AUS DER TRICKKISTE

z.B. ASCII

Bei der Programmierung von Menüs braucht man häufig den ASCII- Wert eines Zeichens. Anstatt nun jedesmal in der Tabelle nachzusehen und dann einen nichtssagenden Zahlenwert ins Programm zu schreiben, ist es sinnvoller, einen Befehl zu definieren, der ein nachfolgendes Zeichen automatisch in ASCII umgerechnet auf den Stack legt. Wenn dieser Befehl ASCII heißt, (der Autor benutzte hier die Bezeichnung $, im 83er Standard ist es jedoch bereits unter der Bezeichnung ASCII bekannt.) dann schreibt man also ASCII A statt 65 . Die folgende Definition bewerkstelligt das:

```
: ASCII  BL WORD HERE 1+ C§ ÄCOMPILEÜ LITERAL ;
IMMEDIATE
```

Das Wort holt das nächste Wort vom Input-Stream, pickt den ersten Buchstaben heraus und legt es auf den Stack. Wenn ASCII während des Kompilierens aufgerufen wird (es ist ja ein IMMEDIATE-Wort, d.h. es wird auch während des Kompilierens ausgeführt !), dann wird der auf dem Stack befindliche Wert als Literal in die gerade übersetzte Definition geschrieben. Der Effekt für die kompilierte Definition ist genau derselbe, als wenn man den ASCII-Wert direkt in den Quelltext schreiben würde.

z.B. RECURSE

In FORTH sind Rekursionen standardmäßig nicht möglich, weil ein Wort vom Compiler erst erkannt wird, wenn es fertig kompiliert ist. Manche Probleme lassen sich aber nun mal besser rekursiv darstellen. Diesem Mangel kann jedoch leicht abgeholfen werden:

```
: RECURSE  ( ruft laufende Definition rekursiv auf )
        ?COMP LATEST PFA CFA , ; IMMEDIATE
```

(Rekursion darf auf 6502-Systemen nicht ohne besondere Beachtung des Returnstacks angewendet werden.)

Der Befehl RECURSE bewirkt also einen Aufruf der Colondefinition , in der er auftritt. Er macht dies, indem er die Codefield-Adresse der laufenden Definition als Aufruf an die gerade aktuelle Speicherstelle schreibt. Vorher wird mit ?COMP noch geprüft, ob gerade kompiliert wird — außerhalb einer Colondefinition ergäbe RECURSE nämlich keinen Sinn.
RECURSE kann bedenkenlos auch innerhalb von Kontrollstrukturen (IF THEN, Schleifen, CASE etc.) verwendet werden — auf eines sollte man jedoch achten: Die Schachtelungstiefe darf nicht beliebig groß werden, da früher oder später (je nach Implementation) einer der beiden Stacks überläuft. Aber zehn bis zwanzig Ebenen dürften allemal drin sein, und mehr braucht man höchst selten.

Um gleich mal die Verwendung von RECURSE zu demonstrieren, folgende Definition:

```
: DEZ  ( dezimalzahl — 16-bit Integer )
       BASE § /MOD DUP IF RECURSE 10 * + ELSE DROP
THEN ;
```

DEZ bewirkt, daß eine 16-bit Zahl auf dem Stack so umgerechnet wird, als ob man sie dezimal eingegeben hätte — gleichgültig, unter welcher Zahlenbasis man sie tatsächlich eingegeben hat.. Wozu das? Nun: Ich fand es immer lästig, daß z.B. bei 10 LIST mal Screen 10 und mal Screen 16 erscheint — je nachdem, ob man dezimal oder in hex arbeitet. Die Definition
: LIST DEZ LIST ; löst das Problem. Jetzt kann ich mich darauf verlassen, daß auch wirklich die (dezimale) Screennummer kommt, die ich haben will — egal mit welcher Zalenbasis ich gerade arbeite.

Für diese TIPS & TRICKS sind wir DR. med.dent. Greiner höchst dankbar.
Natürlich könnt ihr alle hier eure kleinen "Helferlein" , die jeder von uns im Verlaufe seiner Forth-Tätigkeiten aus Faulheit entwickelt, loswerden und darauf hoffen , daß auch andere sie zu schätzen wissen.

Ein Decompiler für F.I.G.-FORTH

von Georg Rehfeldt (unserem C64 - Spezi)

Wie sicherlich die meisten von Euch wissen, erzeugt der FORTH- Compiler beim Schreiben von neuen Wortdefinitionen (z.B. : 2DUP OVER OVER ;) ins Wörterbuch zunächst einen Kopf, der außer dem Namen des Wortes (2DUP) und seiner Länge (4) noch andere Angaben enthält. Zunächst gibt es einen Zeiger auf ein vorhergehend compiliertes Wort; dieser Zeiger wird zum Durchsuchen des Wörterbuchs verwandt. Dann existiert noch ein Zeiger auf ausführbaren Maschinencode, der vom Inneren Interpreter benötigt wird. Die Adresse dieses Zeigers heißt Compilationsadresse des Wortes (CFA in FIG-FORTH). Außer dem Kopf wird in der Regel noch ein Rumpf ins Wörterbuch geschrieben. Bei einer Colon-Definition, ein Wort, das mit ":" eingeleitet und in der Regel mit ";" abgeschlossen wird, werden die Compilationsadressen der nach dem Namen folgenden Worte (OVER OVER) im Wörtebuch nacheinander abgelegt. Eine ":"-Definition besteht also aus dem Kopf und einer Liste von Compilationsadressen, die den Rumpf bildet. Der Innere Interpreter von FORTH kann aus diesen Zeigern bei der Ausführung des Wortes " 2DUP " die Worte " OVER OVER " wiedererkennen, denn jedes FORTH-Wort ist durch seine Compilationsadresse eindeutig gekennzeichnet.

Diese Tatsache macht sich der hier gezeigte Decompiler zunutze. Er ist in der Lage, compilierte Worte wieder lesbar auszulisten. Dies ist zur Fehlersuche in eigenen Worten ganz nützlich, eignet sich jedoch auch zur Analyse schlecht dokumentierter FORTH Versionen und hilft nicht zuletzt dem Anfänger beim Verständnis des Compilationsprozesses in FORTH.

Nicht alle FORTH-Worte compilieren einfach durch Ablage ihrer Compilationsadresse im Wörterbuch, es gibt zum Beispiel die IMMEDIATE-Worte, in der Regel Anweisungen an den Compiler, die etwas anderes ins Wörterbuch schreiben als ihre eigene Compilationsadresse. (So compiliert IF ein 0BRANCH, ELSE ein BRANCH, jeweils mit nachfolgender, relativer Sprungadresse und THEN überhaupt nichts ins Wörterbuch.) Eine ausführliche Beschreibung aller Sonderfälle führt hier zu weit, das Decompilieren selbstformulierter Worte zeigt das Verhalten dieser besonderen Worte am besten.

Die Bedienung des Decompilers ist einfach, einzutippen ist z.B.:

```
DEC 2DUP    (RETURNTASTE)
```

und der Decompiler antwortet mit :

```
2DUP
CFA 1234 INH 5678

1236 OVER
1238 OVER
123A ;S
```

Es gibt einige Worte, die nicht mit ;S oder (;CODE) enden, wie ABORT und QUIT, weil das Ende dieser Worte niemals erreicht wird. Bei diesen Worten kann der Decompiler das Ende des Wortes nicht erkennen und das Listing muß, wie beim vorzeitig gewünschten Abbruch, mit ?TERMINAL beendet werden.
Ein automatischer Decompiler hat einen großen Nachteil: Bei Erweiterungen des Compilers wie z.B. durch selbstdefinierte Definitionsworte, müßte der Decompiler entsprechend erweitert werden. Dies muß der Anwender jedoch selbst tun, oder die mit den grundlegenden Worten aus Screen Nr. 13 geschaffenen einfachen interaktiven Worte in Scren Nr. 17 benutzen, um solche Besonderheiten zu decompilieren.

Viel Spaß mit FORTH. Georg Rehfeld

Screen nr.:7

```
0( Tools dummy def. ==              04.08.1984 Georg Rehfeld ) 0-->
1
2forth definitions hex
3vocabulary tools immediate
4tools definitions
5
6: col.def. ;
7
80 constant const.
90 variable var.
:00 user usr.
:1
:2: == ( cfa -- )       2+ nfa  id.  space ;
:3
:4: case? ( n1 n2 -- tf / n1 ff ) over = dup if swap drop then ;
:5-->
```

Screen nr.:8

```
0
1
2       Die folgenden Worte werden als "TOOLS" zusammengefasst.
3       Dies sind Hilfsworte, die nur zu vergleichen gebraucht
4       werden.
5       == druckt aus der Compilationsadresseeines Wortes sei-
6       nen Namen.
7       Dank an Klaus Schleisiek, von dem die Worte "case?" und
8       die Idee zu .clit, .string, .lit, .branch und .name
9       stammen.
10      Wenn n1 = n2, dann hinterlässt case? "wahr", sonst
11      bleibt n1 erhalten und oben auf dem Stapel ist "falsch".
12      Danach Rücktehr ins Aufrufende Wort.
13
14
15
```

Screen nr.:9

```
0( Tools                            04.08.1984 Georg Rehfeld ) 0
1
2: .clit   ( adr -- adr+1 )   count . ;
3
4: .string ( adr1 -- adr2 )   count 2dup dup . type + ;
5
6: .lit    ( adr -- adr+2 )   dup @ dup + u. 2+ ;
7
8: .branch ( adr -- adr+2 )   dup @ over + u. 2+ ;
9
10: .name  ( adr -- adr+2 )   dup @ == 2+ ;
11
12
13-->
14
15
```

Screen nr.:10

```
0
1
2       .clit     - druckt das adressierte Byte als vorzeichenlose
3                   Zahl.
4       .string   - druckt den count und den adressierten String.
5
6       .lit      - druckt die adressierte Zelle als vorzeichenlose
7                   und vorzeichenbehaftete Zahl.
8       .branch   - druckt die adressierte Zelle als absolute
9                   Sprungadresse und als vorzeichenlose Zahl.
10      .name     - nimmt die adressierte Zelle als Compilations-
11                  adresse eines Wortes und druckt daraus den
12                  Namen des Wortes.
13
14      Bei allen Worten dieses Screens wird adr entsprechnd um 1,
15      2 oder mehr erhöht.
```

Screen nr.:11

```
( Tools  besonders?                 10.08.1984 Georg Rehfeld ) 0
1
2: besonders?   ( adr1 -- adr2 )
3       dup 2+ swap @ ' compile   cfa case?
4                     if .name    ;s  then
5                     ' clit      cfa case?
6                     if .clit    ;s  then
7                     ' lit       cfa case?
8                     if .string  ;s  then
9                     ' lit       cfa case?
10                    if .lit     ;s  then
11                    dup  ' 0branch  cfa =
12                    over ' branch   cfa = or
13                    over ' (loop)   cfa = or
14                    over ' (+loop)  cfa = or
15                    if drop .branch ;s then drop ; -->
```

Screen nr.:12

```
0
1
2       besonders? - prüft ob eine Sonderbehandlung für das gerade
3                    Compilierte Wort erforderlich ist.
4
5         Nach compile, wird das nächste Wort in der gleichen
6         Zeile gedruckt. Zu clit wird das folgende Byte vorzei-
7         chenlos und zu lit die nächste Zelle sowohl vorzeichenlos
8         als auch vorzeichenbehaftet ausgegeben. Bei (.") wird
9         der count und der String ausgegeben. Bei branch,0branch,
10        (loop) and (+loop) wird die Sprungadresse nicht relativ,
11        sondern absolut angegeben.
12
13        adr1 wird entsprechend erhöht, sodaß adr2 auf die
14        nächste cfa zeigt.
15
```

Screen nr.:13

```
0( Tools  identify                  04.08.1984 Georg Rehfeld ) 0
1: liste ( adr -- )
2       begin   cr dup u. .name 2- dup @ )r besonders?
3               r  ' )s         cfa =
4               r) ' (;code)    cfa = or
5               ?terminal or
6       until   drop ;
7
8: identify ( pfa -- )
9       dup cfa @ ' col.def. cfa @ case? if liste  ;s then
10      ' const. cfa @ case? if ." constant " @ u. ;s then
11      ' var.   cfa @ case? if ." variable " @ u. ;s then
12      ' usr.   cfa @ case? if ." user " c@ 10 +origin @ + @
13                                                u. ;s then
14      ' tools  cfa @ case? if ." vocabulary " drop ;s then
15      = if ." code " ;s then ." ???" ;            -->
```

Screen nr.:14

```
0
1
2       liste  - listet ':' -Definitionen aus den Compilations-
3                adressen. adr wird gedruckt, adr @ wird als cfa
4                verstanden und daraus der Name gedruckt. Dann
5                werden Besonderheiten geprüft, siehe besonders?
6                Abbruch erfolgt mit ?terminal oder nach ;s und
7                (;code). Achtung, bei Worten, die nicht mit ;s
8                enden, muß rechtzeitig mit ?terminal abgebroch-
9                en werden!
10
11      identify - ermittelt die Art des Wortes und veranlaßt bei
12               ':' -Definitionen das Listen des Wortes, bei
13               User-,Variablen und Constanten wird der Inhalt
14               ausgegeben.
15
```

Screen nr.:15

```
0( Tools  kopf, dec                 04.08.1984 Georg Rehfeld ) 0
1
2: kopf  ( pfa b -- pfa )
3       cr cr over nfa id. space 40 and if ." immediate " then
4       dup cfa dup ." cfa " u. ." inh " @ u. cr ;
5
6forth definitions
7
8
9: dec   ( -- )
10      -find if tools kopf identify
11             else ." not found " then cr ;
12
13
14
15-->
```

Screen nr.:16

```
0
1
2       kopf   - druckt den Namen des zu Decompilierenden
3                Wortes und einen Hinweis, wenn es "immediate"
4                ist.
5                Außerdem wird die Compilationsadresse und deren
6                Inhalt gedruckt.
7
8
9
10
11
12      dec    - decompiliert das folgende Wort, sofern es ge-
13               funden wird.
14
15
```

Screen nr.:17

```
0( Tools                            04.08.1984 Georg Rehfeld ) 0
1tools definitions
2: n    ( adr -- adr+2 )
3       dup u. .name   quit ;
4
5: l    ( adr -- adr+2 )
6       dup u. .lit    quit ;
7
8: b    ( adr -- adr+2 )
9       dup u. .branch quit ;
10
11: c   ( adr -- adr+2 )
12      dup u. .clit   quit ;
13
14: s   ( adr -- adr+2 )
15      dup u. .string quit ;     forth definitions   ;s
```

Screen nr.:18

```
0
1
2       n      - druckt adr und den Namen aus adr @ .
3
4       l      - druckt adr und adr @ als 16 bit Zahl.
5
6       b      - druckt adr und adr @ als Sprungziel bezogen auf
7                adr .
8       c      - druckt adr und adr c@ als 8 bit Zahl.
9
10      s      - druckt adr und den adressierten String.
11
12      siehe auch Screen Nr.13
13
14      Nochmals, dank an Klaus !
15
```