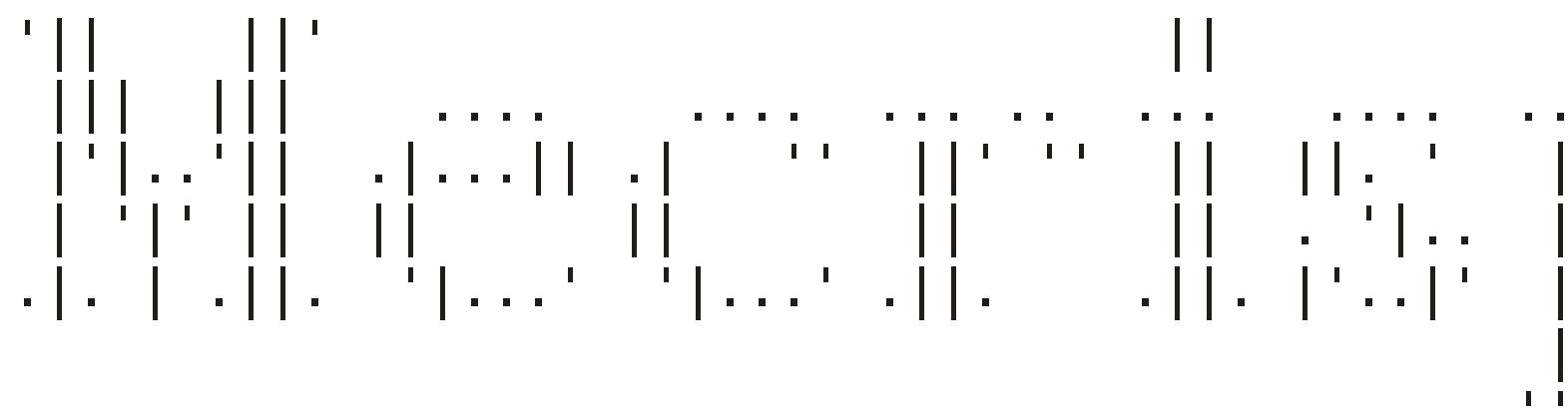
(32C3 Matthias Koch mecrisp.sf.net [>])

Compileroptimierungen für Forth im Microcontroller



// Mecrisp — // MSP430

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

// Mecrisp-Ice — // iCE40 HX1K FPGA

(32C3 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

(32C3 Matthias Koch mecrisp.sf.net [->])

Mecrisp kommt von MSP und dem französischen "écris":
"Du beschreibst den MSP430"

Die Entwicklung begann mit dem MSP430F2274 & MSP430G2553

Läuft mittlerweile auf allen MSP430-Launchpads und
als Mecrisp-Stellaris auch auf vielen
ARM Cortex M0, M3 und M4 Mikrocontrollern
von TI, STM, Freescale, NXP, Infineon.

Außerdem gibt es Mecrisp-Ice für den J1a von James Bowman,
eine um Optimierungen erweiterte Variante von SwapForth.
Ein ganz besonderer Chip: iCE40 HX1K -- ein FPGA !

Klassische Forth-Implementationen haben einen ganz einfachen Compiler, dessen Ausgabe exakt das repräsentiert, was der Benutzer eingegeben hat, meist auf einer "virtuellen Maschine".

Für die Fehlersuche, fürs Verstehen & Dekompilieren großartig:

Beim Jupiter ACE konnten alle Definitionen dekompiliert, editiert, rekompiliert und wieder ausgeführt werden.

In der Canon Cat war es sogar möglich, eine tief im System laufende Definition global durch eine Neue auszutauschen.
(Der Trick, für Neugierige: "Token-Threaded-Forth")

Optimierungen zerstören diesen interessanten Aspekt von Forth, helfen jedoch, schnelleren Code zu erzeugen, der sich mit aktuellen Compilern anderer Sprachen messen kann.

Mein Ziel war, herauszufinden, wie und welche möglichst starke Optimierungen mit den sehr begrenzten Möglichkeiten IN einem Mikrocontroller implementiert werden können:

I.	Tail-Call	J1a			
II.	Konstantenfaltung	J1a	MSP430	ARM	ARM-RA
III.	Inlining	J1a	MSP430	ARM	ARM-RA
IV.	Opcodierungen		MSP430	ARM	ARM-RA
V.	Registerallokation				ARM-RA

--- I. Tail-Call --- (Nur in Mecrisp-Ice)

Wenn ein Call der letzte Befehl in einer Definition ist,
kann die Sequenz

```
  Call Subroutine
  Return
```

durch

```
  Jump Subroutine
```

ersetzt werden.

--- II. Konstantenfaltung --- (Alle Mecrisps)

Wenn das Ergebnis einer Berechnung nur von Konstanten anhängt und damit schon während der Kompilation feststeht, genügt es, wenn nur das Ergebnis einkompiliert wird.

Beispielsweise bewirkt

```
: 42+ ( n -- 42+ ) 6 7 swap * + ;
```

dasselbe wie

```
: 42+ ( n -- 42+ ) 42 + ;
```

nur eben mit weniger Befehlen, die zur Laufzeit nötig sind.

Klassische Implementation des Interpreters (+Compilers)

Token abtrennen, versuchen, es im Dictionary zu finden.
    Gefunden:
      Im Kompilier-Modus und nicht Immediate ?
        --> Einen Aufruf zu dieser Definition einkompilieren
        Ansonsten die Definition ausführen

    Nicht gefunden - versuchen, es als Zahl zu interpretieren:
      Im Kompilier-Modus ?
      --> Etwas, was diese Zahl auf den Stack legen, kompilieren
      Ansonsten die Zahl einfach auf dem Stack liegen lassen

    Ergibt es auch keine gültige Zahl? --> Fehlermeldung, Abbruch

```
( 32C3   Matthias Koch   mecrisp.sf.net      [---->          ]  )
Token abtrennen, versuchen, es im Dictionary zu finden.
Aktuellen Stackfüllstand merken, falls noch nicht geschenen
    Gefunden:
      Im Execute-Modus --> Stackfüllstand vergessen, ausführen.
      Im Kompilier-Modus:
        Definition faltbar und genug Konstanten zur Verfügung ?
        --> Ausführen, fertig.
        Ansonsten alle vorhandenen Konstanten einkompilieren
        und den Stackfüllstand vergessen

        Immediate ? --> Ausführen, fertig.
        Inline ?    --> Opcodes einfügen, fertig
        Ansonsten Aufruf zu dieser Definition einkompilieren.
    Nicht gefunden - versuchen, es als Zahl zu interpretieren:
      Zahl stets auf dem Stack liegen lassen
    Es lässt sich auch nicht als Zahl auffassen ? --> Fehler
```

--- III. Inlining --- (Alle Mecrisps, jedoch unterschiedlich ausgeprägt)

Ist die Definition, die angesprungen werden soll,
kürzer als der Call-Befehl oder zumindest kurz genug,
können mit ein bisschen Vorsicht dessen Opcodes direkt
eingefügt werden. So können die Takte für den Call und
manchmal auch etwas Platz eingespart werden.

```
: Summe ( n1 n2 -- n1+n2 ) + ;
```

Sollte anstelle von   `call #Plus`   lieber als
`add  @r4+, 0(r4)` kompiliert werden.

--- IV. Opcodierungen --- (MSP430, ARM)

Manchmal lassen sich (einige) Konstanten direkt als Parameter in (bestimmte) Opcodes einfügen. Wieso sollten sie also über den Stack übergeben werden, wenn es auch direkt geht ?

Aus : 42+ (n -- 42+) 42 + ; wird dann nicht

```
  decd r4
  mov #42, 0(r4)
  add @r4+, 0(r4)
```

sondern einfach

```
  add #42, 0(r4)
```

--- V. Registerallokator --- (Nur ARM-RA)

Die Idee von Forth basiert auf dem Stack, auf welchem Parameter übergeben werden, wogegen viele Prozessorarchitekturen auf Registern basieren. Insbesondere beim ARM Cortex als Load-Store-Architektur benötigen Stackzugriffe jedoch mehr Takte, als wenn die Operationen direkt zwischen Registern stattfinden würden.

Der Registerallokator versucht nun für den Programmierer transparent, den Stack während der Kompilation auf Register abzubilden. Mecrisp-Stellaris RA ist die erste (mir bekannte ?) Implementation, die diese Optimierung *IN* einem Mikrocontroller durchführen kann.

Dafür ist es nötig, beim Kompilieren ein Stackmodell mitlaufen zu lassen, welches vermerkt, wo welches Stackelement sich gerade im Prozessor tatsächlich befindet. So können Zwischenergebnisse in Registern gehalten werden und einfache Operationen können direkt zwischen Registern wirken. Vor Unterprogrammaufrufen, an Kontrollstrukturverzweigungen und allen Stellen, wo der Stack für den Programmierer sichtbar wird (sein könnte), muss zuvor der "kanonische Grundzustand" des Stacks wiederhergestellt werden.

Soweit die zugrunde liegenden Ideen !

Jetzt wird es Zeit für einige Beispiele und Details,
wie die Optimierungen implementiert werden können.

(32C3 Matthias Koch mecrisp.sf.net [------->])

Mecrisp-Ice basiert auf vielen brandneuen tollen Entwicklungen:

J1a Prozessor und Swapforth	von James Bowman
Project Icestorm	von Clifford Wolf + Mathias Lasser
Yosys	von Clifford Wolf
Arachne-PNR	von Cotton Seed

http://www.excamera.com/sphinx/article-j1a-swapforth.html
http://www.clifford.at/icestorm/
http://www.clifford.at/yosys/
https://github.com/cseed/arachne-pnr

Von mir stammen die Optimierungen im Forth-Compiler, Unicode-Unterstützung, die Möglichkeit, Programme im Flash zu speichern und komfortable Peripherie nach Art des MSP430.

(32C3   Matthias Koch   mecrisp.sf.net   [------- >   ])

Der J1a ist ein von James Bowman entwickelter Stackprozessor
und läuft auf einem Lattice Icestick mit einem iCE40 HX1K FPGA.

Der Befehlssatz ist direkt auf Forth zugeschnitten.

In Mecrisp-Ice implementierte Optimierungen:

  * Konstantenfaltung
  * Automatisches Inline kurzer Definitionen
  * Tail-Call-Optimierung

```
( 32C3   Matthias Koch   mecrisp.sf.net      [-------->      ]  )

 1 constant Anode  16 constant Kathode

: shine ( -- )  \ Let it shine
  Anode Kathode or 4 io!
  Anode            2 io!
;

see shine                                  : shine
1A68 : 8011   Imm  0011       0011           Anode Kathode or
1A6A : 8004   Imm  0004       0004           4
1A6C : 45A6   Call 0B4C   --> io!            io!
1A6E : 8001   Imm  0001       0001           Anode
1A70 : 8002   Imm  0002       0002           2
1A72 : 05A6   Jmp  0B4C   --> io!            io! ; \ Tail-Call
 ok.
```

```
( 32C3   Matthias Koch   mecrisp.sf.net      [-------- >     ]  )

\ Check for discharge due to photocurrent
: finish-measurement ( -- Flag )
  1 io@ Kathode and 0=
;


see finish-measurement                    : finish-measurement
1AC2 : 8001   Imm  0001       0001          1
1AC4 : 45AB   Call 0B56   --> io@           io@
1AC6 : 8010   Imm  0010       0010          Kathode
1AC8 : 6303   Alu             and           and  \ Inline
1ACA : 0016   Jmp  002C   --> 0=            0= ; \ Tail-Call
 ok.
```

(32C3   Matthias Koch   mecrisp.sf.net   [--------->   ])

Zum Einstieg ein kleines Beispiel zur Konstantenfaltung selbst:

```
: 42+ ( n -- n+42 ) 6 7 swap * + ;  ok.
see 42+
1D58 : 802A   Imm  002A       002A
1D5A : 628F   Alu             + exit
 ok.
```

...Der Gedanke der Konstantenfaltung ist:

Berechne alles sofort, was schon beim Kompilieren feststeht,
und kompiliere nur das Ergebnis.

PS: Der J1a-Prozessor ist klein, verständlich und spannend -
unbedingt Verilog-Quelltext lesen und sich überraschen lassen !

(32C3 Matthias Koch mecrisp.sf.net [--------- >])

Mecrisp für dem MSP430 war meine erste Forth-Implementation, der Befehlssatz passt mit seinen flexiblen Addressierungsmodi recht gut zu Forth.

Implementierte Optimierungen:

* Konstantenfaltung
* Inline kurzer Definitionen
* Opcodierungen

```
( 32C3   Matthias Koch   mecrisp.sf.net      [---------->    ]  )

$20 constant P1IN  $21 constant P1OUT $22 constant P1DIR  ok.
$23 constant P1IFG $24 constant P1IES $25 constant P1IE  ok.
$26 constant P1SEL $27 constant P1REN  ok.


: init \ Launchpad hardware initialisations
  8 p1out cbis! \ High
  8 p1ren cbis! \ Pullup for button

  1 64 or p1out cbic! \ LEDs off
  1 64 or p1dir cbis! \ LEDs are outputs
;  ok.
```

```
( 32C3   Matthias Koch   mecrisp.sf.net      [---------- >   ]  )

see init                                    : init
C17E: D2F2  bis.b #8h, &21h                   8 p1out cbis!
C180: 0021
C182: D2F2  bis.b #8h, &27h                   8 p1ren cbis!
C184: 0027
C186: C0F2  bic.b #41h, &21h                  1 64 or p1out cbic!
C188: 0041
C18A: 0021
C18C: D0F2  bis.b #41h, &22h                  1 64 or p1dir cbis!
C18E: 0041
C190: 0022
C192: 4130  mov.w @r1+, r0                  ;
 ok.
```

Die Optimierungen in Mecrisp-Stellaris sind prinzipiell vom Design her mit denen in Mecrisp für den MSP430 identisch.

Deshalb folgen hier Beispiele für Mecrisp-Stellaris RA mit Registerallokator !

Implementierte Optimierungen:

* Konstantenfaltung
* Inline kurzer Definitionen
* Opcodierungen
* Registerallokator

```
: >gray ( u -- x ) dup 1 rshift xor ;  ok.

see >gray
00006B38: 0873  lsrs r3 r6 #1
00006B3A: 405E  eors r6 r3
00006B3C: 4770  bx lr
 ok.

: sqr ( n -- n^2 ) dup * ;  ok.

see sqr
00006B16: 4376  muls r6 r6
00006B18: 4770  bx lr
ok.
```

```
( 32C3   Matthias Koch   mecrisp.sf.net      [------------>  ]  )

27 variable e  31 variable p  0 variable s  ok.

: summe ( -- ) e @ p @ + s ! ;  ok.

see summe
00006C38: 2080  movs r0 #80
00006C3A: 0400  lsls r0 r0 #10
00006C3C: 30BE  adds r0 #BE
00006C3E: 0180  lsls r0 r0 #6
00006C40: 6A83  ldr r3 [ r0 #28 ]
00006C42: 6A42  ldr r2 [ r0 #24 ]
00006C44: 189B  adds r3 r3 r2
00006C46: 6203  str r3 [ r0 #20 ]
00006C48: 4770  bx lr
 ok.
```

Wer neugierig geworden ist und sofort loslegen möchte:

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

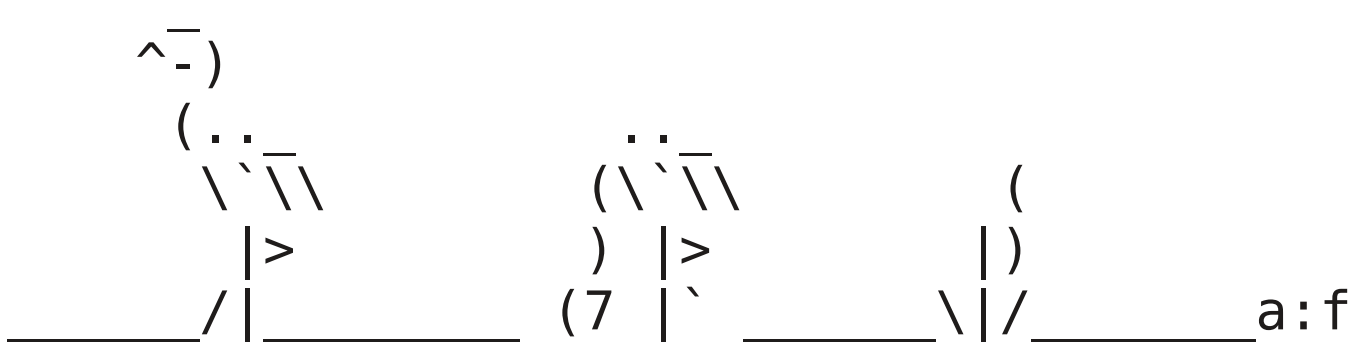
* Im Sonderheft "2015-ARM" der Vierten Dimension werden die Konstantenfaltung und der Registerallokator detailliert beschrieben (Als PDF oder hier auf Papier)

* 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/

(32C3   Matthias Koch   mecrisp.sf.net   [------------>])

  Alles Gute zum neuen Jahr !



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

```
( 32C3   Matthias Koch   mecrisp.sf.net      [o-------------]  )



: random ( -- u )
 ( Zufallszahlen mit dem Rauschen vom Temperatursensor am ADC )
 ( Random numbers with noise of temperature sensor on ADC )
   0
   16 0 do
     shl
    10 analog 1 and
    xor
  loop
;
```

```
see random                                  : random
C2F0: 1206  push.w r6                         0 16 0 do
C2F2: 1205  push.w r5
C2F4: 4305  mov.w #0h, r5
C2F6: 4036  mov.w #10h, r6
C2F8: 0010
C2FA: 8324  sub.w #2h, r4                     \ Die erste Null !
C2FC: 4384  mov.w #0h, 0h(r4)
C2FE: 0000
```

```
( 32C3   Matthias Koch   mecrisp.sf.net       [--o-----------]  )

C300: 54A4  add.w @r4, 0h(r4)                 shl
C302: 0000
C304: 8324  sub.w #2h, r4                     10
C306: 40B4  mov.w #Ah, 0h(r4)
C308: 000A
C30A: 0000
C30C: 12B0  call.w #FB6Ch  --> analog         analog
C30E: FB6C
C310: F394  and.w #1h, 0h(r4)                 1 and
C312: 0000
C314: E4B4  xor.w @r4+, 0h(r4)                xor
C316: 0000
```

```
( 32C3   Matthias Koch   mecrisp.sf.net      [---o----------]  )

C318: 5315  add.w #1h, r5                   loop
C31A: 9506  cmp.w r5, r6
C31C: 23F1  jnz C300
C31E: 4135  mov.w @r1+, r5
C320: 4136  mov.w @r1+, r6
C322: 4130  mov.w @r1+, r0                ;
```

(32C3 Matthias Koch mecrisp.sf.net [----o----------])

```
see bitexp                          : bitexp ( u -- u )
00006C5A: 2EF7  cmp r6 #F7            dup 247 u>
00006C5C: B500  push { lr }           if
00006C5E: D902  bls 00006C66
00006C60: 26F0  movs r6 #F0             drop $F0000000
00006C62: 0636  lsls r6 r6 #18
00006C64: E00C  b 00006C80            else
00006C66: 2E10  cmp r6 #10              dup 16 u<=
00006C68: D801  bhi 00006C6E            if
00006C6A: 0876  lsrs r6 r6 #1             1 rshift
00006C6C: E008  b 00006C80              else
```

```
( 32C3   Matthias Koch   mecrisp.sf.net      [-----o---------]  )

00006C6E: 0033  lsls r3 r6 #0                dup 7 and
00006C70: 2207  movs r2 #7
00006C72: 4013  ands r3 r2
00006C74: 2208  movs r2 #8                   8 or
00006C76: 4313  orrs r3 r2
00006C78: 08F6  lsrs r6 r6 #3                swap 3 rshift
00006C7A: 3E02  subs r6 #2                   2 -
00006C7C: 40B3  lsls r3 r6                   lshift
00006C7E: 461E  mov r6 r3                  then
00006C80: BD00  pop { pc }               then ;
```

```
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
```