufbauskizzen, die zum Nichtfunktionieren oder eventuellem Schadhaftwerden von Bauelementen führen, kann keine Haftung übernommen werden. Sämtliche Veröffentlichungen erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes. Warennamen werden ohne Gewährleistung einer freien Verwendung benutzt.

Liebe Leser,

50 Jahre Forth — ein langer Weg. Wohin wird es noch gehen? Was ist der Stand der Dinge? In Schottland wurde das dieses Jahr genauer betrachtet. Auch von Charles Moore, der wirklich dort hingekommen war. Aber seht selbst, im Heft sind die Links zu den Videos angegeben.

Das Forth–Standard–Committee hat dort ebenfalls getagt. Die Links zu den Dokumenten darüber findet ihr auch im Heft. Hoffentlich gelingt es auch noch, die Autoren all der Proposals zu einem Beitrag hier im Heft zu bewegen. Damit unser Magazin weiter leben kann. Denn wir drucken ein Heft immer dann, wenn es fertig ist. :-) Das gelingt nicht immer 4x im Jahr, auch wenn das weiter angestrebt wird.

Peter Minuth alias Peter Forth hat den Finger am Forth–Puls, von Süd–Amerika aus, via Facebook, so ziemlich in jede Ecke der Welt. Und so konnte er zusammentragen, was da auf dem Win32Forth so geht. Schaut mal rein bei ihm im Netz.

Jostein Skjelstad lässt uns teilhaben an mathematischen Problemen und deren Lösung in Forth, was ich ja immer sehr erhellend finde: Nachvollziehen können, wie so etwas gemacht wird. Vielen Dank dafür! Und gerne mehr davon.

Von Wil Baden stammen die Überlegungen zum Forth Control Flow. Seine Art, Forth zu codieren, finde ich vorbildlich. Leider weilt er ja nicht mehr unter uns. Und es wird schwieriger, an seine Codes zu kommen. Hoffentlich sichert das jemand weiterhin. Das würde ich sehr begrüßen.

Rafael Deliano lässt uns teilhaben an kniffligen Aufgaben aus der Praxis. Die dann elegant mit Forth gelöst werden konnten. Die PS/2 lesen zu können, ist enorm praktisch: Bit–Banging ist da nötig, weil keine MCU das einfach so kann, diese Mischung aus UART und SPI. Doch ist es keine Hexerei.

Und wie man FreeDOS hier und da aufbohrt, zeigt uns Fred Behringer. Zugriff aus dem Forth heraus in die Shell ist nicht jedermanns Sache, aber machbar; ein Vorteil dieser Maschinen: Man hat sie im Griff.

Und dann freue ich mich mit Euch schon auf unsere Jahrestagung 2019. Es geht nach Worms! Ewald und Andrea Rieger richten die Tagung aus. In einem Weingut — lasst Euch das nicht entgehen!

Möge Forth weitere 50 Jahre bestehen!

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-2018-03

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

Ulrich Hoffmann
Bernd Paysan
Carsten Strotmann

Kontakt: Direktorium@Forth-ev.de

50 Jahre Forth

Die Euro–Forth fand dieses Jahr in Schottland statt. Wie hätte man den 50. besser feiern können, als am Firth of Forth mit Blick auf die Forth Bridge? Und mit Charles Moore, dem Erfinder. Und so ändern sich die Zeiten. Hätte man zu Zeiten der Forth–Anfänge noch im Journal nachgelesen was sich da so auf der Tagung zugetragen hat, schaut man heute ins Video. Ihr könnt sozusagen nachträglich teilnehmen. Bernd war fleißig und hat euch alles auf dem Wiki des Forth–eV bereitgestellt.

https://wiki.forth-ev.de/doku.php/events:ef2018:start

Abbildung 1: Charles Moore, September 2018

EuroForth 2018 proceedings

Um euch schon mal einen Ausblick zu geben, was da alles präsentiert worden ist, und wie so eine Euro–Forth zusammengestellt wird, ist hier das Vorwort der *EuroForth 2018 proceedings*, die Anton Ertl zusammengestellt hat, abgedruckt.

> EuroForth is an annual conference on the Forth programming language, stack machines, and related topics, and has been held since 1985. The 34th EuroForth finds us in *Edinburgh* for the first time. The two previous EuroForths were held on *Reichenau Island*, Germany (2016) and in *Bad Vöslau*, Austria (2017). Information on earlier conferences can be found at the EuroForth home page http://www.euroforth.org/. Since 1994, EuroForth has a refereed and a non–refereed track. This year there were three submissions to the refereed track, and two were accepted (67% acceptance rate). For more meaningful statistics, I include the numbers since 2006: 27 submissions, 19 accepts, 70% acceptance rate. Each paper was sent to three program committee members for review, and they all produced reviews. The reviews of all papers are anonymous to the authors: This year two submissions were co-authored by program committee members, one of them by the program chair; the papers were reviewed and the final decision taken without involving the authors. Ulrich Hoffmann served as secondary chair and organized the reviewing and the final decision for the paper written by the program chair. I thank the authors for their papers and the reviewers and program committee for their service. One paper was submitted to the non-refereed track in time to be included in the printed proceedings. ...
>
> Workshops and social events complemented the program. This year's Euro-Forth was organized by *Paul E. Bennett.* Anton

Inhaltsverzeichnis der Druckversion

Refereed Papers

- Ulrich Hoffmann and Andrew Read: A descriptor based approach to Forth strings.
- M. Anton Ertl and Bernd Paysan: Closures — the Forth way.

Non-Refereed Papers

- M. Franck Bensusan: Method dispatch in Oforth.

Late Non-Refereed Papers

- Peter Knaggs: A List Toolkit
- Ulrich Hoffmann and Andrew Read: Forth: A New Synthesis
- Progress Report: Growing Forth with seedForth

Presentation Slides

- Bernd Paysan: net2o: MINOS2 GUI, $quid "crypto"
- M. Anton Ertl: Software Vector Chaining
- Phillip Eaton: Programming in Forth on the Vectrex

The online proceedings also contain papers and presentations that were too late to be included in the printed proceedings.

Im Programm–Komitee waren: M. Anton Ertl, TU Wien (chair); David Gregg, Trinity College Dublin; Ulrich Hoffmann, FH Wedel University of Applied Sciences (secondary chair); Phil Koopman, Carnegie Mellon University; Jaanus Pöial, Tallinn University of Technology; Bradford Rodriguez, T-Recursive Technology und Bill Stoddart.

http://www.complang.tuwien.ac.at/anton/euroforth/ef18/papers/

Forth–200X–Treffen auf der EuroForth 2017

Ja, es geht um das *2017er–Treffen*, nicht das 2018er. An den beiden Tagen vor der *EuroForth 2017 in Bad Vöslau* fand das Treffen des Forth˜200X–Komitees statt, das am nächsten Forth–Standard arbeitet.

Zunächst beschloss das Komitee, den Prozess für Vorschläge zu modernisieren. Neue Vorschläge (Proposals) sollen auf http://forth-standard.org/ gemacht werden,

und nicht mehr in comp.lang.forth und auf der Forth200x–Mailingliste. Dadurch fällt es leichter, einen Überblick über die Proposals und Diskussionsbeiträge zu bekommen.

Es stand ein Erweiterungsvorschlag zur Abstimmung und wurde angenommen: *Quotations* sind namenlose Definitionen innerhalb anderer Definitionen. Eine Quotation beginnt mit `[:` und endet mit `;]` und hinterlässt das xt der Quotation auf dem Stack. Die namenlose Definition `:noname ... ;` funktioniert so ähnlich, hat aber innerhalb von Definitionen eine andere Bedeutung.

Ein Beispiel für die Verwendung von Quotations ist:

```
: hex. ( u -- )
base @ >r
[: hex u. ;] catch
r> base ! throw ;
```

Das ist äquivalent zu:

```
: hex.-helper ( u -- ) hex u. ;
: hex. ( u -- )
base @ >r
['] hex.-helper catch
r> base ! throw ;
```

Quotations erlauben also in erster Linie eine bequemere Verwendung von `catch` und Wörtern, denen Execution-Tokens übergeben werden. Einige sehen das zwar als unwichtig an, aber anderen ist es wiederum so wichtig, dass sie dieses Feature mit Hilfe von Rücksprung–Adressenmanipulation, ein nicht–standardisiertes und z.B. in SwiftForth nicht allgemein funktionierendes Verfahren, nachgebaut haben. Z.B. wurde zum Iterieren über die Elemente einer Liste folgende Konstruktion verwendet:

```
: foo bar list> bla blub ;
```

`LIST>` manipuliert Rücksprungadressen, um die Worte dahinter einmal pro Listenelement auszuführen. Stattdessen kann man jetzt

```
: foo bar [: bla blub ;] map-list ;
```

schreiben. Wobei `map-list` ein xt nimmt und es einmal pro Listenelement ausführt. Quotations bieten also eine nun standardisierte und dank deutlicher Syntax verständlichere Alternative.

Ein Thema in diesem Zusammenhang ist das Zusammenspiel von Quotations und Locals. Innerhalb von Quotations darf man auf Locals zugreifen, die in der Quotation definiert wurden, aber der Zugriff auf Locals äußerer Definitionen ist nicht standardisiert.

Weiters beschloss das Komitee, die *ambiguous conditions*[1] zu überarbeiten: Der Standard sollte für solche Fälle entweder Garantien geben und sie damit standardisieren, oder eine Erklärung bieten, warum der Fall nicht standardisiert wird. Das Komitee hat dann gleich einmal fünf *ambiguous conditions* überarbeitet, es warten aber noch viele weitere.

In diesem Zusammenhang wurde auch beschlossen, das *exception wordset* verpflichtend zu machen. Das erlaubt es, in einigen Fällen, die bisher nicht standardisiert waren, ein `throw` eines bestimmten Codes vorzuschreiben.

Neben diesen Beschlüssen gab es noch Diskussionen zu einer breiten Palette von Themen, u.a. auch zu *Recognizern* und *Multitasking*. Anton

eForth 34.1, Naken Assembler Version

Ting's eForth für den MSP430G2553, das *430eForth43*, gibt es bekanntlich seit diesem Jahr mit ausdrücklicher Zustimmung von Ting auch als Version für den Naken Assembler von Micheal Kohn. eForth ist damit aus den Krallen solch aufgeblähter IDEs, wie dem des Code Composers Studio (CCS), befreit. Dieses eForth ist das minimalste komplette Forthsystem für diesen kleinen Chip. Manfred Mahlow (MM) hat sich damit näher befasst.[2]

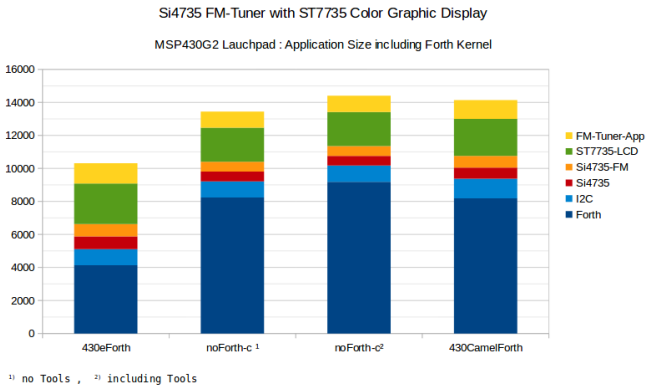


Abbildung 2: Vergleich der Speicherbelegungen

https://github.com/mikalus/eForth43-msp430g2553-naken

Klein, schnell und kompakt wie es ist, kann man damit aber auch in die *Reset–Falle des Flash* tappt. Wir hatten in unserem Magazin darüber berichtet.[3] Damit man bei der interaktiven Entwicklung auch im Flash–Speicher nach Herzenslust herum probieren kann, sind kleine Werkzeuge erforderlich, um das Flash immer wieder leeren zu können. MM hatte auf der Forthtagung 2017 schon vorgestellt wie das sehr platzsparend an das eForth angefügt werden kann, ohne dessen Kern aufzublähen, nämlich als nachladbares Tool. Und nun hat er das für die Naken–asm–Ausgabe des eForth aufbereitet. Als Tiny–Toolbox, zum nachladen. Das Projekt ist auf Github hinterlegt. mk

[1] Fälle, die nicht standardisiert sind: Systeme können unterschiedliches Verhalten zeigen, und Standard–Programme können sich nicht auf ein bestimmtes Verhalten verlassen.

[2] https://wiki.forth-ev.de/lib/exe/fetch.php/events:430eforth-tips_tools_tests2017.pdf

[3] Vierte Dimension 3/2017, Das RAM–ROM–Dilemma von interaktivem Forth in kleinen MCUs, M.Kalus; https://wiki.forth-ev.de/lib/exe/fetch.php/vd-archiv:4d2017-03.pdf

Forth in Südamerika

Tja, ich habe viele Anekdoten darüber und alten Kram, was ich auf Forth alles gebastelt habe. Ihr wisst es nicht, Forth–Freunde aus Deutschland, aber vieles von meiner Forth–Erfahrung habe ich von Euch! Ich bin der verlorene Schüler der VD in Südamerika. Als ich mit meinem Studium anfing, hatte ich einen Kameraden, der in München die VD abonnierte und per Post an mich rüberschickte — vor dem Internet. Als ich meinen ersten 8086–Computer kaufte — ich war in Deutschland und erstand einen CommodorePC — war die erste Software, die ich darauf installierte, das VolksForth, das ich bis heute noch laufen hab. Ich habe es letzte Woche sogar im Raspberry laufen lassen. Auch das Win32forth läuft dort.

All meine professionelle Software habe ich auf LMI–URforth 16– und 32-Bit programmiert. Es war eine schöne Zeit: Forth auf DOS, da war man König, man hatte Zugriff auf jedes einzelne Bit und Byte, man konnte wirklich alles machen. Platinen für den ISA–BUS waren damals so einfach! Heute ist alles sehr geschlossen.

Fast alle meine technische Literatur war auf Deutsch, in Astronomie, wie auch in Elektronik. Damals war ich Abonent für alles über Sterne und Weltraum, c't, Chips, usw. Mit Kalifornien war ich verbunden über BBS für 1 $US/Minute, die ersten internationalen Verbindungen! Später kam das Arpanet, 100 $US für 300 Minuten. Das Modem war 300 Baud!

Ich könnte über alte Projekte etwas berichten, aber es würde mich mehr freuen etwas über Forth auf dem Raspberry–PI zu schreiben. Ich mache jetzt mehrere Experimente damit. Wenn ich was Interessantes zusammenbringe, melde ich mich wieder. Peter

[Wir sind gespannt, was du alles Schönes machen wirst. Mögen deinem Beitrag in diesem Heft, über forthige Node–MCUs, noch weitere Folgen. Viele Grüße nach Südamerika von der FG!]

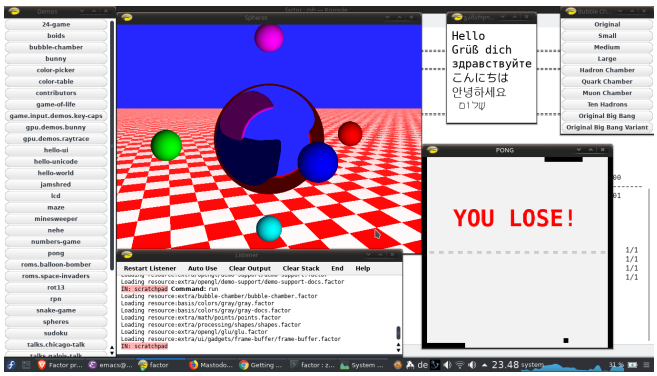
Factor Version 0.98 veröffentlicht

Fünf Jahre nach dem letzten Version ist die Forth–ähnliche Sprache *Factor* nun in der Version 0.98 verfügbar.

Factor ist eine Stack–basierte Sprache mit einem Mix von guten Ideen aus Lisp und Smalltalk. Factor gibt es für Windows, macOS und Linux, jeweils in 64bit– und 32bit–Versionen. Der Factor–Compiler erzeugt kompilierte Maschinensprache–Programme (Stand–Alone) für alle unterstützten Plattformen.

Die Factor Programmier–Umgebung ist ein grafisches Programm mit hytertext–verlinktem Quellcode–Browser. Diese Funktion macht es dem Programmierer einfach, Factor–Quellcode zu finden, zu lesen und zu verstehen.

Factor kommt mit einer umfangreichen Sammlung an Bibliotheks–Routinen und Beispiel–Programmen — probiere `demos run` im Factor–Listener.



In der Version 0.98 wurde Factor behutsam weiterentwickelt: Es gibt eine ganze Reihe neuer Bibliotheken, Unterstützung für die TouchBar auf modernen Apple–Laptops, neue Beispiel–Programme, sowie eine bessere Integration in den Emacs–Editor und Support für neue Editoren wie Microsoft–Visual–Studio–Code.

Eine detailierte Übersicht über die Neuerungen in der Version 0.98 bietet der Blog–Artikel *Factor 0.98 now available.* Und für die Wissbegierigen gibt es ein ausführliches Tutorial.

Links

http://factorcode.org/

https://re-factor.blogspot.com/2018/07/factor-098-now-available.html

https://andreaferretti.github.io/factor-tutorial/

TIOBE Index Oktober 2018

FORTH steht im Ranking für Oktober diesen Jahres auf Platz *42*, mit Worten *zweiundvierzig*! Es hieß schon immer: „FORTH ist die *Antwort*, was ist die Frage?“ :-D Klaus Zobawa

https://de.wikipedia.org/wiki/42_(Antwort)

https://www.tiobe.com/tiobe-index//

(Weitere Meldungen sind über das Heft verteilt zu finden . . .)

Win32forth+visualFORTH controllieren Node–MCU

Peter Minuth

Vor ca. 2 Jahren erwachte mein Interesse am IOT mit „PunyForth“, einem Forthdialekt für den ESP8266, geschrieben von Attila Magyar. *Obwohl diese Implementierung sich sehr gut für meine Zwecke, das Internet of Things (IOT) eignete, entdeckte ich in der Anwendung einen Nachteil. Das Verbindungsinterface wurde in Python und nicht in Forth geschrieben.*

Tatsächlich, wenn alle Tools von Expressif auf Python verfügbar sind, ist es für den Entwickler einfacher, auf dem selben Weg zu bleiben. Aber für einen alten Forthliebhaber wie mich, bedeutete dies nicht nur Herzschmerzen, sondern auch eine unnötige Komplikation im Umgang mit zwei verschiedenen Sprachen. Kurz vor einem „Gehirnkurzschluss“ :) entschloss ich mich, das Interface auf Forth neu zu schreiben.

Im Zoo der Programmiersprachen

Als Ersatz für *Python,* die Schlange, könnte gut ein anderes Tier passen, ein echtes Mammut! Denn mit mehr als 5400 Worten kann man *Win32Forth* von Tom Zimmer nicht besser bezeichnen. Ein Forth, das dank dieser Vielfalt immmer allzeit bereitsteht, um diverse Projekte zu verwirklichen. Nach einer Weile Herum-geForthe in den Win32forth–Quellen, adaptierte ich ein USB Terminal, das sich prima an mein Projekt anpasste — der Mammut war fit und lebendig!

Die Kommunikation zwichen PC und Target lief vom ersten Moment an fehlerfrei. Als ich die Befehle im Win32console–Terminal eintippte, und das `OK>` von Punyforth zurückkam, fühlte ich mich wieder zuhause! [1]

Wer einen ESP8266[2] hat, will doch Wi–Fi![3] So war das nächste Ziel ganz klar (Abb.1). Aus der Fundgrube von Win32–Sourcen kam nochmals die Lösung: Der Folder *Internet* hat die komplette socket library von Thomas Dixon. So steht das TCP–IP Protokoll gleich zur Verfügung ... Zu meinem Glück hatte auch Attila im Punyforth ein Beispiel vom TCP–IP REPL[4], also die Interpreter–Quelle wird statt aus dem UART vom TCP–IP–Stack geholt, und beantwortet. Diesen TCP–IP–REPL hatte ich schon im *TeraTerm*–Terminal getestet, und es lief perfekt. So entstand mit wenig Programmierarbeit die Verbindung von beiden Rechnern über Wi–Fi. Aber der eigentliche Treffer dieser Kombination ist, dass beide Systeme durch den eigenen Interpreter kommunizieren. Das heißt, unbeschränkte Möglichkeiten von Makros, also Wörter hin und her zu schicken und beliebig zu erweitern! Welche Sprache außer Forth kann sowas?

Und das alles entstand mit den schon verfügbaren Forth–Tools plus Win32–Libraries, die ich nur anpassen musste.

In diesem Moment blickte ich auf dem Bildschirm, sah die beiden CPUs in lebendiger Wi–Fi–Kommunikation ... und sagte zu mir selbst „Willkommen Forth im 21. Jahrhundert!“

Dieses System hat mir die Möglichkeit gegeben, den ESP in verschiedenen Geräten meines Hauses als Controller für Steckdosen, Lampen, Wasserpumpen, usw. erfolgreich zu nutzen. Im Internet findet man tausende von Artikeln über IOT und Hausautomation auf Lua, Mphyton, C, usw. Ich aber bin stolz darauf, dass bei uns zu Hause — mit einer richtigen IP — die Steckdosen mit einem herzlichen: „OK>“ grüßen.

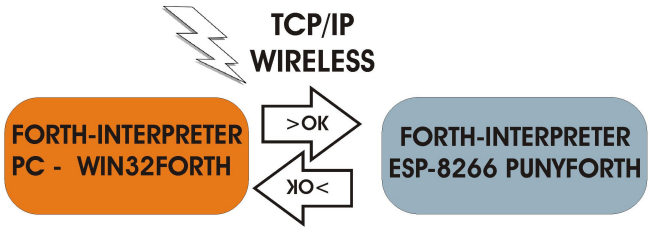


Abbildung 1: Schema der Kommunikation zwischen den Forth–Systemen

Wie das graphische User–Interface entstand

Eines Tages war ich entschlossen, ein Interface in *visualFORTH* auszuprobieren (für ein anderes Programm). VisualForth (vF) ist eine Implementierung von Dirk Bruehl auf Win32forth, das den *FormEditor* erweitert, und die Entwicklung von Programmen und GUI erleichtert.

Obwohl in der Hausautomation alles perfekt mit Punyforth funktionierte, fand ich die gekreuzte Bennenung der verschiedenen IO–Pins ein wenig frustrierend. Denn die Pins haben eine andere Nummerierung als die Ports.

Ich musste bei der Programmierung des ESP immer das Bild mit allen In–Out–Pins vor Augen haben, damit ich keinen Fehler verursache, der vielleicht die NodeMCU

[1] Punyforth hat als Default–Rückmeldung `(stack)` , und ich tauschte es natürlich in den normalen `OK>` Prompt um.)

[2] NodeMCU is an open source IoT platform. It includes firmware which runs on the ESP8266 Wi-Fi SoC from Espressif Systems, and hardware which is based on the ESP-12 module.

[3] Wi-Fi bezeichnet sowohl ein Firmenkonsortium, das Geräte mit Funkschnittstellen zertifiziert, als auch den zugehörigen Markenbegriff.

[4] Read–Eval–Print Loop (REPL)

zerstören würde. (Und aufpassen, dass PunyForth case sensitive ist!) Es wurde für mich klar, dass PunyForth noch ein kleiner Schub fehlte, bis es komplett „im 21. Jahrhundert“ landete.

Es war keine grosse Denkarbeit nötig: Wenn man visualFORTH in einem Fenster hat, Win32forth auf dem Fenster daneben über TCP IP verbunden mit der NodeMCU, und in der Mitte ein Bild mit allen IO–Pins zur Erinnerung ... Eureka! Problem und Lösung erschienen zusammen vor meinen Augen. (Abb.2 und Abb.5)

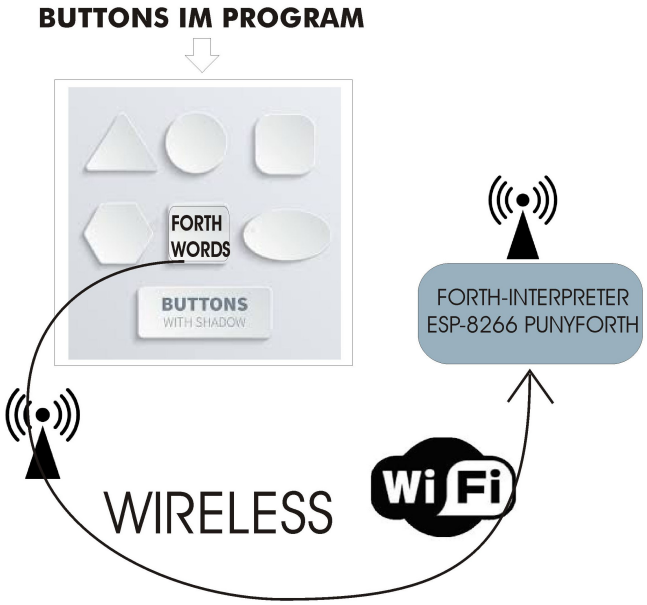


Abbildung 2: Forth–Kommunikation im IOT

VisualFORTH erklärt sich selbst bei der Benutzung

Der erste Schritt war es, das Bild vom ESP8266 mit den IO–Descriptions in einer Form vom vF einbinden. Man definiert einen Bitmap–Button (zusammen mit CTRL) und setzt dieses Photo als Hintergrund in der Mitte der Form. Dann setzt man an jedem Pin einen normalen Button. VisualFORTH kann über den Befehl *Properties* diesen Buttons Namen geben, und Code zuordnen. Dieser Code ist nichts anderes als „... `GPIO-mode`... etc“ im Falle, man programmiert einen I/O, also Forth–Befehle wie `GPIO-read` , oder `GPIO-write` im Falle, dass man vom Pin lesen oder schreiben will, also einen Ausgang setzen. Natürlich kann *jedes* beliebige FORTH–Wort, das Punyforth kennt, ausgeführt werden, denn wir sprechen direkt zum Interpreter! Also ein Button kann ein Programm aufrufen, das Messungen am A/D–Port jede 50 ms ausführt, und der Messwert kommt einfach mit .[5] im Win32forth an! So einfach ist der ganze Prozess. (Abb.3)

In visualFORTH habe ich im File–Menu die Option `TEST`, oder auch im Toolbar ein „Läufer“–Icon, das ist Synonym.

[5] „dot“, also dem PRINT im Forth.

Form Copy–to–Clipboard

In einem anderem Fenster habe ich die Win32forth–IDE, also den Editor. Dann mache ich *File : NEW FILE* und einfach ein *Paste*, um den Code zu kopieren. Die Win32Forth–IDE hat einen Button zum Kompilieren. So kann man alles gleich testen. (Abb.3)

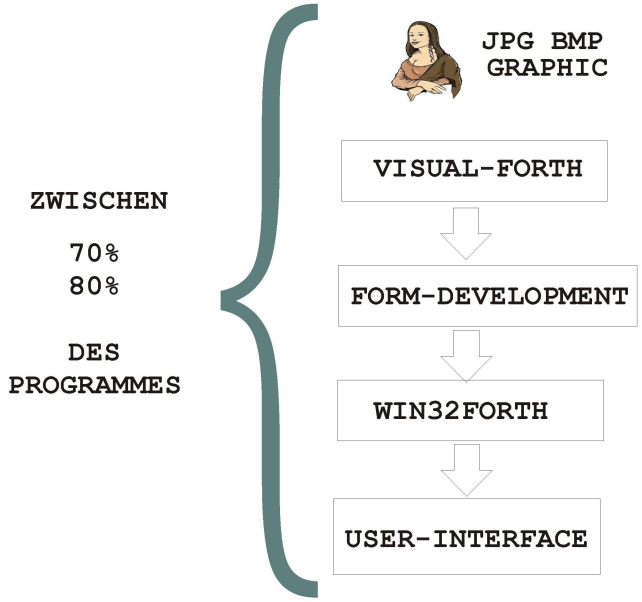


Abbildung 3: Programmiercyclus in Win32Forth + visualFORTH

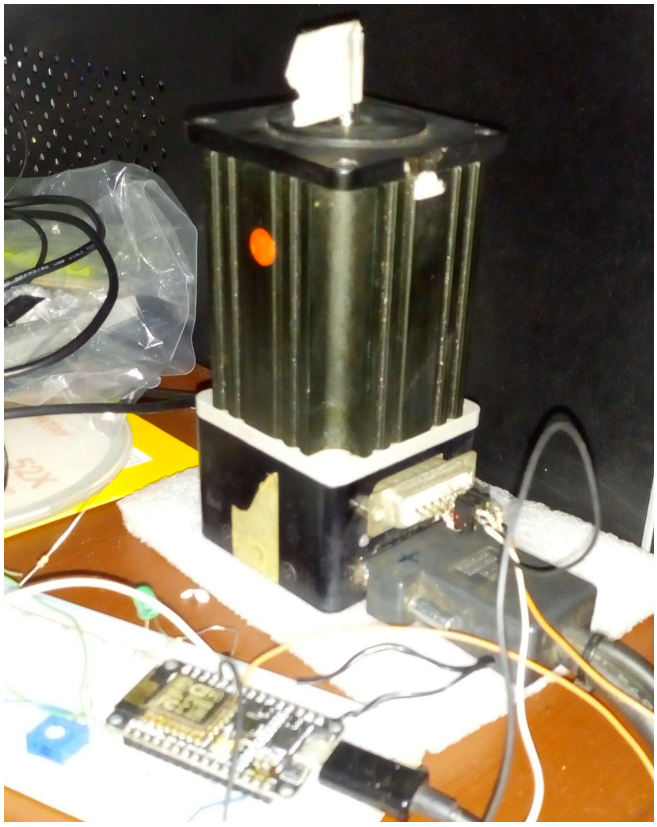


Abbildung 4: Einer meiner Tests: Servomotor ansteuern.

Wie funktioniert der Kontroll–Panel?

Egentlich gibt es nicht nur eine einzige Art und Weise diesen Kontroll–Panel (Abb.5)zu benutzen. Es dient als ein Beispiel, das erweitert oder für spezifische oder ganz andere Zwecke adaptiert werden kann. Ich sehe interessante Anwendungen dafür: Lehrer von technischen Schulen könnten diese Idee in eine wunderbare Experimentierplattform weiterentwickeln. Für einen Physiker oder Biologen im Labor könnte die NodeMCU Instrumente oder Prozesse überwachen und steuern, und per Wi-Fi zum PC für weitere Überarbeitung senden. Auch Maker, die auf Steckplatinen[6] Probeschaltungen mit Forth kontrollieren wollen, können hier Inspiration finden. (Abb.4)

Danksagung

Zum Schluss möchte ich mich herzlichst bei allen Autoren bedanken, die dieses phantastische Tool geschrieben haben. Danke also an Tom Zimmer für Win32forth, an EzraBoyce für den FormEditor, Dirk Bruehl für sein visualFORTH und Attila Magyar für sein Punyforth.

Links

https://sourceforge.net/projects/win32forth/
http://www.visualforth.org/
https://github.com/zeroflag/punyforth

Listings

... gibt es beim Autor. Die Änderungen im Win32Forth, um das hier Vorgestellte zu verwirklichen, sind verstreut in den Quellen. Und im visualFORTH ist der Quellcode in den Forms enthalten. Das lässt sich nicht so ohne Weiteres abdrucken. Aber angehängt sind zwei kleine Code–Beispiele, die zeigen, wie das Ganze funktioniert.

Für den Blinker gibt es außerdem ein Video auf YouTube. Es werden zwei Node–MCUs angesprochen.

https://www.youtube.com/watch?v=NvZHxFj0pt8

Kontakt

eMail: peter4th2017@gmail.com
Facebook: Peter Forth

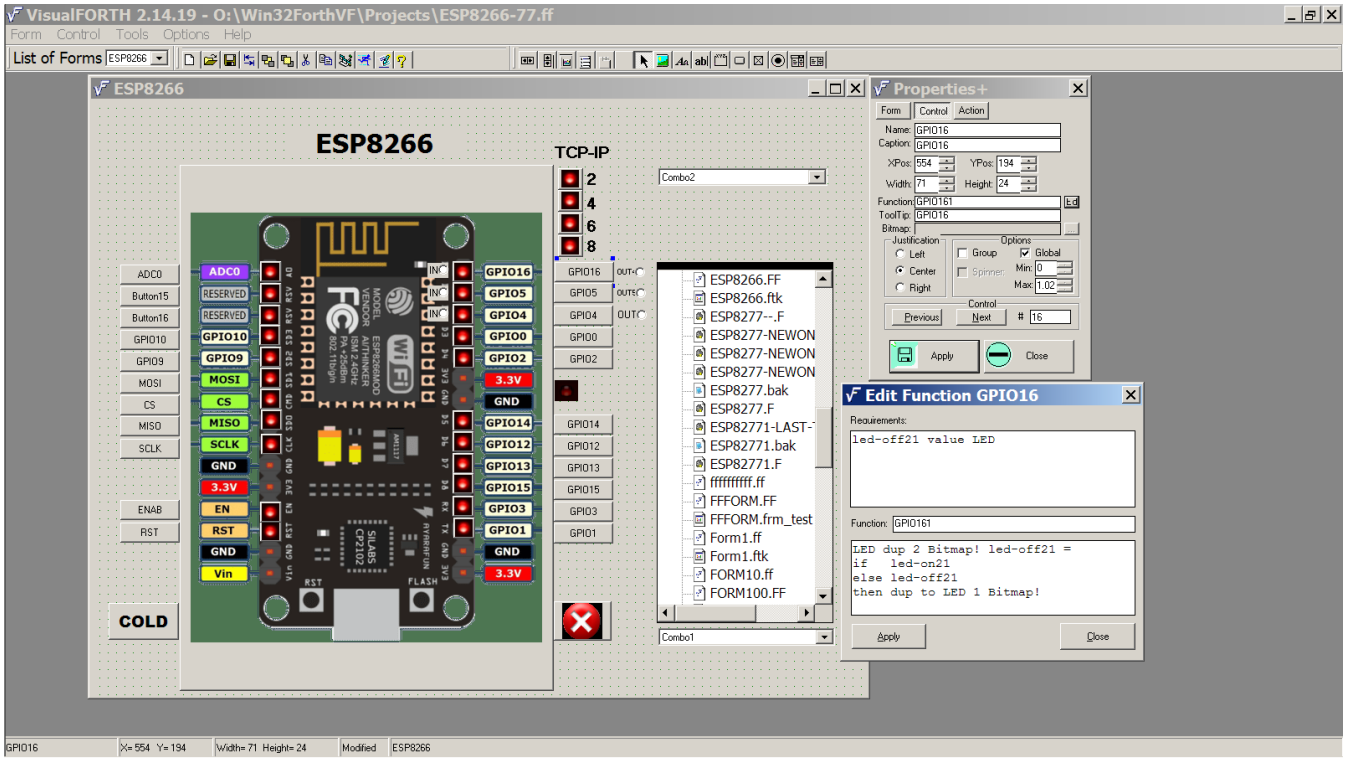


Abbildung 5: Das ESP–Kontrollpanel

Zwei kleine Beispiele

TELTYPE an Node–MCU

„Hallo world“–Blinker, Werte auslesen, Befehle im Node definieren, Funktionen an- und ausschalten ...

```
: telcr ( -- )   13 telsend 10 telsend ; ( CR ans Telnet )
: teltypeCR ( string --)   teltype telcr ; ( teltype ist ein type ueber telnet)
: boton15   s" 2 blink " teltype telcr ; ( led 2 blinkt)
```

[6] den „breadboards“

```
: boton16   s" 0 blink " teltype telcr ; ( led 0 blinkt)

\ read adc0 channel
: read-adc0 ( --n ) ( der ADC vom Nodemcu bekommt man im Win32forth stack)
  s" adc-read . " teltypeCR ;
' read-adc0 SetFunc: ADC0 ( ADC0 ist der Button am Bildschirm , enstand mit VForth )

: definput4 ( --n1....n100) ( 100 Messwerte lesen, aufpassen, alles erfolgt ueber Stack)
  s" : gg1 4 gpio-read ; " teltypeCR ( Port 4 wird gelesen )
  s" : gg 100 0 do gg1 . 1000 ms loop ; " teltypeCR ; ( test 100 mal lesen, je 1000 ms)

: blueledoff ( --) s" 2 GPIO_HIGH gpio-write " teltypeCR ;
\ 2 GPIO_LOW gpio-write
\ 2 GPIO_HIGH gpio-write
: blo blueledoff ;
' blo SetFunc: MOSI
\ ' MISO SetFunc: MISO
\ ' SCLK SetFunc: SCLK
' boton15 SetFunc: MISO ( alle Pins vom Nodemcu werden gleicherweise Programmiert )
' boton16 SetFunc: SCLK
```

Schritt- oder Servomotor–Steuerung

Richtungs- und Positionsignale übergeben an den Node.

```
: stepper-load ( --)
  cr ." >> loading stepper program " cr
  s" : gw gpio-write ; " teltypeCR 1000 ms
  s" : pul 2 1 gw 10 ms 2 0 gw ; " teltypeCR 1000 ms
  s" : tes2 200 0 do pul 3 ms loop ; " teltypeCR 1000 ms
  s" : tes 100 0 do pul 10 ms loop ; " teltypeCR 1000 ms ;
: tes
  cr ." *** stepping **** " cr
  s" tes " teltypeCR ; \ call the stepper prog
: tes2
  cr ." *** stepping fast**** " cr
  s" tes2 " teltypeCR ; \ call the stepper prog
' stepper-load SetFunc: GPIO3 \ Buttons werden mit Schrittmotor Kommandos geladen
' tes SetFunc: GPIO1
' tes2 SetFunc: GPIO15
\ : tes2 1000 0 do pul 3 ms loop ;

\ ------------------------------------------------------------------------------------
: motright ( --) cr ." *** right **** " cr s" 0 0 gw " teltypeCR ;
: motleft ( --) cr ." *** left **** " cr s" 0 1 gw " teltypeCR ;
: switchLED00 ( -- ) \ Anzeige am Bildschirm, eine LED zeigt Rechts-/Linkslauf vom Motor.
  LED dup 68 Bitmap! led-off21 =
    if   led-on21 motright
    else motleft led-off21
    then dup to LED 68 Bitmap! ;
 ' switchLED00 SetFunc: GPIO0
\ ------------------------------------------------------------------------------------
```

[Zum Autor: Peter Minuth lebt heute in Sao Paulo, Brasilien. In Argentinien geboren, wuchs er zweisprachig auf, Deutsch ist seine Vatersprache. Forth nutzt er seit vielen Jahren. Im Studium der Astrophysik und später in der professionellen Arbeit bei der CNC Maschinenherstellung — von einfachen Werkzeugwechslern bei Leiterplattenrobotern bis zu complettem CAD/CAM für die LuftfahrtIndustrie. Hobbys: Sternbeobachtung, Tauchen, Fliegen und Bluesmusik.]

Integer und lineare Gleichungen

Jostein Skjelstad

Viele fortgeschrittene Probleme können nur durch mehrere lineare Gleichungen mit mehreren Unbekannten gelöst werden. Die praktische Methode ist die Matrix-Berechnung. Wenn die Matrix-Operationen in einer allgemeinen Form kodiert sind, kann dieser Code für eine beliebige Anzahl von Unbekannten und Gleichungen ohne Änderungen verwendet werden, nur beschränkt durch den verfügbaren Speicherplatz.

Matrix–Berechnungen im Computer

Was braucht es dazu um so etwas zu verwirklichen?

- Die Gleichungen müssen als Tabellen verfügbar sein.
- Zweidimensionale Tabellen sind erforderlich.
- Drei verschachtelte Schleifen, die für einige der wichtigen Operationen benötigt werden.

Ein Problem erwächst daraus, dass der benötigte dynamische Bereich nicht voraussagbar ist.

Gängige Lösungsmethoden sind als *Gauß–* und *Gauß–Jordan–Algorithmus*[1] bekannt. Sie sind sehr ähnlich, beide verwenden drei verschachtelte Schleifen, um die gegebenen Arrays zu modifizieren. Wie können solche Operationen in Forth umgesetzt werden? Die *Forth Scientific Library (FSL)*[2] enthält ein Matrix–Paket dafür. Dieses Paket ist allgemein gehalten und robust, aber für Computer mit geringer Rechenleistung ungeeignet.

Das FSL–Matrix–Package

Es besteht aus acht *Toolkits*. Eines davon ist das *Matrix–Toolkit* zur Lösung von Gleichungen. Dieses Matrix–Toolkit hat 625 Zeilen Code, darunter auch sehr lange Definitionen, die Längste hat 58 Zeilen. Es arbeitet mit drei verschachtelten Schleifen, und benötigt das *locals word set* und das *floating–point word set*. Das alles entfernt sich doch vom traditionellen Forth–Stil, wie er von Charles Moore und Leo Brodie befürwortet wird, schon sehr.

Aber einen Teil der FSL–Bibliothek werde ich verwenden, die Tabellen–Wörter aus `fsl-util`.

Eine einfachere Lösung

Ich habe versucht, eine einfachere Lösung zu finden, und dabei vorausgesetzt, dass 64–Bits–Integer einen ausreichenden dynamischen Bereich für die anstehende Aufgabe hat.

Zuerst brauchen wir Forthworte, um die Tabellen zu speichern. Dazu werden die folgenden Worte aus `fsl-util` verwendet.

```
cell-    backup one cell
integer  Constant size of a regular integer
marray   monotype array defining word
}        fetch 1-D array address
mmatrix  defining word for a 2-d matrix
}}       fetch 2-D array addresses
```

Damit können wir uns ein Beispiel mit einer bekannten Lösung machen und es als Testdaten verwenden. Z.B. diese vier Gleichungen:

$8x_0 + 7x_1 + 4x_2 + 1x_3 = 45$
$4x_0 + 6x_1 + 7x_2 + 3x_3 = 30$
$6x_0 + 3x_1 + 4x_2 + 6x_3 = 40$
$4x_0 + 5x_1 + 8x_2 + 2x_3 = 30$

Ihre Lösung ist bekannt:
x_0=5 , x_1=0 , x_2=1 , x_3=1

Als Matrix und Vektor geschrieben, sieht das dann so aus, links die Matrix und rechts der Vector:
8 7 4 1		45
4 6 7 3		30
6 3 4 6		40
4 5 8 2		30

Wir definieren nun zwei Tabellen für diese Gleichungen, eine Matrix A und einen Vektor B, und dazu ein Tabelle X für die Lösungen:

```
A{{   B{   X{
```

Und Wörter mit denen die Testdaten ein- und ausgegeben werden können:

```
Data>A   Data>B   .ABX
```

Nun können wir die Tabellen mit Testdaten füllen:

```
4 #u !
45 30 40 30
Data>B
8 7 4 1
4 6 7 3
6 3 4 6
4 5 8 2
Data>A
```

Und probeweise ausgeben:

```
.ABX
```

[1] Der Gauß-Jordan-Algorithmus ist ein Algorithmus aus den mathematischen Teilgebieten der linearen Algebra und Numerik. Mit dem Verfahren lässt sich die Lösung eines linearen Gleichungssystems berechnen. Es ist eine Erweiterung des gaußschen Eliminationsverfahrens. Quelle: https://de.wikipedia.org/wiki/Gau%C3%9F-Jordan-Algorithmus

[2] Die *Forth Scientific Library* wurde von Skip Carter und freiwillig Mitwirkenden und Rezensenten von Code erstellt. Es bietet eine Sammlung von Routinen, die für diejenigen nützlich sein können, die wissenschaftliche oder numerisch intensive Arbeit mit Forth machen wollen.

Jetzt sind wir bereit für den ersten Schritt der Lösung. Wir werden die allgemeinen Regeln für die Arbeit mit Matrizen verwenden, um die linke Tabellen in eine dreieckige Form[3] zu konvertieren. Zunächst die notwendigen Definitionen:

```
Multiplier   ResizeElement   MakeTriangular1
```

Damit kann man schon konvertieren und sich das Zwischenergebnis ansehen:

```
MakeTriangular1 .ABX
```

Nun können die Gleichungen, beginnend mit der letzten Zeile, gelöst werden. Das nennt man eine *Rücksubstitution*. Wir brauchen dazu:

```
0X   backsub
```

Das X-Array enthält danach schon die Lösung:

```
.ABX
```

Hier noch eine Definition, welche diese Schritte zusammenfasst:

```
: solve ( n -- )
  #u ! MakeTriangular 0X backsub ;
```

Benutze sie so:

```
4 solve .abx cr
```

Beachtet bitte, dass bei diesem Verfahren während der Berechnung die Testdaten vom Daten–Stack jedesmal verbraucht, und die Tabellen modifiziert werden. Für einen erneuten Testdurchlauf müssen die orginalen Daten daher auch erneut auf den Stack geschrieben werden.

Änderungen

`MakeTriangular1` verwendet `I, J` und `K` als Loop–Indizes. Das funktioniert in *gforth*, aber es ist nicht pures ANS–Forth, in dem nur `I` und `J` als Loop–Indizes verlangt werden. Um `MakeTriangular` für kleineres Standard–Forth akzeptabel zu machen, muss der dritte Index mit einer gewöhnlichen VARIABLE oder als VALUE simuliert werden. Im Listing zeige ich ab Zeile 137 (ANS) wie das geht, und definiere dort `MakeTriangular2`. Ansehen des Ergebnisses auch dabei wie schon gewohnt:

```
MakeTriangular2 .ABX   0X backsub .ABX
```

Es ist sogar möglich, VALUE für alle drei Indizes zu verwenden. Dann kann der gesamte Triangulations–Prozess in kleinere Wörter aufgeteilt werden — ab Zeile 157, „indices for 3 nested loops“ — steht, wie es geht, um zum `MakeTriangular3` zu gelangen. Auch das lässt sich wie gewohnt testen:

```
MakeTriangular3 .ABX   0X backsub .ABX
```

Alle drei Versionen von `MakeTriangular` ergeben das gleiche Ergebnis, mit kleinen Unterschieden nur in der Geschwindigkeit.

Optimierung

Es ist möglich, einige Multiplikationen in `MakeTriangular` zu eliminieren, weil wir wissen, dass die unteren linken Elemente null sein sollten. Diese Änderung wird hier aber nicht mehr verfolgt.

Erfahrung

Ich habe ein 64–Bit–gforth mit diesen Forth–Worten verwendet, um ein geodätisches Netzwerk mit 27 unbekannten Entfernungen zwischen 50m und 20km mit Millimeter als Einheit zu lösen, ohne dass dabei Probleme mit der Präzision oder Überlauf aufgetreten wäre. Bei sorgfältiger Skalierung führte auch das 32–Bit–gforth dieses Beispiel gut aus.

Dennoch soll darauf hingewiesen werden, dass dieses Programm abstürzen wird, wenn Sie es mit 'schlechten' Daten füttern. Für Daten mit unbekannter Qualität sollte ein Algorithmus verwendet werden, der *pivoting*[4] verwendet, um sehr kleine Werte, oder gar Null–Nenner, zu vermeiden, so, wie es das FSL–Paket auch macht.

Die Auflistung enthält einige Worte[5], die nicht im *core word set* des ANS–Forth enthalten sind. Doch diese können leicht durch *core words* ersetzt, oder mit deren Hilfe definiert werden.

Links

FSL : https://www.taygeta.com/fsl/sciforth.html

Forth Standard : http://forth-standard.org/standard/words

[3] Die Dreiecksform ist ein Sonderfall der Stufenform, bei der jede Zeile genau eine Unbekannte weniger als die vorhergehende hat. Quelle: https://de.wikipedia.org/wiki/Lineares_Gleichungssystem#Reduzierte_Stufenform

[4] Gauss Jordan Elimination Through Pivoting. Quelle z.B.: https://people.richland.edu/james/lecture/m116/matrices/pivot.html

[5] `\ .R VALUE TO 2VARIABLE 2! 2@`

Listings

```
\ Part of fsl-util

\ backup one cell
: cell- [ 1 CELLS ] LITERAL - ;

\ size of a regular integer
1 CELLS CONSTANT integer

\ 1-D array definition
\ ----------------------------
\ | cell_size | data area |
\ ----------------------------

\ monotype array
: marray ( n cell_size -- | -- addr )
   CREATE
   DUP , * ALLOT
   DOES> CELL+
;

\ word that fetches 1-D array addresses
: } ( addr n -- addr[n])
   OVER cell- @
   * SWAP + ALIGNED
;

\ 2-D array definition,
\ Monotype
\ ----------------------------
\ | m | cell_size | data area |
\ ----------------------------

\ defining word for a 2-d matrix
: mmatrix ( n m size -- )
   CREATE
   OVER , DUP ,
   * * ALLOT
   DOES> [ 2 CELLS ] LITERAL +
;

\ word to fetch 2-D array addresses
: }} ( addr i j -- addr[i][j] )
   2>R   \ indices to return stack temporarily
   DUP cell- cell- 2@        \ &a[0][0] size m
   R> * R> + *
   +
   ALIGNED
;
\ end of fsl-util code


\ fixed arrays used here
\ arrays with dynamic dimensions could be used
\ #u is number of unknowns
\ this must be the same as the number of
\ equations and the number of elements in one
\ equation left side

VARIABLE #u

\ allocated size, must be equal or larger than #u
20 CONSTANT arraysize

\ equations left side
arraysize DUP integer mmatrix A{{

\ equations right side
arraysize integer marray  B{

\ solution array
arraysize integer marray  X{

\ filling the arrays with numbers from the stack:
: Data>A ( n1...nn-- )
   0 #u @ 1- DO
      0 #u  @ 1- DO
         A{{ J I }} !
      -1 +LOOP
   -1 +LOOP ;

: Data>B ( n1...nn-- )
   0 #u @ 1- DO  B{ I } !  -1 +LOOP ;

\ displaying all three arrays:
: .ABX ( -- ) CR ." Arrays, A and B and X:"
   #u @  0 DO
      CR ." |"
      #u @  0 DO
         A{{ J I }} @ 8 .R   \ left side
      LOOP
      \ right side
      ."  |   |" B{ I } @ 8 .R   ."  |"
      X{ I } @ 8 .R     \ solution vector
   LOOP ;



\ multiplier stored as numerator denominator
2VARIABLE Multiplier

\ multiply element by multiplier
: ResizeElement  ( n1 -- n2 )
   Multiplier 2@ ( n numerator denominator )
   */  ;

: MakeTriangular1    ( -- )
   #u @ 1 - 0 DO       \ upper row
      #u @ I 1 + DO    \ lower row
         A{{ I J }} @ A{{ J J }}  @
         Multiplier 2!
         B{ I } @ B{ J } @  ResizeElement  -
         B{ I } !      \ convert right side
         #u @ 0  DO     \ convert left side
            A{{ J I }} @   A{{ K I }} @
            ResizeElement   - A{{ J I }} !
         LOOP
      LOOP
   LOOP  ;



\ zero fill solution array
: 0X   ( -- ) #u @ 0 DO   0 X{ I } !   LOOP ;

: backsub   ( -- )  \ back substitution
   0 #u @ 1 - DO
      #u @ I  DO
         B{ J } @ X{ I } @
         A{{ J I }} @ * - B{ J } !
      LOOP
      B{ I } @ A{{ I I }} @ / X{ I } !
   -1 +LOOP ;

: solve   ( n -- ) #u !   MakeTriangular1   0X
   backsub ;



\ ANS

0 VALUE k

: MakeTriangular2    ( -- )
   #u @ 1 - 0 DO  I TO k            \ upper row
      #u @ i 1 + DO                 \ lower row
         A{{ I J }} @ A{{ J J }} @
         Multiplier 2!
         B{ I } @ B{ J } @  ResizeElement  -
         B{ I } !        \ convert right side
         #u @ 0  DO      \ convert left side
```

```
151              A{{ J I }} @   A{{ k I }} @
152              ResizeElement - A{{ J I }} !
153           LOOP
154        LOOP
155     LOOP  ;
156
157
158
159   \ indices for 3 nested loops
160   0 VALUE l   0 VALUE m   0 VALUE n
161
162   : Multiplier! ( -- )   \ stored as a fraction
163     A{{ m l }} @   A{{ l l }} @   multiplier 2! ;
164
165   : ConvertRight   ( -- )
166     B{ m } @   B{ l } @  ResizeElement  - B{ m } ! ;
167
168   : ConvertElement   ( -- )
169     A{{ m n }} @   A{{ l n }} @   ResizeElement
170     -   A{{ m n }} ! ;
171
172   : ConvertLeft   ( -- )
173     #u @ 0   DO   I TO n
174        ConvertElement
175     LOOP ;
176
177   : MakeTriangular3 ( -- )
178       \ outer LOOP, selecting upper row
179       #u @ 1- 0 DO   I TO l
180           \ mid LOOP, selecting lower row
181           #u @ I 1+ DO  I TO m
182             Multiplier! \ find and store multiplier
183             ConvertRight \ single element, no LOOP
184             ConvertLeft  \ inner LOOP
185           LOOP
186       LOOP  ;
187
```

```
188   cr .( finis ) cr cr
189
190
```

```
1    \ testdata
2
3    4 #u !
4
5    : testdata
6    45 30 40 30
7    Data>B
8    8 7 4 1
9    4 6 7 3
10   6 3 4 6
11   4 5 8 2
12   Data>A
13   ;
14
15   .( Verfahren 1:)
16   testdata .abx
17   MakeTriangular1 .abx
18   0X backsub .abx cr
19
20   .( oder zusammengefasst:)
21   testdata   4 solve .abx cr
22
23   .( Verfahren 2:)
24   testdata
25   MakeTriangular2 .ABX
26   0X backsub .ABX cr
27
28   .( Verfahren 3:)
29   testdata
30   MakeTriangular3 .ABX
31   0X backsub .ABX cr
32
33   cr .( finis) cr cr
34
```

(Fortsetzung der Meldungen)

Forth für die Cypress PSoC[6]s: Mecrisp–Stellaris

Die erste Phase der Portierung ist abgeschlossen, die Evaluation–Kits sind, bis auf die beiden PSoC6, schon an den Mikrokontrollerverleih der Forth–Gesellschaft gesendet und können dort bezogen werden. Die Images sind auf *sourcforge* hinterlegt, der unten stehende Link führt zu den bisher portierten 32–bit Arm Cortex M0 PSoC 4 (cy8c4245, cy8c4247 und cy8c4248) sowie zum PSoC 5 (cy8c5868 und cy8c5888).

Außerdem läuft Mecrisp—Stellaris schon auf einem weiteren PSoC5LP (cy8c5888axi), der auf dem FreeSoC2–Board von *sparkfun* zu finden ist. Das Board kam dankenswerter Weise auf der Maker Faire in Hannover von einem Mitarbeiter von *Arrow*.

Die beiden PSoC 6 arbeiten, wenn man WiFi nutzen möchte, mit WICED, einer Eclips Umgebung. Für diese Reihe wurde noch kein Ansatz für die Portierung gefunden, das ist etwas für die langen Abende im Winter.

Cypress betreibt übrigens eine eigene Community–Plattform auf der die Portierungen bekannt gemacht worden sind. Unter PSoC 4 oder PSoC 5 findet ihr sie durch runterscrollen tief in den Stack der Beiträge, oder durch Suche nach „Forth“. Falls ihr Fragen allgemeiner oder technischer Art zu den einzelnen Portierungen habt, könnt ihr sie gerne über die community stellen, mit dem kleinen Nebeneffekt, dass auf den Eintrag ein `PICK` ausgeführt wird und der auf dem `TOS` landet, und damit wieder in Erscheinung tritt. ;-) Klaus Zobawa

Links

https://sourceforge.net/projects/mecrisp/files/Cypress/

https://community.cypress.com/community/MCU

[6] PSoC (programmierbares System–on–Chip) ist eine Familie von integrierten Schaltungen von Cypress Semiconductor. Diese Chips integrieren einen CPU–Kern und Mixed–Signal–Arrays an konfigurierbaren analogen und digitalen Peripheriegeräten.

Goto

Wil Baden

Forth control flow is complete. Everything that **GOTO** *and* **LABEL** *can do, Forth can replicate. Using* **CS-ROLL** *, Forth can implement gotos and labels. On occasions when unstructured control flow is desirable, gotos and labels are clearer than explicitly shuffling the control-flow stack with* **CS-ROLL** *.*

DONALD E. KNUTH, *"Structured Programming with* **goto** *Statements" (1974), reprinted in* ***Literate Programming*** *(1992), discusses situations where gotos are appropriate. Elsewhere*[1] *I will convert and compare his examples in structured Forth and Forth with gotos.*[2]

```
ANEW --GOTO--  DECIMAL         \  Wil Baden  2000-06-10

\  GOTO in Forth

\          LABEL <name>    GOTO <name>

\  In the following implementation, labels are
\  case sensitive and must be recognized by the
\  first ‘2 CELLS‘ characters.
\  Only one ‘GOTO‘ can go to a previous label.
\  This is to keep the programming simple - the ‘AGAIN‘
\  compiled by ‘GOTO‘ to an already defined label
\  consumes the label. Previous labels may be defined
\  more than once to handle more than one backward
\  goto. However, many ‘GOTO‘s can be made to each one
\  of future labels.

\  Because control-flow elements are removed when
\  they are resolved, labels may be redefined.
\  Thus all loops may begin and end with the same
\  labels, such as ‘START‘ and ‘END‘.
\  Or you may use distinct labels.

\  Programming note.
\  Many Forth systems use the data stack for
\  control flow.  Therefore the data stack must be
\  cleared before compiling control-flow words.

\  LABEL   ( "name" -- )
\          ( C: -- dest  OR  orig_1 ... orig_n -- )
\     A destination.
\     If _name_ has no gotos to it, ‘LABEL_name_‘
\     becomes a ‘BEGIN‘, otherwise enough ‘THEN‘s are
\     used to resolve the gotos.
\     As labels are resolved they are removed
\     (from ‘Label-Table‘).

\  GOTO   ( "name" -- )( C: -- orig  OR  dest -- )
\     The origin of an unconditional branch.
\     If _name_ has no ‘LABEL‘, ‘GOTO _name_‘ becomes
\     ‘FALSE IF‘ (or ‘AHEAD‘), otherwise the last
\     ‘LABEL _name_‘ is resolved with ‘AGAIN‘
\     and removed (from ‘Label-Table‘).

\          Needed from Tool Belt

\  Uses  THIRD   3DUP   3DROP   NOT  from ToolBelt.
\  Comment out any words that are already defined.
TRUE 0<> [IF]
  : THIRD  ( x y z -- x y z x )  2 PICK ;
  : 3dup   ( x y z -- x y z x y z ) THIRD THIRD THIRD ;
  : 3DROP  ( x y z -- )  2DROP DROP ;
  : NOT    ( x -- flag )  0= ;
[THEN]

    TRUE 1 RSHIFT INVERT  CONSTANT  Sign-Bit

\  Label-Table         ( -- addr )
\     Extension of the control-flow stack.
\     The contents are double numbers.
\     Label for ‘GOTO‘, "smudged" label for ‘LABEL‘,
\     ‘0.‘ for control-flow words.

\  CS-Count            ( -- addr )
\     Counter for the depth of ‘Label-Table‘.

\  Pickup-Label-for-Lookup  ( "label" -- label . )
\     Get next word from source input
\     and store its first two cells
\     in the top of ‘Label-Table‘.

\     Used in ‘Lookup-Comefrom‘ and ‘Lookup-Goto‘.

\  Lookup-Label     ( label . index -- label . index’ )
\     Look up a label beginning at _index_-1.
\     If label isn’t found, returns -1 as _index’_,
\     otherwise the index where it was found.

\  Lookup-Comefrom  ( "label" -- index )
\     Get next word from source input and look for it
\     as a previous ‘GOTO‘.

\     Used in ‘LABEL‘.

\  Lookup-Goto         ( "label" -- index )
\     Get next word from source input and look for it
\     as a previous ‘LABEL‘.

\     Used in ‘GOTO‘.

\  Resolve-Label       ( index n -- )
\     The equivalent of ‘n CS-ROLL‘ for ‘Label-Table‘.

\  Resolve-the-ComeFroms  ( index -- )
\     Do ‘THEN‘ for each of the previous ‘GOTO‘s.

\     Used in ‘LABEL‘.

100 CONSTANT Max#Labels  \  Undocumented restriction.
CREATE Label-Table  Max#Labels 2* CELLS ALLOT

VARIABLE CS-Count   \ Counter for unresolved control flow.

: Pickup-Label-for-Lookup    ( "label" -- label . )
    BL WORD COUNT  2 CELLS MIN          ( str len)
    CS-Count @ 2* CELLS Label-Table +   ( str len addr)
    dup >R
        0. R@ 2!  SWAP MOVE
```

[1] Wil Baden, 1928 – 2016. Ob er sein Vorhaben noch verwirklicht hat, habe ich nicht herausgefunden.

[2] Um die Art und Weise, wie er Programme kommentiert hat, zu zeigen, wurde nur das Abstract aus dem Listing ausgegliedert, weil es so schön in unser Artikel–Layout passte. Die übrigen Textpassagen wurden an Ort und Stelle im Listing gelassen. Lediglich einige zu lange Kommentar–Zeilen mussten für den Spaltensatz sinngemäß umgebrochen werden. Sein Code musste nicht umgebrochen werden, er pflegte kurze übersichtiche Zeilen in Definitionen.

```
    R> 2@ ( label .) ;

: Lookup-Label    ( label . index -- label . index' )
    BEGIN
        1-  dup 0< NOT WHILE
        3dup 2* CELLS Label-Table + 2@ D=
    UNTIL THEN ;

: Lookup-Comefrom    ( "label" -- index )
    Pickup-Label-for-Lookup      ( label .)
    CS-Count @ Lookup-Label      ( label . index)
    NIP NIP ( index) ;

: Lookup-Goto        ( "label" -- index )
    Pickup-Label-for-Lookup      ( label .)
    Sign-Bit OR
    CS-Count @ Lookup-Label      ( label . index)
    NIP NIP ( index) ;

: Resolve-Label       ( index n -- )
    over - >R                    ( index)
        2* CELLS Label-Table +   ( addr)
        dup 2 CELLS + SWAP       ( addr' addr)
    R> 2* CELLS MOVE ( ) ;

: Resolve-the-Comefroms   ( index -- )
    dup 2* CELLS Label-Table + 2@ ROT  ( label . index)
    BEGIN
        dup 2SWAP 2>R >R    ( index)( R: label . index)
            CS-Count -1 over +! @   ( index cnt)
            2dup 2>R
                SWAP - CS-ROLL postpone THEN
            2R> Resolve-Label  ( )
        R> 2R> ROT             ( label . index)( R: )
        Lookup-Label
    dup 0< UNTIL
    3DROP ( ) ;

: LABEL ( "_label_" -- )
        ( C: -- dest  OR  orig_1 ... orig_n -- )
    Lookup-Comefrom       ( index)
    dup 0< IF DROP        ( )          \  BEGIN
        postpone BEGIN
        CS-Count @ 2* CELLS Label-Table +
            dup >R  @ Sign-Bit OR  R> !
        1 CS-Count +!
    ELSE                  ( index)     \  THEN
        Resolve-the-Comefroms
    THEN ; IMMEDIATE

: GOTO ( "_label_" -- )( C: -- orig  OR  dest -- )
    Lookup-Goto           ( index)
    dup 0< IF DROP        ( )          \  AHEAD
        \ POSTPONE AHEAD
        postpone FALSE  postpone IF
        1 CS-Count +!
    ELSE                  ( index)     \  AGAIN
        CS-Count -1 over +! @  ( index cnt)
        2dup 2>R
            SWAP - CS-ROLL postpone AGAIN
        2R> Resolve-Label  ( )
    THEN ; IMMEDIATE

\                                2000-05-31 Wil Baden

\  IF  WHILE  ELSE  THEN  BEGIN  AGAIN  UNTIL  REPEAT :

\  The standard control-flow words must be redefined
\  so they can be mingled with the label words.

\  'IF' and 'BEGIN' are extended to put an empty label
\  on top of 'Label-Table'.

\  The other control-flow words search 'Label-Table'
\  for the empty label of the last control-flow word.


\  'CS-ROLL' brings the control-flow word to the top of
\  the control-flow stack, and the normal control-flow
\  word is compiled.
\  'Label-Table' is updated equivalently.

\  ':' is extended to initialize Label-Table.

\  As defined in standard Forth, DO-loops can not be
\  mingled with control-flow words.  Use 'LEAVE' to
\  break out of a DO-loop, or rewrite DO-loops
\  as 'BEGIN ... UNTIL'.

\  Mark-Control-Flow      ( -- )
\     Mark control flow for 'IF' and 'BEGIN'
\     in 'Label-Table'.

\  Resolve-Control-Flow   ( -- index )
\     Resolve control flow in 'Label-Table'.

: Mark-Control-Flow      ( -- )
    0.  CS-Count @ 2* CELLS Label-Table +  2!
    1 CS-Count +! ;

: Resolve-Control-Flow   ( -- index )
    0.  CS-Count @ Lookup-Label NIP NIP ( index)
    dup 0< ABORT" Missing Control Flow " ;

: IF   ( C: -- orig )
    postpone IF  Mark-Control-Flow ; IMMEDIATE

: WHILE   ( C: dest -- orig dest )
    Resolve-Control-Flow  ( index)
    CS-Count @ SWAP - 1- CS-ROLL  ( )
    postpone IF  1 CS-ROLL            \  Uses new IF.
    ; IMMEDIATE

: ELSE   ( C: orig_1 -- orig_2 )
    Resolve-Control-Flow  ( index)
    CS-Count @ SWAP - 1- CS-ROLL  ( )
    postpone ELSE
    ; IMMEDIATE

: THEN   ( C: orig -- )
    Resolve-Control-Flow ( index)
    CS-Count -1 over +! @   ( index cnt)
    2dup 2>R
        SWAP - CS-ROLL postpone THEN
    2R> Resolve-Label                    ( )
    ; IMMEDIATE

: BEGIN   ( C: -- dest )
    postpone BEGIN  Mark-Control-Flow ; IMMEDIATE

: AGAIN   ( C: dest -- )
    Resolve-Control-Flow ( index)
    CS-Count -1 over +! @   ( index cnt)
    2dup 2>R
        SWAP - CS-ROLL postpone AGAIN
    2R> Resolve-Label                    ( )
    ; IMMEDIATE

: UNTIL   ( C: dest -- )
    Resolve-Control-Flow ( index)
    CS-Count -1 over +! @   ( index cnt)
    2dup 2>R
        SWAP - CS-ROLL postpone UNTIL
    2R> Resolve-Label                    ( )
    ; IMMEDIATE

: REPEAT   ( C: orig dest -- )
    \  Uses new AGAIN and THEN.
    postpone AGAIN  postpone THEN ; IMMEDIATE

: : : 0 CS-Count ! ;
```

```
261
262  \      * * * * * * *
263  \      * Examples  *
264  \      * * * * * * *
265
266  MARKER Test-and-Forget
267
268  \  Labels only.
269
270  : GCD1  ( m n -- gcd )
271      LABEL START
272          dup 0= IF GOTO END THEN
273          TUCK ( n  m n) MOD ( m n)
274      GOTO START
275      LABEL END  DROP ;
276
277  20451 24140 GCD1 CR .  \  17
278
279  \  Mingled label and control flow.
280
281  : GCD2  ( m n -- gcd )
282      BEGIN
283          dup 0= IF GOTO END THEN
284          TUCK ( n  m n) MOD ( m n)
285      AGAIN
286      LABEL END  DROP ;
287
288  20451 24140 GCD2 CR .  \  17
289
290  \  Multiple Labels
291
292  : Multi-Go  ( -- )
293      4 -2 DO  CR  ." \ "
294          I 1 AND IF GOTO NEXT THEN
295          ." even "
296          I IF GOTO NEXT THEN
297          ." zero "
298      LABEL NEXT
299          I 0< IF GOTO NEXT THEN
300          ." non-negative "
301      LABEL NEXT
302          I .
303      LOOP ;
304
305  Multi-Go
306
307  \ even -2
308  \ -1
309  \ even zero non-negative 0
310  \ non-negative 1
311  \ even non-negative 2
312  \ non-negative 3
313
314  ( END ) Test-and-Forget
315
316  \\  End of GOTO in Forth
317
```

Quellen

http://www.wilbaden.com/neil_bawd/index.html

http://www.boston-baden.com/hazel/dad/

http://www.literateprogramming.com/knuthweb.pdf

(Fortsetzung der Meldungen.)

Minitipp: Bitgruppen

Neulich vermisste Wolfgang S. eine einfache Möglichkeit, beim Schreiben in Register mittels `!` (store), im Quellcode dafür die einzelnen Bitgruppen deutlich erkennbar zu machen. Neben einem Überdefinieren von `?number` oder dem Einsatz von Recognizern, kam mir diese Idee:

```
%000.111.11.10.01.00 drop REGISTER !
```

Hier wird ausgenutzt, dass in einigen (oder gar allen?) Forthdialekten doppeltgenaue Zahlen durch eine beliebige Anzahl von Punkten (vulgo: dots) eingegeben werden können. Ich selbst habe das sofort in meine Programmierweise übernommen.

Wen das `drop` stört, der kann ja etwas „syntactic sugar“ dazu nehmen:

```
: reg! ( dl dh adr -- )
  swap drop ! ;
  \ dl=bitmask dh=doublenumber high byte
  \ adr=register address
```

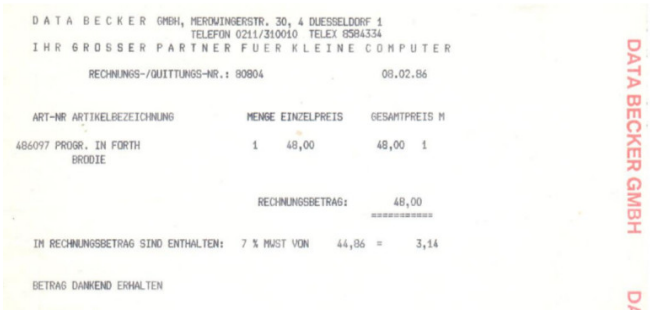
Obiges Beispiel sähe dann so aus:

```
%000.111.11.10.01.00 REGISTER reg!
```

Das Ganze geht auch im Hexadezimalmodus, solange die Bitgruppen sich schön an Nibbelgrenzen halten. Martin

Programmieren können ist mehr als eine Programmiersprache kennen

Manchmal räume ich ja auf. Und was flattert mir da entgegen aus dem guten alten Brodie? Die Quittung vom Kauf des Buches „Programmieren in Forth“ bei Data Becker: 48 DM. Das war ein stolzer Preis damals. Ja, es gab sie, die deutsche Ausgabe von Leo Brodies „Starting Forth“. Bernd Steinbach hatte das im Hansa-Verlag im Jahre 1984 besorgt. „Forth, vom Einstieg bis zum Standard.“ Der Titel hielt, was er versprach, man kannte dann Forth. Programmieren ist jedoch etwas anderes als die bloße Kenntnis einer Sprache. Die Kenntnis der Maschine, mit der man es zu tun h,für denat, und bekannte Prozeduren darauf, Konzepte, Vorstellungen davon, was da innen drin passiert, müssen dazu kommen. Dabei hilft Forth mehr als manch andere Sprache. Es ist maschinennah geblieben. mk



DATA BECKER GMBH, MEROWINGERSTR. 30, 4 DUESSELDORF 1
TELEFON 0211/310010 TELEX 8584334
IHR GROSSER PARTNER FUER KLEINE COMPUTER

RECHNUNGS-/QUITTUNGS-NR.: 80804 08.02.86

ART-NR	ARTIKELBEZEICHNUNG	MENGE	EINZELPREIS	GESAMTPREIS	M
486097	PROGR. IN FORTH BRODIE	1	48,00	48,00	1

RECHNUNGSBETRAG: 48,00

IM RECHNUNGSBETRAG SIND ENTHALTEN: 7 % MWST VON 44,86 = 3,14

BETRAG DANKEND ERHALTEN

DATA BECKER GMBH

Dezimale Prüfziffer

Rafael Deliano

Das Eintippen von Zahlen auf Tastatur ist fehlerträchtig. Eine Prüfsumme, die Fehler erkennt, ist deshalb wünschenswert. Die leistungsfähigen Verfahren, die bei Binärdaten üblich sind, z.B. CRC, passen allerdings nicht. In diesem Beitrag werden Verfahren vorgestellt, die auf Microcontrolern einsetzbar sind. Und im folgenden Beitrag über die PS2-Tastatur wird eine Messstand–Anwendung beschrieben, bei der das Prüfziffer–Verfahren erfolgreich eingesetzt worden ist.

Labels

Hier sollte eine 5–stellige fortlaufende Seriennummer durch eine Prüfziffer geschützt werden (Abb.1). Die meisten Drucker und Laser–Beschrifter haben Software zur Erzeugung von fortlaufenden Nummern eingebaut. Sie würden auch Barcodes mit Prüfziffern erzeugen. Prüfziffern für Seriennummern werden allerdings kaum unterstützt. Das Lesen von Ziffernfolgen aus Files ist als Alternative jedoch möglich. Unkomfortabel, aber flexibel. Man kann dann eine beliebige Prüfziffer verwenden.

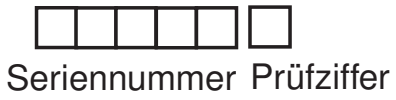


Abbildung 1: Datenformat

Verfahren

Verbreitung hat immer noch Luhn, wie im Patent von 1954 [1] beschrieben. Es wäre dort mechanisch implementiert worden (Abb.7). Luhn wanderte 1924 von Deutschland in die USA aus. Dort nahm er nach dem Krieg eine führende Position in der Forschung von IBM ein [2]. Die IBM–Prüfziffer ist simpel zu implementieren und war verfügbar als die ersten Anwendungen mit Computern entstanden.

In einer ihrer vielen alternativen Bezeichnungen, wie MOD10/EAN, deutet das EAN die zweite Gruppe gängiger Prüfziffern an: Barcodes. Sie haben eine andere Fehlerstatistik als die Eingabe von Hand. Aber durch den Übergang auf die automatisierte Datenerfassung in den 70er Jahren stiegen Verbreitung und Anforderungen an die Fehlersicherung, durch Mikroprozessoren aber auch die Möglichkeiten.

Echter Fortschritt war nicht festzustellen, typisch für numerische Daten sind eine Vielzahl MOD11–Verfahren. Der Divisor ist dann 11 statt 10, weil Primzahl besser ist. Wurde schon in [3] vorgeschlagen. Erzeugt aber Rest 0 ... 10 statt 0 ... 9 wie bei MOD10. Wenn man nicht eine Ziffer 10 typisch "X" genannt darstellen kann, hat man ein Problem. Viele Anwendungen überspringen dann die Zahlen, die das Problem haben, einfach. Das ist keine für den Anwender befriedigende Lösung.

Die dritte Gruppe sind Verfahren, die so aufwendig oder undurchsichtig sind, dass sie ehedem nicht attraktiv waren und deshalb keine praktische Anwendung gefunden haben. Das bekannteste wurde 1969 von dem holländischen Mathematiker Jacobus Verhoeff veröffentlicht (Abb.2). Die Fehlerstatistiken von Beckley waren ihm schon bekannt. Er hat zusätzlich Daten des *Amsterdam Municipal Clearing Office* und des *Netherlands Postal Check and Giro Service* ausgewertet [4] (Tab.1)[1]. Der Fortschritt gegenüber Luhn war damit, dass nun echte Fehlermodelle verfügbar waren.

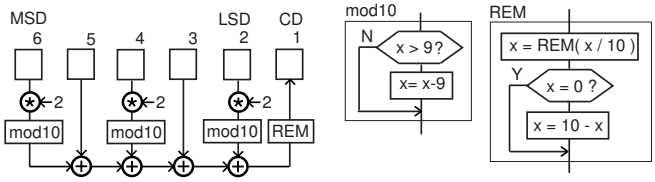


Abbildung 2: Blockschaltbild Luhn

Eine praktische Anwendung von Verhoeff war eine davon abgeleitete Prüfsumme der 11stelligen Seriennummer der neuen deutschen DM–Banknoten. Viel Verbreitung hat das Verfahren dennoch nicht gefunden.

Weitere Verbesserungen sind möglich, die Damm–Prüfsumme enstand erst 2004 [6]. Die Wahrscheinlichkeit der praktischen Anwendung ist aber gering.

Implementierung

Bei Luhn wird jede Ziffer an gerader Position mit 2 multipliziert (Abb.2,3). Wenn sie dann größer als 9 sein sollte, wird 9 subtrahiert. Die Summe aller Ziffern wird durch 10 dividiert und der Rest davon von 10 abgezogen.

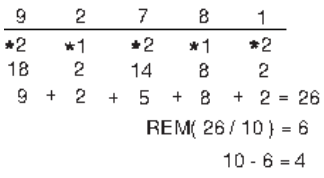


Abbildung 3: Beispiel Luhn

Für MOD11 existiert eine Vielzahl von Varianten. Unübersehbar wurde dabei Luhn überarbeitet und vereinfacht.

[1] Beckley hat 8% Transposition, aber Verhoeff hat die nochmal aufgeteilt. Beckey hat 6% „sonstige“ die bei Verhoeff auch nochmal aufgeteilt sind. Es waren zwei unterschiedliche Untersuchungen, die Rubriken stimmen also nicht passgenau.

		Verhoeff	Beckley
single error	a -> b	60...95%	86%
adjacent transposition	ab -> ba	10...20%	8%
jump transposition	acb -> bca	0,5...1,5%	
twin error	aa -> bb	0,5...1,5%	6%
phonetic error		0,5...1,5%	
jump twin error	aca -> bcb	<1%	
other error		10...20%	

Tabelle 1: Fehlertyp und Häufigkeit

Die Gewichtung wird meist aus einer Tabelle entnommen, die lokale Rückskalierung entfällt. Die Variante der Postbehörden (Abb.5) ist attraktiv, denn sie patcht die überzählige Ziffer. Da hier aber nur 5 Stellen benötigt werden, wurde als Variante-a etwas willkürlich der Anfang der Tabelle *8 ... *3 verwendet.

Demgegenüber hat die gängigere Version MOD11 eine regulärere Gewichtung (Abb.6). Hier als Variante–b getestet. Bei ihr ist normalerweise x=11 als illegal definiert. Das war hier nicht möglich, der Patch von Variante–a wurde deshalb beibehalten.

Verhoeff unterscheidet sich deutlich von Luhn, hat aber den Ruf komplex zu sein wohl zu Unrecht. Die Berechnung beruht auf drei Tabellenzugriffen (Abb.4). Arrays mit 100 Byte sind heute kein Problem, aber 1969 sah die Welt noch anders aus.

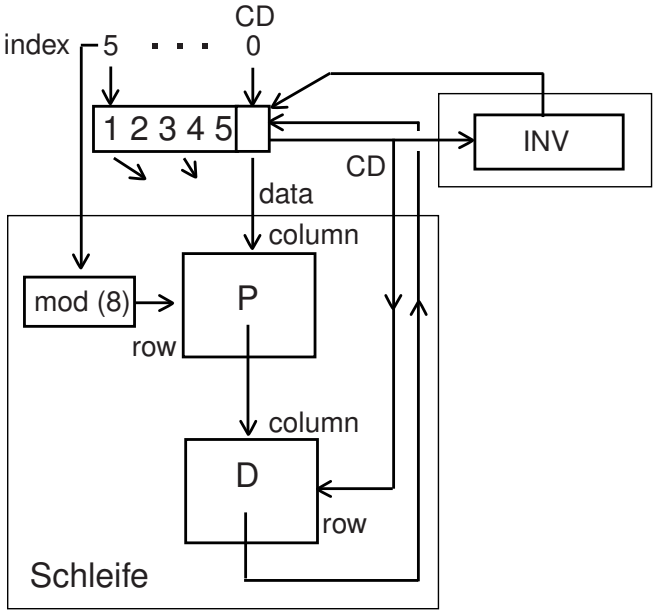


Abbildung 4: Blockschaltbild Verhoeff

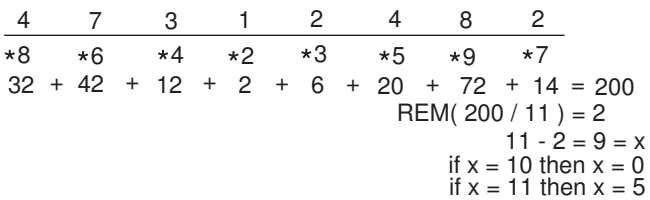


Abbildung 5: MOD11a Postbehörden

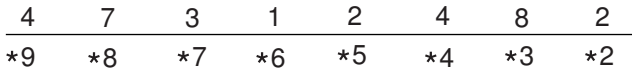


Abbildung 6: MOD11b Standard

Die Prüfziffer *CD* ist Teil des Datenfelds und muss mit 0 initialisiert werden. In der Hauptschleife mit Index i=0...5 erfolgt erst der Zugriff in das Array P mit Datenziffer und dem Index.

Da Datenstrings länger als 7 Stellen sein können, muss der Index mit *mod(8)* reduziert werden. Das Ergebnis wird zusammen mit dem alten Wert von *CD* für den folgenden Zugriff auf das Array D verwendet. Dessen Ergebnis ist ein neuer Wert für *CD* der rückgespeichert wird. Nachdem die Schleife beendet ist, folgt ein Zugriff auf die Tabelle `INV` die den endgültigen Wert der Prüfziffer bestimmt.

Simulation

Erstens sind die Angaben in der Literatur, wie wirksam die Verfahren sind, wenig anschaulich. Zweitens kann man so auch tiefer als mit einigen Testzahlen prüfen, ob man richtig implementiert hat. Als Nutzdaten wird die Seriennummer von 00001 ... 99999 hochgezählt. Für die Störungen wird ein gleichverteiler 16–Bit–LCG–Zufallszahlengerator verwendet. Da mehrere Generatoren benötigt werden, erhalten diese unterschiedliche Startwerte.

Im ersten Test wird durch Zufallszahl eine Position 0 ... 4 in der Seriennummer bestimmt, die verändert werden soll. Und dafür eine zufällige neue Ziffer 0 ... 9 bestimmt. Es kann natürlich sein, dass die Neue der alten Ziffer entspricht. Deshalb wird das wiederholt, bis die Ziffer wirklich gestört ist.

Test 2 ist der klassische Dreher, bei dem zwei benachbarte Ziffern vertauscht werden. Hier wird nur ein Versuch der Störung unternommen, denn eine Zahl wie 11111 ist durch Dreher nicht zu verändern.

Verhoeff liefert gegen simple Fehler ein perfektes Ergebnis (Tab.2).

Luhn hat Problem mit Drehern. Die gepatchten MOD11 sind zwar etwas einfacher als Luhn, aber sehr schlecht.

	Luhn	Verhoeff	MOD11a	MOD11b
TEST1	0	0	1858	1814
TEST2	1657	0	1657	1670

Tabelle 2: Unerkannte Fehler

Zum Listing

Der Forth–Quellcode ist in *nanoFORTH* für den MC68HC908GP32 Controller verfasst. Der GP32 ist nicht mehr im Handel, beim Autor jedoch im Bestand und wird für den Bau von Messständen eingesetzt.

Im Listing findes sich die ungewöhnlichen Klammer-Worte <| und |> . Die sind im Handbuch beschrieben. Das Handbuch ist beim Autor als PDF erhältlich. Die Klammern schalten vom Kommandozeilenmodus auf einen Modus um, in dem Textfiles compiliert werden. So kann man mit gängigen Terminalprogrammen wie Teraterm effizient den Quellcode ins nanoFORTH laden. Will jemand den Quellcode portieren, können diese Klammern einfach weggelassen werden.

Literatur

[1] Patent 2950048 USA, 1954

[2] Luhn, Pioneer of Information Science, selected works, Spartan Books, 1968

[3] Beckley, An Optimum System with Modulus 11, The Computer Bulletin, 11/1967

[4] Verhoeff, Error Detecting Decimal Codes, Mathematical Centre Amsterdam, 1969; second edition 1975.

[5] Website von Andries de Man, Checksum Calculators

[6] Damm, Total anti–symmetrische Quasigruppen, Doktorarbeit, Universität Marburg, 2004

Links

https://de.wikipedia.org/wiki/Pr%C3%BCfziffer

https://de.wikipedia.org/wiki/Pr%C3%BCfsumme

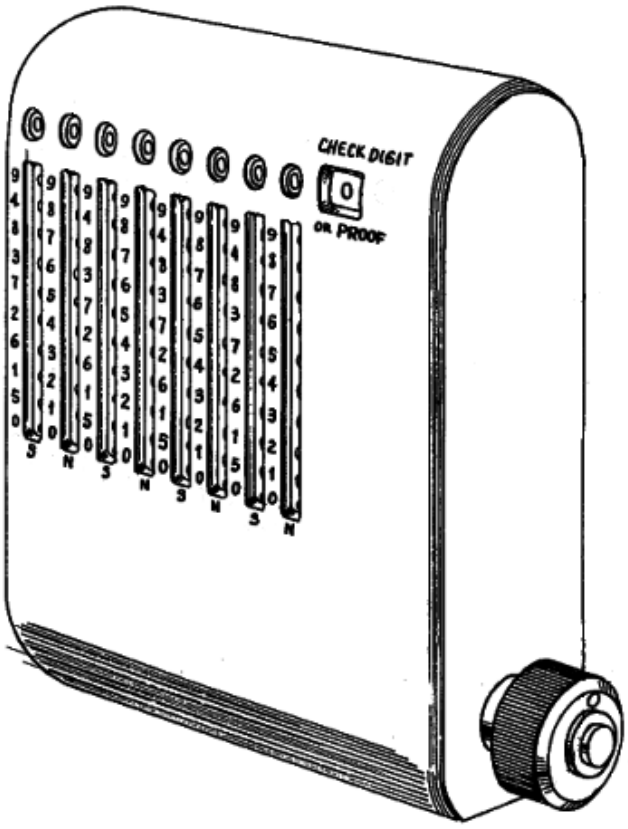


Abbildung 7: Rechner Luhn–Patent [5]

Listing

```
<| HEX \ Luhn

6 ZVARIABLE SN  \ ASCIIs 30 ... 39h
\ SN 0 + left digit
\ ...
\ SN 4 + right digit
SN 5 + CONSTANT CD" \ Checksum Digit

: SN.   \ (  --- ) print
5 0 DO SN I + C@ EMIT LOOP SPACE ;

: CALC  \ (  --- UC1 ) Luhn
0 4 0 DO
SN 4 I - + C@ 0F AND
I 1+ 01 AND
  IF 1<SHIFT DUP 09 U>
    IF 09 - THEN
  THEN +
LOOP
0 D% 10 U/MOD DROP
DUP IF D% 10 SWAP - THEN
30 OR CD" C! ;

\ 75872  CD" -> 2
\ 12345  CD" -> 5
\ 98765  CD" -> 1

|> <| HEX \ MOD11a

TABLE WEIGHT
08 C, 06 C, 04 C, 02 C, 03 C,
            05 C, 09 C, 07 C, \ MOD11a
06 C, 05 C, 04 C, 03 C, 02 C, \ MOD11b

: CALC    \ (  ---  ) MOD11
0 4 0 DO
SN     I + C@ 0F AND
WEIGHT I + C@ U* DROP +
LOOP
0 D% 11 U/MOD DROP
D% 11 SWAP -
DUP D% 10 = IF DROP 0 THEN
DUP D% 11 = IF DROP 5 THEN
30 OR CD" C! ;

|>
<| \ Verhoeff
TABLE D-TAB
0 C, 1 C, 2 C, 3 C, 4 C, 5 C, 6 C, 7 C, 8 C, 9 C,
1 C, 2 C, 3 C, 4 C, 0 C, 6 C, 7 C, 8 C, 9 C, 5 C,
2 C, 3 C, 4 C, 0 C, 1 C, 7 C, 8 C, 9 C, 5 C, 6 C,
3 C, 4 C, 0 C, 1 C, 2 C, 8 C, 9 C, 5 C, 6 C, 7 C,
4 C, 0 C, 1 C, 2 C, 3 C, 9 C, 5 C, 6 C, 7 C, 8 C,
5 C, 9 C, 8 C, 7 C, 6 C, 0 C, 4 C, 3 C, 2 C, 1 C,
6 C, 5 C, 9 C, 8 C, 7 C, 1 C, 0 C, 4 C, 3 C, 2 C,
7 C, 6 C, 5 C, 9 C, 8 C, 2 C, 1 C, 0 C, 4 C, 3 C,
8 C, 7 C, 6 C, 5 C, 9 C, 3 C, 2 C, 1 C, 0 C, 4 C,
9 C, 8 C, 7 C, 6 C, 5 C, 4 C, 3 C, 2 C, 1 C, 0 C,

: D-TAB@ \ ( col row  --- UC1 ) d(j,k) = column, row
D% 10 U* DROP + D-TAB + C@ ;

TABLE P-TAB
0 C, 1 C, 2 C, 3 C, 4 C, 5 C, 6 C, 7 C, 8 C, 9 C,
1 C, 5 C, 7 C, 6 C, 2 C, 8 C, 3 C, 0 C, 9 C, 4 C,
5 C, 8 C, 0 C, 3 C, 7 C, 9 C, 6 C, 1 C, 4 C, 2 C,
8 C, 9 C, 1 C, 6 C, 0 C, 4 C, 3 C, 5 C, 2 C, 7 C,
9 C, 4 C, 5 C, 3 C, 1 C, 2 C, 6 C, 8 C, 7 C, 0 C,
4 C, 2 C, 8 C, 6 C, 5 C, 7 C, 3 C, 9 C, 0 C, 1 C,
2 C, 7 C, 9 C, 3 C, 8 C, 0 C, 6 C, 4 C, 1 C, 5 C,
7 C, 0 C, 4 C, 6 C, 9 C, 1 C, 3 C, 2 C, 5 C, 8 C,

: P-TAB@ \ ( column row --- UC1 ) p(pos,num) =
0 8 U/MOD DROP  \ mod8 reduction for i > 8
D% 10 U* DROP + P-TAB + C@ ;

TABLE INV-TAB
0 C, 4 C, 3 C, 2 C, 1 C, 5 C, 6 C, 7 C, 8 C, 9 C,

: CALC    \ (  ---  )
0 CD" C!
5 0 DO
SN 5 + I - C@ 0F AND
          I   P-TAB@
       CD" C@ D-TAB@ CD" C!
LOOP
CD" C@ INV-TAB + C@ 30 OR CD" C! ;

\ 75872  CD" -> 2
\ 12345  CD" -> 1
|>
```

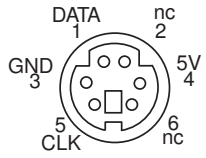
Numerische PS2–Tastatur

Rafael Deliano

Bei Prüfgeräten muss oft die Seriennummer der Baugruppe, die getestet wird, eingegeben werden, um die Rückverfolgbarkeit zu gewährleisten. Ideal wäre ein Barcode. Aber manchmal ist dafür kein Platz auf dem Bauteil, oder die Stückzahlen sind zu klein für aufwändige Lösungen.

Wenn der Benutzer Zahlen eintippen muss, ist eine abgesetzte kleine numerische PC–Tastatur am angenehmsten (Abb.1). Heute ist zwar die USB–Schnittstelle gängig, aber die alte PS2 ist weiterhin erhältlich. Bei der ist die Kabellänge unkritisch, was z.B. für Kassen in Supermärkten wichtig war. PS2 ist im Prinzip ungleich einfacher als USB für den Controller. Für den Test sind zwei Portpins ausreichend (Abb.4).

Abbildung 1: PS2–Tastatur



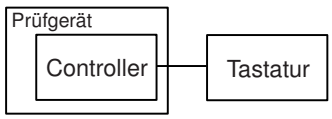


Abbildung 2: Blick auf Buchse am Controller | Blockschaltbild

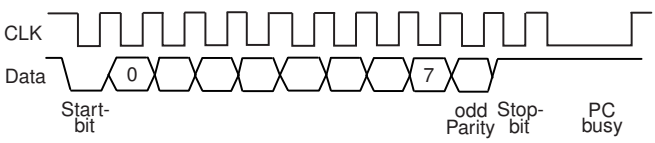


Abbildung 3: Datenübertragung

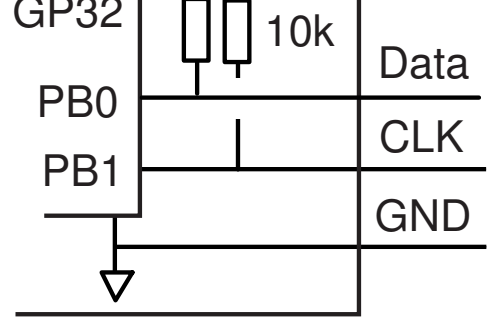


Abbildung 4: Breadboard–Schaltung

Eine bidirektionale Schaltung ist eigentlich nicht nötig, man kann damit ohnehin nur das LED schalten. Nützlicher ist es, den Controller besser gegen Abstürze durch ESD zu sichern (Abb.5).

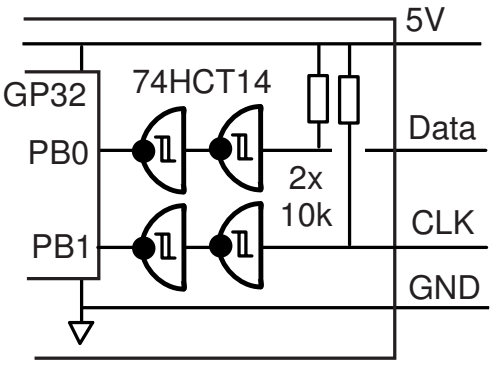


Abbildung 5: Verbesserte Schaltung

Die Datenübertragung von der PS2 ist eine Mischung aus UART und SPI (Abb.3). Die Tastatur sendet bei Tastendruck jeweils einen 3–Byte–Code (Abb.6, Tab.1). Der wird auch nicht durch NumLock verändert. Mangels Shift–Taste ist die alternative aufgedruckte Tastenbelegung hier ohne Bedeutung.

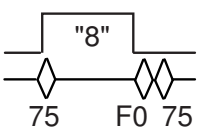


Abbildung 6: Tastendruck

Der Takt kommt von der Tastatur. Bei schnellen Controllern wäre sogar eine portable Programmierung in High–Level–Language möglich, aber Assembler ist bezüglich der Geschwindigkeit weniger riskant.

77 F0 77	NumLock
E0 4A E0	/
7C F0 7C	*
7B F0 7B	-
6C F0 6C	7
75 F0 75	8
7D F0 7D	9
79 F0 79	+
6B F0 6B	4
73 F0 73	5
74 F0 74	6
79 F0 79	+
69 F0 69	1
72 F0 72	2
7A F0 7A	3
E0 5A E0	Enter
70 F0 70	0
66 F0 66	Back Space
71 F0 71	.
E0 5A E0	Enter

Tabelle 1: Tastencodes

Listing

```
<| HEX \ PS2.txt

PB 0 DCONSTANT PS2-DATA-in   \ in
PB 1 DCONSTANT PS2-CLK-in    \ in

: INIT             \ (  ---  )
B% 11111100 DPB C!
B% 00000000  PB C! ;

3 ZVARIABLE N
\ --------------------------
\ test if idle

: WAIT     \ (   ---  )
BEGIN
PS2-CLK-IN B@      PS2-DATA-IN B@ LAND
IF 1
ELSE
  PS2-CLK-IN B@ LNOT PS2-DATA-IN B@ LAND
  IF  0
  ELSE   0
  THEN
THEN
UNTIL ;

\ --------------------------
\ Read PS2 Keyboard

:CODE BIT@             \ out: carry
1 $:           \ wait falling edge CLK
1 $ PS2-CLK-in    BBS,
        PB        LDA,
3 $:           \ wait rising edge CLK
3 $ PS2-CLK-in    BBC,
               A. LSR,
2 $               BCC,
        N 1+      INC,
                  SEC,
2 $:              RTS,
CODE;

:CODE ODD-PARITY?           \ in= carry
                            \ out: C=1
1 $                    BCS,
3 $           N 1+   0 BBS, \ 0,1 ok
2 $                    BRA,
1 $:
3 $           N 1+   0 BBs, \ 1,0 ok
2 $:              CLC, RTS,
3 $:              SEC, RTS,
CODE;

:CODE PS@                     \ (  --- UC1 )
4 $:     ' BIT@    JSR,      \ startbit
4 $                BCS,      \ error if 1 : PC busy
    N 1+ CLR, N 2+ CLR,
 ' BIT@    JSR,  N ROR,      \ bit 0
 ' BIT@    JSR,  N ROR,
 ' BIT@    JSR,  N ROR,
 ' BIT@    JSR,  N ROR,
 ' BIT@    JSR,  N ROR,
 ' BIT@    JSR,  N ROR,
 ' BIT@    JSR,  N ROR,
 ' BIT@    JSR,  N ROR,      \ bit 7
 ' BIT@    JSR,              \ parity odd
 ' ODD-PARITY?     JSR,
2 $                BCS,
       N 2+  02 #. MOV,      \ error CNTR = even
3 $                BRA,
2 $:     ' BIT@    JSR,      \ stopbit
3 $                BCS,
       N 2+  04 #. MOV,      \ error if 0
3 $: \   ' BIT@    JSR,      \ pseudobit: PC-busy
N    LDA, DEX, 0,X STA,
N 2+ LDA, DEX, 0,X STA,
                   RTS,
CODE;

: 3PKEY?         \ (  ---  )
PS@   \ makecode
PS@   \ F0 prefix
PS@   \ breakcode
CR CH. CH. CH. CR ;

: 6PKEY?          \ (  --- )
PS@   \ makecode  shift
PS@   \ ...
PS@   \ breakcode
PS@   \ makecode
PS@   \ F0 prefix
PS@   \ breakcode
CR CH. CH. CH. CH. CH. CH. CR ;

|>

```

PROMPT für Forth auf FreeDOS

Fred Behringer

Eigentlich hatte ich DX–Forth im Sinn. Das war das Forth, für das ich den PROMPT–Befehl (den es in DX–Forth nicht gibt) entwickeln wollte. Das habe ich leider noch nicht geschafft. Ich darf mich hier vorerst mit Turbo–Forth (TF) zufrieden geben. Das aber gleich unter FreeDOS. Gut zu gebrauchen war der SHELL–Befehl. Der lastet aber schwer auf dem Puffer für die Tasten: Nur 15 Anschläge sind möglich. Der Rest wird verschluckt. Die DOS–Shell braucht den Tastenpuffer als Zwischenspeicher. Damit wird ein vernünftiger PROMPT illusorisch. Ich übernehme ein kleines DEB–Programm aus [6] zur Vergrößerung des Puffers, mit dessen Hilfe mir die Anpassung an TF gelingt. Von PROMPT für DOS (FreeDOS) aus können auch die ANSI–Escape–Sequenzen — und Mischungen aus beiden — aufgerufen werden. Auch die stehen jetzt in TF direkt zur Verfügung.

Voraussetzungen

Turbo-Forth — ich verwende es in der 16–Bit–Version — bekommt man vom FTP–Server taygeta.com. FreeDOS ist im Programm–Paket www.sarducd.it als virtuelles Laufwerk a: enthalten. Eine wunderbare Sache! 109 Dateien! Bei Weitem mehr als im Original (bei FeeDOS). Eben „gepackt für SARDU“, dem Meisterwerk aus Sardinien. Liegt bei mir auf einem USB–Stick, kann direkt vom Stick gebootet werden und arbeitet prima. *nansi.sys* muss in der Form *device=nansi.sys* in der *config.sys* eingetragen werden. *nansi.sys, debug.com* und überhaupt alles, was man sich in diesem Zusammenhang wünscht, ist in den 109 Dateien von SARDU enthalten. (Wenn man mit *nansi.sys* wider Erwarten nicht zurechtkommt, bleibt das an- und abschaltbare *ansi.com* von Michael Mefford [5], das auch unter FreeDOS gut funktioniert.) Für die Vergrößerung des Tastatur–Puffers greife ich auf das Programm *kbdbuff.deb* von Thomas Jannot [6] zurück, mit welchem ich per *debug.com* aus dem SARDU–Paket [3] das Programm *keybuf.com* erzeuge.

Der DOS-Befehl PROMPT [Text] ...

ändert das Aussehen der „Eingabeaufforderung“ (engl.: "Prompt"). `Text` Spezifiziert dabei eine neue Eingabeaufforderung. Der FreeDOS–Befehl `PROMPT` kann aus normalen Zeichen sowie den folgenden Spezialcodes bestehen:

$$	$ (Dollarzeichen, trennt die PROMPT-Befehle)
$_	Wagenrücklauf und neue Zeile (CrLf)
$b	| (Befehlsverkettung {Pipe})
$d	Aktuelles Datum
$t	Aktuelle Zeit
$n	Aktuelles Laufwerk
$p	Aktuelles Laufwerk und Pfad
$h	Backspace (löscht das vorangehende Zeichen)
$q	= (Gleichheitszeichen)
$g	> (größer-als-Zeichen)
$l	< (kleiner-als-Zeichen)
$e	Escape-Code (ASCII-Code 27d)
$v	Versionsnummer der Command-Shell von FreeDOS

Wenn man `PROMPT` ohne Parameter eingibt, wird `der` Prompt in die Grundeinstellung zurückversetzt.

Das Vorstehende ist das, was ...

auf dem Bildschirm (so oder so ähnlich) angezeigt wird, wenn man von FreeDOS aus `PROMPT/?` eingibt. Mit diesen 13 Befehlen möchte ich mich hier beschäftigen. Sie können mit den ANSI–Escape–Sequenzen gemischt werdem. Diese werden unter `PROMPT` durch `$e` eingeleitet. Genauer gesagt, gehört zur Einleitung noch `$e[` dazu, was bei den Escape–Sequenzen dem `ESC[` entspricht. Ich habe das in [2] besprochen. Aufgerufen wird von FreeDOS aus mit `PROMPT $x` (also, um die Zeitanzeige als Beispiel zu nennen, mit `$t`). Mein Ziel besteht darin, auszukundschaften, wie der DOS–Befehl `PROMPT` (den es in Turbo–Forth (TF) nicht gibt — überhaupt wohl in keinem Forth) auch unter TF zur Auswirkung gebracht werden kann. In TF gibt es den Befehl `shell` für einen kurzen Abstecher nach DOS und Rückkehr zu TF. Den verwende ich hier. (In DX–Forth habe ich keine `shell` gefunden. Und mit dem dortigen `doscall` bin ich (noch) nicht zurechtgekommen.)

Die Sache mit *command.com* und `/k`

In den TF–Unterlagen zum Befehl `shell` steht viel über *command.com/c* , aber keine Warnung darüber, dass man zum Scheitern verurteilt ist, wenn man nicht *command.com* erstens in der expliziten Form, und zweitens mit dem Parameter /k (also *command.com/k*) verwendet. Eine schnelle Rückversicherung über *command/?* (auf der DOS-Ebene) löst den Fall: PROMPT ist nicht als einfacher (interner oder externer) DOS-Befehl zu werten, sondern stellt einen von *command.com* gesteuerten Sekundär-Befehl dar. Ein solcher wird nicht (wie etwa bei *keybuf.com*) einfach nur ausgeführt und leitet dann (ohne Zutun von *command.com*) automatisch die Rückkehr zu TF ein. Nein, er muss mit einem *command.com/k* (in der Shell) gekennzeichnet werden. Das */k*, um nach Einsatz der shell dem PROMPT auf der DOS–Ebene Handlungsmöglichkeit zu geben. */c* als Parameter in *command.com* wartet keinen PROMPT auf der DOS–Ebene der shell ab, sondern leitet sofort ein `exit` ein. Das heißt aber,

dass dem TF–System (auch in anderen Multi–Befehlen, nicht nur bei PROMPT) gesagt werden muss, dass ein *command.com/k*-Fall vorliegt, wenn das zutrifft. Mit anderen Worten, es muss gesagt werden, wann die */k-Kette* abgebrochen und zu TF zurückgekehrt werden soll. Das hat durch ein `exit` noch auf der DOS–Ebene der Shell zu geschehen. Diese ganze Erkenntnis hat mich viel Kopfzerbrechen und sehr viel Zeit gekostet. Übrigens kann man natürlich TF nach Aufruf der Shell (also nach Wirken von PROMPT) wieder durch ein `bye` verlassen - und noch übrigenser durch ein abermaliges `exit` abermals aufsuchen. Aber nur einmal (Radio Eriwan lässt grüßen)! Das heißt, wenn TF durch einen solchen Kunstgriff zu DOS (ich meine hier immer FreeDOS) entlassen wurde und ein zweites Mal `bye` gedrückt wird, ist es vorbei: Man befindet sich dann unwiederbringlich in DOS und müsste zum Wiedereintritt nach TF das System (unter Verlust aller bisherigen Einstellungen) neu aufrufen.

Die DOS-Shell ohne PROMPT

Bei schon aufgerufenem TF bewirkt die Zeile

```
" dir *.*" shell
```

die Ausführung des DOS–Befehls `DIR *.*` , dass also die Dateien des laufenden Verzeichnisses angezeigt werden und dass anschließend zu TF zurückgekehrt wird. Egal, ob mit (implizitem) Parameter `/c` im (impliziten) *command.com/c* oder auch mit explizitem *command.com/k* , also in der Form

```
" command.com/k dir *.*" shell
```

Ein Fall, bei dem PROMPT benötigt wird

Wiederum bei schon aufgerufenem TF sollte die Direkteingabe (also ohne die gleich zu besprechende DOS–Shell und ohne Veränderung des Tastenpuffers) der Zeile

```
" command.com/k prompt $e[42m$t$e[0m$p$g"
shell exit
```

das Folgende bewirken (das „sollte“ ist wichtig — siehe gleich):

$e[42m	setzt die folgende Zeitangabe auf die Farbe Grün
$t	liefert die laufende Zeit (mit grünem Hintergrund)
$e	liefert
[	in Verbindung mit der nach rechts offenen Klammer
$e[	die Eröffnung der ANSI-Escape-Sequenz (wie bei $e[42m)
$e[0m	und setzt insgesamt (über 0m) alle „Attribute“ zurück.
pg	Schließlich setzt pg den DOS-Prompt wieder auf „normal“. Übrigens gilt: Sämtliche Parameter direkt hinter $ können groß oder auch klein geschrieben werden. Für die ANSI–Escape–Sequenzen gilt das nicht unbedingt!

Setzt man in diesem Beispiel kein `exit`, dann kann man (wegen */k* statt */c*) lange darauf warten, dass etwas passiert. Es passiert nichts! Erst wenn man dann prompt $e[0m eingibt, passiert etwas: Das Farb–Attribut $e[42m wird zurückgesetzt. Wegen */k* befand sich die DOS–Shell noch auf der DOS–Ebene und ließ weitere PROMPTs zu. Wenn man dann (also noch auf der DOS–Ebene) `exit` eingibt, (dann erst) wird die Shell wieder verlassen und das System kehrt zu TF zurück. Wenn man jetzt aber `bye` eingibt, um aus TF wieder rauszukommen, darf man sich ruhig wundern: Vom ursprünglichen FreeDOS–Pfad kann keine Rede mehr sein. Nichts! Die Auflösung: Der PROMPT „ist verstellt“. Mit der Eingabe `prompt` (diesmal also nach einem `bye`) ist man wieder beim „normalen“ DOS–prompt (was auch immer man unter „normaler“ DOS–Eingabe–Aufforderung verstehen mag). Oder aber man hätte den zuletzt eingegebenen PROMPT gleich „richtig" gefasst.

Das hätte aber auch nicht viel gebracht. Denn: Was bei diesem Beispiel noch auffällt: Die Hintergrund–Farbe bleibt Grün, obwohl doch im eingegebenen PROMPT mit $e[0m die Farb–Einstellung wieder zurückgeschaltet werden sollte. Grund: Der Tastenpuffer ist zu klein. Bis zu $t wird der PROMPT gerade noch verarbeitet. Was darüberhinausgeht, wird verschluckt. Doch davon mehr im nächsten Abschnitt.

Man kann aber von dem Umstand Gebrauch machen, dass der nicht–enden–wollende PROMPT bei Einsatz von */k* (in *command.com/k*) in mehrere Teil–PROMPTs aufgespaltet werden kann: Gibt man statt der oben stehenden Beispiel–Eingabe ein:

```
" command.com/k prompt $e[42m$t prompt $e[0m$p$g"
shell exit
```

dann ist die Welt wieder in Ordnung. Schlimm ist nur, dass einem das keiner so recht sagt. Allerdings läuft dieser Fall (mit ... `shell exit`) nicht ganz so glatt, wie man es vielleicht erwarten würde: Mit dem ersten `prompt` wird die Zeit auf grünem Hintergrund aufgerufen. Dann passiert nichts. Wenn man die Eingabetaste drückt, wird wieder (und wieder) der erste PROMPT (grün) aufgespielt. Erst wenn man `exit` eingibt, tut sich was: Dann wird auch der zweite `prompt` ausgeführt: Dessen $e[0m sorgt (dann) für die Rücksetzung der Farb-Einstellung, das pg gibt der DOS-Eingabeaufforderung (nach Beenden von TF und Rückkehr per bye) wieder das gewohnte Aussehen — und die Shell wird wieder verlassen, indem zu TF unter den bisherigen Einstellungen zurückgekehrt wird.

Mit diesem Beispiel habe ich eigentlich schon alles gesagt. Ich darf darauf verzichten, die diversen Variationsmöglichkeiten zu untersuchen. Ich denke, dass ich mit dem im Abschnitt über die Shell gezeigten Verfahren zu mehr Vollständigkeit gelange und lade zum Selbstaustüfteln ein. Doch zunächst noch ...

. . . die Sache mit dem knappen Tastenpuffer

Das oben stehende Beispiel mit `$e[42m` geht gerade noch. Doch bei allen weiteren Versuchen stand ich kurz vor dem Verzweifeln: Sie gingen absolut nicht. Dann kam die Erinnerung: In [4] hatte ich gelesen, dass der Tastatur–Puffer 32 Zeichen (genauer: 15 Anschläge, da pro Taste zwei Bytes benötigt werden und auch noch die Eingabe–Taste fürs Beenden berücksichtigt werden muss) aufnimmt (nur aufnehmen kann!). Bei mehr Zeichen schließt sich der Puffer–Ring und es wird mit dem Belegen wieder von vorn begonnen. Ein schneller Versuch mit

```
" command.com/k prompt $_$_$_$_$_$_$_$_$_$_"
shell exit
```

zeigte, dass bei diesem Versuch keine 10 Zeilenvorschübe herauskamen. So etwa 5 oder 6 waren es. Damit war mir klar, dass die DOS–Shell von TF das Abarbeiten der PROMPT–Befehle dem Tastaturpuffer überließ (solange, bis zumindest alle PROMPT-Zeichen im Puffer lagen und auf die Weiterverarbeitung warteten).

Eine sofortige Abhilfe

Bevor ich das heutige Thema (DOS–PROMPT für Turbo–Forth) abschließe, eine Antwort auf die Frage, wie ich „die Sache mit dem Tastatur–Puffer“ aus der Welt schaffe: Ich greife auf eine Idee von Thomas Jannot [6] zurück: Im Anschluss an den Datenbereich des BIOSs (Segment 60) ist freier Platz für eine Verpflanzung (mit Verlängerung) des Tastatur–Puffers. Davon mache ich Gebrauch. Für die weitere Diskussion setze ich eine solche Verlängerung voraus. (Eine unerwartete und eigentlich überflüssige Schwierigkeit. Ich bespreche sie unten und vor allen Dingen im Listing.)

Die Sache mit der Shell und exit

Ich erkläre in der Auflistung, was man machen kann, um zu einem Tasten–Puffer zu kommen, der 127d Zeichen sammeln kann, welche dann in einem $–PROMPT unter TF verarbeitet werden können. Das erzeugte DOS–Programm (das auch unter FreeDOS läuft) nenne ich *keybuf.com* . Damit habe ich alles bereit, was ich zur Einbeziehung des DOS–Befehls `prompt` in Turbo–Forth brauche:

Ausgangspunkt: RAM–Disk y: (unter FreeDOS mit xmsdsk.exe erzeugt)
In y: liegt: command.com, keybuf.com, debug.com, tf.com
Direkteingabe: TF aufrufen
`" keybuf.com" shell` aufrufen
`" command.com/c"` shell aufrufen
`" command.com/k"` shell aufrufen
Danach: PROMPT $ . . . erzeugen
ev.wtr. PROMPTs $ . . . erzeugen
mit `exit` zurück zu TF
oder mit `bye` raus zu y:
Möglichkeit: Doch noch einmal per
exit (höchst. einmal exit !) zu TF
Oder indirekt: In TF `dos-prompt` erzeugen:

```
: dos-prompt ( -- )
" keybuf.com" shell
" command.com/c prompt $s$h" shell
" command.com/k prompt $s$h" shell ;
```

Dann: PROMPT $ . . . erzeugen
ev.wtr. PROMPTs $ . . . erzeugen
mit `exit` zurück zu TF oder mit `bye` raus zu y:
Möglichkeit: Doch noch einmal per `exit` (höchst. einmal `exit` !) zu TF

Bemerkung: `prompt $s$h` setzt den Cursor eine Position nach rechts und eine Position wieder zurück. `prompt` ohne Parameter würde als Beendigung des prompt–Vorgangs (auf der DOS–Ebene) gewertet werden und ein sofortiges `exit` (zur TF–Ebene) einleiten.

Mit dem Forth–Wort `dos-prompt ( -- )` habe ich das heutige Ziel erreicht. Zur Überprüfung (für mich und für den geneigten Leser) habe ich ganz zum Schluss (in Form eines direkt einsetzbaren eigentlichen Forth–Listings) das oben stehende Wort `dos-prompt` in Verbindung mit dem erweiterten Zeit–PROMPT nochmal aufgeführt (einfach kopieren und per `include incl-pro.txt` an das TF-System anhängen — soweit man mir gefolgt ist und die infrage stehende Datei mit `incl-pro.txt` bezeichnet hat.

Für ein sauberes Aussehen der Zeitanzeige wäre der (deutsche) Schriftzug noch um ein paar Cursor–Positionen nach rechts zu verschieben. Darüber habe ich auf die Schnelle nicht allzu stark nachgedacht. Aber das wäre ja auch nicht unbedingt eine dringende Aufgabe für den DOS–PROMPT unter Forth.

FreeDOS aus SARDU

Mein FreeDOS habe ich dem Programm–Paket *www.sarducd.it* entnommen. Lässt man die Farbgebung im oben stehenden Beispiel weg, ist der Zeit–PROMPT (wenn man mal von dem benötigten command.com–Parameter `/k` absieht) ganz einfach:

```
" command.com/k prompt $t" shell exit
```

liefert, unter anschließender Rückkehr nach TF, die nun farblos dargestellte Zeit:

```
10:15:11.12 am
```

Also die Stunde: 10, Minute: 15, Sekunde: 11.12 und am: ante meridiem.

Ich habe in früheren Zeiten (unter DOS, später auch noch unter XP) eine solche Zeitangabe rechts oben in der Ecke bevorzugt. Damals ging das, jetzt (unter der von Forth verursachten Zusatzaufgabe) sträubt sich der zu klein gewordene PC–Tastatur–Puffer. Mit Hilfe des verlängerten Puffers (siehe Listing) kann ich das aber jetzt (aus TF heraus!) durch[1]

[1] Den gesamten String in einer Commando-Zeile eingeben! Musste hier drucktechnisch umgebrochen werden, damit es ins Spalten–Layout passt.

```
PROMPT$e[s$e[1;66H$t$h$h$h$h$h$h$h$h$s
Uhr$s$s$s$s$s$[u
```

auch bewirken. Ein bisschen umständlich, ja, aber jedenfalls machbar. Ich darf aufschlüsseln:

$e[s	hält die momentane Cursor-Position fest.
$e[1;66H	setzt den Cursor auf Zeile 1 und Spalte 66.
$t	liefert die von FreeDOS stammende englische Zeitangabe
$h	(9x) neun Zeichen zurücksetzen
$s	ein Space setzen
Uhr	diese deutsche Bezeichnung anstatt der englischen verwenden
$s	(5x) die Sekunden und das englische am oder pm durch Spaces ersetzen
$e[u	stellt den Cursor wieder auf den in $e[s festgehaltenen Wert.

Weiteres steht im Listing, das man einfach als Kopie übernehmen kann.

Den Bildschirm löschen und grün färben

```
TF [ret]
dos-prompt [ret]
PROMPT $e[42m$e[u$p$g
exit
```

Mit diesem PROMPT–Befehl (der als Hauptbestandteil das Forth–Wort `dos-prompt` enthält) stellt man den Hintergrund auf Grün. Verzichtet man auf eine dann erfolgende Rückkehr zu TF und geht mit `bye` (statt `exit`) aus der Shell gleich raus, dann bleibt eben diese DOS–Einstellung (sprich: FreeDOS) bestehen. Wenn man abermals `bye` eingibt, für „immer“ (also bis zum nächsten PC-Reset). Wenn man es sich aber anders überlegt hat und nach dem „einmaligen“ `bye` dann `exit` eingibt, gelangt man wieder zu TF zurück. Aber immer noch mit einem grünen Bildschirm. Das ändert sich erst dann, wenn man die Sache (nach `bye` aus TF heraus) mit

```
TF [ret]
dos-prompt [ret]
PROMPT $e[0m$p$g
exit
```

oder nach `dos-prompt [ret]` direkt durch

```
PROMPT $e[42m$e[u$p$g
PROMPT $e[0m$p$g
exit
```

ungeschehen macht.

Literatur und Web–Links

[1] Armbrust, Steven, and Forgeron, Ted: Programmer's Reference Manual for IBM Personal Computers. Homewood, Illinois (1986).

[2] Behringer, Fred: DX-Forth auf FreeDOS mit ANSI–Escape–Sequenzen. Vierte Dimension 2/2018, S.16-21.

[3] http:www.sarducd.it Sardu, das Meisterwerk des sardischen Programm–Entwicklers Davide Costa, findet man im Internet auf Anhieb in allen erdenklichen Ausführungen, auch auf Deutsch. Ein Blick in die Computer–Zeitschriften lohnt sich ebenfalls. Ich habe meine Kopie von einer Begleit–DVD übernommen und sie mit den dort angegebenen Mitteln ohne Schwierigkeiten auf einen USB–Stick verfrachtet. Mit diesem SARDU–Stick kann ich den PC direkt booten — und SARDU (und insbesondere FreeDOS von dem im Text erwähnten virtuellen Laufwerk a: aus) verwenden.

[4] Hartwig, Olaf: PC/XT/AT für Insider. Markt&Technik–Verlag, München (1987).

[5] Mefford, Michael, J.: ANSI.COM . Aus dem PC Magazine des Jahres 1988.

[6] Schild, Gerhard, und Jannot, Thomas: 200 Utilities für PC–/MS–DOS. Eine Sammlung leistungsfähiger Assembler-Routinen, von Profis programmiert, in der Praxis bewährt. Markt&Technik–Verlag, München (1990).

Auflistung der Prozedur zur Vergrößerung des Tastaturpuffers

15 Tasten–Anschläge beim PC–Kompatiblen sind für längere PROMPTs viel zu wenig! Zumindest bei der beabsichtigten Aufbereitung von PROMPT für Forth. Mit diesem Programm von Thomas Jannot stehen 127 zur Verfügung. Das Programm nutzt den „freien“ Speicher von 0060h:0000 bis 0070h:0000 (also die 256d Bytes zwischen den DOS–Segmenten 60 und 70).

Achtung: Ich habe bemerkt, dass man beim Assemblieren über *debug.com* keine Großbuchstaben benötigt. Mit einer Ausnahme: Der Interrupt `INT 20` (Beenden des Programms) muss groß geschrieben werden. Die Rahmen–Instruktionen sind:

`debug.com` entnehme ich dem FreeDOS aus SARDU [3] (virtuelles Laufwerk a:).
`-` ist die Eingabe–Aufforderung von debug.com
`-?` liefert Information über debug.com
`-a100` damit beginnt Assemblieren (com-Programm-Anfang).
...
`-INT 20` damit endet unser keybuf.com–Programm.
`-nkeybuf.com` Wenn die Assemblier–Eingabe beendet ist, muss der Programm–Name eingegeben werden; `keybuf.com` gleich hinter dem `-n`.
`-w` erzeugt das Programm keybuf.com
`-q` damit wird schließlich debug.com verlassen. *keybuf.com* liegt ab dann zum Aufruf bereit.

(von DOS aus oder über `" keybuf.com" shell` im Forth).

Und weil die ganze Prozedur zum Erzeugen, das Assemblieren von *keybuf.com,* doch sehr ungewöhnlich ist, beschreibe ich hier den Ablauf der Terminal–Eingaben Schritt für Schritt. Segment und Offset sind natürlich vom verwendeten System abhängig.

`debug [ret]`
Das System gibt auf neuer Zeile aus:
-
Der Benutzer ergänzt zu:
`-a 100 [ret]`
Das System gibt auf neuer Zeile aus:
`1a36:0100` (Das ist die Anfangs–Adresse: seg:offs)
Der Benutzer ergänzt zu:
`1a36:0100 cld [ret]`
Das System gibt auf neuer Zeile aus:
`1a36:0101`
Der Benutzer ergänzt zu:
`1a36:0101 mov si,5d [ret]`
Das System gibt auf neuer Zeile aus:
`1a36:0104`
Der Benutzer ergänzt zu:
`1a36:0104 aad [ret]` usw.
...
Der letzte Assemblier–Befehl lautet:
`1a36:012b INT 20 [ret]`
Das System gibt auf neuer Zeile aus:
`1a36:012d`
Der Benutzer ergänzt zu:
`1a36:012d [ret]`
Das System gibt auf neuer Zeile aus:
-
Der Benutzer ergänzt zu:
`-[ret]`
Das System gibt auf neuer Zeile aus:
-
Der Benutzer ergänzt zu:
`-[ret]`
Das System gibt auf neuer Zeile aus:
-
Der Benutzer ergänzt zu:
`-rcx [ret]`
Das System gibt auf neuer Zeile aus:
`CX 0000 :`
Der Benutzer ergänzt zu:
`CX 0000 :2d [ret]`
Das System gibt auf neuer Zeile aus:
-
Der Benutzer ergänzt zu:
`-nkeybuf.com [ret]`
Das System gibt auf neuer Zeile aus:
-
Der Benutzer ergänzt zu:
`-w [ret]`
Das Systen meldet auf neuer Zeile:
`Writing 002D bytes.`
und gibt auf neuer Zeile aus:
-
Der Benutzer ergänzt zu:
`-q [ret]`

Der Vorgang ist damit abgeschlossen. Das Assemblat *keybuf.com* liegt auf dem Pfad, auf welchem auch *debug.com* liegt.

Es folgt die gesamte Assemblier–Prozedur der Übersichtlichkeit halber noch einmal so, wie sie sich ohne die zwischengeschalteten Erklärungen ergibt. Das ganze Listing ist natürlich kein Forth–Programm. Es ist ein DEB–Listing.

```
-a 100
1a36:0100 cld
1a36:0101 mov si,5d
1a36:0104 aad
1a36:0106 mov ah,al
1a36:0108 lodsb
1a36:0109 sub al,30
1a36:010b jnb 104
1a36:010d pop ds
1a36:010e mov wo[41a],200
1a36:0114 mov wo[41c],200
1a36:011a mov wo[480],200
1a36:0120 mov by[483],2
1a36:0125 add ah,ah
1a36:0127 mov [482],ah
1a36:012b INT 20
-rcx
-2d
-nkeybuf.com
-w
-q
```

Zum Schluss noch eine Methode zur Überprüfung

Gibt man unter *debug* `-d` ein, dann bekommt man ein Hex–Dump, dessen erste 45 Zeichen in Hex–Darstellung wie folgt aussehen (45d, also 2dh, ist die Länge von *keybuf.com*):

```
FC BE 5D 00 D5 0A 88 C4 AC 2C 30 73 F7 1F C7 06
1A 04 00 02 C7 06 1C 04 00 02 C7 06 80 04 00 02
C6 06 83 04 02 00 E4 88 26 82 04 CD 20
```

Dieselbe Information (und noch mehr) bekommt man auch nach Eingabe von `-u 100` (und danach dann `-u 120`). *debug.com* macht dann das Assemblieren rückgängig (es deassembliert *keybuf.com*).

Forth–Listing

```
1   \ Und hier zur Ueberpruefung des Forth-Wortes dos-prompt
2   \ ------------------------------------------------------
3   \ das Listing, das man nur zu kopieren und per incl-pro.txt (siehe Text)
4   \ an das schon aufgerufene TF-System anzuhaengen braucht.
5
6   \ Die Zeit angezeigt, erhaelt man natuerlich erst, wenn man dos-prompt
7   \ aufruft. Legt man den Zeit-PROMPT in die autoexec.bat, dann erhaelt man
8   \ nach jeder Betaetigung der Eingabe-Taste die jeweils neue aktuelle Zeit,
9   \ vorausgesetzt, man ist per exit wieder zu TF zurueckgekehrt.
10
11  \ incl-pro.txt ist die Datei mit dem Zeit-PROMPT-Ueberpruefungs-Programm.
12  \ Sie muss im selben Pfad sein (bei mir RAM-drive y:) wie die Programme:
13  \ TF.com, keybuf.com, command.com, freedos.com, nansi.sys.
14
15  : dos-prompt ( -- )
16  "  keybuf.com"               shell
17  " command.com/c prompt $s$h" shell
18  " command.com/k prompt $s$h" shell ;
19
20  \ dos-prompt aufrufen. Dann direkt eingeben:
21
22  \ prompt $e[s$e[1;66H$t$h$h$h$h$h$h$h$h$sUhr$s$s$s$s$s$e[u
23
24  \ Jetzt sollte die aktuelle Zeit oben/rechts angezeigt werden. Zu TF zurueck
25  \ kommt man durch Eingabe von exit. Die genauere Aufschluesselung dieses
26  \ prompts habe ich im Textteil im Abschnitt ueber die DOS-Shell besprochen.
27
28  \ Man kann natuerlich anstelle des Zeit-PROMPTs (oder sogar zusaetzlich
29  \ dazu) auch irgendeinen anderen PROMPT eingeben. Das exit zum Beenden des
30  \ Ausflugs nach DOS nicht vergessen! Zu TF wird erst nach Aufruf von exit
31  \ zurueckgekehrt.
32
33  \ ============================================================================
34
```

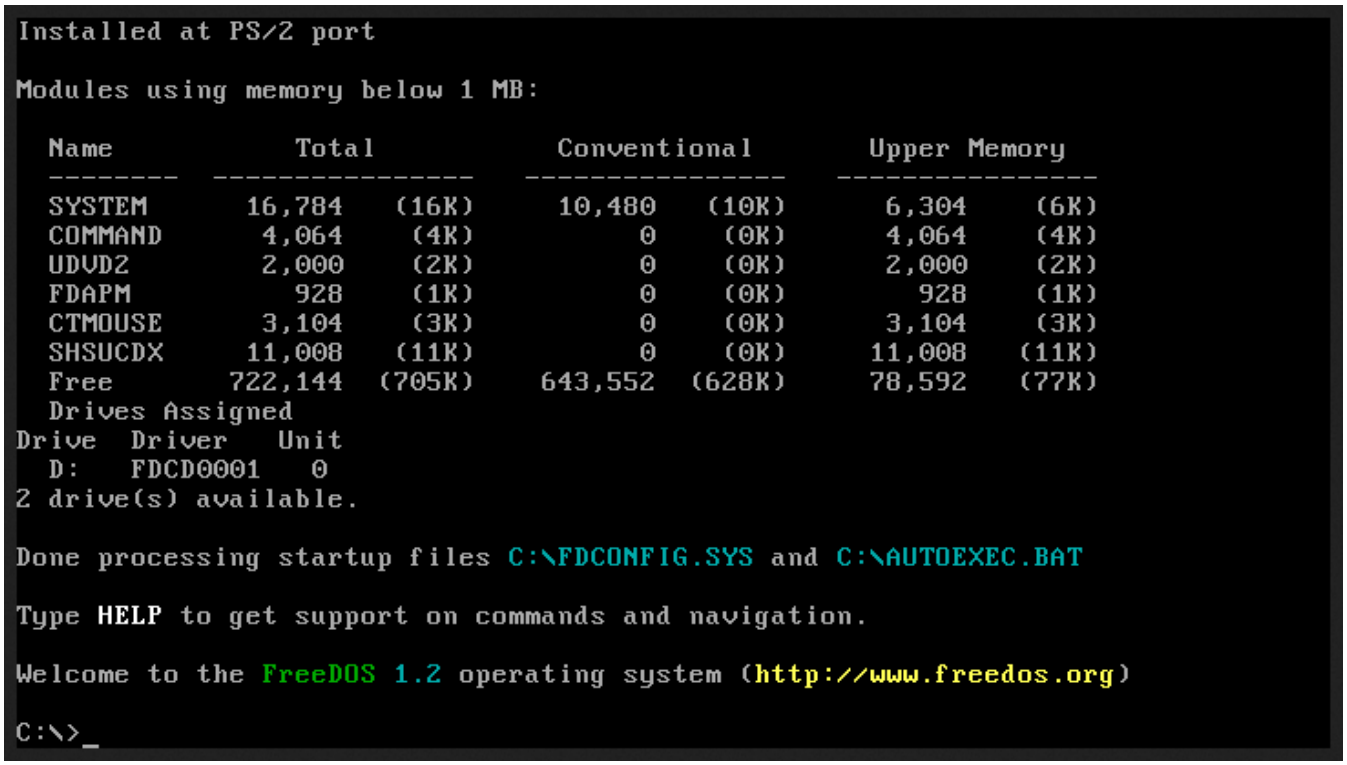


Abbildung 1: Screenshot von http://www.freedos.org/

Forth–Gruppen regional

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 89) – 41 15 46 53
`bernd.paysan@gmx.de`
Treffen: Jeden 4. Donnerstag im Monat um 19:00 in der Pizzeria La Capannina, Weitlstr. 142, 80995 München (Feldmochinger Anger).

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`
Treffen alle 1–2 Monate Freitags im Unperfekthaus Essen
`http://unperfekthaus.de`.
Termine unter: `http://forth-ev.de`

Gruppengründungen, Kontakte

Hier könnte Ihre Adresse oder Ihre Rufnummer stehen — wenn Sie eine Forthgruppe gründen wollen.

μP–Controller Verleih

Carsten Strotmann
`microcontrollerverleih@forth-ev.de`
`mcv@forth-ev.de`

Spezielle Fachgebiete

Forth-Hardware in VHDL microcore (uCore)	**Klaus Schleisiek** Tel.: **(0 75 45) – 94 97 59 3**$_p$ 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*`

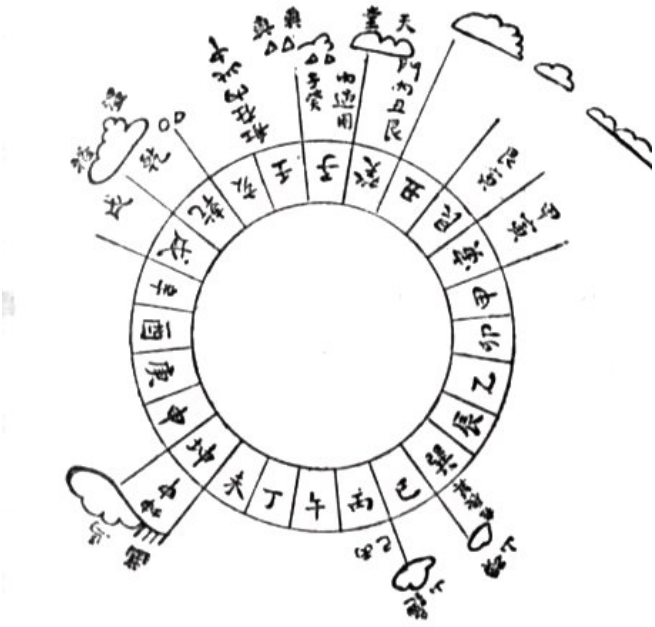
Datum
Forth–Tagung in ???
`https://tagung.forth-ev.de`

Datum
Maker Faire Hannover
`http://www.makerfairehannover.com`

Datum
EuroForth
`http://www.euroforth.org`

27.–30.12.20xx
3Xc3 in Hamburg

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.

Forth–Tagung in Worms

Ewald und Andrea Rieger

Die Tagung findet vom 11. bis 14. April 2019 im Hotel–Weingut Sandwiese *in der Nibelungen-, Dom-, Luther- und Weinstadt Worms statt. Das Hotel Sandwiese — Fahrweg 19, 67550 Worms–Herrsheim — liegt am Rande von Worms–Herrnsheim. In unmittelbarer Nähe befindet sich das Herrnsheimer Schloss mit seinem Park im Stil eines englischen Landschaftsgartens.*

Anreise

Worms ist gut über die A61 Abfahrt Worms–Mitte erreichbar. Bahnreisende erreichen den HBF Worms über Mainz oder Mannheim kommend und steigen dort auf den Bus nach Worms–Herrnsheim um.