```
\ FORTH83 für Super 8 ( Zilog )                 ( 21.11.90/KK )

Erstellt:       16.06.90 ( Klaus Kohl: Version 1.0 )

                Hinweis:
 - Enthält fast vollen FORTH-83-Wortumfang ( FORGET fehlt )
 - ROM-fähig durch Auslagerung von DEFER und Variablen
 - Verwaltung eines HEAP's implementiert
 - Volle Vokabular-Verwaltung nach FORTH-83-Vorschlag

                Letzte Änderungen:
 21.11.90  Korrekturen in (ERROR und DUMP (mit Kürzungen)




\ LOADSCREEN                                    ( 16.06.90/KK )

 &004 &006 THRU         \ Vorbereitung u. Systemkonstanten
\ &114 &118 THRU         \ !T! Tracer laden
 &007 &016 THRU         \ Systeminitialisierung
 &017 &027 THRU         \ Kernroutinen NEXT ... (LOOP
 &028 &034 THRU         \ Konstanten, USER, Variablen, ORIGIN
 &035 &062 THRU         \ weitere Definitionen  C@ ... BLANK
\ &119 LOAD             \ !T! Ein-/Ausschalten des Tracers
 &063 &111 THRU         \ restliches FORTH
 &112 &113 THRU         \ Systemeinstellungen




\\ INFO: Registerbelegung dieses S8-FORTH       ( 16.06.90/KK )

 $D8/$D9        FRP             FORTH-Return-Stack-Pointer
 $DA/$DB        IP              FORTH-Instructions-Pointer

 Als Arbeitsbereich werden immer eine Registergruppe verwendet:
 + E/ F: RR14   AX (AH, AL)     Top of Stack
 + C/ D: RR12   BX (BH, BL)     frei  \
 + A/ B: RR10   CX (CH, CL)     frei   Arbeitsregister
 + 8/ 9: RR8    DX (DH, DL)     frei  /
 + 6/ 7: RR6    FSP             FORTH-Datenstack-Pointer
 + 4/ 5: RR4    FUP             FORTH-USER-Pointer
 + 2/ 3: RR2    FFP             FORTH-Floatingpoint-Pointer
 + 0/ 1: RR0    TEMPX           ( Nicht als Zeiger verwendbar )

Achtung: Immer SB0 eingestellt lassen
\\ INFO: Speicheraufteilung                     ( 13.05.90/KK )

 $0000 ... $001F                ( Interupt-Tabelle )
 $0020 ... DP                   ( FORTH-Kern )
 DP ...                         ( Variablen, Tasks )
 $???? ...                      ( RAM im Testgerät )
 VARORG ... HDP                 ( Variablen-Bereich )
 HDP ... TASKORG                ( HEAP )
 TASKORG ... DISKORG            ( alle TASK-Bereiche )
  ( SLEN bis Task, $80 Bytes für Tasks, RLEN danach )
  ( ..84..PAD..84..64..S0..6..TASK..128..TIB..82..64..R0 )
 DISKORG ... $FFAF              ( Diskettenbuffer )
 $FFB0 ... $FFCF                ( Tabellen für FORTH )
 $FFD0 ... $FFFF                ( Interrupt-Einsprünge )
```

```
\ Vorbereitung und einige Systemkonstanten      ( 19.05.90/KK )

 { $0000 $8000 $FF TFILL  }     \ Target mit $FF füllen
 { $8000 $8000 $FF TFILL  }     \ Target mit $FF füllen

 { $0000 TDP           !  }     \ Programm-Anfang
 { $8000 TVOC          !  }     \ Vokabular-Zeiger setzen
 { $F9AE TVAR          !  }     \ Variablenanfang
 {     0 TVAR_OFFSET   !  }     \ noch keine Var's

  $FFB2 >L: >RAMORG             \ Zeiger auf Tabellenstart
  $FFB6 >L: >DP                 \ Dictionary-Pointer
  $FFB8 >L: >VDP                \ Variablen-Offset
  $FFBE >L: >TASKORG            \ Zeiger auf Taskanfang
  $FFC2 >L: >TASK#              \ Anzahl der Tasks

\ Interrupt-Vektoren, Init                      ( 16.06.90/KK )

\ Alle Interupt-Vektoren zeigen auf $FFD0 bis $FFFF
  $FFD0 , $FFD3 , $FFD6 , $FFD9 ,
  $FFDC , $FFDF , $FFE2 , $FFE5 ,
  $FFE8 , $FFEB , $FFEE , $FFF1 ,
  $FFF4 , $FFF7 , $FFFA , $FFFD ,


  ASSEMBLER  L: JPINIT  $007A JR,  FORTH \ Sprung über Tabellen
\ ASSEMBLER  L: JPINIT  $007A JP,  FORTH \ !T! für Trace

   $00 C, $01 C, $00 C, $00 C,  \ Versionsnummer 01.0/0
   $14 C, $C7 C, $9B C, $C0 C,  \ 07.06.90  19:30:00
   L: INITVAL  $50 ALLOT        \ Standard-Startwerte

\\ NEXT  (DOCOL                                 ( 16.06.90/KK )
\ !T! nur für DEBUG-Zwecke aktiviert
\ !T! Deaktiviert, wenn Opcodes direkt eingesetzt werden
  ASSEMBLER
  L: NEXT       ( -- addr ) ( Kernroutine )
        FRP INCW,   FRP INCW,       ( Returnstack korrigieren )
        BX , $FFCE LDCW   @BX JP,   ( Sprung nach NEXT0 ? )

  L:   (DOCOL   ( -- ) ( Highlevel-Wort starten )
        BX , IP LDW,            ( IP nach BX bringen )
        IPH POP,   IPL POP,     ( Adresse nach IP )
        BL PUSH,   BH PUSH,     ( IP auf Returnstack bringen )
        NEXT,                   ( dann NEXT )
  FORTH


\ Init-Routine Teil 1: Interrupt, I/O           ( 19.05.90/KK )

\   HERE  JPINIT 1 +  ! ( !T! Sprung-Patch zu Init-Start )
 ASSEMBLER
    EI,                 ( Interrupt erlauben )
    DI,                 ( und gleich wieder blockieren )

    SB0,                ( Bank 0 )
    # $C0 SRP,          ( Workreg. $Cx )

    P0   , # $00 LD,    ( Port 0 Adresse beginnt bei 0 )
    P0M  , # $FF LD,    ( Port 0 Low Adress )
    PM   , # $30 LD,    ( Port 1 High Adress )

    H0C  , # $00 LD,
    H1C  , # $00 LD,    ( no Handshake )
```

```
\ Init-Routine Teil 2: Port, Interrupt          ( 13.05.90/KK )

    P2   , # $00 LD,    ( clr P2 )
    P3   , # $00 LD,    ( clr P3 )
    P2AM , # $8A LD,    ( P31,P20,P21 output; P30 input )
    P2BM , # $AA LD,    ( P32,P33,P22,P23 output )
    P2CM , # $AA LD,    ( P34,P35,P24,P25 output )
    P2DM , # $AA LD,    ( P36,P37,P26,P27 output )
    P4   , # $00 LD,    ( clr P4 )
    P4D  , # $FF LD,    ( P4 all input )
    P4OD , # $AA LD,    ( active push/pull )
    SYM  , # $00 LD,    ( disable fast interrupt )
    IPR  , # $10 LD, ( IRQ0>IRQ1>IRQ2>IRQ3>IRQ4>IRQ5>IRQ6>IRQ7)
    IMR  , # $00 LD,    ( disable all interrupt )
    IRQ  , # $00 LD,    ( clear   all interrupt )

\ Init-Routine Teil 3: UART                     ( 13.05.90/KK )

    SB1,                ( Bank 1 )
    UMA , # $70 LD,     ( X16, 8 Bits )
    UBGH , # $0F LDW,   ( 9600 Baud )
    UMB , # $1E LD,     ( use Baudrategenerator )

    SB0,                ( Bank 0 )
    UTC , # $88 LD,     ( enable transmit )
    UIE , # $00 LD,     ( disable all interrupt )
    URC , # $02 LD,     ( enable receive )

\   TEMPH , # $DF LD,   @TEMPH , # $E0 LD,   ( init Write Ptr )
\   TEMPH , # $DE LD,   @TEMPH , # $E0 LD,   ( init Read  Ptr )
\   TEMPH , # $DD LD,   @TEMPH , # $00 LD,   ( init Count     )

\ Init-Routine Teil 4: RAM-Anfang ermitteln     ( 19.05.90/KK )

        EMT , # 02 OR,                          ( ext. Stack )
        FRP , # >RAMORG 2 - LDW,                ( Stack=Disk )

\ RAM-Ende in BX ; AH=Highbyte von RAM-Ende - 1
        BX , # $FFFF LDW,   AL CLR,           ( Offset, AL=0 )
 L: B1:: @BX , AL LDC,   AH , @BX LDC,
        AH , AL CP,   NZ , 1$ JP,   AL DEC,
        @BX , AL LDC,   AH , @BX LDC,
        AH , AL CP,   NZ , 1$ JP,   AL INC,
        BL , # 1 SUB,   BH , # 0 SBC,   NC , B1:: JR,
    1$: AH , BH LD,   BX INCW,



\ Init-Routine Teil 5: ROM-Tabelle ermitteln    ( 13.05.90/KK )

\ AX ab der nächsten 1k-Grenze abwärts nach $A55A suchen lassen
        AH , # $FC AND,   AL CLR,
 L: B2:: CX , @AX LDCW   CH , # $A5 CP,   NZ , 2$ JP,
                         CL , # $5A CP,   Z , 3$ JP,
    2$: AH , # 4 SUB,   NC , B2:: JR,

\ Nichts gefunden: AX=INITVAL-Tabelle
        AX , # INITVAL LDW,
```

```
\ Init-Routine Teil 6: Tabelle übernehmen       ( 19.05.90/KK )

\ AX=Tabellenanfang; BX=RAM-Anfang

\ Tabelle an das Speicherende bringen
    3$: DX , AX LDW,                ( damit AX erhalten bleibt )
        CX , # >RAMORG 3 - LDW,   TEMPL , # $50 LD,
 L: B3:: TEMPH , @DX LDCI,   @CX , TEMPH LDCPI,
        TEMPL , B3:: DJNZ,          ( Tabelle ans Speicherende )

\ Test, ob eine Verschiebung des Programms notwendig ist
        TEMPX , 2 (AX LDCW  TEMPL , AL SUB,  TEMPH , AH SBC,
        TEMPH , TEMPL OR,   Z , 4$ JP,       ( Adresse stimmt )



\ Init-Routine Teil 7: Prg. übernehmen ?        ( 13.05.90/KK )

\ EPROM ins RAM kopieren, AX zeigt danach auf Variablen
\ ! Der tatsächliche Speicheranfang wird dabei ignoriert !
        BX , 2 (AX LDCW   CX , 6 (AX LDCW           ( RAM, DP )
        CL , AL SUB,   CH , AH SBC,   BX DECW, ( Addr., Länge )
 L: B4:: DH , @AX LDCI,   @BX , DH LDCPI,
        CX DECW,   NZ , B4:: JR,
        5$ JP,                    ( dann Variablen übernehmen )

\ Oder Tabelle an RAM-Anfang u. AX/DP korrigieren
    4$: >RAMORG , BX LDCW                    ( RAMORG setzen )
        BX DECW,   CX , # >RAMORG 2 - LDW,   DL , # $50 LD,
 L: B5:: DH , @CX LDCI,   @BX , DH LDCPI,   DL , B5:: DJNZ,
        BX INCW,   >DP , BX LDCW                  ( DP setzen )
        CX , 6 (AX LDCW   AX , CX LDW,     ( Variablenadresse )
\ Init-Routine Teil 8: Var. u. HEAP kopieren    ( 19.05.90/KK )

\ AX=Variablenanfang
\ Die Zieladresse steht in VDP
\ Länge = TASKORG - VDP
    5$: BX , >VDP LDCW                          ( Zieladresse )
        CX , >TASKORG LDCW
        CL , BL SUB,   CH , BH SBC,                   ( Länge )
        BX DECW,                                ( wegen LDCPI )
 L: B6:: DH , @AX LDCI,   @BX , DH LDCPI,
        CX DECW,   NZ , B6:: JR,




\ Init-Routine Teil 9: Taskbereich kopieren     ( 13.05.90/KK )
\ AX zeigt auf ersten TASK-Bereich; BX ins RAM
        CX , >TASK# LDCW              ( Erste Registernummer )
\ SLEN freilassen
 L: B7:: DX , $30 (AX LDCW   BL , DL ADD,  BH , DH ADC,
\ Taskadresse auch in das Register übernehmen
        BX INCW,   4 (CL , BH LD,   5 (CL , BL LD,  ( bis $Bx )
                   FUP , BX LDW,   BX DECW,        ( wenn $Cx )
\ Taskbereiche kopieren
        DX , $32 (AX LDCW                        ( RLEN holen )
        CH , # $80 LD,                ( Task-Bereich kopieren )
 L: B8:: TEMPH , @AX LDCI,  @BX , TEMPH LDCPI,  CH , B8:: DJNZ,
\ RLEN freilassen
        BL , DL ADD,  BH , DH ADC,            ( RLEN addieren )
\ weiter bis auch TASK$C0 kopiert wurde
        CL , # $10 ADD,   CL , # $D0 CP,   ULT , B7:: JR,
```

```
\ Init-Routine Schluß: FORTH-Start              ( 19.05.90/KK )

\ Stacks initialisieren, dann zu COLD+3
       FSP , $10 (FUP LDCW                                ( S0 )
       AX POPW                        ( TOS in den ACCU holen )
       AX  , $12 (FUP LDCW   FRP , AX LDW,                ( R0 )
  L: COLDVEC
       IP  , # 0 LDW,                 ( IP = Warmstartadresse )
       NEXT,

  FORTH




\ EXIT  EXECUTE                                  ( 19.05.90/KK )

  CODE EXIT     ( -- ) ( Ende eines Highlevel-Wortes )
       EXIT,                    ( Direkter Opcode )
\      IPH POP,   IPL POP,      ( !T! IP aus Returnstack laden)
\      NEXT,                    ( !T! dann NEXT ( für Debugger)
  END-CODE

  CODE EXECUTE  ( cfa -- ) ( Ausführen )
       BX , AX LDW,   AX POPW   @BX JP,                END-CODE




( Zusatzroutinen )                               ( 15.05.90/KK )

 ASSEMBLER
  L: IP++->CX   ( -- ) ( Inline-Literal mittels BX nach CX )
       BX , IP LDW,                      ( IP nach BX )
       CH , @BX LDCI,   CL , @BX LDCI, ( <BX>++ nach CX )
       IP , BX LDW,   RET,               ( BX nach IP )
  L: CX->BXVAR  ( -- ) ( CX zeigt auf Offset -- BX=addr )
       BX , @CX LDCW   CX , >VDP LDCW
       BL , CL ADD,   BH , CH ADC,   RET,
 FORTH




\ (2CON  (CON                                    ( 13.05.90/KK )

 ASSEMBLER
\  L: (2CON      ( -- d ) ( Assemblerroutine zu 2CONSTANT )
\       AX PUSHW   BH POP,   BL POP,     ( Konstantenadresse )
\       AH , @BX LDCI,   AL , @BX LDCI,   AX PUSHW      ( Low )
\       AH , @BX LDCI,   AL , @BX LDCI,   NEXT,        ( High )
  L: (CON       ( -- n ) ( Assemblerroutine zu CONSTANT )
       AX PUSHW   BH POP,   BL POP,     ( Konstantenadresse )
       AH , @BX LDCW   NEXT,
 FORTH
```

```
\ (DIC   (VAR                                    ( 13.05.90/KK )

 ASSEMBLER
  L: (DIC       ( -- addr ) ( Nachfolgende Adresse )
        AX PUSHW   AH POP,   AL POP,   NEXT,
  L: (VAR       ( -- addr ) ( Variablenadresse zum Stack )
        AX PUSHW
        CH POP,   CL POP,   CX->BXVAR CALL,
        AX , BX LDW,   NEXT,
 FORTH




\ (DOES>  (USER                                  ( 15.05.90/KK )

 ASSEMBLER
  L: (DOES>     ( addr -- ; R: -- addr2 ) ( Codeteil zu DOES> )
        BX , IP LDW,            ( IP nach BX bringen )
        IPH POP,   IPL POP,     ( Adresse nach IP )
        AX PUSHW   AH POP,   AL POP,    ( Adresse als TOS )
        BL PUSH,   BH PUSH,     ( IP auf Returnstack bringen )
        NEXT,                   ( dann NEXT )
  L: (USER      ( -- addr ) ( Assemblerroutine zu USER )
        AX PUSHW   CH POP,   CL POP,    ( Konstantenadresse )
        BL , @CX LDC,    BH , # 0 LD,   ( Offset holen )
        BL , FUPL ADD,   BH , FUPH ADC, ( + Taskadresse )
        AX , BX LDW,   NEXT,
 FORTH

\ (DEFER  (IS                                    ( 15.05.90/KK )
 ASSEMBLER
  L: (DEFER     ( ??? ) ( Assemblerroutine zu DEFER )
        CH POP,   CL POP,          ( Zeiger auf PFA des DEFER )
        CX->BXVAR CALL,   CX , @BX LDCW           ( CFA holen )
        @CX JP,                             ( Befehl ausführen )
 FORTH
| CODE (IS      ( csa -- ; es folgt DEFER-Wort )
        IP++->CX CALL,                  ( Inline-Lit. -> CX )
        CL , # 3 ADD,  CH , # 0 ADC,    ( zur PFA )
        CX->BXVAR CALL,                 ( zur Variablen-Addr. )
        @BX , AX LDCW                   ( CFA zur PFA )
  END-CODE
 ASSEMBLER
  L: (DROP   AX POPW   NEXT,            ( Neuer TOS holen )
 FORTH
\ (LIT  ("                                       ( 16.05.90/KK )

| CODE (LIT     ( -- n ) ( Codeteil zu LITERAL )
        IP++->CX CALL,                  ( Inline-Lit. -> CX )
        AX PUSHW                        ( TOS sichern )
        AX , CX LDW,                    ( Literal nach AX )
        NEXT,
   END-CODE
| CODE ("       ( -- addr ) ( Codeteil zu " )
        AX PUSHW   AX , IP LDW,           ( TOS=Stringadresse )
        BL , @AX LDC,   BH , # 0 LD,         ( Länge nach BX )
        SCF,  BL , AL ADC,  BH , AH ADC,    ( Adresse+Länge+1 )
        IP , BX LDW,                        ( Neue IP-Adresse )
        NEXT,
  END-CODE
```

```
\ BRANCH  ?BRANCH                                   ( 13.05.90/KK )

  CODE BRANCH   ( -- ) ( IP um Inline-Offset verändern )
        BX , IP LDW,   CX , @BX LDCW    ( Adresse nach CX )
        IP , CX LDW,   NEXT,            ( CX nach IP; NEXT )
  END-CODE

  CODE ?BRANCH  ( f -- ) ( Bedingter Sprung )
        AH , AL OR,   AX POPW           ( Flag testen )
        Z , ' BRANCH JR,                ( Sprung wenn Wert=0 )
        IP INCW,   IP INCW,   NEXT,     ( IP + 2 ausführen )
  END-CODE




( (DO )                                             ( 15.05.90/KK )

| CODE (DO      ( limit start -- ) ( DO-Runtimeroutine )
        IP++->CX CALL,                  ( Inline-Lit. -> CX )
        CL PUSH,   CH PUSH,             ( Endadresse )
        BX POPW   BX DECW,  BH , # $80 XOR,
        BL PUSH,  BH PUSH,              ( Limit XOR 8000 - 1)
        BL , AL SUB,   BH , AH SBC,
        BL PUSH,   BH PUSH,             ( limit-start XOR 8000)
        (DROP JP,                       ( neues TOS )
   END-CODE




\ (?DO  LEAVE                                       ( 13.05.90/KK )

| CODE (?DO     ( limit start --- ) ( ?DO-Runtimeroutine )
       BX , 0 (FSP LDCW                 ( Limit nach BX )
       BL , AL CP,   NZ , ' (DO JR,     ( Loop, wenn ungleich )
       BH , AH CP,   NZ , ' (DO JR,     ( Loop, wenn ungleich )
       AX POPW   AX POPW                ( Stack korrigieren )
       ' BRANCH JR,                     ( Zum Ende )
   END-CODE

  CODE LEAVE   ( -- ) ( Aussprung aus aktueller DO-Schleife )
        FRPL , # 4 ADD,   FRPH , # 0 ADC,   EXIT,
  END-CODE                                          RESTRICT



\ (LOOP  (+LOOP                                    ( 16.06.90/KK )

| CODE (LOOP    ( -- ) ( Schleifenende )
        AX PUSHW   AX , # 1 LDW,   ( ' (+LOOP JP, )
  END-CODE
( !!! Achtung: Geht nur, weil kein Header vorhanden ist )
| CODE (+LOOP   ( n -- )
        BH POP,   BL POP,               ( Zähler laden )
        BL , AL SUB,   BH , AH SBC,     ( - n )
        BL PUSH,   BH PUSH,             ( wieder speichern )
        AX POPW                         ( nächste TOS laden )
        NOV , ' BRANCH JP,              ( Zurück, wenn ok )
        IP INCW,   IP INCW,             ( IP korrigieren )
        FRPL , # 6 ADD,   FRPH , # 0 ADC, ( Stack korrigieren )
        NEXT,                           ( dann weiter )
  END-CODE
```

```
\ Systemconstanten                                ( 16.06.90/KK )
 { $FFB0 $20 -1 TFILL }          ( Speicher mit -1 vorbelegen )
  $A55A  $FFB0  !            ( Kennung für einen Startbereich )
  $FFB2 CONSTANT RAMORG      INITVAL RAMORG ! ( Tabellenstart )
  $FFB4 CONSTANT VOC-LINK              ( Vokabular-Verkettung )
  $FFB6 CONSTANT DP                      ( Dictionary-Pointer )
  $FFB8 CONSTANT VDP                          ( Offset-Zeiger )
  $FFBA CONSTANT VLEN                                ( Zähler )
  $FFBC CONSTANT HDP         $F9AE HDP      !   ( Heap-Anfang )
                             $F9AE >TASKORG !  ( Task-Bereich )
  $FFC0 >L: LASTTASK         $FA9C LASTTASK !      ( Taskaddr )
                             $C0   >TASK#   ! ( addr des Task )
  $FFC4 >L: DISKORG          $FBAE DISKORG  !    ( Diskbuffer )
  $F8E0 $FFC6 !   $AF $FFC8 C!  ( Für SPR@!: LD ___,___ ; RET )
  $4B2E $FFC9 !   $4B6F $FFCB !   $686C $FFCD !  $0D $FFCF C!
\ NEXT0 ( TRACE ) $FFCE  ! ( !T! $FFCE/F für Tracer verwendet )
\ NOOP  FALSE  TRUE  BL                           ( 19.05.90/KK )

\ NOOP für DEFER's
  CODE NOOP             NEXT,   END-CODE

\ Flags und häufig verwendete Zahlenwerte:
  CODE FALSE    ( -- 0 )        AX PUSHW               END-CODE
 ASSEMBLER   L: TOS=FALSE    AX , # 0 LDW,   NEXT,  FORTH

  CODE TRUE     ( -- -1 )       AX PUSHW               END-CODE
 ASSEMBLER   L: TOS=TRUE     AX , # -1 LDW,  NEXT,  FORTH

    $20 CONSTANT BL             ( Leerzeichen )



\ USER-Variablen                                  ( 16.06.90/KK )

 &16                    ( ersten 16 Bytes für Multitasker )

     USER S0            ( $10 : Anfang des Datenstacks )
     USER R0            ( $12 : Anfang des Returnstacks )

     USER STATE         ( $14 : Flag für Interpreter/Compiler )

\    USER ULEN          ( $16 : Anzahl der belegten User's )
 2 + USER BLK           ( $18 : aktueller Block )
     USER >IN           ( $1A : Offset )



\ USER-Variablen                                  ( 26.02.90/KK )

     USER SCR           ( $1C : Screennummer des Fehlers )
     USER R#            ( $1E : Offset bei Fehler )

     USER SPAN          ( $20 : Zeichenanzahl im Buffer )
     USER #TIB          ( $22 : Zeichenlänge )

     USER BASE          ( $24 : aktuelle Zahlenbasis )
     USER HLD           ( $26 : Zeiger für Zahlenumwandlung )
     USER DPL           ( $28 : 16/32-Bit - Nachkommastellen )
```

```
\ USER-Variablen                                  ( 16.06.90/KK )

     USER INPUT          ( $2A : Zeiger auf INPUT-Befehle )
     USER OUTPUT         ( $2C : Zeiger auf OUTPUT-Befehle )
     USER ERRORHANDLER   ( $2E : Zeiger auf Fehlerroutine )

\    USER SLEN           ( $30 : Länge des Datenstack-Bereichs )
\    USER RLEN           ( $32 : Länge des Returnstack-Bereichs )
 4 + | USER (PAD         ( $34 : Adresse des PAD's )

 { DROP }




\ Variablen                                       ( 16.06.90/KK )

  VARIABLE CURRENT       ( Current-Vokabular )
| VARIABLE (CONTEXT      ( 6 Context-Vokabulare )
   { 0 TVAR,  0 TVAR,  0 TVAR,  0 TVAR,  0 TVAR, }

  VARIABLE LAST          ( Letzte LFA )

  VARIABLE ?HEAD         ( Flag, ob headerlos )




\ Aufbau des TASK$C0-Bereiches                    ( 27.02.90/KK )

 { TDP @  $FA9C TDP ! }                 ( Bereich für Task$C0 )
 ASSEMBLER
  L: ORIGIN     ( Anfang des ersten Taskbereiches )
        ORIGIN JP,   # $C0 SRP,           ( Tasks-Verkettungen )
        FRP POPW  IP POPW  RET, ( FRP u. IP laden, dann Start )
 FORTH
        $FFFF ,                                        ( frei )
        $FA96 , $FBAE ,                               ( S0, R0 )
        $00EE ERRORHANDLER 2 + !                        ( SLEN )
        $0092 ERRORHANDLER 4 + !                        ( RLEN )
        $FA02 ERRORHANDLER 6 + !                        ( (PAD )
        $A4  ORIGIN  C!              ( Patch: CP r,r statt JP )
 {       TDP !            }           ( Programmanfang setzen )

\ C@  C!  @  !                                    ( 16.06.90/KK )

   CODE C@      ( addr -- b )
        AL , @AX LDC,   AH , # 0 LD,   NEXT,           END-CODE
   CODE C!      ( b addr -- )
        BX POPW   @AX , BL LDC,   (DROP JP,            END-CODE

   CODE @       ( addr -- n )
        BH , @AX LDCI,   AL , @AX LDC,   AH , BH LD,
        NEXT,                                          END-CODE
   CODE !       ( n addr -- )
        BX POPW   @AX , BX LDCW   (DROP JP,            END-CODE
```

```
\ SP@  SP!  DCLEAR  DUP  SWAP  DROP              ( 16.06.90/KK )

  CODE SP@      ( -- sp ) ( Aktuelle Stackadresse )
        AX PUSHW   AX , FSP LDW,   NEXT,               END-CODE
  CODE SP!      ( sp -- ) ( Datenstack auf sp setzen )
        FSP , AX LDW,   (DROP JP,                      END-CODE
  CODE DCLEAR   ( ... -- )
        FSP , $10 (FUP LDCW   (DROP JP,                END-CODE

  CODE DUP      ( n -- n n )          ( entspricht 0 PICK )
        AX PUSHW        NEXT,                          END-CODE
  CODE SWAP     ( n1 n2 -- n2 n1 )
        BX POPW   AX PUSHW   AX , BX LDW,   NEXT,      END-CODE
  CODE DROP     ( n -- )
        (DROP JP,                                      END-CODE

\ RP@  RP!  >R  R>                              ( 16.06.90/KK )

  CODE RP@      ( -- rp ) ( Aktuelle Returnstackadresse )
        AX PUSHW   AX , FRP LDCW   NEXT,     END-CODE
  CODE RP!      ( sp -- ) ( Returnstack auf sp setzen )
        FRP , AX LDW,   (DROP JP,            END-CODE  RESTRICT

  CODE >R       ( D: n --   ; R:   -- n )
        AL PUSH,   AH PUSH,   (DROP JP,      END-CODE  RESTRICT
  CODE R>       ( D:   -- n ; R: n --   )
        (DIC JP,                             END-CODE  RESTRICT




\ R@  RDROP  PUSH                               ( 16.06.90/KK )

  CODE R@       ( D:   -- n ; R: n -- n )
        AX PUSHW   AH POP,   AL POP,
                   AL PUSH,  AH PUSH,   NEXT,          END-CODE
  CODE RDROP    ( D: -- ; R: n -- )
        FRP INCW,   FRP INCW,   NEXT,        END-CODE  RESTRICT

| L: (POP       ( -- ; R: addr val -- )
   ] R> R> ! EXIT [
  : PUSH        ( addr -- ; R: -- addr val (pop )
   R> SWAP  DUP >R  @ >R  (POP >R  >R ;




\ ?DUP  OVER  TUCK  NIP  ROT  -ROT              ( 16.05.90/KK )

  CODE ?DUP     ( n -- 0 | n n )
        BL , AL LD,   BL , AH OR,   NZ , ' DUP JP,
    1$: NEXT,                                          END-CODE
  CODE OVER     ( n1 n2 -- n1 n2 n1 ) ( entspricht 1 PICK )
        AX PUSHW   AX , 2 (FSP LDCW   NEXT,            END-CODE
  : TUCK        ( n1 n2 -- n2 n1 n2 ) ( entspricht UNDER )
   SWAP OVER ;
  CODE NIP      ( n1 n2 -- n2 )
        FSP INCW,   FSP INCW,   NEXT,                  END-CODE
  : ROT         ( n1 n2 n3 -- n2 n3 n1 )
   >R SWAP R> SWAP ;
  : -ROT        ( n1 n2 n3 -- n3 n1 n2 )
   SWAP >R SWAP R> ;
```

```
\ PICK                                              ( 16.05.90/KK )

  CODE PICK     ( ... n0 n -- ... n0 nn ) ( n-ter Stackwert )
        AL , AL ADD,   AH , AH ADC,                       ( n*2 )
        AL , FSPL ADD,   AH , FSPH ADC,      ( + Stackpointer )
        BX , AX LDW,   AX , @BX LDCW   NEXT,     ( Wert holen )
   END-CODE
```

```
\ ROLL                                              ( 16.06.90/KK )

  CODE ROLL     ( no nn nm .. n0 n -- no nm ... n0 nn )
        AL , AL ADD,   AH , AH ADC,                       ( n*2 )
        BX , AX LDW,  BL , FSPL ADD,   BH , FSPH ADC,    ( SP )
        DX , @BX LDCW                              ( Wert holen )
        CL , AL LD,   CL , AH OR,  Z , 2$ JP,           ( 0 ? )
    1$: CL , -1 (BX LDC,   1 (BX , CL LDC,   BX DECW,
        AX DECW,   NZ , 1$ JP,         ( bis Stack verschoben )
    2$: AX POPW   AX , DX LDW,   NEXT,
   END-CODE
```

```
\ 2DUP  2SWAP  2DROP                                ( 19.05.90/KK )

  : 2DUP        ( d -- d d )
   OVER OVER ;

  : 2SWAP       ( d1 d2 -- d2 d1 )
   3 ROLL   3 ROLL ;

  CODE 2DROP    ( d -- )
        AX POPW   (DROP JP,                          END-CODE
```

```
\ I  J                                              ( 13.05.90/KK )

  CODE I        ( -- i ) ( Innersten Schleifenzähler )
        AX PUSHW   BX , FRP LDW,
        AX , 2 (BX  LDCW   CX , 0 (BX LDCW   ( Ende, Zähler )
        AL , CL SUB,   AH , CH SBC,   NEXT,           END-CODE

  CODE J        ( -- j ) ( nächst äußeren Schleifenzähler )
        AX PUSHW   BX , FRP LDW,
        AX , 8 (BX  LDCW   CX , 6 (BX LDCW   ( Ende, Zähler )
        AL , CL SUB,   AH , CH SBC,   NEXT,           END-CODE
```

```
\ AND  OR  XOR  NOT                                ( 23.02.90/KK )

  CODE AND      ( w1 w2 -- w ) ( Logische AND-Verknüpfung )
   BX POPW   AL , BL AND,   AH , BH AND,  NEXT,        END-CODE

  CODE OR       ( w1 w2 -- w ) ( Logische OR-Verknüpfung )
   BX POPW   AL , BL OR,   AH , BH OR,  NEXT,          END-CODE

  CODE XOR      ( w1 w2 -- w ) ( Logische XOR-Verknüpfung )
   BX POPW   AL , BL XOR,   AH , BH XOR,  NEXT,        END-CODE

  CODE NOT      ( w1 -- w2 ) ( Einer-Komplement )
   AL COM,   AH COM,   NEXT,                          END-CODE



\ 2*  2/  U2/  FLIP                                ( 16.06.90/KK )

  CODE 2*       ( n -- 2n )
   AL , AL ADD,   AH , AH ADC,  NEXT,                 END-CODE

  CODE 2/       ( n -- n/2 )
   AH SRA,   AL RRC,   NEXT,                          END-CODE

  CODE U2/      ( u -- u/2 )
   RCF,   AH RRC,   AL RRC,   NEXT,                   END-CODE

  CODE FLIP     ( $xxyy -- $yyxx ) ( High und Low vertauschen )
   BL , AH LD,   AH , AL LD,   AL , BL LD,   NEXT,    END-CODE



\ +  -  NEGATE                                     ( 10.02.90/KK )

  CODE +        ( n1 n2 -- n ) ( Summe )
   BX POPW   AL , BL ADD,   AH , BH ADC,   NEXT,      END-CODE

  CODE -        ( n1 n2 -- n ) ( Differenz n1-n2 )
   BX , AX LDW,   AX POPW
   AL , BL SUB,   AH , BH SBC,   NEXT,                END-CODE

  CODE NEGATE   ( n -- -n ) ( Zweierkomplement )
   AL COM,   AH COM,   AX INCW,   NEXT,               END-CODE




\ 1+  2+  1-  2-                                   ( 13.05.90/KK )

  CODE 2-       ( n -- n-2 )
   AX DECW,   AX DECW,   NEXT,                        END-CODE

  CODE 1-       ( n -- n-1 )
   AX DECW,   NEXT,                                   END-CODE

  CODE 1+       ( n -- n+1 )
   AX INCW,   NEXT,                                   END-CODE

  CODE 2+       ( n -- n+2 )
   AX INCW,   AX INCW,   NEXT,                        END-CODE
```

```
\ D+  D-  DNEGATE                                   ( 19.02.90/KK )

  CODE D+       ( d1 d2 -- d3 )
        BX POPW   CX POPW   DX POPW
        BL , DL ADD,   BH , DH ADC,   BX PUSHW
        AL , CL ADC,   AH , CH ADC,   NEXT,            END-CODE

  CODE D-       ( d1 d2 -- d3 )
        CX , AX LDW,   DX POPW   AX POPW   BX POPW
        BL , DL SUB,   BH , DH SBC,   BX PUSHW
        AL , CL SBC,   AH , CH SBC,   NEXT,            END-CODE

  : DNEGATE     ( d -- -d )
   FALSE DUP ( 0. )  2SWAP  D- ;


\ 0<  0=  0>                                        ( 23.02.90/KK )

  CODE 0<       ( n -- f ) ( Test, ob Wert negativ )
        AH , AH OR,  PL , TOS=FALSE JP,  TOS=TRUE JP,  END-CODE

  CODE 0=       ( n -- f )
        AH , AL OR,  Z , TOS=TRUE JP,   TOS=FALSE JP,  END-CODE

  CODE 0>       ( n -- f )
        AL , AH OR,   NZ , 1$ JP,   NEXT,
    1$: AH , AH OR,   MI , TOS=FALSE JP,   TOS=TRUE JP, END-CODE




\ <  =  >  U<                                       ( 21.11.90/KK )

  CODE <        ( n1 n2 -- f )
        BX POPW   BL , AL SUB,   BH , AH SBC,   ( n1-n2 )
        LT , TOS=TRUE JP,   TOS=FALSE JP,              END-CODE
  : =           ( n1 n2 -- f )
   XOR IF FALSE EXIT THEN TRUE ;
  : >           ( n1 n2 -- f )
   SWAP < ;

  CODE U<       ( u1 u2 -- f )
        BX POPW   BL , AL SUB,   BH , AH SBC,   ( n1-n2 )
        C , TOS=TRUE JP,   TOS=FALSE JP,               END-CODE



\ ABS  MIN  MAX  UWITHIN  CASE?                     ( 16.06.90/KK )

  : ABS         ( n -- u )
   DUP 0< IF NEGATE THEN ;

| : ?SWAPDROP   IF SWAP THEN DROP ;
  : MIN         ( n1 n2 -- n1 | n2 )
   2DUP >  ?SWAPDROP ;
  : MAX         ( n1 n2 -- n1 | n2 )
   2DUP <  ?SWAPDROP ;

  : UWITHIN     ( u u1 u2 -- f )
   >R OVER SWAP U< 0= SWAP R> U< AND ;

  : CASE?       ( n1 n2 -- n1 0 | -1 wenn gleich )
   OVER - ?DUP NIP 0= ;
```

```
\ D<  DABS                                          ( 16.06.90/KK )

  CODE  D<      ( d1  d2 --> f )
        BX POPW   CX POPW   DX POPW
        BL , DL SUB,   BH , DH SBC,
        AL , CL SBC,   AH , CH SBC,
        AX , # -1 LDW,   LT , HERE 4 + JR,   ( !!! nach NEXT, )
        AX INCW,
        NEXT,
  END-CODE

  : DABS        ( d -- du )
   DUP 0< IF DNEGATE THEN ;



\ UM*                                               ( 20.01.90/KK )

  CODE UM*      ( u1 u2 -- ud )
        BX POPW
        CH , AL LD,   CX , BH MULT,                  ( U2L*U1H )
        DH , AH LD,   DX , BL MULT,                  ( U2H*U1L )
        AL PUSH,      AH , BH MULT,                  ( U2H*U1H )
        BH POP,       BH , BL MULT,                  ( U2L*U1L )
        BH , CL ADD,   AL , CH ADC,   AH , # 0 ADC, ( + ZW1 )
        BH , DL ADD,   AL , DH ADC,   AH , # 0 ADC, ( + ZW2 )
        BX PUSHW   NEXT,
   END-CODE



\ UM/MOD                                            ( 20.01.90/KK )

  CODE UM/MOD   ( ud u -- r q )
        BX POPW   CX POPW                          ( ud laden )
        DH , # $10 LD,                               ( 16 mal )
   L: UM/MOD1
        CL , CL ADD,  CH RLC,  BL RLC,  BH RLC,   NC , 2$ JP,
        BL , AL SUB,   BH , BH SBC,   3$ JP,
    2$: BL , AL SUB,   BH , AH SBC,   NC , 3$ JP,
        BL , AL ADD,   BH , AH ADC,   CL DEC,
    3$: CL INC,   DH , UM/MOD1 DJNZ,             ( wiederholen )
        AX , CX LDW,   BX PUSHW   NEXT,
   END-CODE



\ M*  *                                             ( 13.05.90/KK )

  : M*          ( n1 n2 -- d )
    >R  FALSE  TUCK  OVER 0<  IF R@ + THEN
    R@                    0<  IF OVER + THEN
    SWAP R>  UM*  D+ ;

  : *            ( n1 n2 -- n )
   UM* DROP ;
```

```
\ M/MOD  M/  /MOD  /  MOD                          ( 10.02.90/KK )

  : M/MOD       ( d n -- mod quot )
   DUP >R ABS OVER 0< IF TUCK + SWAP THEN  UM/MOD
       R@ 0< IF NEGATE OVER IF SWAP R@ + SWAP 1- THEN THEN
       RDROP ;
  : M/          ( d n -- q )
   M/MOD NIP ;
  : /MOD        ( n1 n2 -- r q )
   OVER 0< SWAP M/MOD ;

  : /           ( n1 n2 -- q )
   /MOD NIP ;
  : MOD         ( n1 n2 -- r )
   /MOD DROP ;

\ */MOD  */                                       ( 28.12.89/KK )

  : */MOD       ( n1 n2 n3 -- r q )
   >R M* R> M/MOD ;

  : */          ( n1 n2 n3 -- q )
   */MOD NIP ;









\ +!  ON  OFF  GR@  GR!                           ( 16.06.90/KK )

  : +!          ( n addr -- )
        TUCK  @  +  SWAP  ! ;

  : ON          ( addr -- )
        TRUE  SWAP  ! ;
  : OFF         ( addr -- )
        FALSE  SWAP  ! ;

  CODE GR@      ( addr -- b )
        AL , @AL LD,   AH , # 0 LD,   NEXT,            END-CODE

  CODE GR!      ( b addr -- )
        BX POPW   @AL , BL LD,   (DROP JP,             END-CODE

\ SPR@  SPR!                                       ( 16.06.90/KK )

  ASSEMBLER
   L: (SPR@!    ( BX=Adressen; CL=Flag ; AX vom Stack )
        AH , AH OR,   Z , HERE 3 + JR,   SB1,
        CX , # $FFC7 LDW,   @CX , AL LDC,   @CX , BH LDCPD,
        @CX JP,
  FORTH

  CODE SPR!     ( b addr -- )
        BX POPW   BH , # $D9 LD,   (SPR@! CALL,
        SB0,   (DROP JP,                           END-CODE

  CODE SPR@     ( addr -- f )
        BH , # $F8 LD,   (SPR@! CALL,
        SB0,   AH CLR,   NEXT,                     END-CODE
```

```
\ COUNT  -TRAILING                                  ( 03.05.90/KK )

  CODE COUNT    ( addr -- addr+1 len )
        BX , AX LDW,   AL , @BX LDCI,   AH , # 0 LD,
        BX PUSHW   NEXT,                              END-CODE

  CODE -TRAILING   ( addr len1 -- addr len2 )
        BX , 0 (FSP LDCW   BX DECW,       ( Adresse-1 nach BX )
        BL , AL ADD,   BH , AH ADC,   AX INCW,     ( Zum Ende )
    L: -TRAILING1
        AX DECW,   Z , 2$ JP,                   ( Bis Länge 0 )
        CL , @BX LDCD,   CL , # $20 CP,   Z , -TRAILING1 JR,
    2$: NEXT,                                         END-CODE



\ CMOVE  CMOVE>                                     ( 02.05.90/KK )

  CODE CMOVE    ( addr1 addr2 len -- ) ( addr1 zuerst )
        BX POPW   CX POPW    BX DECW,           ( Reg's laden )
        DH , AL LD,   DH , AH OR,   Z , 2$ JP,    ( Länge=0 ? )
    L: CMOVE1   DL , @CX LDCI,   @BX , DL LDCPI,
                AX DECW,   NZ , CMOVE1 JR,
    2$: (DROP JP,                                     END-CODE
  CODE CMOVE>   ( addr1 addr2 len -- ) ( addr1+len-1 zuerst )
        BX POPW   CX POPW    CX DECW,           ( Reg's laden )
        DH , AL LD,   DH , AH OR,   Z , 2$ JP,    ( Länge=0 ? )
        BL , AL ADD,   BH , AH ADC,
        CL , AL ADD,   CH , AH ADC,
    L: CMOVE>1  DL , @CX LDCD,   @BX , DL LDCPD,
                AX DECW,   NZ , CMOVE>1 JR,
    2$: (DROP JP,                                     END-CODE
\ FILL  ERASE  BLANK                                ( 16.06.90/KK )

  CODE FILL     ( addr len char -- )
        BX POPW   CX POPW   CX DECW,
        AH , BL LD,   AH , BH OR,   Z , (DROP JP, ( wenn len=0)
    L: FILL1
        @CX , AL LDCPI,   BX DECW,   NZ , FILL1 JR,
        (DROP JP,                                     END-CODE

  : ERASE       ( addr len -- )         FALSE FILL ;
  : BLANK       ( addr len -- )         BL    FILL ;



\  FIRST  TIB  ...  DEPTH                           ( 16.06.90/KK )

  : FIRST       ( -- addr )   DISKORG @ ;        ( Diskpuffer )
  $80 USER TIB  ( -- addr )   { DROP }        ( Eingabepuffer )
  : PAD         ( -- addr )   (PAD @ ;    ( PAD als USER-Var. )
  : HEAP        ( -- addr )   HDP @ ;          ( Heap-Adresse )
  : HERE        ( -- addr )   DP @ ;            ( Dictionary )
  : WDP@        ( -- addr )   PAD &84 - ;       ( WORD-Basis )

  : DEPTH       ( -- n ) ( Anzahl der Datenstackwerte )
   SP@  S0 @  SWAP -  2/ ;

  : ->VAR ( addr -- vaddr )   @  VDP @  + ;        ( Varaddr. )
| : CURRENT@    CURRENT @  2+  ->VAR ;
\ : HEAP?       ( cfa -- f )  HEAP  >TASKORG @  UWITHIN ;
```

```
\ ALLOT  C,  ,   HALLOT  H,  VALLOT  V,         ( 16.05.90/KK )

  : ALLOT       ( n -- )        DP +! ;
  : C,          ( c -- )        HERE  1 ALLOT  C! ;
  : ,           ( n -- )        HERE  2 ALLOT   ! ;

| : VHALLOT     ( n -- n ) ( Spalt zwischen HEAP und Variable )
   DUP >R  VDP @  DUP R@ -  DUP VDP !  VLEN @
       R>  0<  IF  CMOVE> EXIT  THEN  CMOVE ;
  : HALLOT      ( n -- ) ( erfordert Variablen-Verschiebung )
    VHALLOT  NEGATE HDP +! ;
  : H,          ( n -- ) 2 HALLOT  HEAP ! ;
  : VALLOT      ( n -- ) ( erfordert Variablen-Verschiebung )
    VHALLOT  VLEN +! ;
  : V,          ( n -- ) 2 VALLOT  HEAP 2- !  ;

\ Fehlermeldungen                               ( 16.06.90/KK )

  : ERROR       ( error | csa -1 | 0 -- )
   ?DUP IF ERRORHANDLER @ EXECUTE THEN ;
  : ?ERROR      ( flag error -- )
   SWAP IF ERROR EXIT THEN DROP ;

  : ?STACK      ( -- ) ( Test ob Stack in Ordnung )
       DEPTH  0<  IF DCLEAR $0002 ERROR THEN
   $20 DEPTH  U<  IF DCLEAR $0008 ERROR THEN ;
  : ?PAIRS      ( n1 n2 -- )  -  $000A ?ERROR ;
  : NODEFER     ( -- )          $000C  ERROR ;
  DEFER NOTFOUND    HERE IS NOTFOUND
| : (NOTFOUND   ( csa -- )    DROP  "  ??? "  TRUE  ERROR ;
  CODE INTERROR ( ??? )       DI,  ENTER,             END-CODE
   ] RDROP R> $0015 ERROR [
\ Forwärts-Definitionen und I/O                 ( 19.05.90/KK )

  FORWARD INTERPRET             ( Interpreter/Compiler )
  FORWARD QUIT                  ( Eingabe-Hauptschleife )

 ASSEMBLER
  L:   (INPUT:          ( Befehl für Umsetzung von INPUT )
        BH POP,   BL POP,   $2A (FUP , BX LDCW   NEXT,
  L:   (OUTPUT:         ( Befehl für Umsetzung von OUTPUT )
        BH POP,   BL POP,   $2C (FUP , BX LDCW   NEXT,
 FORTH

| : (INVECTOR:    R> C@   INPUT  @  +   @   EXECUTE ;
| : (OUTVECTOR:   R> C@   OUTPUT @  +   @   EXECUTE ;


\ Zeichen ausgeben                              ( 16.06.90/KK )

  0 OUTVECTOR: EMIT? ( -- f )    ( Statusabfrage vom Ausgang )
| CODE SIO_EMIT?        ( -- f )
        BL , UTC LD,                                END-CODE
  ASSEMBLER
   L: SIO_EMIT?1   BL , # 1 AND,   Z , ' FALSE JP,   ' TRUE JP,
  FORTH

  2 OUTVECTOR: EMIT  ( char -- ) ( 1 Zeichen ausgeben )
| CODE SIO_EMIT         ( char -- ) ( nur an Konsole ausgeben )
   L: SIO_EMIT1 AH , UTC LD,   SIO_EMIT1 , AH # 1 BTJRF,
        UIO , AL LD,   (DROP JP,                    END-CODE

  OUTPUT: SIO_OUTPUT   SIO_EMIT?  SIO_EMIT ;
```

```
\ Zeichen holen                                    ( 16.06.90/KK )

  0 INVECTOR: KEY?      ( -- f ) ( Statusabfrage vom Eingang )
| CODE SIO_KEY?         ( -- f ) ( Ausgabestatus holen )
       BL , URC LD,   SIO_EMIT?1 JR,                  END-CODE

  2 INVECTOR: KEY       ( -- char ) ( 1 Zeichen lesen )
| CODE SIO_KEY          ( -- char ) ( Zeichen von Konsole )
       AX PUSHW
   L: SIO_KEY1  AL , URC LD,   SIO_KEY1 , AL # 0 BTJRF,
       AL , UIO LD,  AH , # 0 LD,   NEXT,             END-CODE

  INPUT:  SIO_INPUT    SIO_KEY?   SIO_KEY  ;



\ TYPE  SPACE  SPACES  CR  | DEL                ( 16.06.90/KK )

  : TYPE        ( addr count -- ) ( Text ausgeben )
   FALSE ?DO  COUNT EMIT  LOOP  DROP  ;
  : SPACE       ( -- ) ( ein Leerzeichen ausgeben )
   BL  EMIT ;
  : SPACES      ( +n -- ) ( n Leerzeichen ausgeben )
   FALSE MAX  FALSE  ?DO  SPACE  LOOP  ;

( !!! Achtung: Entsprechende Codes direkt eingefügt )
| : (."         ( -- ) ( Inline-String ausgeben )
   R> COUNT 2DUP + >R TYPE ;                        RESTRICT
  : CR          ( -- ) ( Zum Anfang der nächsten Zeile )
   ."
"  ( $0D/$0A ) ;
| : DEL         ( -- ) ( Backspace ausgeben )
   ." □ □" ( $08/SPACE/$08 ) ;
\ STOP?  | (."  | NAME.                          ( 16.06.90/KK )

  : STOP?       ( -- f )          ( Warten bei <> ^X )
   KEY? DUP                       ( Taste ? )
   IF  KEY &24 = IF EXIT THEN     ( wenn ^X: f = -1 )
       DROP  KEY &24 =            ( Warten, Taste = ^X? )
   THEN ;                         ( ansonsten f = 0 )

| : DOTQ        ( addr len -- )
   COUNT TYPE ;

| : NAME.       ( csa -- ) ( nur maximal 31 Zeichen )
   ?DUP  IF COUNT  $1F AND  TYPE  EXIT  THEN  ." ???" ;



\ | DECODE  EXPECT                               ( 16.06.90/KK )

| : DECODE      ( char -- )
     8 CASE?  IF  DUP  IF DEL 1- THEN                EXIT THEN
   $0D CASE?  IF  DUP  SPAN !                        EXIT THEN
   $18 CASE?  IF    FALSE ?DO DEL LOOP  FALSE        EXIT THEN
   DUP $1F >  IF  >R  2DUP +  R@ SWAP C!  R> EMIT 1+  EXIT THEN
   DROP ;

  : EXPECT      ( addr count -- )
   SPAN !   FALSE
   BEGIN  DUP SPAN @  U<  WHILE  KEY DECODE  REPEAT
   2DROP SPACE ;
```

```
\ DECIMAL  HEX  <#  #  #S  HOLD  SIGN  #>       ( 16.06.90/KK )

  : DECIMAL     $0A BASE ! ;    ( Zahlenbasis 10 )
  : HEX         $10 BASE ! ;    ( Zahlenbasis 16 )

  : <#    PAD HLD ! ;
  : HOLD  HLD @  1-  DUP HLD !  C! ;
| : DU/BASE  BASE @ UM/MOD -ROT ;
  : #     FALSE  DU/BASE  DU/BASE
          $30 +  $39 OVER U< IF 7 + THEN HOLD ;
  : #S    BEGIN  #  2DUP  OR 0=  UNTIL ;
  : SIGN  0< IF $2D HOLD THEN ;
  : #>    2DROP  HLD @  PAD OVER - ;



\ D.R  D.  U.R  U.  .R  .                       ( 16.05.90/KK )

  : D.R   >R DUP >R DABS <# #S R> SIGN #>
          R> OVER - SPACES TYPE ;
  : D.    FALSE D.R SPACE ;

  : U.R   FALSE SWAP D.R ;
  : U.    FALSE D. ;

  : .R    >R DUP 0< R> D.R ;
  : .     DUP 0< D. ;




\ CONVERT                                       ( 16.05.90/KK )

  : CONVERT     ( d1 ^string1 --- d2 ^string2 )
   BEGIN   1+  DUP >R  C@  $30 -
           DUP $0A U< ?DUP 0=
           IF 7 - DUP $0A BASE @ UWITHIN ?DUP NIP THEN
   WHILE   SWAP BASE @ UM* DROP ROT BASE @ UM* D+  R>
   REPEAT  R> ;







\ NUMBER? ( 32-Bit-Version )                    ( 16.05.90/KK )

  DEFER NUMBER? ( csa -- addr ff | d 0> | n 0< )
| : (NUMBER?    ( csa -- addr ff | d 0> | n 0< )
   FALSE FALSE  ROT
   DUP 1+ C@  ASCII -  =   DUP >R   -          ( Vorzeichen ? )
   DUP >R   CONVERT                                ( 1. Teil )
   DUP C@  ASCII .  =                                ( "." ? )
   IF   DUP >R   CONVERT   DUP  1-  DUP R> -
   ELSE DUP  TRUE
   THEN DPL !  1- R> =  OVER C@ BL -  OR   ( nur ., ohne BL ? )
   IF  NIP NIP RDROP FALSE  EXIT  THEN  DROP
   R>  IF  DNEGATE  THEN
   DPL @  DUP 0<   IF  NIP  ELSE  1+  THEN ;
  ' (NUMBER? IS NUMBER?
```

```
\ COMMAND!  R/W                                    ( 16.05.90/KK )

  : COMMAND!    ( n command -- f )
   KEY?  IF 2DROP $0007 EXIT THEN
   &27 EMIT  EMIT   DUP FLIP EMIT  EMIT  KEY ;

  DEFER R/W     ( addr blk r/w -- f )
| : (R/W        ( addr blk r/w -- f )
   IF   1 COMMAND!  ?DUP 0=
      IF  &1024 FALSE  DO KEY OVER C! 1+ LOOP  FALSE  THEN
   ELSE 2 COMMAND!  ?DUP 0=
      IF  DUP &1024 TYPE                       FALSE  THEN
   THEN NIP ;
  ' (R/W  IS  R/W


\ SAVE-BUFFERS  EMPTY-BUFFERS  FLUSH  UPDATE    ( 16.05.90/KK )

  : SAVE-BUFFERS  ( -- )        ( speichern nach UPDATE )
   FIRST @  DUP $7FFF AND  SWAP 0<
   IF   FIRST 2+ OVER FALSE R/W ?DUP IF NIP ERROR THEN
   THEN FIRST ! ;

  : EMPTY-BUFFERS ( -- )        ( löschen )
   $7FFF FIRST !   FIRST 2+ &1024 BLANK ;

  : FLUSH         ( -- )        ( speichern und löschen )
   SAVE-BUFFERS  EMPTY-BUFFERS ;

  : UPDATE        ( -- )        ( Aktueller Block updaten )
   FIRST @  DUP $7FFF U<  IF $8000 FIRST +! THEN ;

\ BLOCK  BUFFER                                   ( 16.05.90/KK )

  : BUFFER      ( u -- addr )   ( Blockadresse liefern )
   $7FFE OVER U<  $0009 ?ERROR  ( # > $7FFE ? )
   SAVE-BUFFERS  FIRST !        ( alte sichern )
   FIRST 2+ ;                   ( Nummer setzen, Adresse )

  : BLOCK       ( u -- addr )   ( Block liefern )
   DUP >R  FIRST @ $7FFF AND  - ( gleich ? )
   IF   R@ BUFFER  R@ TRUE R/W ?DUP
      IF NIP  $7FFF FIRST !  ERROR THEN
   THEN RDROP FIRST 2+ ;




\ LOAD  THRU  -->                                 ( 15.05.90/KK )

  : LOAD        ( u -- )
   BLK PUSH  >IN PUSH
   BLK !     >IN OFF    INTERPRET ;

  : THRU        ( n1 n2 -- )
   1+ SWAP  DO  I  [COMPILE] LOAD  LOOP ;

  : -->         ( -- )
   1 BLK +!   >IN OFF ;                                IMMEDIATE
```

```
\ N>LINK  L>NAME  NAME>  BODY>  LINK>           ( 19.05.90/KK )

  : N>LINK      ( nfa -- lfa )  2- ;
  : L>NAME      ( lfa -- nfa )  2+ ;

  : NAME>       ( nfa -- cfa )
    COUNT  TUCK  $1F AND  +  SWAP  $20 AND  IF @ THEN ;
  : BODY>       ( pfa -- cfa )  3 - ;
  : LINK>       ( lfa -- cfa )  2+  NAME> ;




\ >NAME  >BODY  >LINK                          ( 21.11.90/KK )

  : >NAME       ( cfa -- nfa | 0 )
   >R  VOC-LINK
   BEGIN  @  ?DUP
   WHILE  DUP 2+  ->VAR
      BEGIN  @  ?DUP
      WHILE  DUP LINK>  R@  =
         IF  NIP  RDROP  L>NAME  EXIT  THEN
      REPEAT
   REPEAT RDROP FALSE ;
  : >BODY       ( cfa -- pfa )  3 + ;
  : >LINK       ( cfa -- lfa )  >NAME DUP IF 2- THEN ;



\ Vokabularverwaltung                          ( 15.05.90/KK )

  : CONTEXT     ( -- addr )
   (CONTEXT   DUP @   + ;
  : DEFINITIONS ( -- addr )
   CONTEXT @  CURRENT ! ;
  : ALSO        ( -- ) ( CONTEXT erweitern )
   (CONTEXT @  $0A  >   $0016 ?ERROR
   CONTEXT @  2 (CONTEXT +!  CONTEXT ! ;

 ASSEMBLER  L: (VOC   (DOES> CALL,              FORTH
        ] CONTEXT !  EXIT [
  VOCABULARY FORTH

  : ONLYFORTH   ( -- ) ( FORTH als CURRENT und 2*CONTEXT )
   2 (CONTEXT !   [COMPILE] FORTH   ALSO   DEFINITIONS ;
\ ORDER                                        ( 19.05.90/KK )

| : (ORDER      ( (pfa -- )
   BODY>  >NAME   NAME.  SPACE ;

  : ORDER       ( -- ) ( Alle Vokabulare ausgeben )
   (CONTEXT 2+  CONTEXT  ?DO  I @ (ORDER  -2 +LOOP
   SPACE SPACE
   CURRENT                    @ (ORDER  ;

  : VOCS        ( -- ) ( Alle Vokabulare anzeigen )
   VOC-LINK  BEGIN  @ ?DUP  WHILE  DUP (ORDER  REPEAT ;
```

```
\ WORDS                                              ( 07.06.90/KK )

  : WORDS       ( Ausgabe der Befehlsliste )
   CR  FALSE >R  CONTEXT @ 2+ ->VAR    ( Zähler, Adresse )
   BEGIN @  DUP                 ( Linkfeldadresse )
   WHILE  HERE OVER U< IF ." | " R> 2+ >R THEN
         DUP L>NAME  DUP NAME. SPACE SPACE
                     C@ $1F AND ( Länge des Namens )
         R> +  2+  &64 OVER U<
         IF   DROP FALSE CR
\         ELSE &20  OVER &20 MOD - DUP SPACES +
         THEN
         >R
         STOP?
   UNTIL DROP RDROP ;

\ DUMP                                               ( 21.11.90/KK )

  : DUMP        ( addr bank count -- )
   BASE PUSH  HEX
   1-  $10 /  1+   FALSE
   ?DO  CR  DUP 4 U.R SPACE
         ( DUP) $10 FALSE DO SPACE DUP C@ 2 U.R 1+ LOOP ( DROP)
\        3 SPACES
\            $10 FALSE DO DUP C@  DUP BL U<
\                     IF DROP ASCII . THEN  EMIT 1+ LOOP
        STOP? IF LEAVE THEN
   LOOP DROP ;



\ .S                                                 ( 21.11.90/KK )

  : .S          ( Alle Stackwerte ausgeben )
   DEPTH  FALSE MAX  $10 MIN  ?DUP
   IF FALSE
       DO  I PICK U.  LOOP   EXIT
\      DO  CR   I PICK
\           BASE @
\           OVER                    $11    .R
\           OVER  HEX     ."  ( $"    4  U.R
\           SWAP  DECIMAL ."  , &"    5  U.R ."  ) "
\           BASE !
\           STOP?  IF LEAVE THEN
\      LOOP
   THEN  ." <empty> " ;

\ Neue Routine für \WORD                             ( 15.05.90/KK )
| CODE (\WORD   ( char f addr1 glen off1 -- addr2 len off2 )
   CX POPW   CL , AL SUB,   CH , AH SBC,          ( glen-off1 )
   BX POPW   BL , AL ADD,   BH , AH ADC,         ( addr1+off1 )
   DX POPW   TEMPX POPW                  ( Flag und Zeichen )
   DH , DL OR,   Z , 2$ JP,              ( SKIP nur bei f<>0 )
   CX INCW,   AX DECW,              ( Vorbereitung für SKIP )
 L: W1$ AX INCW,   CX DECW,   Z , 2$ JP,       ( bis Zähler=0 )
        TEMPH , @BX LDCI,   TEMPH , TEMPL CP,   Z , W1$ JR,
        BX DECW,
   2$:  BX PUSHW   DX , # -1 LDW,   CX INCW,
 L: W3$ DX INCW,   CX DECW,   Z , 4$ JP,       ( bis Zähler=0 )
        TEMPH , @BX LDCI,   TEMPH , TEMPL CP,   NZ , W3$ JR,
        AX INCW,               ( auf Zeichen nach char zeigen )
   4$:  AL , DL ADD,   AH , DH ADC,
   DX PUSHW   NEXT,                                  END-CODE
```

```
\ | \WORD  WORD  |-WORD  |?NAME                ( 15.05.90/KK )

| : \WORD        ( char flag -- addr ) ( flag=0: kein SKIP )
   BLK @  ?DUP  IF  BLOCK &1024  ELSE  TIB #TIB @  THEN
   >IN @  (\WORD  >IN !
   WDP@   2DUP C!  1+  TUCK OVER +  >R  CMOVE
                               BL  R>  C!    WDP@  ;

  : WORD   TRUE  \WORD ;  ( mit Überlesen gleicher Zeichen )
| : -WORD  FALSE \WORD ;  ( ohne Überlesen )

| : ?NAME  BL WORD  C@ 0=  $0006 ?ERROR  WDP@  ;




( ?CAP )                                      ( 15.05.90/KK )

 DEFER ?CAP     ( addr -- addr ) ( evtl. Großschrift )

| CODE (?CAP    ( addr -- addr ) ( Großschrift )
        BX , AX LDW,   CL , @BX LDCI,   CL INC,   2$ JP,
  L: (CAP1   CH , @BX LDCI,
        CH , # $61 CP,   ULT , 2$ JP,
        CH , # $7A CP,   UGT , 2$ JP,
        CH , # $20 SUB,   @BX , CH LDCPD,   BX INCW,
  2$:   CL , (CAP1 DJNZ,   NEXT,                    END-CODE

 ' (?CAP  IS  ?CAP      ( Befehle nur in Großschrift )



\ | ((FIND                                    ( 15.05.90/KK )

 ASSEMBLER
  L: ((FIND ( BX=csa CX=vocaddr -- BX=csa ZF=1 | BX=lfa AL=-1 )
        CX INCW,   CX INCW,   AX , @CX LDCW    ( Offset holen )
        CX , VDP LDCW   CL , AL ADD,   CH , AH ADC,
   1$:  DH , @CX LDCI,   CL , @CX LDC,   CH , DH LD,
        DH , CL OR,   NZ , 2$ JP,   RET,   ( Nicht gefunden ? )
   2$:  BX PUSHW   CX PUSHW
        CX INCW,   CX INCW,   AL , @CX LDCI,   AL , # $1F AND,
        AH , @BX LDCI,   AH , AL CP,   NZ , 4$ JP,  ( Länge ? )
   L: ((FIND3   DH , @BX LDCI,   DL , @CX LDCI,      ( Rest ? )
               DH , DL CP,   NZ , 4$ JP,   AL , ((FIND3 DJNZ,
        BX POPW   CX POPW   AL CLR,  AL DEC,  RET, ( gefunden )
   4$:  CX POPW   BX POPW   1$ JP,      ( weiter zur nächsten )
 FORTH
\ | (FIND                                     ( 15.05.90/KK )

| CODE (FIND    ( -1 vocaddr ... csa -- csa 0 | lfa -1 )
        BX , AX LDW,                          ( csa nach BX )
    1$: CX POPW                               ( CX = offset )
        CH , # -1 CP,  NZ , 2$ JP,
        CL , # -1 CP,  NZ , 2$ JP,
        BX PUSHW   TOS=FALSE JP,            ( nicht gefunden )
    2$: DX POPW   DL , BL CP,   NZ , 3$ JP,      ( Duplikat ? )
               DH , BH CP,   Z , 2$ JP,
    3$: DX PUSHW  ((FIND CALL,   Z , 1$ JP,       ( Pfadsuche )
    4$: DX POPW   DX INCW,   NZ , 4$ JP,  ( Stack korrigieren )
        BX PUSHW   AH , AL LD,   NEXT,             ( gefunden )
   END-CODE
```

```
\ FIND                                              ( 15.05.90/KK )
\ addr1 --- addr1 false   name not found
\ addr1 --- addr2 <0      found, IMMEDIATE
\ addr1 --- addr2 !2!     found, RESTRICT
  : FIND        ( csa -- cfa f | csa 0 )
   ?CAP                          ( Umwandlung in Großschrift )
   >R TRUE                                    ( Ende der Liste )
      CURRENT @                                      ( Current )
      CONTEXT 2+  (CONTEXT 2+  DO  I @  2 +LOOP    ( Contexts )
   R> (FIND DUP
   IF  DROP L>NAME  DUP NAME>  SWAP C@
       DUP  $40 AND IF 2 ELSE 1 THEN             ( RESTRICT? )
       SWAP $80 AND IF NEGATE THEN               ( IMMEDIATE? )
   THEN ;


\ [  ]  COMPILE  [COMPILE]                          ( 16.05.90/KK )

  : [     STATE OFF ;                                    IMMEDIATE

  : ]     STATE ON ;

  : COMPILE       ( -- )
   R>  DUP 2+  >R  @  , ;                                 RESTRICT

  : [COMPILE]   ( -- )
   ?NAME FIND IF , EXIT THEN NOTFOUND ;      IMMEDIATE RESTRICT




\ LITERAL  $,  ASCII                                ( 15.05.90/KK )

  : LITERAL     ( value -- ) ( Literal kompilieren )
   COMPILE (LIT  ,  ;                          IMMEDIATE RESTRICT

  : $,          ( char -- ) ( String compilieren )
   -WORD  HERE  OVER C@ 1+  DUP ALLOT  CMOVE ;

  : ASCII       ( -- char |  )
   BL  WORD  1+ C@              ( Zeichen holen )
   STATE @  IF [COMPILE] LITERAL THEN ;               IMMEDIATE




\ ' [']  "  ."  .(                                   ( 19.05.90/KK )

  : '           ( -- csa ) ( cfa ermitteln )
   ?NAME FIND 0= IF (NOTFOUND THEN ;
  : [']         ( -- csa | ) ( cfa compilieren )
   [COMPILE] ' [COMPILE] LITERAL ;           IMMEDIATE RESTRICT

  : "           ( -- csa ) ( Interpretermodus: String zum PAD )
   STATE @ IF   COMPILE ("   ASCII "  $,  EXIT  THEN
           ( ELSE ) PAD ASCII "  -WORD  2DUP C@ 1+ CMOVE>
           ( THEN ) ;                                     IMMEDIATE

  : ."          ( -- ) ( String compilieren )
   COMPILE (." ASCII " $, ;                  IMMEDIATE RESTRICT
  : .(          ( -- ) ( String sofort ausgeben )
   ASCII )  -WORD  DOTQ ;                             IMMEDIATE
```

```
\ (  \  \\                                          ( 15.05.90/KK )

  : (           ( -- )
   ASCII )  -WORD DROP ;                                IMMEDIATE

  : \           ( -- )
   BLK @ IF   >IN @  &64  2DUP MOD  - +
         ELSE #TIB @                    THEN >IN ! ;  IMMEDIATE

  : \\          ( -- )
   BLK @ IF &1024 ELSE #TIB @ THEN >IN ! ;              IMMEDIATE




\ (ERROR                                            ( 21.11.90/KK )

  : (ERROR      ( $xxxx | csa -1 -- ) ( Fehler als Text )
   ASCII "  DUP EMIT   WDP@  DOTQ   EMIT    1+ ?DUP
   IF   1- ."  ? Error $"  BASE @  HEX  SWAP U.  BASE !
   ELSE ."  ? " DOTQ
   THEN CR
   BLK @ ?DUP  IF   SCR !  >IN @  R# !  STATE OFF
               ELSE STATE @ IF ." ] " THEN
               THEN QUIT ;




\ ERROR"  ABORT"                                    ( 16.05.90/KK )

| : (ERROR"     ( f -- ) ( Test und Fehler )
   SWAP IF TRUE ERROR THEN DROP ;
  : ERROR"      ( ... error -- ... )
   [COMPILE] "  STATE @  IF COMPILE THEN  (ERROR" ;   IMMEDIATE

| : (ABORT"     ( ... f -- ) ( Test; Stack löschen; Fehler )
    SWAP IF >R DCLEAR R> TRUE ERROR THEN DROP ;
  : ABORT"      ( ... f -- ... | )
   [COMPILE] "  STATE @  IF COMPILE THEN  (ABORT" ;   IMMEDIATE




\ HIDE ... IMMEDIATE ... | )                        ( 15.05.90/KK )

| : LAST?         LAST @ ?DUP ;
| : LAST!         LAST? IF TUCK C@ OR SWAP C! THEN ;
  : HIDE          LAST? IF N>LINK @ CURRENT@ ! THEN ;
  : REVEAL        LAST? IF N>LINK   CURRENT@ ! THEN ;

  : IMMEDIATE     $80 LAST! ;
  : RESTRICT      $40 LAST! ;
\ : INDIRECT      $20 LAST! ;

  : |             ?HEAD @ 0= IF ?HEAD ON THEN ;
  : -HEADERS      1 ?HEAD ! ;
  : HEADERS       ?HEAD OFF ;
```

```
\ CREATE                                          ( 16.06.90/KK )
  DEFER CREATE
| : (CREATE      ( -- ) ( Header mit (VAR anlegen )
   ?NAME  DUP FIND NIP
   IF  CR DUP DOTQ ."  is redefined " THEN
   DUP  C@ $1F MIN 2DUP SWAP C!  1+              ( length<32 )
   ?HEAD @                                 ( Header zum HEAP? )
   IF   1 ?HEAD +!
        HERE  H,   2DUP $1F +  SWAP C!       ( Zeiger auf cfa )
        DUP HALLOT   HEAP SWAP CMOVE>                ( Namen )
        CURRENT@  @  H,  HEAP                    ( Verkettung )
   ELSE >R  HERE  TUCK  2+  R> DUP 2+  ALLOT  CMOVE>
        CURRENT@  @  OVER !                      ( Verkettung )
   THEN DUP  2+ LAST !   CURRENT@  !        ( zurückschreiben )
   $F6 C,  (DIC , ;                    ( zeigt ins Dictionary )
   ' (CREATE IS CREATE
\ VARIABLE  CONSTANT  USER                        ( 16.06.90/KK )

| : RUNTIME     LAST @  NAME> 1+ ! ;

  : VARIABLE    ( name ; -- ; -- addr )
   CREATE  VLEN @ ,  FALSE V,  (VAR RUNTIME ;

  : CONSTANT    ( name ; n -- ; -- n )
   CREATE  ,  (CON  RUNTIME ;

\ : USER        ( name ; -- ; -- addr )
\  ULEN @  DUP $7E OVER U<  IF DROP $000B ERROR THEN
\  CREATE  C,  (USER  RUNTIME
\  2 ULEN +! ;


\ :  ;  DOES>                                    ( 15.05.90/KK )

  : :           ( name ; -- 0 ; -- )
   CURRENT @  CONTEXT !
   CREATE  HIDE  FALSE  -3 ALLOT  $1F C,  ] ;

  : ;           ( name ; 0 -- ; -- )
   DUP FALSE ?PAIRS  DROP
   COMPILE EXIT  REVEAL  [COMPILE] [ ;       IMMEDIATE RESTRICT

| : (;CODE      ( -- )
   R> RUNTIME ;                                      RESTRICT
  : DOES>       ( -- )
   COMPILE (;CODE  $F6 C,  (DOES> , ;        IMMEDIATE RESTRICT


\ VOCABULARY  DEFER  IS                           ( 19.05.90/KK )

  : VOCABULARY  ( name ; -- ; -- )
    CREATE                                          ( Header )
    HERE   VOC-LINK @ ,  VLEN @ ,
           VOC-LINK !    FALSE  V,
    (VOC RUNTIME ;

  : DEFER       ( name ; -- ; -- )
   [COMPILE] VARIABLE   ['] NODEFER  V,  (DEFER  RUNTIME ;

  : IS          ( csa -- )
   [COMPILE] '
   DUP 1+ @  ['] (DEFER - IF DROP $0013 ERROR THEN
   STATE @  IF   COMPILE (IS ,  EXIT  THEN
                 >BODY  ->VAR  !      ;              IMMEDIATE
```

```
\ BRANCH  ?BRANCH  <MARK ... >RESOLVE  RESUME   ( 16.05.90/KK )

  : <MARK       ( -- addr ) ( merkt sich aktuelle Adresse )
   HERE ;
  : <RESOLVE    ( addr -- ) ( korrigiert Adresse im Befehl )
   , ;
  : >MARK       ( -- addr ) ( merkt sich Adresse des Befehls )
   HERE  DUP 2+  , ;
  : >RESOLVE    ( addr -- ) ( korrigiert Adresse im Befehl )
   HERE  SWAP !  ;

| : RESUME      ( 2 addr -2 ... -- )
   BEGIN  2-  WHILE  >RESOLVE  REPEAT ;
| : (?PAIRS     ( n x -- | n bei Fehler )
   OVER ?PAIRS DROP ;

\ IF  ELSE  THEN                                  ( 21.11.90/KK )

  : IF          ( -- addr 1 )
   COMPILE ?BRANCH >MARK 1 ;                 IMMEDIATE RESTRICT

  : ELSE        ( addr 1 -- addr2 -1 )
   1 (?PAIRS                              ( Test )
   COMPILE  BRANCH  >MARK TRUE  ROT     ( neue Adresse )
   >RESOLVE ;                    ( für IF )   IMMEDIATE RESTRICT

  : THEN        ( addr ±1 -- )
   DUP ABS 1 ?PAIRS DROP                  ( Test )
   >RESOLVE ;            ( für IF/ELSE )      IMMEDIATE RESTRICT



\ BEGIN  WHILE  UNTIL  REPEAT                     ( 16.05.90/KK )

  : BEGIN       ( -- 2 addr 2 )
   2 <MARK 2 ;                               IMMEDIATE RESTRICT

  : WHILE       ( .. addr 2 -- .. addr1 -2 addr 2 )
   DUP  2 ?PAIRS
   COMPILE ?BRANCH  >MARK  -2   2SWAP ;      IMMEDIATE RESTRICT

  : UNTIL       ( 2 .. addrn -2 addr 2 -- )
   2 (?PAIRS
   COMPILE ?BRANCH <RESOLVE    RESUME ;      IMMEDIATE RESTRICT

  : REPEAT      ( 2 .. addrn -2 addr 2 -- )
   2 (?PAIRS
   COMPILE BRANCH  <RESOLVE    RESUME ;      IMMEDIATE RESTRICT
\ ( ?DO ) DO  LOOP  +LOOP                        ( 21.11.90/KK )

  : ?DO         ( -- )
   COMPILE (?DO >MARK  <MARK 4  ;            IMMEDIATE RESTRICT
  : DO          ( -- )
   COMPILE (DO  >MARK  <MARK 4  ;            IMMEDIATE RESTRICT

  : LOOP        ( addr 4 -- )
   4 (?PAIRS
   COMPILE (LOOP   <RESOLVE  >RESOLVE ;      IMMEDIATE RESTRICT
  : +LOOP       ( addr 4 -- )
   4 (?PAIRS
   COMPILE (+LOOP  <RESOLVE  >RESOLVE ;      IMMEDIATE RESTRICT
```

```
\ SAVESYSTEM                                        ( 16.05.90/KK )

  : SAVESYSTEM  ( -- )
   RAMORG DUP 2-  SWAP @  $50 CMOVE       ( Zustand speichern )
   HERE  RAMORG @ -              ( Programmlänge )
         VLEN @           +     ( + Variablen-Länge )
         $D0 >TASK# @ - 8 *  +  ( + $80*Taskanzahl )
   3  COMMAND!  ERROR            ( = Gesamtlänge )
   RAMORG @  HERE OVER -  TYPE                     ( Programm )
   VDP    @  VLEN @       TYPE                   ( Variablen )
   LASTTASK
   $CF  >TASK# @  DO  @  DUP $80 TYPE 1+  $10 +LOOP
   DROP ;



\ (INTERPRET                                        ( 16.06.90/KK )
  DEFER INTERPRET
| : (INTERPRET
   BEGIN
     BL WORD  DUP C@  WHILE  FIND ?DUP
     IF  STATE @
        IF   0<  IF  EXECUTE  ELSE  ,  THEN
        ELSE ABS 2- IF  EXECUTE  ELSE  $000E ?ERROR  THEN  THEN
     ELSE  NUMBER? ?DUP
        IF STATE @  IF 0> IF SWAP [COMPILE] LITERAL THEN
                             [COMPILE] LITERAL
                  ELSE DROP THEN
        ELSE NOTFOUND THEN
     THEN  ?STACK
   REPEAT DROP ;
  ' (INTERPRET  IS  INTERPRET
\ (QUIT  VERSION  IDENT                             ( 21.11.90/KK )

  DEFER QUIT
| : (QUIT       ( Hauptschleife )
   R0 @ RP!
   BEGIN  TIB $50 EXPECT   SPAN @  #TIB !
          >IN OFF  BLK OFF  INTERPRET
          STATE @  IF CR ." ] " ELSE ."  ok" CR THEN REPEAT ;
   -2 ALLOT     ( EXIT wird nicht mehr benötigt )
  ' (QUIT  IS  QUIT


  : IDENT       ( -- )
    CR ."  Zilog Super8-FORTH V1.0 " CR ;


\ 'ABORT  ABORT  'COLD  COLD                        ( 16.06.90/KK )

  DEFER 'ABORT  ' NOOP IS 'ABORT        ( Programmstart )
  : ABORT       ( ... -- ) ( Neustart mit leeren Stacks )
   DCLEAR   STATE OFF   ?HEAD OFF
   'ABORT   IDENT   QUIT   ;   -2 ALLOT

  DEFER 'COLD   ' NOOP IS 'COLD         ( Reset-Maßnahme )
  CODE COLD     $0020 JP,   END-CODE     ( Kaltstart )
  L: (COLD
     ]  $7FFF FIRST !   ONLYFORTH   'COLD   ABORT    [
  (COLD  COLDVEC 2 +  !                ( Einsprung übernehmen )
```

```
\ Abspeichern der Einstellungen                ( 16.06.90/KK )

 ( Setzen der wichtigsten USER-Variablen )
  { TDP     @ }         DP              !
  { TVAR    @ }         VDP             !
  { TVAR_OFFSET @ }     VLEN            !
  &10                   BASE            !
  ' SIO_INPUT  >BODY    INPUT           !
  ' SIO_OUTPUT >BODY    OUTPUT          !
  ' (ERROR              ERRORHANDLER    !
  $36  STATE 2 +  !     ( Anzahl belegter Bytes speichern )

  { TLINK @ }           ' FORTH >BODY 2 + @ { TVAR @ } +  !
\ { TLINK @ }           VOCLINK         ! ( !T! nur für Tests )
  ' FORTH >BODY         VOC-LINK        !

\ Interuptvektoren, INITVAL übernehmen          ( 16.06.90/KK )

 { TDP @   $FFD0 TDP !  }
  ' INTERROR    ASSEMBLER
    DUP CALL,   DUP CALL,   DUP CALL,   DUP CALL,
    DUP CALL,   DUP CALL,   DUP CALL,   DUP CALL,
    DUP CALL,   DUP CALL,   DUP CALL,   DUP CALL,
    DUP CALL,   DUP CALL,   DUP CALL,   DUP CALL,
                FORTH        DROP
 { TDP ! }

( Tabelle, Variablenbereich und Task übernehmen )
  RAMORG 2 -    INITVAL    { $50 TCMOVE } ( Tabelle übernehmen )
  VDP @         HERE       VLEN @  DUP  ALLOT  { TCMOVE }
  ORIGIN        HERE       $80       DUP  ALLOT  { TCMOVE }
 { CR CR .( THERE bei ) HEX THERE U. }
( Zusätze für Trace: Hex-Ausgaben )             ( 19.05.90/KK )
 ASSEMBLER
      L: AXHEXOUT       ( AX als HEX-Zahl mit SPACE )
        1$ CALL,   AH , AL LD,   1$ CALL,
        AH , # $20 LD,   3$ JP,
  1$: L: AHHEXOUT       ( AH als HEX-Zahl ausgeben )
        AH PUSH,  AH RR,   AH RR,   AH RR,   AH RR,
        2$ CALL,   AH POP,
  2$: L: AHLNOUT        ( Low-Nibble des Akku's als Hexwert )
        AH , # $0F AND,   AH , # $30 OR,
        AH , # $3A CP,   C , 3$ JP,   AH , # $07 ADD,
  3$: L: AHOUT          ( Akku ausgeben )
        AH PUSH,
      L: SWAIT   AH , &235 LD,   SWAIT , AH # 1 BTJRF,
        &239 POP,   RET,
 FORTH
( Zusätze für Trace: Name suchen )              ( 19.05.90/KK )
 ASSEMBLER
  L: VOCLINK  0  { T, }         ( !T! Zeiger auf oberste Wort )
  L: NEXT0      $0F { TC, }                ( !T! NEXT-Befehl )
  L: ???   { 3 TC,   ASCII ?  DUP TC, DUP TC, TC, }
  L: (>NAME     ( AX = cfa -- CX = nfa ; BX verwendet )
        CX , # VOCLINK LDW,                       ( VOC-LINK )
  L: >N1: BH , @CX LDCI,   BL , @CX LDC,        ( nächste LFA )
        CH , BH LD,   CH , BL OR,   NZ , 2$ JP,      ( Ende ? )
        CX , # ??? LDW,   RET,               ( nicht gefunden )
    2$: BX PUSHW   BX INCW,   BX INCW,            ( zur NFA )
        CL , @BX LDCI,   CL , # $1F AND,
        BL , CL ADD,   BH , # 0 ADC,            ( NFA+Länge+1 )
        CX POPW   BH , AH CP,  NZ , >N1: JR,        ( = cfa ? )
                  BL , AL CP,  NZ , >N1: JR,
        CX INCW,   CX INCW,   RET,     FORTH        ( zur NFA )
```

```
( Zusätze für Trace: Stacks ausgeben )          ( 19.05.90/KK )

 ASSEMBLER
  L: STACKS.    ( BX=Offset; AL=Größe )
        AL PUSH,   AH , # ASCII : LD,   AHOUT CALL,
        AH POP,   AHHEXOUT CALL,
        AH , # $20     LD,   AHOUT CALL,
        AH , # ASCII = LD,   AHOUT CALL,
        AH , @BX LDCI,   AL , @BX LDCI,   AXHEXOUT CALL,
        AH , @BX LDCI,   AL , @BX LDCI,   AXHEXOUT CALL,
        AH , @BX LDCI,   AL , @BX LDCI,   AXHEXOUT CALL,
        AH , # ASCII | LD,   AHOUT CALL,
        AH , # $20     LD,   AHOUT JP,
 FORTH


( TRACE )                                       ( 19.05.90/KK )

 ASSEMBLER
  L: TRACE      ( Patch in NEXT )
        AX PUSHW                              ( TOS speichern )
        AH , # $D LD,  AHOUT CALL,  AH , # $A LD,  AHOUT CALL,
        AX , IP LDW,   AXHEXOUT CALL,                   ( IP )
        BX , IP LDW,   AH , @BX LDCI,   AL , @BX LDCI,
        AX PUSHW   AXHEXOUT CALL,   AX POPW           ( CFA )
        (>NAME CALL,
        BH , @CX LDC,   BH , # $0F AND,   BH PUSH,
 L: T1: CX INCW,  AH , @CX LDC,  AHOUT CALL,  BH , T1: DJNZ,
        BL POP,   BH , # $10 LD,   BH , BL SUB,
 L: T2: AH , # $20 LD,   AHOUT CALL,   BH , T2: DJNZ,


( TRACE  2. Teil )                              ( 14.06.90/KK )
        AH , # ASCII S LD,   AHOUT CALL,
        BX , FSP LDW,
        AX , $10 (FUP LDCW                              ( S0 )
        AL , BL SUB,   AH , BH SBC,
        AH SRA,   AL RRC,    STACKS. CALL,
        AH , # ASCII R LD,   AHOUT CALL,
        BX , FRP LDW,  ( BX INCW,  BX INCW, )
        AX , $12 (FUP LDCW                              ( R0 )
        AL , BL SUB,   AH , BH SBC,
        AH SRA,   AL RRC,    STACKS. CALL,
  L: T3: AH , &236 LD,  AH , # 1 AND,   Z , T3: JP,
        AH , &239 LD,  AH , # 3 CP,    NZ , 4$ JP,
        AX , # NEXT0 LDW,   $FFC6 , AX LDCW
  4$:   AX POPW   $0F { TC, }
 FORTH
( TRON  TROFF )                                 ( 16.06.90/KK )

   CODE TRON    ( -- )
        BX , # TRACE LDW,
        $FFCE , BX LDCW
        NEXT,                                       END-CODE

   CODE TROFF   ( -- )
        BX , # NEXT0 LDW,
        $FFCE , BX LDCW
        NEXT,                                       END-CODE
```