X-Forth#

Author: Carsten Strotmann
Language: FORTH
Compiler/Interpreter: X-FORTH /ATASM
Published: 10.2003

Sourcecode of X-Forth for Atari 800/800XL/130XE, indirect threaded Forth based on FIG-Forth

Code compiles with ATASM -> http://atasm.sf.net

```
; ---------------------------------------------------------------------------
; X-FORTH 1.1b
; RELEASE 18.10.2003
; Homepage: http://www.strotmann.de/twiki/bin/view/APG/ProjXForth
; License: GNU Public License (GPL)
;
; based on Sources from fig-forth and Andreas Jung
;
; compiles with ATASM --> http://atasm.sourceforge.net
;---------------------------------------------------------------------------

; Flags

DEBUG     = 0
KEYWAIT   = 1
FINDDEBUG = 0
FILE      = 1
DYNMEMTOP = 0

; Startadresse im Speicher

  .BANK
  * = $2000

; Kernal-Routinen des Atari800

  DOSVEC = $000A
  DOSINI = $000C

; FORTH-Systemkonstanten. Sie muessen gegebenenfalls an die jeweilige
; Hardware angepasst werden.

  BOS    = $8E             ; Start des Daten-Stacks in der Zeropage
  TOS    = $C6             ; Zeiger auf den TOS                        ($C6)
  N      = $F0             ; temporaerer Arbeitsspeicher               ($F0) (orig TOS
  IP     = N+8             ; Instruction Pointer IP                    ($F8)
  W      = IP+3            ; Codefeld-Pointer W                        ($FB)
  UP     = W+2             ; User-Pointer UP                           ($FD)
  XSAVE  = UP+2            ; temporaerer Speicher fuer das X-Register ($FF)

;
  TIBX   = $0100           ; Terminal Input Buffer, 84 Bytes
  MEM    = $B800           ; Ende des FORTH-Speichers
  UAREA  = MEM-128         ; User-Area, 128 Bytes

; Der Speicher zwischen dem Ende des Dictionaries und DAREA steht fuer
; Benutzerprogramme zur Verfuegung.
; Es folgen nun die Boot-up-Parameter, d.h. Sprungvektoren und
```

```
; Parameter zur Systembeschreibung.

  ORIG   NOP                ; 0+ORIGIN: Kaltstart ueber COLD
   JMP PFA_COLD                 ;
  REENTR NOP                ; 4+ORIGIN: Warmstart ueber WARM
   JMP WARM           ;
   .WORD $0004        ; 8+ORIGIN: "6502" zur Basis 36
   .WORD $5ED2        ;
   .WORD NFA_LASTWORD ; 12+ORIGIN: NFA des letzten Wortes
   .WORD 126          ; 14+ORIGIN: externer Backspace-Charakter
   .WORD UAREA        ; 16+ORIGIN: initialer User-Pointer UP
   .WORD TOS          ; 18+ORIGIN: Startwert fuer S0
   .WORD $1FF         ; 20+ORIGIN: Startwert fuer R0
   .WORD TIBX         ; 22+ORIGIN: Startwert fuer TIB
   .WORD 31           ; 24+ORIGIN: Startwert fuer WIDTH
   .WORD 0            ; 26+ORIGIN: Startwert fuer WARNING
   .WORD TOP          ; 28+ORIGIN: Startwert fuer FENCE
   .WORD TOP          ; 30+ORIGIN: Startwert fuer DP
   .WORD PFA_FORTH+6  ; 32+ORIGIN: Startwert fuer VOC-LINK

; Die folgenden NOP's bewirken eine Verschiebung des Dictionaries, so
; dass kein Codefeld auf eine Adresse der Form $XXFF faellt. Aufgrund
; eines Bugs im 6502-Prozessor wuerde dann naemlich der indirekte
; Sprung an der Adresse W-1 nicht richtig funktionieren. Bei jeder
; Aenderung des Dictionaries muessen die folgenden Zeilen gegebenenfalls
; geaendert werden.

;    NOP
;    NOP
 NOP


; >>>>>>>>>>>>>>>> LIT <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( LIT [ -> n ] legt die inline folgende Zahl auf den Stack.                  )
;( ========================================================================== )

NFA_LIT .CBYTE $83, "LIT"
LFA_LIT .WORD 0
CFA_LIT .WORD PFA_LIT
PFA_LIT

   LDA (IP),Y ; Lo-Byte der inline folgenden Zahl holen
   PHA        ; und auf den Stack retten
   INC IP     ; Instruction-Pointer inkrementieren
   BNE L30
   INC IP+1
L30
   LDA (IP),Y ; Hi-Byte der inline folgenden Zahl holen
L31
   INC IP     ; Instruction-Pointer inkrementieren
   BNE PUSH
   INC IP+1

PUSH
   DEX        ; Datenstack-Pointer dekrementieren
   DEX

PUT
```

```
   STA 1,X    ; Hi-Byte (ist in A) auf den Datenstack legen
   PLA        ; Lo-Byte vom Stack holen
   STA 0,X    ; und auf den Datenstack legen

  ; ========================================================================
  ; Der Adressinterpreter NEXT holt die Adresse des naechsten Secondaries
  ; und fuehrt es aus.
  ; ========================================================================

NEXT
   LDY #1
   LDA (IP),Y ; Hi-Byte der Wortadresse
   STA W+1    ; im W-Register ablegen
   DEY
   LDA (IP),Y ; Lo-Byte der Wortadresse
   STA W      ; im W-Register ablegen
.IF debug
   PHA
   LDA DEBUGFLG
   BEQ DL1
   PLA
   JSR TRACE  ; DEBUG
   PHA
DL1 PLA
.ENDIF
   CLC
   LDA IP     ; IP um 2 inkrementieren
   ADC #2
   STA IP
   BCC L54
   INC IP+1
L54
   JMP W-1    ; indirekten Sprung nach (W) ausfuehren

; >>>>>>>>>>>>>>>> CLIT <<<<<<<<<<<<<<<<
;( ========================================================================= )
;( CLIT [ -> b ] legt das inline folgende Byte auf den Stack.                )
;( ========================================================================= )

NFA_CLIT .CBYTE $84,"CLIT"
LFA_CLIT .WORD NFA_LIT
CFA_CLIT .WORD PFA_CLIT
PFA_CLIT
  LDA (IP),Y ; inline folgendes Byte holen
  PHA        ; und auf den Stack legen
  TYA        ; Akkumulator (Hi-Byte) mit Y=0 laden
  BEQ L31    ; und in die Routine LIT verzweigen

  ; ========================================================================
  ; Die Routine SETUP poppt n Zellen vom Datenstack und schreibt sie in den
  ; temporaeren Speicher ab Adresse N. Die Zahl n wird im Akkumulator
  ; uebergeben.
  ; ========================================================================

SETUP
  ASL A   ; n verdoppeln, ergibt Anzahl zu poppender Bytes
  STA N-1 ; Anzahl zu poppender Bytes sichern
L63
  LDA 0,X ; ein Byte vom Datenstack holen (zu Anfang ist Y=0)
```

```
  STA N,Y ; und im Speicherbereich ab Adresse N ablegen

  INX     ; Datenstack-Pointer inkrementieren
  INY     ; Zielindex inkrementieren
  CPY N-1 ; sind bereits 2*n Bytes gepoppt?
  BNE L63 ; nein, dann weitermachen
  LDY #0  ; Y.=0 wiederherstellen
  RTS     ; und fertig

; >>>>>>>>>>>>>>>> EXECUTE <<<<<<<<<<<<<<<
;( ======================================================================== )
;( EXECUTE [ addr -> ] fuehrt das Wort aus, dessen CFA auf dem Stack liegt.  )
;( ======================================================================== )

NFA_EXECUTE .CBYTE $87,"EXECUTE"
LFA_EXECUTE .WORD NFA_CLIT
CFA_EXECUTE .WORD PFA_EXECUTE
PFA_EXECUTE
  LDA 0,X ; Lo-Byte von addr
  STA W   ; in das W-Register schreiben
  LDA 1,X ; Hi-Byte von addr
  STA W+1 ; in das W-Register schreiben
  INX     ; addr vom Datenstack nehmen
  INX
  JMP W-1 ; Secondary ab addr ausfuehren

; >>>>>>>>>>>>>>>> BRANCH <<<<<<<<<<<<<<<
;( ======================================================================== )
;( BRANCH [ ] bewirkt einen relativen Sprung, abhaengig vom inline folgenden )
;( Offset.                                                                   )
;( ======================================================================== )

NFA_BRANCH .CBYTE $86, "BRANCH"
LFA_BRANCH .WORD NFA_EXECUTE
CFA_BRANCH .WORD PFA_BRANCH
PFA_BRANCH
  CLC
  LDA (IP),Y ; Lo-Byte des Offsets holen
  ADC IP     ; zu aktuellem IP hinzuaddieren
  PHA        ; und auf den Stack retten
  INY
  LDA (IP),Y ; Hi-Byte des Offsets holen
  ADC IP+1   ; zu aktuellem IP hinzuaddieren
  STA IP+1   ; und im IP-Register ablegen
  PLA        ; Lo-Byte des neuen IP-Inhaltes vom Stack holen
  STA IP     ; und im IP-Register ablegen
  JMP NEXT+2 ; dort weitermachen, LDY #1 kann uebersprungen werden

; >>>>>>>>>>>>>>>> 0BRANCH <<<<<<<<<<<<<<<
;( ======================================================================== )
;( 0BRANCH [ f -> ] bewirkt einen relativen Sprung, abhaengig vom inline     )
;( folgenden Offset, wenn f Null ist.                                        )
;( ======================================================================== )

NFA_0BRANCH .CBYTE $87, "0BRANCH"
LFA_0BRANCH .WORD NFA_BRANCH
CFA_0BRANCH .WORD PFA_0BRANCH
PFA_0BRANCH
       INX            ; f vom Datenstack nehmen
```

```
       INX
       LDA $FE,X       ; f testen
       ORA $FF,X
       BEQ PFA_BRANCH  ; verzweigen, falls f=0
  BUMP CLC
       LDA IP          ; sonst IP ueber den Offset hinwegsetzen
       ADC #2
       STA IP
       BCC L122
       INC IP+1
  L122 JMP NEXT        ; und weitermachen

; >>>>>>>>>>>>>>>> (LOOP) <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( [LOOP] [ ] inkrementiert den Loop-Index um 1 und testet auf Erreichen der )
;( Abbruchbedingung. Eventuell wird gemaess einem inline folgenden Offset    )
;( verzweigt.                                                                )
;( ========================================================================== )

NFA_BRACKETLOOP .CBYTE $86, "(LOOP)"
LFA_BRACKETLOOP .WORD NFA_0BRANCH
CFA_BRACKETLOOP .WORD PFA_BRACKETLOOP
PFA_BRACKETLOOP
  L130 STX XSAVE       ; X-Register retten
       TSX             ; Stackpointer nach X bringen
       INC $101,X      ; Loop-Index I inkrementieren
       BNE PL1
       INC $102,X
  PL1  CLC
       LDA $103,X      ; Hi(Schleifenlimit-I-1) berechnen
       SBC $101,X
       LDA $104,X
       SBC $102,X
  PL2  LDX XSAVE       ; X-Register wiederherstellen
       ASL A           ; falls obige Differenz nicht negativ:
       BCC PFA_BRANCH  ; an den Schleifenanfang verzweigen
       PLA             ; sonst Schleifenlimit und I vom Stack nehmen
       PLA
       PLA
       PLA
       JMP BUMP        ; und Offset uebergehen

; >>>>>>>>>>>>>>>> (+LOOP) <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( [+LOOP] [ n -> ] inkrementiert den Loop-Index um n und testet auf Errei-  )
;( chen der Abbruchbedingung. Eventuell wird gemaess einem inline folgenden  )
;( Offset verzweigt.                                                         )
;( ========================================================================== )

NFA_BRACKETPLUSLOOP .CBYTE $87, "(+LOOP)"
LFA_BRACKETPLUSLOOP .WORD NFA_BRACKETLOOP
CFA_BRACKETPLUSLOOP .WORD PFA_BRACKETPLUSLOOP
PFA_BRACKETPLUSLOOP
  INX        ; Datenstackpointer inkrementieren
  INX
  STX XSAVE  ; X-Register retten
  LDA $FF,X  ; Hi-Byte von n
  PHA        ; zweimal auf den Stack legen
  PHA
```

```
  LDA $FE,X  ; Lo-Byte von n holen
  TSX        ; Stackpointer nach X holen
  INX        ; Hi-Byte von n interessiert momentan nicht
  INX
  CLC
  ADC $101,X ; Lo-Byte von n auf I aufaddieren
  STA $101,X
  PLA        ; Hi-Byte von n holen
  ADC $102,X ; und auf I aufaddieren
  STA $102,X
  PLA        ; nochmal Hi-Byte von n holen
  BPL PL1    ; falls n positiv: bei (LOOP) weitermachen
  CLC
  LDA $101,X ; sonst Hi(I-Schleifenlimit-1) berechnen
  SBC $103,X
  LDA $102,X
  SBC $104,X
  JMP PL2    ; und hiermit bei (LOOP) weitermachen

; >>>>>>>>>>>>>>>> (DO) <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( [DO] [ n1 n2 -> ] bringt die Loop-Parameter [Startindex, Limit] zum       )
;( Return-Stack.                                                             )
;( ========================================================================== )

NFA_BRACKETDO .CBYTE $84, "(DO)"
LFA_BRACKETDO .WORD NFA_BRACKETPLUSLOOP
CFA_BRACKETDO .WORD PFA_BRACKETDO
PFA_BRACKETDO
   LDA 3,X  ; Limit zum Returnstack bringen
   PHA
   LDA 2,X
   PHA
   LDA 1,X  ; Startindex zum Returnstack bringen
   PHA
   LDA 0,X
   PHA
POPTWO
   INX      ; n2 von Datenstack poppen
   INX
POP
   INX      ; n1 vom Datenstack poppen
   INX
   JMP NEXT ; und weitermachen

; >>>>>>>>>>>>>>>> R <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( R [ -> n ] kopiert das oberste Element des Return-Stacks zum Parameter-   )
;( Stack.                                                                    )
;( ========================================================================== )

NFA_R .CBYTE $81, "R"
LFA_R .WORD NFA_BRACKETDO
CFA_R .WORD PFA_R
PFA_R
  STX XSAVE  ; Datenstackpointer retten
  TSX        ; X-Register als Index in den Returnstack benutzen
  LDA $101,X ; Lo-Byte vom Returnstack holen
  PHA        ; und merken
```

```
  LDA $102,X ; Hi-Byte vom Returnstack holen
  LDX XSAVE  ; Datenstackpointer wiederherstellen
  JMP PUSH   ; n auf den Datenstack pushen und fertig

; >>>>>>>>>>>>>>> I <<<<<<<<<<<<<<<
;( ========================================================================== )
;( I [ -> n ] legt den aktuellen Loop-Index auf den Stack.                    )
;( ========================================================================== )

NFA_I .CBYTE $81, "I"
LFA_I .WORD NFA_R
CFA_I .WORD PFA_R     ; Link zu "R", Befehl identisch
PFA_I

; >>>>>>>>>>>>>>> J <<<<<<<<<<<<<<<
;( ========================================================================== )
;( J [ -> n ] kopiert den Zaehlindex der aesseren Scheife auf den Datenstack )
;(                                                                            )
;( ========================================================================== )

NFA_J .CBYTE $81, "J"
LFA_J .WORD NFA_I
CFA_J .WORD PFA_J
PFA_J
  STX XSAVE  ; Datenstackpointer retten
  TSX        ; X-Register als Index in den Returnstack benutzen
  LDA $105,X ; Lo-Byte vom Returnstack holen
  PHA        ; und merken
  LDA $106,X ; Hi-Byte vom Returnstack holen
  LDX XSAVE  ; Datenstackpointer wiederherstellen
  JMP PUSH   ; n auf den Datenstack pushen und fertig


; >>>>>>>>>>>>>>> DIGIT <<<<<<<<<<<<<<<
;( ========================================================================== )
;( DIGIT [ c n1 -> n2 tf ] falls ok, [ c n1 -> ff ] falls schlecht. Wandelt  )
;( den ASCII-Charakter c in sein Zahlen-Aequivalent n2 um. Als Basis wird    )
;( n1 benutzt. Ist c ein gueltiges Ziffernsymbol im n1-System, so wird tf=1  )
;( zurueckgegeben, sonst ff=0.                                               )
;( ========================================================================== )

NFA_DIGIT .CBYTE $85, "DIGIT"
LFA_DIGIT .WORD NFA_J
CFA_DIGIT .WORD PFA_DIGIT
PFA_DIGIT
       SEC
       LDA 2,X  ; Zeichen c holen
       SBC #$30 ; und '0' subtrahieren
       BMI L234 ; falls negativ: Misserfolg melden
       CMP #$A  ; mit 10 vergleichen
       BMI L227 ; kleiner, dann weiter
       SEC
       SBC #7   ; sonst weitere 7 abziehen
       CMP #$A  ; falls kleiner als 10:
       BMI L234 ; Misserfolg melden
  L227
       CMP 0,X  ; ermittelte Zahl mit Basis n1 vergleichen
       BPL L234 ; Zahl groesser oder gleich Basis, dann Misserfolg melden
       STA 2,X  ; sonst Zahl in n2 zurueckgeben
```

```
       LDA #1   ; tf zurueckgeben (Y=0)
       PHA
       TYA
       JMP PUT
  L234 TYA      ; Misserfolg: ff zurueckgeben (Y=0)
       PHA
       INX
       INX
       JMP PUT

; >>>>>>>>>>>>>>>> (FIND) <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( [FIND] [ addr1 addr2 -> pfa b tf ] falls ok, [ addr1 addr2 -> ff ] sonst. )
;( Sucht im Dictionary, beginnend bei der NFA addr2, nach dem String addr1.  )
;( Wird das Wort gefunden, so wird seine PFA, das Count-Byte und ein true-   )
;( Flag uebergeben, sonst lediglich ein false-Flag.                          )
;( ========================================================================== )

NFA_BRACKETFIND  .CBYTE $86, "(FIND)"
LFA_BRACKETFIND  .WORD NFA_DIGIT
CFA_BRACKETFIND  .WORD PFA_BRACKETFIND
PFA_BRACKETFIND
       LDA #2      ; addr1 und addr2
       JSR SETUP   ; in den Speicher ab Adresse N poppen
       STX XSAVE   ; Datenstackpointer retten
  L249 LDY #0      ; Vergleich beginnt beim Count-Byte
       LDA (N),Y   ; Count-Byte des Woerterbucheintrages holen
       EOR (N+2),Y ; und mit Count-Byte des Strings vergleichen
       AND #$3F    ; dabei die beiden obersten Bits ignorieren
       BNE L281    ; bei Ungleichheit verzweigen
  L254 INY         ; naechstes Zeichen anvisieren
       LDA (N),Y   ; naechstes Zeichen des Woerterbucheintrages holen
       EOR (N+2),Y ; und mit naechstem Zeichen des Strings vergleichen
       ASL A       ; hoechstes Bit in den Uebertrag schieben
       BNE L280    ; bei Ungleichheit verzweigen
       BCC L254    ; weiter vergleichen falls String-Ende noch nicht erreicht
       LDX XSAVE   ; bei Gleichheit Datenstackpointer wiederherstellen
       DEX         ; auf dem Datenstack Platz fuer pfa und b schaffen
       DEX
       DEX
       DEX
       CLC
       TYA         ; Offset des letzten String-Zeichens
       ADC #5      ; plus 5 ergibt Offset des Parameterfeldes
       ADC N       ; plus NFA ergibt PFA
       STA 2,X     ; Lo-Byte der PFA in den Datenstack schreiben
       LDY #0
       TYA         ; eventuell aufgetretenen Uebertrag
       ADC N+1     ; in Hi-Byte der PFA beruecksichtigen
       STA 3,X     ; Hi-Byte der PFA in den Datenstack schreiben
       STY 1,X     ; Hi-Byte von b auf Null setzen
       LDA (N),Y   ; Count-Byte des gefundenen Wortes holen
       STA 0,X     ; und in den Datenstack schreiben
       LDA #1      ; True-Flag vorbereiten
       PHA
       JMP PUSH    ; auf den Datenstack legen und fertig
  L280 BCS L284    ; Suche ueberspringen falls String-Ende erreicht
  L281 INY         ; naechstes Zeichen anvisieren
       LDA (N),Y   ; naechstes Zeichen des Woerterbucheintrages holen
```

```
.IF finddebug
       JSR OUTCH   ; debug - Druckt die Suche nach einem Word
       LDA (N),Y   ; debug - wenn Fehler in der Verkettung bestehen,
                   ; debug - Woerter nicht gefunden werden
.ENDIF
       BPL L281    ; hoechstes Bit nicht gesetzt, dann weitersuchen
  L284 INY         ; sonst Byte hinter dem Namenfeld anvisieren
       LDA (N),Y   ; Lo-Byte des Linkfeldes holen
       TAX         ; und in das X-Register laden
       INY         ; Hi-Byte des Linkfeldes anvisieren
       LDA (N),Y   ; und holen
       STA N+1     ; und anstelle des alten Wertes von addr2 ablegen
       STX N       ; das gleiche mit dem Lo-Byte des Linkfeldes
       ORA N       ; im Linkfeld angegebene Adresse testen
       BNE L249    ; ungleich Null, dann erneut vergleichen
       LDX XSAVE   ; sonst Datenstackpointer wiederherstellen
       LDA #0      ; False-Flag vorbereiten
       PHA
       JMP PUSH    ; auf den Datenstack legen und fertig

; >>>>>>>>>>>>>>>> ENCLOSE <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( ENCLOSE [ addr1 c -> addr1 n1 n2 n3 ] separiert einen Text beginnend ab   )
;( der Adresse addr1 mit dem Delimiter c. n1 ist der Byte-Offset zum ersten  )
;( von c verschiedenen ASCII-Charakter, n2 der Byte-Offset zum ersten        )
;( Delimiter-Charakter c hinter dem Text und n3 der Byte-Offset zum ersten   )
;( Charakter, der auf diesen Delimiter folgt. Das Zeichen ASCII-Null dient   )
;( als unbedingter Delimiter, ab dem nicht mehr weitergesucht wird.          )
;( ======================================================================== )

NFA_ENCLOSE .CBYTE $87,"ENCLOSE"
LFA_ENCLOSE .WORD NFA_BRACKETFIND
CFA_ENCLOSE .WORD PFA_ENCLOSE
PFA_ENCLOSE
       LDA #2      ; Parameter addr1 und c
       JSR SETUP   ; in den Speicher ab N poppen
       TXA
       SEC         ; Platz im Datenstack fuer addr1, n1, n2 und n3
       SBC #8      ; bereitstellen
       TAX
       STY 3,X     ; Hi-Byte von n2 auf Null setzen (Y=0)
       STY 1,X     ; Hi-Byte von n1 ebenso
       DEY
  L313 INY         ; naechstes Zeichen im Text ab addr1 anvisieren
       LDA (N+2),Y ; Zeichen holen
       CMP N       ; mit c vergleichen
       BEQ L313    ; alle c's am Anfang ueberlesen
       STY 4,X     ; Position des ersten Nicht-c in n1 ablegen
  L318 LDA (N+2),Y ; naechstes Zeichen im Text ab addr1 holen
       BNE L327    ; nicht Null, dann verzweigen
       STY 2,X     ; sonst Position der Null in n2
       STY 0,X     ; und in n3 ablegen
       TYA
       CMP 4,X     ; Position mit Position des ersten Nicht-c vergleichen
       BNE L326    ; andere Position, dann verzweigen
       INC 2,X     ; sonst n2 inkrementieren
  L326 JMP NEXT    ; und fertig
  L327 STY 2,X     ; aktuelle Position in n2 ablegen
       INY         ; naechstes Zeichen anvisieren
```

```
       CMP N       ; aktuelles Zeichen mit c vergleichen
       BNE L318    ; bei Ungleichheit weitersuchen
       STY 0,X     ; sonst naechste Position im n3 ablegen
       JMP NEXT    ; und fertig

; >>>>>>>>>>>>>>>> EMIT <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( EMIT [ c -> ] sendet den ASCII-Charakter c zum Standard-Ausgabegeraet und )
;( erhoeht die USER-Variable OUT um 1.                                       )
;( ======================================================================== )

NFA_EMIT .CBYTE $84, "EMIT"
LFA_EMIT .WORD NFA_ENCLOSE
CFA_EMIT .WORD PFA_EMIT
PFA_EMIT
XEMIT
  TYA        ; A.=0 (Y=0)
  SEC
  LDY #$1A   ; Offset der User-Variablen OUT
  ADC (UP),Y ; OUT inkrementieren
  STA (UP),Y
  INY
  LDA #0
  ADC (UP),Y ; auch Hi-Byte von OUT beruecksichtigen
  STA (UP),Y
  TYA        ; Y Register auf den Stack retten
  PHA
  LDA 0,X    ; Zeichen c holen
  STX XSAVE  ; Datenstackpointer retten
  JSR OUTCH  ; Zeichen ausgeben
  LDX XSAVE  ; Datenstackpointer wiederherstellen
  PLA
  TAY        ; Y Register wiederherstellen
  JMP POP    ; c vom Datenstack entfernen und fertig

; >>>>>>>>>>>>>>>> KEY <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( KEY [ -> c ] wartet auf eine Tastaturbetaetigung und uebergibt den ASCII- )
;( Code der gedrueckten Taste.                                               )
;( ======================================================================== )

NFA_KEY .CBYTE $83, "KEY"
LFA_KEY .WORD NFA_EMIT
CFA_KEY .WORD PFA_KEY
PFA_KEY
XKEY
  STX XSAVE  ; Datenstackpointer retten
XKEY1
  JSR GETCH  ; Zeichen von der Tastatur einlesen (im accu)
  LDX XSAVE  ; Datenstackpointer wiederherstellen
  JMP PUSHOA ; Akkumulator als c auf den Datenstack und fertig

; >>>>>>>>>>>>>>>> ?TERMINAL <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( ?TERMINAL [ -> f ] liefert f=1 zurueck, wenn eine Taste betaetigt wurde,  )
;( sonst f=0.                                                                )
;( ======================================================================== )

NFA_QUERYTERMINAL .CBYTE $89, "?TERMINAL"
```

```
LFA_QUERYTERMINAL .WORD NFA_KEY
CFA_QUERYTERMINAL .WORD PFA_QUERYTERMINAL
PFA_QUERYTERMINAL
XQTER

  CH = $02FC

  LDA CH    ; wenn $FF dann keine Taste gedrückt
  CLC
  ADC #1    ; +1 = 0 wenn keine Taste gedrückt
  BEQ XQTER1
  LDA #1    ; ansonsten 1
XQTER1
  JMP PUSHOA ; Akkumulator auf den Datenstack legen und fertig

; >>>>>>>>>>>>>>>> CR <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( CR [ ] gibt einen Zeilenvorschub auf den Bildschirm aus.                   )
;( ========================================================================== )

NFA_CR .CBYTE $82, "CR"
LFA_CR .WORD NFA_QUERYTERMINAL
CFA_CR .WORD PFA_CR
PFA_CR
XCR   STX XSAVE  ; Datenstackpointer retten
      TYA
      PHA        ; Y Register retten
      LDA #155   ; Carriage Return
      JSR OUTCH  ; auf dem Bildschirm ausgeben
      LDX XSAVE  ; Datenstackpointer wiederherstellen
      PLA        ; Y Register wiederherstellen
      TAY
      JMP NEXT   ; und fertig

; >>>>>>>>>>>>>>>> CMOVE <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( CMOVE [ from to count ] kopiert von Adresse from nach Adresse to genau    )
;( count Bytes in aufsteigender Richtung.                                    )
;( ========================================================================== )

NFA_CMOVE .CBYTE $85, "CMOVE"
LFA_CMOVE .WORD NFA_CR
CFA_CMOVE .WORD PFA_CMOVE
PFA_CMOVE
       LDA #3      ; alle drei Parameter
       JSR SETUP   ; in den Speicherbereich ab Adresse N poppen
  L370 CPY N       ; noch nicht count Bytes kopiert?
       BNE L375    ; dann verzweigen
       DEC N+1     ; auch Hi-Byte von count beruecksichtigen
       BPL L375    ; und verzweigen, wenn noch Bytes zu kopieren sind
       JMP NEXT    ; sonst fertig
  L375 LDA (N+4),Y ; ein Byte holen
       STA (N+2),Y ; und im Zielbereich ablegen
       INY         ; naechstes Byte anvisieren
       BNE L370    ; bei Ueberlauf des Y-Registers:
       INC N+5     ; Hi-Byte von from inkrementieren
       INC N+3     ; Hi-Byte von to inkrementieren
       JMP L370    ; und weiterkopieren
```

```
; >>>>>>>>>>>>>>>> U* <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( U* [ u1 u2 -> ud ] multipliziert die beiden vorzeichenlosen Zahlen u1 und )
;( u2 und gibt das Ergebnis als doppeltgenaue, vorzeichenlose Zahl ud        )
;( zurueck.                                                                  )
;( ========================================================================== )

NFA_UMULT .CBYTE $82, "U*"
LFA_UMULT .WORD NFA_CMOVE
CFA_UMULT .WORD PFA_UMULT
PFA_UMULT
       LDA 2,X  ; u1 nach N/N+1 retten
       STA N    ; und ud auf Null setzen (Y=0)
       STY 2,X
       LDA 3,X
       STA N+1
       STY 3,X
       LDY #16  ; Y-Register als Zaehler (16 Bit)
  L396 ASL 2,X  ; Zwischenergebnis ud verdoppeln
       ROL 3,X
       ROL 0,X  ; und rechts in u2 hineinschieben
       ROL 1,X  ; dabei das jetzt hoechste Bit von u2 holen
       BCC L411 ; Bit=0, dann verzweigen
       CLC
       LDA N    ; sonst das gerettete u1 auf ud aufaddieren
       ADC 2,X
       STA 2,X
       LDA N+1
       ADC 3,X
       STA 3,X
       LDA #0   ; auch das Hi-Word von ud beruecksichtigen
       ADC 0,X
       STA 0,X
  L411 DEY      ; naechstes Bit anvisieren
       BNE L396 ; noch nicht alle Bits durch, dann weiterrechnen
       JMP NEXT ; sonst fertig

; >>>>>>>>>>>>>>>> U/ <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( U/ [ ud u1 -> u2 u3 ] dividiert die doppelt genaue, vorzeichenlose Zahl   )
;( ud durch die vorzeichenlose Zahl u1 und uebergibt den Quotienten u3 und   )
;( den Rest u2 als vorzeichenlose Zahlen.                                    )
;( ========================================================================== )

NFA_UDIV .CBYTE $82, "U/"
LFA_UDIV .WORD NFA_UMULT
CFA_UDIV .WORD PFA_UDIV
PFA_UDIV
       LDA 4,X  ; Hi-Word von ud nach [4/5,X] bringen
       LDY 2,X  ; und Lo-Word von ud nach [2/3,X]
       STY 4,X  ; wobei das Lo-Word bereits einmal verdoppelt wird
       ASL A
       STA 2,X
       LDA 5,X
       LDY 3,X
       STY 5,X
       ROL A
       STA 3,X
       LDA #16  ; Schleife wird 16 mal durchlaufen
```

```
       STA N    ; N ist Schleifenzaehler
  L433 ROL 4,X  ; Hi-Word von ud verdoppeln
       ROL 5,X
       SEC
       LDA 4,X  ; u1 vom Hi-Word von ud abziehen
       SBC 0,X  ; Ergebnis nach Y/A
       TAY
       LDA 5,X
       SBC 1,X
       BCC L444 ; falls Ergebnis nicht negativ:
       STY 4,X  ; Subtraktion an ud tatsaechlich durchfuehren
       STA 5,X
  L444 ROL 2,X  ; Lo-Word von ud nach links schieben
       ROL 3,X  ; dabei eventuell eine 1 in den Quotienten hineinschieben
       DEC N    ; naechstes Bit anvisieren
       BNE L433 ; noch nicht alle Bits durch, dann weitermachen
       JMP POP  ; sonst u1 vom Datenstack entfernen und fertig

; >>>>>>>>>>>>>>>> AND <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( AND [ n1 n2 -> n3 ] ermittelt die bitweise Und-Verknuepfung der Zahlen n1 )
;( und n2.                                                                    )
;( ========================================================================== )

NFA_AND .CBYTE $83, "AND"
LFA_AND .WORD NFA_UDIV
CFA_AND .WORD PFA_AND
PFA_AND
     LDA 0,X ; Lo-Byte von n1 AND n2 berechnen
     AND 2,X
     PHA
     LDA 1,X ; Hi-Byte von n1 AND n2 berechnen
     AND 3,X
BINARY
     INX     ; n2 vom Datenstack entfernen
     INX
     JMP PUT ; Ergebnis auf den Datenstack legen und fertig

; >>>>>>>>>>>>>>>> OR <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( OR [ n1 n2 -> n3 ] ermittelt die bitweise Oder-Verknuepfung der Zahlen    )
;( n1 und n2.                                                                 )
;( ========================================================================== )

NFA_OR .CBYTE $82, "OR"
LFA_OR .WORD NFA_AND
CFA_OR .WORD PFA_OR
PFA_OR
  LDA 0,X ; Lo-Byte von n1 OR n2 berechnen
  ORA 2,X
  PHA
  LDA 1,X ; Hi-Byte von n1 OR n2 berechnen
  ORA 3,X
  INX     ; n2 vom Datenstack entfernen
  INX
  JMP PUT ; Ergebnis auf den Datenstack legen und fertig

; >>>>>>>>>>>>>>>> XOR <<<<<<<<<<<<<<<<
;( ========================================================================== )
```

```
;( XOR [ n1 n2 -> n3 ] ermittelt die bitweise Exklusiv-Oder-Verknuepfung der )
;( Zahlen n1 und n2.                                                          )
;( ========================================================================= )

NFA_XOR .CBYTE $83, "XOR"
LFA_XOR .WORD NFA_OR
CFA_XOR .WORD PFA_XOR
PFA_XOR
  LDA 0,X ; Lo-Byte von n1 XOR n2 berechnen
  EOR 2,X
  PHA
  LDA 1,X ; Hi-Byte von n1 XOR n2 berechnen
  EOR 3,X
  INX     ; n2 vom Datenstack entfernen
  INX
  JMP PUT ; Ergebnis auf den Datenstack legen und fertig

; >>>>>>>>>>>>>>>> SP@ <<<<<<<<<<<<<<<<
;( ========================================================================= )
;( SP@ [ -> addr ] ermittelt die Adresse des TOS [vor dem Aufruf].           )
;( ========================================================================= )

NFA_SPFETCH .CBYTE $83, "SP@"
LFA_SPFETCH .WORD NFA_XOR
CFA_SPFETCH .WORD PFA_SPFETCH
PFA_SPFETCH
   TXA      ; Adresse des TOS wird durch das X-Register angegeben
PUSHOA
   PHA      ; Akkumulator als Lo-Byte
   LDA #0   ; und 0 als Hi-Byte
   JMP PUSH ; auf den Datenstack legen und fertig


; >>>>>>>>>>>>>>>> SP! <<<<<<<<<<<<<<<<
;( ========================================================================= )
;( SP! [ ] reinitialisiert den Stack-Pointer.                                )
;( ========================================================================= )

NFA_SPSTORE .CBYTE $83, "SP!"
LFA_SPSTORE .WORD NFA_SPFETCH
CFA_SPSTORE .WORD PFA_SPSTORE
PFA_SPSTORE
  LDY #6     ; User-Variable S0 enthaelt Initialwert des Datenstackpointers
  LDA (UP),Y ; Inhalt der User-Variablen holen
  TAX        ; in den Datenstackpointer X laden
  JMP NEXT   ; und fertig

; >>>>>>>>>>>>>>>> RP! <<<<<<<<<<<<<<<<
;( ========================================================================= )
;( RP! [ ] reinitialisiert den Return-Stack-Pointer.                         )
;( ========================================================================= )

NFA_RPSTORE .CBYTE $83, "RP!"
LFA_RPSTORE .WORD NFA_SPSTORE
CFA_RPSTORE .WORD PFA_RPSTORE
PFA_RPSTORE
  STX XSAVE  ; Datenstackpointer retten
  LDY #8     ; User-Variable R0 enthaelt Initialwert des Returnstackpointers
  LDA (UP),Y ; Inhalt der User-Variablen holen
```

```
  TAX        ; und in den Returnstackpointer laden
  TXS
  LDX XSAVE  ; Datenstackpointer wiederherstellen
  JMP NEXT   ; und fertig

; >>>>>>>>>>>>>>>> ;S <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( ;S [ ] beendet die Ausfuehrung des aktuellen Wortes und kehrt zum aufru-  )
;( fenden Wort zurueck.                                                      )
;( ========================================================================== )

NFA_EXIT .CBYTE $82, ";S"
LFA_EXIT .WORD NFA_RPSTORE
CFA_EXIT .WORD PFA_EXIT
PFA_EXIT
  PLA      ; Returnadresse vom Returnstack poppen
  STA IP   ; und in den IP laden
  PLA
  STA IP+1
  JMP NEXT ; und fertig

; >>>>>>>>>>>>>>>> LEAVE <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( LEAVE [ ] erzwingt das Verlassen der aktuellen DO ... LOOP - oder DO ...  )
;( +LOOP - Schleife, indem der Abbruchwert auf den Stand des aktuellen       )
;( Index-Wertes gesetzt wird.                                                )
;( ========================================================================== )

NFA_LEAVE .CBYTE $85, "LEAVE"
LFA_LEAVE .WORD NFA_EXIT
CFA_LEAVE .WORD PFA_LEAVE
PFA_LEAVE
  STX XSAVE  ; Datenstackpointer retten
  TSX        ; X-Register als Index in den Returnstack benutzen
  LDA $101,X ; Limit auf den Wert des Schleifenzaehlers I setzen
  STA $103,X
  LDA $102,X
  STA $104,X
  LDX XSAVE  ; Datenstackpointer wiederherstellen
  JMP NEXT   ; und fertig

; >>>>>>>>>>>>>>>> >R <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( >R [ n -> ] legt die Zahl n auf den Return-Stack.                         )
;( ========================================================================== )

NFA_RPUSH .CBYTE $82, ">R"
LFA_RPUSH .WORD NFA_LEAVE
CFA_RPUSH .WORD PFA_RPUSH
PFA_RPUSH
  LDA 1,X  ; HI-Byte von n
  PHA      ; auf den Returnstack pushen
  LDA 0,X  ; Lo-Byte von n
  PHA      ; auf den Returnstack pushen
  INX      ; n vom Datenstack entfernen
  INX
  JMP NEXT ; und fertig

; >>>>>>>>>>>>>>>> R> <<<<<<<<<<<<<<<<
```

```
;( ========================================================================= )
;( R> [ -> n ] poppt das oberste Element vom Return-Stack und legt es auf    )
;( den Parameterstack.                                                       )
;( ========================================================================= )

NFA_RPOP .CBYTE $82, "R>"
LFA_RPOP .WORD NFA_RPUSH
CFA_RPOP .WORD PFA_RPOP
PFA_RPOP
  DEX      ; Platz auf dem Datenstack fuer n schaffen
  DEX
  PLA      ; Lo-Byte vom Returnstack poppen
  STA 0,X  ; und in den Datenstack schreiben
  PLA      ; Hi-Byte vom Returnstack poppen
  STA 1,X  ; und in den Datenstack schreiben
  JMP NEXT ; und fertig


; >>>>>>>>>>>>>>>> 0= <<<<<<<<<<<<<<<<
;( ========================================================================= )
;( 0= [ n -> f ] uebergibt f=true, wenn die Zahl n gleich Null ist, sonst    )
;( false.                                                                    )
;( ========================================================================= )

NFA_NULLEQUAL .CBYTE $82, "0="
LFA_NULLEQUAL .WORD NFA_RPOP
CFA_NULLEQUAL .WORD PFA_NULLEQUAL
PFA_NULLEQUAL
       LDA 0,X  ; n testen
       ORA 1,X
       STY 1,X  ; Hi-Byte von f auf Null setzen (Y=0)
       BNE L613 ; Lo-Byte von f auf 1 setzen, falls n=0
       INY
  L613 STY 0,X
       JMP NEXT ; und fertig

; >>>>>>>>>>>>>>>> 0< <<<<<<<<<<<<<<<<
;( ========================================================================= )
;( 0< [ n -> f ] uebergibt f=true, wenn die Zahl n kleiner Null ist, sonst   )
;( false.                                                                    )
;( ========================================================================= )

NFA_LTNULL .CBYTE $82, "0<"
LFA_LTNULL .WORD NFA_NULLEQUAL
CFA_LTNULL .WORD PFA_LTNULL
PFA_LTNULL
  ASL 1,X  ; hoechstes Bit von n holen
  TYA      ; und im Akkumulator zu 1 oder 0 werden lassen (Y=0)
  ROL A
  STY 1,X  ; Hi-Byte von f auf Null setzen
  STA 0,X  ; Lo-Byte von f auf obiges Ergebnis setzen
  JMP NEXT ; und fertig

; >>>>>>>>>>>>>>>> + <<<<<<<<<<<<<<<<
;( ========================================================================= )
;( + [ n1 n2 -> n3 ] berechnet die Summe der Zahlen n1 und n2.               )
;( ========================================================================= )

NFA_PLUS .CBYTE $81, "+"
```

```
LFA_PLUS .WORD NFA_LTNULL
CFA_PLUS .WORD PFA_PLUS
PFA_PLUS
  CLC
  LDA 0,X  ; Summe n3.=n1+n2 berechnen und in den Datenstack schreiben
  ADC 2,X
  STA 2,X
  LDA 1,X
  ADC 3,X
  STA 3,X
  INX      ; n3 vom Datenstack entfernen
  INX
  JMP NEXT ; und fertig

; >>>>>>>>>>>>>>>> D+ <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( D+ [ d1 d2 -> Summe ] berechnet die doppeltgenaue Summe der beiden        )
;( doppeltgenauen Zahlen d1 und d2.                                          )
;( ========================================================================== )

NFA_DPLUS .CBYTE $82, "D+"
LFA_DPLUS .WORD NFA_PLUS
CFA_DPLUS .WORD PFA_DPLUS
PFA_DPLUS
  CLC
  LDA 2,X    ; Lo-Word der Summe berechnen
  ADC 6,X
  STA 6,X
  LDA 3,X
  ADC 7,X
  STA 7,X
  LDA 0,X    ; Hi-Word der Summe berechnen
  ADC 4,X
  STA 4,X
  LDA 1,X
  ADC 5,X
  STA 5,X
  JMP POPTWO ; d2 vom Stack entfernen und fertig

; >>>>>>>>>>>>>>>> MINUS <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( MINUS [ n1 -> n2 ] negiert die Zahl n1.                                   )
;( ========================================================================== )

NFA_NEGATE .CBYTE $85, "MINUS"
LFA_NEGATE .WORD NFA_DPLUS
CFA_NEGATE .WORD PFA_NEGATE
PFA_NEGATE
   SEC

MINUS1
   TYA
   SBC 0,X  ; Differenz 0-n1 berechnen (Y=0)
   STA 0,X
   TYA
   SBC 1,X
   STA 1,X
   JMP NEXT ; und fertig
```

```
; >>>>>>>>>>>>>>>> DMINUS <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( DMINUS [ d1 -> d2 ] negiert die doppeltgenaue Zahl d1.                   )
;( ======================================================================== )

NFA_DMINUS .CBYTE $86, "DMINUS"
LFA_DMINUS .WORD NFA_NEGATE
CFA_DMINUS .WORD PFA_DMINUS
PFA_DMINUS
  SEC
  TYA
  SBC 2,X    ; Lo-Word der Differenz 0-d1 berechnen (Y=0)
  STA 2,X
  TYA
  SBC 3,X
  STA 3,X
  JMP MINUS1 ; Hi-Word der Differenz berechnen und fertig

; >>>>>>>>>>>>>>>> OVER <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( OVER [ n1 n2 -> n1 n2 n1 ] legt den Second auf den Stack.                )
;( ======================================================================== )

NFA_OVER .CBYTE $84, "OVER"
LFA_OVER .WORD NFA_DMINUS
CFA_OVER .WORD PFA_OVER
PFA_OVER
  LDA 2,X  ; n1 holen
  PHA
  LDA 3,X
  JMP PUSH ; auf den Datenstack legen und fertig

; >>>>>>>>>>>>>>>> DROP <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( DROP [ n -> ] entfernt das oberste Element vom Stack.                    )
;( ======================================================================== )

NFA_DROP .CBYTE $84, "DROP"
LFA_DROP .WORD NFA_OVER
CFA_DROP .WORD POP
PFA_DROP

; >>>>>>>>>>>>>>>> SWAP <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( SWAP [ n1 n2 -> n2 n1 ] vertauscht die beiden obersten Stack-Elemente.   )
;( ======================================================================== )

NFA_SWAP .CBYTE $84, "SWAP"
LFA_SWAP .WORD NFA_DROP
CFA_SWAP .WORD PFA_SWAP
PFA_SWAP
  LDA 2,X ; Lo-Byte von n1 holen
  PHA     ; und merken
  LDA 0,X ; Lo-Byte von n2 holen
  STA 2,X ; und damit das Lo-Byte von n1 ueberschreiben
  LDA 3,X ; Hi-Byte von n1 holen und im Akkumulator merken
  LDY 1,X ; Hi-Byte von n2 holen
  STY 3,X ; und damit das Hi-Byte von n1 ueberschreiben
```

```
  JMP PUT ; gemerktes n1 in den Datenstack schreiben und fertig

; >>>>>>>>>>>>>>>> PICK <<<<<<<<<<<<<<
;( ========================================================================== )
;( PICK [ n1  -> n2  ] holt das n-te Element des Stacks                       )
;( ========================================================================== )

NFA_PICK .CBYTE $84, "PICK"
LFA_PICK .WORD NFA_SWAP
CFA_PICK .WORD PFA_PICK
PFA_PICK
    STX XSAVE   ; Stackzeiger sichern
    TXA         ; Stackzeiger in den Accu
    SEC
    ADC 0,X     ; Lo-Byte von n1 holen,
    ADC 0,X     ; verdoppeln und zum Stackzeiger addieren
    TAX
    DEX         ; in den Stackzeiger schieben, zeigt nun auf Element n1
    LDA 0,X     ; Lo-Byte von n2 holen
    PHA         ; und merken
    LDA 1,X     ; Hi-Byte von n2 holen
    LDX XSAVE

    JMP PUT     ; gemerktes n2 in den Datenstack schreiben und fertig

; >>>>>>>>>>>>>>>> DUP <<<<<<<<<<<<<<
;( ========================================================================== )
;( DUP [ n -> n n ] dupliziert das oberste Stack-Element.                     )
;( ========================================================================== )

NFA_DUP .CBYTE $83, "DUP"
LFA_DUP .WORD NFA_PICK
CFA_DUP .WORD PFA_DUP
PFA_DUP
  LDA 0,X  ; n holen
  PHA
  LDA 1,X
  JMP PUSH ; auf den Datenstack pushen und fertig


; >>>>>>>>>>>>>>>> +! <<<<<<<<<<<<<<<
;( ========================================================================== )
;( +! [ n addr -> ] inkrementiert die Zahl, auf die die Adresse addr zeigt,  )
;( um n.                                                                      )
;( ========================================================================== )

NFA_PLUSSTORE .CBYTE $82,"+!"
LFA_PLUSSTORE .WORD NFA_DUP
CFA_PLUSSTORE .WORD PFA_PLUSSTORE
PFA_PLUSSTORE
       CLC
       LDA (0,X)  ; Lo-Byte der durch addr adressierten Zahl
       ADC 2,X    ; plus Lo-Byte von n
       STA (0,X)  ; an der Adresse addr ablegen
       INC 0,X    ; addr inkrementieren
       BNE L754
       INC 1,X
  L754 LDA (0,X)  ; Hi-Byte der durch addr adressierten Zahl
       ADC 3,X    ; plus Hi-Byte von n
```

```
       STA (0,X)  ; an der Adresse addr+1 ablegen
       JMP POPTWO ; n und addr vom Datenstack entfernen und fertig

; >>>>>>>>>>>>>>>> TOGGLE <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( TOGGLE [ addr b -> ] fuehrt eine bitweise Exklusiv-Oder-Verknuepfung des  )
;( durch addr adressierten Bytes mit dem Bitmuster b durch.                  )
;( ======================================================================== )

NFA_TOGGLE .CBYTE $86, "TOGGLE"
LFA_TOGGLE .WORD NFA_PLUSSTORE
CFA_TOGGLE .WORD PFA_TOGGLE
PFA_TOGGLE
  LDA (2,X)  ; durch addr adressiertes Byte holen
  EOR 0,X    ; mit b XOR-verknuepfen
  STA (2,X)  ; und wieder im Speicher ablegen
  JMP POPTWO ; addr und b vom Datenstack entfernen und fertig

; >>>>>>>>>>>>>>>> @ <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( @ [ addr -> n ] ermittelt die durch addr adressierte Zahl.                )
;( ======================================================================== )

NFA_FETCH .CBYTE $81, "@"
LFA_FETCH .WORD NFA_TOGGLE
CFA_FETCH .WORD PFA_FETCH
PFA_FETCH
       LDA (0,X) ; Lo-Byte der durch addr adressierten Zahl holen
       PHA       ; und merken
       INC 0,X   ; addr inkrementieren
       BNE L781
       INC 1,X
  L781 LDA (0,X) ; Hi-Byte holen und im Akkumulator merken
       JMP PUT   ; gemerkte Zahl in den Datenstack schreiben und fertig

; >>>>>>>>>>>>>>>> C@ <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( C@ [ addr -> b ] ermittelt das durch addr adressierte Byte.               )
;( ======================================================================== )

NFA_CFETCH .CBYTE $82, "C@"
LFA_CFETCH .WORD NFA_FETCH
CFA_CFETCH .WORD PFA_CFETCH
PFA_CFETCH
  LDA (0,X) ; durch addr adressiertes Byte holen
  STA 0,X   ; und in den Datenstack schreiben
  STY 1,X   ; Hi-Byte auf Null setzen (Y=0)
  JMP NEXT  ; und fertig

; >>>>>>>>>>>>>>>> ! <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( ! [ n addr -> ] speichert die Zahl n an die Adresse addr.                 )
;( ======================================================================== )

NFA_STORE .CBYTE $81, "!"
LFA_STORE .WORD NFA_CFETCH
CFA_STORE .WORD PFA_STORE
PFA_STORE
       LDA 2,X    ; Lo-Byte von n holen
```

```
       STA (0,X)  ; und bei addr abspeichen
       INC 0,X    ; addr inkrementieren
       BNE L806
       INC 1,X
  L806 LDA 3,X    ; Hi-Byte von n holen
       STA (0,X)  ; und bei addr+1 abspeichern
       JMP POPTWO ; n und addr vom Datenstack entfernen

; >>>>>>>>>>>>>>>> C! <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( C! [ b addr -> ] speichert das Byte b an die Adresse addr.                 )
;( ========================================================================== )

NFA_CSTORE .CBYTE $82, "C!"
LFA_CSTORE .WORD NFA_STORE
CFA_CSTORE .WORD PFA_CSTORE
PFA_CSTORE
  LDA 2,X    ; Byte b holen
  STA (0,X)  ; und bei addr abspeichern
  JMP POPTWO ; b und addr vom Datenstack entfernen und fertig

; >>>>>>>>>>>>>>>> : <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( : [ ] leitet eine Colon-Definition ein.                                    )
;( ========================================================================== )

NFA_COLON .BYTE $81+$40, 186 ; IMMEDIATE
LFA_COLON .WORD NFA_CSTORE
CFA_COLON .WORD DOCOL
PFA_COLON
.WORD CFA_QUERYEXEC ; ?EXEC
.WORD CFA_STORECSP ; !CSP
.WORD CFA_CURRENT ; CURRENT
.WORD CFA_FETCH ; @
.WORD CFA_CONTEXT ; CONTEXT
.WORD CFA_STORE ; !
.WORD CFA_CREATE ; CREATE
.WORD CFA_RIGHTBRACKET ; ]
.WORD CFA_BRACKETCODE ; (;CODE)
                ; das neue Wort fuehrt zunaechst folgenden Code aus:
DOCOL
  LDA IP+1      ; Instruction Pointer IP auf den Returnstack pushen
  PHA
  LDA IP
  PHA
.IF debug
  PHA
  LDA DEBUGFLG
  BEQ DL2
  JSR TCOLON    ; DEBUG
DL2 PLA
.ENDIF
  CLC
  LDA W         ; Wortadressregister
  ADC #2        ; plus zwei ergibt Adresse des ersten Tokens
  STA IP        ; dort mit der Interpretation fortfahren
  TYA
  ADC W+1       ; auch Hi-Byte des IP beruecksichtigen (Y=0)
  STA IP+1
```

```
  JMP NEXT      ; Interpretation an neuer Stelle fortsetzen

; >>>>>>>>>>>>>>>> ; <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( ; [ ] schliesst eine Colon-Definition ab.                                  )
;( ========================================================================== )
; : ;
;  ?CSP        ( ueberpruefen, ob die Stackposition noch die alte ist )
;  COMPILE ;S  ( Semis an das Ende des Wortes kompilieren             )
;  SMUDGE      ( Wort gueltig machen                                  )
;  [COMPILE] [ ( in den Execute-Mode uebergehen                       )
; ; IMMEDIATE

NFA_SEMIS .BYTE $81+$40, 187 ; IMMEDIATE
LFA_SEMIS .WORD NFA_COLON
CFA_SEMIS .WORD DOCOL
PFA_SEMIS
.WORD CFA_QUERYCSP ; ?CSP
.WORD CFA_COMPILE ; COMPILE
.WORD CFA_EXIT ; ;S
.WORD CFA_SMUDGE ; SMUDGE
.WORD CFA_LEFTBRACKET ; [
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> CONSTANT <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( CONSTANT [ n -> ] definiert eine Konstante.                               )
;( ========================================================================== )

NFA_CONSTANT .CBYTE $88, "CONSTANT"
LFA_CONSTANT .WORD NFA_SEMIS
CFA_CONSTANT .WORD DOCOL
PFA_CONSTANT
.WORD CFA_CREATE ; CREATE
.WORD CFA_SMUDGE ; SMUDGE
.WORD CFA_COMMA ; ,
.WORD CFA_BRACKETCODE ; (;CODE)
            ; das neue Wort fuehrt den folgenden Code aus:
  DOCON LDY #2

  LDA (W),Y ; Lo-Byte des Wertes n im Parameterfeld holen
  PHA       ; und merken
  INY       ; Hi-Byte von n anvisieren
  LDA (W),Y ; und holen
  JMP PUSH  ; n auf den Datenstack pushen und fertig

; >>>>>>>>>>>>>>>> VARIABLE <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( VARIABLE [ n -> ] definiert und initialisiert eine Variable.              )
;( ========================================================================== )

NFA_VARIABLE .CBYTE $88,"VARIABLE"
LFA_VARIABLE .WORD NFA_CONSTANT
CFA_VARIABLE .WORD DOCOL
PFA_VARIABLE
.WORD CFA_CONSTANT ; CONSTANT
.WORD CFA_BRACKETCODE ; (;CODE)
           ; aber bei Aufruf den folgenden Code ausfuehren:
  DOVAR CLC
```

```
  LDA W    ; Lo-Byte der PFA der Variablen ermitteln
  ADC #2
  PHA      ; und merken
  TYA
  ADC W+1  ; Hi-Byte der PFA ermitteln und merken
  JMP PUSH ; PFA auf den Datenstack pushen und fertig

; >>>>>>>>>>>>>>>> USER <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( USER [ n -> ] definiert eine User-Variable mit dem Offset n.              )
;( ======================================================================== )

NFA_USER .CBYTE $84, "USER"
LFA_USER .WORD NFA_VARIABLE
CFA_USER .WORD DOCOL
PFA_USER
.WORD CFA_CONSTANT ; CONSTANT
.WORD CFA_BRACKETCODE ; (;CODE)
            ; aber bei Aufruf den folgenden Code ausfuehren:
DOUSE
  LDY #2
  CLC

  LDA (W),Y ; Lo-Byte des Offsets der Uservariablen
  ADC UP    ; plus Userpointer ergibt Adresse der Uservariablen
  PHA       ; merken
  LDA #0
  ADC UP+1  ; Hi-Byte holen und im Akkumulator merken
  JMP PUSH  ; gemerkte Adresse auf den Datenstack legen und fertig

NFA_PREV .= NFA_USER

;( ======================================================================== )
;( Definition diverser Konstanten.                                          )
;( ======================================================================== )

; >>>>>>>>>>>>>>>> 0 <<<<<<<<<<<<<<<<
; CONSTANT 0     ( Null                                              )

NFA_0 .CBYTE $81, "0"
LFA_0 .WORD NFA_PREV
CFA_0 .WORD DOCON
PFA_0 .WORD 0

; >>>>>>>>>>>>>>>> 1 <<<<<<<<<<<<<<<<
; CONSTANT 1     ( Eins                                              )

NFA_1 .CBYTE $81, "1"
LFA_1 .WORD NFA_0
CFA_1 .WORD DOCON
PFA_1 .WORD 1

; >>>>>>>>>>>>>>>> 2 <<<<<<<<<<<<<<<<
; CONSTANT 2     ( Zwei                                              )

NFA_2 .CBYTE $81,"2"
LFA_2 .WORD NFA_1
CFA_2 .WORD DOCON
PFA_2 .WORD 2
```

```
; >>>>>>>>>>>>>>>> 3 <<<<<<<<<<<<<<<<
; CONSTANT 3     ( Drei                                              )

NFA_3 .CBYTE $81, "3"
LFA_3 .WORD NFA_2
CFA_3 .WORD DOCON
PFA_3 .WORD 3

; >>>>>>>>>>>>>>>> BL <<<<<<<<<<<<<<<<
; CONSTANT BL    ( Blank                                             )

NFA_BL .CBYTE $82, "BL"
LFA_BL .WORD NFA_3
CFA_BL .WORD DOCON
PFA_BL .WORD 32


; >>>>>>>>>>>>>>>> ABORTINIT <<<<<<<<<<<<<<<<
; CONSTANT ABORTINIT    ( ABORT INIT ADRESSE                          )

NFA_ABORTINIT .CBYTE $89, "ABORTINIT"
LFA_ABORTINIT .WORD NFA_BL
CFA_ABORTINIT .WORD DOCON
PFA_ABORTINIT .WORD ABORTINIT

NFA_PREV .= NFA_ABORTINIT

.IF debug
;( ========================================================================== )
;( Definition Debug Flag.                                                    )
;( ========================================================================== )
; >>>>>>>>>>>>>>>> DEBUGFLAG <<<<<<<<<<<<<<<<
; CONSTANT 0     ( Null                                              )

NFA_DEBUGFLAG .CBYTE $89, "DEBUGFLAG"
LFA_DEBUGFLAG .WORD NFA_PREV
CFA_DEBUGFLAG .WORD DOCON
PFA_DEBUGFLAG .WORD DEBUGFLG
NFA_PREV .= NFA_DEBUGFLAG

; >>>>>>>>>>>>>>>> DEBUGON <<<<<<<<<<<<<<<<
NFA_DEBUGON .CBYTE $87, "DEBUGON"
LFA_DEBUGON .WORD NFA_PREV
CFA_DEBUGON .WORD DOCOL
PFA_DEBUGON
.WORD CFA_CLIT
.BYTE 1       ; 1
.WORD CFA_DEBUGFLAG ; DEBUGFLAG
.WORD CFA_STORE     ; !
.WORD CFA_EXIT    ; ;S

NFA_PREV .= NFA_DEBUGON

; >>>>>>>>>>>>>>>> DEBUGOFF <<<<<<<<<<<<<<<<
NFA_DEBUGOFF .CBYTE $88, "DEBUGOFF"
LFA_DEBUGOFF .WORD NFA_PREV
CFA_DEBUGOFF .WORD DOCOL
PFA_DEBUGOFF
```

```
.WORD CFA_CLIT
.BYTE 0       ; 0
.WORD CFA_DEBUGFLAG ; DEBUGFLAG
.WORD CFA_STORE     ; !
.WORD CFA_EXIT    ; ;S

NFA_PREV .= NFA_DEBUGOFF

.ENDIF ; DEBUG

; >>>>>>>>>>>>>>>> +ORIGIN <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( +ORIGIN [ n -> addr ] uebergibt die Adresse Origin+n. Origin ist die      )
;( tiefste Adresse im Forth-Kern.                                            )
;( ========================================================================== )

NFA_PLUSORIGIN .CBYTE $87, "+ORIGIN"
LFA_PLUSORIGIN .WORD NFA_PREV
CFA_PLUSORIGIN .WORD DOCOL
PFA_PLUSORIGIN
.WORD CFA_LIT ; LIT
.WORD ORIG
.WORD CFA_PLUS ; +
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> S0 <<<<<<<<<<<<<<<<
; 6  USER S0       ( Initialisierungswert fuer Daten-Stack-Pointer           )

NFA_S0 .CBYTE $82, "S0"
LFA_S0 .WORD NFA_PLUSORIGIN
CFA_S0 .WORD DOUSE
PFA_S0 .WORD 6

; >>>>>>>>>>>>>>>> R0 <<<<<<<<<<<<<<<<
; 8  USER R0       ( Initialisierungswert fuer Return-Stack-Pointer          )

NFA_R0 .CBYTE $82, "R0"
LFA_R0 .WORD NFA_S0
CFA_R0 .WORD DOUSE
PFA_R0 .WORD 8

; >>>>>>>>>>>>>>>> TIB <<<<<<<<<<<<<<<<
; 10 USER TIB      ( Startadresse des Terminal-Input-Buffers                 )

NFA_TIB .CBYTE $83, "TIB"
LFA_TIB .WORD NFA_R0
CFA_TIB .WORD DOUSE
PFA_TIB .WORD 10

; >>>>>>>>>>>>>>>> WIDTH <<<<<<<<<<<<<<<<
; 12 USER WIDTH    ( Maximallaenge von Forth-Namen                           )

NFA_WIDTH .CBYTE $85, "WIDTH"
LFA_WIDTH .WORD NFA_TIB
CFA_WIDTH .WORD DOUSE
PFA_WIDTH .WORD 12

; >>>>>>>>>>>>>>>> WARNING <<<<<<<<<<<<<<<<
; 14 USER WARNING  ( Kontrollvariable fuer Systembotschaften                 )
```

```
NFA_WARNING .CBYTE $87, "WARNING"
LFA_WARNING .WORD NFA_WIDTH
CFA_WARNING .WORD DOUSE
PFA_WARNING .WORD 14

; >>>>>>>>>>>>>>>> FENCE <<<<<<<<<<<<<<<<
; 16 USER FENCE    ( unterste Adresse des mit FORGET loeschbaren Bereichs     )

NFA_FENCE .CBYTE $85, "FENCE"
LFA_FENCE .WORD NFA_WARNING
CFA_FENCE .WORD DOUSE
PFA_FENCE .WORD 16

; >>>>>>>>>>>>>>>> DP <<<<<<<<<<<<<<<<
; 18 USER DP       ( Dictionary Pointer                                       )

NFA_DP .CBYTE $82, "DP"
LFA_DP .WORD NFA_FENCE
CFA_DP .WORD DOUSE
PFA_DP .WORD 18

; >>>>>>>>>>>>>>>> VOC-LINK <<<<<<<<<<<<<<<<
; 20 USER VOC-LINK ( Adresse einer Zelle in der Definition des letzten Vokab. )

NFA_VOCLINK .CBYTE $88, "VOC-LINK"
LFA_VOCLINK .WORD NFA_DP
CFA_VOCLINK .WORD DOUSE
PFA_VOCLINK .WORD 20

; >>>>>>>>>>>>>>>> SOURCE-ID <<<<<<<<<<<<<<<<
; 22 USER SOURCE-ID  ( Nummer des geoffneten Kanals                )

NFA_SOURCEID .CBYTE $89,"SOURCE-ID"
LFA_SOURCEID .WORD NFA_VOCLINK
CFA_SOURCEID .WORD DOUSE
PFA_SOURCEID .WORD 22

; >>>>>>>>>>>>>>>> IN <<<<<<<<<<<<<<<<
; 24 USER IN       ( Byte-Offset im gegenwaertigen Eingabe-Textbuffer         )

NFA_IN .CBYTE $82, "IN"
LFA_IN .WORD NFA_SOURCEID
CFA_IN .WORD DOUSE
PFA_IN .WORD 24

; >>>>>>>>>>>>>>>> OUT <<<<<<<<<<<<<<<<
; 26 USER OUT      ( Anzahl mittels EMIT ausgegebener Zeichen                 )

NFA_OUT .CBYTE $83, "OUT"
LFA_OUT .WORD NFA_IN
CFA_OUT .WORD DOUSE
PFA_OUT .WORD 26

; >>>>>>>>>>>>>>>> SCR <<<<<<<<<<<<<<<<
; 28 USER SCR      ( Nummer des zuletzt aufgerufenen Screens                  )

NFA_SCR .CBYTE $83, "SCR"
LFA_SCR .WORD NFA_OUT
```

```
CFA_SCR .WORD DOUSE
PFA_SCR .WORD 28

; >>>>>>>>>>>>>>>> OFFSET <<<<<<<<<<<<<<<<
; 30 USER OFFSET   ( Block-Offset zum Disk-Drive, auf das zugegriffen wird    )

NFA_OFFSET .CBYTE $86, "OFFSET"
LFA_OFFSET .WORD NFA_SCR
CFA_OFFSET .WORD DOUSE
PFA_OFFSET .WORD 30

; >>>>>>>>>>>>>>>> CONTEXT <<<<<<<<<<<<<<<<
; 32 USER CONTEXT  ( Pointer zum Context-Vokabular                           )

NFA_CONTEXT .CBYTE $87, "CONTEXT"
LFA_CONTEXT .WORD NFA_OFFSET
CFA_CONTEXT .WORD DOUSE
PFA_CONTEXT .WORD 32

; >>>>>>>>>>>>>>>> CURRENT <<<<<<<<<<<<<<<<
; 34 USER CURRENT  ( Pointer zum Current-Vokabular                           )

NFA_CURRENT .CBYTE $87,"CURRENT"
LFA_CURRENT .WORD NFA_CONTEXT
CFA_CURRENT .WORD DOUSE
PFA_CURRENT .WORD 34

; >>>>>>>>>>>>>>>> STATE <<<<<<<<<<<<<<<<
; 36 USER STATE    ( ungleich Null, wenn System im Compile-Mode              )

NFA_STATE .CBYTE $85, "STATE"
LFA_STATE .WORD NFA_CURRENT
CFA_STATE .WORD DOUSE
PFA_STATE .WORD 36

; >>>>>>>>>>>>>>>> BASE <<<<<<<<<<<<<<<<
; 38 USER BASE     ( gegenwaertig vereinbarte Zahlenbasis                    )

NFA_BASE .CBYTE $84, "BASE"
LFA_BASE .WORD NFA_STATE
CFA_BASE .WORD DOUSE
PFA_BASE .WORD 38

; >>>>>>>>>>>>>>>> DPL <<<<<<<<<<<<<<<<
; 40 USER DPL      ( Anzahl der Digits rechts vom Dezimalpunkt               )
NFA_DPL .CBYTE $83, "DPL"
LFA_DPL .WORD NFA_BASE
CFA_DPL .WORD DOUSE
PFA_DPL .WORD 40

; >>>>>>>>>>>>>>>> FLD <<<<<<<<<<<<<<<<
; 42 USER FLD      ( Kontrollvariable fuer formatierte I/O                   )

NFA_FLD .CBYTE $83, "FLD"
LFA_FLD .WORD NFA_DPL
CFA_FLD .WORD DOUSE
PFA_FLD .WORD 42

; >>>>>>>>>>>>>>>> CSP <<<<<<<<<<<<<<<<
```

```
; 44 USER CSP      ( gesicherter Stackpointer-Stand                          )

NFA_CSP .CBYTE $83, "CSP"
LFA_CSP .WORD NFA_FLD
CFA_CSP .WORD DOUSE
PFA_CSP .WORD 44

; >>>>>>>>>>>>>>>> R# <<<<<<<<<<<<<<<<
; 46 USER R#       ( Position des Editor-Cursors                             )

NFA_RNUM .CBYTE $82, "R#"
LFA_RNUM .WORD NFA_CSP
CFA_RNUM .WORD DOUSE
PFA_RNUM .WORD 46

; >>>>>>>>>>>>>>>> HLD <<<<<<<<<<<<<<<<
; 48 USER HLD      ( Adresse des letzten Zeichens bei Umwandlung Zahl->String )

NFA_HLD .CBYTE $83, "HLD"
LFA_HLD .WORD NFA_RNUM
CFA_HLD .WORD DOUSE
PFA_HLD .WORD 48

; >>>>>>>>>>>>>>>> 1+ <<<<<<<<<<<<<<<<
;( ========================================================================= )
;( 1+ [ n1 -> n2 ] inkrementiert die oberste Zahl auf dem Stack.             )
;( ========================================================================= )
;
; : 1+ 1 + ;

NFA_1PLUS .CBYTE $82,"1+"
LFA_1PLUS .WORD NFA_HLD
CFA_1PLUS .WORD DOCOL
PFA_1PLUS
.WORD CFA_1 ; 1
.WORD CFA_PLUS ; +
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> 2+ <<<<<<<<<<<<<<<<
;( ========================================================================= )
;( 2+ [ n1 -> n2 ] inkrementiert die oberste Zahl auf dem Stack um 2.        )
;( ========================================================================= )
;
; : 2+ 2 + ;

NFA_2PLUS .CBYTE $82,"2+"
LFA_2PLUS .WORD NFA_1PLUS
CFA_2PLUS .WORD DOCOL
PFA_2PLUS
.WORD CFA_2 ; 2
.WORD CFA_PLUS ; +
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> HERE <<<<<<<<<<<<<<<<
;( ========================================================================= )
;( HERE [ -> addr ] uebergibt die Adresse der ersten freien Stelle im        )
;( Dictionary.                                                               )
;( ========================================================================= )
;
```

```
; : HERE DP @ ; ( Adresse, auf die der Dictionary Pointer zeigt, zurueckgeben )

NFA_HERE .CBYTE $84,"HERE"
LFA_HERE .WORD NFA_2PLUS
CFA_HERE .WORD DOCOL
PFA_HERE
.WORD CFA_DP ; DP
.WORD CFA_FETCH ; @
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> ALLOT <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( ALLOT [ n -> ] vergroessert das Dictionary um n Bytes. Hierbei ist n eine )
;( vorzeichenbehaftete Zahl, kann also auch negativ sein.                    )
;( ========================================================================== )
;
; : ALLOT DP +! ;                    ( Dictionary-Pointer um n inkrementieren )

NFA_ALLOT .CBYTE $85,"ALLOT"
LFA_ALLOT .WORD NFA_HERE
CFA_ALLOT .WORD DOCOL
PFA_ALLOT
.WORD CFA_DP ; DP
.WORD CFA_PLUSSTORE ; +!
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> , <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( , [ n -> ] fuegt die Zahl n an das Dictionary an.                         )
;( ========================================================================== )
;
; : , HERE ! 2 ALLOT ;    ( n hinter das Dictionary schreiben und DP erhoehen )

NFA_COMMA .CBYTE $81,","
LFA_COMMA .WORD NFA_ALLOT
CFA_COMMA .WORD DOCOL
PFA_COMMA
.WORD CFA_HERE ; HERE
.WORD CFA_STORE ; !
.WORD CFA_2 ; 2
.WORD CFA_ALLOT ; ALLOT
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> C, <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( C, [ b -> ] fuegt das Byte b an das Dictionary an.                        )
;( ========================================================================== )
;
; : C, HERE C! 1 ALLOT ;  ( b hinter das Dictionary schreiben und DP erhoehen )

NFA_CCOMMA .CBYTE $82,"C,"
LFA_CCOMMA .WORD NFA_COMMA
CFA_CCOMMA .WORD DOCOL
PFA_CCOMMA
.WORD CFA_HERE ; HERE
.WORD CFA_CSTORE ; C!
.WORD CFA_1 ; 1
.WORD CFA_ALLOT ; ALLOT
.WORD CFA_EXIT ; ;S
```

```
; >>>>>>>>>>>>>>>> - <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( - [ n1 n2 -> n3 ] berechnet die Differenz n1-n2.                           )
;( ========================================================================== )
;
; : - MINUS + ;                          ( n2 negieren und auf n1 aufaddieren )

NFA_MINUS .CBYTE $81,"-"
LFA_MINUS .WORD NFA_CCOMMA
CFA_MINUS .WORD DOCOL
PFA_MINUS
.WORD CFA_NEGATE ; MINUS
.WORD CFA_PLUS ; +
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> = <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( = [ n1 n2 -> f ] vergleicht die Zahlen n1 und n2 und gibt true zurueck,   )
;( wenn sie gleich sind, sonst false.                                         )
;( ========================================================================== )
;
; : = - 0= ;            ( Differenz von n1 und n2 auf Gleichheit mit 0 testen )

NFA_EQUAL .CBYTE $81,"="
LFA_EQUAL .WORD NFA_MINUS
CFA_EQUAL .WORD DOCOL
PFA_EQUAL
.WORD CFA_MINUS ; -
.WORD CFA_NULLEQUAL ; 0=
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> U< <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( U< [ u1 u2 -> f ] liefert f=true, wenn u1<u2 ist, wobei u1 und u2 als     )
;( vorzeichenlose Zahlen aufgefasst werden.                                   )
;( ========================================================================== )
;
; : U< - 0< ;                              ( vergleichen durch Subtraktion )

NFA_ULT .CBYTE $82,"U<"
LFA_ULT .WORD NFA_EQUAL
CFA_ULT .WORD DOCOL
PFA_ULT
.WORD CFA_MINUS ; -
.WORD CFA_LTNULL ; 0<
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> < <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( < [ n1 n2 -> f ] liefert f=true, wenn n1<n2 ist.                           )
;( ========================================================================== )

NFA_LT .CBYTE $81,"<"
LFA_LT .WORD NFA_ULT
CFA_LT .WORD PFA_LT
PFA_LT
  SEC
  LDA 2,X   ; Testsubtraktion n1-n2 durchfuehren
```

```
  SBC 0,X
  LDA 3,X
  SBC 1,X
  STY 3,X   ; Hi-Byte von f auf Null setzen (Y=0)
  BVC L1258 ; kein Ueberlauf, dann verzweigen
  EOR #$80  ; sonst Vorzeichenbit der Differenz umdrehen
  L1258 BPL L1260 ; Differenz positiv, dann verzweigen (Y=0)
  INY       ; sonst true-Flag vorbereiten
  L1260 STY 2,X   ; ermitteltes Flag in den Datenstack schreiben
  JMP POP   ; n2 vom Stack entfernen und fertig

; >>>>>>>>>>>>>>>> > <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( > [ n1 n2 -> f ] liefert f=true, wenn n1>n2 ist.                          )
;( ========================================================================== )
;
; : > SWAP < ;                                                ( n1>n2 gdw n2<n1 )

NFA_GT .CBYTE $81,">"
LFA_GT .WORD NFA_LT
CFA_GT .WORD DOCOL
PFA_GT
.WORD CFA_SWAP ; SWAP
.WORD CFA_LT ; <
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> ROT <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( ROT [ n1 n2 n3 -> n2 n3 n1 ] rotiert die drei obersten Stackelemente.     )
;( ========================================================================== )
;
; : ROT >R SWAP R> SWAP ;    ( n1 mit Hilfe des Returnstacks nach vorne holen )

NFA_ROT .CBYTE $83,"ROT"
LFA_ROT .WORD NFA_GT
CFA_ROT .WORD DOCOL
PFA_ROT
.WORD CFA_RPUSH ; >R
.WORD CFA_SWAP ; SWAP
.WORD CFA_RPOP ; R>
.WORD CFA_SWAP ; SWAP
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> SPACE <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( SPACE [ ] sendet ein ASCII-Space zum Ausgabegeraet.                       )
;( ========================================================================== )
;
; : SPACE BL EMIT ;

NFA_SPACE .CBYTE $85,"SPACE"
LFA_SPACE .WORD NFA_ROT
CFA_SPACE .WORD DOCOL
PFA_SPACE
.WORD CFA_BL ; BL
.WORD CFA_EMIT ; EMIT
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> -DUP <<<<<<<<<<<<<<<<
```

```
;( ========================================================================== )
;( -DUP [ n1 -> n1 ] falls n1=0, [ n1 -> n1 n1 ] sonst. Dupliziert den TOS,  )
;( wenn er ungleich Null ist.                                                )
;( ========================================================================== )
;
; : -DUP DUP IF DUP ENDIF ;

NFA_MINUSDUP .CBYTE  $84,"-DUP"
LFA_MINUSDUP .WORD NFA_SPACE
CFA_MINUSDUP .WORD DOCOL
PFA_MINUSDUP
.WORD CFA_DUP ; DUP
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL1-*
.WORD CFA_DUP ; DUP
LBL1
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> TRAVERSE <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( TRAVERSE [ addr1 n -> addr2 ] gibt mit addr2 die Adresse des jeweils      )
;( anderen Ende eines Namens an, Laengenbyte inklusive. Ist n=1, so wird     )
;( aufsteigend gesucht, ist n=-1, so wird absteigend gesucht.                )
;( ========================================================================== )
;
; : TRAVERSE
;  SWAP
;  BEGIN OVER + 127 OVER C@ < UNTIL   ( solange suchen bis Byte>127 )
;  SWAP DROP
; ;

NFA_TRAVERSE .CBYTE $88,"TRAVERSE"
LFA_TRAVERSE .WORD NFA_MINUSDUP
CFA_TRAVERSE .WORD DOCOL
PFA_TRAVERSE
.WORD CFA_SWAP ; SWAP
LBL2
.WORD CFA_OVER ; OVER
.WORD CFA_PLUS ; +
.WORD CFA_CLIT ; CLIT
.BYTE 127
.WORD CFA_OVER ; OVER
.WORD CFA_CFETCH ; C@
.WORD CFA_LT ; <
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL2-*
.WORD CFA_SWAP ; SWAP
.WORD CFA_DROP ; DROP
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> LATEST <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( LATEST [ -> addr ] uebergibt die NFA des zuletzt im Current-Vokabular     )
;( definierten Wortes.                                                       )
;( ========================================================================== )
;
; : LATEST CURRENT @ @ ;

NFA_LATEST .CBYTE $86,"LATEST"
```

```
LFA_LATEST .WORD NFA_TRAVERSE
CFA_LATEST .WORD DOCOL
PFA_LATEST
.WORD CFA_CURRENT ; CURRENT
.WORD CFA_FETCH ; @
.WORD CFA_FETCH ; @
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> LFA <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( LFA [ pfa -> lfa ] wandelt die PFA eines Wortes in die entsprechende LFA  )
;( um.                                                                       )
;( ========================================================================== )
;
; : LFA 4 - ;

NFA_LFA .CBYTE $83,"LFA"
LFA_LFA .WORD NFA_LATEST
CFA_LFA .WORD DOCOL
PFA_LFA
.WORD CFA_CLIT ; CLIT
.BYTE 4
.WORD CFA_MINUS ; -
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> CFA <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( CFA [ pfa -> cfa ] wandelt die PFA eines Wortes in die entsprechende CFA  )
;( um.                                                                       )
;( ========================================================================== )
;
; : CFA 2 - ;

NFA_CFA .CBYTE  $83,"CFA"
LFA_CFA .WORD NFA_LFA
CFA_CFA .WORD DOCOL
PFA_CFA
.WORD CFA_2 ; 2
.WORD CFA_MINUS ; -
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> NFA <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( NFA [ pfa -> nfa ] wandelt die PFA eines Wortes in die entsprechende NFA  )
;( um.                                                                       )
;( ========================================================================== )
;
; : NFA 5 - -1 TRAVERSE ;

NFA_NFA .CBYTE $83,"NFA"
LFA_NFA .WORD NFA_CFA
CFA_NFA .WORD DOCOL
PFA_NFA
.WORD CFA_CLIT ; CLIT
.BYTE 5
.WORD CFA_MINUS ; -
.WORD CFA_LIT ; LIT
.WORD -1
.WORD CFA_TRAVERSE ; TRAVERSE
```

```
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> PFA <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( PFA [ nfa -> pfa ] wandelt die NFA eines Wortes in die entsprechende PFA  )
;( um.                                                                       )
;( ========================================================================== )
;
; : PFA 1 TRAVERSE 5 + ;

NFA_PFA .CBYTE $83,"PFA"
LFA_PFA .WORD NFA_NFA
CFA_PFA .WORD DOCOL
PFA_PFA
.WORD CFA_1 ; 1
.WORD CFA_TRAVERSE ; TRAVERSE
.WORD CFA_CLIT ; CLIT
.BYTE 5
.WORD CFA_PLUS ; +
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> !CSP <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( !CSP [ ] rettet die aktuelle Stackpointer-Position in die User-Variable   )
;( CSP.                                                                      )
;( ========================================================================== )
;
; : !CSP SP@ CSP ! ;

NFA_STORECSP .CBYTE $84,"!CSP"
LFA_STORECSP .WORD NFA_PFA
CFA_STORECSP .WORD DOCOL
PFA_STORECSP
.WORD CFA_SPFETCH ; SP@
.WORD CFA_CSP ; CSP
.WORD CFA_STORE ; !
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> ?ERROR <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( ?ERROR [ f n -> ] gibt die  Fehlermeldung Nummer n aus, wenn f=true ist.  )
;( ========================================================================== )
;
; : ?ERROR SWAP IF ERROR ELSE DROP ENDIF ;

NFA_QUERYERROR .CBYTE $86,"?ERROR"
LFA_QUERYERROR .WORD NFA_STORECSP
CFA_QUERYERROR .WORD DOCOL
PFA_QUERYERROR
.WORD CFA_SWAP ; SWAP
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL3-*
.WORD CFA_ERROR ; ERROR
.WORD CFA_BRANCH ; BRANCH
.WORD LBL4-*
LBL3
.WORD CFA_DROP ; DROP
LBL4
.WORD CFA_EXIT ; ;S
```

```
; >>>>>>>>>>>>>>>> ?COMP <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( ?COMP [ ] gibt eine entsprechende Fehlermeldung aus, wenn sich das System )
;( nicht im Compile-Mode befindet.                                           )
;( ========================================================================== )
;
; : ?COMP STATE @ 0= 17 ?ERROR ;

NFA_QUERYCOMP .CBYTE $85,"?COMP"
LFA_QUERYCOMP .WORD NFA_QUERYERROR
CFA_QUERYCOMP .WORD DOCOL
PFA_QUERYCOMP
.WORD CFA_STATE ; STATE
.WORD CFA_FETCH ; @
.WORD CFA_NULLEQUAL ; 0=
.WORD CFA_CLIT ; CLIT
.BYTE 17
.WORD CFA_QUERYERROR ; ?ERROR
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> ?EXEC <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( ?EXEC [ ] gibt eine entsprechende Fehlermeldung aus, wenn sich das System )
;( nicht im Execute-Mode befindet.                                           )
;( ========================================================================== )
;
; : ?EXEC STATE @ 18 ?ERROR ;

NFA_QUERYEXEC .CBYTE $85,"?EXEC"
LFA_QUERYEXEC .WORD NFA_QUERYCOMP
CFA_QUERYEXEC .WORD DOCOL
PFA_QUERYEXEC
.WORD CFA_STATE ; STATE
.WORD CFA_FETCH ; @
.WORD CFA_CLIT ; CLIT
.BYTE 18
.WORD CFA_QUERYERROR ; ?ERROR
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> ?PAIRS <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( ?PAIRS [ n1 n2 -> ] gibt eine entsprechende Fehlermeldung aus, wenn n1    )
;( und n2 ungleich sind. Das Wort wird waehrend der Kompilation benutzt, um  )
;( festzustellen, ob zwei strukturierte Sprachelemente zusammengehoeren.     )
;( ========================================================================== )
;
; : ?PAIRS - 19 ?ERROR ;

NFA_QUERYPAIRS .CBYTE $86,"?PAIRS"
LFA_QUERYPAIRS .WORD NFA_QUERYEXEC
CFA_QUERYPAIRS .WORD DOCOL
PFA_QUERYPAIRS
.WORD CFA_MINUS ; -
.WORD CFA_CLIT ; CLIT
.BYTE 19
.WORD CFA_QUERYERROR ; ?ERROR
.WORD CFA_EXIT ; ;S
```

```
; >>>>>>>>>>>>>>>> ?CSP <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( ?CSP [ ] gibt eine entsprechende Fehlermeldung aus, wenn die aktuelle     )
;( Stack-Position von der in der User-Variablen CSP geretteten abweicht.     )
;( ========================================================================== )
;
; : ?CSP SP@ CSP @ - 20 ?ERROR ;

NFA_QUERYCSP .CBYTE $84,"?CSP"
LFA_QUERYCSP .WORD NFA_QUERYPAIRS
CFA_QUERYCSP .WORD DOCOL
PFA_QUERYCSP
.WORD CFA_SPFETCH ; SP@
.WORD CFA_CSP ; CSP
.WORD CFA_FETCH ; @
.WORD CFA_MINUS ; -
.WORD CFA_CLIT ; CLIT
.BYTE 20
.WORD CFA_QUERYERROR ; ?ERROR
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> ?LOADING <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( ?LOADING [ ] gibt eine entsprechende Fehlermeldung aus, falls nicht       )
;( geladen wird.                                                             )
;( ========================================================================== )
;
; : ?LOADING SOURCE-ID @ 0= 22 ?ERROR ;

NFA_QUERYLOADING .CBYTE  $88,"?LOADING"
LFA_QUERYLOADING .WORD NFA_QUERYCSP
CFA_QUERYLOADING .WORD DOCOL
PFA_QUERYLOADING
.WORD CFA_SOURCEID ; SOURCE-ID
.WORD CFA_FETCH ; @
.WORD CFA_NULLEQUAL ; 0=
.WORD CFA_CLIT ; CLIT
.BYTE 22
.WORD CFA_QUERYERROR ; ?ERROR
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> COMPILE <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( COMPILE [ ] kompiliert das inline folgende Wort in das Dictionary.        )
;( ========================================================================== )
;
; : COMPILE
;  ?COMP     ( nur im Compile-Mode erlaubt                                  )
;  R>        ( Fortsetzungsadresse holen                                    )
;  DUP 2+ >R ( inline folgendes Wort in der Ausfuehrung ueberspringen       )
;  @ ,       ( inline folgendes Wort an das Dictionary anfuegen             )
; ;

NFA_COMPILE .CBYTE $87,"COMPILE"
LFA_COMPILE .WORD NFA_QUERYLOADING
CFA_COMPILE .WORD DOCOL
PFA_COMPILE
.WORD CFA_QUERYCOMP ; ?COMP
.WORD CFA_RPOP ; R>
```

```
.WORD CFA_DUP ; DUP
.WORD CFA_2PLUS ; 2+
.WORD CFA_RPUSH ; >R
.WORD CFA_FETCH ; @
.WORD CFA_COMMA ; ,
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> [ <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( [ [ ] schaltet in den Execute-Mode um.                                   )
;( ======================================================================== )
;
; : [ 0 STATE ! ; IMMEDIATE

NFA_LEFTBRACKET .CBYTE $81+$40, "[" ; IMMEDIATE
LFA_LEFTBRACKET .WORD NFA_COMPILE
CFA_LEFTBRACKET .WORD DOCOL
PFA_LEFTBRACKET
.WORD CFA_0 ; 0
.WORD CFA_STATE ; STATE
.WORD CFA_STORE ; !
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> ] <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( ] [ ] schaltet in den Compile-Mode um.                                   )
;( ======================================================================== )
;
; : ] 192 STATE ! ;

NFA_RIGHTBRACKET .CBYTE $81,"]"
LFA_RIGHTBRACKET .WORD NFA_LEFTBRACKET
CFA_RIGHTBRACKET .WORD DOCOL
PFA_RIGHTBRACKET
.WORD CFA_CLIT ; CLIT
.BYTE 192
.WORD CFA_STATE ; STATE
.WORD CFA_STORE ; !
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> SMUDGE <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( SMUDGE [ ] dreht das SMUDGE-Bit im Header des zuletzt definierten Wortes  )
;( um, d.h. macht das Wort gueltig, wenn es vorher ungueltig war, und        )
;( umgekehrt.                                                               )
;( ======================================================================== )
;
; : SMUDGE LATEST 32 TOGGLE ;

NFA_SMUDGE .CBYTE $86,"SMUDGE"
LFA_SMUDGE .WORD NFA_RIGHTBRACKET
CFA_SMUDGE .WORD DOCOL
PFA_SMUDGE
.WORD CFA_LATEST ; LATEST
.WORD CFA_CLIT ; CLIT
.BYTE 32
.WORD CFA_TOGGLE ; TOGGLE
.WORD CFA_EXIT ; ;S
```

```
; >>>>>>>>>>>>>>>> HEX <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( HEX [ ] setzt die I/O-Zahlenbasis auf 16.                                  )
;( ========================================================================== )
;
; : HEX 16 BASE ! ;

NFA_HEX .CBYTE $83,"HEX"
LFA_HEX .WORD NFA_SMUDGE
CFA_HEX .WORD DOCOL
PFA_HEX
.WORD CFA_CLIT ; CLIT
.BYTE 16
.WORD CFA_BASE ; BASE
.WORD CFA_STORE ; !
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> DECIMAL <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( DECIMAL [ ] setzt die I/O-Zahlenbasis auf 10.                              )
;( ========================================================================== )
;
; : DECIMAL 10 BASE ! ;

NFA_DECIMAL .CBYTE $87,"DECIMAL"
LFA_DECIMAL .WORD NFA_HEX
CFA_DECIMAL .WORD DOCOL
PFA_DECIMAL
.WORD CFA_CLIT ; CLIT
.BYTE 10
.WORD CFA_BASE ; BASE
.WORD CFA_STORE ; !
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> (;CODE) <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( [;CODE] [ ] ueberschreibt das Codefeld des soeben definierten Wortes mit   )
;( der unmittelbar auf [;CODE] folgenden Adresse und kehrt zum aufrufenden    )
;( Wort zurueck.                                                              )
;( ========================================================================== )
;
; : (;CODE) R> LATEST PFA CFA ! ;

NFA_BRACKETCODE .CBYTE $87,"(;CODE)"
LFA_BRACKETCODE .WORD NFA_DECIMAL
CFA_BRACKETCODE .WORD DOCOL
PFA_BRACKETCODE
.WORD CFA_RPOP ; R>
.WORD CFA_LATEST ; LATEST
.WORD CFA_PFA ; PFA
.WORD CFA_CFA ; CFA
.WORD CFA_STORE ; !
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> ;CODE <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( ;CODE [ ] beendet die Colon-Definition eines Definitionswortes und         )
;( beginnt die Spezifikationsphase des Assemblerteiles.                       )
;( ========================================================================== )
```

```
;
; : ;CODE
;  ?CSP              ( Stackpointer-Position ueberpruefen     )
;  COMPILE (;CODE)   ( [;CODE] kompilieren                    )
;  [COMPILE] [       ( in den Execute-Mode umschalten         )
;  SMUDGE            ( soeben definiertes Wort gueltig machen )
; ; IMMEDIATE

NFA_CODE .CBYTE $85+$40,";CODE" ; IMMEDIATE
LFA_CODE .WORD NFA_BRACKETCODE
CFA_CODE .WORD DOCOL
PFA_CODE
.WORD CFA_QUERYCSP ; ?CSP
.WORD CFA_COMPILE ; COMPILE
.WORD CFA_BRACKETCODE ; (;CODE)
.WORD CFA_LEFTBRACKET ; [
.WORD CFA_SMUDGE ; SMUDGE
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> <BUILDS <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( <BUILDS [ ] beginnt die Definitionsphase eines High-Level-Definitions-    )
;( wortes.                                                                   )
;( ========================================================================== )
;
; : <BUILDS 0 CONSTANT ;          ( provisorischen Konstanten-Header aufbauen )

NFA_BUILDS .CBYTE $87,"<BUILDS"
LFA_BUILDS .WORD NFA_CODE
CFA_BUILDS .WORD DOCOL
PFA_BUILDS
.WORD CFA_0 ; 0
.WORD CFA_CONSTANT ; CONSTANT
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> DOES> <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( DOES> [ ] beendet die Definitionsphase eines High-Level-Definitionswortes )
;( und leitet die Definitionsphase des entsprechenden Ausfuehrungsteils ein. )
;( ========================================================================== )
;
; : DOES>
;  R>              ( Adresse des auf DOES> folgenden High-Level-Codes      )
;  LATEST PFA !    ( in das Parameterfeld des soeben def. Wortes schreiben )
; ;CODE             ; das neue Wort fuehrt den folgenden Code aus:
;  DODOE LDA IP+1  ; Instruction-Pointer IP auf den Returnstack pushen
;  PHA
;  LDA IP
;  PHA
;  LDY #2
;  LDA (W),Y ; IP mit der Adresse laden, die als erstes im
;  STA IP    ; Parameterfeld des Wortes abgelegt ist.
;  INY       ; Dies ist die Adresse unmittelbar hinter dem DOES>
;  LDA (W),Y ; des definierenden Wortes.
;  STA IP+1
;  CLC
;  LDA W     ; Adresse des zweiten Eintrags im Parameterfeld des
;  ADC #4    ; Wortes holen
;  PHA
```

```
;  LDA W+1
;  ADC #0
;  JMP PUSH  ; auf den Datenstack legen und fertig
; END-CODE

NFA_DOES .CBYTE $85,"DOES>"
LFA_DOES .WORD NFA_BUILDS
CFA_DOES .WORD DOCOL
PFA_DOES
.WORD CFA_RPOP ; R>
.WORD CFA_LATEST ; LATEST
.WORD CFA_PFA ; PFA
.WORD CFA_STORE ; !
.WORD CFA_BRACKETCODE ; (;CODE)
            ; das neue Wort fuehrt den folgenden Code aus:
  DODOE LDA IP+1  ; Instruction-Pointer IP auf den Returnstack pushen
  PHA
  LDA IP
  PHA

  LDY #2
  LDA (W),Y ; IP mit der Adresse laden, die als erstes im
  STA IP    ; Parameterfeld des Wortes abgelegt ist.
  INY       ; Dies ist die Adresse unmittelbar hinter dem DOES>
  LDA (W),Y ; des definierenden Wortes.
  STA IP+1
  CLC
  LDA W     ; Adresse des zweiten Eintrags im Parameterfeld des
  ADC #4    ; Wortes holen
  PHA
  LDA W+1
  ADC #0
  JMP PUSH  ; auf den Datenstack legen und fertig

; >>>>>>>>>>>>>>>> COUNT <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( COUNT [ addr1 -> addr2 n ] ermittelt zum String, auf dessen Count-Byte    )
;( addr1 zeigt, die eigentliche Startadresse addr2 und die Anzahl n der      )
;( Zeichen.                                                                  )
;( ========================================================================== )
;
; : COUNT DUP 1+ SWAP C@ ;                      ( addr2=addr1+1 und n=[addr1] )

NFA_COUNT .CBYTE $85,"COUNT"
LFA_COUNT .WORD NFA_DOES
CFA_COUNT .WORD DOCOL
PFA_COUNT
.WORD CFA_DUP ; DUP
.WORD CFA_1PLUS ; 1+
.WORD CFA_SWAP ; SWAP
.WORD CFA_CFETCH ; C@
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> TYPE <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( TYPE [ addr count -> ] gibt den aus count Zeichen bestehenden String,     )
;( beginnend ab Adresse addr, auf die Standardausgabe aus.                   )
;( ========================================================================== )
;
```

```
; : TYPE
;  -DUP IF             ( falls count<>0, d.h. falls Zeichen auszugeben sind: )
;    OVER + SWAP DO    ( fuer alle I von addr bis addr+count-1:              )
;      I C@ EMIT       ( Zeichen in Adresse I ausgeben                       )
;    LOOP
;  ELSE                ( falls count=0:                                      )
;    DROP              ( nichts ausgeben, addr vom Stack nehmen              )
;  ENDIF
; ;

NFA_TYPE .CBYTE $84,"TYPE"
LFA_TYPE .WORD NFA_COUNT
CFA_TYPE .WORD DOCOL
PFA_TYPE
.WORD CFA_MINUSDUP ; -DUP
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL5-*
.WORD CFA_OVER ; OVER
.WORD CFA_PLUS ; +
.WORD CFA_SWAP ; SWAP
.WORD CFA_BRACKETDO ; (DO)
LBL6
.WORD CFA_I ; I
.WORD CFA_CFETCH ; C@
.WORD CFA_EMIT ; EMIT
.WORD CFA_BRACKETLOOP ; (LOOP)
.WORD LBL6-*
.WORD CFA_BRANCH ; BRANCH
.WORD LBL7-*
LBL5
.WORD CFA_DROP ; DROP
LBL7
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> -TRAILING <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( -TRAILING [ addr n1 -> addr n2 ] verkuerzt den durch addr und n1 bezeich- )
;( neten String um Blanks am Textende, d.h. die Zeichen in addr+n2 bis       )
;( addr+n1-1 sind Blanks.                                                    )
;( ======================================================================== )
;
; : -TRAILING
;  DUP 0 DO                ( wiederhole die folgende Schleife maximal n1-mal )
;    OVER OVER + 1 - C@    ( Zeichen an der Adresse addr+n-1                 )
;    BL - IF               ( falls es kein Blank ist:                        )
;      LEAVE               ( Suche beenden                                   )
;    ELSE                  ( sonst:                                          )
;      1 -                 ( n dekrementieren                                )
;    ENDIF
;  LOOP
; ;

NFA_MINUSTRAILING .CBYTE $89,"-TRAILING"
LFA_MINUSTRAILING .WORD NFA_TYPE
CFA_MINUSTRAILING .WORD DOCOL
PFA_MINUSTRAILING
.WORD CFA_DUP ; DUP
.WORD CFA_0 ; 0
.WORD CFA_BRACKETDO ; (DO)
```

```
LBL8
.WORD CFA_OVER ; OVER
.WORD CFA_OVER ; OVER
.WORD CFA_PLUS ; +
.WORD CFA_1 ; 1
.WORD CFA_MINUS ; -
.WORD CFA_CFETCH ; C@
.WORD CFA_BL ; BL
.WORD CFA_MINUS ; -
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL9-*
.WORD CFA_LEAVE ; LEAVE
.WORD CFA_BRANCH ; BRANCH
.WORD LBL10-*
LBL9
.WORD CFA_1 ; 1
.WORD CFA_MINUS ; -
LBL10
.WORD CFA_BRACKETLOOP ; (LOOP)
.WORD LBL8-*
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> (.") <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( [."] [ ] gibt den inline folgenden String auf die Standardausgabe aus.    )
;( ======================================================================== )
;
; : (.")
;  R COUNT          ( Adresse und Laenge des inline folgenden Strings        )
;  DUP 1+ R> + >R   ( Returnadresse um die Gesamtlaenge des Strings erhoehen )
;  TYPE             ( String auf die Standardausgabe ausgeben               )
; ;

NFA_BRAKETDOTQUOTE .CBYTE $84,"(.",34,")"
LFA_BRAKETDOTQUOTE .WORD NFA_MINUSTRAILING
CFA_BRAKETDOTQUOTE .WORD DOCOL
PFA_BRAKETDOTQUOTE
.WORD CFA_R ; R
.WORD CFA_COUNT ; COUNT
.WORD CFA_DUP ; DUP
.WORD CFA_1PLUS ; 1+
.WORD CFA_RPOP ; R>
.WORD CFA_PLUS ; +
.WORD CFA_RPUSH ; >R
.WORD CFA_TYPE ; TYPE
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> ." <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( ." [ ] gibt den im Eingabestrom folgenden String auf die Standardausgabe  )
;( aus. Im Compile-Mode werden [."] und der String kompiliert.               )
;( ======================================================================== )
;
; : ."
;  34                  ( Zeichen '"'                                 )
;  STATE @ IF          ( falls Compile-Mode:                         )
;   COMPILE (.")      ( [."] kompilieren                           )
;    WORD               ( folgenden String kompilieren              )
;    HERE C@ 1+ ALLOT   ( Dictionary-Pointer entsprechend erhoehen )
```

```
;  ELSE                 ( falls Execute-Mode:                      )
;    WORD               ( folgenden String holen                   )
;    HERE COUNT TYPE    ( und auf die Standardausgabe ausgeben     )
;  ENDIF
; ; IMMEDIATE

NFA_DOTQUOTE .BYTE $C2,".",$A2
LFA_DOTQUOTE .WORD NFA_BRAKETDOTQUOTE
CFA_DOTQUOTE .WORD DOCOL
PFA_DOTQUOTE
.WORD CFA_CLIT ; CLIT
.BYTE 34
.WORD CFA_STATE ; STATE
.WORD CFA_FETCH ; @
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL11-*
.WORD CFA_COMPILE ; COMPILE
.WORD CFA_BRAKETDOTQUOTE ; (.")
.WORD CFA_WORD ; WORD
.WORD CFA_CFETCH ; C@
.WORD CFA_1PLUS ; 1+
.WORD CFA_ALLOT ; ALLOT
.WORD CFA_BRANCH ; BRANCH
.WORD LBL12-*
LBL11
.WORD CFA_WORD ; WORD
.WORD CFA_COUNT ; COUNT
.WORD CFA_TYPE ; TYPE
LBL12
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> EXPECT <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( EXPECT [ addr count -> ] holt bis zu count Zeichen vom Keyboard - wenn    )
;( nicht vorher CR eingegeben wird - und legt sie ab Adresse addr ab, durch  )
;( ein oder mehrere Null-Bytes abgeschlossen.                                )
;( ========================================================================== )
;
; : EXPECT
;  OVER + OVER DO            ( fuer I von addr bis addr+count-1 tue:         )
;    KEY                     ( Taste holen                                   )
;    DUP 126 +ORIGIN @ = IF  ( falls externes Backspace:                     )
;      DROP 8                ( Taste durch internes Backspace ersetzen       )
;      OVER I =              ( 1 falls am linken Anschlag, sonst 0           )
;      DUP R> 2 - + >R       ( I um 1 [falls Anfang] oder 2 dekrementieren   )
;      -                     ( ASCII 8 [BS] oder ASCII 7 [BEL] zuruecklassen )
;    ELSE                    ( falls kein externes Backspace:                )
;      DUP 155 = IF          ( falls CR:                                     )
;  LEAVE DROP BL 0           ( Eingabe beenden, Blank ausgeben, 0 speichern  )
;      ELSE                  ( falls kein CR:                                )
;  DUP                       ( Zeichen ausgeben und speichern                )
;      ENDIF
;      I C!                  ( eingegebenes Zeichen speichern                )
;      0 I 1+ !              ( 0 in der darauffolgenden Zelle speichern      )
;    ENDIF
;    EMIT                    ( eingegebenes Zeichen ausgeben                 )
;  LOOP
;  DROP                      ( Adresse addr vergessen                        )
; ;
```

```
NFA_EXPECT .CBYTE $86,"EXPECT"
LFA_EXPECT .WORD NFA_DOTQUOTE
CFA_EXPECT .WORD DOCOL
PFA_EXPECT
.WORD CFA_OVER ; OVER
.WORD CFA_PLUS ; +
.WORD CFA_OVER ; OVER
.WORD CFA_BRACKETDO ; (DO)
LBL13
.WORD CFA_KEY ; KEY
.WORD CFA_DUP ; DUP
.WORD CFA_CLIT  ; CLIT
.BYTE $0E
.WORD CFA_PLUSORIGIN ; +ORIGIN
.WORD CFA_FETCH ; @
.WORD CFA_EQUAL ; =
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL14-*
.WORD CFA_DROP ; DROP
.WORD CFA_CLIT  ; CLIT
.BYTE 126
.WORD CFA_OVER ; OVER
.WORD CFA_I ; I
.WORD CFA_EQUAL ; =
.WORD CFA_DUP ; DUP
.WORD CFA_RPOP ; R>
.WORD CFA_2 ; 2
.WORD CFA_MINUS ; -
.WORD CFA_PLUS ; +
.WORD CFA_RPUSH ; >R
.WORD CFA_MINUS ; -
.WORD CFA_BRANCH ; BRANCH
.WORD LBL15-*
LBL14
.WORD CFA_DUP ; DUP
.WORD CFA_CLIT ; CLIT
.BYTE 155
.WORD CFA_EQUAL ; =
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL16-*
.WORD CFA_LEAVE ; LEAVE
.WORD CFA_DROP ; DROP
.WORD CFA_BL ; BL
.WORD CFA_0 ; 0
.WORD CFA_BRANCH ; BRANCH
.WORD LBL17-*
LBL16
.WORD CFA_DUP ; DUP
LBL17
.WORD CFA_I ; I
.WORD CFA_CSTORE ; C!
.WORD CFA_0 ; 0
.WORD CFA_I ; I
.WORD CFA_1PLUS ; 1+
.WORD CFA_STORE ; !
LBL15
.WORD CFA_EMIT ; EMIT
.WORD CFA_BRACKETLOOP ; (LOOP)
```

```
.WORD LBL13-*
.WORD CFA_DROP ; DROP
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> QUERY <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( QUERY [ ] holt bis zu 80 Zeichen - wenn nicht vorher CR eingegeben wird - )
;( vom Terminal und legt sie im Terminal-Input-Buffer TIB ab. Ausserdem wird )
;( die User-Variable IN auf 0 gesetzt.                                       )
;( ========================================================================== )
;
; : QUERY
;  TIB @ 80 EXPECT ( bis zu 80 Zeichen in den Terminal-Input-Buffer einlesen )
;  0 IN !          ( Interpretation beginnt beim Offset 0                    )
; ;

NFA_QUERY .CBYTE $85,"QUERY"
LFA_QUERY .WORD NFA_EXPECT
CFA_QUERY .WORD DOCOL
PFA_QUERY
.WORD CFA_TIB ; TIB
.WORD CFA_FETCH ; @
.WORD CFA_CLIT ; CLIT
.BYTE 80
.WORD CFA_EXPECT ; EXPECT
.WORD CFA_0 ; 0
.WORD CFA_IN ; IN
.WORD CFA_STORE ; !
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> X <<<<<<<<<<<<<<<<
; ( ========================================================================== )
; ( X [ ] bricht die Interpretation einer Zeile Text bedingungslos ab. X ist  )
; ( ein Pseudonym fuer den Namen, der nur aus dem ASCII-0-Zeichen besteht.    )
; ( ========================================================================== )
;
; ( 32897 HERE )                  ( hex 8081 und aktuelle Adresse merken )
;
; : X
;  SOURCE-ID @ IF                ( falls von Disk geladen wird:         )
;    1 BLK +!                    ( naechsten Block interpretieren       )
;    0 IN !                      ( beginnend beim Offset 0              )
;    BLK @ 0 B/SCR U/ DROP 0= IF ( falls Screen-Ende erreicht wurde:    )
;      ?EXEC R> DROP             ( Interpretation ganz abbrechen        )
;    ENDIF
;  ELSE                          ( falls nicht von Disk geladen wird:   )
;    R> DROP                     ( Interpretation ganz abbrechen        )
;  ENDIF
; ; IMMEDIATE

NFA_X .BYTE $C1, $80
LFA_X .WORD NFA_QUERY
CFA_X .WORD DOCOL
PFA_X
;.WORD CFA_BLK ; BLK
;.WORD CFA_FETCH ; @
;.WORD CFA_0BRANCH ; 0BRANCH
;.WORD LBL18-*
;.WORD CFA_1 ; 1
```

```
;.WORD CFA_BLK ; BLK
;.WORD CFA_PLUSSTORE ; +!
;.WORD CFA_0 ; 0
;.WORD CFA_IN ; IN
;.WORD CFA_STORE ; !
;.WORD CFA_BLK ; BLK
;.WORD CFA_FETCH ; @
;.WORD CFA_0 ; 0
;.WORD CFA_BSCR ; B/SCR
;.WORD CFA_UDIV ; U/
;.WORD CFA_DROP ; DROP
;.WORD CFA_NULLEQUAL ; 0=
;.WORD CFA_0BRANCH ; 0BRANCH
;.WORD LBL19-*
;.WORD CFA_QUERYEXEC ; ?EXEC
;.WORD CFA_RPOP ; R>
;.WORD CFA_DROP ; DROP
;LBL19
;.WORD CFA_BRANCH ; BRANCH
;.WORD LBL20-*
LBL18
.WORD CFA_RPOP ; R>
.WORD CFA_DROP ; DROP
LBL20
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> FILL <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( FILL [ addr n b -> ] fuellt n Bytes ab Adresse addr mit dem Wert b.       )
;( ======================================================================== )
;
; : FILL
;  SWAP >R OVER C!     ( Byte b in Adresse addr speichern                    )
;  DUP 1+ R> 1 - CMOVE ( n-1 Bytes aufsteigend von addr nach addr+1 kopieren )
; ;

NFA_FILL .CBYTE $84,"FILL"
LFA_FILL .WORD NFA_X
CFA_FILL .WORD DOCOL
PFA_FILL
.WORD CFA_SWAP ; SWAP
.WORD CFA_RPUSH ; >R
.WORD CFA_OVER ; OVER
.WORD CFA_CSTORE ; C!
.WORD CFA_DUP ; DUP
.WORD CFA_1PLUS ; 1+
.WORD CFA_RPOP ; R>
.WORD CFA_1 ; 1
.WORD CFA_MINUS ; -
.WORD CFA_CMOVE ; CMOVE
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> ERASE <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( ERASE [ addr n -> ] fuellt n Bytes ab Adresse addr mit Nullen.           )
;( ======================================================================== )
;
;: ERASE 0 FILL ;
```

```
NFA_ERASE .CBYTE $85,"ERASE"
LFA_ERASE .WORD NFA_FILL
CFA_ERASE .WORD DOCOL
PFA_ERASE
.WORD CFA_0 ; 0
.WORD CFA_FILL ; FILL
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> BLANKS <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( BLANKS [ addr count -> ] fuellt count Bytes ab Adresse addr mit Blanks.   )
;( ========================================================================== )
;
; : BLANKS BL FILL ;

NFA_BLANKS .CBYTE $86,"BLANKS"
LFA_BLANKS .WORD NFA_ERASE
CFA_BLANKS .WORD DOCOL
PFA_BLANKS
.WORD CFA_BL ; BL
.WORD CFA_FILL ; FILL
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> HOLD <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( HOLD [ c -> ] fuegt das Zeichen c in einen Ziffernstring ein. Das Wort    )
;( wird in Verbindung mit <# und #> gebraucht.                               )
;( ========================================================================== )
;
;: HOLD
;  -1 HLD +!   ( Zeiger im Ziffernstring weitersetzen                )
;  HLD @ C!    ( Zeichen c in das Byte, auf das HLD zeigt, speichern )
; ;

NFA_HOLD .CBYTE $84,"HOLD"
LFA_HOLD .WORD NFA_BLANKS
CFA_HOLD .WORD DOCOL
PFA_HOLD
.WORD CFA_LIT ; LIT
.WORD -1
.WORD CFA_HLD ; HLD
.WORD CFA_PLUSSTORE ; +!
.WORD CFA_HLD ; HLD
.WORD CFA_FETCH ; @
.WORD CFA_CSTORE ; C!
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> PAD <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( PAD [ -> addr ] uebergibt die Startadresse des Text-Ausgabe-Buffers PAD.  )
;( ========================================================================== )
;
; : PAD HERE 68 + ;

NFA_PAD .CBYTE $83, "PAD"
LFA_PAD .WORD NFA_HOLD
CFA_PAD .WORD DOCOL
PFA_PAD
.WORD CFA_HERE ; HERE
```

```
.WORD CFA_CLIT ; CLIT
.BYTE 68
.WORD CFA_PLUS ; +
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> WORD <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( WORD [ c -> c-addr ] bringt das im Input-Strom folgende, durch das        )
;( Zeichen c abgeschlossene Wort als String nach HERE. Der String beginnt    )
;( mit einem Count-Byte.                                                     )
;( ========================================================================== )
;
;: WORD
;  TIB @            ( Adresse des Terminal-Input-Buffers           )
;  IN @ +           ( Adresse, ab der gelesen wird                 )
;  SWAP ENCLOSE     ( Wort separieren                              )
;  HERE 34 BLANKS   ( Bereich HERE ... HERE+33 mit Blanks fuellen  )
;  IN +!            ( IN zeigt jetzt auf Offset hinter dem Wort    )
;  OVER - >R        ( Wortlaenge auf den Return-Stack legen        )
;  R HERE C!        ( Wortlaenge als Count-Byte nach HERE speichern )
;  +                ( Startadresse des separierten Wortes          )
;  HERE 1+          ( Zieladresse des Wortes                       )
;  R>               ( Anzahl der zu kopierenden Zeichen            )
;  CMOVE            ( separiertes Wort kopieren                    )
;  HERE             ( HERE auf dem Stack hinterlassen              )
; ;

NFA_WORD .CBYTE $84, "WORD"
LFA_WORD .WORD NFA_PAD
CFA_WORD .WORD DOCOL
PFA_WORD
.WORD CFA_TIB       ; TIB
.WORD CFA_FETCH     ; @
.WORD CFA_IN        ; IN
.WORD CFA_FETCH     ; @
.WORD CFA_PLUS      ; +
.WORD CFA_SWAP      ; SWAP
.WORD CFA_ENCLOSE   ; ENCLOSE
.WORD CFA_HERE      ; HERE
.WORD CFA_CLIT      ; CLIT
.BYTE 34
.WORD CFA_BLANKS    ; BLANKS
.WORD CFA_IN        ; IN
.WORD CFA_PLUSSTORE ; +!
.WORD CFA_OVER      ; OVER
.WORD CFA_MINUS     ; -
.WORD CFA_RPUSH     ; >R
.WORD CFA_R         ; R
.WORD CFA_HERE      ; HERE
.WORD CFA_CSTORE    ; C!
.WORD CFA_PLUS      ; +
.WORD CFA_HERE      ; HERE
.WORD CFA_1PLUS     ; 1+
.WORD CFA_RPOP      ; R>
.WORD CFA_CMOVE     ; CMOVE
.WORD CFA_HERE      ; HERE
.WORD CFA_EXIT      ; ;S

; >>>>>>>>>>>>>>>> (NUMBER) <<<<<<<<<<<<<<<<
```

```
;( ========================================================================== )
;( [NUMBER] [ d1 addr1 -> d2 addr2 ] akkumuliert den Ziffernstring ab        )
;( Adresse addr1+1 zur doppeltgenauen, vorzeichenlosen  Zahl d2 auf. addr2   )
;( ist die Adresse des ersten gefundenen Zeichens, das keine gueltige Ziffer )
;( darstellt.                                                                )
;( ========================================================================== )
;
; : (NUMBER)
;  BEGIN
;    1+               ( auf naechstes Zeichen zeigen                    )
;    DUP >R           ( diesen Zeiger auf dem Return-Stack merken        )
;    C@               ( Zeichen holen                                   )
;    BASE @ DIGIT     ( und in Ziffer umwandeln                         )
;  WHILE              ( solange es sich um eine gueltige Ziffer handelt: )
;    SWAP BASE @ U*   ( Hi-Word von d mit der Basis multiplizieren      )
;    DROP             ( Hi-Word dieses Produktes vergessen              )
;    ROT BASE @ U*    ( Lo-Word von d mit der Basis multiplizieren      )
;    D+               ( Summe Basis*d+n berechnen                       )
;    DPL @ 1+ IF      ( falls DPL nicht den Wert -1 enthaelt:           )
;      1 DPL +!       ( DPL inkrementieren                              )
;    ENDIF
;    R>               ( neuen Zeiger vom Return-Stack zurueckholen      )
;  REPEAT             ( wiederholen                                     )
;  R>                 ( falls fertig: Zeiger vom Returnstack holen      )
; ;

NFA_PARENTNUMBER .CBYTE $88,"(NUMBER)"
LFA_PARENTNUMBER .WORD NFA_WORD
CFA_PARENTNUMBER .WORD DOCOL
PFA_PARENTNUMBER
LBL23
.WORD CFA_1PLUS ; 1+
.WORD CFA_DUP ; DUP
.WORD CFA_RPUSH ; >R
.WORD CFA_CFETCH ; C@
.WORD CFA_BASE ; BASE
.WORD CFA_FETCH ; @
.WORD CFA_DIGIT; DIGIT
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL24-*
.WORD CFA_SWAP ; SWAP
.WORD CFA_BASE ; BASE
.WORD CFA_FETCH ; @
.WORD CFA_UMULT ; U*
.WORD CFA_DROP ; DROP
.WORD CFA_ROT ; ROT
.WORD CFA_BASE ; BASE
.WORD CFA_FETCH ; @
.WORD CFA_UMULT ; U*
.WORD CFA_DPLUS ; D+
.WORD CFA_DPL ; DPL
.WORD CFA_FETCH ; @
.WORD CFA_1PLUS ; 1+
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL25-*
.WORD CFA_1 ; 1
.WORD CFA_DPL ; DPL
.WORD CFA_PLUSSTORE ; +!
LBL25
```

```
.WORD CFA_RPOP ; R>
.WORD CFA_BRANCH ; BRANCH
.WORD LBL23-*
LBL24
.WORD CFA_RPOP ; R>
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> NUMBER <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( NUMBER [ addr -> d ] wandelt den String, auf dessen Count-Byte addr       )
;( zeigt, in eine doppeltgenaue Zahl d um. Das Count-Byte selbst wird nicht  )
;( beruecksichtigt, der Ziffernstring muss durch ein Blank abgeschlossen     )
;( sein.                                                                     )
;( ======================================================================== )
;
; : NUMBER
;  0 0 ROT                  ( Anfangsbedingungen fuer [NUMBER] schaffen      )
;  DUP 1+ C@                ( erstes Zeichen nach dem Count-Byte holen       )
;  45 =                     ( Vorzeichenflag: 1 falls Minuszeichen, sonst 0  )
;  DUP >R                   ( Vorzeichenflag auf den Returnstack legen       )
;  +                        ( falls Vorzeichen: Adresse inkrementieren       )
;  -1                       ( noch kein Dezimalpunkt gefunden                )
;  BEGIN                    ( Ziffern und Punkte einlesen und umwandeln:     )
;    DPL !                  ( Dezimalpunkt-Position in DPL speichern         )
;    (NUMBER)               ( folgende Ziffern in d aufakkumulieren          )
;    DUP C@ BL -            ( Abbruch der Schleife, falls Blank angetroffen  )
;  WHILE                    ( sonst weitermachen                             )
;    DUP C@  46 - 0 ?ERROR  ( Fehler falls kein Punkt                        )
;    0                      ( Punkt gefunden, dann 0 in DPL speichern [s.o.] )
;  REPEAT
;  DROP                     ( Adresse vergessen                              )
;  R> IF                    ( falls Minuszeichen am Anfang:                  )
;    DMINUS                 ( Zahl negieren                                  )
;  ENDIF
; ;

NFA_NUMBER .CBYTE $86,"NUMBER"
LFA_NUMBER .WORD NFA_PARENTNUMBER
CFA_NUMBER .WORD DOCOL
PFA_NUMBER
.WORD CFA_0 ; 0
.WORD CFA_0 ; 0
.WORD CFA_ROT ; ROT
.WORD CFA_DUP ; DUP
.WORD CFA_1PLUS ; 1+
.WORD CFA_CFETCH ; C@
.WORD CFA_CLIT ; CLIT
.BYTE 45
.WORD CFA_EQUAL ; =
.WORD CFA_DUP ; DUP
.WORD CFA_RPUSH ; >R
.WORD CFA_PLUS ; +
.WORD CFA_LIT ; LIT
.WORD -1
LBL26
.WORD CFA_DPL ; DPL
.WORD CFA_STORE ; !
.WORD CFA_PARENTNUMBER ; (NUMBER)
.WORD CFA_DUP ; DUP
```

```
.WORD CFA_CFETCH ; C@
.WORD CFA_BL ; BL
.WORD CFA_MINUS ; -
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL27-*
.WORD CFA_DUP ; DUP
.WORD CFA_CFETCH ; C@
.WORD CFA_CLIT ; CLIT
.BYTE 46
.WORD CFA_MINUS ; -
.WORD CFA_0 ; 0
.WORD CFA_QUERYERROR ; ?ERROR
.WORD CFA_0 ; 0
.WORD CFA_BRANCH ; BRANCH
.WORD LBL26-*
LBL27
.WORD CFA_DROP ; DROP
.WORD CFA_RPOP ; R>
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL28-*
.WORD CFA_DMINUS ; DMINUS
LBL28
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> -FIND <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( -FIND [ -> pfa b tf ] falls Suche erfolgreich, sonst [ -> ff ]. Liest das )
;( naechste Wort und bringt es, mit Count-Byte versehen, nach HERE.          )
;( Anschliessend wird das CONTEXT- und das CURRENT-Vokabular nach dem Wort   )
;( abgesucht. Wird es gefunden, so wird seine PFA, das Count-Byte b und ein  )
;( true-Flag uebergeben, sonst lediglich ein false-Flag.                    )
;( ========================================================================== )
;
; : -FIND
;  BL WORD                  ( naechstes Wort lesen und nach HERE bringen )
;  HERE CONTEXT @ @ (FIND)  ( CONTEXT-Vokabular nach dem Wort absuchen   )
;  DUP 0= IF                ( falls nicht gefunden:                      )
;    DROP                   ( false-Flag vergessen                       )
;    HERE LATEST (FIND)     ( CURRENT-Vokabular nach dem Wort absuchen   )
;  ENDIF
; ;

NFA_MINUSFIND .CBYTE $85,"-FIND"
LFA_MINUSFIND .WORD NFA_NUMBER
CFA_MINUSFIND .WORD DOCOL
PFA_MINUSFIND
.WORD CFA_BL ; BL
.WORD CFA_WORD ; WORD
.WORD CFA_CONTEXT ; CONTEXT
.WORD CFA_FETCH ; @
.WORD CFA_FETCH ; @
.WORD CFA_BRACKETFIND  ; (FIND)
.WORD CFA_DUP ; DUP
.WORD CFA_NULLEQUAL ; 0=
.WORD CFA_0BRANCH ; 0BRACH
.WORD LBL29-*
.WORD CFA_DROP ; DROP
.WORD CFA_HERE ; HERE
.WORD CFA_LATEST ; LATEST
```

```
.WORD CFA_BRACKETFIND  ; (FIND)
LBL29
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> (ABORT) <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( [ABORT] [ ] wird im Anschluss an eine Fehlermeldung ausgefuehrt, wenn die )
;( User-Variable WARNING den Wert -1 hat. Es wird lediglich das Wort ABORT   )
;( aufgerufen.                                                               )
;( ========================================================================== )
;
; : (ABORT) ABORT ;

NFA_PARENTABORT .CBYTE $87,"(ABORT)"
LFA_PARENTABORT .WORD NFA_MINUSFIND
CFA_PARENTABORT .WORD DOCOL
PFA_PARENTABORT
.WORD CFA_ABORT ; ABORT
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> ERROR <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( ERROR [ n -> in blk ] gibt je nach Inhalt der User-Variablen WARNING den  )
;( Text der Zeile n relativ zu Zeile 0 in Screen 4 aus [WARNING=1] oder      )
;( lediglich eine verkuerzte Fehlermeldung unter Angabe der Fehlernummer n   )
;( [WARNING=0]. Ist WARNING=-1, so wird [ABORT] aufgerufen. Schliesslich     )
;( wird der Inhalt der User-Variablen IN und BLK zurueckgegeben.             )
;( ========================================================================== )
; : ERROR
;   WARNING @ 0< IF (ABORT) THEN
;   HERE COUNT TYPE ."?" MESSAGE SP!
;   IN @ BLK @ QUIT ;

NFA_ERROR .CBYTE $85, "ERROR"
LFA_ERROR .WORD NFA_PARENTABORT
CFA_ERROR .WORD DOCOL
PFA_ERROR
.WORD CFA_WARNING ; WARNING
.WORD CFA_FETCH ; @
.WORD CFA_LTNULL ; 0<
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL30-*
.WORD CFA_PARENTABORT ; (ABORT)
LBL30
.WORD CFA_HERE ; HERE
.WORD CFA_COUNT ; COUNT
.WORD CFA_TYPE ; TYPE
.WORD CFA_BRAKETDOTQUOTE ; (.")
.BYTE 2
.BYTE 63
.BYTE 32
.WORD CFA_MESSAGE ; MESSAGE
.WORD CFA_SPSTORE ; SP!
;.WORD CFA_IN ; IN      ; auskommentiert da wir keine blockbefehle haben
;.WORD CFA_FETCH ; @    ; to do --> umsetzen fuer Dateizugriff
;.WORD CFA_BLK ; BLK
;.WORD CFA_FETCH ; @
.WORD CFA_QUIT ; QUIT
.WORD CFA_EXIT ; ;S
```

```
; >>>>>>>>>>>>>>>> ID. <<<<<<<<<<<<<<<
;( ======================================================================== )
;( ID. [ addr -> ] kopiert das Namenfeld des Wortes, auf dessen NFA addr     )
;( zeigt, nach PAD und gibt den Namen, gefolgt von einem Blank, auf die      )
;( Standardausgabe aus.                                                      )
;( ======================================================================== )
;
;( Die mit [*] gekennzeichneten Zeilen sind hinzugefuegt worden, damit auch  )
;( auf Rechnern mit einem erweiterten 8-Bit-Zeichensatz der Name korrekt     )
;( ausgegeben wird.                                                          )
;
; : ID.
;  PAD 32 95 FILL          ( PAD mit 32 Underscores fuellen                  )
;  DUP PFA LFA             ( LFA des Wortes ermitteln                        )
;  OVER -                  ( Differenz mit NFA ergibt Laenge des Namenfeldes )
;  SWAP OVER               ( Laenge noch mal retten [*]                      )
;  PAD SWAP CMOVE          ( gesamtes Namenfeld nach PAD kopieren            )
;  1 - PAD + 128 TOGGLE    ( MS-Bit des letzten Zeichens loeschen [*]        )
;  PAD COUNT 31 AND TYPE   ( Namen mit korrigiertem Count-Byte ausgeben      )
;  SPACE                   ( Blank ausgeben                                  )
; ;

NFA_IDDOT .CBYTE $83,"ID."
LFA_IDDOT .WORD NFA_ERROR
CFA_IDDOT .WORD DOCOL
PFA_IDDOT
.WORD CFA_PAD ; PAD
.WORD CFA_CLIT ; CLIT
.BYTE 32
.WORD CFA_CLIT ; CLIT
.BYTE 95
.WORD CFA_FILL ; FILL
.WORD CFA_DUP ; DUP
.WORD CFA_PFA ; PFA
.WORD CFA_LFA ; LFA
.WORD CFA_OVER ; OVER
.WORD CFA_MINUS ; -
.WORD CFA_SWAP ; SWAP
.WORD CFA_OVER ; OVER
.WORD CFA_PAD ; PAD
.WORD CFA_SWAP ; SWAP
.WORD CFA_CMOVE ; CMOVE
.WORD CFA_1 ; 1
.WORD CFA_MINUS ; -
.WORD CFA_PAD ; PAD
.WORD CFA_PLUS ; +
.WORD CFA_CLIT ; CLIT
.BYTE 128
.WORD CFA_TOGGLE ; TOGGLE
.WORD CFA_PAD ; PAD
.WORD CFA_COUNT ; COUNT
.WORD CFA_CLIT ; CLIT
.BYTE 31
.WORD CFA_AND ; AND
.WORD CFA_TYPE ; TYPE
.WORD CFA_SPACE ; SPACE
.WORD CFA_EXIT ; ;S
```

```
; >>>>>>>>>>>>>>>> CREATE <<<<<<<<<<<<<<<<
;( ============================================================================ )
;( CREATE [ ] erzeugt einen Wort-Header im Dictionary.                          )
;( ============================================================================ )
;
; : CREATE
;  TIB HERE 160 + U< 2 ?ERROR ( nur 6502: Fehler falls kein Platz mehr       )
;  -FIND IF                   ( Namenstring ablegen; falls schon vorhanden:  )
;    DROP NFA ID.             ( Namen ausgeben                               )
;    4 MESSAGE SPACE          ( Fehlermeldung 4 und Blank ausgeben           )
;  ENDIF
;  HERE DUP C@ WIDTH @ MIN    ( Laenge des evtl. gekuerzten Namen            )
;  1+ ALLOT                   ( plus Count-Byte: ins Dictionary einverleiben )
;  DP C@ 253 = ALLOT          ( nur 6502: Codefeld darf nicht auf FF liegen  )
;  DUP 160 TOGGLE             ( hoechstes und SMUDGE-Bit setzen              )
;  HERE 1 - 128 TOGGLE        ( hoechstes Bit im letzten Buchstaben setzen   )
;  LATEST ,                   ( Link-Feld auf vorheriges Wort zeigen lassen  )
;  CURRENT @ !                ( Current-Kette beginnt b. NFA d. neuen Wortes )
;  HERE 2+ ,                  ( naechste Adresse in das Codefeld schreiben   )
; ;

NFA_CREATE .CBYTE $86,"CREATE"
LFA_CREATE .WORD NFA_IDDOT
CFA_CREATE .WORD DOCOL
PFA_CREATE
.WORD CFA_TIB ; TIB
.WORD CFA_HERE ; HERE
.WORD CFA_CLIT ; CLIT
.BYTE 160
.WORD CFA_PLUS ; +
.WORD CFA_ULT ; U<
.WORD CFA_2 ; 2
.WORD CFA_QUERYERROR ; ?ERROR
.WORD CFA_MINUSFIND ; -FIND
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL31-*
.WORD CFA_DROP ; DROP
.WORD CFA_NFA ; NFA
.WORD CFA_IDDOT ; ID.
.WORD CFA_CLIT ; CLIT
.BYTE 4
.WORD CFA_MESSAGE ; MESSAGE
.WORD CFA_SPACE ; SPACE
LBL31
.WORD CFA_HERE ; HERE
.WORD CFA_DUP ; DUP
.WORD CFA_CFETCH ; C@
.WORD CFA_WIDTH ; WIDTH
.WORD CFA_FETCH ; @
.WORD CFA_MIN ; MIN
.WORD CFA_1PLUS ; 1+
.WORD CFA_ALLOT ; ALLOT
.WORD CFA_DP ; DP
.WORD CFA_CFETCH ; C@
.WORD CFA_CLIT ; CLIT
.BYTE 253
.WORD CFA_EQUAL ; =
.WORD CFA_ALLOT ; ALLOT
.WORD CFA_DUP ; DUP
```

```
.WORD CFA_CLIT ; CLIT
.BYTE 160
.WORD CFA_TOGGLE ; TOGGLE
.WORD CFA_HERE ; HERE
.WORD CFA_1 ; 1
.WORD CFA_MINUS ; -
.WORD CFA_CLIT ; CLIT
.BYTE 128
.WORD CFA_TOGGLE ; TOGGLE
.WORD CFA_LATEST ; LATEST
.WORD CFA_COMMA ; ,
.WORD CFA_CURRENT ; CURRENT
.WORD CFA_FETCH ; @
.WORD CFA_STORE ; !
.WORD CFA_HERE ; HERE
.WORD CFA_2PLUS ; 2+
.WORD CFA_COMMA ; ,
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> [COMPILE] <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( [COMPILE] [ ] kompiliert das im Input-Stream folgende Wort in das         )
;( Dictionary.                                                               )
;( ======================================================================== )
;
; : [COMPILE]
;  -FIND 0= 0 ?ERROR    ( Fehlermeldung ausgeben, falls Wort nicht vorhanden )
;  DROP CFA ,           ( sonst CFA des Wortes in das Dictionary kompilieren )
; ; IMMEDIATE

NFA_BRACKETCOMPILE .CBYTE $89+$40,"[COMPILE]" ; IMMEDIATE
LFA_BRACKETCOMPILE .WORD NFA_CREATE
CFA_BRACKETCOMPILE .WORD DOCOL
PFA_BRACKETCOMPILE
.WORD CFA_MINUSFIND ; -FIND
.WORD CFA_NULLEQUAL ; 0=
.WORD CFA_0 ; 0
.WORD CFA_QUERYERROR ; ?ERROR
.WORD CFA_DROP ; DROP
.WORD CFA_CFA ; CFA
.WORD CFA_COMMA ; ,
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> LITERAL <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( LITERAL [ n -> n ] falls Execute-Mode, [ n -> ] falls Compile-Mode.       )
;( Im Compile-Mode wird die Zahl n mitsamt einem LIT in das Dictionary       )
;( kompiliert. Im Execute-Mode hat der Aufruf keinen Effekt.                 )
;( ======================================================================== )
;
; : LITERAL
;  STATE @ IF      ( falls Compile-Mode:                    )
;    COMPILE LIT   ( Wort LIT in das Dictionary kompilieren )
;    ,             ( Zahl n in das Dictionary kompilieren   )
;  ENDIF
; ; IMMEDIATE

NFA_LITERAL .CBYTE $87+$40,"LITERAL" ; IMMEDIATE
LFA_LITERAL .WORD NFA_BRACKETCOMPILE
```

```
CFA_LITERAL .WORD DOCOL
PFA_LITERAL
.WORD CFA_STATE ; STATE
.WORD CFA_FETCH ; @
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL32-*
.WORD CFA_COMPILE ; COMPILE
.WORD CFA_LIT ; LIT
.WORD CFA_COMMA ; ,
LBL32
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> DLITERAL <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( DLITERAL [ d -> d ] falls Execute-Mode, [ d -> ] falls Compile-Mode.      )
;( Im Compile-Mode wird die doppeltgenaue Zahl d mitsamt entsprechenden      )
;( LITs in das Dictionary kompiliert. Im Execute-Mode hat der Aufruf keinen  )
;( Effekt.                                                                   )
;( ========================================================================== )
;
; : DLITERAL
;  STATE @ IF            ( falls Compile-Mode:               )
;    SWAP                ( Hi- und Lo-Word von d vertauschen )
;    [COMPILE] LITERAL   ( LIT und Lo-Word von d kompilieren )
;    [COMPILE] LITERAL   ( LIT und Hi-Word von d kompilieren )
;  ENDIF
; ; IMMEDIATE

NFA_DLITERAL .CBYTE $88+$40,"DLITERAL" ; IMMEDIATE
LFA_DLITERAL .WORD NFA_LITERAL
CFA_DLITERAL .WORD DOCOL
PFA_DLITERAL
.WORD CFA_STATE ; STATE
.WORD CFA_FETCH ; @
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL33-*
.WORD CFA_SWAP ; SWAP
.WORD CFA_LITERAL ; LITERAL
.WORD CFA_LITERAL ; LITERAL
LBL33
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> ?STACK <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( ?STACK [ ] gibt eine Fehlermeldung aus, falls der Stack ueber seine       )
;( Grenzen gewachsen ist.                                                    )
;( ========================================================================== )
;
; : ?STACK
;  SP@ ASM TOS SWAP U< 1 ?ERROR   ( Fehlermeldung falls Stack-Unterlauf )
;  SP@ ASM BOS U< 7 ?ERROR   ( Fehlermeldung falls Stack-Ueberlauf )
; ;

NFA_QUERYSTACK .CBYTE $86,"?STACK"
LFA_QUERYSTACK .WORD NFA_DLITERAL
CFA_QUERYSTACK .WORD DOCOL
PFA_QUERYSTACK
.WORD CFA_SPFETCH ; SP@
.WORD CFA_LIT ; LIT
```

```
.WORD TOS
.WORD CFA_SWAP ; SWAP
.WORD CFA_ULT ; U<
.WORD CFA_1 ; 1
.WORD CFA_QUERYERROR ; ?ERROR
.WORD CFA_SPFETCH ; SP@
.WORD CFA_LIT ; LIT
.WORD BOS
.WORD CFA_ULT ; U<
.WORD CFA_CLIT ; CLIT
.BYTE 7
.WORD CFA_QUERYERROR ; ?ERROR
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> INTERPRET <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( INTERPRET [ ] ist der Kern des aeusseren Interpreters. Text wird geholt   )
;( und, je nach Systemzustand, ausgefuehrt oder kompiliert. ASCII Null been- )
;( det die Prozedur.                                                         )
;( ========================================================================== )
;
; : INTERPRET
;  BEGIN                          ( Endlosschleife                          )
;    -FIND IF                     ( Wort lesen; falls vorhanden:            )
;      STATE @ < IF               ( falls Compile-Mode und nicht IMMEDIATE: )
;  CFA ,                          ( CFA des Wortes kompilieren              )
;      ELSE                       ( sonst:                                  )
;  CFA EXECUTE                    ( Wort ausfuehren                         )
;      ENDIF
;      ?STACK                     ( Stack ueberpruefen                      )
;    ELSE                         ( falls Wort nicht vorhanden:             )
;      HERE NUMBER                ( Wort in Zahl umwandeln                  )
;      DPL @ 1+ IF                ( falls Dezimalpunkt vorhanden:           )
;  [COMPILE] DLITERAL             ( doppelt-genaue Zahl kompilieren         )
;      ELSE                       ( falls kein Dezimalpunkt vorhanden:      )
;  DROP [COMPILE] LITERAL         ( einfach genaue Zahl kompilieren         )
;      ENDIF
;      ?STACK                     ( Stack ueberpruefen                      )
;    ENDIF
;  AGAIN                          ( Ende der Endlosschleife                 )
; ;

NFA_INTERPRET .CBYTE $89,"INTERPRET"
LFA_INTERPRET .WORD NFA_QUERYSTACK
CFA_INTERPRET .WORD DOCOL
PFA_INTERPRET
LBL34
.WORD CFA_MINUSFIND ; -FIND
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL35-*
.WORD CFA_STATE ; STATE
.WORD CFA_FETCH ; @
.WORD CFA_LT ; <
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL36-*
.WORD CFA_CFA ; CFA
.WORD CFA_COMMA ; ,
.WORD CFA_BRANCH ; BRANCH
.WORD LBL37-*
```

```
LBL36
.WORD CFA_CFA ; CFA
.WORD CFA_EXECUTE ; EXECUTE
LBL37
.WORD CFA_QUERYSTACK ; ?STACK
.WORD CFA_BRANCH ; BRANCH
.WORD LBL38-*
LBL35
.WORD CFA_HERE ; HERE
.WORD CFA_NUMBER ; NUMBER
.WORD CFA_DPL ; DPL
.WORD CFA_FETCH ; @
.WORD CFA_1PLUS ; 1+
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL39-*
.WORD CFA_DLITERAL ; DLITERAL
.WORD CFA_BRANCH ; BRANCH
.WORD LBL40-*
LBL39
.WORD CFA_DROP ; DROP
.WORD CFA_LITERAL ; LITERAL
LBL40
.WORD CFA_QUERYSTACK ; ?STACK
LBL38
.WORD CFA_BRANCH ; BRANCH
.WORD LBL34-*
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> IMMEDIATE <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( IMMEDIATE [ ] dreht das Precedence-Bit des zuletzt definierten Wortes um. )
;( ======================================================================== )
;
; : IMMEDIATE LATEST 64 TOGGLE ;

NFA_IMMEDIATE .CBYTE $89, "IMMEDIATE"
LFA_IMMEDIATE .WORD NFA_INTERPRET
CFA_IMMEDIATE .WORD DOCOL
PFA_IMMEDIATE
.WORD CFA_LATEST ; LATEST
.WORD CFA_CLIT ; CLIT
.BYTE 64
.WORD CFA_TOGGLE ; TOGGLE
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> VOCABULARY <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( VOCABULARY [ ] definiert ein Vokabular.                                  )
;( ======================================================================== )
;
; : VOCABULARY
;  <BUILDS             ( provisorischen Header des Vokabulars aufbauen       )
;    -24447 ,          ( Header 81 A0 eines Dummy-Wortes " " aufbauen        )
;    CURRENT @ CFA ,   ( NFA des letzten " "-Dummys in LFA von " " eintragen )
;    HERE              ( aktueller Wert von DP = CFA von " "                 )
;    VOC-LINK @ ,      ( Link zum letzten Vokabular in CFA von " " eintragen )
;    VOC-LINK !        ( CFA von " " in der User-Variablen VOC-LINK sichern  )
;  DOES>               ( das neue Vokabular-Wort tut bei Aufruf folgendes:   )
;  2+ CONTEXT !        ( LFA von " " in CONTEXT ablegen                      )
```

```
; ;

NFA_VOCABULARY .CBYTE $8A, "VOCABULARY"
LFA_VOCABULARY .WORD NFA_IMMEDIATE
CFA_VOCABULARY .WORD DOCOL
PFA_VOCABULARY
.WORD CFA_BUILDS ; <BUILDS
.WORD CFA_LIT ; LIT
.WORD -24447
.WORD CFA_COMMA ; ,
.WORD CFA_CURRENT ; CURRENT
.WORD CFA_FETCH ; @
.WORD CFA_CFA ; CFA
.WORD CFA_COMMA ; ,
.WORD CFA_HERE ; HERE
.WORD CFA_VOCLINK ; VOC-LINK
.WORD CFA_FETCH ; @
.WORD CFA_COMMA ; ,
.WORD CFA_VOCLINK ; VOC-LINK
.WORD CFA_STORE ; !
.WORD CFA_DOES ; DOES>
DOES156
.WORD CFA_2PLUS ; 2+
.WORD CFA_CONTEXT ; CONTEXT
.WORD CFA_STORE ; !
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> FORTH <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( FORTH [ ] macht das FORTH-Vokabular zum CONTEXT-Vokabular.                 )
;( ========================================================================== )
;
; VOCABULARY FORTH IMMEDIATE
;
; FORTH DEFINITIONS

NFA_FORTH .CBYTE $85+$40, "FORTH"
LFA_FORTH .WORD NFA_VOCABULARY
CFA_FORTH .WORD DODOE
PFA_FORTH
.WORD DOES156
.BYTE 129
.BYTE 160
.WORD NFA_MON
.WORD 0

; >>>>>>>>>>>>>>>> DEFINITIONS <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( DEFINITIONS [ ] macht das Context-Vokabular auch current.                  )
;( ========================================================================== )
;
; : DEFINITIONS CONTEXT @ CURRENT ! ;

NFA_DEFINITIONS .CBYTE $8B, "DEFINITIONS"
LFA_DEFINITIONS .WORD NFA_FORTH
CFA_DEFINITIONS .WORD DOCOL
PFA_DEFINITIONS
.WORD CFA_CONTEXT ; CONTEXT
.WORD CFA_FETCH ; @
```

```
.WORD CFA_CURRENT ; CURRENT
.WORD CFA_STORE ; !
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>> ( <<<<<<<<<<<<<<<
;( ======================================================================== )
;( [ [ ] leitet einen Kommentar ein, der bis zur naechsten schliessenden     )
;( Klammer reicht.                                                          )
;( ======================================================================== )
;
; : ( 41 WORD DROP ; IMMEDIATE         ( bis zur naechsten "Klammer zu" ueberlesen )

NFA_LEFTPAREN .CBYTE  $81+$40, "(" ; IMMEDIATE
LFA_LEFTPAREN .WORD NFA_DEFINITIONS
CFA_LEFTPAREN .WORD DOCOL
PFA_LEFTPAREN
.WORD CFA_CLIT ; CLIT
.BYTE 41
.WORD CFA_WORD ; WORD
.WORD CFA_DROP ; DROP
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>> QUIT <<<<<<<<<<<<<<<
;( ======================================================================== )
;( QUIT [ ] schaltet auf Keyboard-Eingabe um, beendet den Compile-Mode und   )
;( tritt in eine Endlosschleife ein, in der der Benutzer Eingaben macht und  )
;( diese anschliessend interpretiert werden.                                 )
;( ======================================================================== )
; : QUIT
;  0 SOURCE-ID !     ( auf Keyboard-Eingabe umschalten                       )
;  [COMPILE] [       ( auf Execute-Mode umschalten                           )
;  BEGIN             ( Endlosschleife:                                       )
;    RP!             ( Returnstack-Pointer reinitialisieren                  )
;    CR              ( Zeilenvorschub ausgeben                               )
;    QUERY           ( eine Zeile Text in den Terminal-Input-Buffer einlesen )
;    INTERPRET       ( Zeile interpretieren, ASCII 0 beendet Interpretation  )
;    STATE @ 0= IF   ( falls Execute-Mode:                                   )
;      ." OK"        ( "OK" ausgeben                                         )
;    ENDIF
;  AGAIN             ( Ende der Endlosschleife                               )
; ;

NFA_QUIT .CBYTE $84, "QUIT"
LFA_QUIT .WORD NFA_LEFTPAREN
CFA_QUIT .WORD DOCOL
PFA_QUIT
.WORD CFA_0 ; 0
.WORD CFA_SOURCEID ; SOURCE-ID
.WORD CFA_STORE ; !
.WORD CFA_LEFTBRACKET ; [
LBL41
.WORD CFA_RPSTORE ; RP!
.WORD CFA_CR ; CR
.WORD CFA_QUERY ; QUERY
.WORD CFA_INTERPRET ; INTERPRET
.WORD CFA_STATE ; STATE
.WORD CFA_FETCH ; @
.WORD CFA_NULLEQUAL ; 0=
.WORD CFA_0BRANCH ; 0BRANCH
```

```
.WORD LBL42-*
.WORD CFA_BRAKETDOTQUOTE ; (.")
.BYTE 2
.BYTE 79
.BYTE 75
LBL42
.WORD CFA_BRANCH ; BRANCH
.WORD LBL41-*
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> ABORT <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( ABORT [ ] reinitialisiert die Stacks, schaltet in den Execute-Mode um und )
;( geht in den aeusseren Interpreter QUIT ueber.                             )
;( ========================================================================== )
;
; : ABORT
;  SP! DECIMAL DR0               ( Initialisierungen vornehmen          )
;  CR ." X-FORTH  1.1"           ( Systemmeldung ausgeben               )
;  [COMPILE] FORTH DEFINITIONS   ( FORTH zum aktuellen Vokabular machen )
;  QUIT                          ( zum aeusseren Interpreter QUIT       )
; ;

NFA_ABORT .CBYTE $85,"ABORT"
LFA_ABORT .WORD NFA_QUIT
CFA_ABORT .WORD DOCOL
PFA_ABORT
.WORD CFA_SPSTORE  ; SP!
.WORD CFA_DECIMAL ; DECIMAL
.WORD CFA_CR  ; CR
.WORD CFA_FORTH   ; FORTH
.WORD CFA_DEFINITIONS ; DEFINITIONS
;.WORD CFA_BRAKETDOTQUOTE ; (.")
;.BYTE 22,"X-FORTH 1.1c 040410/cs"
ABORTINIT
.WORD CFA_QUIT ; QUIT
.WORD CFA_EXIT  ; ;S

; >>>>>>>>>>>>>>>> COLD <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( COLD [ ] fuehrt einen Kaltstart des Systems durch.                        )
;( ========================================================================== )

NFA_COLD .CBYTE $84, "COLD"
LFA_COLD .WORD NFA_ABORT
CFA_COLD .WORD PFA_COLD
PFA_COLD
  SEI              ; maskierbare Interrupts sperren
                   ; USERAREA default siehe ORIGIN+16
.IF DYNMEMTOP      ; USERAREA nach MEMTOP ($2E5)-$100 setzen
  LDA $02E5
  STA ORIG+16
  LDA $02E6
  SEC
  SBC #2
  STA ORIG+17
.ENDIF

RESET
```

```
  LDA DOSINI
  STA DOSIN+1
  LDA DOSINI+1
  STA DOSIN+2

  LDA ORIG+12      ; NFA des letzten Wortes im Dictionary
  STA PFA_FORTH+4  ; in das Vokabular-Wort FORTH eintragen
  LDA ORIG+13
  STA PFA_FORTH+5
  LDY #21          ; Kaltstart: Uservariablen 0 bis VOC-LINK init.
  BNE L2433        ; d.h. zusaetzlich FENCE, DP und VOC-LINK

WARM
DOSIN
  JSR $E474    ; wird ueberschrieben!
  LDY #15          ; Warmstart: nur Uservariablen 0 bis WARNING init.

L2433
  LDA ORIG+16      ; User-Pointer UP initialisieren
  STA UP
  LDA ORIG+17
  STA UP+1
L2437
  LDA ORIG+12,Y    ; Uservariablen mit Werten ab 12+ORIGIN initialis.
  STA (UP),Y
  DEY
  BPL L2437
  LDA #>PFA_ABORT    ; Instruction-Pointer IP initialisieren:
  STA IP+1           ; Interpretation beginnt mit dem Wort ABORT
  LDA #<PFA_ABORT
  STA IP
  CLD              ; Dezimalarithmetik ausschalten
  LDA #$6C         ; Opcode fuer JMP (), d.h. indirekten Sprung
  STA W-1          ; unmittelbar vor das W-Register schreiben
  CLI              ; maskierbare Interrupts zulassen

  LDA #<WARM
  STA DOSINI
  LDA #>WARM
  STA DOSINI+1

  JMP PFA_RPSTORE  ; Returnstackpointer initialisieren und ABORT

; >>>>>>>>>>>>>>>> S>D <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( S>D [ n -> d ] wandelt die Zahl n in eine doppeltgenaue Zahl d um.       )
;( ========================================================================== )
;
; : S>D DUP 0< MINUS ;               ( $0000 oder $FFFF als Hi-Word erzeugen )

NFA_STOD .CBYTE $83, "S>D"
LFA_STOD .WORD NFA_COLD
CFA_STOD .WORD DOCOL
PFA_STOD
.WORD CFA_DUP ; DUP
.WORD CFA_LTNULL ; 0<
.WORD CFA_NEGATE ; MINUS
.WORD CFA_EXIT ; ;S
```

```
; >>>>>>>>>>>>>>>> +- <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( +- [ n1 n2 -> n3 ] gibt n3=-n1 zurueck, wenn n2 negativ ist, sonst n3=n1. )
;( ========================================================================== )
;
; : +- 0< IF MINUS ENDIF ;

NFA_PLUSMINUS .CBYTE $82,"+-"
LFA_PLUSMINUS .WORD NFA_STOD
CFA_PLUSMINUS .WORD DOCOL
PFA_PLUSMINUS
.WORD CFA_LTNULL ; 0<
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL43-*
.WORD CFA_NEGATE ; MINUS
LBL43
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> D+- <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( D+- [ d1 n -> d2 ] gibt d2=-d1 zurueck, wenn n negativ ist, sonst d2=d1.  )
;( Hierbei sind d1 und d2 doppeltgenaue Zahlen.                              )
;( ========================================================================== )
;
; : D+- 0< IF DMINUS ENDIF ;

NFA_DPLUSMINUS .CBYTE $83,"D+-"
LFA_DPLUSMINUS .WORD NFA_PLUSMINUS
CFA_DPLUSMINUS .WORD DOCOL
PFA_DPLUSMINUS
.WORD CFA_LTNULL ; 0<
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL44-*
.WORD CFA_DMINUS ; DMINUS
LBL44
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> ABS <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( ABS [ n -> u ] berechnet den absoluten Betrag der Zahl n.                 )
;( ========================================================================== )
;
; : ABS DUP +- ;

NFA_ABS .CBYTE $83,"ABS"
LFA_ABS .WORD NFA_DPLUSMINUS
CFA_ABS .WORD DOCOL
PFA_ABS
.WORD CFA_DUP ; DUP
.WORD CFA_PLUSMINUS ; +-
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> DABS <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( DABS [ d -> ud ] berechnet den absoluten Betrag der doppeltgenauen        )
;( Zahl d.                                                                   )
;( ========================================================================== )
;
; : DABS DUP D+- ;
```

```
NFA_DABS .CBYTE  $84,"DABS"
LFA_DABS .WORD NFA_ABS
CFA_DABS .WORD DOCOL
PFA_DABS
.WORD CFA_DUP ; DUP
.WORD CFA_DPLUSMINUS ; D+-
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> MIN <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( MIN [ n1 n2 -> min ] berechnet das Minimum der Zahlen n1 und n2.          )
;( ======================================================================== )
;
; : MIN
;  OVER OVER > IF   ( falls n1>n2 ist:                )
;    SWAP           ( n1 in den TOS tauschen          )
;  ENDIF
;  DROP             ( groessere Zahl im TOS vergessen )
; ;

NFA_MIN .CBYTE $83,"MIN"
LFA_MIN .WORD NFA_DABS
CFA_MIN .WORD DOCOL
PFA_MIN
.WORD CFA_OVER ; OVER
.WORD CFA_OVER ; OVER
.WORD CFA_GT ; >
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL45-*
.WORD CFA_SWAP ; SWAP
LBL45
.WORD CFA_DROP ; DROP
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> MAX <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( MAX [ n1 n2 -> max ] berechnet das Maximum der Zahlen n1 und n2.          )
;( ======================================================================== )
;
; : MAX
;  OVER OVER < IF   ( falls n1<n2 ist:                )
;    SWAP           ( n1 in den TOS tauschen          )
;  ENDIF
;  DROP             ( kleinere Zahl im TOS vergessen )
; ;

NFA_MAX .CBYTE $83,"MAX"
LFA_MAX .WORD NFA_MIN
CFA_MAX .WORD DOCOL
PFA_MAX
.WORD CFA_OVER ; OVER
.WORD CFA_OVER ; OVER
.WORD CFA_LT ; <
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL46-*
.WORD CFA_SWAP ; SWAP
LBL46
.WORD CFA_DROP ; DROP
```

```
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> M* <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( M* [ n1 n2 -> d ] multipliziert die beiden einfachgenauen Zahlen n1 und   )
;( n2 und gibt das Ergebnis als doppeltgenaue Zahl d zurueck.                )
;( ======================================================================== )
;
; : M*
;  OVER OVER XOR >R   ( Vorzeichen-Flag auf den Return-Stack bringen   )
;  ABS SWAP ABS U*    ( absolute Betraege von n1 und n2 multiplizieren )
;  R> D+-             ( eventuell Ergebnis negieren                    )
; ;

NFA_MSTAR .CBYTE $82,"M*"
LFA_MSTAR .WORD NFA_MAX
CFA_MSTAR .WORD DOCOL
PFA_MSTAR
.WORD CFA_OVER ; OVER
.WORD CFA_OVER ; OVER
.WORD CFA_XOR ; XOR
.WORD CFA_RPUSH ; >R
.WORD CFA_ABS ; ABS
.WORD CFA_SWAP ; SWAP
.WORD CFA_ABS ; ABS
.WORD CFA_UMULT ; U*
.WORD CFA_RPOP ; R>
.WORD CFA_DPLUSMINUS ; D+-
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> M/ <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( M/ [ d n1 -> n2 n3 ] berechnet zur doppeltgenauen Zahl d und zur einfach- )
;( genauen Zahl n1 den Quotienten n3=d/n1 sowie den Rest n2, der das Vorzei- )
;( chen des Zaehlers d erhaelt.                                              )
;( ======================================================================== )
;
; : M/
;  OVER >R >R      ( Hi-Wort von d und n1 auf den Return-Stack bringen )
;  DABS R ABS U/   ( |d| durch |n1| teilen, Quotient liegt auf dem TOS )
;  R> R XOR +-     ( Quotient negieren, falls sign[d]<>sign[n1]        )
;  SWAP R> +-      ( Rest negieren, falls d negativ ist                )
;  SWAP            ( Quotient wieder in den TOS tauschen               )
; ;

NFA_MSLASH .CBYTE $82,"M/"
LFA_MSLASH .WORD NFA_MSTAR
CFA_MSLASH .WORD DOCOL
PFA_MSLASH
.WORD CFA_OVER ; OVER
.WORD CFA_RPUSH ; >R
.WORD CFA_RPUSH ; >R
.WORD CFA_DABS ; DABS
.WORD CFA_R ; R
.WORD CFA_ABS ; ABS
.WORD CFA_UDIV ; U/
.WORD CFA_RPOP ; R>
.WORD CFA_R ; R
.WORD CFA_XOR ; XOR
```

```
.WORD CFA_PLUSMINUS ; +-
.WORD CFA_SWAP ; SWAP
.WORD CFA_RPOP ; R>
.WORD CFA_PLUSMINUS ; +-
.WORD CFA_SWAP ; SWAP
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> * <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( * [ n1 n2 -> n3 ] multipliziert die Zahlen n1 und n2.                    )
;( ======================================================================== )
;
; : * U* DROP ;          ( multiplizieren und Hi-Word des Produktes vergessen )

NFA_STAR .CBYTE $81,"*"
LFA_STAR .WORD NFA_MSLASH
CFA_STAR .WORD DOCOL
PFA_STAR
.WORD CFA_UMULT ; U*
.WORD CFA_DROP ; DROP
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> /MOD <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( /MOD [ n1 n2 -> Rest Quot ] teilt die Zahl n1 durch die Zahl n2 und       )
;( uebergibt den Rest und den Quotienten.                                    )
;( ======================================================================== )
;
; : /MOD >R S>D R> M/ ;             ( n1 doppeltgenau machen und M/ aufrufen )

NFA_SLASHMOD .CBYTE $84,"/MOD"
LFA_SLASHMOD .WORD NFA_STAR
CFA_SLASHMOD .WORD DOCOL
PFA_SLASHMOD
.WORD CFA_RPUSH ; >R
.WORD CFA_STOD ; S>D
.WORD CFA_RPOP ; R>
.WORD CFA_MSLASH ; M/
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> / <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( / [ n1 n2 -> Quotient ] berechnet den Quotienten n1/n2.                   )
;( ======================================================================== )
;
; : / /MOD SWAP DROP ;

NFA_SLASH .CBYTE $81,"/"
LFA_SLASH .WORD NFA_SLASHMOD
CFA_SLASH .WORD DOCOL
PFA_SLASH
.WORD CFA_SLASHMOD ; /MOD
.WORD CFA_SWAP ; SWAP
.WORD CFA_DROP ; DROP
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> MOD <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( MOD [ n1 n2 -> mod ] berechnet den Rest der Division n1/n2.               )
```

```
;( ========================================================================== )
;
; : MOD /MOD DROP ;

NFA_MOD .CBYTE $83,"MOD"
LFA_MOD .WORD NFA_SLASH
CFA_MOD .WORD DOCOL
PFA_MOD
.WORD CFA_SLASHMOD ; /MOD
.WORD CFA_DROP ; DROP
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> */MOD <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( */MOD [ n1 n2 n3 -> n4 n5 ] berechnet [n1*n2]/n3 und uebergibt in n4 den  )
;( Rest und in n5 den Quotienten. Es wird mit einem doppeltgenauen Zwischen- )
;( produkt n1*n2 gerechnet.                                                  )
;( ========================================================================== )
;
; : */MOD >R M* R> M/ ;

NFA_STARSLASHMOD .CBYTE $85, "*/MOD"
LFA_STARSLASHMOD .WORD NFA_MOD
CFA_STARSLASHMOD .WORD DOCOL
PFA_STARSLASHMOD
.WORD CFA_RPUSH ; >R
.WORD CFA_MSTAR ; M*
.WORD CFA_RPOP ; R>
.WORD CFA_MSLASH ; M/
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> */ <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( */ [ n1 n2 n3 -> n4 ] berechnet den Quotienten [n1*n2]/n3. Es wird mit    )
;( einem doppeltgenauen Zwischenprodukt n1*n2 gerechnet.                     )
;( ========================================================================== )
;
; : */ */MOD SWAP DROP ;

NFA_STARSLASH .CBYTE $82,"*/"
LFA_STARSLASH .WORD NFA_STARSLASHMOD
CFA_STARSLASH .WORD DOCOL
PFA_STARSLASH
.WORD CFA_STARSLASHMOD ; */MOD
.WORD CFA_SWAP ; SWAP
.WORD CFA_DROP ; DROP
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> M/MOD <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( M/MOD [ ud1 u2 -> u3 ud4 ] dividiert die vorzeichenlose, doppeltgenaue    )
;( Zahl ud1 durch die vorzeichenlose Zahl u2 und uebergibt den Rest u3 sowie )
;( den doppeltgenauen Quotienten ud4.                                        )
;( ========================================================================== )
;
;( Funktionsweise des Algorithmus: Seien B=2^16 und ud1=a*B+b. Mit / sei die )
;( ganzzahlige Division bezeichnet, mit % die Restbildung. Dann gilt:        )
;( ud4 = ud1/u2 = [a*B+b]/u2 = [a/u2]*B + [[a%u2]*B+b]/u2 und                )
;( u3  = ud1%u2 = [a*B+b]%u2 = [[a%u2]*B+b]%u2.                              )
```

```
;( Im untenstehenden Kommentar bezeichne q=[[a%u2]*B+b]/u2.                  )
;
;: M/MOD        ( b   a     u2                    )
;  >R           ( b   a                 RS: u2   )
;  0 R          ( b   a     0     u2    RS: u2   )
;  U/           ( b   a%u2  a/u2        RS: u2   )
;  R> SWAP >R   ( b   a%u2  u2          RS: a/u2 )
;  U/           ( u3  q                 RS: a/u2 )
;  R>           ( u3  q     a/u2                 )
; ;

NFA_MMOD .CBYTE $85, "M/MOD"
LFA_MMOD .WORD NFA_STARSLASH
CFA_MMOD .WORD DOCOL
PFA_MMOD
.WORD CFA_RPUSH ; >R
.WORD CFA_0 ; 0
.WORD CFA_R ; R
.WORD CFA_UDIV ; U/
.WORD CFA_RPOP ; R>
.WORD CFA_SWAP ; SWAP
.WORD CFA_RPUSH ; >R
.WORD CFA_UDIV ; U/
.WORD CFA_RPOP ; R>
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> MESSAGE <<<<<<<<<<<<<<<<
;( ========================================================================= )
;( MESSAGE [ n -> ] gibt die Meldung Nummer n auf die Standardausgabe aus.   )
;( Hat die User-Variable WARNING einen Wert ungleich Null, so ist dies der   )
;( Inhalt der n-ten Zeile relativ zur Zeile 0 in Screen 4, Laufwerk 0.       )
;( ========================================================================= )

NFA_MESSAGE .CBYTE $87, "MESSAGE"
LFA_MESSAGE .WORD NFA_MMOD
CFA_MESSAGE .WORD DOCOL
PFA_MESSAGE
.WORD CFA_BRAKETDOTQUOTE ; (.")
.BYTE 6,"MSG # "
.WORD CFA_DOT ; .
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> ' <<<<<<<<<<<<<<<<
;( ========================================================================= )
;( '  [ -> addr ] ermittelt die PFA des folgenden Wortes. Im Compile-Mode    )
;( wird die PFA des folgenden Wortes zusammen mit einem LIT kompiliert.      )
;( ========================================================================= )
;
; : '
;  -FIND               ( Wort im Dictionary suchen                          )
;  0= 0 ?ERROR         ( Fehlermeldung ausgeben, falls nicht gefunden       )
;  DROP                ( Count-Byte des Wortes vergessen                    )
;  [COMPILE] LITERAL   ( PFA auf dem Stack lassen oder mit LIT kompilieren  )
; ; IMMEDIATE

NFA_TICK .BYTE $81+$40, 167 ; IMMEDIATE
LFA_TICK .WORD NFA_MESSAGE
CFA_TICK .WORD DOCOL
PFA_TICK
```

```
.WORD CFA_MINUSFIND ; -FIND
.WORD CFA_NULLEQUAL ; 0=
.WORD CFA_0 ; 0
.WORD CFA_QUERYERROR ; ?ERROR
.WORD CFA_DROP ; DROP
.WORD CFA_LITERAL ; LITERAL
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> FORGET <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( FORGET [ ] loescht alle Woerter ab dem im Input-Strom folgenden aus dem   )
;( Dictionary. Sind das Current- und das Context-Vokabular nicht identisch,  )
;( so wird eine Fehlermeldung ausgegeben.                                    )
;( ========================================================================== )
;
; : FORGET
;  CURRENT @ CONTEXT @ - 24 ?ERROR   ( Fehlermeldung falls CURRENT<>CONTEXT  )
;  [COMPILE] '                       ( PFA des folgenden Wortes              )
;  DUP FENCE @ < 21 ?ERROR           ( Fehlermeldung falls geschuetzt        )
;  DUP NFA DP !                      ( Dictionary-Pointer herabsetzen        )
;  LFA @ CURRENT @ !                 ( jetzt letztes Wort im Curr.-V. merken )
; ;

NFA_FORGET .CBYTE $86,"FORGET"
LFA_FORGET .WORD NFA_TICK
CFA_FORGET .WORD DOCOL
PFA_FORGET
.WORD CFA_CURRENT ; CURRENT
.WORD CFA_FETCH ; @
.WORD CFA_CONTEXT ; CONTEXT
.WORD CFA_FETCH ; @
.WORD CFA_MINUS ; -
.WORD CFA_CLIT ; CLIT
.BYTE 24
.WORD CFA_QUERYERROR ; ?ERROR
.WORD CFA_TICK ; '
.WORD CFA_DUP ; DUP
.WORD CFA_FENCE ; FENCE
.WORD CFA_FETCH ; @
.WORD CFA_LT ; <
.WORD CFA_CLIT ; CLIT
.BYTE 21
.WORD CFA_QUERYERROR ; ?ERROR
.WORD CFA_DUP ; DUP
.WORD CFA_NFA ; NFA
.WORD CFA_DP ; DP
.WORD CFA_STORE ; !
.WORD CFA_LFA ; LFA
.WORD CFA_FETCH ; @
.WORD CFA_CURRENT ; CURRENT
.WORD CFA_FETCH ; @
.WORD CFA_STORE ; !
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> BACK <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( BACK [ addr -> ] kompiliert die Distanz von HERE nach addr in das         )
;( Dictionary.                                                               )
;( ========================================================================== )
```

```
;
; : BACK HERE - , ;

NFA_BACK .CBYTE $84,"BACK"
LFA_BACK .WORD NFA_FORGET
CFA_BACK .WORD DOCOL
PFA_BACK
.WORD CFA_HERE ; HERE
.WORD CFA_MINUS ; -
.WORD CFA_COMMA ; ,
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> BEGIN <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( BEGIN [ -> addr n ] legt die aktuelle Adresse HERE und die Strukturken-   )
;( nung n=1 auf den Stack. Das Wort wird innerhalb von Colon-Definitionen    )
;( verwendet und leitet eine der Strukturen BEGIN...UNTIL, BEGIN...AGAIN     )
;( oder BEGIN...WHILE...REPEAT ein.                                          )
;( ========================================================================== )
;
; : BEGIN ?COMP HERE 1 ; IMMEDIATE

NFA_BEGIN .CBYTE $85+$40,"BEGIN" ; IMMEDIATE
LFA_BEGIN .WORD NFA_BACK
CFA_BEGIN .WORD DOCOL
PFA_BEGIN
.WORD CFA_QUERYCOMP ; ?COMP
.WORD CFA_HERE ; HERE
.WORD CFA_1 ; 1
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> ENDIF <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( ENDIF [ addr n -> ] korrigiert die in der Adresse addr angelegte proviso- )
;( rische Sprungdistanz auf ihren korrekten Wert, naemlich nach HERE. Eine   )
;( Fehlermeldung wird ausgegeben, wenn die Strukturkennung n<>2 ist und      )
;( somit kein IF-Konstrukt abgeschlossen wurde.                              )
;( ========================================================================== )
;
; : ENDIF
;  ?COMP         ( Fehlermeldung falls nicht Compile-Mode                    )
;  2 ?PAIRS      ( Fehlermeldung falls kein IF-Konstrukt abgeschlossen wurde )
;  HERE OVER -   ( Sprungdistanz berechnen                                   )
;  SWAP !        ( und in die durch addr bezeichnete Zelle eintragen         )
; ; IMMEDIATE

NFA_ENDIF .CBYTE $85+$40, "ENDIF" ; IMMEDIATE
LFA_ENDIF .WORD NFA_BEGIN
CFA_ENDIF .WORD DOCOL
PFA_ENDIF
.WORD CFA_QUERYCOMP ; ?COMP
.WORD CFA_2 ; 2
.WORD CFA_QUERYPAIRS ; ?PAIRS
.WORD CFA_HERE ; HERE
.WORD CFA_OVER ; OVER
.WORD CFA_MINUS ; -
.WORD CFA_SWAP ; SWAP
.WORD CFA_STORE ; !
.WORD CFA_EXIT ; ;S
```

```
; >>>>>>>>>>>>>>>> THEN <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( THEN [ addr n -> ] ist ein Alias fuer ENDIF.                             )
;( ======================================================================== )
;
; : THEN [COMPILE] ENDIF ; IMMEDIATE

NFA_THEN .CBYTE $84+$40, "THEN" ; IMMEDIATE
LFA_THEN .WORD NFA_ENDIF
CFA_THEN .WORD DOCOL
PFA_THEN
.WORD CFA_ENDIF ; ENDIF
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> DO <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( DO [ -> addr n ] legt die aktuelle Adresse HERE und die Strukturkennung   )
;( n=3 auf den Stack und kompiliert ein [DO] in das Dictionary. Das Wort     )
;( wird innerhalb von Colon-Definitionen verwendet und leitet eine der       )
;( beiden Strukturen DO...LOOP oder DO...+LOOP ein.                          )
;( ======================================================================== )
;
; : DO
;  COMPILE (DO)   ( Runtime-Exekutive [DO] kompilieren                       )
;  HERE 3         ( Adresse HERE und Strukturkennung n=3 auf den Stack legen )
; ; IMMEDIATE

NFA_DO .CBYTE $82+$40, "DO" ; IMMEDIATE
LFA_DO .WORD NFA_THEN
CFA_DO .WORD DOCOL
PFA_DO
.WORD CFA_COMPILE ; COMPILE
.WORD CFA_BRACKETDO ; (DO)
.WORD CFA_HERE ; HERE
.WORD CFA_3 ; 3
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> LOOP <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( LOOP [ addr n -> ] kompiliert die Runtime-Exekutive [LOOP] zusammen mit   )
;( einer Ruecksprung-Distanz nach addr in das Dictionary und ueberprueft, ob )
;( die Strukturkennung n=3 uebergeben und somit eine DO-Struktur korrekt     )
;( abgeschlossen wurde.                                                      )
;( ======================================================================== )
;
; : LOOP
;  3 ?PAIRS         ( Fehler falls kein DO-Konstrukt abgeschlossen wurde )
;  COMPILE (LOOP)   ( Runtime-Exekutive [LOOP] kompilieren               )
;  BACK             ( Ruecksprung-Distanz kompilieren                    )
; ; IMMEDIATE

NFA_LOOP .CBYTE $84+$40,"LOOP" ; IMMEDIATE
LFA_LOOP .WORD NFA_DO
CFA_LOOP .WORD DOCOL
PFA_LOOP
.WORD CFA_3 ; 3
.WORD CFA_QUERYPAIRS ; ?PAIRS
.WORD CFA_COMPILE ; COMPILE
```

```
.WORD CFA_BRACKETLOOP ; (LOOP)
.WORD CFA_BACK ; BACK
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>> +LOOP <<<<<<<<<<<<<<<
;( ======================================================================== )
;( +LOOP [ addr n -> ] kompiliert die Runtime-Exekutive [+LOOP] zusammen mit )
;( einer Ruecksprung-Distanz nach addr in das Dictionary und ueberprueft, ob )
;( die Strukturkennung n=3 uebergeben und somit eine DO-Struktur korrekt     )
;( abgeschlossen wurde.                                                      )
;( ======================================================================== )
;
; : +LOOP
;  3 ?PAIRS          ( Fehler falls kein DO-Konstrukt abgeschlossen wurde )
;  COMPILE (+LOOP)   ( Runtime-Exekutive [+LOOP] kompilieren              )
;  BACK              ( Ruecksprung-Distanz kompilieren                    )
; ; IMMEDIATE

NFA_ADDLOOP .CBYTE $85+$40, "+LOOP" ; IMMEDIATE
LFA_ADDLOOP .WORD NFA_LOOP
CFA_ADDLOOP .WORD DOCOL
PFA_ADDLOOP
.WORD CFA_3 ; 3
.WORD CFA_QUERYPAIRS ; ?PAIRS
.WORD CFA_COMPILE ; COMPILE
.WORD CFA_BRACKETPLUSLOOP ; (+LOOP)
.WORD CFA_BACK ; BACK
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>> UNTIL <<<<<<<<<<<<<<<
;( ======================================================================== )
;( UNTIL [ addr n -> ] kompiliert einen bedingten Ruecksprung nach Adresse   )
;( addr in das Dictionary und ueberprueft, ob die Strukturkennung n=1 ueber- )
;( geben und somit eine BEGIN-Struktur korrekt abgeschlossen wurde.          )
;( ======================================================================== )
;
; : UNTIL
;  1 ?PAIRS          ( Fehler falls kein BEGIN-Konstrukt abgeschlossen wurde )
;  COMPILE 0BRANCH   ( bedingten Sprung kompilieren                          )
;  BACK              ( Ruecksprung-Distanz kompilieren                       )
; ; IMMEDIATE

NFA_UNTIL .CBYTE $85+$40, "UNTIL" ; IMMEDIATE
LFA_UNTIL .WORD NFA_ADDLOOP
CFA_UNTIL .WORD DOCOL
PFA_UNTIL
.WORD CFA_1 ; 1
.WORD CFA_QUERYPAIRS ; ?PAIRS
.WORD CFA_COMPILE ; COMPILE
.WORD CFA_0BRANCH ; 0BRANCH
.WORD CFA_BACK ; BACK
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>> END <<<<<<<<<<<<<<<
;( ======================================================================== )
;( END [ addr n -> ] ist ein Alias fuer UNTIL.                               )
;( ======================================================================== )
;
; : END [COMPILE] UNTIL ; IMMEDIATE
```

```
NFA_END .CBYTE $83+$40, "END" ; IMMEDIATE
LFA_END .WORD NFA_UNTIL
CFA_END .WORD DOCOL
PFA_END
.WORD CFA_UNTIL ; UNTIL
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> AGAIN <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( AGAIN [ addr n -> ] kompiliert einen unbedingten Ruecksprung nach Adresse )
;( addr in das Dictionary und ueberprueft, ob die Strukturkennung n=1 ueber- )
;( geben und somit eine BEGIN-Struktur korrekt abgeschlossen wurde.          )
;( ========================================================================== )
;
; : AGAIN
;  1 ?PAIRS          ( Fehler falls kein BEGIN-Konstrukt abgeschlossen wurde  )
;  COMPILE BRANCH    ( unbedingten Sprung kompilieren                         )
;  BACK              ( Ruecksprung-Distanz kompilieren                        )
; ; IMMEDIATE

NFA_AGAIN .CBYTE $85+$40,"AGAIN" ; IMMEDIATE
LFA_AGAIN .WORD NFA_END
CFA_AGAIN .WORD DOCOL
PFA_AGAIN
.WORD CFA_1 ; 1
.WORD CFA_QUERYPAIRS ; ?PAIRS
.WORD CFA_COMPILE ; COMPILE
.WORD CFA_BRANCH ; BRANCH
.WORD CFA_BACK ; BACK
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> REPEAT <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( REPEAT [ addr1 n1 addr2 n2 -> ] kompiliert einen unbedingten Ruecksprung  )
;( nach Adresse addr1 in das Dictionary und korrigiert die Sprungdistanz bei )
;( Adresse addr2 derart, dass hinter den soeben kompilierten Sprung gesprun- )
;( gen wird. Darueberhinaus wird ueberprueft, ob mit n1=1 ein BEGIN-         )
;( Konstrukt abgeschlossen wurde, der wegen n2=4 ein WHILE enthielt.         )
;( ========================================================================== )
;
; : REPEAT
;  >R >R                    ( addr2 und n2 auf den Return-Stack retten       )
;  [COMPILE] AGAIN          ( unbedingten Ruecksprung nach addr1 kompilieren )
;  R> R>                    ( addr2 und n2 zurueckholen                      )
;  2 - [COMPILE] ENDIF      ( Vorwaertssprung bei Adresse addr2 korrigieren  )
; ; IMMEDIATE

NFA_REPEAT .CBYTE $86+$40, "REPEAT" ; IMMEDITE
LFA_REPEAT .WORD NFA_AGAIN
CFA_REPEAT .WORD DOCOL
PFA_REPEAT
.WORD CFA_RPUSH ; >R
.WORD CFA_RPUSH ; >R
.WORD CFA_AGAIN ; AGAIN
.WORD CFA_RPOP ; R>
.WORD CFA_RPOP ; R>
.WORD CFA_2 ; 2
.WORD CFA_MINUS ; -
```

```
.WORD CFA_ENDIF ; ENDIF
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> IF <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( IF [ -> addr n ] kompiliert einen bedingten Sprung mit noch provisori-    )
;( scher Sprungdistanz und legt die Adresse, an der die Distanz steht, sowie )
;( die Strukturkennung n=2 auf den Stack. Das Wort wird in Colon-Definitio-  )
;( nen zur Einleitung eines IF...ENDIF- oder IF...ELSE...ENDIF-Konstruktes   )
;( benutzt.                                                                  )
;( ========================================================================== )
;
; : IF
;  COMPILE 0BRANCH   ( bedingten Sprung kompilieren            )
;  HERE              ( Adresse addr, an der die Distanz steht  )
;  0 ,               ( provisorische Sprungdistanz kompilieren )
;  2                 ( Strukturkennung n=2                     )
; ; IMMEDIATE

NFA_IF .CBYTE $82+$40,"IF" ; IMMEDIATE
LFA_IF .WORD NFA_REPEAT
CFA_IF .WORD DOCOL
PFA_IF
.WORD CFA_COMPILE ; COMPILE
.WORD CFA_0BRANCH ; 0BRANCH
.WORD CFA_HERE ; HERE
.WORD CFA_0 ; 0
.WORD CFA_COMMA ; ,
.WORD CFA_2 ; 2
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> ELSE <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( ELSE [ addr1 n1 -> addr2 n2 ] kompiliert einen unbedingten Sprung mit     )
;( noch provisorischer Sprungdistanz und korrigiert die Sprungdistanz in     )
;( addr1 derart, dass hinter den soeben kompilierten Sprung gesprungen wird. )
;( Darueberhinaus wird ueberprueft, ob mit n1=2 ein IF-Konstrukt um einen    )
;( ELSE-Zweig ergaenzt wurde. Auf dem Stack werden die Adresse addr2, an der )
;( sich die soeben kompilierte provisorische Distanz befindet, und die       )
;( Strukturkennung n2=2 hinterlassen.                                        )
;( ========================================================================== )
;
; : ELSE
;  2 ?PAIRS                ( Fehler falls kein IF-Konstrukt          )
;  COMPILE BRANCH          ( unbedingten Sprung kompilieren          )
;  HERE                    ( Adresse addr2, an der die Distanz steht )
;  0 ,                     ( provisorische Distanz kompilieren       )
;  SWAP 2 [COMPILE] ENDIF  ( Sprungdistanz bei addr1 korrigieren     )
;  2                       ( Strukturkennung n2=2                    )
; ; IMMEDIATE

NFA_ELSE .CBYTE $84+$40,"ELSE" ; IMMEDIATE
LFA_ELSE .WORD NFA_IF
CFA_ELSE .WORD DOCOL
PFA_ELSE
.WORD CFA_2 ; 2
.WORD CFA_QUERYPAIRS ; ?PAIRS
.WORD CFA_COMPILE ; COMPILE
.WORD CFA_BRANCH ; BRANCH
```

```
.WORD CFA_HERE ; HERE
.WORD CFA_0 ; 0
.WORD CFA_COMMA ; ,
.WORD CFA_SWAP ; SWAP
.WORD CFA_2 ; 2
.WORD CFA_ENDIF ; ENDIF
.WORD CFA_2 ; 2
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> WHILE <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( WHILE [ -> addr n ] kompiliert einen bedingten Vorwaertssprung mit noch   )
;( provisorischer Sprungdistanz und hinterlaesst auf dem Stack die Adresse   )
;( addr, an der die provisorische Sprungdistanz steht, und die Strukturken-  )
;( nung n=4.                                                                 )
;( ========================================================================== )
;
; : WHILE [COMPILE] IF 2+ ; IMMEDIATE

NFA_WHILE .CBYTE $85+$40, "WHILE" ; IMMEDIATE
LFA_WHILE .WORD NFA_ELSE
CFA_WHILE .WORD DOCOL
PFA_WHILE
.WORD CFA_IF ; IF
.WORD CFA_2PLUS ; 2+
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> SPACES <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( SPACES [ n -> ] gibt n Spaces auf die Standardausgabe aus.                )
;( ========================================================================== )
;
; : SPACES
;  0 MAX -DUP IF   ( falls n>0 ist:    )
;    0 DO          ( wiederhole n-mal: )
;      SPACE       ( Blank ausgeben    )
;    LOOP
;  ENDIF
; ;

NFA_SPACES .CBYTE $86,"SPACES"
LFA_SPACES .WORD NFA_WHILE
CFA_SPACES .WORD DOCOL
PFA_SPACES
.WORD CFA_0 ; 0
.WORD CFA_MAX ; MAX
.WORD CFA_MINUSDUP ; -DUP
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL50-*
.WORD CFA_0 ; 0
.WORD CFA_BRACKETDO ; (DO)
LBL51
.WORD CFA_SPACE ; SPACE
.WORD CFA_BRACKETLOOP ; (LOOP)
.WORD LBL51-*
LBL50
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> <# <<<<<<<<<<<<<<<<
```

```
;( ========================================================================= )
;( <# [ ] leitet die Umwandlung einer doppeltgenauen Zahl in einen String,   )
;( der von PAD-1 an abwaerts erzeugt wird, ein.                              )
;( ========================================================================= )
;
; : <# PAD HLD ! ;                              ( Laufzeiger HLD initialisieren )

NFA_LTSHARP .CBYTE $82,"<#"
LFA_LTSHARP .WORD NFA_SPACES
CFA_LTSHARP .WORD DOCOL
PFA_LTSHARP
.WORD CFA_PAD ; PAD
.WORD CFA_HLD ; HLD
.WORD CFA_STORE ; !
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> #> <<<<<<<<<<<<<<<<
;( ========================================================================= )
;( #> [ ud -> addr n ] beendet die Umwandlung einer doppeltgenauen Zahl in   )
;( einen String und uebergibt seine Startadresse addr und seine Laenge n.    )
;( ========================================================================= )
;
; : #>
;  DROP DROP    ( evtl. noch vorhandenen Rest ud vergessen )
;  HLD @        ( Startadresse addr des Strings            )
;  PAD OVER -   ( Laenge n des Strings                     )
; ;

NFA_SHARPGT .CBYTE $82,"#>"
LFA_SHARPGT .WORD NFA_LTSHARP
CFA_SHARPGT .WORD DOCOL
PFA_SHARPGT
.WORD CFA_DROP ; DROP
.WORD CFA_DROP ; DROP
.WORD CFA_HLD ; HLD
.WORD CFA_FETCH ; @
.WORD CFA_PAD ; PAD
.WORD CFA_OVER ; OVER
.WORD CFA_MINUS ; -
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> SIGN <<<<<<<<<<<<<<<<
;( ========================================================================= )
;( SIGN [ n d -> d ] fuegt an den im Aufbau befindlichen Zahlenstring ein    )
;( Minuszeichen an, falls n negativ ist. Die doppeltgenaue Zahl d bleibt     )
;( unveraendert.                                                             )
;( ========================================================================= )
;
; : SIGN
;  ROT 0< IF   ( falls n<0 ist:                    )
;    45 HOLD   ( Zeichen "-" an den String anfuegen )
;  ENDIF
; ;

NFA_SIGN .CBYTE  $84,"SIGN"
LFA_SIGN .WORD NFA_SHARPGT
CFA_SIGN .WORD DOCOL
PFA_SIGN
.WORD CFA_ROT ; ROT
```

```
.WORD CFA_LTNULL ; 0<
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL52-*
.WORD CFA_CLIT ; CLIT
.BYTE 45
.WORD CFA_HOLD ; HOLD
LBL52
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> # <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( # [ ud1 -> ud2 ] ermittelt aus der vorzeichenlosen, ganzen Zahl die       )
;( niederwertigste Ziffer und fuegt sie an den im Aufbau befindlichen       )
;( Zahlenstring an. Der verbleibende Rest ud2 wird auf dem Stack uebergeben. )
;( ======================================================================== )
;
; : #
;  BASE @ M/MOD   ( einfachgenaue Ziffer und doppeltgenauen Rest ermitteln )
;  ROT            ( Ziffer nach vorne holen                                )
;  9 OVER < IF    ( falls Ziffer groesser als 9 ist:                       )
;    7 +          ( 7 addieren, um im Buchstabenbereich zu landen          )
;  ENDIF
;  48 +           ( "0" addieren; ergibt Ziffer als ASCII-Zeichen          )
;  HOLD           ( Ziffer an den String anfuegen                          )
; ;

NFA_SHARP .CBYTE $81,"#"
LFA_SHARP .WORD NFA_SIGN
CFA_SHARP .WORD DOCOL
PFA_SHARP
.WORD CFA_BASE ; BASE
.WORD CFA_FETCH ; @
.WORD CFA_MMOD ; M/MOD
.WORD CFA_ROT ; ROT
.WORD CFA_CLIT ; CLIT
.BYTE 9
.WORD CFA_OVER ; OVER
.WORD CFA_LT ; <
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL53-*
.WORD CFA_CLIT ; CLIT
.BYTE 7
.WORD CFA_PLUS ; +
LBL53
.WORD CFA_CLIT ; CLIT
.BYTE 48
.WORD CFA_PLUS ; +
.WORD CFA_HOLD ; HOLD
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> #S <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( #S [ ud1 -> ud2 ] fuegt solange Ziffern an den im Aufbau befindlichen    )
;( Zahlenstring an, bis der Rest, der als ud2 uebergeben wird, 0 geworden   )
;( ist.                                                                     )
;( ======================================================================== )
;
; : #S
;  BEGIN
```

```
;    #                ( eine Ziffer erzeugen und an den String anfuegen  )
;    OVER OVER OR 0=   ( solange bis der verbleibende Rest 0 geworden ist )
;  UNTIL
; ;

NFA_SHARPS .CBYTE $82,"#S"
LFA_SHARPS .WORD NFA_SHARP
CFA_SHARPS .WORD DOCOL
PFA_SHARPS
LBL54
.WORD CFA_SHARP ; #
.WORD CFA_OVER ; OVER
.WORD CFA_OVER ; OVER
.WORD CFA_OR ; OR
.WORD CFA_NULLEQUAL ; 0=
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL54-*
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>> D.R <<<<<<<<<<<<<<<
;( ======================================================================== )
;( D.R [ d n -> ] gibt die doppeltgenaue Zahl d rechtsbuendig in einem Feld  )
;( der Breite n auf die Standardausgabe aus.                                )
;( ======================================================================== )
;
; : D.R
;  >R                ( Feldbreite n auf den Return-Stack retten             )
;  SWAP OVER         ( [ hi d ] erzeugen; hi enthaelt das Vorzeichen von d  )
;  DABS              ( absoluten Betrag von d ermitteln                     )
;  <# #S SIGN #>     ( d in einen String umwandeln und Vorzeichen beruecks. )
;  R> OVER - SPACES  ( zunaechst entsprechend viele Blanks ausgeben         )
;  TYPE              ( anschliessend erzeugten String ausgeben              )
; ;

NFA_DDOTR .CBYTE $83, "D.R"
LFA_DDOTR .WORD NFA_SHARPS
CFA_DDOTR .WORD DOCOL
PFA_DDOTR
.WORD CFA_RPUSH ; >R
.WORD CFA_SWAP ; SWAP
.WORD CFA_OVER ; OVER
.WORD CFA_DABS ; DABS
.WORD CFA_LTSHARP ; <#
.WORD CFA_SHARPS ; #S
.WORD CFA_SIGN ; SIGN
.WORD CFA_SHARPGT ; #>
.WORD CFA_RPOP ; R>
.WORD CFA_OVER ; OVER
.WORD CFA_MINUS ; -
.WORD CFA_SPACES ; SPACES
.WORD CFA_TYPE ; TYPE
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>> .R <<<<<<<<<<<<<<<
;( ======================================================================== )
;( .R [ n1 n2 -> ] gibt die Zahl n1 rechtsbuendig in einem Feld der Breite   )
;( n2 auf die Standardausgabe aus.                                          )
;( ======================================================================== )
;
```

```
; : .R >R S>D R> D.R ;    ( Zahl n1 doppeltgenau machen und mit D.R ausgeben )

NFA_DOTR .CBYTE $82,".R"
LFA_DOTR .WORD NFA_DDOTR
CFA_DOTR .WORD DOCOL
PFA_DOTR
.WORD CFA_RPUSH ; >R
.WORD CFA_STOD ; S>D
.WORD CFA_RPOP ; R>
.WORD CFA_DDOTR ; D.R
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> D. <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( D. [ d -> ] gibt die doppeltgenaue Zahl d auf die Standardausgabe aus.    )
;( ========================================================================== )

; : D. 0 D.R SPACE ;               ( Zahl in einem Feld der Breite 0 ausgeben )

NFA_DDOT .CBYTE $82,"D."
LFA_DDOT .WORD NFA_DOTR
CFA_DDOT .WORD DOCOL
PFA_DDOT
.WORD CFA_0 ; 0
.WORD CFA_DDOTR ; D.R
.WORD CFA_SPACE ; SPACE
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> . <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( . [ n -> ] gibt die Zahl n auf die Standardausgabe aus.                   )
;( ========================================================================== )
;
;: . S>D D. ;              ( Zahl n doppeltgenau machen und mit D. ausgeben )

NFA_DOT .CBYTE $81,"."
LFA_DOT .WORD NFA_DDOT
CFA_DOT .WORD DOCOL
PFA_DOT
.WORD CFA_STOD ; S>D
.WORD CFA_DDOT ; D.
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> ? <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( ? [ addr -> ] gibt die Zahl, die an der Adresse addr gespeicher ist, aus. )
;( ========================================================================== )
;
;: ? @ . ;

NFA_QUEST .CBYTE $81, "?"
LFA_QUEST .WORD NFA_DOT
CFA_QUEST .WORD DOCOL
PFA_QUEST
.WORD CFA_FETCH ; @
.WORD CFA_DOT ; .
.WORD CFA_EXIT ; ;S

; >>>>>>>>>>>>>>>> U. <<<<<<<<<<<<<<<<
```

```
;( ========================================================================= )
;( U. [ u -> ] gibt die vorzeichenlose Zahl u auf die Standardausgabe aus.   )
;( ========================================================================= )

; : U. 0 D. ;    ( Zahl vorzeichenlos doppeltgenau machen und mit D. ausgeben )

NFA_UDOT .CBYTE $82,"U."
LFA_UDOT .WORD NFA_QUEST
CFA_UDOT .WORD DOCOL
PFA_UDOT
.WORD CFA_0 ; 0
.WORD CFA_DDOT ; D.
.WORD CFA_EXIT ; ;S
PREVLINK .= NFA_UDOT

;( ========================================================================= )
;( Definition diverser Konstanten und Variablen, die zur Verwaltung des      )
;( Disk-Systems benoetigt werden.                                            )
;( ========================================================================= )

; >>>>>>>>>>>>>>>> C/L <<<<<<<<<<<<<<<<
; 64        CONSTANT C/L         ( Zeichen pro Eingabezeile                 )

NFA_C_L .CBYTE $83, "C/L"
LFA_C_L .WORD PREVLINK
CFA_C_L .WORD DOCON
PFA_C_L .WORD 32
PREVLINK .= NFA_C_L

; >>>>>>>>>>>>>>>> CALL <<<<<<<<<<<<<<<<
;( ========================================================================= )
;( CALL  Sprung zur ASM Subroutine                                           )
;( ========================================================================= )
NFA_CALL .CBYTE $84, "CALL"
LDA_CALL .WORD PREVLINK
CFA_CALL .WORD PFA_CALL
PFA_CALL
  STX XSAVE
  LDA 0,X
  STA PFA_CALL1
  LDA 1,X
  STA PFA_CALL1+1

  .BYTE $20        ; JSR
PFA_CALL1
  .WORD 0
  LDX XSAVE
  STA 0,X
  STY 1,X
  JMP NEXT

PREVLINK .= NFA_CALL

; >>>>>>>>>>>>>>>> WORDS <<<<<<<<<<<<<<<<
;( ========================================================================= )
;( WORDS [ ] gibt die Namen aller Worte, beginnend beim Context-Volabular,   )
;( auf die Standardausgabe aus. Die Ausgabe kann mit einer beliebigen Taste  )
;( abgebrochen werden.                                                       )
;( ========================================================================= )
```

```
;: WORDS
;  128 OUT !               ( Initialisierung, damit zu Anfang ein CR erfolgt )
;  CONTEXT @ @             ( NFA des letzten Wortes im Context-Vokabular     )
;  BEGIN
;    OUT @ C/L > IF        ( falls Zeilenende ueberschritten wurde:          )
;      CR                  ( in neue Zeile gehen                             )
;      0 OUT !             ( Ausgabe in Spalte 0 der neuen Zeile beginnen    )
;    ENDIF
;    DUP ID. SPACE SPACE   ( Namen des aktuellen Wortes ausgeben             )
;    PFA LFA @             ( NFA des naechsten Wortes ermitteln              )
;    DUP 0= ?TERMINAL OR   ( Ende der Linkerkette oder Taste gedrueckt?      )
;  UNTIL                   ( ja dann fertig, sonst weiter ausgeben           )
;  DROP                    ( letzte NFA vergessen                            )
;

NFA_WORDS .CBYTE $85, "WORDS"
LFA_WORDS .WORD PREVLINK
CFA_WORDS .WORD DOCOL
PFA_WORDS
.WORD CFA_CLIT ; CLIT
.BYTE 128
.WORD CFA_OUT ; OUT
.WORD CFA_STORE ; !
.WORD CFA_CONTEXT ; CONTEXT
.WORD CFA_FETCH ; @
.WORD CFA_FETCH ; @
LBL74
.WORD CFA_OUT ; OUT
.WORD CFA_FETCH ; @
.WORD CFA_C_L ; C/L
.WORD CFA_GT ; >
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL75-*
.WORD CFA_CR ; CR
.WORD CFA_0 ; 0
.WORD CFA_OUT ; OUT
.WORD CFA_STORE ; !
LBL75
.WORD CFA_DUP ; DUP
.WORD CFA_IDDOT ; ID.
;.WORD CFA_SPACE ; SPACE
;.WORD CFA_SPACE ; SPACE
.WORD CFA_PFA ; PFA
.WORD CFA_LFA ; LFA
.WORD CFA_FETCH ; @
.WORD CFA_DUP ; DUP
.WORD CFA_NULLEQUAL ; 0=
.WORD CFA_QUERYTERMINAL ; ?TERMINAL
.WORD CFA_OR ; OR
.WORD CFA_0BRANCH ; 0BRANCH
.WORD LBL74-*
.WORD CFA_DROP ; DROP
.WORD CFA_EXIT ; ;S

PREVLINK .= NFA_WORDS

.IF FILE
;( ======================================================================= )
;( Definitionen fuer Dateibefehle                                          )
```

```
;( ========================================================================== )
FAM_R   = 4
FAM_W   = 8
FAM_RW  = FAM_R + FAM_W

IO_OPEN   = $3
IO_GETREC = $5
IO_GETCHR = $7
IO_PUTREC = $9
IO_PUTCHR = $B
IO_CLOSE  = $C

ICFLG     = $340
ICCOM     = $342
ICSTA     = $343
ICBAL     = $344
ICBAH     = $345
ICBLL     = $348
ICBLH     = $349
ICAX1     = $34A
ICAX2     = $34B

CIOV    = $E456

;( ========================================================================== )
;( Hilfsroutinen fuer die Dateibefehle                                       )
;( ========================================================================== )

GETFILEID
        LDA 0,X ; get fileid
GETFILEID0
        ASL
        ASL
        ASL
        ASL
        TAY
        RTS

JCIOV
        STX XSAVE
        TYA
        TAX
        JSR CIOV
        TXA
        TAY
        LDX XSAVE
        RTS

;( ========================================================================== )
;( Hilfswoerter fuer die Dateibefehle                                        )
;( ========================================================================== )
; >>>>>>>>>>>>>>>> R/O <<<<<<<<<<<<<<<<
; CONSTANT R/O    4

NFA_RO .CBYTE $83, "R/O"
LFA_RO .WORD PREVLINK
CFA_RO .WORD DOCON
PFA_RO .WORD FAM_R
PREVLINK .= NFA_RO
```

```
; >>>>>>>>>>>>>>>> R/W <<<<<<<<<<<<<<<<
; CONSTANT R/O    12

NFA_RW .CBYTE $83, "R/W"
LFA_RW .WORD PREVLINK
CFA_RW .WORD DOCON
PFA_RW .WORD FAM_RW
PREVLINK .= NFA_RW

; >>>>>>>>>>>>>>>> W/O <<<<<<<<<<<<<<<<
; CONSTANT R/O    8

NFA_WO .CBYTE $83, "W/O"
LFA_WO .WORD PREVLINK
CFA_WO .WORD DOCON
PFA_WO .WORD FAM_W
PREVLINK .= NFA_WO

; >>>>>>>>>>>>>>>> CIO <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( CIO [ ] Schnittstelle zur ATARI CIO                                       )
;( ======================================================================== )
NFA_CIO .CBYTE $83, "CIO"
LDA_CIO .WORD PREVLINK
CFA_CIO .WORD PFA_CIO
PFA_CIO
    JMP NEXT

PREVLINK .= NFA_CIO

; >>>>>>>>>>>>>>>> FREEIOCB <<<<<<<<<<<<<<<<
;( ======================================================================== )
;( FreeIOCB [ -- freeiocb ] ermittelt naechsten freien IOCB                  )
;( ======================================================================== )
NFA_FREEIOCB .CBYTE $88, "FREEIOCB"
LDA_FREEIOCB .WORD PREVLINK
CFA_FREEIOCB .WORD PFA_FREEIOCB
PFA_FREEIOCB
        JSR FREEIOCB0
        LSR A
        LSR A
        LSR A
        LSR A           ; div 16
        PHA             ; iocb und
        LDA #0          ; highbyte
        JMP PUSH        ; auf den Datenstack legen

FREEIOCB0
        LDA #$70        ; bei IOCB 7 anfangen
L_FREEIOCB2
        TAY
        LDA ICFLG,Y     ; Flag holen und
        CMP #$FF
        BEQ L_FREEIOCB1 ; testen ob frei (ICFLG = $FF)
        TYA             ; nicht frei!
        SEC
        SBC #$10
        BNE L_FREEIOCB2
```

```
L_FREEIOCB1
        TYA
        RTS

PREVLINK .= NFA_FREEIOCB

; >>>>>>>>>>>>>>>> CLOSE-FILE <<<<<<<<<<<<<<<<
;( ============================================================================ )
;( CLOSE-FILE  [fileid -- ior ] Datei schliessen                                 )
;( ============================================================================ )
NFA_CLOSEFILE .CBYTE $8A, "CLOSE-FILE"
LDA_CLOSEFILE .WORD PREVLINK
CFA_CLOSEFILE .WORD PFA_CLOSEFILE
PFA_CLOSEFILE
    JSR GETFILEID
    LDA #IO_CLOSE
    STA ICCOM,Y

    JSR JCIOV

    LDA ICSTA,Y
    BMI CLOSEFILE1
    LDA #0

CLOSEFILE1
    PHA
    LDA #0  ; Store ior highbyte
    JMP PUT

PREVLINK .= NFA_CLOSEFILE

; >>>>>>>>>>>>>>>> CREATE-FILE <<<<<<<<<<<<<<<<
;( ============================================================================ )
;( CREATE-FILE  [caddr u fam -- ior ] Datei anlegen                              )
;( ============================================================================ )
NFA_CREATEFILE .CBYTE $8B, "CREATE-FILE"
LDA_CREATEFILE .WORD PREVLINK
CFA_CREATEFILE .WORD PFA_CREATEFILE
PFA_CREATEFILE
    JMP NEXT

PREVLINK .= NFA_CREATEFILE

; >>>>>>>>>>>>>>>> DELETE-FILE <<<<<<<<<<<<<<<<
;( ============================================================================ )
;( DELETE-FILE  [caddr u -- ior ] Datei loeschen                                 )
;( ============================================================================ )
NFA_DELETEFILE .CBYTE $8B, "DELETE-FILE"
LDA_DELETEFILE .WORD PREVLINK
CFA_DELETEFILE .WORD PFA_DELETEFILE
PFA_DELETEFILE
    JMP NEXT

PREVLINK .= NFA_DELETEFILE

; >>>>>>>>>>>>>>>> FLUSH-FILE <<<<<<<<<<<<<<<<
;( ============================================================================ )
;( FLUSH-FILE  [fileid -- ior ] Dateibuffer schreiben                            )
;( ============================================================================ )
```

```
NFA_FLUSHFILE .CBYTE $8A, "FLUSH-FILE"
LDA_FLUSHFILE .WORD PREVLINK
CFA_FLUSHFILE .WORD PFA_FLUSHFILE
PFA_FLUSHFILE
    JMP NEXT

PREVLINK .= NFA_FLUSHFILE

; >>>>>>>>>>>>>>>> OPEN-FILE <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( OPEN-FILE  [caddr u fam -- fileid ior ] Datei oeffnen
;( ========================================================================== )
NFA_OPENFILE .CBYTE $89, "OPEN-FILE"
LDA_OPENFILE .WORD PREVLINK
CFA_OPENFILE .WORD PFA_OPENFILE
PFA_OPENFILE
        JSR FREEIOCB0   ; get free iocb
        LDA #IO_OPEN
        STA ICCOM,Y
        LDA 4,X ; Lowbyte Filename
        STA ICBAL,Y
        LDA 5,X ; HighByte Filename
        STA ICBAH,Y
        LDA 0,X ; fam
        STA ICAX1,Y
        LDA #0
        STA ICAX2,Y

        JSR JCIOV

        INX
        INX

        LDA ICSTA,Y
        STA 0,X
        BMI OPENFILE1
        DEC 0,X ; ior = 0 = ok

OPENFILE1
        LDA #0  ; Store ior
        STA 1,X
        STA 3,X

; DIV $10
        TYA
        CLC
        LSR A
        LSR A
        LSR A
        LSR A

        STA 2,X ; Store fileid
        JMP NEXT

PREVLINK .= NFA_OPENFILE

; >>>>>>>>>>>>>>>> RENAME-FILE <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( RENAME-FILE  [caddr1 u1 caddr2 u2 -- ior ] Datei umbenennen               )
```

```
;( ========================================================================== )
NFA_RENAMEFILE .CBYTE $8B, "RENAME-FILE"
LDA_RENAMEFILE .WORD PREVLINK
CFA_RENAMEFILE .WORD PFA_RENAMEFILE
PFA_RENAMEFILE
        JMP NEXT

PREVLINK .= NFA_RENAMEFILE

; >>>>>>>>>>>>>>>> READ-FILE <<<<<<<<<<<<<<<
;( ========================================================================== )
;( READ-FILE  [caddr u1 fileid -- u2 ior ] Datei an Adresse Lesen            )
;( ========================================================================== )
NFA_READFILE .CBYTE $89, "READ-FILE"
LDA_READFILE .WORD PREVLINK
CFA_READFILE .WORD PFA_READFILE
PFA_READFILE
        JSR GETFILEID
        LDA 2,X ; get lowbyte length
        STA ICBLL,Y
        LDA 3,X ; get highbyte length
        STA ICBLH,Y
        LDA 4,X ; get lowbyte address
        STA ICBAL,Y
        LDA 5,X ; get highbyte address
        STA ICBAH,Y
        LDA #IO_GETCHR ; Get Characters CIO Command
        STA ICCOM,Y

        JSR JCIOV
        INX
        INX

        LDA ICSTA,Y
        STA 0,X

        BMI READFILE1
        DEC 0,X     ; OK, ior = 0

READFILE1
        CLC
        LDA ICBLL,Y ; get real length
        STA 2,X
        LDA ICBLH,Y ; get real length
        STA 3,X

        LDA #0
        STA 1,X
        JMP NEXT

PREVLINK .= NFA_READFILE

; >>>>>>>>>>>>>>>> READ-LINE <<<<<<<<<<<<<<<
;( ========================================================================== )
;( READ-LINE  [caddr u1 fileid -- u2 flag ior ] Datei zeilenweise lesen     )
;( ========================================================================== )
NFA_READLINE .CBYTE $89, "READ-LINE"
LDA_READLINE .WORD PREVLINK
CFA_READLINE .WORD PFA_READLINE
```

```
PFA_READLINE
        JSR GETFILEID
        LDA 2,X ; get lowbyte length
        STA ICBLL,Y
        LDA 3,X ; get highbyte length
        STA ICBLH,Y
        LDA 4,X ; get lowbyte address
        STA ICBAL,Y
        LDA 5,X ; get highbyte address
        STA ICBAH,Y
        LDA #IO_GETREC ; Get Record CIO Command
        STA ICCOM,Y

        JSR JCIOV

        LDA ICSTA,Y
        STA 0,X
        BMI READLINE1
        DEC 0,X     ; OK, ior = 0

READLINE1
        CLC
        LDA ICBLL,Y ; get real length
        STA 4,X
        ADC ICBAL,Y ; calculate end of string
        STA 2,X
        DEC 2,X

        LDA ICBLH,Y
        STA 5,X
        ADC ICBAH,Y
        STA 3,X

        LDA #0
        STA (2,X)

        STA 1,X
        STA 2,X     ; store flag = true
        STA 3,X
        JMP NEXT

PREVLINK .= NFA_READLINE

; >>>>>>>>>>>>>>>> REFILL <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( REFILL  [ fileid -- flag ] Eingabebuffer fuellen                                )
;( ========================================================================== )
; : REFILL
;   TIB @ 128 SOURCE-ID @ READ-LINE SWAP IN ! SWAP DROP
; ;
NFA_REFILL .CBYTE $86, "REFILL"
LDA_REFILL .WORD PREVLINK
CFA_REFILL .WORD DOCOL
PFA_REFILL
.WORD CFA_TIB
.WORD CFA_FETCH         ; get Terminal Input Buffer (TIB)
.WORD CFA_LIT
.WORD $80               ; 80 Char. max
.WORD CFA_SOURCEID
```

```
.WORD CFA_FETCH         ; get Channel
.WORD CFA_READLINE
.WORD CFA_SWAP
.WORD CFA_IN
.WORD CFA_STORE
.WORD CFA_SWAP
.WORD CFA_DROP
.WORD CFA_EXIT


PREVLINK .= NFA_REFILL

; >>>>>>>>>>>>>>>> WRITE-FILE <<<<<<<<<<<<<<<<
;( ============================================================================ )
;( WRITE-FILE  [caddr u fileid -- ior ] Datei von Adresse schreiben           )
;( ============================================================================ )
NFA_WRITEFILE .CBYTE $8A, "WRITE-FILE"
LDA_WRITEFILE .WORD PREVLINK
CFA_WRITEFILE .WORD PFA_WRITEFILE
PFA_WRITEFILE
        JSR GETFILEID
        LDA #IO_PUTCHR ; Put Characters CIO Command
        STA ICCOM,Y
WRITE

        LDA 2,X ; get lowbyte length
        STA ICBLL,Y
        LDA 3,X ; get highbyte length
        STA ICBLH,Y
        LDA 4,X ; get lowbyte address
        STA ICBAL,Y
        LDA 5,X ; get highbyte address
        STA ICBAH,Y

        JSR JCIOV

        INX
        INX
        INX
        INX

        LDA ICSTA,Y
        STA 0,X
        BMI WRITEFILE1
        DEC 0,X     ; OK, ior = 0

WRITEFILE1

        LDA #0
        STA 1,X
        JMP NEXT



PREVLINK .= NFA_WRITEFILE

; >>>>>>>>>>>>>>>> WRITE-LINE <<<<<<<<<<<<<<<<
;( ============================================================================ )
;( WRITE-LINE  [caddr u fileid -- ior ] Datei zeilenweise schreiben           )
```

```
;( ========================================================================== )
NFA_WRITELINE .CBYTE $8A, "WRITE-LINE"
LDA_WRITELINE .WORD PREVLINK
CFA_WRITELINE .WORD PFA_WRITELINE
PFA_WRITELINE
        JSR GETFILEID
        LDA #IO_PUTREC ; Put Record CIO Command
        STA ICCOM,Y
        JMP WRITE

PREVLINK .= NFA_WRITELINE


; >>>>>>>>>>>>>>>> INCLUDE-FILE <<<<<<<<<<<<<<<
;( ========================================================================== )
;( INCLUDE-FILE  [fileid -- ] Liest und interpretiert Datei aus fileid       )
;( ========================================================================== )
; : INCLUDE-FILE
;   SOURCE-ID !
;   BEGIN
;     REFILL
;     128 <
;     WHILE
;       INTERPRET
;     REPEAT
;   SOURCE-ID @ CLOSE-FILE
;   QUIT ;

NFA_INCLUDEFILE .CBYTE $8C, "INCLUDE-FILE"
LDA_INCLUDEFILE .WORD PREVLINK
CFA_INCLUDEFILE .WORD DOCOL
PFA_INCLUDEFILE
.WORD CFA_SOURCEID              ; SOURCE-ID
.WORD CFA_STORE                 ; !
INCLUDEFILE1                    ; BEGIN
.WORD CFA_REFILL                ; REFILL
.WORD CFA_LIT                   ; LITERAL
.WORD $80                       ; 128
.WORD CFA_LT                    ; <
.WORD CFA_0BRANCH               ; WHILE
.WORD INCLUDEFILE2-*
.WORD CFA_INTERPRET             ; INTERPRET
.WORD CFA_BRANCH
.WORD INCLUDEFILE1-*            ; REPEAT
INCLUDEFILE2                    ;
.WORD CFA_SOURCEID              ; SOURCE-ID
.WORD CFA_FETCH                 ; @
.WORD CFA_CLOSEFILE             ; CLOSE-FILE
.WORD CFA_QUIT                  ; QUIT
.WORD CFA_EXIT                  ; S;

PREVLINK .= NFA_INCLUDEFILE

; >>>>>>>>>>>>>>>> INCLUDED <<<<<<<<<<<<<<<
;( ========================================================================== )
;( INCLUDED  [caddr u -- ] Liest und interpretiert Datei - Name an caddr     )
;( ========================================================================== )
; : INCLUDED
;   R/O
```

```
;   OPEN-FILE
;   128 < IF
;     INCLUDE-FILE
;     SOURCE-ID @ CLOSE-FILE
;      0 SOURCE-ID !            ( Input-Buffer SOURCE-ID zuruecksetzen )
;   ENDIF

NFA_INCLUDED .CBYTE $88, "INCLUDED"
LDA_INCLUDED .WORD PREVLINK
CFA_INCLUDED .WORD DOCOL
PFA_INCLUDED
.WORD CFA_RO
.WORD CFA_OPENFILE
.WORD CFA_LIT
.WORD $80
.WORD CFA_LT
.WORD CFA_0BRANCH
.WORD INCLUDED1-*
.WORD CFA_INCLUDEFILE
.WORD CFA_SOURCEID
.WORD CFA_FETCH
.WORD CFA_CLOSEFILE
.WORD CFA_0
.WORD CFA_SOURCEID
.WORD CFA_STORE
INCLUDED1
.WORD CFA_EXIT        ; S;

PREVLINK .= NFA_INCLUDED

; >>>>>>>>>>>>>>>> FILE" <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( FILE" [ -- u caddr ] Ermittelt Filename im Input Strom                     )
;( ========================================================================== )
; : FILE"
;   1536 BL BLANKS
;   HERE BL BLANKS
;   34 WORD HERE COUNT 1536 SWAP CMOVE
;   1536 0 ;

NFA_FILE .BYTE $85,"FILE",$A2
LDA_FILE .WORD PREVLINK
CFA_FILE .WORD DOCOL
PFA_FILE

.WORD CFA_LIT       ;
.WORD 1536          ; $600
.WORD CFA_BL        ; BL (32)
.WORD CFA_BLANKS    ; BLANKS
.WORD CFA_HERE      ; HERE
.WORD CFA_BL        ; BL (32)
.WORD CFA_BLANKS    ; BLANKS
.WORD CFA_CLIT
.BYTE 34            ; 34 (")
.WORD CFA_WORD      ; WORD
.WORD CFA_COUNT     ; COUNT
.WORD CFA_LIT
.WORD 1536          ; $600
.WORD CFA_SWAP      ; SWAP
```

```
.WORD CFA_CMOVE     ; CMOVE
.WORD CFA_LIT
.WORD 1536          ; $600
.WORD CFA_0         ; 0
.WORD CFA_EXIT      ; S;

PREVLINK .= NFA_FILE

; >>>>>>>>>>>>>>>> INCLUDE" <<<<<<<<<<<<<<<<
;( ========================================================================== )
;( INCLUDE" Liest und interpretiert Datei - Name folgt im Input Strom        )
;( ========================================================================== )
; : INCLUDE"
;   FILE" INCLUDED ;
;   IMMEDIATE

NFA_INCLUDE .BYTE $C8,"INCLUDE",$A2
LDA_INCLUDE .WORD PREVLINK
CFA_INCLUDE .WORD DOCOL
PFA_INCLUDE
.WORD CFA_FILE      ; FILE"
.WORD CFA_INCLUDED  ; INCLUDED
.WORD CFA_EXIT      ; ;S

PREVLINK .= NFA_INCLUDE

.ENDIF ; FILE COMMANDS

; >>>>>>>>>>>>>>>> MON <<<<<<<<<<<<<<<<

NFA_MON .CBYTE $83, "MON"
LFA_MON .WORD PREVLINK
CFA_MON .WORD PFA_MON
PFA_MON


  ; Im originalen FIG-Listing wird davon ausgegangen, dass das aufrufende
  ; Programm ein Monitor ist.

 STX XSAVE        ; Datenstackpointer retten

 LDA DOSIN+1      ; Resetvector zurueck
 STA DOSINI
 LDA DOSIN+2
 STA DOSINI+1

 JMP (DOSVEC)     ; zum DOS zurueckkehren
 NOP              ; nach BRK wird ein Byte uebersprungen
 LDX XSAVE        ; Datenstackpointer wiederherstellen
 JMP NEXT         ; und fertig


NFA_LASTWORD = NFA_MON

;-------------------------------------------------------------------------------
; Hilfsroutinen
;-------------------------------------------------------------------------------

GETCH
```

```
    TYA         ; Y Register retten
    PHA
    JSR GETCHK

    TAX
    PLA         ; Y Register wiederherstellen
    TAY
    TXA
    RTS

GETCHK
    LDA $E425   ;  PUSH address of GETKEY OS Routine
    PHA     ;  to returnstack
    LDA $E424
    PHA
    RTS

OUTCH
    STX XSAVE
    TAX
    TYA         ; Y Register retten
    PHA
    JSR OUTCHK

    PLA         ; Y Register wiederherstellen
    TAY
    LDX XSAVE
    RTS

OUTCHK      ; in das ATARI OS springen (CIO Routine)
    LDA $E407
    PHA
    LDA $E406
    PHA
    TXA
    RTS

.IF debug

;
;    This is a temporary trace routine, to be used until FORTH
;    is generally operating. Then NOP the terminal query
;    "JSR ONEKEY". This will allow user input to the text
;    interpreter. When crashes occur, the display shows IP, W,
;    and the word locations of the offending code. When all is
;    well, remove : TRACE, TCOLON, PRNAM, DECNP, and the
;    following monitor/register equates.
;
;
;
;    Monitor routines needed to trace.
;
;XBLANK    EQU $D0AF         ; print one blank
;CRLF      EQU $D0D2         ; print a carriage return and line feed.
;HEX2      EQU $D2CE         ; print accum as two hex numbers
;LETTER    EQU $D2C1         ; print accum as one ASCII character
;ONEKEY    EQU $D1DC         ; wait for keystroke
XW        = $43           ; scratch reg. to next code field add
NP        = $45           ; scratch reg. pointing to name field
```

```
;
;
DEBUGFLG  .BYTE 0       ; debugflg
DXSAVE    .BYTE 0       ; X Register Sicherungsplatz

TRACE     STX DXSAVE     ; X Register Retten
          PHA           ; Accu retten
          TYA
          PHA           ; Y Register retten
          JSR CRLF
          LDA IP+1
          JSR HEX2
          LDA IP
          JSR HEX2       ; print IP, the interpreter pointer
          JSR XBLANK
;
;
          LDY #0
          LDA (IP),Y
          STA XW
          STA NP         ; fetch the next code field pointer
          INY
          LDA (IP),Y
          STA XW+1
          STA NP+1
          JSR PRNAM      ; print dictionary name
;
          LDA XW+1
          JSR HEX2       ; print code field address
          LDA XW
          JSR HEX2
          JSR XBLANK
;
          LDX DXSAVE      ; print TOS Cell
          LDA 1,X
          JSR HEX2
          LDX DXSAVE
          LDA 0,X
          JSR HEX2
          JSR XBLANK

;
          LDA DXSAVE      ; print stack location in zero-page
          JSR HEX2
          JSR XBLANK
;
          LDA #1         ; print return stack bottom in page 1
          JSR HEX2
          TSX
          INX
          TXA
          JSR HEX2
          JSR XBLANK
;
.IF keywait
          JSR ONEKEY     ; wait for operator keystroke
.ENDIF
          LDX DXSAVE      ; just to pinpoint early problems
```

```
          PLA            ; Y Register wiederherstellen
          TAY
          PLA            ; Accu wiederherstellen
          RTS
;
;    TCOLON is called from DOCOLON to label each point
;    where FORTH 'nests' one level.
;
TCOLON    STX DXSAVE     ; X Register retten
          PHA           ; Accu retten
          TYA
          PHA           ; Y Register retten
          LDA W
          STA NP         ; locate the name of the called word
          LDA W+1
          STA NP+1
          JSR CRLF
          LDA #$3A       ; ':
          JSR LETTER
          JSR XBLANK
          JSR PRNAM
;@        JSR ONEKEY     ; wait for operator keystroke
          LDX DXSAVE     ; X Register weiderherstellen
          PLA
          TAY           ; Y Register wiederherstellen
          PLA           ; Accu wiederherstellen
          RTS
;
;    Print name by it's code field address in NP
;
PRNAM
ysave =   $3FF

          JSR DECNP
          JSR DECNP
          JSR DECNP
          LDY #0
PN1
          JSR DECNP
          LDA (NP),Y     ; loop till D7 in name set
          BPL PN1
PN2       INY
          LDA (NP),Y
          STY YSAVE
          JSR LETTER     ; print letters of name field
          LDY YSAVE
          LDA (NP),Y
          BPL PN2
          JSR XTAB
          LDY #0
          RTS
;
;    Decrement name field pointer
;
DECNP     LDA NP
          BNE DECNP1
          DEC NP+1
DECNP1    DEC NP
          RTS
```

```
;
;*********************************************************
;    Monitor routines needed to trace.
;
; print a carriage return and line feed.
;
CRLF
        LDA     #155
        JMP     OUTCH

; print one blank
;
XBLANK  LDA     #$20
        JMP     OUTCH
; print one TAB
;
XTAB    LDA     #127
        JMP     OUTCH

; print accum as two hex digits
HEX2    PHA
        LSR     A
        LSR     A
        LSR     A
        LSR     A
        JSR     HEX2A
        PLA
HEX2A   AND     #$0F
        JSR     HXDGT
        JMP     OUTCH
;
;convert hex digit to ASCII
;
HXDGT   CMP     #$0A
        BCC     HXDGT1
        CLC
        ADC     #7
HXDGT1  ADC     #'0
        RTS

;
; print accum as one ASCII character
;
LETTER
         AND     #$7F
         CMP     #$20            ;compare ASCII space
         BCS     LETTER1         ;good if >= ' '
         LDA     #'.
LETTER1 JMP      OUTCH

;
; wait for keystroke
;
ONEKEY  JMP     GETCH

.ENDIF ; DEBUG

  TOP .END ; Ende des Assemblerlistings
```

```
.BANK
* = $2E0
.WORD ORIG
```