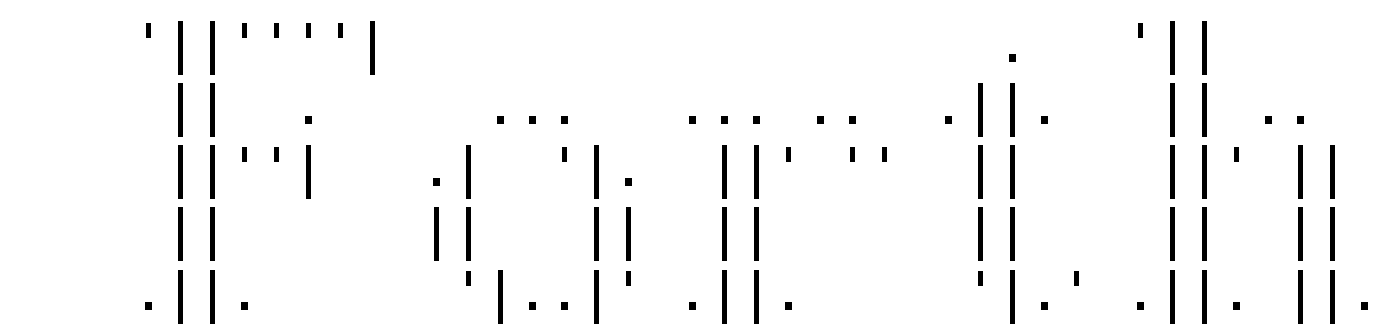
(Forth Matthias Koch mecrisp.sf.net [>])

Mikrocontroller mal anders: Von innen !



// Mecrisp // MSP430

// Mecrisp-Stellaris // ARM Cortex M0, M3, M4, M7

// Mecrisp-Ice // iCE40 FPGAs

(Forth Matthias Koch mecrisp.sf.net [>])

Eine kleine Vorstellung von mir:

Diplom-Physiker mit Nebenfach Gartenbau

Doktorand im Hannoverschen Zentrum für optische Technologien
"Laserspektroskopie an Algen"

Persönliche Interessen:
Obstanbau, Radionavigation, Meeresbiologie,
Historische Tänze, Apnoe-Tauchen, Jonglieren

Programmiersprachen:
Assembler, Forth, Pascal

(Forth Matthias Koch mecrisp.sf.net [->])

Was ist Forth ?

* Sehr kleiner Compiler im Mikrocontroller
* Interaktiv, Mensch "chattet" mit Chip
* Zwei Stacks
* Dictionary
* Umgekehrte polnische Notation
* Echtzeitfähig
* Neuerdings mit starken Compileroptimierungen

Erste Gehversuche:

Eintippen:	Antwort:
1 2 + .	3 ok.
13 8 - .	5 ok.
5 dup * .	25 ok.
1 2 3 .s	Stack: [3] 1 , 2 , 3 *> ok.

Definitionen:

```
: quadrat dup * ;        ok.
9 quadrat .              81 ok.
```

```
: Name    Legt eine neue Definition an
;         Definition fertig !
```

```
(  Forth    Matthias Koch   mecrisp.sf.net   [-- >           ]  )

  Kommentare und Konventionen:

    : quadrat ( n -- n^2 ) dup * ; \ Quadriert eine Zahl


    n    Zahl mit Vorzeichen  (N wie in der Mathematik)
    u    Vorzeichenlose Zahl  (unsigned)
    x    Bitmaske
    ?    Flag

    d    Doppeltlange Zahl                    (double)
    ud   Vorzeichenlose doppeltlange Zahl     (unsigned double)

    c-addr  Byte-Adresse                      (char address)
    a-addr  "Ausgerichtete" Adresse           (aligned address)
```

```
(  Forth    Matthias Koch   mecrisp.sf.net   [--->           ]  )

  Mit dem Stack jonglieren:                 Returnstack:

  dup       ( x -- x x )                    >r  ( x --  R: -- x )
  swap      ( x1 x2 -- x2 x1 )              r@  ( -- x  R: x -- x )
  over      ( x1 x2 -- x1 x2 x1 )           r>  ( -- x  R: x -- )
  drop      ( x -- )                      rdrop ( --    R: x -- )

  rot       ( x1 x2 x3 -- x2 x3 x1 )
  nip       ( x1 x2 -- x2 )
  tuck      ( x1 x2 -- x2 x1 x2 )

  Einblicke:                Rechnungen und Logik:
  .s          ( -- )        + - * / mod              ( n1 n2 -- n )
   words      ( -- )        and or xor               ( x1 x2 -- x )
   see Name ( -- )          lshift rshift arshift    ( x1 u  -- x )
```

```
  Verzweigungen:
  Bedingung  if  True-Zweig  else  False-Zweig  then

    : flag. ( ? -- ) if ." True" else ." False" then ;

     0 .flag                False  ok.
    -1 .flag                True  ok.

  Vergleiche:

    = <>                 ( x1 x2 -- ? )
    < <= >= >            ( n1 n2 -- ? )
    u< u<= u>= u>        ( u1 u2 -- ? )
    0= 0<                ( x -- ? )
```

```
(  Forth    Matthias Koch   mecrisp.sf.net   [---->          ]  )

  Schleifen:

    begin ... again
    begin ... flag until
    begin ... while ... repeat



: ascii     ( -- ) begin key . again ;
: anykey    ( -- ) begin [char] * emit key? until key drop ;
: ascii-esc ( -- ) begin key dup 27 <> while . repeat drop ;
```

Speicher lesen und schreiben

@ (a-addr -- x)	Speicherstelle lesen	MSP430: 16 Bits
! (x a-addr --)	Speicherstelle schreiben	ARM: 32 Bits

c@ (c-addr -- c)	Byte lesen
c! (c c-addr --)	Byte schreiben

Im ARM zusätzlich h@ und h! für 16-Bit-Speicherzugriffe

Konstanten und Variablen

```
 42 constant antwort
  3 variable pi

         pi @ .         3  ok.
antwort pi !            ok.
         pi @ .        42  ok.
```

Viel Platz reservieren: 1024 buffer: Kilobyte

Zählende Schleifen:

```
: multiplikationstabelle ( -- ) cr
  11 1 do
         11 1 do
                i j * . space
             loop
         cr
       loop
;


: sterne ( u -- ) 0 ?do [char] * emit loop ;
```

```
(  Forth    Matthias Koch   mecrisp.sf.net   [------>        ]  )

  Zahlen und ihre Basis:

  binary decimal hex ( -- ) Zahlenbasis wählen

   . ( n -- )  Zahl  mit Vorzeichen ausgeben
  u. ( u -- )  Zahl ohne Vorzeichen ausgeben


  42   -5   %110100   #10   $1F

  Doppeltlange Zahlen: ( low high )
                          Dafür gibt's d+ d- 2dup 2swap 2drop...
    5. ud.        5  ok.
  -31.  d.      -31  ok.
```

```
(  Forth    Matthias Koch   mecrisp.sf.net   [------ >       ]  )
1 constant Anode     \ Ein kleines Beispiel für den MSP430G2553:
2 constant Kathode    \ Helligkeitsmessung mit einer Leuchtdiode

: Strahle ( -- )
  Anode   P2OUT cbis!
  Kathode P2OUT cbic!
  Anode Kathode or P2DIR cbis!  ;

 : Lichtmessungs-Vorbereitung ( -- )
   Anode   P2OUT cbic!            \ Sperrschichtkapazität
   Kathode P2OUT cbis!            \ durch Verpolung laden
   Anode Kathode or P2DIR cbis!
   begin Kathode P2IN cbit@ until \ Warte bis Kathode geladen ist
   Kathode P2DIR cbic!   ;

: Kathodenochgeladen? ( -- ? ) Kathode P2IN cbit@ ;
```

```
(  Forth    Matthias Koch   mecrisp.sf.net   [------->       ]  )

: Wiedunkel? ( -- u )
  Lichtmessungs-Vorbereitung
  0
  begin
    50 us
    Kathodenochgeladen?
  while
    1+
  repeat
;

: Demo ( -- )  Strahle 1000 ms  500 pwm-init
  begin
    Wiedunkel? dup pwm . cr
  key? until ;
```

Bis hierhin war nichts allzu Ungewöhnliches dabei -
das meiste findet sich so oder so ähnlich auch in
anderen Programmiersprachen wieder.

Forth kann aber auch etwas ganz Besonderes:

Es ist möglich, den Compiler zu erweitern,
eigene Kontrollstrukturen zu definieren und die
Sprache an die zu lösende Schwierigkeit anzupassen.

Das geht, weil in Forth die Grenze zwischen
Ausführen und Kompilieren fließend ist.

```
(  Forth    Matthias Koch   mecrisp.sf.net   [--------->      ]  )

Eine Spezialität von Forth: IMMEDIATE

: case 0 immediate ; ( init count of ofs )

: of  ( #of -- orig #of+1 / x -- )
    1+    ( count ofs )
    >r    ( move off the stack in case the control-flow )
          ( stack is the data stack. )
    postpone over  postpone = ( copy and test case value)
    postpone if    ( add orig to control flow stack )
    postpone drop  ( discards case value if = )
    r>             ( we can bring count back now )
immediate ;
```

```
(  Forth    Matthias Koch   mecrisp.sf.net   [-------- >     ]  )

: endof ( orig1 #of -- orig2 #of )
    >r   ( move #of off the stack )
    postpone else
    r>   ( we can bring count back now )
immediate ;

: endcase  ( orig1..orign #of -- )
    postpone drop  ( discard case value )
    begin
     dup
    while
      1- swap postpone then
    repeat
    drop
immediate ;
```

```
(  Forth    Matthias Koch   mecrisp.sf.net   [--------->     ]  )
  Noch eine Spezialität von Forth: <builds does>

: constant <builds ( x -- ) , does> ( -- x ) @ ;

Nehmen wir das mal auseinander:

: constant      Legt eine neue Definition an.
  <builds       Was geschehen soll, wenn eine neue Definition
                mit Hilfe von "constant" angelegt wird
  ,             Fügt einen Wert ans Dictionary an

  does>         Was die Definition, die erschaffen wird,
                später tun soll. does> legt einen Pointer bereit
  @             Hole die Konstante, die beim Bauen im Dictionary
                an passender Stelle abgelegt worden ist.
;               Fertig !
```

Blick unter die Haube:

Innendrin ist Forth ziemlich einfach aufgebaut.

```
: quit ( -- ) begin query interpret ."  ok." cr again ;
```

Query lässt uns eine Zeile eintippen.
Interpret wertet sie aus.
Wenn wir beim OK ankommen, ist wohl alles gut gegangen.
Und gleich nochmal !

Nicht gut gegangen ? Dann werden die Stacks gelöscht
und es geht ohne OK wieder von vorne los.

Interpret:

An den Leerzeichen ein Token ausschneiden

Suche es im Dictionary

Gefunden: Im Ausführmodus oder Immediate: Ausführen.
          Im Kompiliermodus: Call kompilieren.
Nicht gefunden ? Versuche, es als Zahl aufzufassen:

  Im Ausführ-Modus: Auf dem Stack liegenlassen
  Im Kompilier-Modus: Code, der die Zahl auf den Stack legt

Auch keine gültige Zahl ? --> Fehler.

(Forth Matthias Koch mecrisp.sf.net [---------- >])

Noch ein paar nützliche Helferlein für ARM und MSP430:

```
compiletoram
compiletoflash
reset
eraseflash

init
```

(Forth Matthias Koch mecrisp.sf.net [---------->])

"Artenvielfalt":

Sprachstandard DPANS94:
http://lars.nocrew.org/dpans/dpans6.htm

Forth200x:
http://www.forth200x.org/documents/html/core.html

Historisch, aber gut zum Verstehen:
FIG-Forth-Listings in Assembler (Forth Interest Group)

(Forth Matthias Koch mecrisp.sf.net [---------- >])

Empfehlung MSP430, für Einsteiger und diejenigen,
die auch gern mal Assembler programmieren möchten:

MSP430G2553: Der kleinste unterstützte MSP430, DIP20-Gehäuse
MSP430FR4133 Launchpad: SMD, aber mit LCD
MSP430FR6989 Launchpad: SMD, aber mit LCD und viel Speicher

Empfehlung ARM, für diejenigen, die große Ideen haben:

TI LM4F120/TM4C123 Launchpad: Klein, stark, überschaubar
Freescale Freedom KL25Z: Arduino-Pinout, chic, sehr komfortabel
STM32L053 Discovery: E-Paper-Anzeige
LPC1114FN28: Kommt im DIP28-Gehäuse, aber eher schwach

Für Fortgeschrittene, komplizierter:

STM32F3 Discovery: Analoges vom Feinsten, Inertialnavigation
STM32F4 Discovery: Sound und sehr viel Rechenleistung

(Forth Matthias Koch mecrisp.sf.net [------------ >])

Mecrisp-Ice und FPGAs

- Lattice HX1K Icestick
- Lattice HX8K Breakout Board
- Nandland Go, basiert auf HX1K

Nur Mut !

```
(  Forth    Matthias Koch   mecrisp.sf.net   [-------------> ]  )
```

Wer neugierig geworden ist und sofort loslegen möchte:

* Alle MSP430-Launchpads und viele ARM-Cortex M0, M3, M4, M7 Boards werden von Mecrisp und Mecrisp-Stellaris unterstützt

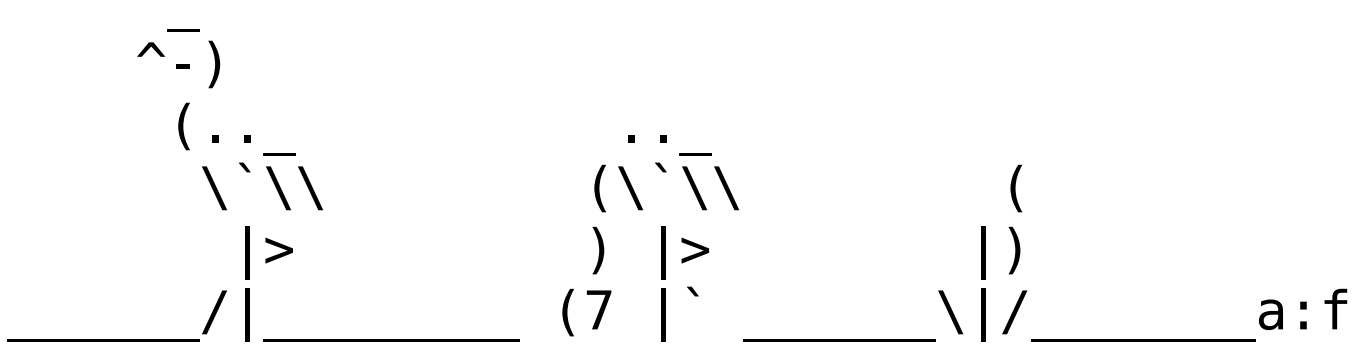
* Open Source Forths gibt es in allen Formen und Varianten - hier nur eine ganz kleine Auswahl für andere Architekturen:

PC:	Gforth	https://www.gnu.org/software/gforth/
AVR:	AmForth	http://amforth.sourceforge.net/
PIC:	FlashForth	http://flashforth.com/
Z80:	CamelForth	http://www.camelforth.com/

* Buch "Starting Forth" von Leo Brodie
  https://www.forth.com/starting-forth/
* Sonderheft "2015-ARM" der Vierten Dimension für Neugierige
  http://www.forth-ev.de/

(Forth   Matthias Koch   mecrisp.sf.net   [------------ >])

  May the Forth be with you !



 \ Ich freue mich auf viele E-Mails: \
  \ m-atthias@users.sf.net   \
   \   matthias.koch@hot.uni-hannover.de   \

```
12345678901234567890123456789012345678901234567890123456789012345
2
3    297 mm x 167 mm (16:9) Rahmen 10 mm überall.
4     DejaVu Sans Mono Schriftgröße 20
5
6    Format für den Vortrag:
7     17 Zeilen, 65 Buchstaben
8
9
0
1
2
3
4
5
6
7
```