1986

# VIERTE DIMENSION

Vierte Dimension
Volume II/Nr. 3

## QUEUES
## PERFORMANCE ANALYSIS
## DEFINING WORDS
## MACH 1
## FORTHS FÜR IBM-PC

# FORTH MAGAZIN

5.00 DM

Anzeige

# Mach Forth
# Multi Task Forth 83 für Ihren Macintosh
# Jetzt neue Version 1.25 !!!

**Mach1** ist ein multi-tasking-fähiges Forth für den Macintosh. Entwickelt als ein echtes 32-bit Forth 83 für und auf dem Macintosh wird es an der grössten Universität des Silicon Valley zum Lehren von Forth und 68000 Assembler verwendet.

**Mach ist eine echte Alternative zu anderen Programmierumgebungen** auf dem Macintosh. Interaktiv mit der Möglichkeit des direkten Wechsels (sogar über den Switcher) vom Editor (Edit wird mitgeliefert) zum Kernel, bietet es die Möglichkeit der schnellen und effizienten Programmentwicklung. Dabei brauchen Sie im Gegensatz zu anderen Versionen dieser Programmiersprache nicht auf floating point und lokale Variablen zu verzichten.

**Mach ist ein sehr schnelles Forth.** Durch den subroutine-threaded Code ergibt sich ein echter Geschwindigkeitsvorteil gegenüber jedem anderen bisher bekannten Forth. Mach ist nur doppelt so langsam wie compiliertes C und liegt damit durchaus im Bereich normaler Pascal Compiler.

**Mach erlaubt die einfache Generierung von Stand-Alone-Applications.** Durch das Wort "Turnkey" erzeugen Sie ein Programm, das nur ca. 16k länger ist als der Code ohne das Kernel. Jedes Programm, das unter Mach erstellt wurde, kann ohne Lizenzgebühren vertrieben werden.

**Mach ermöglicht Multi-Tasking.** Mach ist die einzige Programmierumgebung für dem Macintosh, die unbegrenztes Multi-Tasking zulässt.

**Mach hat eine Workspace Funktion**, die es erlaubt den gegenwärtigen Stand eines Projektes zu sichern, ohne am nächsten Tag bereits kompilierte Definitionen neu kompilieren zu müssen.

**Mach hat einen Standard 6800 Assembler.**

**Mach ist zukunftssicher.** Mach ist heute schon kompatibel mit MC 68020 Boards (das Sieb des Erathostenes dauert auf einem 16 MHz 68020 Board in Mach nur 1.19 Sekunden!!!!!!)

**Mach kann noch viel mehr:**
**Mach** kann sprechen (durch MacinTalk).
**Mach** hat Zugriff auf die Toolbox.
**Mach** hat Templates zur einfachen Erstellung von Menüs usw.
**Mach** kennt vectored I/O (das heisst Mach kann alle Ausgabedevices des Mac - inklusive Appletalk - bedienen und auch von dort Input annehmen).

Kurz: **Mach ist eine ideale Umgebung** für alle, die eine einfache aber extrem leistungsfähige Umgebung suchen und dabei nicht auf Standardsprachen wie C oder Pascal festgelegt sind. Mach stellt auch im Preis/Leistungsverhältnis eine einzigartige Alternative zu anderen Programmiersystemen dar.

## In Kürze auch für den Atari Computer !!!

**Hardware-, Software-, Dokumentation-, Entwicklungsgesellschaft mbH**
Alt Moabit 83
1000 Berlin 21
Tel.: 030 / 392 66 69

## EDITORIAL

Ich muß mich heute hier kurzfassen, weil ich 'nur' noch 32 Seiten für unser FORTH MAGAZIN zur Verfügung habe. Denn 32 Seiten sind genau ein Bogen Folie für den Offsetdruck. Und an diese Grenze müssen wir uns der Kosten wegen zunächst einmal halten. Doch dieses ist eine weise Beschränkung zugunsten bessere Qualität der Hefte. So haben wir mehr Möglichkeiten bei der Gestaltung erhalten und der Herstellungs-Zyklus - edieren, layouten, drucken - ist auch professioneller und damit pünktlicher geworden. Mal sehen, was sich in dieser Seitenzahl alles unterbringen läßt. Ich hoffe, daß auch in Zukunft mit *Forth weiter gut zu arbeiten sein wird* - und die erfreuliche Tendenz anhält, die Ergebnisse hier im FORTH MAGAZIN zu veröffentlichen.

### +++ +++ +++ AKTUELLE NACHRICHTEN +++ +++ +++

#### JAHRESVERSAMMLUNG

der Forth Gesellschaft 1986 wird am 15/16 November in der Lüneburger Heide sein. Das Tagungsprogramm und der genaue Ort werden noch bekannt gegeben. Achten sie auf die Einladung!

#### VOLKSFORTH UPDATE 3.8

noch vor Weihnachten. Alle Atari-User werden gebeten, Änderungswünsche, Bugs, usw. zum Volksforth, den Quellen oder dem Handbuch an die Autoren zu übermitteln. Die Version 3.8 wird von den bisher bekannt gewordenen Fehlern bereinigt sein, eine erweiterte Dokumentation der Quellen bieten und mit einem GEM-EDITOR ausgestattet. Das System wird zudem relocatibel gemacht. Schreiben sie an: Bernd Pennemann, Steilshooperstr.46, 2000 Hamburg 60.

#### MF1600VME

ist da. Auf 2 Doppeleurocards wurde dieser Computer um die neue METAFORTH MF1600CPU und eine 20MHz Clock herum gebaut. (Metaforth Computer Systems Ltd., Newlands High Technology Centre, Inglemire Lane, Hull, England HU6 7TQ. Telefon: 0482-855927, Telex: 592530 UNIHUL G)
Vertrieb auf dem Kontinent: Forth Systeme Angelika Flesch, Postfach 1226, 7820 Titisee-Neustadt, Telefon: 07651-1665 od. 3304.

INHALT




*Impressum*
Titel: VIERTE DIMENSION, Magazin für die Mitglieder der Forth Gesellschaft eV.
Herausgeber: Forth Gesellschaft eV, Friedensallee 92, 2000 Hamburg 50.
Forth: Klaus Schleisiek und die Mitglieder des Review-Boards sowie alle namentlich genannten Autoren.
Auflage: 500 Stück europaweit. Druck: *Offsetrepro Lüdemann, Wuppertal*
Erscheinungsweise: In jedem Quartal.
Redaktion & Anzeigenleitung: Michael Kalus, Präsidentenstraße 40, 5830 Schwelm, Telefon 02336-62204.
Redaktionsschluß: Der mittlere Quartalsmonat.

## LESERBRIEFE

### "Brief an den Editor" von J.E.Thomas, Birmingham, Alabama, USA, aus 'Forth Dimensions' VIII/1 86.

**Invisible, and Fairly Elegant: 32-Bit Forth**

Dear Marlin,

Thanks for Volume VIII I particularly liked "Synonyms and Macros" and hot-patching. I had wanted these and hadn't gotten around to writing them. They made Forth-83 look impressive: I could copy the 83-Standard ones word for word, and they worked before I understood them! The pseudo-interrupt technique is a nice idea I never considered. The whole thing is well *done*.

After reading Michael Hore's excellent letter (VII/3) about standards for Forth on improved micros, I have some alternate suggestions.

I have not yet seen Forth implemented on a full thirty-two-bit machine. I am using Forth-83 on 64K of a 68000. *I would like to* see the thirty-two bit standard look just like the sixteen-bit standard. **DUP** would be a thirty-two-bit **DUP**, **2DUP** would be a sixty-four-bit **DUP** and , would allocate thirty-two bits in the dictionary, which would have a maximum size of roughly four gigabytes (or less, depending on the trouble and expense of excessive memory). Everything would seem quite familiar. Mr. Hore's first point was that existing software should continue to run. Many simple Forth-83 programs would run without modification, except to a few basic words such as **ARRAY**. The problems that I imagine coming up are that, first, some things depend on going around the -32768/+32767 boundary and, second, sometimes people use *ad hoc* methods for things like arrays — for example, getting the address of an array element, and then adding two to get the address of the next element. Limiting a thirty-two-bit Forth to keep these functional would cripple it. Transmit for the thirty-two-bit Forth to be able to set aside 64K to run 79-Standard or 83-Standard programs.

Mr. Hore suggested setting up a constant named **LSIZE** which gives the stack word length, while the compilation address length stays at sixteen bits. @ and ! would transmit sixteen bits to allow **DO** ... **I@** ... **I +LOOP** to work. So long as the manufacturers make **LSIZE** in byte multiples, we can set **LSIZE** to 2, 4, 6, etc. I personally have set up the synonym **LS** and I now attempt to remember to write **DO** ... **I@** ... **LS +LOOP**.

*I see no problem with this general plan* for thirty-two-bit Forth. If your program is written with **LSIZE** and you don't use the -32768->32767 shunt (which I've used only for random number generators), it can be made to run in double precision with no run-time penalty but using twice the variable storage. *If it might not be* compatible, then you load a truncated vocabulary. Everything pretends to be sixteen bits starting at a base pointer, + and *, sign extend, etc. No problem.

What about segmented memory for sixteen-bit machines? I believe that this can be used very nicely within the standard. First, why not set aside 64K (more or less) *for a nice editor and mass* storage blocks? **VOCABULARY** can shift between 64K base addresses without the user having to pay any attention. If you had the memory, wouldn't you like a nice, big editor and a little RAM disk that just waited for you without using any of your 64K?

Are you at all cramped in 64K? We can bend the standard just a little bit and put all the machine code behind a different base address. Again, **EXECUTE** picks the correct base address without your knowledge. Of course, if you want to modify your machine code from Forth, you will need a couple of special words. ! and @ won't do it any more.

Do you want big arrays? Why not let **VARIABLE** set up space in a new area? You can have up to 64K of variables with sixteen-bit addressing. If you set up **VARIABLE**s, which is just like **VARIABLE** except that it starts at a different base address, then you can have a 64K array. With a slightly more complicated definition for **VARIABLE**, you should be able to set up arrays with up to 64K per dimension, although you'll run into physical limitations pretty fast.

The only reason Forth mixes machine code and Forth parameter values and variable contents together is for convenience, and with an 8086/8088 this isn't really convenient any more. They can be split up in a way that is almost invisible and is fairly elegant.

## Zum Stringstack

VDII/1, Stringstack von K.Schleisiek. Das Wort "JOIN ist ein wenig unsicher. Verbindet man nämlich zwei Strings mit zusammen mehr als 255 Zeichen, so wird in das Countbyte eine falsche Länge eingetragen, was das Wort TER in eine Endlosschleife stürzt. Um dieses zu vermeiden, habe ich noch ein ?STRING in "JOIN eingefügt und zwar vor dem R>.
Markus Redeker, Herten

## Forth Unterricht anbieten

Ich glaube es gibt viel mehr Forth-Anfänger, als wir wahrhaben wollen. Fast jeder meiner Bekannten - Computerbekannte natürlich - hat schon von Forth gehört und einige Tippversuche damit hinter sich. Da aber vielen die Sprache zu umständlich erscheint (*Originalzitat: "So komische Symbole wie @ ! und !, das liest sich wie ein Comic!"*) und durch viele kleine Hindernisse der Einstieg erschwert wird, ist keiner tiefer eingedrungen. Ich versuche natürlich wo es geht Basisarbeit zu leisten und den Leuten guten Forth-Stil beizubringen. Ekelhafterweise werden in *Programmzeitschriften - wenn* überhaupt - fast nur seitenlange Definitionen oder ähnlicher Mist abgedruckt. Wenn man sowas sieht, kann man sich natürlich für Forth nicht begeistern. Was können wir dagegen tun?

Bis zum nächsten Mal...

Konrad Scheller

## Sieves-Benchmark

In FORTH DIMENSIONS, Vol.VIII,No.2, fand sich diese richtiggestellte Version einer Routine zur Berechnung der Anzahl von Primzahlen, Sieves-Benchmark. Dies sei hier weitergegeben.

```
DECIMAL

8192 CONSTANT size
size SQRT 1+ CONSTANT flicklimit ( if you don't have SQRT, just use 4 / to be safe)
VARIABLE flags size ALLOT

: primes ( --number of primes)          ( does the primes once)
        flags size 01 FILL              ( initialize the array )
        0                               ( prime counter)
        size 2                          ( range of numbers to check )
        DO
          flags I + C@                  ( see if I is a prime)
            IF
            I flicklimit <      ( no need to try and flick flags once
                                  you get past size^0.5, just keep adding
                                  up the primes)
              IF
                I                   ( this is a prime. Used to increment the +loop)
                flags size +  flags I + I + ( range of addresses to tag)
                DO
                  0 I C! DUP        ( flick flags at multiples of I)
                +LOOP
              DROP      ( drop the I that was used for +LOOP)
              THEN
            1+          ( increment the prime counter )
          THEN
        LOOP ;
```

**How it works:** Initialize an array to all ones. Start with the first prime number, which is two. Clear all bytes in the flags array that are multiples of two. Then do it for the threes. The fourth byte in "flags" is a zero, so skip to five. Since four is non-prime, all of its multiples were handled by another number, two in this case. You only have to continue this process up to the integer square root of the array size, since in this case 91*91 is not in the array and the 91*90 spot was already zeroed by a previous prime number. If you dump the array, you'll see that bytes 2, 3, 5, 7, 11, 13, etc., are marked with ones, and the others have been zeroed. (While this program runs, it counts all the prime numbers it finds.)

**Saddled With Benchmarks**

Dear Mr. Ouverson:

Now that Mach1 (our Forth compiler) is becoming a rather efficient development system, we are somewhat concerned with the benchmarks that Forths are saddled with when comparing them to other languages. The only sieves we've seen people use must be corrupt versions of those called "BYTE" and "Colburn" sieves. They are not only wrong (there are 1028 primes between zero and 8192, not 1899) but, *judging from the stack manipulations, were* perhaps rewritten by people who were unfamiliar with, or unfriendly to Forth. Has someone pirated our benchmarks?

When a person truly wants to find prime numbers (and, indeed, our *customers are* always asking us for the higher primes), he'd like to do it simply, efficiently and correctly. Could you please publish the correct "BYTE" and "Colburn" sieves, and this Forth-83 sieve as ones which do just that?

Thank you,

Terry Noyes
Palo Alto Shipping Co.
Menlo Park, California

## Preisrätsel

Prämiert wird diejenige Einsendung, die für die Definition in Bild 1 den einzig passenden Namen vergibt und die Funktion verständlich erklärt.
Klaus Schleisiek, Hamburg

(VD: Wie ich hörte, soll die Prämie ein BIM-Clown Komputer und eine Woche Hamburg auf Klaus' Kosten sein! Na denn man los.)

```
: ???   ( u1 - u2 )
   dup 0= if exit then
   1 swap 0 do 2+ dup +loop
   1- 2/ ;
```

* * *

## AUS DEM VEREIN

### Das Forthbüro in Hamburg

Das Büro der Forth Gesellschaft eV hat seit dem 1-JUN-86 eine neue Adresse, die Bankverbindung ist die alte geblieben. Hier im Büro gibt es jeden Dienstags von 18-20 Uhr mündliche Auskünfte und Support für Forthler. Zu allen anderen Zeiten piept einem der TREE entgegen !

Zum gleichen Dienstags-Termin kann auch (nach vorheriger Absprache) das Büro besucht werden, z.B. zwecks Literatursuche oder TREE-Bestaunens.
Marco Pauck

| Das Forthbüro |
|---|
| Forth Gesellschaft eV, Friedensallee 92, 2000 Hamburg 50 |
| Tel.: 040 - 3904204 |
| Postgiroamt Hamburg, Kto: 56 32 11 -208, BLZ: 200 100 20 |

## Na endlich, München!

Zu meiner großen Freude sehe ich, wie die erste lokale Forthgruppe in München Gestalt annimmt. Heinz Schnitter ist dabei der Kristallisationspunkt, und das freut mich doppelt! Wer seit dem Dachsbergtreffen in Neukirchen-Flyn dabei ist und auch im letzten Jahr bei der Jahresversammlung in Bad Bergzabern oder auf Burg Stettenfels bei der Euroforml war, wird ihn kennen.

Meine Meinung dazu: Wer ernsthaft daran interessiert ist, Forth kennenzulernen, oder mit Praxisfragen als Forthanwender bisher allein stand, hat im Raum München ab heute mit der Gruppe um ihn eine sehr gute Adresse an der Hand. Vermutlich wird es sich lohnen zu den Treffen zu gehen. Es gibt seit langem Anzeichen dafür, daß in München und Umgebung bereits in Industrie und Forschung für allerlei interessanten Anwendungen sehr viel mit Forth gearbeitet wird, Und ich hoffe, daß sich dort eine Idee der Forth Gesellschaft durchsetzt: Die public domain des Forth.

Ein schöner Zug des Forth ist es ja, daß man nicht gleich Betriebsgeheimnisse veraten muß, um die Forth-öffentlichkeit zu bereichern, was, wenn dieses viele tun, für alle ein Gewinn wird. Es reicht ja aus, ein gut gelungenes Modul aus einer Applikation vorzustellen. Oder einfach zu berichten, was man mit Forth gemacht hat. (Sicher finden sich Leute, die liebend gerne gutes Geld dafür gäben, eine Applikation fertig zu erwerben, wenn sie nur wüßten wo...)

Ich wünsche der lokalen Gruppe München viel Erfolg.

* * *

## DAS INTERESSANTE WORT

?" ( char -- f )

?" ist immer dann nützlich, wenn in Abhängigkeit von einem Zeichen (Byte) eine Aktion ausgelöst werden soll. Oder dann, wenn man überprüfen will, ob der auf dem Stack liegende Wert ein ganz bestimmtes Zeichen ist. Statt ASCII A = kann man nun ?" A" schreiben.
HEXDIGIT? ist zb. ein Wort, das eine hexadezimale Ziffer in die zugehörige Zahl umwandelt. Der Code ist in Forth-83 geschrieben.
Klaus Schleisiek, Hamburg

```
\ einige nuetzliche worte zu ?"                          ks  1 jun 86
  : move    ( from to len -- )
     >r  2dup u<  IF  r> cmove> exit  THEN  r> cmove ;
  : place   ( addr len to -- )
     over >r   rot   over 1+   r> move  c!  ;
  : word    ( char -- here )   word dup count here swap place ;

\ WORD redefinieren, wenn System-WORD den string nicht nach
\ HERE packt.
  : under   ( n0 n1 -- n1 n0 n1 )   swap over ;
```

```
\ ?"   zeichen vergleich mit in-line string          ks  1 jun 86
  : ,"     Ascii " word     c@ 1+ allot ;
  : (?"  ( 8b -- index )
     r> dup count + >r    ( haengt vom Prozessor ab, hier Z80)
     swap >r    count under +
     BEGIN  over
     WHILE  1-    dup c@ r@ =     IF  r> drop   drop exit  THEN
            swap 1- swap    REPEAT    r> drop   drop ;
  : ?"    compile (?" ," ; immediate
  : hexdigit?  ( char -- n tf / ff )
     ?" 0123456789ABCDEF"  dup IF  1- True  THEN ;
```

*Forthsysteme im Vergleich*

Das FORTH MAGAZIN 'Vierte Dimension' druckt Beiträge über Forthsysteme gerne ab, um zur Bildung der allgemeinen Kritikfähigkeit bei der Beurteilung von Forthsystemen beizutragen. Wir weisen hier aber nochmal ausdrücklich darauf hin, daß Anwender mit anderen Interessen - und anderen Kriterien - als der jeweilige Autor, durchaus zu anderen Beurteilungen gelangen können. So kann ein 'primitives' uralt-minimal-Forth durchaus die richtige Lösung einer Applikation sein (Steuerung mit Einplatinenrechner, intelligente Peripherie, etc).

'Gut' oder 'nicht gut' ist immer eine Frage des: Gut wofür? Möge sich der Leser sein eigenes Urteil bilden, ob ein Forthsystem für ihn geeignet ist oder nicht. Aber eines ist sicher, public domain Systeme sind kein Risiko, denn sie kosten so gut wie nichts. Und um Erfahrungen mit Forth zu sammeln, sind sie allemal gut genug.

Artikel, die Forth-Versionen verschiedener Hersteller vergleichen, geben die Meinung des jeweiligen Autors wieder. Der Autor sollte daher im Artikel stets deutlich darlegen, was seine Kriterien waren.

Das FORTH MAGAZIN wünscht sich mehr solcher Berichte über Forthsysteme, am liebsten über die Erfahrungen, Tücken, Bugs etc mit dem eigenen System. Die Schöpfer der übrigen Systeme sind dann ebenfalls herzlich eingeladen, ihre Produkte durch Anwenderberichte eingehender beschreiben zu lassen.

* * *

Forth Coding Convention

Im letzten Heft wurde über Forth STYLE gesprochen. Kim R. Harris hat eine 44 Seiten starke Sammlung herausgebracht, in der Vorschläge für Konventionen zur Gestaltung von Forthcode gezeigt werden. Das Heft ist als Leitfaden zu verstehen und nicht als Vorschrift. Es ist ein guter Führer hin zu lesbaren und gut dokumentierten Forthprogrammen. Behandelt werden: form of comments, first line comments, placing definitions, spacing between words and phrases, indentation, stack diagrams, descriptors, fields, variable number of stack arguments, multiple interface comments, general guideline to wordnames, control procedures, input stream operators, data structure words.

Kopien können in der Redaktion des FORTH MAGAZINES bezogen werden. mk

# Public Domain Forths für den IBM-PC

George Bruziks, Koblenz

## FORTH auf IBM-Kompatiblen

Nach dem Einstieg in die riesige Welt der IBM-Software stand ich einer großen Zahl von Forths gegenüber. Von diversen Anbietern werden mehr oder weniger komplette Systeme zu Preisen ab 250,- DM angeboten, aber interessant ist gerade für Hobbyisten wie mich die Public-Domain-Software. 5 verschiedene Forth-Versionen hatte ich zu Verfügung. Einerseits macht es Spaß, aus einem so großen Angebot wählen zu können, andererseits ist es aber auch langwierig und manchmal auch langweilig, sich durch verschiedene Editoren und Disketten-Betriebssysteme zu kämpfen.

Bei allen Versionen waren die Anleitungen auf den gelieferten Disketten, leider teilweise im Wordstar-Format, was den Ausdruck nur mit Wordstar erlaubt. Ich machte einige Geschwindigkeitstests, hierbei nahm ich auch eine mir zugängliche professionelle Version als Vergleich hinzu.

Folgende Kriterien finde ich für die Beurteilung einer Forth-Implementation auf einem IBM wichtig:

1. Geschwindigkeit

   Wenn Forth auch sicherlich schnell ist, so gibt es dennoch deutliche Unterschiede bei der Ausführungsgeschwindigkeit. Nur diese finde ich bei der Beurteilung wichtig, die Compilierzeit habe ich nicht betrachtet.

2. Schnittstelle zum Betriebssystem

   MS-DOS ist unbestritten ein ausgereiftes und komfortables Betriebssystem. Warum also aus puristischen Gründen auf die genormte Schnittstelle verzichten? Außerdem ist es manchmal nicht nur bequem, sondern auch wichtig, fremde Files von Forth aus lesen zu können, bzw. mit Forth geschriebene Daten von anderen Programmen weiterverarbeiten zu lassen.

3. Normgerechtigkeit

   Die schönsten Features und die beste Ausführungsgeschwindigkeit sind nicht viel wert, wenn man dafür auf die die Kommunikation mit anderen Forthlern verzichten muß. Exotische Dialekte sind deshalb uninteressant und fördern auch nicht die Verbreitung der Sprache.

4. Dokumentation

   Jede Implementation sollte in mindestens einer Beziehung dokumentiert sein: da wo sie vom Standard der Sprache abweicht. Eine Dokumentation kann keine Einführung in die Sprache Forth sein und braucht auch nicht jedes Standardwort zu beschreiben, dafür gibt es ausgezeichnete Literatur. Nur dort, wo der Standard verlassen wird, muß der Unterschied dokumentiert sein.

Die Benchmarktests erheben keineswegs den Anspruch auf Professionalität. Mir ging es darum, die verschiedenen Systeme miteinander vergleichen zu können. Der Code der einzelnen Tests ist als Listing im Anhang. Zum Vergleich habe ich die Tests auch mit einem professionellen System durchgeführt ( PC FORTH, Copyright by Laboratory Microsystems ). Alle Zeiten in Sekunden, durchgeführt mit einem 8088 Prozessor bei 4.77 MHz. Der letzte Test ( bmV20 ) wurde mit einem "getunten" PC durchgeführt: Taktfrequenz 8 MHz und V20-Prozessor. Er ist vom Listing her identisch mit bm8.

| System | bm1 | bm2 | bm3 | bm4 | bm5 | bm6 | bm7 | bm8 | bm-V20 |
|---|---|---|---|---|---|---|---|---|---|
| PC Forth | 33.8 | 29.8 | 70 | 32.0 | 44 | 45.8 | 34.0 | 37 | 18 |
| Uniforth | 24.0 | 23.0 | 68 | 31.0 | 42 | 43.0 | 33.0 | 50 | 24 |
| Laxen & P. | 35.6 | 33.0 | 87 | 40.0 | 343 | 55.8 | 41.6 | 253 | 133 |
| MVP | 95 | 65 | 165 | 64 | 468 | 91 | 64 | 343 | 185 |
| Dallas | ≈ Forth-H | | | Zeiten genau wie MVP | | | | | |
| FIG | 49 | 42 | 145 | 46 | 427 | 64 | 50 | 308 | 167 |

Die teilweise fürchterlichen Zeiten bei bm5 und bm8 rühren daher, daß die Programmierer es sich teilweise recht leicht gemacht haben: für arithmetische Operationen wurden einige 32-bit Worte in Assembler definiert, der Rest wurde in Hochsprache darauf aufbauend geschrieben. Natürlich macht diese Technik es leicht, ein Forth-System auf verschiedenen Prozessoren zu implementieren. Diese Technik führt aber auch zu Geschwindigkeitsunterschieden um bis zu 1000 % ( siehe bm5 ), was ich nicht mehr tragbar finde. Was aber fast schon zornig macht, ist die Art mit der Code immer weiter abgeschrieben wird, so daß man eklatante Mißstände der ersten Stunde im 1985er Code findet.

## Zusammenfassung

Abgesehen von Uniforth ( und die haben die Cursortasten auch nicht gefunden ) schlägt man sich mit mühseligen Zeilen-Editoren herum. Für den Zugriff auf mittlerweile praktisch überall vorhandenen reichlichen Speicher findet man nirgends wirklich einfache Routinen; hier wird der konservative PC von den neuen 68000ern QL, Atari und Amiga überholt, denn für diese Maschinen stehen Forths mit 32 Bit breiten Adressen zur Verfügung und werden wohl für einen neuen Standard sorgen. Als weiteres, schwerwiegendes Manko stellt sich das teilweise unkompatible Disk-Format dar. Warum soll ich von Forth aus *nicht mit anderen Dateien kommunizieren* dürfen?

Unter diesen Gesichtspunkten bleiben zwei Forth-Versionen übrig, die interessant sind:

1. Laxen & Perry Forth
   Das gesamte Forth liegt als Forth-Source vor und kann daher gut verbessert werden. Man kann einen vernünftigen Editor schreiben, man kann, man kann. Er ist eben noch nicht da. Trotzdem, für Hacker die große Chance! ( Vielleicht hat ja schon jemand so ganz allein für sich ....? )

2. Uniforth
   Ein schönes System für jeden, der schreiben will und sich nicht mit dem System selbst herumschlagen will. Sehr schnell, komfortables I/O, mit Tricks ist auch der gesamte Speicher nutzbar, brauchbarer Editor.

Die restlichen Versionen? Gut fürs Museum.

# Public Domain Forths für den IBM-PC

## FIG-FORTH

Anbieter:
8086 FIG Forth Version 1.0 A
Joe Smith
University of Pennsylvania

Lieferumfang:
1 Diskette mit ausführbarem Code. 21 Seiten Informationen. Source-Code in Assembler, nur geeignet für Seattel Computer Products Assembler.

Besonderheiten:
Keine MS-DOS Files auf Diskette, also kein Kontakt zur Außenwelt möglich, aber in der Dokumentation ist ein etwas mühseliger Weg beschrieben. Files zwischen MS-DOS und FIG-DOS mit Hilfe von DeBug auszutauschen. Besonderheiten gegenüber dem originalen FIG sind in der Dokumentation aufgeführt.

Editor:
Zeilenorientierter Editor, wie in Starting Forth beschrieben.

Vorteile:
* FIG Standard

Nachteile:
* langsame Divisionsroutine
* ungewöhnlicher Assembler macht Änderungen im Source schwer
* zeilenorientierter Editor
* kein MS-DOS Diskettenformat
* keine Backspacetaste !!!

## FORTH-H

Anbieter:
Dallas-Forth 1.0
Glen Haydon

Version:
F 79 Dialekt ( MVP )

Lieferumfang:
1 Diskette, 2 Seiten Dokumentation

Besonderheiten:
Als Dokumentation wird auf das Buch "All about Forth" verwiesen. Nicht MS-DOS-kompatibel.

Editor:
Zeileneditor

Vorteile:
* gegenüber F 79 einheitliche Darstellung der doppeltgenauen Worte

Nachteile:
* nicht MS-DOS kompatibel
* langsam

## MVP-FORTH

Anbieter:
MVP-Forth Version 1.0305.03
Mountain View Press Inc

Version:
Forth 79 Dialekt, weitgehend FIG-kompatibel

Lieferumfang:
2 Disketten. Kurze Liste der verfügbaren Worte ohne irgendwelche Erklärungen, 2 Seiten Beschreibung mit den wichtigsten Warnungen.

Besonderheiten:
Diese Forthversion richtet sich ihr eigenes Betriebssystem ein. Diskettenformat nicht MS-DOS kompatibel. Die Bildschirmdarstellung mit 40 Zeichen/Zeile erinnert mehr an Telespiele als an seriöse Anwendungen.

Editor:
Zeilenorientierter Editor.

Vorteile:
* gegenüber 79 einheitliche Darstellung der doppeltgenauen Worte
* viel Sourcecode

Nachteile:
* unpraktisches Bildschirmformat
* keine MS-DOS Kompatibilität
* langsam

## UNIFORTH-SAMPLER

Anbieter:
Unified Software Systems, USA 1985

Version:
Forth 83 mit geringen Einschränkungen

Lieferumfang:
1 Diskette mit fertig compiliertem System incl. Editor und Floating-Point-Paket. 20 Blocks Beispiele. Kein Sourcecode des Systems. 40 Seiten Beschreibung mit guter Erklärung der Besonderheiten sowie 40 Seiten mit einer Liste der wichtigsten Worte.

Besonderheiten:
Bei den Beispielen sind Definitionen, die den Zugriff auf den gesamten Speicherbereich des PC zulassen. Als einzige aller getesteten Versionen wird der Schleifenindex unsigned benutzt, d.h. eine Schleife mit 50000 0 do loop wird wirklich 50tausend mal durchlaufen. Im Gegensatz zum 83er Standard wird die Division nicht gs'floored'. Turnkey-Applikationen werden unterstützt. Sehr komfortable I/O-Routinen. Diskettenbetrieb wird hervorragend unterstützt. Insbesondere ist nicht nur blockorientiertes Lesen und Schreiben möglich, sondern auch random-access. Einfaches Betreiben von zwei Disk-Kanälen nebeneinander. Kein Multi-User oder Multitasking vorgesehen.

Editor:
Der mitgelieferte Editor arbeitet nicht zeilenorientiert, sondern full-screen. Er ist nicht sehr schnell und nur mäßig komfortabel. Leider liegt er nicht im Source vor.

Vorteile:
* schneller Code
* brauchbarer Editor
* Floating-Point-Paket
* bequemes Filehandling
* Turnkey-Möglichkeit

Nachteile:
* kein Sourcecode, daher schlecht änderbar

## Laxen & Perry Forth

Anbieter:
8086 Forth Model
Version 2.2 Modified 10 Apr 85
by F.-Th. Ernst (MS-DOS, SMART)

Version:
Forth 83

Lieferumfang:
2 Disketten mit ablauffähigem System, Meta-Compiler, komplettem Source, Editor und Extensions. Der gesamte Code ist kommentiert. Keine gesonderte Beschreibung. Kein Floating-Point.

Besonderheiten:
Ein echtes Forth 83. Durch den mitgelieferten Source ist es ein System für Hacker. Ein Feature ist ein Multitasker, der sehr interessant realisiert, aber leider nur wenig kommentiert ist. Der Decompiler mit der komfortablen Möglichkeit, sich auch den Source-Screen des Wortes anzeigen zu lassen, ist sehr eindruckevoll. Das Konzept der "Shadow-Screens" - der Source steht in der ersten Hälfte des Files, die Kommentarscreens in der zweiten - ist eher eine Schnapsidee.

Editor:
Für den Gebrauch des Editors muß das System mit ANSI.SYS konfiguriert sein. Der Editor ist zeilenorientiert, liegt aber als Source vor.

Vorteile:
* alles liegt im Source-Code vor
* mitgelieferter, wenn auch langsamer, Multitasker
* voll Standard
* guter Decompiler

Nachteile:
* langsamer Code
* unbequemer Editor
* standardgemäße Division gewöhnungsbedürftig
* ungewöhnliche Schreibweise im Assembler

## Public Domain Forths für den IBM-PC

```
SCR # 1
  0 ( Benchmarktests                  8.86 )
  1
  2 : BM1  1000 0 DO 1000 0 DO LOOP LOOP ;
  3
  4 : BM2 1000 0 DO 1000 0 DO 2 +LOOP ;
  5
  6 : BM3 10 0 DO
  7       1 BEGIN 1+ DUP 64000 = UNTIL DROP
  8       LOOP ;
  9
 10 : BM4 10 0 DO 30000 0 DO I 100 + DROP LOOP LOOP ;
 11
 12 : BM5 10 0 DO 30000 1 DO I 7 / DROP LOOP LOOP ;
 13
 14 0 VARIABLE TEST
 15 : BM6 50 0 DO 10000 0 DO I TEST ! LOOP LOOP ;


SCR # 3
  0 ( Benchmarktests cont )
  1
  2 0 VARIABLE NPRIM
  3
  4 : PRIM? DUP 2/ 2 DO DUP I MOD           ( n --- )
  5                   IF
  6                   ELSE 1 NPRIM +! LEAVE
  7                   THEN LOOP DROP ;
  8
  9 : BM8   3 NPRIM !
 10        DUP 8 DO I PRIM? LOOP DUP
 11        CR CR ." Bis " . ." sind es "
 12        NPRIM @ - . ." Primzahlen " CR ;
 13
 14
 15 ;S


SCR # 2
  0 ( Benchmarktests cont )
  1
  2 : BMV 30000 0 DO 25
  3                  IF 100
  4                     IF
  5                     ELSE
  6                     THEN
  7                  ELSE
  8                  THEN
  9               LOOP ;
 10
 11 : BM7 8 0 DO BMV LOOP ;
 12
 13
 14
 15
```

**Public-Domain Floating Point and Double/Quad Precision**

Dear Marlin,

Forth has gone too long without a complete, standard wordset for extended arithmetic functions! Instead of continually reinventing the wheel with hardware-dependent math packages and spending time writing yet another version of D*, the Forth community needs to expend its energy addressing new horizons.

In order to foster the rapid growth of a standard, I am placing my book, *MVP-FORTH Integer and Floating-Point Math* (MVP book series, volume three), in the public domain. This book contains a complete, machine-independent, high-level MVP-FORTH glossary and implementation for thirty-two-bit integer math, sixty-four-bit integer math and thirty-two-bit floating-point math with transcendental functions. The book also includes assembler source code for critical words on popular CPUs, to give execution speeds comparable to other high-level languages.

The math package included with the book has been stable and in active use for several years. The word definitions are similar or identical to many of the other proposed "standards." However, this wordset has the advantages of machine independence and public-domain implementation. The book and source code on disk may be ordered from Mountain View Press.

I think that a standard for Forth arithmetic will only emerge from the evolution of a public-domain implementation in common use. I encourage anyone with comments on, or improvements to the implementation described in my book, to write me in care of Mountain View Press.

Phil Koopman
North Kingstown, Rhode Island

## Machi, ein Forthsystem für den Macintosh

Holger Blum, Hamburg

Der Artikel wurde mit Mac-Write auf einem Macintosh plus erstellt und mit dem Imagewriter von Apple gedruckt. Er ist auf den folgenden Seiten original wiedergegeben, um Gelegenheit zu geben, die Schrift und Graphikfähigkeit des Systems anzusehen.

Im Anschluß an den Artikel wurde noch die Tabelle 2 aufgenommen, welche das Forth Vocabulary des Machi zeigt, sowie Bild 8, welches die Aufmachung der Quellcodes im Editor des Mac zeigt. Nicht abgebildet sind aus Gründen der beschränkten Seitenzahl die übrigen Vocabularies, Tools und Utilities.

# MACH1

Ich finde, der MACINTOSH™ ist zurzeit der PC mit der besten Benutzeroberfläche . Zwei clevere Forthler aus Califonien haben in Zusammenarbeit mit einer Reihe von Studenten ( sorgfältiges Debugging! ) ein F-83 FORTH auf der 68000 Maschine implementiert das meiner Meinung nach einen echten Meilenstein darstellt.
Warum ?
Einem excellentes Verständnis der Vorteile und Eigenschaften des MAC und seiner leichten Bedienung
wurde das FORTH unterlegt. Es hat mit den früheren FORTH-Versionen für den MAC anderer Firmen soviel gemein wie mein alter KIM aus dem Jahre 1975 mit dem leicht zu handhabenden Rechner C-64 ( beide mit 6502 CPU ).

PASCAL und C-Freunde können neidisch werden wenn man vergleicht wie kinderleicht es ist in kurzer Zeit Programme mit Grafik , Multikasking oder Sprachausgabe zu erstellen.

Die Fehlermeldungen und eine offenbar im Hintergrund ablaufende Korrektur sorgt dafür daß man sich ziemliche Trotteligkeiten erlauben kann ohne daß das Programm abstürzt.

Der Name MACH1™ den Derrick Miley und Terry Noyes ihrem Produkt gegeben haben ist mehr als angebracht. Man fühlt sich plötzlich im Cockpit eines Jets und kann endlich einmal schnell und komfortabel alles das entwickeln und programmieren , was man schon immer mal erledigen wollte aber aus Zeitmangel nicht konnte.

Nach dem Einschieben der Diskette in den MAC erscheint folgendes Bild:

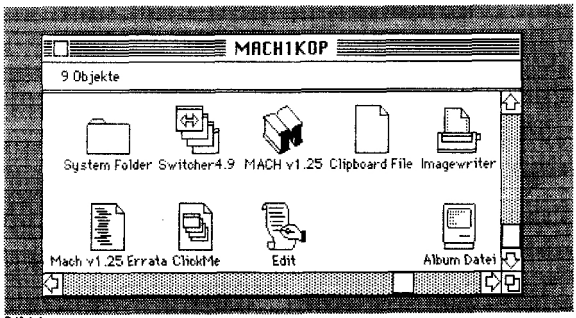



Bild 1

Das eigentliche FORTH-Programm wird geöffnet durch zweimaliges Anklicken des stilisierten M mit der Maus.

Das leere Blatt, welches jetzt erscheint ist sofort unser visuelles Interface zur FORTH-Maschine MACH1. Die Eingabe im interaktiven Modus kann sofort beginnen ( Bild 2).
MACH1 benutzt zunächst nur das FORTH-Vokabular .Benötigt man noch andere, zum Beispiel das Vokabular der Sprachausgabe genannt TALKING, so schreibt man an den Beginn des Programms ALSO < Name > .

```
ALSO TALKING
SAY HELLO AS" /HEHLOW2"
: GREETINGS 5 0 DO HELLO LOOP ;

GREETINGS <cr> OK <0>
```

Nach Eingabe des Wortes GREETINGS ertönt laut und deutlich fünfmal hintereinander ein englisch gesprochenes "Hello".

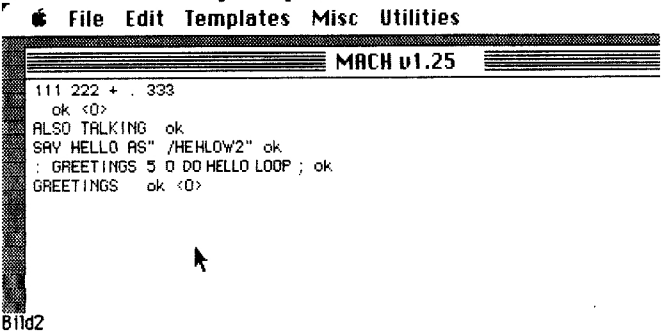



Bild2

Möchte man wissen , welche Worte im Vocabulary TALKING noch enthalten sind, so gibt man einfach
den Namen des Vocabular ein und klickt in der Menüleiste unter MISC den Begriff " WORDS" an. ( Bild 3):

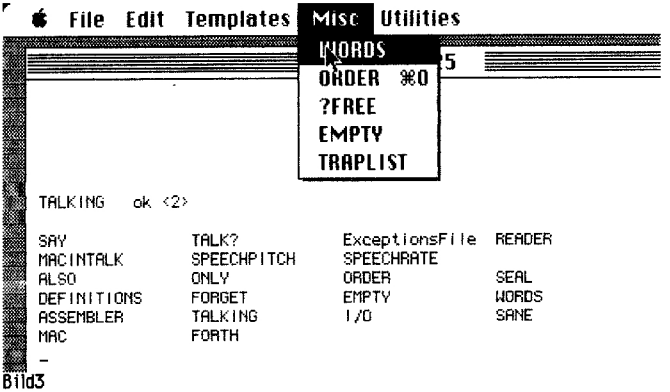



Bild3

Jedes FORTH-Wort ist - endlich mal ein gut und interessant geschriebenes dickes Manual - einzeln hervorragend erklärt.
Ich fand einzelne FORTH-Wörter so erfrischend klar dargestellt, daß es auch für den absoluten Einsteiger eine Freude sein wird, die (400 !) Seiten durchzulesen.

Glücklicherweise hat man davon Abstand genommen, eine vollkommen neue FORTH-Philosophie für den MAC zu entwickeln. Gute alte FIG-Forth Programme laufen sofort fehlerfrei.
Bei zwei alten Programmen, die ich aus Zeitschriften gedankenlos eintippte, hielt das Programm beim Ablaufen brav an, und ich konnte derart zwei Druckfehler im Urtext feststellen.

Als FORTH-Wörter lassen sich die mehreren hundert Utilities der MACINTOSH-Toolbox aufrufen in denen zum Beispiel phantastisch effektive graphische Bausteine zur Hand sind.
Insoweit ist MACH1 ganz sicherlich ein Programm, um in FORTH sehr gute Programme für den MACINTOSH™ zu erstellen.
Selbstverständlich kann man seine eigenen WINDOWS erstellen und nicht zuletzt habe ich hier erstmals einfach und problemslos das Problem des Multitasking gelöst gefunden. ( Allein der Abschnitt über Multitasking wäre es wert hier abgedruckt zu werden, so gut haben es die Entwickler auch diktaktisch verstanden ihr Produkt an den Mann zu bringen.)

Was die Geschwindigkeit anbetrifft so kann sich MACH1 mit den sehr viel weniger übersichtlichen
C-Compilern bestens vergleichen lassen:
1.
========== SIEVE ==========
Seconds

2.8 - ..... Assembly

6.8- ..... MegaMax C

7.5 - ..... Mach v1.25

13.0 - ..... Mach v1.0 - Mach v1.2

23.0 - ..... MacForth

24.0 - ..... Neon

---

MACH1 ist " subroutine-threaded" und erzeugt 68000 Maschinencode.
Für den Mac-Besitzer wichtig ist jedoch vor allem, daß man echte "stand alone applications" erzeugen kann, die kein gesondertes runtime modul brauchen und nur 16 k overhead haben.
Ein nicht zu übersehender Vorteil bei der Programmerstellung ist der Editiermodus des MACINTOSH der es gestattet das Programm als Textfile zu bearbeiten. ( Suche den langen Text durch, ändere das gesucht Wort um in das neue Wort usw.)

Auf komfortable Weise wird das Editieren ermöglich durch die Verwendung des sog. "Switchers".
Durch einen Druck mit der MAUS ist man im Editor, schreibt ändert sein Programm. Ein erneutes
Anklicken des Switchers und wir sind wieder in MACH1.

Als MULTI -Window Editor ausgelegt können gleichzeitig bis zu **vier** Text Files geöffnet sein zwischen denen man natürlich ebenfalls sehr komfortabel Textbausteine ( = FORTH Programme)
hin und her schieben kann.(Bild 4)

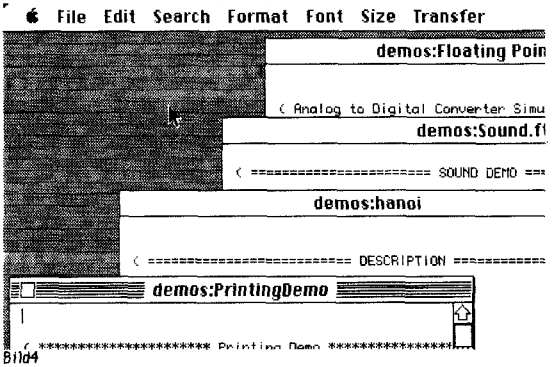


Bild4

Überhaupt eignet sich der mausgepflegte Bildschirm des MAC sehr gut um alle möglichen Pushbuttons für Steuerungen etc. zu erzeugen. Diese Pushbuttons können auf verschiedenen Windows zeigen von denen wiederum eine ganze Anzahl auf dem Bildschirm gleichzeitig geöffnet sein können. (Bild 5)
Die Implementierung des Floating Point Package ist m.E. sehr gut gelungen. Das Vokabular heißt SANE und ist die Abkürzung von Standard Apple Numeric Environment ,welches wiederum eine Obergruppe des IEEE 754 Binary Floating-Point Arithmetic Standards darstellt. Auch hier also wieder eine weise Beschränkung auf ausgefeilte, erprobte Dinge statt des Versuches das Rad neu zu erfinden. Die FP hat 80 Bit mit 64 Bit Mantisse entsprechend 19-20 Digits Genauigkeit.

Mitgeliefert wird als "DEMO" u.a. ein Terminalprogramm und ein Analog/Digitalwandler. Diese beiden Programme sind äußerst übersichtlich und prägnant , im Vergleich zu BASIC-Listings, die ich früher sah.

Ich habe MACH1 von der Firma Hardware, Software,Dokumentation
Alt Moabit 83 ,1000 Berlin 31, Tel. 030/ 3926669
Mein Eindruck ist, daß die Leute ihr Produkt kennen und auch gern am Telefon alle Fragen beantworten.

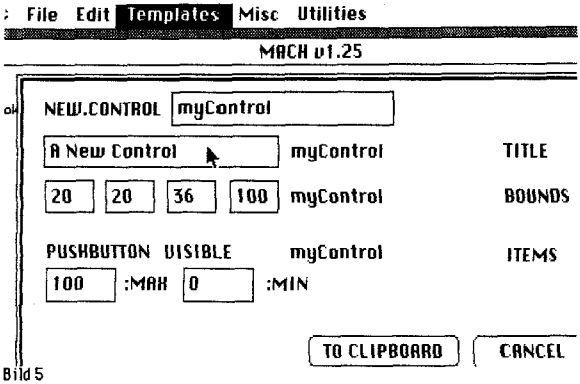



Bild 5

Bild 6 enthält das Vokabular zum A/D Wandler
Bild 7 enthält das Vokabular zum Terminal Programm (MAC-Toolbox-Worte)
Der Anhang enthält alle Words des FORTH selbst.

Zusammenfassung:

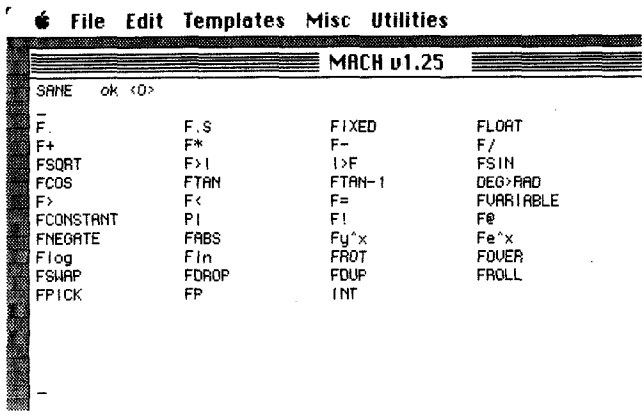
Mit MACH1 ist ein FORTH für den MACKINTOSH™ auf dem Markt, das die gute kreative Welt des MAC mit der guten kreativen Welt des FORTH vereinigt. Keine unübersichtlichen Screens mehr und für den Normalverbraucher soviel vorgefertigte Toolbox- Programme wie er nur will.
Im Grunde genommen sollte dieses FORTH die erste Sprache für den MAC sein.

Holger Blum

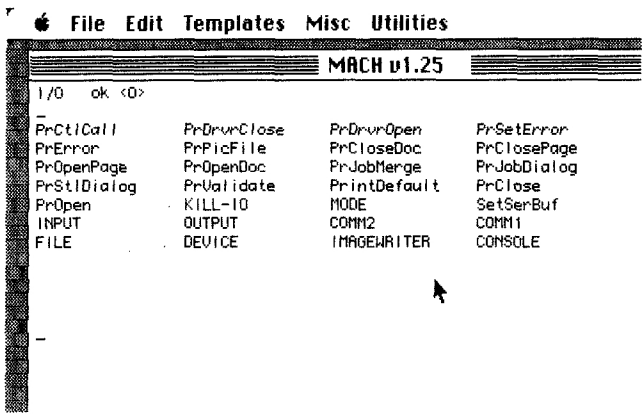
```
( make_sine_wave uses Floating Point to calculate an integer Look-Up table )

  : make_sine_wave
       60 0 DO                      ( make 60 different values )
              I 6 *
              I>F                   ( generate a number from 0 to 360 )
              DEG>RAD FSIN          ( evaluate the SINE, a number from 0 to 1 )
              1024. F*              ( scale the result by 1024 )
              F>I                   ( convert back into integer format )
              SINE_TABLE I 2* + W!     ( put data into table )
            LOOP ;

  make_sine_wave                  .( building sine wave table ) CR


  : TICKCOUNT ( - n )             ( returns the time since system startup )
       call tickcount ;


  : A/D ( - )
       TICKCOUNT
       60 MOD                     ( prints a number between 0 and 60.)
       2* SINE_TABLE + W@         ( index into the array in what appears )
       L_EXT                      ( to be a real time manner. )
       . CR ;

  INT                             ( return to integer format )

  ONLY  FORTH                     ( remove SANE from the search order )
```

BILD 8

Vokabular zum A/D Wandler (Bild 6)




Vokabular zum Terminal Programm ( Bild 7)



TAB.7

```
FORTH
-
TURNKEY         NEW-SEGMENT     WORKSPACE       ACTIVATE
BUILD           BACKGROUND      TERMINAL        OPERATOR
RELEASE         GET             UP              TIMER
COUNTER         PAUSE           QUIT            {
DEPTH           ABORT           ABORT"          ?TERMINAL
KEY             VARIABLE        CONSTANT        LAST
RECURSIVE       SMUDGE          GLOBAL          (
->              [COMPILE]       COMPILE         IMMEDIATE
[               ]               ;               :
CREATE          [']             '               :=
PAD             VERBOSE         STATE           HERE
NP              ?FREE           VOCABULARY      WAKE
SLEEP           #TIB            SPAN            BLK
TIB             BASE            STATUS          >IN
TASK->          USER            HEX             DECIMAL
BINARY          L_EXT           W_EXT           MIN
MAX             0>              0<              0=
>               <               =               <>
U<              OR              AND             XOR
NOT             R>              >R              2DROP
DROP            .S              ROLL            PICK
ROT             2OVER           OVER            2SWAP
SWAP            ?DUP            2DUP            DUP
NEGATE          ABS             UM/MOD          MOD
/MOD            2/              /               SQRT
*/MOD           */              2*              UM*
*               2-              2+              1-
1+              -               D<              DNEGATE
D+              +               CMOVE>          CMOVE
FILL            +!              !               ?
@               W!              W@              C!
C@              C,              W,              ,
LITERAL         EXIT            VALLOT          ALLOT
VERIFY          "               .(              ."
LEAVE           J               I'              R@
!               +LOOP           LOOP            DO
REPEAT          WHILE           UNTIL           AGAIN
BEGIN           ENDCASE         ENDOF           OF
CASE            THEN            ELSE            IF
-TRAILING       SPACES          SPACE           CR
.               .R              U.              DUMP
#S              #               SIGN            HOLD
#>              <#              ASCII           TYPE
COUNT           EMIT            QUERY           EXPECT
INTERPRET       NUMBER          DPL             CONVERT
EXECUTE         WORD            END-CODE        CODE
;CODE           DOES>           FIND            BODY>LINK
LINK>BODY       >BODY           BLOCK           BUFFER
INCLUDE"        LOAD            VIRTUAL         LIST
UPDATE          FLUSH           SAVE-BUFFERS    EMPTY-BUFFERS
DISK            FORTH-83                        ALSO
ONLY            ORDER           SEAL            DEFINITIONS
FORGET          EMPTY           WORDS           ASSEMBLER
TALKING         I/O             SANE            MAC
FORTH
-
```

# Defining Words, eine Einführung in die Anwendung

Konrad Scheller, Forchheim

Ich möchte heute eine Gruppe von Worten ansprechen, denen man bei der Einführung in Forth von Anfang an begegnet, die man in der Regel aber erst im Anschluß an die Grundübungen verstehen und schätzen lernt. Es sind dies die DEFINING WORDS. Ein Ausdruck, den man schwerlich übersetzen kann, ohne den Sinn zu entstellen. Es sind Worte, die Klassen von Worten definieren.

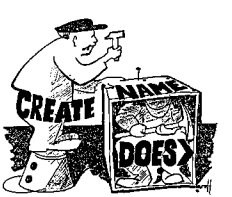
Einige solcher Definig Words werden wie selbstverständlich benutzt. Die Worte COLON und VARIABLE und CONSTANT sind solche Beispiele (Das Wort COLON wird meist kurz : geschrieben. Engl. colon = Doppelpunkt). Jedes durch sie erzeugte Wort zeigt ein jeweils typisches Verhalten. Alle COLON-Wörter führen die bis zum Semicolon aufgezählten Worte aus, alle Variablen legen ihre Parameterfeldadresse auf den Stack, und alle Konstanten legen den ihnen zugewiesenen Wert auf den Stack.

Die Defining Words haben zwei Teile, eine Compilieranweisung für die Laufzeit un eine Laufzeitanweisung für das Compilierte. Das Compilierverhalten bewegt sich z.B. um das Wort CREATE herum im ersten Teil des Defining Word und das Laufzeitverhalten wird bestimmt durch den DOES> Teil des Moduls. Doch lang ist der Weg der Theorie, kurz der des Beispiels.

*Beispiel:* Beim Commodore 64 gibt es die Speicherstelle 646, über die man die momentane Cursorfarbe bestimmt. Ändert man diese Stelle, ändert sich die Cursorfarbe bei der nächsten Zeichenausgabe. Im Code im Screen 1 wurden 16 Farbennamen definiert, mit deren Hilfe man den Cursor programmieren kann. Wie man sofort sieht, ist der einzige Unterschied zwischen SCHWARZ und WEISS der Name und der Parameter, der in die Adresse CURSORFARBE geschrieben wird. Das kann man vereinfachen.

In Screen 2 steht ein Stück Code, welches CREATE und DOES> benutzt, um das Gleichen einfacher zu erreichen. Im Wort FARBE erzeugt CREATE den Namen der Farbe im Dictionary und C, kompiliert dahinter in das Parameterfeld den Wert für die Farbe, der später wieder ausgelesen werden kann. Das Parameterfeld ist hier also nur ein Byte lang. Mit DOES> wird nun das Laufzeit-Verhalten des neugeschaffenen Wortes festgelegt. DOES> legt bei der Ausführung des neuen Wortes - z.B. PURPUR - die Parameterfeldadresse auf den Stack, also die Adresse, in der 4 gespeichert ist. Mit C@ wird nun von dort der Farbcode auf den Stack geholt und mit C! nach CURSORFARBE geschrieben. Alle Worte, die durch FARBE definiert werden, verhalten sich ebenso.

In Screen 3 gehen wir noch eine Stufe weiter und setzten unser Defining Word FARBE in die Schleife FARBEN ein. So werden dann automatisch die nächsten 16 Worte im INPUT STREAM des Quelltextes zu Farben und ihnen wird als Parameter der laufende Index der Schleife zugeteilt.




Defining Words sparen Speicherplatz, wenn viele Worte eines gleichen Datentypes gebraucht werden. So wird die Sequenz C@ CURSORFARBE C! nur einmal abgespeichert und praktisch wie eine Routine von allen Farben aufgerufen. Der DOES> Teil kann beliebig komplex sein.

(Die hier gezeigte Methode ist die des Forth-83-Standards und zum Beispiel in den public domain Forths F83 oder VolksForth anzutreffen. Das Konzept der Defining Words im figForth ist etwas anders. Hier wird der erste Teil der Definition mit BUILDS> eingeleitet. Wer damit Probleme hat, wende sich an die nächstgelegene lokale Forthgruppe oder an die Redaktion.)

```
SCREEN # 1
 0) // DEFINING WORDS DEMO                                    KSCH-MAY86
 1) DECIMAL
 2)
 3) 646 CONSTANT CURSORFARBE
 4)
 5) : SCHWARZ   0 CURSORFARBE C! ;   : ORANGE     8 CURSORFARBE C! ;
 6) : WEISS     1 CURSORFARBE C! ;   : BRAUN      9 CURSORFARBE C! ;
 7) : ROT       2 CURSORFARBE C! ;   : ROSA      10 CURSORFARBE C! ;
 8) : CYAN      3 CURSORFARBE C! ;   : GRAU1     11 CURSORFARBE C! ;
 9) : PURPUR    4 CURSORFARBE C! ;   : GRAU2     12 CURSORFARBE C! ;
10) : GRUEN     5 CURSORFARBE C! ;   : H-GRUEN   13 CURSORFARBE C! ;
11) : BLAU      6 CURSORFARBE C! ;   : H-BLAU    14 CURSORFARBE C! ;
12) : GELB      7 CURSORFARBE C! ;   : GRAU3     15 CURSORFARBE C! ;
13)
14)
15)

SCREEN # 2
 0) // DEFINING WORDS DEMO                                    KSCH-MAY86
 1)
 2) 646 CONSTANT CURSORFARBE
 3)
 4) : FARBE      ( N -- )    CREATE C,
 5)   DOES>      ( -- )      C@   CURSORFARBE C! ;
 6)
 7)  0 FARBE SCHWARZ     1 FARBE WEISS       2 FARBE ROT
 8)  3 FARBE CYAN        4 FARBE PURPUR      5 FARBE GRUEN
 9)  6 FARBE BLAU        7 FARBE GELB        8 FARBE ORANGE
10)  9 FARBE BRAUN      10 FARBE ROSA       11 FARBE GRAU1
11) 12 FARBE GRAU2      13 FARBE HELLGRUEN  14 FARBE HELLBLAU
12) 15 FARBE GRAU3
13)
14)
15)

SCREEN # 3
 0) // DEFINING WORDS DEMO
 1)
 2) 646 CONSTANT CURSORFARBE
 3)
 4) : FARBE   ( n -- ) create c,
 5)    Does>  ( -- )   c§ cursorfarbe c! ;
 6) : FARBEN  ( n -- ) 0 do  I farbe  loop ;
 7)
 8) 16 Farben
 9)    SCHWARZ WEISS ROT CYAN
10)    PURPUR GRUEN BLAU GELB
11)    ORANGE BRAUN ROSA GRAU1
12)    GRAU2 HELLGRUEN HELLBLAU GRAU3
13)
14)
15)
```

* * *

Buchbesprechung

Rüdiger Nicolovius, Graphik mit GKS
Hanser Verlag, 1986, ISBN 3-446-14557-5

Die Computergraphik hat in den letzten Jahren eine große Verbreitung erlangt. Damit ist aber auch ein Problem deutlicher als je zuvor offen zutage getreten: die fehlende Kompatibilität der Graphik-Software. Die Graphik-Fähigkeiten der verschiedenen Rechnertypen basieren z.T. sehr unter-

schiedlichen Hardware-Konzepten. Da gibt es z.B. Plotter, Printer und Bildschirmgeräte, mit Vektor- oder Rasterdarstellung, ein- oder mehrfarbig, mit oder ohne Eingabemöglichkeit usw. Ebenso unterschiedlich sind die Anwendungen der Computergraphik. Sie reichen von einfachen Balkendiagrammen bis hin zu realistischen Landschaftsdarstellungen in Echtzeit für Flugsimulatoren u.ä. Entsprechend dieser Bandbreite haben sich die unterschiedlichsten *Graphiksoftware-Konzepte* entwickelt mit dem Ergebnis, daß Graphiksoftware bisher als sehr schwer portierbar galt. Es gibt zwar schon seit längerem sog. de-Facto-Standards wie Calcomp oder Plot-10, doch diese sind meist auf einen sehr begrenzten Bereich beschränkt. Aus dem Bedarf nach einem universellen, *Hardware-unabhängigen Graphikstandard* entstand das, im wesentlichen an der TU Darmstadt entwickelte, Graphische Kern System (GKS). GKS ist zwar nicht der einzige 'große' Standard (es gibt z.B. noch den ACM-Core und PHIGS), doch nach der internationalen Normung durch die ISO erfreut sich GKS nunmehr einer steigenden Beliebtheit. Sogar *auf PC's sind bereits Implementationen* realisiert.

Nun zum Buch von R.Nicolovius. Es entstand anschliessend an eine Veranstaltungsreihe hier an der Uni Hamburg, wo in kleinen Gruppen eine Einführung in GKS gegeben wurde. Dennoch ist das Buch kein Vorlesungsskript, sondern ein Lehrbuch, das mit seiner praxisorientierten Herangehensweise eine Lücke in der GKS-Literatur schließt. Der Leser wird Schritt für Schritt mit den Konzepten von GKS vertraut gemacht und ist, durch die ihn von Anfang an begleitenden zahlreichen Programm-Beispiele, schnell in der Lage, eigene Programme zu schreiben. Und das dürfte für die meisten Leser das erstrebte Ziel sein. Für die Programm-Beispiele wurde FORTRAN gewählt, was zwar verständlich erscheint, wenn man bedenkt, daß FORTRAN die meistverwendete Sprache für Graphik-Implementationen ist und auch die ersten GKS-Implementationen zuerst in FORTRAN realisiert wurden. Doch didaktisch ist es zumindest aufgrund der kryptischen Funktionsnamen problematisch (Bsp.: GQLVkS steht *für Gks inQuire leVel of* gKS). Insgesamt möchte ich das Buch als gelungen bezeichnen und es allen empfehlen, die sich mit Programmierung unter GKS beschäftigen oder sich nur einen verständlichen, praxisnahen Einblick in den gegenwärtigen Stand(ard) der Computergraphik verschaffen möchten, ohne sich *für fortgeschrittene Themen wie 3D oder Animation* zu interessieren. Schließlich macht ein recht ausführlicher Anhang mit tabellarischer Auflistung der GKS-Funktionen das Buch auch noch als Nachschlagewerk nutzbar.

Was bedeutet GKS nun für den Forthler? Angesichts der großen Probleme, die Forth mit Standards hat und der *forth-untypischen Philosophie von GKS* des alles-in-einem, wird GKS sicher nicht der zukünftige Graphik-Standard für Forth sein. Der Drang zur Standardisierung ist jedoch auch in Forth nicht zu übersehen und einige Forthler haben sich bereits Gedanken über Standard-Graphik-Worte gemacht, ohne daß dieses publik wurde. *Hier wird GKS*, ebenso wie Pascal oder FORTRAN, vom 'unwissenden Rest der Computerei' als Vergleich herangezogen werden, und wenn Forthler nicht nur ein Dasein als esoterische Randgruppe fristen wollen, so müssen sie sich solchen Vergleichen stellen. Abgesehen davon wäre ein 'kleines GKS' (Level 0a) auch in Forth sicherlich eine Implementation wert.

* * *

## FORTHQUELLEN

### Zeichensatz-Generator für C64

Andreas Carl, Berlin

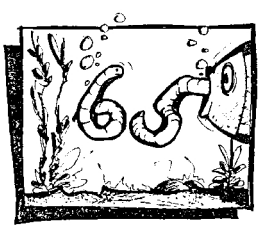
Der Ort des Zeichensatzes wird über Bit3, Bit2 und Bit1 des VIC-Kontrollregisters 53272 eingegeben. Um den Ort zu ändern kann folgender Basic-poke benutzt werden:
POKE53272,(PEEK(53272)AND240)OR A
wobei A einer der Werte aus Tab1 ist.
Für A=4 und A=6 wird der Zeichensatz aus dem ROM genommen!

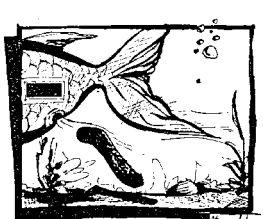
```
A=0   RAM 0000-2047
A=2   RAM 2048-4095
A=4   RAM 4096-6143
 .        .
 .        .
 .        .

tab1
```

Ich definiere zunächst zwei Worte, STANDARD und OWN, um zwischen dem normalen und einem eigenen Zeichensatz umschalten zu können. Der eigene Zeichensatz soll im 4K langen Speicherbereich ab 8192 stehen (A=8). CSET kopiert den ROM-Zeichensatz in diesen Bereich. Um diesen editieren zu können, benötige ich ein Eingabefeld MASK auf dem Bildschirm. Hierin wird in der linken Hälfte das zu editierende Zeichen im Großformat angezeigt, während rechts das neue Bitmuster eingetragen werden kann. Zu diesem Zweck erzeugt .BYTE aus einem Byte eine Folge von Sternchen und Leerzeichen, während GETBYTE eine solche Folge aus dem rechten Bildschirmfeld in die Variable BYTE ablegt. Das Wort CH@ holt nun ein ganzes Zeichen aus dem RAM in das linke Bildschirmfeld, indem es achtmal .BYTE aufruft. Umgekehrt wandelt CH! das Muster im rechten Bildfeld in ein Bitmuster um und speichert es in den Zeichensatz ein, indem es achtmal GETBYTE aufruft. CGEN ist der fertige Zeichensatz-Generator. CGEN nimmt den Wert des zu bearbeitenden Zeichens vom Stack, baut die Maske auf und ediert in BEGIN ... UNTIL das Zeichen neu bis zu einem abschließenden <RETURN>, um dann das Zeichen mit CH! abzuspeichern.

```
#SCR 01

 0       ( CHARACTERS )
 1
 2 : OWN      53272 C@ 240 AND 8 OR 53272 C! ;
 3 : STANDARD 53272 C@ 240 AND 4 OR 53272 C! ;
 4
 5 : CROM 56334 C@ 254 AND 56334 C! 1 C@ 251 AND 1 C! ;
 6 : CRAM 1 C@ 4 OR 1 C! 56334 C@ 1 OR 56334 C! ;
 7
 8 : CSET CROM 53248 8192 4096 CMOVE CRAM ;
 9
10 : M1 ."      ++++++++++  ++++++++++" CR ;
11 : M2 8 0 DO ."      +          +" CR LOOP ;
12 : MASK 147 EMIT CR 5 SPACES 18 EMIT ." CHARACTER-GENERATOR"
13       146 EMIT ." :" CR CR CR CR ."      OLD          NEW"
14       CR CR M1 M2 M1 CR ;
15                                  -->
```

```
#SCR 02

 0       ( CONT. )
 1   0 VARIABLE BYTE   1363 CONSTANT UR
 2
 3 : SETOLD 19 EMIT 8 0 DO CR LOOP 6  0 DO 29 EMIT LOOP ;
 4 : SETNEW 19 EMIT 9 0 DO CR LOOP 20 0 DO 29 EMIT LOOP ;
 5
 6 : 2* 1 SWAP -1 DO 2 * LOOP 2 / ;
 7 : .BYTE   8 0 DO 2 * DUP 256 AND IF ." *" ELSE ."  " ENDIF LOOP
 8         CR 6 0 DO 29 EMIT LOOP DROP ;
 9 : GETBYTE 0 BYTE ! 8 0 DO DUP I + C@ 32 = 0= IF 7 I - 2* BYTE +!
10         ENDIF LOOP DROP ;
11 : CH@ 8 * 8192 + 8 0 DO DUP I + C@ .BYTE LOOP DROP ;
12 : CH! 8 * 8192 + 8 0 DO UR I 40 * + GETBYTE
13                      BYTE @ OVER I + C! LOOP DROP ;
14 : CGEN MASK DUP 1091 C! SETOLD DUP CH@ SETNEW
15       BEGIN KEY DUP EMIT 13 = UNTIL CH! ;
```





# Performance Analysis in Threaded Code Systems

Michael Perry

## Abstract

The importance of performance analysis to the iterative design process is discussed. Several techniques for performance analysis in Forth systems are described. Some related debugging techniques are mentioned.

## Iterative Design

Forth encourages the use of iterative design methodology, which involves the rapid repetition of a short design/implement/test cycle. Most Forth systems provide *reasonably good* tools for finding bugs in code and flaws in algorithms. Few systems provide adequate tools for diagnosing performance problems. A good rule of thumb is that a program spends ninety percent of its time executing ten percent of the code. When some performance goal must be met it is neccessary to find those routines where most time is spent and make them run faster. Forth encourages modular programming, and it is easy to replace a slow routine once it has *been found*.

## Memory Allocation

One requirement which many of these tools share is the allocation of memory for storing the results, usually a counter for each word in the dictionary. A very useful technique is available on systems which have a view field in the header of the dictionary entry for each word. The view field points to the source code block on disk. Once having used a system with the ability to automatically locate the source for any word, it is difficult to go back to a system which lacks it.

The view field can be borrowed for other uses, such as the counter needed for these tools. When the view field is not available, an array with one entry per word in the dictionary is required. For compatability, the word >VIEW should convert the address of a code field into the address of the corresponding cell This kind of code is very dependent on the particular system and CPU being used. Examples later in the paper are for the MC68000 running F83, a public-domain, 83-standard Forth system.

## Monte Carlo Analysis

One simple tool which gives a statistical indication of the frequency of execution of a word uses a real time clock interrupt routine which periodically samples the interpreter pointer (IP) or program counter (PC). This can be used in a couple of different ways, either by incrementing a counter for the word being executed, or by building a histogram of IP or PC activity.

There are several ways to decide which counter to increment. Assuming a post-incrementing, indirect-threaded system, IP points to a pointer to the code field of the next word to execute. It would be convenient, but wrong, to increment that word's counter. It may or may not be a slow routine; after all, it is not even being executed at the moment. To increment the counter for the word into which the IP points is better (and slower), but still incorrect, because that word merely called the word being executed.

It is best to increment the counter for the word currently executing, which is pointed to by the cell before the one pointed to by the IP.

```
LABEL BUMP   ( increment counter of word being executed. )
   -2 IP D) A0 MOVE   A0 ) A0 MOVE   TO-VIEW #) JSR   1 A1 ) ADDQ   RTE
( TO-VIEW converts a code field address in A0 to a view field address in A1 )
```

To build a histogram, an array is allocated whose size depends on the desired resolution. Two variables contain the upper and lower limits of the range of addresses of interest. If the IP (or PC, as the case may be) is inside the window of interest, the cell corresponding to its relative position is incremented. The results are displayed as a histogram, and can give an idea of where most time is spent. After each pass the window can be made narrower to get more precise results, but more time will be required to get the same number of counts inside the window.

## Static Frequency Analysis

Another method is static frequency analysis. A counter is associated with each word in the dictionary, and they are initially set to zero. Each time the word is encountered by the compiler its count is incremented. After compiling, a table of counts is printed. While this does not show the relative execution times or frequency of execution, it at least indicates which words may deserve some attention.

Implementation requires writing a new compiler loop which finds each word and increments its count. Immediate words such as IF present a problem. The best solution is probably to redefine them to increment the counter for any words which they compile. Comments and words which terminate compilation must be executed as usual while compiling.

```
: COMPILER   ( -- )   setup
   BEGIN  BL WORD FIND DUP 0>
     IF  DROP EXECUTE
     ELSE  IF  1 OVER >VIEW +!  ,  ELSE  DROP  THEN
     THEN  finished?
   UNTIL  ;

: IF   ( -- adr flag )
   COMPILE ?BRANCH  >MARK   1 ['] ?BRANCH >VIEW +!  ; IMMEDIATE
```

## Dynamic Frequency Analysis

In some systems the address interpreter ( NEXT ) is a single routine shared by all code words. In other systems it is distributed, with each word having its own copy. A distributed NEXT is usually faster, but prevents the use of some interesting techniques which involve replacing NEXT with a routine which does additional work.

In dynamic frequency analysis, NEXT is replaced by a routine which increments the counter for the word which is about to be executed, in addition to performing the usual NEXT function.

```
LABEL BUMP-NEXT
   IP )+ A0 MOVE   TO-VIEW #) JSR   1 A1 ) ADDQ   A0 ) A0 MOVE   A0 ) JMP
( Note the similarity to the BUMP routine given earlier. )
```

As before, all counters are zeroed before running the code to be analysed, and a table of counts is printed afterwards. This technique still does not show which routines take the longest to execute, but it does show which are executed most frequently, and that is often a good indication of where time is being spent.

Another limitation is that real time code will almost certainly fail to be fast enough while running the test, and this may change the behavior of the system enough to render the results meaningless. In such time critical cases, only hardware tools will accurately reflect the operation of the system.

## Assembler Cycle Count Generation

An assembler in a Forth system is relatively simple. Ordinarily it provides a minimum of error checking, and can be defined in from three to fifteen screens or so. It is possible to add features such as error checking to the assembler. For performance analysis, information about instruction and addressing mode timing can be added to the assembler. Whenever an instruction is assembled, a *cycle counter is incremented by the duration* of the instruction and its addressing mode or modes. In many cases the actual execution time will differ because of pipelining, variable execution times, or inaccurate data in the manual. However, it is usually possible to get a good idea of the worst case *time, and that is normally* the best guide to performance.

The execution time for a high level definition can be calculated as the sum of the times of the components, plus the execution time of NEXT times the number *of components*, plus the entry and exit times. For this calculation to be possible, the timing information for every word must be available. This can be built in to the dictionary entry for each word by the meta-compiler; it is more difficult if the Forth system's source is in assembly language. When target compiling small applications the counts can be kept just in the host system.

One significant limitation of this technique is that most words contain some control structures. In the case of IF ELSE THEN structures, the times of the two branches can be calculated and the larger value used. For DO LOOP and BEGIN WHILE REPEAT or BEGIN UNTIL structures the situation is much more difficult. The structure could be assigned a duration equal to one pass through its body, but this is not likely to be correct.

Fortunately those words on which performance usually depends are those buried deep inside loops of one kind or another, and so it is not so neccessary to analyse higher level words accurately.

## Debugging

As mentioned before, several debugging tools can be built on special versions of NEXT. The simplest is a breakpoint. In NEXT the IP is compared to the contents of a variable which contains the address at which a breakpoint has been set. If they are the same, execution is aborted or a breakpoint handler is executed. An array of addresses can be used instead of a single variable if multiple breakpoints are desired.

A debugger can use a NEXT which tests the IP against a pair of addresses, and executes a trace routine if it is between them. Another method is to execute trace whenever the return stack depth has some value, but that approach is a bit trickier to use.

A related tool protects the contents of an address. NEXT compares the present contents of the protected address to a saved value, and enters the debugger if they differ.

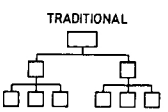
```
CREATE SAVED 4 ALLOT   ( address, value )
LABEL PROTECTING-NEXT
   SAVED #) A0 LEA   A0 )+ A1 LMOVE   A0 ) D0 MOVE
   A1 ) D0 CMP  0<> IF   TRACING #) JMP   THEN   ( fall through )
LABEL NORMAL-NEXT
   IP )+ A0 MOVE   A0 ) A0 MOVE   A0 ) JMP
( TRACING sets the debugger window to point to the word which caused the )
( change, and NORMAL-NEXT is a copy of the normal NEXT code. )
LABEL PROTECTING
   PROTECTING-NEXT #) JMP
CODE PATCH-NEXT   ( -- )
   PROTECTING #) )NEXT #) LONG MOVE   NEXT  END-CODE
( installs a jump to the new NEXT on top of the old NEXT )
: PROTECT   ( address -- )
   DUP @ SWAP SAVED 2!   PATCH-NEXT  ;
```

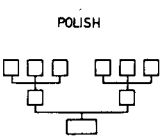
## Conclusions

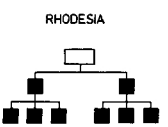
Many powerful tools for debugging and analysis are possible in a threaded-code system. The flexible and open nature of these systems allow interesting dynamic modifications to their structure and behavior. Continued exploration of the possiblilities will doubtless prove rewarding.

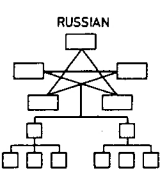
## Bibliography


Laxen, H.:  Debugging Techniques
   Forth Dimensions, v.6 #2 and #3
McClees, H.:  A Functional Usage Analyser
   Rochester Forth Conference Proceedings, 1983
Russell, J. & Solntseff, N.:  Break Point Utility
   Rochester Forth Conference Proceedings, 1984
Spreier, P.:  Performance Monitoring in Forth
   FORML Conference Proceedings, 1981
Whitney, A. & Conrad, M.:  Call Forth for Realtime Control Programming
   Computer Design, v.22 #5

# Queues in Forth

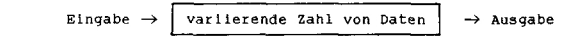
Bernd Pennemann, Hamburg

*In dem an anderer Stelle beschriebenen Terminalprogramm [1] wurde die Datenstruktur QUEUE benutzt. Sie soll hier getrennt beschrieben werden, da sie auch in anderen Anwendungen einsetzbar ist.*

In diesem Artikel wird anhand eines Beispiels gezeigt, wann Queues eingesetzt werden sollten. Anschließend werden ihre Eigenschaften beschrieben. Eine Implementation von Queues wird vorgestellt und ihre Funktionsweise durch Beispiele dargestellt. Zum Schluß folgt ein Listing der Implementation.

Queues (oder auch first in first out memories, abgekürzt FIFOs) werden immer dann benötigt, wenn zwei Prozesse miteinander Daten austauschen müssen, aber nicht synchron ablaufen.
Zum Beispiel können Drucker oft nicht gleichzeitig Daten empfangen und drucken. Der Drucker nimmt dann eine Zeile Text mit sehr hoher Geschwindigkeit auf und druckt anschließend lange aus. Er kann die Daten also nur stoßweise aufnehmen. Eine serielle Schnittstelle kann dagegen kontinuierlich Zeichen empfangen. Würde man in einem solchen Fall Schnittstelle und Drucker einfach zusammenschalten, so wäre die Datenübertragung sehr langsam. Erst würde eine Zeile von der Schnittstelle empfangen, während der Drucker den größten Teil der Zeit wartet und anschließend druckt der Drucker, während die Schnittstelle (bzw. der Sender, der an sie Daten übertragen will) darauf wartet, daß der Drucker wieder Daten aufnehmen kann.
Eine wesentliche Verbesserung dieser Situation ließe sich durch eine Queue erreichen.
Ein anderes Beispiel für die Verwendung von Queues findet man in dem vom Autor veröffentlichten Terminalprogramm. Dort dient eine Queue zur Zwischenspeicherung von durch eine serielle Schnittstelle empfangenen Zeichen, die erst später verarbeitet werden können.

Eine Queue ist ein Speicher variabler Länge, in den die Daten an der einen Seite hineingeschoben und an der anderen herausgeholt werden:




Eine Queue kann asynchron geschrieben und gelesen werden. Man kann z.B. Daten stoßweise eingeben und kontinuierlich auslesen, oder, wie in dem obigen Beispiel erforderlich, Daten kontinuierlich eingeben und stoßweise auslesen. Natürlich kann man nicht mehr Daten auslesen, als vorher hineingeschrieben wurden.
Die Menge der Daten, die eine Queue enthält, kann also wie beim Stack des Forth-Systems variieren. Jedoch werden die Daten anders als beim Stack immer in der Reihenfolge ausgelesen, wie sie geschrieben wurden. Das zuerst geschriebene Zeichen wird also auch zuerst ausgelesen (beim Stack wird das zuletzt ausgegebene Zeichen zuerst ausgelesen).

Meine Realisierung einer Queue in Forth besteht aus den folgenden 6 Worten:

```
CREATEQUEUE          ( len --)
```
Ein definierendes Wort, das eine Queue erzeugt, die zunächst leer ist. Die maximale Menge an Daten (die Länge), die die Queue speichern kann, ist len-1 Bytes. Wird die so erzeugte Queue später ausgeführt, macht sie sich zur aktuellen Queue.

```
QUEUE                ( -- adr)
```
Eine Variable, die auf die aktuelle Queue zeigt.

```
QC!                  ( 8b--)
```
Schreibt 8b in die aktuelle Queue. Eine Fehlerbedingung besteht, falls die maximale Länge der Queue dadurch überschritten wird.

```
QC@                  ( --8b)
```
Liest 8b aus der aktuellen Queue aus. Eine Fehlerbedingung besteht, falls die Queue keine Daten enthält.

```
QFULL?               ( --f)
```
Das Flag f ist wahr, falls keine weiteren Daten in der aktuellen Queue gespeichert werden dürfen. D.h. die Queue hat ihre maximale Länge erreicht.

```
QEMPTY?              ( --f)
```
Das Flag f ist wahr, falls die aktuelle Queue keine Daten enthält. D.h. es wurden genau so viele Daten ausgelesen wie hineingeschrieben.

```
QRESET               ( --)
```
entleert die aktuelle Queue.

```
NEXT                      ( adr -- adr')
```
adr ist die Adresse eines Datums in der aktuellen Queue. adr' ist dann die Adresse des folgenden Datums.

```
ADVANCE                   ( adr -- )
```
Der Zeiger, der sich in der Adresse adr befindet, wird um ein Datum weitergezählt.

Die Worte QC@ und QC! führen keine überprüfung durch, ob die Queue voll oder leer ist, da es prinzipiell zwei Möglichkeiten gibt, beim Lesen auf eine leere Queue zu reagieren. Die erste besteht darin, eine Fehlermeldung auszugeben und das Programm abzubrechen. Bei der anderen wird der lesende Prozeß solange verzögert, bis die Queue wieder Daten enthält. Welche der beiden Möglichkeiten in einer Anwendung die richtige ist, weiß man natürlich im voraus nicht. Bei dem oben angeführten Beispiel von Drucker und serieller Schnittstelle würde man die zweite Möglichkeit wählen, da der Drucker im Mittel schneller als die serielle Schnittstelle ist.

Ähnliche Überlegungen gelten natürlich auch für den Fall, daß Daten in eine volle Queue geschrieben werden sollen.

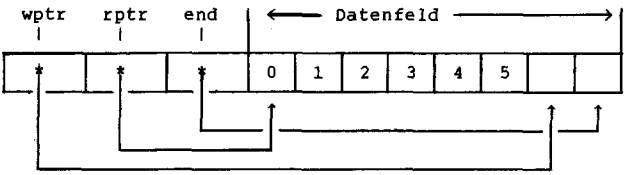
Eine Queue besteht aus einem Forth-Wort, dessen Parameterfeld in vier Felder unterteilt ist. Diese vier Felder enthalten den Schreibzeiger, den Lesezeiger, einen Zeiger auf das Ende der Queue und das Feld, daß die Daten enthält. Für der Zugriff auf diese

Felder habe ich die primitiven Operatoren NEXT und ADVANCE eingeführt.

Wozu die Zeiger benötigt werden und wie die Queue funktioniert, will ich im folgenden Beispiel erklären. Dazu geben wir folgendes ein:

```
8 Createqueue testqueue
: beispiel     testqueue  6 0  DO  I Ascii 0 + qc!  LOOP ;
beispiel
```

Das Parameterfeld des Wortes TESTQUEUE sähe dann so aus:




Der Schreibzeiger wurde hier mit WPTR bezeichnet, der Lesezeiger mit RPTR und der Endzeiger mit END. Der Inhalt von RPTR zeigt also auf den Anfang des Datenfeldes, während WPTR auf das siebte Zeichen zeigt, also auf die Speicherstelle, die als nächstes beschrieben wird. Da der Zeiger END sich nicht ändert, lasse ich ihn im folgenden weg, um das Bild zu vereinfachen.

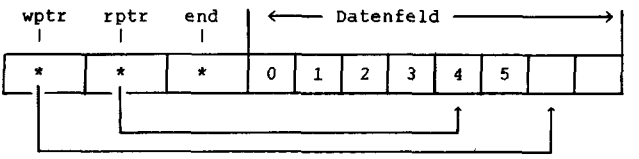
Tippt man ein :

```
: read     4 0 DO qc@ emit LOOP ;
read
```

so erhält man die Ausgabe :

```
     0123
```

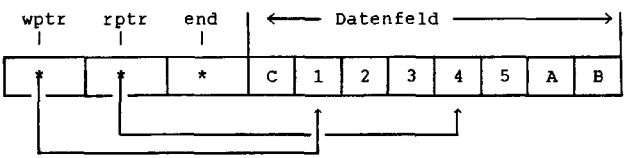
und das Parameterfeld sieht so aus :




Jetzt zeigt RPTR auf das fünfte Zeichen, das als nächstes gelesen wird.

Schließlich wollen wir noch den Fall betrachten, daß noch drei Bytes in der Queue gespeichert werden :

```
  Ascii A qc!  Ascii B qc!  Ascii C qc!
```




Wenn das Ende des Speicherbereichs erreicht wird, springen die Schreib- und Lesezeiger wieder an den Anfang des Bereichs zurück. Der Speicherbereich wird also zyklisch durchlaufen. Für das zurückspringen ist das Wort NEXT zuständig. Da es oft benutzt wird, sollte es unbedingt in Code geschrieben werden !

Zum Schluß folgt noch das Listing, wobei UNDER wie folgt definiert ist :

```
: UNDER    ( 16b1 16b2 -- 16b2 16b1 16b2 )
             swap over  ;
```

```
0 \ Create Queue for storing characters                  bp 16Jun86 Auf diesem Screen wird eine Queue-Struktur defini    bp 16Jun86
1
2 Variable queue                  \ this Variable points to Queue  QUEUE ist eine Variable, die auf die aktuelle Queue zeigt
3 : Createqueue   ( len -- )                                         In Multitasking-Systemen sollte das eine Uservariable sein.
4     Create  here 6 + dup , dup , over + 1- , allot                 CREATEQUEUE erzeugt eine Queue der Länge len. Wird der Name
5     Does>  queue ! ;                                                 der neuen Queue genannt, macht sie sich zu aktuellen Queue.
6
7 : next          ( adr -- adr' )   \ adr is pointer in data field NEXT     adr ist die Adresse eines Bytes in der Queue. Adr' ist
8     1+  queue @ 4 + @  over u<  IF drop queue @ 6 + THEN ;                 die Adresse des folgenden Bytes
9 : advance       ( adr-- ) dup @ next  swap ! ;                 ADVANCE   Der Zeiger adr wird ein Byte weitergestellt.
A
B : qempty?       ( -- f)   queue @  dup  @ next  swap  2+ @  = ;  QEMPTY?  f ist wahr, falls die Queue leer ist.
C : qfull?        ( -- f)   queue @  dup  @       swap  2+ @  = ;  QFULL?   f ist wahr, falls die Queue voll ist.
D : qc@           ( -- c)   queue @ 2+  dup @ c@  swap advance ;   QC@      das Zeichen c wird aus der Queue gelesen...
E : qc!           ( c --)   queue @   under @ c!       advance ;   QC!      das Zeichen c wird in die Queue geschrieben...
F : qreset        ( --)     queue @  dup 2+ @ swap ! ;             QRESET    der Inhalt der Queue wird gelöscht
```

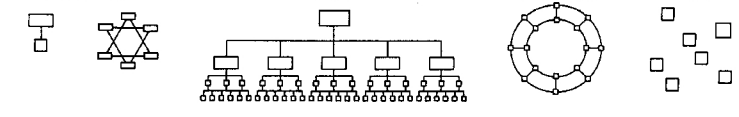
[1] B. Pennemann, "Ein Terminalprogramm in Forth" , Vierte Dimension II,2 (1986)

Bemerkung zum Artikel "Queues in Forth"

Zwei Namen in der Queue-Struktur finde ich unglücklich gewählt. Es sind dies QRESET und NEXT.

Das Wort, welches prüft, ob der Ringspeicher leer ist, wurde forth-englisch treffend QEMPTY? genannt. Das Wort, welches diesen Ringspeicher entleert, könnte dann QEMPTY genannt werden, und nicht QRESET. Die Namen der Worte sollten - wann immer es geht - signalisieren, was sie bewirken und nicht, wie sie das machen. So ist eben CLEARSTACK gegenüber SP0! auf Anhieb klar.

Der Name NEXT ist im Forth eigentlich schon fest vergeben. So heißt die zentrale Routine jeder virtuellen Forthmaschine. NEXT wird auch Innerer Interpreter genannt. Die hier bezeichnete Routine incrementiert die Adressen der Queue. Wie wäre es also mit QINC als Name für diese Routine?
Michael Kalus



FORTH GRUPPEN

Hamburg
  Treffen jeden vierten Samstag im Monat ab 16:00Uhr in der Berufsfachschule für Radio- und Fernsehtechnik, Eimsbüttelerstr.64-66.
  Kontakt: Bernd Pennemann, 040-6900539
Karsruhe
  Treffen jeden dritten Mittwoch im Monat ab 19:00Uhr im Jugend- und Begegnungszentrum, Krohnenplatz.
  Kontakt: Michael Weiss, 0721-854994
Paderborn
  Treffen in der Gruningerstr.20 nach Verabredung.
  Kontakt: Thomas Asche, 05251-26496
Wuppertal
  Treffen jeden vierten Freitag im Monat ab 20:00Uhr im Bahnhof Ottenbruch, Funckstrasse, W'tal-Elberfeld.
  Kontakt: Michael Kalus, Präsidentenstr.40, 5830 Schwelm, 02336-82204

Graphik und Animation
  Kontakt: Marco Pauck, 040-3900139

Berlin (*)
  Kontakt: Hans Madlung, 030-4141831
Braunschweiger Raum (*)
  Kontakt: Eckhard Heyne, 05352-58087
Darmstadt (*)
  Kursus in der Volkshochschule. Kontakt: Andreas Soeder, 06257-2744
Freiburger Raum (*)
  Kontakt: Markus Gimbel, 07641 - 42819
Moers (*)
  Kontakt: Hans Chrapia, 0203-3793274
Münchner Raum (*)
  Kontakt-1: Heinz Schnitter, privat 089-3103385, Labor 089-32095276.
  Kontakt-2: Jürgen Helbig und Robert Schörghuber, 089-2283531
  Kontakt-2: Ekkehard Flögel, 08021-8414
Rhein/Nekar Raum (*)
  Thomas Prinz, priv 06271-2830, beruflich 06221-701687

Villach östereich (*)
  Forth-Club in Kärnten. Treffen nach Verabredung. Kontakt: Heinz Klambauer, 04242-33566

Belgien FIG Chapter
  Kontakt: Luk van Loock, Lariksdreff 20, 2120 Schoten, Telefon 03-658-6343

Holland FIG Chapter
  Kontakt: Adriaan van Roosmalen, Heusden Houtsestraat 134, 4817 We Breda, Telefon 31-76-713104

Schweiz FIG Chapter
  Kontakt: Max Hugelshofer, ERNI & Co., Elektro-Industrie, Stationsstrasse, 8306 Bruttisellen,
  Telefon 01-833-3333

(*) = Forth-Enthusiasten oder Gruppen, die noch nicht offiziell als Gruppe der FG anerkannt sind bzw. zu dem Zweck noch Verbindungen zu weiteren Forthlern suchen. Wenn Sie interessiert sind, kann hier auch ihr Name stehen. Schreiben Sie an die VIERTE DIMENSION.

## FORTH-SYSTEME Angelika Flesch

| | |
|---|---|
| **cpcForth** mit Assembler,Editor,Grafik unter AMSDOS Disk. | 178,-- DM |
| **HES Modul** für **C64**, 16K Cartridge | 168,-- DM |
| **C64-SuperFORTH** von ParSec mit Expertensystem, Floating Point, Matrix Algebra, 4 Diskettenseiten, 400 S.Doc. | 398,-- DM |
| Atari **4xFORTH** schnelles 32 Bit FORTH mit **GEM** | 548,-- DM |
| **LMI PC/FORTH** für den Atari ST, Relokatibles **32 Bit FORTH** mit 68000 Assembler, Multifileeditor,Floating Point und weitere Hilfsprogramme | 598,50 DM |
| **MultiFORTH** für den **Amiga** | 598,-- DM |
| **LMI IBM-PC/FORTH** unter MSDOS | 598,50 DM |
| **LMI PC/FORTH** unter CP/M80, CP/M86 | 398,-- DM |
| **EXPERT II** Disk für verschiedene FORTH-Systeme | 198,-- DM |
| Mac Intosh **Mach 2**, Multitasking Forth mit Windows | 299,-- DM |
| Coloring Book für Atari ST | 125,-- DM |

**FORTH Bücher**

| | |
|---|---|
| **Starting FORTH** v. Brodie, dt. | 48,-- DM |
| **Thinking FORTH** v. Brodie, dt. | 48,-- DM |
| **Mastering FORTH**, eine Einführung in FORTH83 engl. | 78,-- DM |
| Designing and Programming **Personal Expert Systems** | 65,-- DM |
| **EXPERT II** Expertensystem in FORTH | 98,-- DM |
| **FORTH Encyclopedia** | 98,-- DM |
| **Advanced MSDOS** v. Ray Duncan (LMI) **Microsoft Press** | 78,-- DM |

*FORML und Rochester Proceedings sowie alle Ausgaben der FORTH Dimensions ebenfalls ab Lager lieferbar.*

FORTH-SYSTEME
*Angelika Flesch*
Postfach 1226
7820 Titisee-Neustadt
Tel.: 07651/1665 od. 3304

Unsere neue Anschrift ab 01.01.1987

Postfach 1103
Kühnheimerstr.
D-7814 Breisach
Tel.: 07667/551

**FORTH-SYSTEME Angelika Flesch**
sucht zum 1. Januar 1987 für
**SOFTWARE-ENTWICKLUNG**

einen jungen, dynamischen Mitarbeiter/Mitarbeiterin mit Kenntnissen und Erfahrungen mit der Programmiersprache **FORTH**.

Englische Sprachkenntnisse sind erforderlich.

Ihr Arbeitsplatz wird sich in unseren neuen Geschäftsräumen in 7814 Breisach befinden. Ihr Arbeitsgebiet wird auch Kundenunterstützung vor Ort sowie telefonische Beratung umfassen.

Bitte bewerben Sie sich schriftlich mit aussagefähigen Unterlagen (Lichtbild) und Gehaltsvorstellung.

**FORTH-SYSTEME Angelika Flesch**
**Postfach 12 26**
**D-7820 Titisee-Neustadt**